Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Métodos de Reducción de Datos para Clasificación con Máquinas de Soporte Vectorial

Tesis que presenta

Asdrúbal López Chau

para obtener el grado de

Doctor en Ciencias en Computación

Asesores

Dra. Xiaoou Li **Dr. Wen Yu**

México, Distrito Federal Septiembre, 2013

Centro de Investigación y
de Estudios Avanzados del
Instituto Politécnico Nacional
Unidad Zacatenco
Departamento de Computación

Data Reduction Methods for Classification with Support Vector Machines

Dissertation submitted by

Asdrúbal López Chau

for the degree of
Doctor of Philosophy (PhD) in Computer Science

Supervisors

Dr. Xiaoou Li

Dr. Wen Yu

Mexico City September, 2013

Abstract

Support Vector Machine (SVM) is a state-of-the-art classification method whose model is a hyperplane of maximum margin. SVMs produce a high classification accuracy, a compact model and have an extraordinary generalization capability. In spite of these attractive features, this classifier has the disadvantage of being unsuitable for large data sets, because its training phase is costly.

In this research, two methods to decrease the size of the training data sets are proposed, in order to improve the training time of a SVM. The first method presented in this work uses a convex-concave hull to detect objects in data sets that are located on the outer boundaries of data, this method is suitable for low dimensional data sets. The second method uses the concept of entropy to detect objects that are close to others with opposite label; this method can work with an arbitrary number of dimensions.

Our methods allow to apply SVMs on large data sets. In fact, these methods also improve the training time on medium-size data sets. The proposed methods were validated using publicly available data sets and comparing performance against other state of the art methods. After applying the novel methods, the training time of SVM is considerably improved whereas the achieved classification accuracy is only slightly degraded.

Resumen

La Máquina de Soporte Vectorial o Máquina de vectores de soporte (SVM, por sus siglas en inglés) es un clasificador del estado del arte cuyo modelo es un hiperplano de margen máximo. Las SVMs alcanzan una elevada precisión de clasificación, generan un modelo compacto y tienen un poder de generalización extraordinario. A pesar de estas atractivas características, este clasificador tiene la desventaja de no ser apropiado para conjuntos de datos grandes, debido a que su fase de entrenamiento es costosa.

En esta investigación, se proponen dos métodos para disminuir el tamaño de conjuntos de datos, estos mejoran el tiempo de entrenamiento de las SVMs. El primer método presentado en este trabajo usa una cubierta cóncava-convexa para detectar objetos localizados en los bordes externos de conjuntos de datos; este método es adecuado para conjuntos de datos de baja dimensionalidad. El segundo método usa el concepto de entropía para detectar objetos que se encuentran cerca de otros de clase opuesta; este método puede trabajar con un número arbitrario de dimensiones.

Los métodos de reducción de datos propuestos permiten aplicar SVM sobre conjuntos de datos grandes. De hecho, estos métodos también mejoran el tiempo de entrenamiento en conjuntos de datos medianos. Los métodos propuestos fueron validados usando conjuntos de datos disponibles públicamente y comparando su desempeño con respecto al de otros métodos del estado del arte. Después de aplicar los nuevos métodos, el tiempo de entrenamiento de las SVMs mejora de manera considerable, mientras que la precisión alcanzada sólo es disminuida ligeramente.

Publications

The products developed in this research are the following:

1. Journal paper: *Convex and concave hulls for classification with support vector machine*, **Asdrúbal López Chau**, Xiaoou Li, Wen Yu, Neurocomputing, Available online 11 July 2013, ISSN 0925-2312, http://dx.doi.org/10.1016/j.neucom.2013.05.040. (http://www.sciencedirect.com/science/article/pii/S0925231213006449)

2. Journal paper: *Support vector machine classification for large datasets using decision tree and Fisher's linear discriminant*, **Asdrúbal López Chau**, Xiaoou Li, Wen Yu, Future Generation Computer Systems, Available online 10 July 2013, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2013.06.021. (http://www.sciencedirect.com/science/article/pii/S0167739X13001350)

3. Journal paper: *Large Data sets Classification Using Convex-Concave Hull and Support Vector Machine*, **Asdrúbal López Chau**, Xiaoou Li and Wen Yu, Soft Computing, Springer Verlag, 12 pages, pp 793-804, ISSN 1432-7643, 2013, doi10.1007/s00500-012-0954-x

4. Journal paper: *Structural Health Monitoring of Tall Buildings with Numerical Integrator and Convex-Concave Hull Classification*, Suresh Thenozhi, Wen Yu, **Asdrúbal López Chau**, and Xiaoou Li, Mathematical Problems in Engineering, vol. 2012, Article ID 212369, 15 pages, 2012. doi:10.1155/2012/212369

5. Book chapter: *Border Samples Detection for Data Mining Applications Using non Convex Hulls*, **Asdrúbal López Chau**, Xiaoou Li, Wen Yu, Jair Cervantes and Pedro

Mejía. Advances in Soft Computing LNCS, Vol. 7095, ISBN 978-3-642-25329-4, pp 261-272. Año 2011.

6. Book chapter: *A Fast SVM Training Algorithm Based on a Decision Tree Data Filter*, Jair Cervantes, **Asdrúbal Lopez** and Farid Garcia y Adrían Trueba. Advances in Artificial Intelligence LNCS, Vol. 7094, ISBN 978-3-642-25323-2, pp 261-272, 187-197. Año 2011.

7. Conference paper: *Data Selection Using Decision Tree for SVM Classification*, **Lopez-Chau, A.**; Garcia, L.L.; Cervantes, J.; Xiaoou Li; Wen Yu, Tools with Artificial Intelligence (ICTAI), 2012 IEEE 24th International Conference on , vol.1, no., pp.742,749, 7-9 Nov. 2012

8. Conference paper: *Fast Splice Site Classification Using Support Vector Machines in Imbalanced Data sets*, Jair Cervantes, **Asdrúbal López Chau**, Adrian Trueba Espinoza and Jose Sergio Ruiz Castilla, Proceedings of the 2011 international conference on bioinformatics and computational biology. Volume I ISBN: 1-60132-170-8,1-60132-171-6(1-60132-172-4), pp 136-141. Año 2011.

9. Doctoral consortium: *Data stream classification*, **Asdrúbal López Chau**, November 9^{th} 2010, Mexican international conference on artificial intelligence. Pachuca, Hidalgo. México.

10. Conference paper: *SV candidates pre-selection strategy using non convex hulls*, **Asdrúbal López Chau**, Xiaoou Li, Wen Yu and Jair Cervantes, 2010, 7th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2010), México.

To Lutecia

Thank you for illuminating my life in every existence

Asdrúbal

Acknowledgments

The number of people that helped me to finish this research easily exceeds the number of words in this document. Due to the limited space in it, many people are not mentioned in these acknowledgments. Thanks to them too.

I wish to thank to my wife Lutecia. Her mere presence in my life has been a motive to walk along the eightfold path. Also, I would lile to thank to all my family. As well as Dr. Xiaoou Li and Dr. Wen Yu, my thesis advisers, for giving me the opportunity to continuing with graduate studies.

An acknowledgment to the reviewers of this work. Dr. Carlos A. Coello Coello, his teachings, comments and suggestions dramatically improved this work. I will be always indebted to him; Dr. Debrup Chakraborty, his observations enhanced this work. His amazing and elegant classes will be a permanent source of inspiration for me; Dr. Jair Cervantes, his suggestions and discussions about this research encouraged me to continue and finish this work, he has become an invaluable friend; Dr. Iván López Arevalo and Dr. Miguel González Mendoza, their comments were very important.

I am grateful with all professors at the Computer Science Department at CINVESTAV-IPN, specially, to Dr. José G. Rodríguez, Dr. Oliver Sch$\ddot{u}$tze and Dr. Jorge Buenabad.

The assistance of Pedro Guevara López, José de Jesús Medel Juárez, Miguel Angel Miguez Escorcia, Jose Carlos Quezada and Cristina Flores Amador at the beginning of this journey was inestimable. I appreciate the help of Francisco Javier García Lavalley, Rodolfo Téllez Cuevas, Raymundo Ocaña and Alejandro Mendieta for the facilities they provided me to continue with this research work. A special gratitude to all my friends at the Universidad Autónoma del

Estado de México and at the Universidad Autónoma del Estado de Hidalgo.

I feel privileged to have studied at CINVESTAV-IPN. Also, I feel fortunate to have met valuable friends. Liliana Puente Maury, her prodigious help played a very important role. Furthermore, thanks to Farid García, Lisbeth Masahua, Ivonne Ávila, Kimberly García, Gabriela Sánchez, Luis Enrique Ramírez, Joel Villanueva, Israel Buitrón, Saúl Zapotecas, Cuauhtemoc Mancillas, Daniel Escogido and Eduardo Pérez.

Some friends gave me moral support during last years: Jorge Bautista López, Lourdes López García, Carlos Alberto Rojas Hernández and Jair García Lamont.

Finally, but no less important, my gratefulness to Sofia Reza, Felipa Rosas and Erika Berenice Ríos for their extraordinary support.

This research would not have been possible without the student grant of CONACYT.

Contents

List of Figures

Page

List of Tables

Page

1
Introduction

> If we can really understand the problem, the answer will come out of it, because the answer is not separate from the problem
>
> *Jiddu Krishnamurti*

The theoretical and technological advances in Computer Sciences and Computer Engineering have made possible the current capabilities of storing a huge amount of information at negligible costs. In fact, the rate of production of data is growing more and more. Since 2002, when the so-called *digital age* started [1], information has been preferentially stored in digital media. By 2007, almost 97% of information was stored in this way [1]. The capacity for information storing, including digital and Analog Devices was estimated in more than 295 Exabytes[1] in 2008.

The inclusion of complex electronic devices in virtually every place (security systems, sale points, automobiles, industrial control systems, etc.) and the networking capabilities of many systems (PDA, GPS, smart phones, computers, web applications, etc.) gave as a result a massive generation of information. According to the most-recent study presented in June 2011 [2] the amount of data generated is estimated exceed 1.8 zettabytes (ZB) [2],[3] (1.8 trillion gigabytes). Every hour, thousands of mega Bytes (MB) are generated, and it is also estimated that the amount of data will grow at least 50 times in the next 10 $years$[4].

[1] Exabyte (EB)= 10^{18} Bytes

[2] Zettabyte (ZB)= 10^{21} Bytes

[3] The exact number is $1, 987, 262, 613, 861, 770, 000, 000$ Bytes, http://www.emc.com/leadership/programs/digital-universe.htm

[4] http://www.emc.com/microsites/bigdata/index.htm

Data are worthless if there are no mechanisms to extract useful or interesting knowledge from them. The first successful attempts to explore this area were achieved by statisticians, using parametric models to explain the data. In pure statistical methods, a model is generally proposed a priori, and then the parameters are adjusted with the minimal possible error using as less as possible data. It is known that using only classical statistical tools is now impractical because the amount of data has dizzily increased, data are incomplete or noisy and the underlying models that generate the data are complex and can change over time.

The area of automatic extraction of knowledge from databases emerged in the late 1980s as a support to understand data digitally stored. The proposal was to create flexible and powerful techniques with the ability to be driven by data instead of to be driven by a model. The main goal was to extract new, interesting and worth knowledge from large amounts of data [3] [4] with minimal or no human intervention. Much progresses have been accomplished, but there is much work is pending yet, for example: scaling algorithms for big data; developing methods for high speed data streams; adapting algorithms for distributed, multi core and parallel platforms and creating new methods specific for dynamic environments on specific devices (low power consumption or very limited resources).

The techniques developed for mining knowledge can be grouped in four main categories [5] [6]: Classification, Clustering, Regression, and Association rules.

Classification is the task related to predict the associated type or category associated to a given object, i.e., for a previously previously unseen object trying to identify the category it belongs to. The category, called *class*, can be represented by discrete values where the ordering among values has no meaning. Classification is said supervised because it is necessary to start with a set of labeled objects (this set is called training data set) to build a model to predict the labels as accurately as possible.

Linear classification methods use hyperplanes as decision boundaries, in general these methods solve an optimization problem to determine the separating hyperplane, or, more strictly, an affine plane. Some classification methods use kernels, which are non linear functions that permit to work in a higher dimensional space where data can be treated as linearly separable, even when they are not linearly separable in the original input.

The SVM is a state-of-the-art classification method that uses an optimal separating hyperplane (linear boundary) to classify. The objects in the training data set that determine the optimal separating hyperplane are called support vectors (SV) and they are obtained by solving a quadratic programming problem (QPP).

Table 1.1: **Training methods for SVM**

Algorithm	**Data set size**[a]	**Features**	**Type of method**[b]
SMO (1998)[12]	32,562	14	Decomposition
SMO improved (2001)[13]	24,692	300	Decomposition
LibSVM (2005)[14]	100,000	54	Decomposition
RCH (2007)[8]	618	2	Geometric
SCH (2009)[11]	16,000	14	Geometric
LASVM (2005)[15]	521,012	14	Data reduction
Hybrid DT SVM (2010)[16]	22,696	123	Data reduction
Hyperplane Distance (2011)[17]	20,000	22	Data reduction

[a] Maximum number of objects reported in corresponding article
[b] Technique used by the method

The SVM can produce linear or nonlinear optimal boundaries via kernels. Theoretically, the optimal separating hyperplane solved by SVM produces the best generalization possible for the linearly separable case. It has been demonstrated [7] that methods for classification having optimal linear boundaries converge to the SVM, so all its benefits mentioned before are inherent [8] [9] [10] [11].

Current methods for training SVMs can be categorized as: data reduction, based on geometric properties, decomposition, variants of SVM and others. Table 1.1 shows representative methods for training SVM.

The sequential minimal optimization (SMO) is probably the most popular method for training SVMs, however, it is not the fastest. LibSVM is a library based on SMO, which outperforms SMO in the literature, and such a behaviour was corroborated in our experiments. The Reduced Convex Hull (RCH) and Scaled Convex hull (SCH) are geometric methods that do not scale well in practice, they work well only with a few hundreds of examples.

In this research, we develop two novel reduction methods for improving the training time of SVM classifiers. The idea behind data reduction methods is the observation that, in general, the number of SV is small compared with the number of objects in data sets [18] [19] [20]. The objects with a high probability of being support vectors (support vector candidates) are used to train the SVM, the goal is to quickly detect the support vector candidates to reduce the training time of the SVM.

Motivation and goals

SVM classifiers have been successfully adopted in many applications such as credit rating [21], chemistry [22], spam filtering [23] [24], control of electric machines [25] and marketing [26], among others. The model produced by SVM is compact, geometrically interpretable and its performance usually surpasses the classification accuracy of other methods. In spite of their characteristics, SVM classifiers have a noticeable problem; the training phase consumes about $O(n^3)$ time and $O(n^2)$ memory space [27] [28]. This prevents the use of SVM with large data sets.

Enabling SVM on large data sets is an interesting problem. There are different types of methods for training SVMs. Methods based on geometric properties of SVMs work well for linearly separable cases, however, they do not achieve good accuracy on the linearly inseparable cases. The Decomposition methods can be used with large training sets but they converge slowly. Variants of SVM improve training time of SVM at the expense of classification accuracy. Data reduction has shown scale better than the other approaches, and, additionally, it can be applicable to other classification methods.

The main goal of this research is to develop novel data reduction methods to improve the training time of the SVM classifier. These methods make possible to apply SVMs on large data sets achieving an acceptable level of classification accuracy while keeping a training time significantly lower than that of state-of-the-art methods.

The specific objectives are the following:

- Analyze the state-of-the-art methods for training SVMs on large data sets, in order to identify the advantages and disadvantages of these methods.
- Analyze the geometric properties of SVMs to propose a method based on a non-convex hull, to detect objects on the boundaries of the data distribution.
- Develop a novel method to detect objects located close to objects with opposite label.
- Determine the computational complexity of the proposed methods.
- Test the developed methods with publicly available benchmark data sets.
- Evaluate the performance of the proposed methods against the most representative algorithms for training SVMs.

Contributions

In this work, we propose two novel data reduction methods for the SVM classifier. The proposed methods outperform the state-of-the-art methods for training SVMs such as LibSVM, SMO, RCH, SCH and SVM^{Light}. A brief description of the contributions is provided next:

1. A fast reduction method based on non-convex hull. This method is suitable for low dimensional data sets containing tens of thousands of objects. The method uses a convex hull as a reference to guide a search of instances located on the boundaries of the training set. The result is a super set of vertexes of $\mathcal{CH}$ that form a non convex hull.

2. A method based on the entropy concept. This method uses a decision tree to discover low entropy regions, which are treated as clusters. Two main variants were developed. The first one uses Fisher's linear discriminant to detect objects located close to clusters with an opposite class. The second variant uses the objects close to centers of clusters to train a SVM.

Document Organization

The rest of this document is organized as follows. Chapter 2 is devoted to describe the classification task, covering the basic background, terminology used, measures for evaluation of classifiers and the classification methods available that are relevant for the purposes of this thesis.

Chapter 3 shows the state of the art on training methods for SVM. Also, the results of a preliminary experiment are presented to show that the closest examples to an opposite class are good candidates to be SV candidates.

In Chapter 4, a data reduction method able to work with low dimensional data sets is presented. The method is based on non-convex hulls and search objects located on the boundaries of sets of points.

A method that uses a decision tree is explained in Chapter 5. This method can be applied to large data sets regardless of the number of features of each object. A minor variant of this method is also included in Chapter 5. This variant is faster but it is less accurate.

The general conclusions as well as some possible paths for future work are presented in last part of this thesis.

2

Classification

A man of knowledge lives by acting, not by thinking about acting
Carlos Castaneda

The background on the classification task is presented in this Chapter. We begin with some definitions and explain the main supervised and unsupervised methods, focusing on the following classification methods: support vector machines, decision trees and Fisher's linear discriminant. These three classifiers are related to the methods developed during this research. Finally, this Chapter provides a description of the main techniques used to evaluate classifiers.

2.1 Preliminaries

A data set is a collection of data that contains individual data units, which are called objects.

> The terms **instance**, **object**, **record**, **sample**, **pattern** or **example** will be used interchangeably throughout this document; in all cases, the meaning is: an element of a given labeled or unlabeled data set.

The objects are composed of features also known as attributes or properties. The features are usually measures of real-world objects or relations between entities. The number of features is known as the *dimension* of an example. Instances usually have a small number of features, generally tens of them; however, some data sets have many features[1].

[1] The *URL Reputation* data set [23] has $3,231,961$ attributes.

There are five main types of attributes, these are:

- Nominal attributes, which are not numerical, i.e.; they are simply a label. An example can be the color of something or the gender of a person.
- Ordinal variables. These are similar to nominal ones; however, their values can be arranged in a meaningful order, for example, light, medium or heavy.
- Numeric attributes. Their values are integer or real numbers.
- Interval features, which are quantities. Their values are not only ordered, but also measured in fixed and equal units [3].
- Binary or Boolean. It is a special case of a nominal variable; it takes only two possible values. Typical examples are: true or false, 0 or 1, male or female, etc.

Through this document, a data set is represented with an X. N is the number of examples in X, i.e., $N = |X|$. The number of features is referred as d.
The data sets used in this work have the form

$$X = \left\{ (x_i, y_i), i = 1, ...N., x_i \in R^d, y_i = \{C_1, C_2\} \right\} \tag{2.1}$$

A data set is said to be *labeled*, if all or most of its records contain a special attribute, called the *class attribute*; this indicates that it has some significance or is meaningful. The purpose of the class attribute is to identify the sample as being of a particular category or class. If the data set does not contain a class attribute, then it is called *unlabeled*.

Table 2.1 shows a fragment of the Iris data set [29], taken from [30]. This set is probably the most famous data set for classification. Each row in the labeled Iris data set represents an instance, which has four properties and a class attribute.

Methods for retrieving knowledge from data can be categorized as supervised or unsupervised. Both kinds of methods use data sets to build models. The supervised methods build models from labeled data sets. The models obtained must be able to predict the class of previously unseen objects. The unsupervised methods discover groups of similar objects from unlabeled data sets.

Table 2.1: **Fragment of the Iris data set**

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
⋮				
7.0	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
⋮				

The main supervised methods are classification and regression. Classification methods use labeled data sets whose class attribute is categorical. The three key objectives in classification are high classification accuracy, comprehensibility (ability of a human expert to understand the classification model) and compactness (size of model) [31]. The regression is similar to classification; however, the labels are of numeric type, i.e.; the class attribute is continuous.

The principal unsupervised learning methods are: clustering, outlier analysis and frequent pattern analysis. Clustering consists of identifying similar objects based on distances. These algorithms are categorized in hierarchical, partitioning, density-based, constraint-based clustering, grid-based methods and Model-based methods [32]. Outlier analysis consists in identifying samples that do not comply with the general behavior or model of data. Frequent patterns analysis includes those methods devoted to search for recurring relationships in a given data set.

Supervised and unsupervised methods are related to data mining (DM). The latter refers to a group of data-driven methods, devoted to extract knowledge from large amounts of data [4][5]. DM can be found in the literature with the following names: knowledge extraction, exploratory data analysis, information discovery, information harvesting, data archeology and data pattern processing [5].

2.2 Classifiers

Classification is a supervised learning technique that consists in assigning an object to one of a set of predefined categories. The input data for a classifier is a labeled data set X, in which the class attribute is of categorical type; the goal is to build a model to predict the (categorical) label of previously unseen samples. An unknown probability distribution is used to extract examples from a data set, then a model is created utilizing these objects. The model can predict accurately new examples if these are generated using the same probability distribution.

In general, the available information is not enough to have a clear relationship between inputs and output values. The goal in the classification task is to construct a *decision function* to make *good* predictions, i.e., learn a map from input $\mathcal{X}$ into output Y, or, mathematically

$$f : \mathcal{X} \mapsto Y \tag{2.2}$$

where

f : is the decision function,

$\mathcal{X}$: is the input space,

Y : is the output, the set of labels.

The *training* of a classification method consists in using pairs (x_i, y_i) to build decision functions [28]. Because the decision functions are used to classify objects, the former are known as classifiers.

The decision functions partition the input space into a number of regions [33] (see eq. (2.3)), defined by *decision regions*, *decision boundaries* or *decision surfaces* [34]. After a decision function has been learned from X, a given example x_k can be mapped into a partition A_j and a label $y \in Y$ is assigned to x_k.

$$\mathcal{X} = \bigcup_j A_j \text{ such that } \bigcap_j A_j = \varnothing \tag{2.3}$$

where

A_j : The $j-th$ partition of $\mathcal{X}$.

The decision functions are also known as *classification models*. They are used as explanatory tools to understand hidden relations in a data set, or to predict the class label of unknown records.

Some algorithms for building classifiers use dot products and norms; these concepts are presented in Definitions 1 and 2. The dot product is a tool for measuring angles and lengths in geometry. Dot product is closely related with the concept of norm in vector spaces. Inner products and norms are treated in classification as a measure of dissimilarity among instances.

Definition 1 (Dot product) *The dot product between two vectors $u, v \in R^d$ is a map $\langle \cdot, \cdot \rangle : R^d \times R^d \to R$, it is computed with*

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = u^T v = \sum_{i=1}^{d} u_i v_i$$

The dot product fulfills the following properties:

1. *Commutative.* $\langle u, v \rangle = \langle v, u \rangle$
2. *Distributive.* $\langle u, v + w \rangle = \langle u, v \rangle + \langle u, w \rangle$
3. *Bilinear.* $\langle u, rv + w \rangle = r \langle u, v \rangle + \langle u, w \rangle$
4. *Scalar multiplication.* $\langle au, bv \rangle = ab \langle v, u \rangle$

Definition 2 (Norm) *A Norm is a function which assigns a length to vectors. The norm between two vectors $u, v \in R^d$ is a map $f : R^d \to R$, it satisfies*

1. *Positive homogeneity:* $\forall u \in R^d$, $\alpha \geq 0$, $f(\alpha u) = \alpha f(u)$.
2. *Triangle inequality:* $\forall u, v \in R^d$, $f(u + v) \leq f(u) + f(v)$.
3. *Definiteness:* $\forall u \in R^d$, $f(u) = 0$ *implies* $u = 0$.

Definition 3 presents convex sets, which play a central role in some classification methods, specially those that use a linear decision boundary.

Definition 3 (Convex Set) *A set $C \subseteq R^d$ is said to be* ***convex*** *if and only if for every $c_1, c_2 \in C$, $\alpha \in R$ s.t. $0 \leq \alpha \leq 1$*

$$\alpha \boldsymbol{c}_1 + (1 - \alpha)\boldsymbol{c}_2 \in C, \ \forall \ \boldsymbol{c}_1, c_2 \in C \tag{2.4}$$

The classification methods that use linear boundaries, search for points in the line segment defined by two points that belong to a set C. According to Definition 3, if C is convex, then the points that are found by these methods will lie in C. An example of convex sets are the half spaces, which are important for classification. A closed half space is either of the two parts into which a hyperplane divides an affine space, i.e., $\{w : w^t u \leq b\}, \{w : w^t u \geq b\}$.

Definition 4 (Convex Hull) *The convex hull of a set $X \subseteq R^d$ is defined as*

$$\mathcal{CH}(X) \left\{ w : w = \sum_{i=1}^{n} a_i x_i, a_i \geq 0, \sum_{i=1}^{n} a_i = 1, x_i \in X \right\} \tag{2.5}$$

A convex hull of a set is the intersection of all convex sets that contain the set. See Definition 4 for a complete description of convex hull. One way to ensure a classification algorithm is working in a convex set, is to compute the convex hull of input data set; and then, allow the algorithm to run in it. In most cases this approach is very costly.

The $w = \sum_{i=1}^{n} a_i x_i, a_i \geq 0, \sum_{i=1}^{n} a_i = 1$, in Definition 4, is called a convex combination. In the algorithms, the a_i are seen as a weight of instances, or as probabilities.

Definition 5 describes a hyperplane.

Definition 5 (Linear hyperplane) *A linear hyperplane is a R^{d-1}-dimensional space H of a vector space V, i.e.,*

$$H = \{v \in V, \ s.t. \ \langle \boldsymbol{u}, \boldsymbol{v} \rangle = 0\}$$

Where
u is a fixed nonzero vector in V.

Geometric methods for classification, such as linear models or SVM, divide the space into non overlapped regions or partitions. The boundaries (decision surfaces) of such partitions are defined with hyperplanes.

Eidelheit separation theorem is shown in Theorem 2.1; it is important for linear classifiers. The decision surface of these classifiers is a hyperplane, which produces two closed half spaces where each one contains instances of one class. Figure 2.1 shows a graphical representation of the theorem.

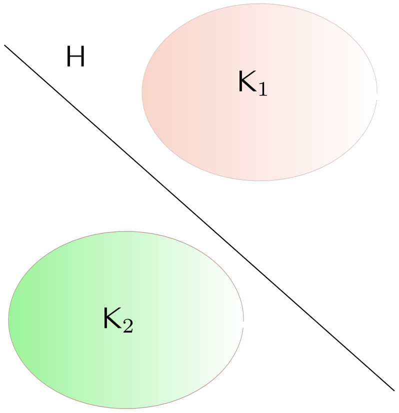


Figure 2.1: **A graphic example of separating hyperplane**

Theorem **2.1 (Eidelheit Separation Theorem)** *Let K_1 and K_2 be two convex sets in a real vector space X, s.t. K_1 contains interior points, and K_2 contains no interior points of K_1. Then, there is a closed hyperplane H separating K_1 and K_2. In other words, K_1 and K_2 lie in the opposite half-spaces determined by the hyperplane H.*

In order to determine separating hyperplanes, the distance between a convex set and a point is usually computed. Theorem 2.2 shows that the point in the convex that is closest to an exterior point to the set, is unique. Figure 2.2 is used to exemplify this fact.

Theorem **2.2 (Minimum Distance to a Convex Set)** *Let x be a vector in a Hilbert space S and let K be a convex subset of S. Then, there is a unique vector $k_0 \in K$ such that*

$$\left\| x - k_0 \right\| \leq \left\| x - k \right\| \ \forall \ k \in K \tag{2.6}$$

Furthermore, a necessary and sufficient condition that k_0 be a unique minimizing vector is that

$$\langle x - k, x - k_0 \rangle \leq 0 \ \forall \ k \in K \tag{2.7}$$

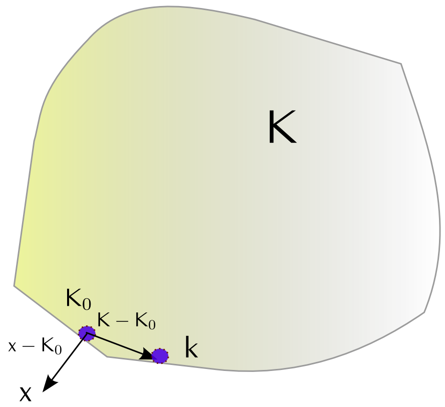


Figure 2.2: **Minimum distance to convex set**

Duality is an important concept in optimization theory and also in classification. This concept allows to represent a problem using another one which is equivalent. The latter can be simpler to solve or to understand. An example of duality is the Minimum Norm Duality Theorem 2.3, which states that the minimum distance from a point to a convex set C, is equal to the maximum of the distances from the point to the (supporting) hyperplanes that separate the point and the set C [35].

Theorem 2.3 **(Minimum Norm Duality)** *Let x_1 be a point in a real normed space X and let $d > 0$ denote its distance from the convex set K having support functional h, then*

$$d = \underset{x \in K}{inf} \left\| x - x_1 \right\| = \underset{\left\| x^* \right\| \leq 1}{max} \left[\langle x_1, x`* \rangle - h(x^*) \right] \tag{2.8}$$

Where the maximum on the right is achieved by some $x_0 \in X^*$.
If the infimum on the left is achieved by some $x_0 \in K$, then $-x_0^*$ is aligned with $x_0 - x$.

Multi class Problems

In binary classification problems, there are two categories of examples in the training data set. The typical structure of data sets is the same as shown in eq. (2.1); in practice the classes C_1 and C_2 take values 0 and 1, or +1 and −1.

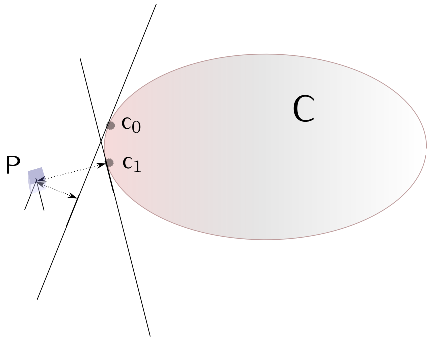


Figure 2.3: **Minimum Norm Duality**

The data sets used for multi-class problems have more than two classes. The structure of these data sets is:

$$X = \{(x_i, y_i) : x \in X, x \in R^d, y_i = \{C_1, C_2, ..., C_L\}\} \tag{2.9}$$

The publicly available data sets usually contain a number of classes L, with $L \leq 20$[2].

For multi-class problems, more than one decision function is used. The form of the discriminant function is

$$g_i(x) > g_j(x) \; \forall \; i \neq j \tag{2.10}$$

where

$i = 1, ..., L$

$j = 1, ..., L$

The last discriminant (eq. (2.10)) can be interpreted as a network, which selects the category corresponding to the largest margin [36]. Other approaches, which do not use discriminant functions can be found in [37][38][39], they are:

1. One-against-all Classification.
2. Using $\frac{k(k-1)}{2}$ pair wise classifiers with one of the voting schemes listed below:
   - Majority Voting
   - Pairwise Coupling

[2] We analyzed the data sets for the classification task available at http://archive.ics.uci.edu/ml/

3. Extending the formulation of SVM to support the k-class problem.

- Construct the decision function by considering all classes at once.
- Construct one decision function for each class, by considering only the training data points that belong to that particular class.

2.2.1 Linear Models

The linear models for classification are among the best understood [27]. In these models, the decision boundaries are linear combinations of inputs x; the decision surfaces are $(d-1)$ dimensional hyperplanes. The simplest linear model takes the form

$$y = \omega^T x + \omega_0 \tag{2.11}$$

where

$y \in R$

$\omega \in R^d$, called *weight vector*.

$\omega_0 \in R$, called the *bias*.

Figure 2.4 shows an example of a linear boundary in two dimensions. In the linear model, the vector ω is orthogonal to the decision boundary. The boundary is defined by $\omega^T x + \omega_0 = 0$. The decision boundary separates the space in two semi planes.

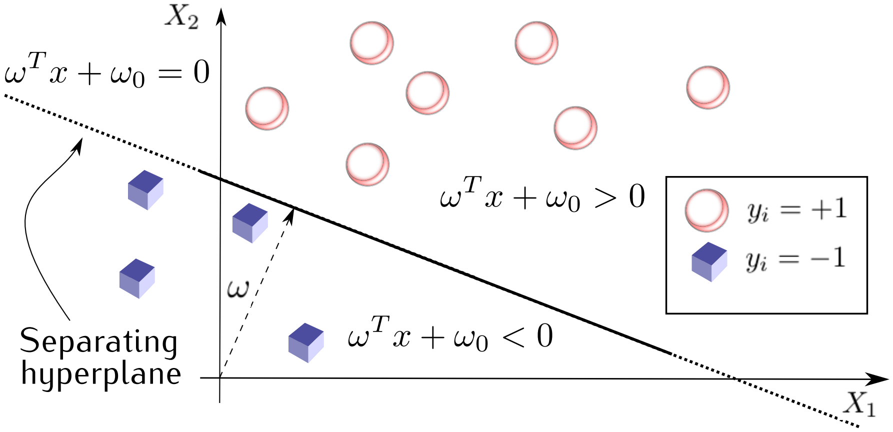

X_2
$\omega^T x + \omega_0 = 0$
$\omega^T x + \omega_0 > 0$
$y_i = +1$
$y_i = -1$
ω
Separating hyperplane
$\omega^T x + \omega_0 < 0$
X_1

Figure 2.4: **Linear Decision Boundary**

In Figure 2.4, all the samples can be perfectly classified. These type of data sets are referred to as *linearly separable data sets*.

In the simplest classification problem, there are only two possible values for the label of each instance, the data set X takes the form shown in eq. (2.1).

The vector ω and the bias ω_0 are obtained after the training. The new incoming samples are tested using $y = \omega^T x + \omega_0$; the output y is a real number. Discriminant functions are used to decide (predict) the label of a sample x_i. A common form of these functions (for the binary classification problem) is

$$\text{IF } (f(\omega^T x_i + \omega_0)) \begin{cases} \geq 0 \text{ THEN } y_i = +1 \\ < 0 \text{ THEN } y_i = -1 \end{cases} \tag{2.12}$$

This discriminant is called the *sign* function.

2.2.2 Support Vector Machines

SVMs classifiers prevent the over-fitting by using a separating hyperplane with maximum margin.

SVMs compute a hard-margin separating hyperplane for linearly separable data sets. It is enough to perfectly separate the examples. The linearly inseparable case is more difficult. In this case, a *soft-margin* separating hyperplane and/or the so-called kernel trick are necessary. Hard-margin and soft-margin separating hyperplanes refer to those that forbid or allow misclassifications, respectively.

SVMs Classifiers for Linear Separable Case

Considering a linearly separable data set X, a linear decision function can be determined. Because X is linearly separable, no sample in X satisfies

$$\omega^T x + b = 0 \tag{2.13}$$

we have, then

$$\omega^T x_i + \omega_0 \begin{cases} \geq +1 \ \forall \ y_i = +1, i = 1, ..., N \\ \leq -1 \ \forall \ y_i = -1 \end{cases} \tag{2.14}$$

It is not difficult to see that (2.14) is equivalent to

$$y_i(\omega^T x_i + b) = y_i(< \omega, x_i > +b) \geq 1, i = 1, ..., N \tag{2.15}$$

Assuming that the hyperplanes (2.17) and (2.18) include both at least one element of X, the optimal separating hyperplane (2.16) is in the middle of them [28].

$$\omega^T x + b = 0 \tag{2.16}$$

$$\omega^T x + b = +1 \tag{2.17}$$

$$\omega^T x + b = -1 \tag{2.18}$$

The generalization region of a classifier of this type is the space within the hyperplanes (2.17) and (2.18).

Definition 6 (Margin) *The distance between the separating hyperplane and the closest object to the hyperplane is called **the margin**.*

Definition 7 (Maximum separating hyperplane) *It is the separating hyperplane that has the greatest margin. This is considered the optimal separating hyperplane, and has the best generalization capability.*

The optimal separating hyperplane

In Figure 2.5, the points x_1 and x_2 satisfy:

$$\begin{cases} y_1(w^T x_1 + b) = 1, y_1 = +1 \\ y_2(w^T x_2 + b) = 1, y_2 = -1 \end{cases} \tag{2.19}$$

Projecting x_1 and x_2 on vector ω

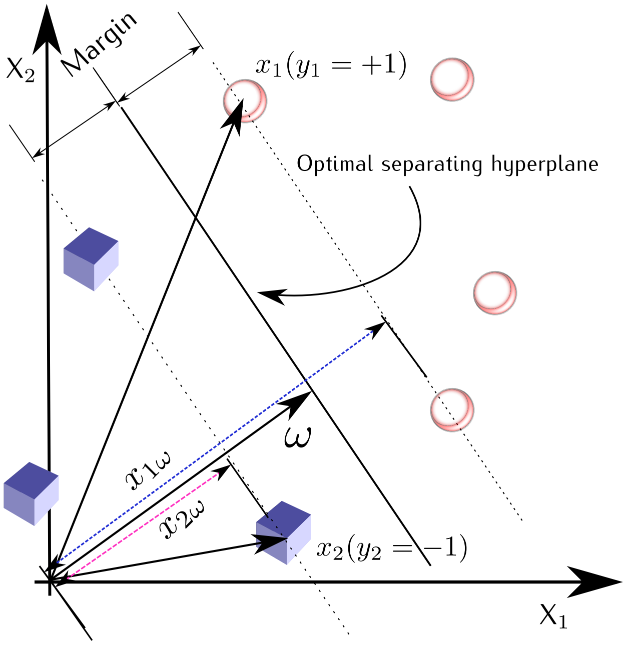


Figure 2.5: **Margin computation for linearly separable case**

$$x_{1\omega} = \frac{< x_1, \omega >}{< \omega, \omega >} = \frac{x_1^T \omega}{< \omega, \omega >} \tag{2.20}$$

$$x_{2\omega} = \frac{< x_2, \omega >}{< \omega, \omega >} = \frac{x_2^T \omega}{< \omega, \omega >} \tag{2.21}$$

The margin is determined the of difference of projections (2.20) and (2.21)

$$\begin{aligned} Margin &= \frac{< x_1, \omega >}{< \omega, \omega >} - \frac{< x_2, \omega >}{< \omega, \omega >} \\ &= \frac{< x_1, \omega > - < x_2, \omega >}{< \omega, \omega >} \end{aligned} \tag{2.22}$$

Using (2.19) we have

$$< x_1, \omega >= 1 - b \tag{2.23}$$

and

$$< x_2, \omega >= -1 - b \tag{2.24}$$

Substituting (2.23) and (2.24) into (2.22) we obtain

$$Margin = \frac{1 - b - (-1 - b)}{< \omega, \omega >} = \frac{2}{< \omega, \omega >} = \frac{2}{\omega^2} \tag{2.25}$$

In order to compute the maximum separating hyperplane (maximize (2.25), it is necessary to minimize ω^2. The following nonlinear optimization problem arises:

$$\min_{\omega} \quad \omega^2 \tag{2.26}$$

s.t.

$$y_i(\omega^T x_i + b) \geq 1, i = 1, ..., N \tag{2.27}$$

Non linear programming problems with equality and inequality constraints have, in general, the following form:

$$min \ f(x) \quad x \in R^d \tag{2.28}$$

subject to

$$h_j(x) = 0 \ j = 1, ...m$$

$$g_j(x) \geq 0 \ j = m + 1, ...p$$

The optimization of non linear problems is based on Karush-Kuhn-Tucker (KKT) conditions, which were proved in 1951.

The KKT conditions state that, if both the objective function $f(x)$ and all its constraints are once differentiable at the point x^*, and the first-order constraint qualification holds at x^*; then, the necessary conditions x^* to be a local minimum are that exist Lagrange multipliers α^* and β^* such that satisfy

$$h_j(x^*) = 0 \;\; j = 1, ...m \tag{2.29a}$$

$$g_j(x^*) \geq 0 \;\; j = 1, ...p \tag{2.29b}$$

$$\alpha^* g_j(x^*) = 0 \;\; j = 1, ...p \tag{2.29c}$$

$$\alpha^* \geq 0 \tag{2.29d}$$

$$\nabla L(x^*, \beta^*, \alpha^*) = 0 \tag{2.29e}$$

Where $L(x, \beta.\alpha)$ in (2.29e) is the Lagrangian defined in (2.30).

$$L(x^*, \beta^*, \alpha^*) = f(x) + \sum_{j=1}^{m} h_j(x) - \sum_{j=m+1}^{p} g_j(x) \tag{2.30}$$

The Lagrangian of the Quadratic Programming Problem (QPP) (2.26) becomes

$$L(\omega, b, \alpha) = \frac{\omega^2}{2} - \sum_{i=1}^{N} \alpha_i \{y_i(\omega^T x_i + b) - 1\} \tag{2.31}$$

With $\alpha_i \geq 0$.

The optimal solution of (2.31) is given by the saddle point, at which (2.31) is minimized with respect to vector ω and offset b (primal space). At the same time, it is maximized with respect to Lagrange multipliers α_i (dual space). The gradient of (2.31) with respect to the primal variables vanishes at the saddle point, giving:

$$\begin{aligned}\frac{\partial L(\omega, b, \alpha)}{\partial \omega} &= \frac{\partial \frac{\omega^2}{2}}{\partial \omega} - \frac{\partial \sum_{i=1}^{N} \alpha_i \{y_i(\omega^T x_i + b) - 1\}}{\partial \omega} \\ &= \omega - \sum_{i=1}^{N} \alpha_i x_i y_i = 0 \qquad (2.32)\end{aligned}$$

and

$$\begin{aligned}\frac{\partial L(\omega, b, \alpha)}{\partial b} &= \frac{\partial \frac{\omega^2}{2}}{\partial b} - \frac{\partial \sum_{i=1}^{N} \alpha_i \{y_i(\omega^T x_i + b) - 1\}}{\partial b} \\ &= \sum_{i=1}^{N} \alpha_i y_i = 0 \qquad (2.33)\end{aligned}$$

and from (2.29c)

$$\alpha_i \{y_i(\omega^T x_i + b) - 1\} = 0 \; i = 1, ..., N \qquad (2.34)$$

Using (2.32) and (2.33)

$$\omega = \sum_{i=1}^{N} \alpha_i x_i y_i \qquad (2.35)$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \qquad (2.36)$$

The samples X that fulfill $\alpha_i \neq 0$ (2.34) are called the **support vectors** (SVs). The SVs satisfy $y_i(\omega^T x_i + b) = 1$.

Substituting (2.35) and (2.36) in Lagrangian (2.31) produces

$$\begin{aligned}
L &= \frac{1}{2}\omega^T\omega - \sum_{i=1}^{N}\alpha_i\left\{y_i\left(\omega^T x_i + b\right) - 1\right\} && (2.37)\\
&= \frac{1}{2}\omega^T\omega - \omega^T\sum_{i=1}^{N}\alpha_i y_i x_i - b\sum_{i=1}^{N}\alpha_i y_i + \sum_{i=1}^{N}\alpha_i\\
&= \frac{1}{2}\omega^T\omega - \omega^T\omega - b\sum_{i=1}^{N}\alpha_i y_i + \sum_{i=1}^{N}\alpha_i\\
&= -\frac{1}{2}\omega^T\omega + \sum_{i=1}^{N}\alpha_i\\
&= -\sum_{i,j=1}^{N}\alpha_i,\alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{N}\alpha_i\\
&= -\sum_{i,j=1}^{N}\alpha_i,\alpha_j y_i y_j < x_i, x_j > + \sum_{i=1}^{N}\alpha_i
\end{aligned}$$

the dual problem is now

$$\max_{\alpha}\ \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1,j=1}^{N}\alpha_i\alpha_j y_i y_j x_i^T x_j \qquad (2.38)$$

s.t.

$$\sum_{i=1}^{N}\alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

Solution of (2.38) yields

$$\omega^* = \sum_{i=1}^{N} \alpha_i^* y_i x_i \tag{2.39}$$

$$b = \frac{1}{N_{sv}} \sum_{i=1}^{N_{sv}} (\frac{1}{y_i} - x_i^T \omega^*), \ \ x_i \text{ is a SV} \tag{2.40}$$

Where

N_{SV} The number of support vectors.

Given a sample **x**, it is classified according to the following discriminant function:

$$\text{IF } \left(w^*\mathbf{x} + b = \sum_{i=1}^{N} \alpha_i^* y_i x_i^T \mathbf{x} + \frac{1}{N_{sv}} \sum_{i=1}^{N_{sv}} (\frac{1}{y_i} - x_i^T \omega^*) \right) \begin{cases} > 0 \text{ THEN } y_i = +1 \\ < 0 \text{ THEN } y_i = -1 \\ = 0 \text{ THEN x is on the boundary} \end{cases} \tag{2.41}$$

SVMs Classifiers for Linearly Inseparable Cases

In the real-world, linearly separable data sets are not common. When the classes of a given data set X overlap, there is not feasible solution for the previous derivation of SVMs classifiers.

In order to achieve a solution, soft-margin hyperplanes are used. Mathematically, this means that the constraints (2.27) are relaxed, by introducing non negative slack variables ξ in them (see eq. (2.43)). This ensures that a feasible solution exists. Figure 2.6 illustrates a soft-margin optimal separating hyperplane in two dimensions.

The formulation of the optimization problem for this linearly inseparable case becomes then

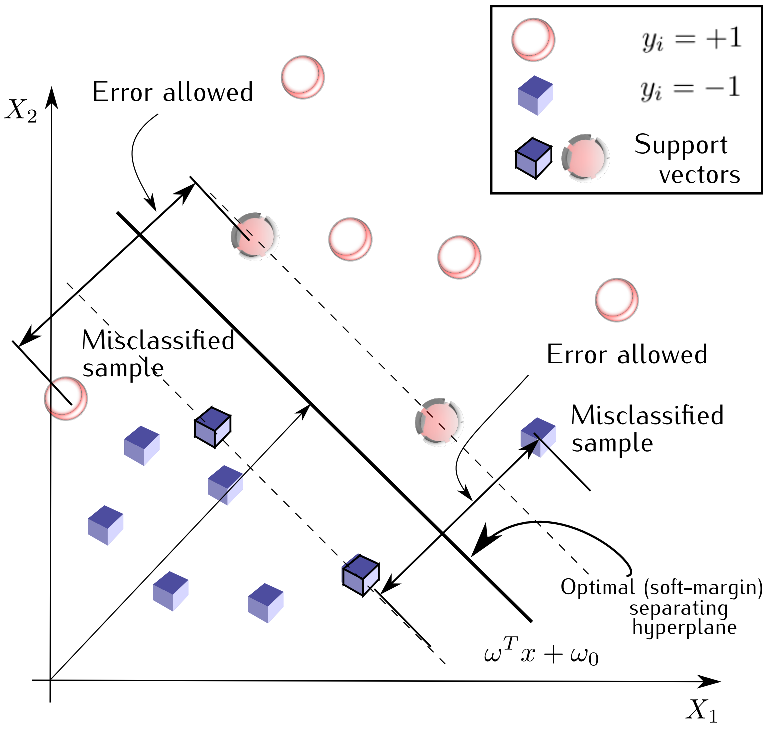


Figure 2.6: **Soft-margin computation for linearly inseparable case**

$$\min_{\omega} \quad \omega^2 + C \sum_{i=1}^{N} \xi_i^p \tag{2.42}$$

s.t.

$$y_i \left(\omega^T x_i + b \right) \geq 1 - \xi_i, i = 1, \ldots N \tag{2.43}$$

$$\xi_i \geq 0 \tag{2.44}$$

Where

C is a penalty weight that controls the trade-off between the maximization of the margin and the minimization of the classification error.

p is usually has the value 1 (L1 soft-margin support vector machine) or 2 (L2 soft-margin support vector machine).

Similarly to the linear separable case, the Lagrangian in this case is

$$L(\omega, b, \alpha, \beta) = \omega^2 + C\sum_{i=1}^{N} \xi_i^p - \sum_{i=1}^{N} \alpha_i \left\{ y_i \left(\omega^T x_i + b \right) - 1 + \xi_i \right\} - \sum_{i=1}^{N} \beta_i \xi_i \tag{2.45}$$

where

$\alpha_i \geq 0$ and $\beta_i \geq 0$ are Lagrange multipliers.

By the KKT conditions

$$\frac{\partial L(\omega, b, \alpha, \beta)}{\partial \omega} = 0 \tag{2.46}$$

$$\frac{\partial L(\omega, b, \alpha, \beta)}{\partial b} = 0 \tag{2.47}$$

$$\frac{\partial L(\omega, b, \alpha, \beta)}{\partial \beta} = 0 \tag{2.48}$$

and

$$\alpha_i \left\{ y_i \left(\omega^T x_i + b \right) - 1 + \xi_i \right\} = 0 \tag{2.49}$$

$$\beta_i \xi_i = 0 \tag{2.50}$$

$$\alpha_i \geq 0, \beta_i \geq 0, \xi_i \geq 0 \tag{2.51}$$

Proceeding as in the linearly separable case, it is possible to obtain:

$$\omega = \sum_{i=1}^{N} \alpha_i x_i y_i \tag{2.52}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.53}$$

$$\alpha_i + \beta_i = C, i = 1, ...N \tag{2.54}$$

Substituting in (2.45) produces:

$$\max_{\alpha} L(\omega, b, \alpha, \beta) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{2.55}$$

s.t.

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0 \quad i = 1, ..., N$$

Use of Kernels

For linearly inseparable cases, it is possible to obtain an optimal separating (soft-margin) hyperplane. The generalization capability of such separating hyperplane could be degraded, and so does the accuracy of the classifier. This is because the slack variables reduce the margin [28].

In order to improve the generalization capabilities of SVMs, the separating hyperplane is computed in a high-dimensional space, called the *feature space*. The data set becomes linearly separable in the feature space.

The instances $x_i \in X$ are mapped into the higher-dimensional feature space:

$$x \in R^d \mapsto [\phi_1(x), ..., \phi_d(x)]^T \in R^m \tag{2.56}$$

where

m : Dimension of feature space ($m > d$)

ϕ: The mapping, it is usually a non linear function.

The linear decision model is represented as:

$$f(x) = \sum_{i=1}^{N} \omega_i \phi_i(x) + b = \sum_{i=1}^{N} y_i \alpha_i \langle \phi(x_i), \phi(x) \rangle + b \tag{2.57}$$

A problem with eq. (2.57) is its complexity. This problem is more accentuated when the dimension of the feature space is large. Evaluation of eq. (2.57) is impossible, if the dimension of feature space is infinite.

In order to avoid the direct computation required to map examples to the feature space, a kernel is used; it is defined as follows:

Definition 8 (Kernel[40]) *A function $k : X \times X \mapsto R$ is called kernel if exists ϕ and H such that*

$$k(x_i, x_j) = \left\langle \phi(x_i), \phi(x_j) \right\rangle \quad \forall x_i, x_j \in X \tag{2.58}$$

where
X a non empty set
H a Hilbert space
$\phi : X \mapsto H$ a feature map

In the previous definition, $K = \left(k(x_i, x_j)\right)_{i,j=1}^{N}, x_{i,j} \in X$ is named the Kernel matrix, or the Gram matrix.

Not all functions are a kernel. Mercer's theorem 2.4 helps to identify which functions can be a kernel [40][27][28][41]:

Theorem **2.4 (Mercer theorem)** *A symmetric function K(x,y) can be expressed as an inner product*

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

for some ϕ if and only if K(x,y) is positive definite, or equivalently:

$$\begin{pmatrix} K(x_1, x_1) & K(x_1, x_2) & \dots \\ K(x_2, x_1) & \ddots & \\ \vdots & & \\ & & \end{pmatrix}$$

is positive definite for any collection $\{x_1, x_2, ..., x_n\}$

Kernel functions allow to compute dot products in feature space without need for an explicit mapping, so (2.57) becomes

$$f(x) = \sum_{i=1}^{N} y_i \alpha_i k(x_i, x) + b \tag{2.59}$$

The advantage of eq. (2.59) over eq. (2.57), lies on the number of operations required to compute the dot product. Evaluating a kernel function is not necessarily proportional to the number of features in the feature space.

Three of the most common kernels widely used in practice are the following ones.

Kernel name	Kernel function
Gaussian RBF	$K(x_i, x_j) = e^{-\frac{x_i - x_j{}^2}{2\sigma^2}}$
Polynomial	$k(x_i, x_j) = \left[x_i^T x_j + 1\right]^d$
Sigmoid	$k(x_i, x_j) = tanh\left(\left[x_i^T x_j + b\right]\right)$

2.2.3 Decision Trees

Decision or induction trees make partitions of the input space, recursively. At each phase, the partitions are purer. Here, the term pure, means that the partitions contain the majority of its samples with the same label.

In general, the training methods for decision trees select an attribute and use it to partition the data. The selection of an attribute is based on a measure, which determines the best way to partition a subset. The measures are typically defined in terms of the class distribution of the instances, before and after splitting a subset. In practice, these measures are based upon the degree of impurity of the produced partitions. The most common impurity measures are:

$$Entropy(t) = -\sum_{i=0}^{c-1} p(i|t)log_2 p(i|t) \tag{2.60}$$

$$Gini = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2 \tag{2.61}$$

$$Classification\ error(t) = 1 - max[p(i|t)] \tag{2.62}$$

Graphically, decision trees can be seen as a flowchart-like tree structure. Such a structure has a *root node*, which is the topmost in the tree. A decision tree has *internal nodes* (those that are not leaves), that represent a test on an attribute; it also has *terminal* or *leaf nodes*, that maintain a class label. The internal nodes have exactly one incoming edge and two or more outgoing edges (branches). Each branch from an internal node represents an outcome of the test. Figure 2.7 [42] shows an example of a decision tree used to solve a toy example: a mammal classification problem.

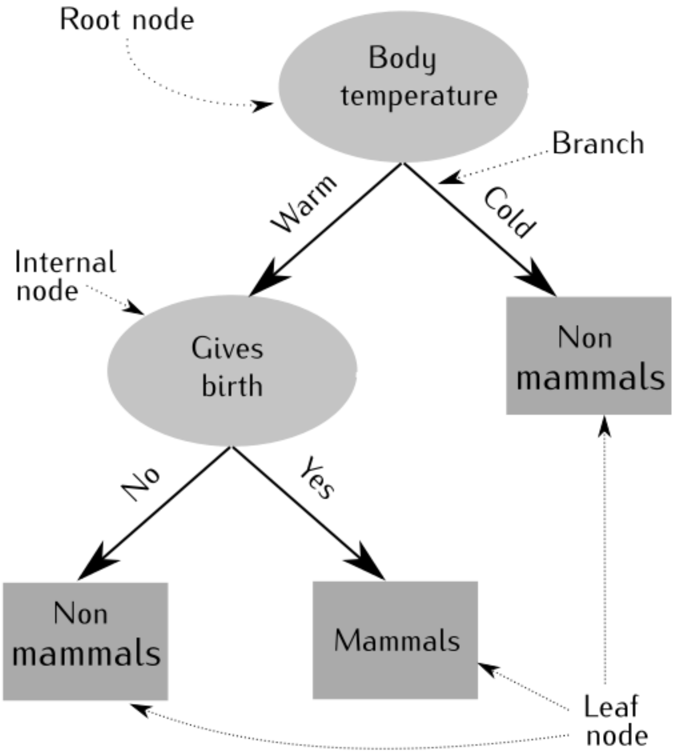


Figure 2.7: **An example of a decision tree**

The task of finding the best decision tree is computationally infeasible in most cases[42]. In practice, the methods used to construct sub-optimal (but accurate) decision trees use a greedy strategy.

2.2.4 Fisher's Linear Discriminant

Fisher's linear discriminant can be considered as a method for linear dimensionality reduction. The method is based on minimizing the projected class overlapping that maximizes the distance between class means, while minimizing the variance within each class. Figure 2.8 shows an example of the essence of this method.

Consider a given data set defined as in eq. (2.1). For binary classification problem, this set can be partitioned into two disjoint subsets $D^+ = \{x_i \in X \ s.t. \ y_i = +1\}$ and $D^- = \{x_i \in X \ s.t. \ y_i = -1\}$. The means μ^+ and μ^- are computed with:

$$\begin{aligned} \mu^{+} &= \frac{1}{|D^{+}|}\sum_{i=1}^{|D^{+}|} x, \;\; x \in D^{+} \\ \mu^{+} &= \frac{1}{|D^{-}|}\sum_{i=1}^{|D^{-}|} x, \;\; x \in D^{-} \end{aligned} \tag{2.63}$$

Let $\omega \in R^{d}$ be a vector. If every object in X is projected on ω, the means of projections are given by

$$\begin{aligned} \mathbf{m}^{+} &= \frac{1}{|D^{+}|}\sum_{i=1}^{|D^{+}|} p_{i}, \;\; x \in D^{+} \\ \mathbf{m}^{-} &= \frac{1}{|D^{-}|}\sum_{i=1}^{|D^{-}|} p_{i}, \;\; x \in D^{-} \end{aligned} \tag{2.64}$$

Where

$p_i = \omega^{T} x_i$

Combining (2.63) and (2.64), we obtain:

$$\begin{aligned} \mathbf{m}^{+} &= \frac{1}{|D^{+}|}\sum_{i=1}^{|D^{+}|} p_{i} = \frac{1}{|D^{+}|}\sum_{i=1}^{|D^{+}|} \omega^{T} x = \omega^{T}\frac{1}{|D^{+}|}\sum_{i=1}^{|D^{+}|} x = \omega^{T}\mu^{+} \\ \mathbf{m}^{-} &= \frac{1}{|D^{-}|}\sum_{i=1}^{|D^{-}|} p_{i} = \frac{1}{|D^{-}|}\sum_{i=1}^{|D^{-}|} \omega^{T} x = \omega^{T}\frac{1}{|D^{-}|}\sum_{i=1}^{|D^{-}|} x = \omega^{T}\mu^{-} \end{aligned} \tag{2.65}$$

Fisher's linear discriminant method searches for a vector ω that maximizes the separation between means $\mathbf{m}^{+}$ and $\mathbf{m}^{-}$, and at the same time, that minimizes the scattering of subsets D^{+} and D^{-}.

The distance between $\mathbf{m}^{+}$ and $\mathbf{m}^{-}$ is

$$|\mathbf{m}^{+} - \mathbf{m}^{-}| = |\omega^{T}\mu^{+} - \omega^{T}\mu^{-}| \tag{2.66}$$

The scatter, also called the within-class variance [36][34], is defined as:

$$\tilde{s}_+^2 = \frac{1}{|D^+|}\sum_{i=1}^{|D^+|}(y_i - \mu_+) \tag{2.67}$$

$$\tilde{s}_-^2 = \frac{1}{|D^-|}\sum_{i=1}^{|D^-|}(y_i - \mu_-)$$

In order to measure the scattering of D^+ and D^-, the following optimization problem must be solved:

$$\max_{\omega} J(\omega) = \frac{|\mathbf{m}^+ - \mathbf{m}^-|^2}{\tilde{s}_+^2 + \tilde{s}_-^2} \tag{2.68}$$

The denominator in (2.68) is known as total within-class variance. A more useful form of is the following:

$$\max_{\omega} J(\omega) = \frac{\omega^T S_B \omega}{\omega^T S_W \omega} \tag{2.69}$$

Where

S_B=$(\mu_- - \mu_+)(\mu_- - \mu_+)^T$ called the between-class covariance matrix.

S_W= $\sum_{x_i \in D^+}(x_i - \mu_+)(x_i - \mu_+)^T + \sum_{x_j \in D^-}(x_j - \mu_-)(x_j - \mu_-)^T$ called the total within-class covariance matrix.

Solution of (2.69) is

$$\omega = S_W^{-1}(\mu_+ - \mu_-) \tag{2.70}$$

Computing eq. (2.70) consumes $O(d^2|X|)$ time.

A problem occurs with this classifier when the data distribution is multi-modal; furthermore, when there exists overlapping between classes, the vector ω cannot be enough to clearly discriminate between classes. To face this problem, kernelized versions of Fisher's linear discriminant have been proposed.

2.3 Model Evaluation

The evaluation of performance of classifiers is very important. A classifier is evaluated to know how well it predicts unobserved data. The capability of a classifier to predict on independent data is called *generalization*. Generalization extremely important in practice, because it is used to select a models; it also gives a measure on the quality of a model (model assessment)

[43].

The most common measurement to evaluate the performance of a classifier is through its *classification accuracy*.

Definition 9 (Clasification accuracy of a classifier) *It is the percentage of examples in X_{test} that are correctly classified by the classifier. Accuracy is also known as recognition rate of a classifier and is computed with*

$$Acc = \frac{N_{correct}}{N}$$

Where

$N_{Correct}$: *Number of examples correctly classified by the classifier.*

$N = |X_{Test}|$: *Size of testing data set.*

Two important measures related to the accuracy are *true error* and *error rate*.

Definition 10 (True error)

$$true\ error = \underset{x_i \in D}{Pr}[h(x) \neq g(x)]$$

Where

X : *Input space,* $x_i \in X$.

$h(x)$: *The classifier (hypothesis)*

$g(x)$: *Target function, it is in general unknown*

D : *Distribution, it defines the probability of finding* x_i *in input space* X.

Definition 11 (Error rate) *The error rate is the percentage of misclassified examples in X_{Test} committed by the classifier.*

$$error\ rate = \frac{N_{error}}{N} = 1 - Acc$$

With

N_{error} : *Number of misclassification committed by classifier.*

The true error of a classifier $h(x)$, with respect to a target $g(x)$ and distribution D, is the probability that h misclassifies an instance x_i, randomly chosen according to D.

The error rate or sample error, is the error rate of a classifier over the available data. This error can be measured in practice as shown in Definition 11.

Because the true error cannot be measured, it is necessary to estimate it. In order to compute a confidence interval of it, the following formula is used in practice [44]:

$$confidence\ interval = error\ rate \pm Q\sqrt{\frac{error\ rate(1 - error\ rate)}{n}} \tag{2.71}$$

With Q having the following typical values: Q = 2.58 for 99% probability and 1.96 for 95% probability.

When the accuracy of a classifier is being evaluated, two general scenarios can happen. The first (less common) scenario occurs when there is plenty of data. The second (most common) happens when the amount of data is limited. For the fist case, the given data set X is divided into two subsets: X_{Test} and X_{Tr}:

$$X : \begin{cases} X_{Tr}, \text{Training } 70\% \\ X_{Test}, \text{Testing } 30\% \end{cases} \tag{2.72}$$

A classifier is trained using X_{Tr}, and its accuracy is measured on X_{Test}. The latter contains previously unseen instances. It is said that a classifier has a good *generalization capability*, if it accurately predicts the class of samples in X_{Test}.

For the second scenario, i.e., when the data set is not large, it is necessary to apply efficient sample re-use. Current techniques include cross-validation and the bootstrap. The original is now partitioned into three subsets: Training, Testing and Validation.

$$X : \begin{cases} X_{Tr}, \text{Training } 50\% \\ X_{Test}, \text{Testing } 25\% \\ X_{Val}, \text{Validation } 25\% \end{cases} \tag{2.73}$$

The Training set is used to calibrate the models. The Validation set is used to estimate the prediction error (model selection). The Test set is used for assessment of the generalization error of the final chosen model [43].

There are several techniques to create the subsets from a given data set X; most common techniques are Holdout, Random sub sampling, Cross validation, and Bootstrap.

In the Holdout selection method, the given X is randomly partitioned into two disjoint sets:

$$\begin{aligned} X &= X_{Tr} \cup X_{Test} \\ \varnothing &= X_{Tr} \cap X_{Test} \end{aligned} \tag{2.74}$$

In the random sub sampling technique, the holdout method is applied a number of K times, and the overall accuracy estimate is taken as the average of the accuracies obtained from each iteration.

For the cross validation method, there are three variants: The k-fold cross validation, leave-one-out and stratified cross-validation.

The k-fold cross validation makes random K partitions from X. Each partition has approximately an identical size as the others. The classifier is trained and tested K times; each time a distinct partition is used for testing and the rest is used as the training data set. This approach is different from holdout and sub sampling, in the sense that each object is used the same number of times for training and once for testing. The estimated accuracy is the overall number of correct classifications from the K iterations, divided by the total number of objects in X.

For leave-one-out variant only one sample is “left out“ at a time for the test set. In stratified cross-validation, the folds are ranked so that the class distribution of the examples in each fold is approximately the same as that in the initial data [3].

The bootstrap method works by sampling X uniformly with replacement, i.e.; an object can be selected again every time with the same probability.

Alternatives to accuracy and error rate are the *specificity* and *sensitivity*. They take into account the proportion of negative examples (i.e. samples x_i s.t. $y_i = -1$) that are correctly detected and the ratio of positive examples that are correctly classified, respectively.

Definition 12 (Specificity)

$$specificity = \frac{t_{neg}}{Neg}$$

With

t_{neg} : *Number of instances* $x_i \in X_{Test}$ *s.t.* $y_i = -1$ *and that have been correctly detected by the classifier.*

Neg : *Number of instances* $x_i \in X_{Test}$ *s.t.* $y_i = -1$

Definition 13 (Sensitivity)

$$specificity = \frac{t_{pos}}{pos}$$

With

t_{pos} : *Number of instances* $x_i \in X_{Test}$ *s.t.* $y_i = +1$ *and that have been correctly detected by classifier.*

Pos : *Number of instances* $x_i \in X_{Test}$ *s.t.* $y_i = +1$

Both sensitivity and specificity are used with imbalanced data sets. These have a large number of instances of a type, and a few samples of the opposite type.

Another useful tool to observe how well (or bad) a classifier can predictthe class of instances is the *Confusion Matrix*. It helps to see how the errors are distributed across the classes. The confusion matrix is applicable to detect if a classification problem is complicated, by observing its diagonal.

A confusion matrix is a $|Y| \times |Y|$ matrix whose entry c_{ij} denotes the number of instances in $X_{Testing}$ having class i and the classifier has assigned a label j.

Table 2.2 shows an artificial example of the confusion matrix. Suppose $|X_{tr}| = 38$, then $c_{1,1} = 10$ and $c_{1,2} = 1$ means that the classifier has correctly classified ten samples and incorrectly one sample. The accuracy of the classifier is computed as

$$Acc = \frac{Trace(Confusion\ \ Matrix)}{|X_{Tr}|}$$

Table 2.2: **Example of the confusion matrix for a hypothetical binary classification problem**

		Predicted class	
		$y_i = +1$	$y_i = -1$
Actual class	$y_i = +1$	10	1
	$y_i = -1$	2	25

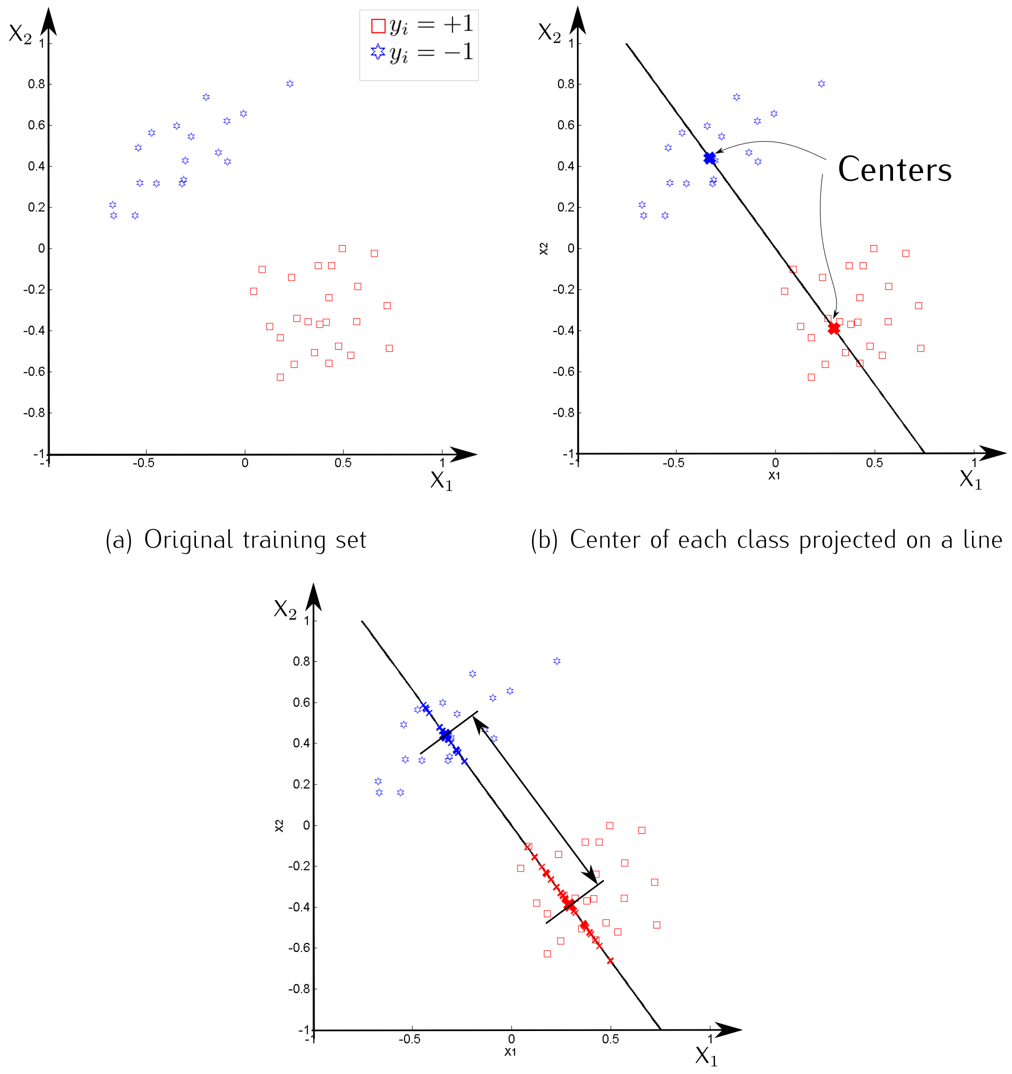
The *Receiver Operating Characteristic* (ROC) curve is also used to compare two classifiers. The ROC curve plots sensitivity versus specificity. As the parameters of a classification rule change. The ROC curve is considered a summary for assessing the trade-off between sensitivity and specificity. The area under the ROC curve is called the c-statistic.

2.4 Conclusions

Classification is a supervised learning method. It consists in assigning an object to one of a set of predefined categories (categorical values or labels). Supervised learning methods need a labeled data set to construct a model from it; the model is used to predict labels of instances that have not been seen before.

Some examples of classification methods are linear models, SVM, decision trees, probabilistic models, rule based models, artificial neural networks and ensembles of classifiers. An overview of some of these methods was presented in this Chapter. The SVM was studied in more detail than the other methods.

SVM optimizes the linear boundary (separating hyperplane) previously proposed in other models. The Perceptron and the linear models also use a separating hyperplane, but it is not the one with the largest margin. This is the main difference and advantage of SVM with respect to many other classifiers. Theoretically, the optimal separating hyperplane improves generalization capability. This has been confirmed in many real-world applications. The use of a kernel enables SVM to work in higher (infinity) dimensional spaces. This allows to classify data sets that are not linearly separable in the original input space.



(a) Original training set

(b) Center of each class projected on a line

(c) Projections on the vector that maximizes class separation

Figure 2.8: **An example of Fisher's linear discriminant**

3

Training Support Vector Machines with Large Data sets

It is impossible to recognize a wrong way without
knowing the right way
George Gurdjieff

In order to train SVM with large data sets, a number of methods have been developed in the last years. These methods have been classified in the following categories: data reduction, decomposition, variants of SVM and other methods that use heuristics or parallelism. In this Chapter we describe these methods. As a preamble to the algorithms developed in this research, we show, experimentally, that the examples close to the decision boundaries are good candidates to be support vectors.

Training methods for SVM are usually classified [45] according to the implemented strategies:

1. **Data reduction** For most cases, it has been found that after training a SVM, the number of SV is small compared to N [18][19]. The basic idea behind the data reduction strategy is to select instances with a high probability of being SV, and then, train SVM with them.

2. **Decomposition** The training time of SVMs can be reduced if only the active constraints of QPP are taken into account [46]. A similar idea to active sets methods for optimization is applied in decomposition methods. In the active set approach, two sets are used: the working set and the set of fixed variables. The optimization is made only on the working

set. For the case of SVM, the working set is usually composed of instances that violate the KKT conditions.

3. **Variants of SVM** Some researchers have modified the original QPP problem for SVM to speed up its training time, at the expense of losing classification accuracy [45]. Most of the variants of SVMs conclude with a system of linear equations, which is solved efficiently if the number of features is moderate, i.e., around 100.

4. **Others** We include here all those techniques not considered in the other categories: the use of parallel computations, caching, geometric approach and alpha seeding.

3.1 Data reduction methods

For the linearly separable case, the optimal separating hyperplane of SVMs depends completely on instances located closest to the separation boundary [7]. This instances are the so-called SV. It is well known that for most data sets, the number of SV is a small portion compared to N. Training an SVM using only the SV yields the same hyperplane than that obtained with the whole data set. The training time is certainly the shortest in the former case. This is the main motivation for attempting to reduce the size of the data sets before training an SVM.

In order to achieve the best classification accuracy, and improve the training time of SVMs, data reduction methods should preserve all the SV of data sets. These methods should remove all instances that are not SV. Data selection can be viewed as the incorporation of prior knowledge into SVMs.

The current approaches to achieve a reduction of training data sets can be classified into these categories:

1. Random Sampling methods

2. Distance-based methods

3.1.1 Random Sampling Methods

Random sampling techniques are useful when the $N \gg d$. The general idea behind this kind of algorithms, is to approximate the optimal separating hyperplane by using a small subset of the training set X_{tr}. In [47][48][49], iterative methods which introduce misclassified examples into a reduced set X_r are presented. At the beginning of the process, all samples in set X_{tr} are given the same weight, which is related with the probability of being chosen, and a number of K samples are selected randomly. This produces a separating hyperplane which can produce classification errors. The misclassified samples (violators) in X_{tr} are selected, and their probability of being chosen is increased. This process is iterated several rounds. Although it is reported that these methods converge to the global solution, there are some problems with them. First, it is difficult to choose the value of K. If it is small, then the QPP can be quickly solved, however, the produced model could have a high bias. If K is large, the QPP is costly to be solved. Second, it is not clear how the initial weight must be chosen, and how it should be updated during iterations. Algorithm 1 shows the pseudo code for these techniques.

Algorithm 1: Iterative Random Sampling, Generic Algorithm

```
Input  :
        $X_{tr}$: Training set
        $K$: Number of samples to select in each iteration
        $\lambda$: Initial weight
Output:
        $X_r$: A subset of $X_{tr}$
begin
    Assign to each sample in $X_{tr}$ a probability $\lambda$ of being chosen
    repeat
        $X_r$ $\Leftarrow$Select randomly $K$ samples from $X_{tr}$
        $h_{sub}$ $\Leftarrow$ Train SVM with $X_r$
        Classify $X_{tr}$ using sub optimal separating hyperplane $h_{sub}$
        Change the probability (weight $\lambda$) of each sample of being chosen according to
        whether it is misclassified or not
    until No misclassified samples;
    End
```

Reduced SVM (RSVM) [50] produces a very compact SVM, based on a random selection of samples. It is shown in [51][52] that uniform random sampling is the optimal robust selection scheme, in terms of several statistical measures. RSVM reformulates the QPP to come up with a much smaller problem, whose number of variables is about 1 to 10% of $|X_{tr}|$, and the number of constraints is equal to $|X_{tr}|$. The reformulation of QPP (eq. (2.42)) consists of using 2-norm ($p = 2$) and weighting the slack variables ξ_i by the factor $C/2$. The next problem shows these two changes:

$$\min_{\omega,b,\xi} \frac{1}{2}\left(\omega^T\omega + b^2\right) + \frac{C}{2}\sum_{i=1}^{N}\xi_i^2 \tag{3.1}$$

s.t.

$$y_i\left(\omega^T x_i + b\right) + \xi_i \geq 1, i = 1, \ldots N \qquad \xi_i \geq 0$$

RSVM translates the constrained QPP into an unconstrained one:

$$\min_{\omega,b,\xi} \frac{1}{2}\left(\omega^T\omega + b^2\right) + \frac{C}{2}\mathcal{F}\left(1 - y_i\left(\omega^T x_i - b\right)\right) \tag{3.2}$$

where $\mathcal{F}(\cdot) = max\,(0, \cdot)$.

In [53], it is shown that this problem is a strongly convex minimization problem, with a unique solution. The last transformation of RSVM, consists in using a smoothing, in order to apply an Armijo-Newton like method. RSVM uses about 10% of X_{tr} to solve the reformulated QPP, and the entire training set to refine the classification accuracy of the separating hyperplane. Refining means to select the best coefficients of the chosen kernel functions.

In practice, RSVM has a problem; the distance between selected examples must exceed a certain tolerance. This introduces an extra cost, which is not reported in the literature.

3.1.2 Distance-based methods

The distances between objects, or the distance from separating hyperplane to the examples, has been used as a guide to select instances from data sets.

Two heuristics commonly used are the following: (1) The samples which are located closest to others with opposite label are probably SV [18], (2) samples far from a hyperplane do not contribute to the definition of decision boundary [54].

By far, the Euclidean distance is the most commonly used in these algorithms; however, Mahalanobis and Hausdorff distances have been also utilized. The former gives a measure of length in a natural sense, but it only gives a reference between two objects. The Mahalanobis distance (eq. (3.3)) takes into account the correlation between the variables; it is used to measure similarities between multidimensional random variables or groups of objects. Mahalanobis distance can be seen as a measure of divergence or distance between groups in terms of multiple characteristics. The Hausdorff distance (eq. (3.4)) computes the maximum distance from a set to the nearest point in other set. $d(x, y)$ is any metric between the points x, y.

$$d_M(x_i) = \sqrt[2]{(x_i - \mu)^T S^{-1}(x_i - \mu)} \tag{3.3}$$

With

μ the mean

S the covariance matrix

$$max\{sup_{x \in X} inf_{y \in Y} \ d(x, y), sup_{y \in Y} inf_{x \in X} \ d(x, y)\} \tag{3.4}$$

The Mahalanobis distance is used in [55] to propose a data reduction algorithm; the examples on the boundaries are detected by measuring the relative differences of the Mahalanobis distances. The hypothesis is that important samples correspond to the ones with large relative differences of Mahalanobis distances. The use of Mahalanobis instead of Euclidean is justified because the former is invariant to rotations; it is also invariant to linear transformations of the input variables [55]. This method has the disadvantage of needing to compute all distances between every point and the set with different class.

In [18], an algorithm that uses the Hausdorff distance from an example to the objects with different class is proposed. This criterion selects patterns near to the decision boundary. The distance from a training example to the closest training examples of the opposite class is used as an approximation to the convex hull. This approach is related to the nearest point problem:

$$d(x_i) = \min_{j: y_j \neq y_i} \left\| x_i - x_j \right\| \tag{3.5}$$

In [56] [57], all the Euclidean distances between objects with opposite label are first computed. The closest samples are then selected. This method works in the input space. The algorithm in [45] shows how to select samples in feature space. The results shown in all these papers correspond to small data sets.

Algorithm 2: Distances Based Sampling, Generic Algorithm

```
Input :
        $X_{tr}$: Training set
Output:
        $X_r$: A subset of $X_{tr}$
begin
    Separate into positive and negative classes:
    $X^+ = \{x_i \in X_{tr}\ s.t.\ y_i = +1\}$
    $X^- = \{x_i \in X_{tr}\ s.t.\ y_i = -1\}$
    Compute distances from objects in $X^+$ to a point or reference of $X^-$
    $X_r \leftarrow$Choose objects whose distance satisfy a criterion
    End
```

In [18], an algorithm that detects instances near to the class boundaries is introduced. It selects all the examples with the same label, which are contained within a hyper sphere of maximum radius. In order to choose examples around a point, it is necessary to increase the diameter of the sphere. Because each instance is tested as a center, this represents the major drawback of this approach.

In general, distance based algorithms are not too efficient, because they have to compute a large number of distances. Their worst case is $O(n^2)$ in time and space. Algorithm 2 shows the general strategy of this kind of algorithms. An opportunity area is to develop novel methods that select examples by computing only a small number of distances.

Some proposals to reduce the size of training sets, are based on neighborhood properties of SV. The fundamental idea is that SV are located close to opposite class examples. To exemplify the idea, we use the figure 3.1. The area around two SV is represented by circles, and the Euclidean distances from an SV to the points within each area are represented by straight lines. Most of the time, a number of objects (neighbors) around an SV have an opposite class. This is more pronounced at points close to decision boundaries.

The pioneering work "'condensed nearest neighbor rule"', presented in [58],[59], takes advantage of this observation. The algorithm proposed in [59] discovers samples close to decision boundaries based on the *Mutual Neighborhood Value* (MNV). It is computed between any two samples of a set, as the sum of the conventional nearest neighbor ranks of these two samples with respect to each other. Mathematically, let be $x_i, x_j \in X$ with $X \subset R^d$, suppose

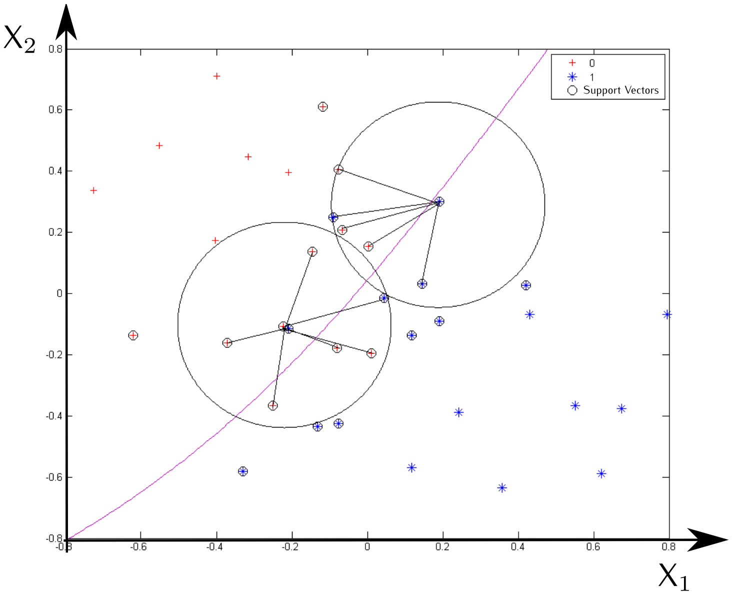


Figure 3.1: **Some neighbors of support vectors have opposite label**

x_j is the m^{th} nearest neighbor of x_i, and x_i is the n^{th} nearest neighbor of x_j, then MNV is defined as in eq. (3.6).

$$MNV(x_i, x_j) = m + n \text{ with m, n} \in \{0, 1, ..., N-1\} \tag{3.6}$$

The value of m and n equals to zero when $i = j$. Considering only the first K nearest neighbors of each point, then if either x_i or x_j, or both, are not found in each other's k-nearest neighborhood, then x_i and x_j do not belong to the mutual neighborhood. Samples that are near the decision boundary will have low values of MNV, and their distances will be short. Recently, in [60] a similar approach has been presented.

Some algorithms that select examples located between the overlap region around the decision boundary are shown in [61], [62], [63] [64], [18] and [65]. These algorithms are also based on neighborhood properties of SV. The general strategy is to begin with a subset of objects randomly chosen from the training set. Then, patterns near the decision boundary are found, and their neighbors are examined successively until all the patterns near the decision boundary are chosen and evaluated. Two main disadvantages of this method are: it is unsuitable for linearly separable cases, and it only works when the classes are overlapped. The method presented in [66] is quite similar, but it uses fuzzy C-means clustering to select samples on the boundaries of the class distribution.

Two methods devoted to discover points on boundaries of data sets are BRIM and BORDER. The BRIM algorithm was proposed in [67]. It selects points that are on the exterior boundaries of data distribution. BRIM works as follows: for each sample in the data set, its neighborhood is computed and the point with maximal density in this neighborhood is selected; this point is called an attractor. Then, the distribution feature of the points in the neighborhood in the direction of the attractor is calculated. This has a double effect: first, outliers are ignored, because there are no other points and the outlier itself is the attractor. Second, the neighborhood is split into two parts, one with a large number of points (the side of the attractor) and another with few points. The boundary degree of each point is computed, and those with a higher boundary degree are selected. The boundary degree is the relation between the number of objects in each part of the neighborhood.

The BORDER method presented in [68] detects instances on the boundaries by selecting those with the lowest number of reverse k-nearest neighbors (RkNN), see Definition 14 [69]. The RkNN of an object p consists of the points that look upon p as one of their k-nearest neighbors.

Definition 14 (Reverse K-Nearest Neighbor) *Given a data set X, a query point p, a positive integer k, and a distance metric $d(\quad)$, the reverse k nearest neighbors of p, denoted as $RkNN_p(k)$, is a set of points p_i such that $pi \in X$ and $\forall p_i; p \in RkNN_{pi}(k)$, where $RkNN_{pi}(k)$ are the k nearest neighbors of point p_i.*

The RkNN examines the neighborhood of an object p considering the "view" of the whole data set instead of the object itself; the number of RkNN decreases as the distance of a point from the center increases [68]. This fact is used by Border to the identification of boundary points that lie between two or more distributions. The Border algorithm begins by finding the k-nearest neighbors for each example in the data set; a RkNN number is assigned to each sample and those whose RkNN number is smaller than a user defined threshold detected as boundary points.

The problem with both BORDER and BRIM is that their complexity is about $O(n^2)$, so they are unsuitable for large data sets.

Different from detecting samples on the boundaries, a number of techniques begin selecting a small number of samples to compute an initial separating hyperplane. Then this number of samples is used to choose samples iteratively from the data set in order to compute a more accurate separating hyperplane. The Sequential Bootstrapped Accelerator (SBA) algorithm [70] uses this approach. SBA selects a quite small subset of the training data set to train an SVM; the computed hyperplane is used to look for the data which is farthest away from the hyperplane and that is misclassified. This iterates until no more misclassified points can be found. Figure 3.2 exemplifies this idea. Figure 3.2(a) shows a toy data set and the optimal separating hyperplane. In Figure 3.2(b) most examples have been removed, the point misclassified by the separating hyperplane would be used as the candidate in the next iteration. It is important to notice that this approach does not work for the linearly inseparable case. A kernel can be applied to compute the distances:

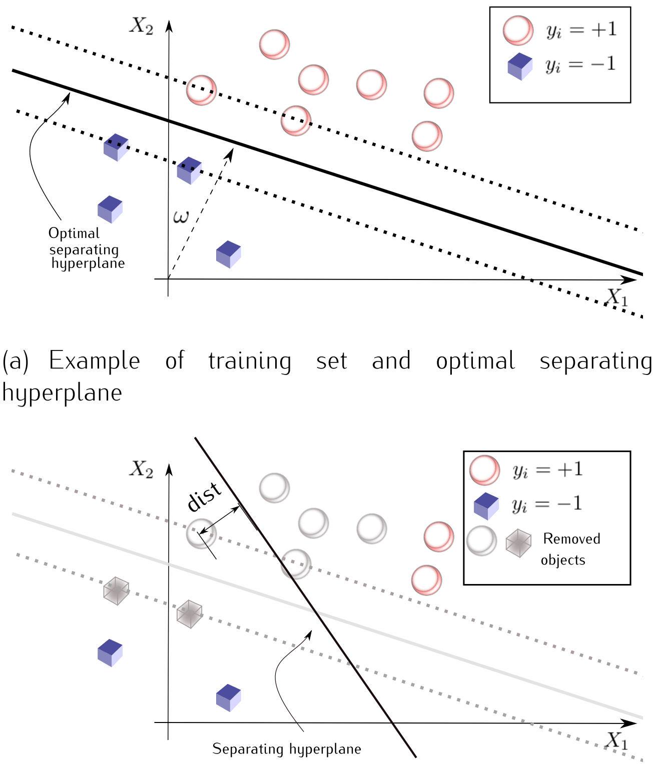
$$dist(x_i) = \sum y_i \alpha_i K(x, x_i) + b \pm 1$$

The main advantage of SBA is that the related QPP is very small, whereas its major drawback is that all samples need to be tested each time the separating hyperplane is updated. In [71] a similar strategy is used; however, a number of clusters are used instead of all points. These clusters are detected with Fuzzy C-means and minimum enclosing balls.

An algorithm based on minimum enclosing ball (MEB) clustering is presented in [72], as an extension of the method presented in [71]. The algorithm is conformed of four steps: (1) data selection via MEB clustering, (2) SVM classification, (3) declustering, and (4) second-stage SVM classification. After the training data are partitioned by the proposed clustering method, the centers of the clusters are used for the first SVM classification. Then, the clusters whose centers are support vectors are utilized in the second SVM classification. A disadvantage in the method presented in [72] occurs in the first step. It requires to know a priori two parameters; the number of balls and their radii. In [72], a guess is realized. Because it is not clear how to propose the parameters of the algorithm, an estimate of the number of balls can be obtained from the data, using the method of cross validation. However, this is a very time consuming task.

3.2 Decomposition Methods

The optimal separating hyperplane for an SVM is computed solving the QPP (eq. (2.26) or (2.42)). When a kernel is used, the kernel matrix (eq. (2.58)) is involved in the solution of the QPP.



(a) Example of training set and optimal separating hyperplane

(b) A small subset of training set and separating hyperplane

Figure 3.2: **Separating hyperplane is used to select examples**

The Gramm matrix is as large as the square of the training set size, and in addition, such matrix is dense; this makes classical optimization methods unsuitable to be directly applied on this situation [38][12]. To show this, consider the problem:

$$\min_{x} \frac{1}{2} x^T Hx \text{ with H definite positive} \quad (3.7)$$

$$\text{s.t.}$$

$$Ax \leq b, \; A \in R^{m \times n}$$

The explicit solution of problem (3.7) is given explicitly by

$$x = -H^{-1}\left(H^{-1}A^{T}\left(AH^{-1}A^{T}\right)^{-1}\right)b \tag{3.8}$$

The main drawback this solution, is that inverting a matrix takes about $O(n^3)$ time, so this kind of direct solutions should be avoided. The naive alternative of recomputing the kernel matrix when needed is not practical, because all its values are frequently used.

Decomposition methods tackle the problem of training an SVM by optimizing iteratively only on the variables belonging to a subset of tractable size. This is the so-called working or active set. The variables that do not belong to the working set are fixed and form the so-called fixed set. Decomposition methods can be classified into primal and dual methods. They aim for dual(primal) feasibility, while maintaining primal (dual) feasibility and complementary slackness. Algorithm 3 [73] shows the general scheme of the active set methods.

A clear advantage in this scheme, in addition to its proved convergence [74][75], is that its memory requirements grow linearly with the number of training examples. On the other hand, because only a fraction of the variables is being considered in each iteration, it is time consuming [45] if elements in the working set are not carefully selected. It has been observed that the active set method can oscillate nearby the solution [73].

The most important element in decomposition methods for them to converge quickly is the selection of the subset of variables in the working set [38]. One method, commonly used, consists in selecting those samples that violated the most KTT conditions [76][77][19].

One of the firsts decomposition methods was Chunking [54]. It repetitively obtains the maximum margin hyperplane from a number of instances (called the chunk), and then forms a new chunk with the SV from the previous solution and some recent instances.

The SVM^{light} [78] implementation selects working set members using the steepest feasible direction of descent. It has only a number of non-zero elements; the variables corresponding to these elements will constitute the current working set.

Probably, the most famous decomposing algorithm is SMO [12]. SMO considers the smallest size working set: only two training samples. In SMO, the parameters to optimize in each iteration are two if $\sum_{i=1}^{m} \alpha_i y_i = 0$ holds. This problem can be solved analytically without requiring the use of any library. The key elements in SMO are two: A heuristic step, that finds the best pair of parameters to optimize; and the use of an analytic expression to ensure that the Lagrangian increases monotonically [79].

Algorithm 3: Active Set Method, Generic Algorithm

```
Input  : A QPP to solve
         $x_1$: A feasible point
         $\mathcal{A}$: Initial active set
Output:
         $x^*$: Solution of QPP
begin
    while no solution found do
        if $\delta = 0$ does not solve (3.9) then
            Solve (3.9) for $s^k$
            Find $\alpha^{(k)}$ to solve (3.10) and set $x^{(k+1)} = x^{(k)} + \lambda^k s^{(k)}$
            if $\lambda^{(k)} < 1$ then
                $\mathcal{A} = \mathcal{A} \cup p$
        else
            Compute Lagrange multipliers $\lambda^{(k)}$ and solve (3.11) if $\lambda_q^{(k)} \geq 0$ then
                return $x^* = x^{(k)}$
            else
                remove q from $\mathcal{A}$
        k⇐k+1
        End
    Where

        $$\min_{\delta} \frac{1}{2} \delta T G \delta + \delta^T g^{(k)} \qquad (3.9)$$
        s.t
        $$a_i^T \delta = 0, \ i \in \mathcal{A}$$

        $$\alpha^{(k)} = min \left( 1, \min_{i: i \notin \mathcal{A}, a_i^T s^{(k)} < 0} \frac{b_i - a_i^T x^{(k)}}{a_i^T s^{(k)}} \right) \qquad (3.10)$$

        $$\min_{i \in \mathcal{A} \cap I} \lambda_i^{(k)} \qquad (3.11)$$
```

LibSVM [80] is an algorithm based on SMO with the improvement of a more advanced selection mechanism for the working set, which uses the second-order information method previously shown in [14].

3.3 Variants-based methods

Approximate versions of standard SVM have been proposed to improve its training time at the expense of accuracy. The least square SVM (LS-SVM) [81] changes the original QPP by using a linear system of equations that can be solved explicitly or by using a conjugate gradient method [81].

Original SVMs classify instances by assigning them a label, depending on which side of a separating hyperplane they are. The PSVM (Proximal SVM) [82] takes a different approach; the instances are classified by assigning them to the closest of two parallel lines, these lines (hyperplanes) cluster samples that cannot be linearly separated. The problem is thus reduced to that of solving a linear system in $O(n^3)$ time.

3.4 Other methods

In this section, techniques which are different from the previous ones are presented.

3.4.1 Parallel implementations

Rather than detecting samples that are likely SV, or decomposing the QPP in smaller sub problems, some algorithms try to speed up the training time of SVM by parallel implementations.

The Parallel implementation of QPP is difficult, this is because there is a strong data dependency [83]. Most approaches divide the training set into independent subsets to train SVMs in different processors, as in [83], [84] and [85]. In [86], the kernel matrix of SVMs is approximated by block diagonal matrices. The original optimization problem can be decomposed into hundreds of sub problems, which are easy to solve in a parallel fashion. A similar parallel computation of the kernel matrix for high dimensional data space is implemented in [87].

In [88], the authors use the variable projection decomposition technique. adapted to work in parallel; this work was inspired on the SVM^{light} implementation of [78].

A distributed SVM algorithm for row and column-wise data distribution is described in

[89], which cannot be used with non linear kernels. A parallel MPI-based decomposition solver for training support vector machines has been implemented in [90], whereas multi-processor shared memory (SMP) clusters have been introduced in [91].

3.4.2 Alpha seeding

Almost all SVM training methods begin by assuming that all Lagrangian multipliers (α_i) have zero values. The alpha seeding approach exposed in [92] consists in providing initial estimates of the α_i values, for the starting of QPP. According to the results produced by alpha seeding, it seems to be a practical method to improve the training time of SVMs [92]. Recently, in [93] has been proposed a similar method.

A very simple algorithm called DirectSVM is proposed in [94]. Instead of solving the related QPP, DirectSVM uses an iterative scheme based on two heuristics. This method requires to compute the closest pair of points in the data set, at the beginning, and several times during the iterations if necessary, which introduces an intensive workload. This is a disadvantage of the method.

3.4.3 On-line training

Training an SVM from scratch when one or more samples are added to the training data set is not computationally efficient. Incremental methods have been developed to address this problem. Currently, these training methods can be classified depending on how they work: (a) Algorithms that use batches of instances, e.g. [95][96], (b) Algorithms that use partitions and (c) other methods.

The general idea of methods of type (a) is as follows: retain the SV previously computed and then add one or more samples, afterwards solve a new optimization problem of small size. This approach resembles Chunking (see subsection. 3.2) but it is different because it produces low accuracy in on-line classification [97]. Variants of (a) have also been proposed. The method presented in [98] consists in updating a set of linearly independent vectors to construct the separating function. A single-pass algorithm for training SVM is presented in [82], and it takes advantage of PSVM[82]. In this algorithm, instead of finding the separating hyperplane, the instances that avoid linear separability are clustered by two hyperplanes. In the incremental step, the SV are retained whereas all the other samples are removed.

Some other methods (type (b)) create at least three partitions with the arrived instances:

the partition of SV, the partition of error samples and the remaining set [99]. The key is to move each sample to the correct partition and then update. Most algorithms use the KTT conditions to determine where to move samples. Examples of this technique are in [97][100][101].

There are several algorithms that use an approach different from those of type (a) and (b). The idea of controlling the changes of the optimal separating hyperplane by using the previously computed in the optimization problem as a weighted penalty was proposed in [102]. Recently, [103], used similar idea by coupling two SVMs combined with a time window. In [104] the authors use a property of the radial basis kernel, updating the separating hyperplane locally.

The "Huller" algorithm shown in [105] treats SVMs as NPPs, solving a simple optimization problem. This algorithm is very fast but cannot deal well with linearly inseparable or noisy data sets.

3.5 Preliminary experiments

It can be found in the literature that SVs are located near to decision boundaries [54][7][106] or are the closest to opposite class examples[18][107][108]. Other data reduction methods use simple random sampling [48][65][109][110]. In this section, we explore both approaches and present some results.

For the linearly separable case, the optimal separating hyperplane is determined by the closest points that belong to the convex hull of each class [7]. In Most cases these points are also close to opposite class examples [111][112]. The trivial idea for reducing the size of the training sets for classification with an SVM consists in selecting the closest objects with an opposite label.

Consider the toy data set shown in Figure 3.3. Figure 3.4 shows the smallest distances from positive(negative) objects to negative(positive) examples. The distances marked with a circle correspond to the real SV obtained after training an SVM with a linear kernel. Figures 3.5 and 3.6 show another toy example that is linearly inseparable.

The objects closest to the opposite class are also SVs for the last two toy examples. In order to explore the behavior of this approach to detect SV candidates, we executed an experiment using the following real-world data sets: Breast Cancer, Diabetes and Ionosphere. Table 3.1 shows the main features of them. We used the radial basis function kernel $\left(K(x_1, x_2) = exp(-\gamma \left\|x_1 - x_2\right\|^2)\right)$. The value of the parameter γ was set to value $1/N$ in all cases. The goal was to observe the general performance of the approach based on distances.

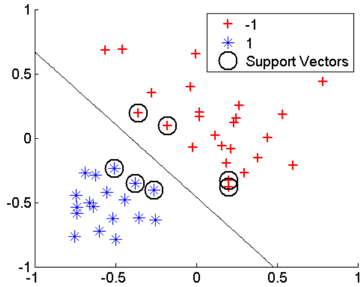


Figure 3.3: **Toy example of a linearly separable data set**

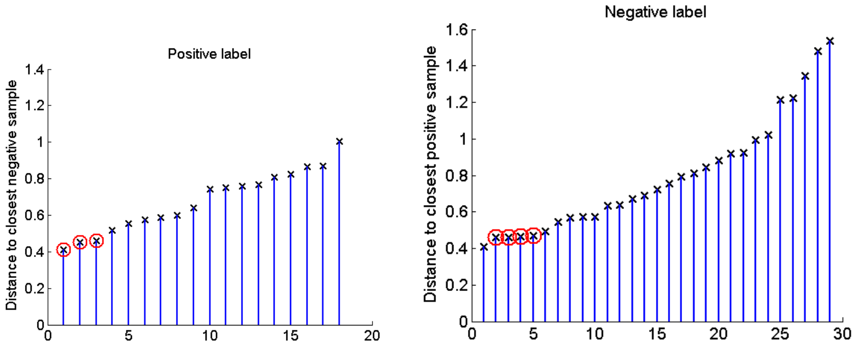


Figure 3.4: **Distances to the closest example, linearly separable case**

Many strategies to choose SV candidates from a data set using the distances as a guide can be designed, for example, (a) Select the initial M ($M \ll N$) objects closest to the opposite class, (b) prefer objects, randomly, with a probability for each object of being chosen set as a function of its distance or (c) Get rid off examples whose distance is greater than a certain threshold. We implemented strategy (a) in this experiment, i.e.; we chose a number of X_r objects that form the first Dist_{used} smallest distances. Strategy (b) depends on random numbers, and can give unrepeatable results, strategy (c) is similar to strategy (a).

Table 3.1: **Data sets for testing SV candidate selection using a naive approach**

Data set	**Size**	**Dim**	$\|y_i = +1\|$	$\|y_i = -1\|$
Breast-Cancer	683	10	444	239
Diabetes	768	8	500	268
Ionosphere	351	34	126	225

Figure 3.7 shows the general procedure to test and compare our methods with other ones.

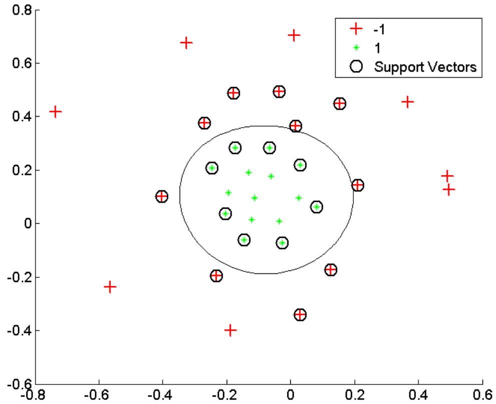


Figure 3.5: **Toy example of a linearly inseparable data set**

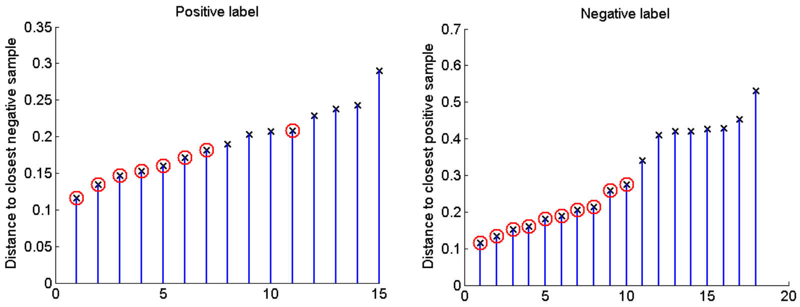


Figure 3.6: **Distances to the closest example, linearly inseparable case**

The training set is separated into training and testing set. A classification model is generated with the entire training set, this is referred as Model A in Figure 3.7. We apply our data reduction methods on the training set to obtain a "‘reduced data set"’. This is used to train a SVM classifier; a Model B is obtained. Both models are tested with the training set, the classification accuracy is measured. The training times of our methods include the time to reduce the training set and the time to train a SVM classifier.

Table 3.2 shows the results obtained. The reported results are the average of 100 runs of each experiment. The standard deviation is very large, and the accuracy is low if one takes about 1% of the training set. In general, the classification accuracy achieved with this naive selection method is lower than to classification accuracy achieved using the whole data set.

It is important to note that in this experiment, the training of the SVM took place in both feature and input space, whereas the selection of examples is executed in the input space. For both cases, the accuracy achieved with the approach based on distances gives approximate

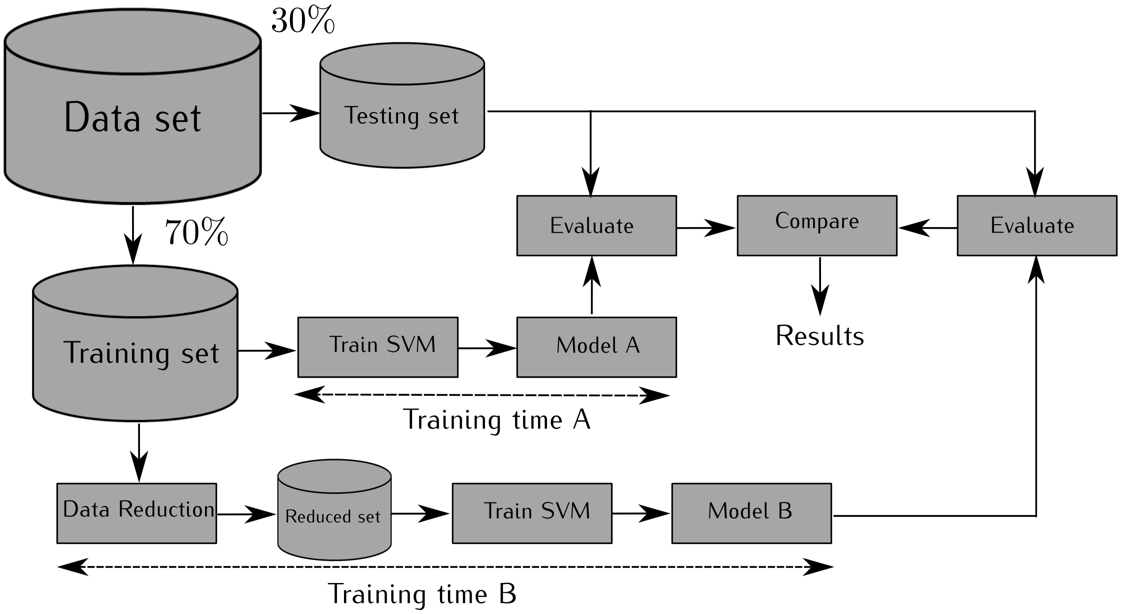


Figure 3.7:
Framework for comparing our methods

results even if the SVM training method is using a linear or a Gaussian kernel; this can be attributed to the fact that the Gaussian kernel preserves distance [112].

A problem with this approach is that the computation of distances has a complexity of about $O(n^2)$ in time and space; therefore, this method is unsuitable for large data sets. That is why we chose small real-world data sets for our experiment.

Data selection using simple random sampling is the cheapest computational method for reducing the size of training sets. This consists in taking a portion of the training set, randomly, and using it to build an SVM. Table 3.3 shows the results of 100 runs for each data set using this approach. It can be seen that the accuracy obtained with the 1% of training set is better than that obtained with the method based on distances; however, the standard deviation is considerably higher. Using about 10% of the training set seems a good tradeoff between accuracy, standard deviation and size of training set. It is important to notice that the standard deviation remains high for simple random sampling.

Table 3.2: **Results using selection based on distances**

Data set	$\|SV_{wG}\|$	$\|SV_{wL}\|$	**Dist**	**Dist**$_{used}$	$\|X_r\|$	$\|SV_{rG}\|$	$\|SV_{rL}\|$	**Acc**$_{wG}$**/stddev**	**Acc**$_{wL}$**/stddev**	**Acc**$_{rG}$**/stddev**	**Acc**$_{rL}$**/stddev**
Breast-Cancer	53	37.96	106, 116	10	14	11	11	97.14/1.00	96.7/1.10	64.76/10.76	57.45/10.55
				20	22	16	16			65.28/6.97	63.33/5.29
				40	37	20	30			72.82/8.79	91.91/4.23
				100	78	26	24			86.13/4.16	91.88/4.64
				200	102	31	29			94.07/1.78	93.98/2.44
				400	233	38	30			96.33/1.10	96.99/1.36
				1000	349	46	63			96.80/0.96	97.01/1.13
Diabetes	537	281	134, 000	10	19	19	19	64.92/2.90	76.91/2.34	58.87/8.48	58.87/11.48
				20	34	33	31			59.63/11.46	63.33/7.60
				40	61	58	52			65.25/2.80	64.96/7.49
				100	109	109	97			64.71/2.82	69.67/5.51
				200	191	191	131			65.12/2.88	74.03/4.09
				400	246	243	164			65.57/2.66	75.27/3.01
				1000	371	356	218			65.03/2.49	76.78/2.86
Ionosphere	168	75	3, 936	10	10	9	9	92.01/2.43	86.23/2.96	48.98/16.78	80.94/6.80
				20	11	11	10			56.08/15.14	80.83/6.13
				40	37	18	14			64.45/4.90	76.19/7.5
				100	59	30	22			65.02/4.77	76.78/5.76
				200	97	47	32			68.19/3.18	86.83/4.63
				400	127	60	43			74.97/3.01	79.66/4.01
				1000	158	72	52			83.67/2.33	83.37/3.30

The meaning of column names.

$|SV_{wG}|$: Number of SV using whole data set ans Gaussian kernel, $|SV_{wL}|$: Number of SV using whole data set and linear kernel

Dist : Total number of distances computed, Dist$_{used}$: The number of distances considered to select SV

$|X_r|$: Size if reduced training set, $|SV_{rG}|$: Number of SV that coincide with SV_{wG}

$|SV_{rL}|$: Number of SV that coincide with SV_{wL}

Acc$_{wG}$/$stdDev$: Accuracy/standard deviation achieved using whole data set and Gaussian kernel

Acc$_{wL}$/$stdDev$: Accuracy/standard deviation achieved using whole data set and linear kernel

Acc$_{rG}$/$stdDev$: Accuracy/standard deviation achieved using reduced data set and Gaussian kernel

Acc$_{rL}$/$stdDev$: Accuracy/standard deviation achieved using reduced data set and linear kernel

Algorithm 4: Naive data reduction based on distantes

```
Input :
    $X$: Set of points
Output:
    A subset of $X$
begin
    Separate the data set $X$ into two subsets:
    $X^+ = \{x_i \in X \text{ s.t. } y_i = +1\}$
    $X^- = \{x_i \in X \text{ s.t. } y_i = -1\}$
    For each $x_i \in X^+,\ x_j \in X^-$ compute $d_{i,j} = \sqrt{(x_i - x_j)^2}$
    Select examples in $X$ taking into account the $d_{i,j}$.
```

Table 3.3: **Results using selection based on simple random sampling**

Data set	Size X_r(%)	$\lvert SV_{rG}\rvert$	$\lvert SV_{rL}\rvert$	Acc$_{rG}$(%)/stdDev	Acc$_{rL}$(%)/stdDev
Breast-Cancer	1%	3	2	76.44/13.56	87.46/10.41
	10%	13	7	85.44/12.15	95.91/4.12
	50%	33	15	95.20/4.01	96.62/3.01
Diabetes	1%	5	3	52.01/19.19	66.52/14.01
	10%	53	24	65.57/17.33	73.06/10.77
	50%	268	137	65.54/13.78	76.71/9.32
Ionosphere	1%	2	1	57.76/16.90	60.00/16.50
	10%	20	13	74.79/10.08	77.50/9.00
	50%	70	42	89.27/4.29	85.75/5.80

The meaning of column names.

Size X_r(%): Portion of training set randomly selected

$|SV_{rG}|$, $|SV_{rL}|$, Acc$_{rG}$(%)/stdDev, Acc$_{rL}$(%)/stdDev: Same as table 3.2.

3.6 Conclusions

The state-of-the-art regarding methods for training SVMs was presented in this chapter. These methods are categorized in Data reduction, Decomposition, Variants of SVM and other algorithms.

Data reduction methods allow to improve the training time of an SVM at the expense of degrading classification accuracy. The observation that objects that define the optimal separating hyperplane are usually close to decision boundaries, or located close to opposite class examples, can be used as a guide to select SV candidates. We explored the performance of this approach using two naive strategies; the first is based on distances, and the second is based on simple random selection.

The objects whose distances are the closest to opposite class examples correspond to SV, regardless of the type of kernel adopted by the SVM (linear or a Gaussian). Using only a small number of candidates selected with respect to distance produces low accuracy, and high standard deviation. Simple random selection is probably the cheapest method, computationally speaking. It can achieve a higher accuracy than the previous approach, with a smaller number of examples. However, when using this apporach the standard deviation remains very high.

4 Data reduction method based on convex-concave hull

As above, so below

Hermes Trismegistus

The pioneering works on classification, proposed classifiers that used linear decision boundaries to predict the class of patterns. Although the first classifiers worked only on linearly separable cases, they provided powerful insights about the geometry of classification.

Linear decision boundaries are simple to understand in the input space. The linearly separable case can be graphically represented in two dimensions, along with a separating hyperplane. Intuition says that something similar happens in higher dimensions, and this is supported mathematically.

The classification of linearly separable data sets can be solved by using convex hulls. However, these data sets are not common in real-world applications. Convex hulls can be adapted for these problems. A reduced convex hull is used in [7] to classify linearly inseparable data sets.

In this chapter, we introduce a novel data reduction method called convex-concave hull. It improves the training time of SVM classifier. This method does not require to modify convex hulls. Also, it can be applied on both, linearly separable and inseparable cases.

4.1 Convex hull for classification

Linear classifiers, such as the simple perceptron and SVMs, compute linear decision boundaries; these are linear combinations of inputs. The SVM with linear kernel is pretty similar to the simple perceptron, in the sense that they both compute a linear decision boundary. However, in the former, the decision boundary is the optimal. The hyperplane found by SVM has the largest margin.

Figure 4.1 shows the geometry of the decision boundaries for the linearly separable case. The separating hyperplane is determined by vector ω, and the distance from the origin is given by $\frac{b}{||\omega||}$.

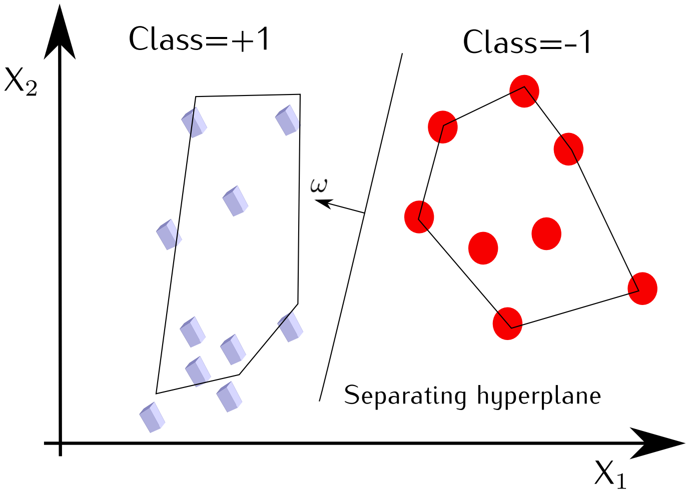


Figure 4.1: **Linear decision boundary for a binary classification problem**

For all linearly separable cases, no point can be expressed as a linear combination of the subsets $X^+ = \{x \in X$ s.t. class x is $+1\}$ and $X^- = \{x \in X$ s.t. class x is -1$\}$. The two convex hulls $\mathcal{CH}(X^+)$ and $\mathcal{CH}(X^-)$ do not intersect [7].

Computing the "best" separating hyperplane, is equivalent to finding the closest pair of points ($x^+ \in X^+$ and $x^- \in X^-$) that belong to $\mathcal{CH}(X^+)$ and $\mathcal{CH}(X^+)$, respectively [7][113][114]. In order to identify these two points, the following problem must be solved:

$$\min_{x_i \in X^+ x_j \in X^-} \sum_{i=1}^{M} \alpha_i x_i - \sum_{j=1}^{L} \alpha_j {x_j}^2 \tag{4.1}$$

s.t.

$$\sum_{i=1}^{M} \alpha_i = 1, \sum_{j=1}^{L} \alpha_j = 1, \;\; j = 1...M, i = 1, ..., N$$

and $\alpha_{i,j} \geq 0$

Where

$X^+ \subseteq R^d, X^- \subseteq R^d$

$\mid X^+ \mid = M, \mid X^- \mid = L$

The separating hyperplane is the normal vector to the line that joins the closest points. It is called "optimal", in the sense that it has the maximum margin. As explained in Chapter 2, the margin is the distance from the hyperplane to closest pattern. This is exemplified in Figure 4.2.

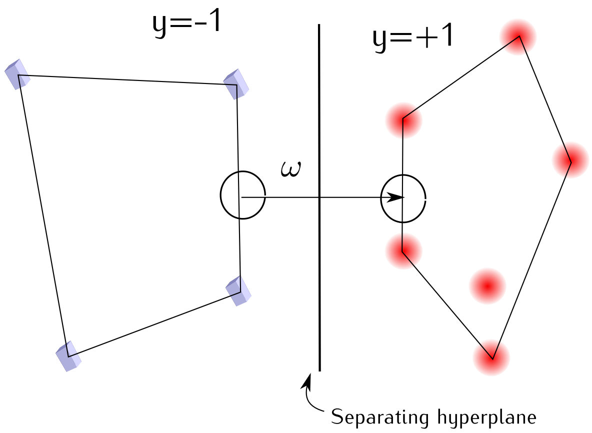


Figure 4.2: **A separating hyperplane is defined by the closest pair of points in the convex hulls**

The solution of problem (4.1) is equivalent to SVM [7][115]. This can be explained, based on the principle of Duality, in the area of Minimum Norm Problems. Observe that the hyperplanes must satisfy the equations

$$\begin{gathered}\langle x_i, \omega \rangle \geq \alpha, \ \forall \ i = 1, ..., M \\ \text{and } \exists \ x_i \in X^+ \ s.t. \langle x_i, \omega \rangle = \alpha\end{gathered} \tag{4.2}$$

$$\begin{gathered}\langle x_j, \omega \rangle \leq \beta, \ \forall \ j = 1, ..., L \\ \text{and } \exists \ x_j \in X^- \ s.t. \langle x_j, \omega \rangle = \beta\end{gathered} \tag{4.3}$$

The distance between these two hyperplanes — eq. (4.2) and eq. (4.3) — is computed by

$$D = \frac{\alpha - \beta}{\omega} \tag{4.4}$$

The optimal separating hyperplane is the one that maximizes D. In order solve this problem, ω must be minimized, and $(\alpha - \beta)$ maximized. Seen otherwise, t is necessary to minimize $-(\alpha - \beta)$.

Using these observations, the problem to be solved is transformed into

$$\begin{aligned} &\underset{\omega,\alpha,\beta}{max} \frac{1}{2}\omega^2 - (\alpha - \beta) \\ &\text{s.t.} \\ &\langle x_i, \omega \rangle \geq \alpha \ \forall \ i = 1, \ldots, M \\ &-\left\langle x_j, \omega \right\rangle \geq -\beta \ \forall \ j = 1, \ldots, L \end{aligned} \tag{4.5}$$

The Lagrangian of eq. (4.5) is

$$L(\omega, \alpha, \beta, u, v) = \frac{1}{2}\omega^2 - (\alpha - \beta) - \lambda_i \left(\langle x_i, \omega \rangle - \alpha\right) - \lambda_j \left(\left\langle x_j, \omega \right\rangle + \beta\right) \tag{4.6}$$

with

$$\begin{aligned} &i = 1, \ldots, M \text{ and } j = 1, \ldots, L \\ &\lambda_i \geq 0, \lambda_j \geq 0 \end{aligned}$$

The dual optimization problem is

$$\underset{\omega,\alpha,\beta,u,v}{max} L(\omega, \alpha, \beta, u, v) \tag{4.7}$$

According to the KKT conditions, for eq. (2.29a) to eq. (2.29e), we have

$$\begin{aligned} \nabla_\omega L(\omega, \alpha, \beta, u, v) &= \omega - \lambda_i x_i + \lambda_j x_j &= 0 \qquad (4.8) \\ \nabla_\alpha L(\omega, \alpha, \beta, u, v) &= -1 - \lambda_i &= 0 \qquad (4.9) \\ \nabla_\beta L(\omega, \alpha, \beta, u, v) &= 1 - \lambda_j &= 0 \qquad (4.10) \end{aligned}$$

The gradient of this Lagrangian is

$$\omega = \sum_{j=1}^{L} \lambda_j x_j - \sum_{i=1}^{M} \lambda_i x_i \tag{4.11}$$

From (4.9), (4.10) and (4.6):

$$\sum_{i=1}^{M} \lambda_i = 1,\ \ \lambda_i \geq 0 \tag{4.12}$$

$$\sum_{j=1}^{L} \lambda_j = 1,\ \ \lambda_j \geq 0 \tag{4.13}$$

Interchanging λ by α yields

$$\min_{x_i \in X^+ x_j \in X^-} \sum_{i=1}^{M} \alpha_i x_i - \sum_{j=1}^{L} \alpha_j {x_j}^2 = \omega^2$$

s.t.

$$\sum_{i=1}^{M} \alpha_i = 1, \sum_{j=1}^{L} \alpha_j = 1,\ \ j = 1...M, i = 1, ..., N$$

and $\alpha_{i,j} \geq 0$

which is equivalent to (2.26).

Most data sets are not linearly separable. In these situation, the convex hulls do intersect, and therefore, the solution to problem (4.1) does not exist. Current geometric methods for training SVMs, take the idea presented in [7]. It consists in shrinking or reducing the $\mathcal{CH}$ to avoid overlapping. In [8], a reduced convex hull (RCH) is used. The RCH is defined as

$$RCH(X, \mu) = \left\{ \omega : \omega = \sum_{i=1}^{k} a_i x_i \text{ where } \sum_{i=}^{k} a_i = 1, x_i \in X, \mu < 1 \text{ and } 0 \leq a_i \leq \mu \right\}$$

Given a linearly inseparable data set, the initially overlapping convex hulls can be reduced to become separable. This is achieved by selecting a suitable value of μ. Figure 4.3 shows an example of the shape of RCH.

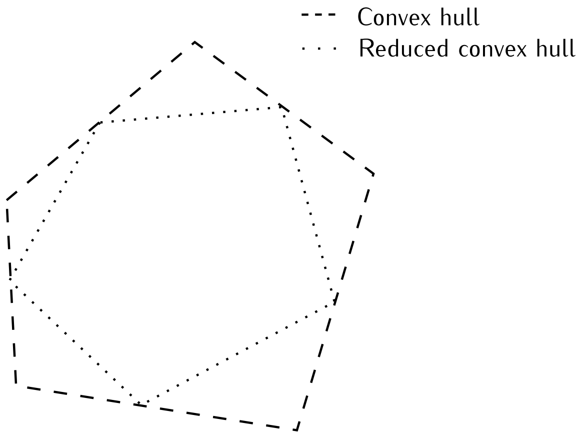


Figure 4.3: **Reduced convex hull,** $\mu = 0.5$

There are some problems with the reduced convex hulls: The parameter μ must be carefully selected. A large value of μ does not avoid intersection of convex hulls; a very small value of μ produces many vertices of the RCH. Another intrinsic problem with RCH is that it changes the shape of original set of points; the separating hyperplane misclassifies many examples.

One variant of a convex hull is the scaled convex hull (SCH), defined by

$$SCH(X, \lambda) = \left\{ \omega : \omega = \sum_{i=1}^{k} a_i \left(\lambda x i + (1 - \lambda) m \right) \right\}$$

With

$\sum_{i=}^{k} a_i = 1, x_i \in X, \lambda \leq 1$ and $0 \leq a_i \leq 1$

In SCH, the shape of the convex hull is maintained. Figure 4.4 shows a SCH in two dimensions.

The training times of SVMs using RCH or SCH are large, and the classification accuracy is degraded with both methods.

In our method, it is unnecessary to change the convex hull. Instead we detect the objects that are close to exterior boundaries of class distribution and use them to train a SVM.

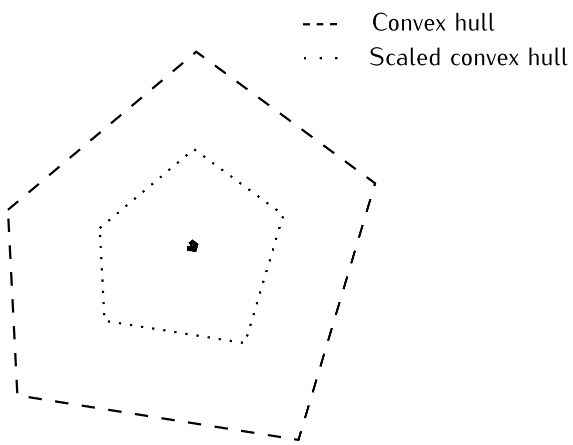


Figure 4.4: **Scaled convex hull,** $\lambda = 0.5$

In order to detect objects on the boundary, we use the convex hull first, and then, we create non-convex hulls.

4.2 Non-Convex Hull

The border points $\mathcal{B}(X)$ of a set $X \in R^2$ are the vertices of a *convex-concave hull*, if they satisfy the following properties:

1. The vertices of a convex-concave hull $\mathcal{B}(X)$ are all the vertices of a convex hull ($\mathcal{CH}$), plus the points that are "'close"' to the edges of $\mathcal{CH}(X)$:

$$\mathcal{B}(X) = V_{\mathcal{CH}(X)} \cup Close_{\mathcal{CH}(X)} \tag{4.14}$$

Where

$V_{\mathcal{CH}(X)}$ is the set of the vertices of the convex hull.

$Close_{\mathcal{CH}(X)}$ is the set of points that are close to the edges of $\mathcal{CH}(X)$.

The degree of closeness, is different from point to point. There is no unique set of border points.

If the points in $\mathcal{B}(X)$ are joined properly, it results in a concave polygon. A polygon is concave if any internal angle is greater than 180°. Figure 4.5 shows a concave polygon.

In general, the $\mathcal{B}(X)$ is not a convex hull.

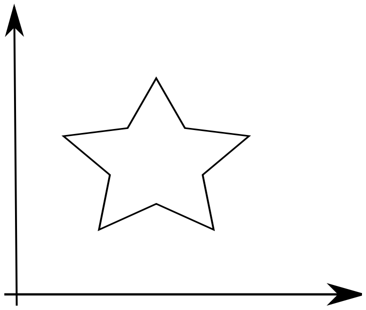


Figure 4.5: **An example of a concave polygon**

2. The border points should exclude at least one member of X:

$$\mathcal{B}(X) \subset X \tag{4.15}$$

A direct result is that the cardinality of $\mathcal{B}(X)$ fulfills:

$$\max |\mathcal{B}(X)| < N \tag{4.16}$$

3. The minimum size of $\mathcal{B}(X)$ is three (a triangle can never be concave). In most cases, when the cardinality of X is large, the minimum size of $\mathcal{B}(X)$ is equal to the number of vertices of $\mathcal{CH}(X)$.

The polygon formed with $\mathcal{B}(X)$ is convex, if there are no points "'close"' to any vertex of $\mathcal{CH}(X)$. Similarly to a convex hull, this polygon "'envelopes"' the elements in X. In general, $\mathcal{B}(X)$ is not convex, and we call it a convex-concave hull of X.

In the following subsection, we describe a method to detect the vertices of a convex-concave hull.

4.3 Searching for the vertices of convex-concave hull

We begin with the set of vertices in the $\mathcal{CH}(X)$, where $X = \{x \in R^2\}$. As shown in Figure 4.6, for any two adjacent vertices v_i and v_j of $\mathcal{CH}(X)$, all points in X must be located at one side of the line that passes through v_i and v_j.

We call v_i the starting point, and v_j the stopping point. The general procedure to detect objects "'close"' to the edge defined by these vertices is summarized as follows:

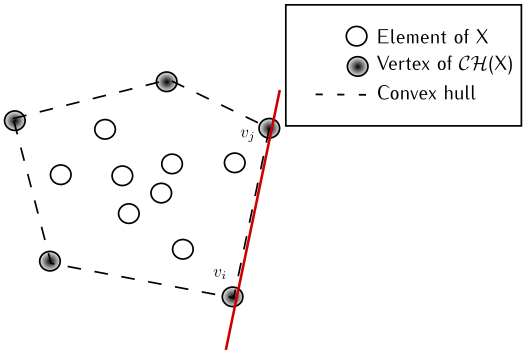


Figure 4.6: **The convex hull of a set of points X and two adjacent vertices**

1) The starting point is considering the "‘currentPoint"’.

2) The point that fulfills the following is considered "‘close"’ to the current edge:

- It is one of the K-nearest neighbors of currentPoint.
- It has the smallest angle with respect to v_i and v_j.
- It does not intersect the polygon formed with the previously selected points.

3) Once a point that fulfills the last conditions is found, it becomes the new currentPoint.

4) The steps 2 and 3 are repeated until the stopping point is reached.

Algorithm 5 shows implementation details for this procedure.

Figure 4.7 shows the points close to an edge of the convex hull. In this example, the number of nearest neighbors is equal to three.

In order to compute $\mathcal{B}(X)$, the previous procedure is applied on each pair of adjacent vertices. Algorithm 6 implements this task.

Figure 4.8 shows real examples of the convex-concave hull $\mathcal{B}(X)$ computed via Algorithms 5 and 6.

It is possible to obtain a different set $\mathcal{B}(X)$ by changing the parameter K in Algorithm 5. For example, in Figure 4.8 the values of K are: (a) $K = 9$, (b) $K = 4$ and (c) $K = 6$. The greater the value of K, the more similar it is to the convex hull.

Remark 4.1 *Algorithm 5 shows how the space between two consecutive extreme points is explored. The underlying idea is similar to Jarvis' march [116], where a set of points is wrapped. However, Jarvis' march considers all points, whereas Algorithm 5 uses only local neighbors.*

Algorithm 5: Search closest points

```
Input :
        $X$ : A set of points
        $S$: Two adjacent vertices $\{v_i, v_j\}$ of $\mathcal{CH}(X)$
        $K$: Number of neighbors
        $\theta$ : The previous angle between adjacent vertices
Output:
        $Close_{\mathcal{CH}(X)}$ : The set of points close to the edge defined by adjacent vertices
$\{v_i, v_j\}$
begin
    $currentPoint \leftarrow v_i$;
    $stopPoint \leftarrow v_j$;
    $Close_{\mathcal{CH}(X)} \leftarrow v_i$;
    Maximum value of K is the size of set X
    $K \leftarrow max\{|X|, K\}$;
    while $currentPoint \neq v_j$ do
        $candidates$ ←get the K nearest points to $currentPoint$;
        Compute the angle of every $p_i \in candidates$ w.r.t. to angle $\theta$;
        Sort $candidates$ by increasing angle;
        foreach $p_i \in candidates$ do
            $L$ is a segment of line defined by $p_i$ and $currentPoint$;
            if $L$ does not intersect the convex-concave hull then
                $Close_{\mathcal{CH}(X)} \leftarrow Close_{\mathcal{CH}(X)} \cup p_i$;
                exit this loop;
        if no point was added in the last loop then
                $Close_{\mathcal{CH}(X)} \leftarrow Close_{\mathcal{CH}(X)} \cup v_j$;
            return $Close_{\mathcal{CH}(X)}$;
        Remove $currentPoint$ from $X$;
        Update $\theta$ using the two more recently added points of $Close_{\mathcal{CH}(X)}$;
        if $currentPoint = v_j$ then
            return $Close_{\mathcal{CH}(X)}$;
        $currentPoint \leftarrow p_i$;
```

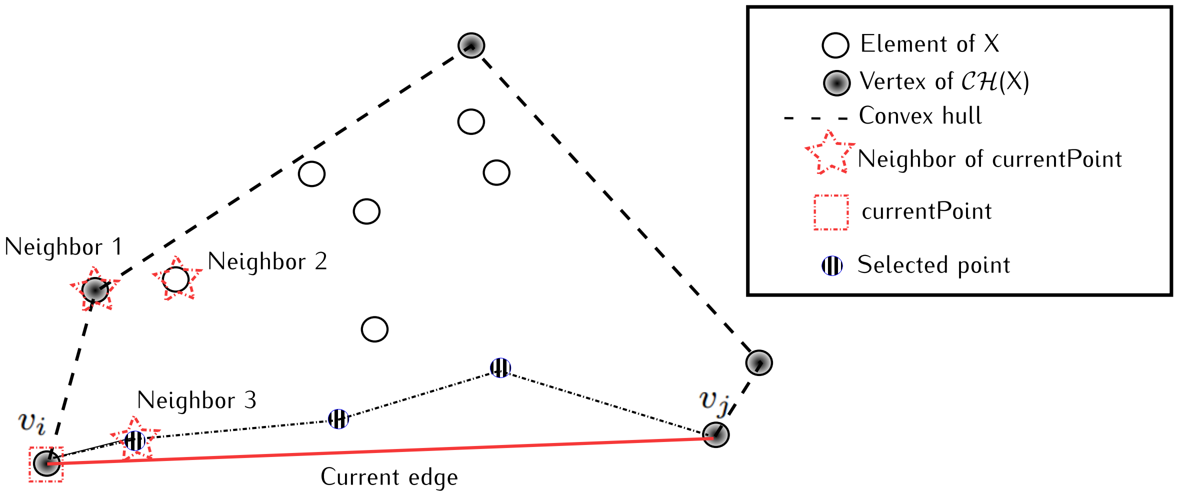


Figure 4.7: **Points close to an edge of $\mathcal{CH}(X)$**

Remark 4.2 *The convex-concave hull has the following advantages over the KNN concave hull [117]: 1) The starting and stopping points of our algorithms are set before, using the convex hull. This ensures that all vertices of the convex hull are always in the vertices of a convex-concave hull. 2) The angles in [117] are entirely depended on the previously computed ones. In our method, the extreme points are used to compute the angles. This allows an easy concurrent implementation. 3) Our algorithm does not use recursive invocations. The stop point is added to the set of the border points when it cannot find more any points. This saves the detection time of $\mathcal{B}(X)$.*

The properties of a convex-concave hull are used to formulate a pre-processing step of data sets for SVMs. Now, we show how to take advantage of these properties.

According to the first property, $\left(\mathcal{B}(X) = V_{\mathcal{CH}(X)} \cup Close_{\mathcal{CH}(X)}\right)$, the set of vertexes of the convex-concave hull $\mathcal{B}(X)$ is a super-set of the vertexes of convex hull $\mathcal{V}_{CH(X)}$, i.e., $V_{\mathcal{CH}(X)} \subseteq \mathcal{B}(X)$.

Algorithm 6 computes $\mathcal{CH}(X)$, and then, it uses two extreme points v_i, v_j to search for the closest points, via Algorthm 5. At the last iteration, $\mathcal{B}(X)$ contains all vertices of $\mathcal{CH}(X)$. According to property 1, if $\mathcal{B}(X)$ is used to train a SVM, then the optimal separating hyperplane is computed. This is true in the case in which the X is linearly separable.

The property 2, ($\mathcal{B}(X) \subset X$), ensures that the number of points selected is at most N, where $N = |X|$. In practice, the size of X is often reduced.

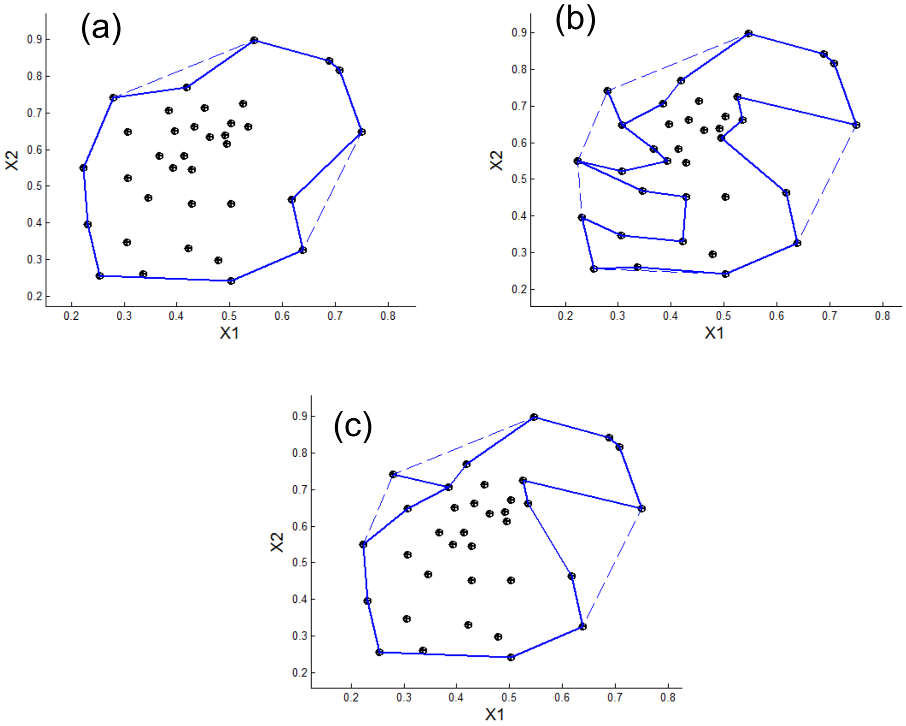


Figure 4.8: **Convex-concave hull computed with different values of K, (a) K = 9, (b) K = 4 and (c) K = 6.**

Finally, property 3 ensures that $\mathcal{B}(X)$ is not an empty set; this is interpreted to mean that SVMs will have data to be trained. A special case occurs when the size of X is less than 3; however, in such case, no data reduction is required.

The parameter K, in Algorithm 5, is quite important. It decides the number of nearest neighbors of currentPoint. If K is chosen to be large enough, for example, $K \geq N$, then the rest of elements in X are nearest neighbors. In this case, the proposed Algorithms converge to a convex hull. This is because for a currentPoint v_i, the point that satisfies the three conditions to be a closest point is the stopping point.

Up to this moment, the properties of a convex-concave hull have been useful only for the linearly separable case. The direct application of convex hulls is not valuable for the general case of classification. The problem with real-world data sets is that the convex hulls usually overlap.

The idea behind RCH and SCH methods is to transform the problems into linearly separable ones. According to the geometry of classification, the patterns that define the

```
Algorithm 6: Convex-Concave Searching Scheme
Input  :
        $X$: Set of points
        $K$: Number of candidates
Output:
        $\mathcal{B}(X)$: Convex-concave hull
begin
    //Set of vertices begins empty
    $\mathcal{B}(X) \leftarrow \varnothing$ Compute $\mathcal{CH}(X)$
    The vertices of $\mathcal{CH}(X)$ are $v_1,...,v_n$;
    for each pair of adjacent vertexes $(v_i, v_j)$ do
        $\theta \leftarrow$ compute angle defined by $(v_i, v_j\ )$;
        Detect points "close" to edge $(v_i, v_j\ )$;
        Apply Algorithm 5 using $(X, (v_i, v_j), K, \theta)$  $\mathcal{B}(X) \leftarrow \mathcal{B}(X) \cup Close_{\mathcal{CH}(X)}$
```

linear decision boundary are the closest points to the opposite (and modified) convex hull. The vertices of convex hulls are enough to compute the separating hyperplane.

Instead of reducing or scaling convex hulls, our data reduction method takes a different approach. First, we create partitions, and then, we compute the convex-concave hulls. The results are joined in a set. This is a super set of $\mathcal{B}(X)$:

$$\bigcup_i \mathcal{B}(X^i) \supseteq \mathcal{B}(X) \supseteq \mathcal{CH}(X)$$

Where

X^i the $i-th$ partition of set X.

$\cap X^i = \varnothing$

$\cup X^i = X$.

We use Figure 4.9 to explain this. A set X has been been partitioned into four subsets. The convex-concave hull of each one of them has been computed. The set of vertices of convex hull $V_{\mathcal{CH}(X)}$ is a subset of $\cup V_{\mathcal{CH}(X^i)}$.

Remark 4.3 *If the Algorithms 5 and 6 are applied on the disjoint partitions of X; the joining of results contain all the vertices of $\mathcal{CH}(X)$. The points in intersection of convex hulls are also included in $\mathcal{B}(X)$. This is helpful for linearly inseparable cases for SVM classification.*

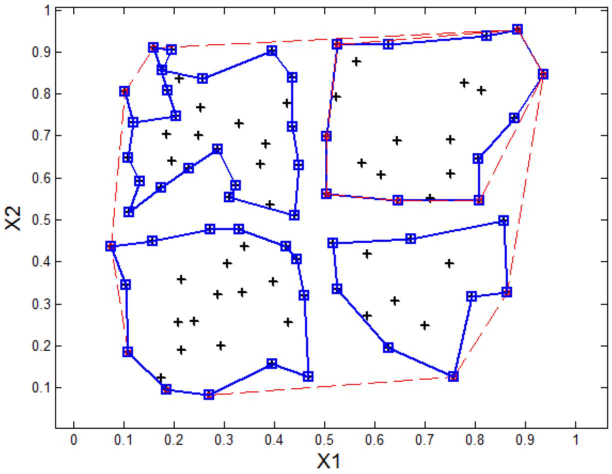


Figure 4.9: **A super set of $\mathcal{B}(X)$ obtained by applying Algorithms 5 and 6 on partitions**

4.3.1 Pre processing

The convex-concave hull algorithms shown in the previous section work for sets with two features. In order to extend our method to more than two dimensions, a dimensionality reduction is necessary. The principal component analysis (PCA) is a common method to reduce the dimension of data sets; however, we do not use PCA. It is costly, and, additionally, the features are linear combinations of features.

We select the two dimensions with the lowest variance, and then, we compute the convex-concave hull. The other features are used to search for more points.

The general approach to apply our algorithms on data sets with higher dimensions is the following:

1. Partition the input space to create a number of partitions X^i.
2. Select two features from X^i to create a set $Y^i \in R^2$. The Y^i can be viewed as X^i with all its features fixed, at exception of two.
3. Apply convex-concave hull searching on Y^i.
4. Explore fixed features for more points.
5. Join the points recovered in the previous steps.

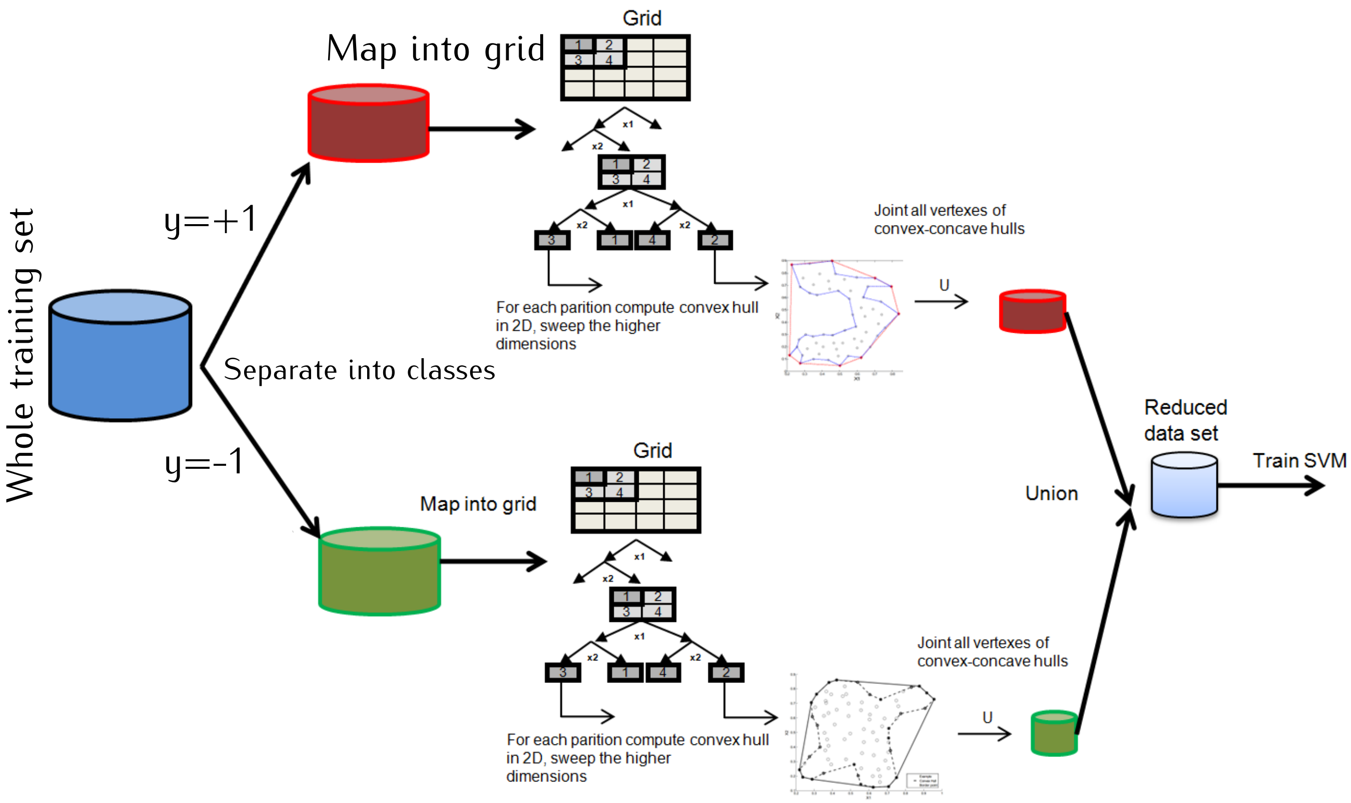

Whole training set
y=+1
y=-1
Separate into classes
Map into grid
Grid
1
2
3
4
x1
x2
For each parition compute convex hull in 2D, sweep the higher dimensions
Joint all vertexes of convex-concave hulls
U
Union
Reduced data set
Train SVM

Figure 4.10: **General process of the convex-concave hull method**

Partition the input space

The partition of input space must be fast to avoid a bottleneck problem. We use a binary tree data structure to manage a grid $\mathcal{G}$. All points in X are mapped into a grid $\mathcal{G}$, i.e., they are inserted into a cell of it. Figure 4.11 shows the structure of the grid. The cells are hyper boxes. Each side such a hyper boxes has a length

$$side_i = \frac{max(d_i) + min(d_i)}{2^{h_g}}$$

Where

h_g the height of the binary tree, or, the granularity of $\mathcal{G}$.

d_i is the $i - th$ feature of the training set.

The size of each cell is controlled by the height of the tree. Figure 4.12 shows how the binary tree manages the grid. The h_g decides how many times one feature should be halved. Each leaf of the tree represents a cell.

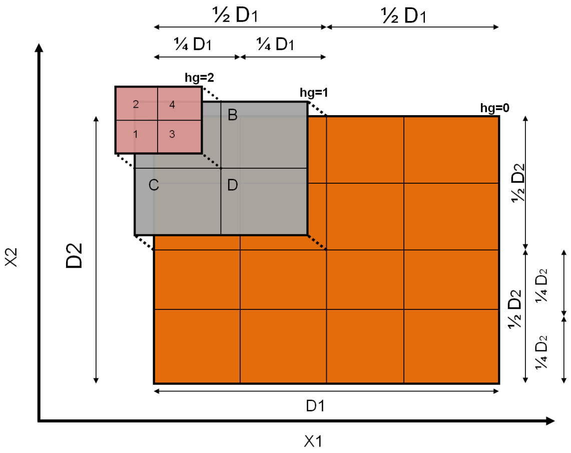


Figure 4.11: **Partition of the input space using a grid**

The mapping from X into grid also reduces the repeated points; if a set of points in X are very close to each other, they are mapped into the same cell.

We use two examples to show how the grid and the convex-concave hull work together. Figure 4.13 shows the result of applying the algorithms on the whole points ($h_g = 0$). Figure 4.14 shows the result with $h_g = 1$. The border point is realized by the union set operation.

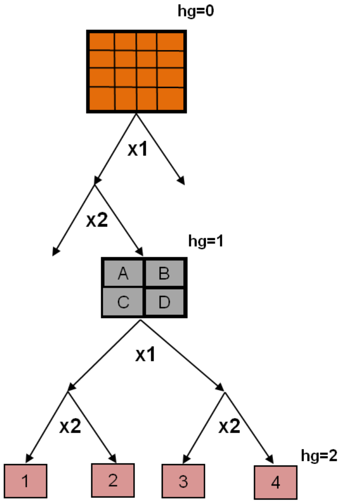


Figure 4.12: **Binary tree represents the grid**

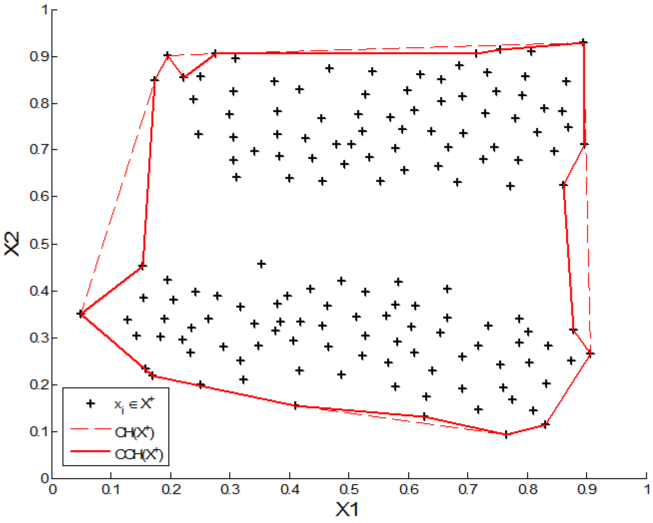


Figure 4.13: **Example with granularity** $h_g = 0$

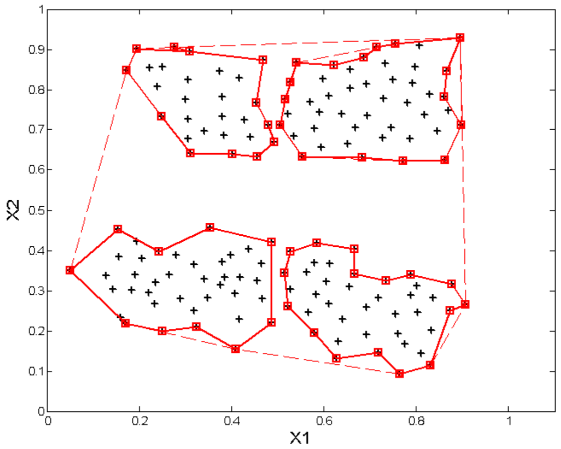


Figure 4.14: **Example with granularity $h_g = 1$**

4.3.2 Searching for convex-concave hull vertices in higher dimensions

The convex-concave hull search utilizes the convex hull as an intermediate step. It is well known that computing the convex hull in more than three dimensions is computationally expensive, and this should be avoided.

We overcome this difficulty by searching the vertices of the convex-concave hull in two dimensions. The other features are used to find more points. After searching in the two selected dimensions, we move in one dimension at a time and apply this process iteratively. The effect of this procedure in three dimensions is similar to slicing a cube. An example of convex-concave hull in three dimensions is shown in Figure 4.15.

In order to cross the dimensions, we use the following strategy: We choose the cells of the grid $\mathcal{G}$ that are at height $h_b < h_g$ in the binary tree, searching from heigh $h_a < h_g$ (see Figure 4.16). Then we reach the next node of the tree and repeat the process. The selection of $h_a < h_b$ can be realized quickly by testing the values $0, 1, 2.., h_g - 1$.

4.4 SVM Classification via Convex-Concave Hull

Figure 4.10 shows the general steps to apply our methods with SVMs. These steps can be summarized as follows:

1. Split the data set into two subsets: positive an negative records.

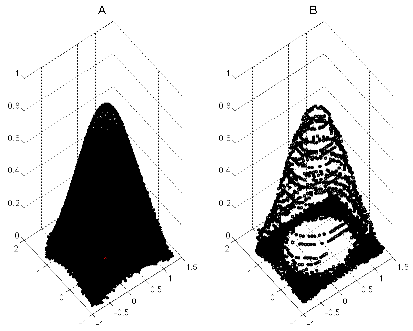


Figure 4.15: **Example of result on a toy example with three dimensions**

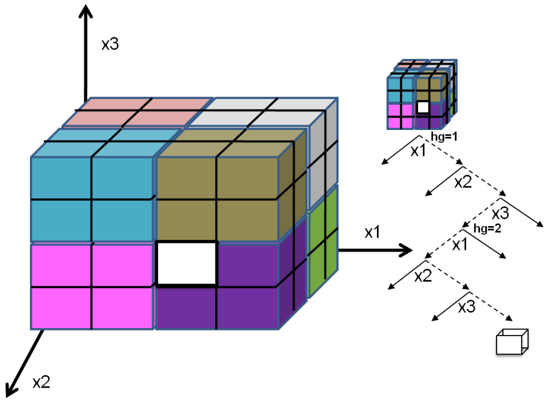


Figure 4.16: **Partition in higher dimensions**

2. Create partitions

3. Compute the convex-concave hulls on partitions.

4. Join the previous results.

5. Train a SVM

Basically, SVM classification can be grouped into two types: linearly separable and linearly inseparable cases. The grid and convex-concave hull algorithms are suitable for the linearly separable and linear inseparable cases. Figure 4.17 shows an example of application of our methods on a linearly separable case.

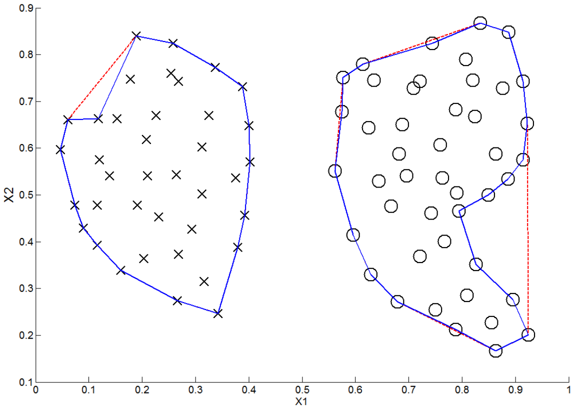


Figure 4.17: **Linearly separable case, $h_g = 0$**

In the linearly inseparable case, the convex hulls $\mathcal{CH}(X)$ are intersected, see Figure 4.18. Because the SVs are generally located on the exterior boundaries of the data distribution, they are not the vertices of $\mathcal{CH}(X^+)$ and $\mathcal{CH}(X^-)$ [118][7]. On the other hand, the vertices of the convex-concave hull are the border points, and they can be used as SVs.

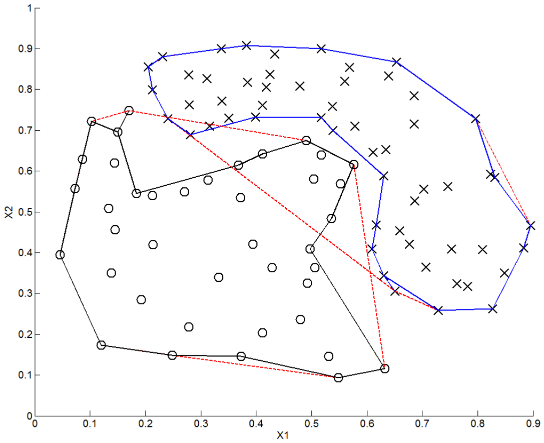


Figure 4.18: **Linearly inseparable case, $h_g = 0$**

The parameters of the kernel are chosen utilizing the grid search method, widely used in the literature. That method tests the performance of a classifier using different parameter values in an interval.

4.5 Performance analysis

In this section, we show the memory space and computation time complexities of our CCH-SVM method.

It is not easy to analyze the exact complexity of the normal SVM algorithm. For n input data, this operation involves a multiplication of matrices of size n, which has complexity $O(n^{2.3})$ and $O(n^{2.83})$ at worst [119]. In the following, we assume that a QP implementation of each stage of SVM takes $O(n^3)$ computation time and $O(n^2)$ memory space.

4.5.1 Memory space

During the pre processing phase of our method, all the examples in the training set X are mapped into a grid $\mathcal{G}$ by inserting them in a binary tree.The height of the tree is h_g (granularity). Considering the following facts:

- Each example has d dimensions and those data types are double (8 bytes),
- The internal nodes in the binary tree occupy each one about 40 bytes (two references to the child nodes and internal variables),

The amount of memory to manage all examples in $\mathcal{G}$ is

$$2^{3+h_g}(d+5) - 40 \text{ bytes}$$

, where h_g is the height of the binary tree, d is the number of features in the data set X. This amount of memory is computed as follows: The maximum number of leaves in a binary tree of height h_g is given by 2^{h_g}. Each leaf contains a vector of d features, and each feature is stored in 8 bytes, the amount of memory used by the leaves is

$$2^{h_g} \cdot 8 \cdot d = 2^{h_g} 2^3 \cdot d$$

. It is well known that the number of internal nodes in a binary tree of height h_g is given by $2^{h_g} - 1$. Because each internal node in the implemented binary tree uses about 40 bytes. The amount of memory used for the internal nodes is

$$40 \cdot (2^{h_g} - 1)$$

The total amount of memory used by a binary tree is composed of the leaves and the internal nodes, i.e.,

$$(2^{h_g}2^3 \cdot d) + 40 \cdot (2^{h_g} - 1)$$

This can be simplified as $2^{hg} \cdot 2^3 \cdot d + 40 \cdot (2^{hg} - 1) = 2^{hg} \cdot 2^3 d + 2^{hg} \cdot 40 - 40 = 2^{hg} \cdot 2^3 \cdot d + 2^{hg} \cdot 2^3 \cdot 5 - 40$. Grouping terms we obtain $2^{h_g} \cdot 2^3(d+5) - 40$ bytes.

This amount of memory is used when all leaves of the binary tree exist. In this case, more than one example is usually mapped into the same leaf of the tree. Reducing the amount of memory originally required to allocate the training set X, *i.e.*, $8 \cdot |X| \cdot d$ bytes. With $|X|$ the size of the training set, namely n.

The memory used by Algorithms 5 and 6 can be ignored, because they use little memory space compared with the binary tree.

4.5.2 Computational time

In order to analyze the computational cost of the proposed method, we separate it in the following phases:

- Create a grid $\mathcal{G}$ via a binary tree. The computational time for creating $\mathcal{G}$ is obtained with the insertion of examples in a binary tree. It is $O(n \log_2 n)$ with $n = |X|$ examples.

- Detect the vertexes of the convex-concave hulls in partitions Y_i by looking downward the tree from height h_a down to height h_b.

  Algorithm 6 uses the partition Y_i to obtain the vertexes of a convex hull. The cost is $O(|Y_i| \log_2 |Y_i|)$.

  To simplify the analysis we consider a uniform distribution of examples, in this case the size of Y_i can be approximated as $\frac{|X|}{2^{h_b}}$.

  For Algorithms 5 and 6, the worst case occurs if all examples of Y_i are considered as vertexes; this happens when Y_i has few elements. The computational time is small compared with these few samples. In the general case, Algorithm 5 searches a small subset from Y_i by the K-nearest neighbors method (KNN). The computational time of the simple KNN is $O(d|X|^2)$. For our case, it becomes $O\left(d\left(\frac{|X|^2}{2^{2\cdot(h_b)}}\right)\right)$.

  Because we search at each node that is located at heigh h_a, this becomes a bottleneck of our method when dealing with higher dimensions. This is the major disadvantage of our method; it is scalable with the size of the training set but not with the number of features. We have observed that our method is unsuitable for more than four dimensions.

The total computational time on the pre-processing steps is

$$O(|X|\log_2|X|) + O(\frac{|X|}{2^{h_b}}\log_2\frac{|X|}{2^{h_b}}) + O\left(d\left(\frac{|X|^2}{2^{2\cdot(h_b)}}\right)\right) \tag{4.17}$$

- Train a SVM using a QPP solver. We assume that a QP implementation of a SVM takes $O(n^3)$ time and $O(n^2)$ space for $|X|$ input data. On the other hand, the number of vertexes of convex-concave hull is greater than the vertexes of convex hull $|\mathcal{B}(X)| > |\mathcal{V}_{\mathcal{CH}(X)}|$. However, $|\mathcal{B}(X)| \ll |X|$, because not all points are on the boundary. The time to train an SVM using $|\mathcal{B}(X)|$ is lower than the time used to train a SVM with the whole training set X, i.e., $O(|\mathcal{B}(X)|^3)$

4.6 Results

We used eight data sets to compare our algorithms with other methods. Four data sets are publicly available in the UCI repository; one was modified, and four more were created to observe the performance of each method.

The data sets from the UCI repository are: Four classes, Skin-no-skin, Breast cancer and Haberman's survival. The data set modified was Checkerboard (Figure 4.19), and the synthetic data sets are Cross (figure 4.20) Rotated-cross (Figure 4.21) and Balls aDb (Figure 4.22). All of them all are linearly inseparable. Table 4.1 shows a summary of the data sets used in the experiments. The synthetic data set Balls aDb has "a" features and size "b"×1000.

Our algorithms were compared with the SMO [1] [12], library LIBSVM [2]. [80] and the reduced convex hull SVM (RCHSVM) [8]. The SMO and LIBSVM are trained with the original data set. The RCHSVM finds the closest points within a reduced convex hull. Our convex-concave hull SVM (CCHSVM) uses the border points detected by the proposed algorithms.

All experiments were run on a computer with the following features: Core i7 2.2 GHz processor, 8.0 GB RAM and Windows 7 operating system. The algorithms are implemented in the Java language. The maximum amount of random access memory given to the Java virtual machine is set to 2.0 GB. The reported results correspond to 100 runs of each experiment.

[1] implementation http://wiki.pentaho.com/display/DATAMINING/SMO

[2] http://www.cs.iastate.edu/ yasser/wlsvm.html

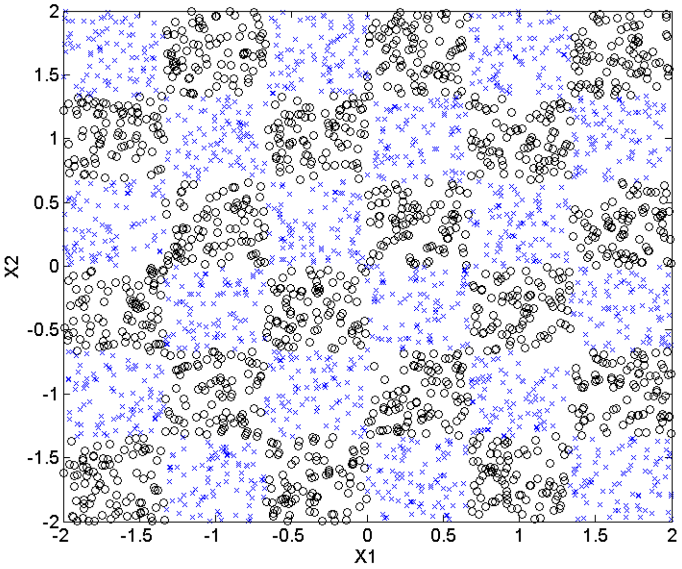


Figure 4.19: **Checkerboard data set**

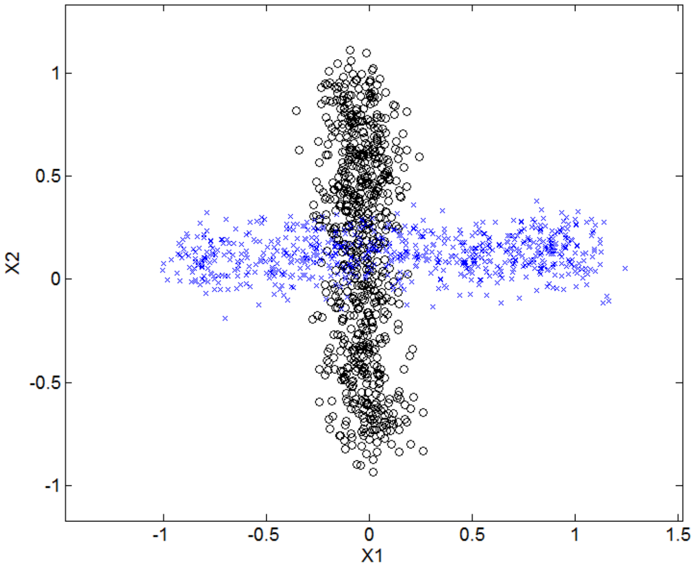


Figure 4.20: **Cross artificial data set**

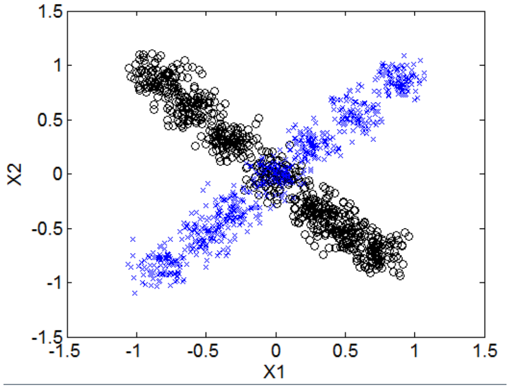


Figure 4.21: **Rotated-cross artificial data set**

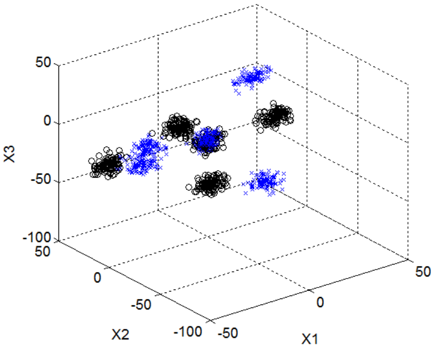


Figure 4.22: **Balls artificial data set**

Table 4.1: **Data sets used in the experiments for the convex-concave hull method**

Data set	Features	Size	$y_i = +1$	$y_y = -1$	classes
Four-class	2	862	307	555	2
Checkerboard50	2	50,000	13,000	12,000	2
Cross	2	90,000,	50,000,	40,000	2
Rotated-cross	2	90,000,	50,000,	40,000	2
Skin-no-Skin	3	245,057	50,859	194,198	2
Haberman's Survival	3	306	225	81	2
Balls aDb	a	b	b/2	b/2	2
Breast cancer	9	286	201	85	2

For each experiment, the training data are chosen randomly from 70% of the data set; the rest data was used for testing.

The kernel used in all experiments is a radial basis function. The RBF kernel is chosen as:

$$f(x, z) = \exp\left(-\frac{(x-z)^T (x-z)}{2\gamma^2}\right) \tag{4.18}$$

where γ was selected using the grid search method.

4.6.1 Experiment 1: Size of the training set

In this experiment, we use the checkerboard data set [120] which is commonly used for evaluating SVM implementations.

Although this data set can be reduced applying a random selection of examples [50], the checkerboard data set is useful for illustrating the scaling properties of our algorithms.

We first examine how the training data size affects the training time and the classification accuracy of our convex-concave hull SVM (CCHSVM). We use 500; $1,000$; $2,000$; $5,000$; $10,000$; $50,000$ and $100,00,000$ examples to train CCHSVM and LibSVM. The comparison results of 100 runs of experiments are shown in Table 4.2.

In Table 4.2, the columns *Avg Acc* and *Avg training* are the averages of the classification accuracy and the training times respectively. The column *stdDev* is the standard deviation of the accuracy. For experiment 1, the values $\gamma = 0.9$ and $C = 1.0$ were chosen using the grid search method.

Table 4.2: **Classification results for the data set Checkerboard**

Method	Data set size ×1000	Avg training time (ms)	Avg Acc %	stdDev
LibSVM	0.5	16.66	85.04	0.27
CCHSVM	0.5	28.13	85.32	0.19
LibSVM	1	46.36	91.34	0.27
CCHSVM	1	61.86	91.01	0.19
LibSVM	2	108.10	98.31	0.27
CCHSVM	2	156.46	94.90	0.19
LibSVM	5	362.00	97.12	0.12
CCHSVM	5	336.80	96.35	0.15
LibSVM	10	1023.40	98.27	0.33
CCHSVM	10	792.80	97.15	0.27
LibSVM	50	15,692.93	99.28	0.07
CCHSVM	50	5,185.00	98.60	0.27
LibSVM	100	47,670.80	99.58	0.05
CCHSVM	100	13,434.81	98.32	0.15

The parameters of the CCHSVM algorithm were selected using a grid method. The values for this experiment were $K = 50$, $h_g = 9$, $h_a = 4$ and $h_b = 8$.

We can see that our CCHSVM has less training time than LibSVM when the size of the training set is large. For data sets with few hundreds of examples, the LibSVM outperforms our method. The classification accuracy is slightly lower than that obtained with LibSVM.

The pre-processing time reduces the size of the training set; this improves the training time of SVMs. The size of the training set is important when the set contains thousands of examples. In other cases, the pre-processing step is not valuable.

When the data size is increased, the training time is augmented with LibSVM, while with our method, it only increases a little. Although the classification accuracy cannot be significantly improved when data size is very large, it does not get worse, and the testing accuracy is still acceptable.

Figure 4.23 shows performance of CCHSVM algorithm. It can be seen that with more than $10,000$ examples, our method is significantly better than LibSVM.

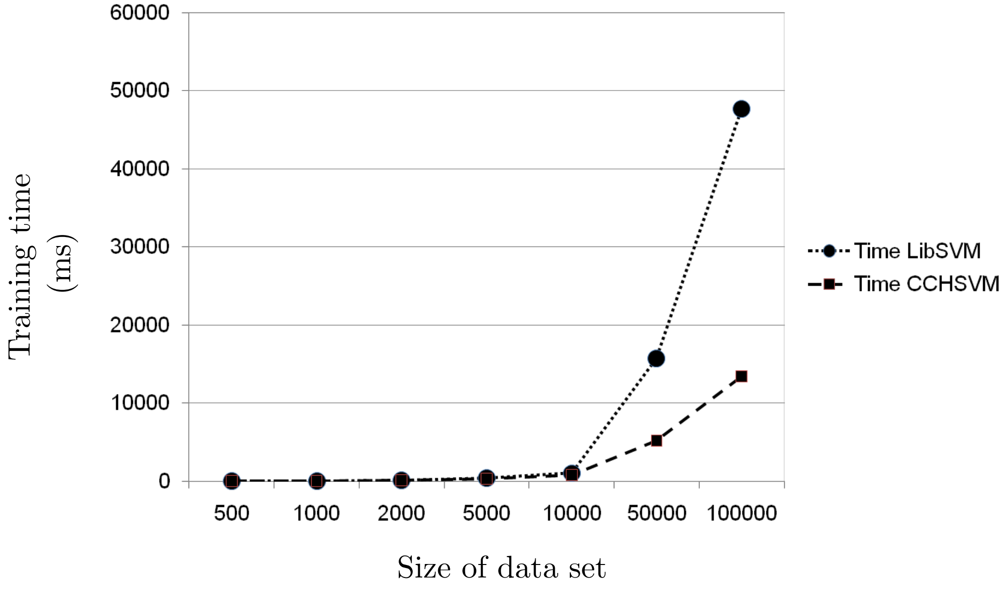


Figure 4.23: **Performance of CCHSVM with respect to size of the training set**

4.6.2 Experiment 2: Parameters

The CCHSVM method has four parameters: h_g, h_a, h_b and K. The h_g controls the granularity of the grid, i.e., the height of the binary tree in which the examples of the training set are mapped into. The parameter h_a is used to create the partitions in higher dimensions. The binary tree is traversed down to heigh h_b. The parameter K controls the number of neighbors to detect border points close to edges of the convex hull.

We used the synthetic Balls3D40 data set ($40,000$ examples) to analyze the effect of the parameters of the algorithm. The LibSVM method was used as a reference. Table 4.3 shows the results. For experiment 1, the values $\gamma = 1.9$ and $C = 2.5$ were chosen using the grid method.

The most important effect of parameter K is on the size of the reduced set. Its effect on the classification accuracy and running time seems not too important.

The parameters h_g and h_a have an impact in the training time and accuracy. The higher the value of h_g the deeper the binary tree and also the greater the number of cells in the grid.

Table 4.3: **Effect of parameters of the convex-concave hull method**

Method	h_a	h_b	h_g	K	Avg size reduced	Avg training time (ms)	Avg Acc	stdDev %
LibSVM	–	–	–	–	–	1,164,500.00	99.69	0.13
CCHSVM	4	7	12	50	9,277.04	23,443.00	99.83	0.24
CCHSVM	4	7	12	5	9,992.02	23,790.00	99.83	0.20
CCHSVM	4	7	8	50	9,244.01	19,945.00	99.75	0.26
CCHSVM	5	9	12	5	17,585.01	90,130.00	99.92	0.19
CCHSVM	5	9	12	500	8,460.05	20,130.00	99.71	0.22
CCHSVM	4	6	6	100	7,760.80	9,898.20	99.73	0.24
CCHSVM	4	7	8	100	7,920.10	13,260.00	99.76	0.16
CCHSVM	4	5	7	100	4,245.12	5,815.00	92.35	0.19
CCHSVM	4	5	8	100	4,287.33	8,656.60	91.97	0.20
CCHSVM	3	6	8	100	4,500.40	3,079.40	94.55	0.19
CCHSVM	3	6	8	300	4,395.40	3,296.81	92.91	0.53

The value of h_a affects the number of partitions. A larger value of this parameter involves a major number of searches for boundary points. The accuracy is degraded when h_a is low because many support vectors are omitted.

4.6.3 Experiment 3: Comparative with other methods

In this experiment, we compare our method with the SMO, LibSVM and RCHSVM. We use four benchmark data sets and three synthetic ones. The Balls3D100 contains $100,000$ examples, and it has three features. Both the Cross and the Rotated-cross data sets have two features and $100,000$ examples.

For experiment 3, the parameters were selected using the grid search method. The values for the first data set are LibSVM $\gamma = 0.68$ and $C = 1.0$, SMO $\gamma = 0.68$ and $C = 1.0$, RCHSVM $\gamma = 0.1$ and $C = 0.5$, CCHSVM $\gamma = 0.9$ and $C = 3.0$. The values that produced the better results in this experiment were $K = 5$, $h_g = 10$ and $h_a = 5$ and $h_b = 9$ for all data sets except for Skin-NoSkin, in this case the values used were $K = 7$, $h_g = 12$ and $h_a = 9$.

The training of SVMs with data sets that contain tens of thousands or more examples, is generally expensive. The results obtained in this experiment are shown in Table 4.4. The RCHSVM algorithm does not scale well with the size of the training set. The LibSVM and SMO produce the best accuracies and standard deviations in practically all cases, but their training times are very high. Our CCHSVM method can train SVM very fast; the accuracy and standard deviation are slightly degraded but yet acceptable. For small data sets, our method is not the best in terms of the training time.

4.7 Conclusions

This Chapter proposes a novel method for SVM classification, the CCHSVM. By using convex-concave hull and a grid method to avoid costly computations in higher dimensions, the CCHSVM overcomes the problems of slow training times of an SVM and the low accuracy of many geometric properties based methods.

The key point of our method is the detection of vertices of a convex-concave hull, which corresponds to the examples located on the boundaries of the data set. Experimental results demonstrate that our approach has good classification accuracy, while the training time is significantly faster than other SVM classifiers. The classification accuracy is higher than that of other geometric methods such as reduced convex hull.

The training time of SVM has been significantly reduced. Our method is unsuitable for small data sets. The accuracy achieved by CCHSVM is maintained slightly lower than that of the classical SVM classifiers which use the whole data set, and the training time is not reduced.

However, in the case of larger data sets, the classification accuracies are almost the same than that of other SVM methods; in contrast, the training time is greatly improved.

Table 4.4: **Comparison with other methods**

Method	**Data set**	**Training time (ms)**	**Acc %**	**stdDev**
SMO	Four-class	17.00	95.33	0.20
LibSVM	Four-class	6.10	99.01	0.0
RCHSVM	Four-class	27.00	98.08	0.12
CCHSVM	Four-class	22.83	98.55	0.14
SMO	Checkerboard50	27,740.00	98.08	0.19
LibSVM	Checkerboard50	19,001.00	98.2	0.01
RCHSVM	Checkerboard50	48,225.00	92.9	.24
CCHSVM	Checkerboard50	3,791.01	96.82	0.68
SMO	Cross	27,740.00	98.08	0.19
LibSVM	Cross	19,001.00	98.2	0.01
RCHSVM	Cross	program crashes	program crashes	program crashes
CCHSVM	Cross	5,234.20	95.82	0.68
SMO	Rotated-Cross	27,740.00	98.08	0.19
LibSVM	Rotated-Cross	19,001.00	98.2	0.01
RCHSVM	Rotated-Cross	program crashes	program crashes	program crashes
CCHSVM	Rotated-Cross	4,986.20	95.82	0.68
SMO	Skin-NoSkin	3,100,254.00	96.08	0.32
LibSVM	Skin-NoSkin	1,860,014.00	96.34	0.09
RCHSVM	Skin-NoSkin	program crashes	program crashes	program crashes
CCHSVM	Skin-NoSkin	12,780.00	94.72	0.23
SMO	Haberman's survival	9.46	72.31	0.02
LibSVM	Haberman's survival	9.30	72.25	0.12
RCHSVM	Haberman's survival	12.30	72.10	0.13
CCHSVM	Haberman's survival	11.45	73.94	0.10
SMO	Balls3D100	2,100,013.25	95.26	0.02
LibSVM	Balls3D100	2,000,663.02	96.10	0.10
RCHSVM	Balls3D100	program crashes	program crashes	program crashes
CCHSVM	Balls3D100	15,260.50	97.66	0.21
SMO	Breast cancer	95.00	96.78	0.01
LibSVM	Breast cancer	80.01	96.93	0.13
RCHSVM	Breast cancer	128.25	91.35	0.23
CCHSVM	Breast cancer	1,026.66	95.13	0.21

5 Data reduction with decision tree and Fisher's linear discriminant

Out of one thousand who seek me...one is mine

Jesus of Nazareth

A novel data reduction algorithm, called DTFSVM (Decision Tree and Fisher's linear discriminant for SVMs), is presented in this chapter. DTFSVM discovers low entropy regions in the input space, and analyzes them to detect objects close to others with an opposite label. Our method has advantages over simple random sampling and also over distance based methods. First, DTFSVM does not use any kind of unplanned selection, which allows repeatable results; second, it does not compute all distances between objects, avoiding the bottleneck of naive algorithms. We also introduce a minor variant of the DTFSVM; this variant uses a directed random selection, which improves even more the training time at the expense of degrading the classification accuracy of SVMs classifiers. We tested DTFSVM and its variant on seventeen publicly available data sets. The results are very competitive compared with respect to LibSVM and the SMO. The training of the SVM is reduced in several orders; the classification accuracy is almost preserved, and its standard deviation is quite low.

5.1 Decision trees and SVMs

Among the currently classification methods, SVMs produce a high accuracy, a have a compact model and extraordinary generalization capability. On the other hand, decision trees are classifiers that produce models comprehensible by human experts. In general, the algorithms to induce decision trees are not costly. Decision trees are used in this thesis to improve the training time of SVMs.

Data reduction methods based on decision trees have been proposed before. They allow the use of SVMs on large data sets. In [121] each partition discovered by a decision tree is used to train a SVM. That method uses the fact that the partitions are less "complicated" than the entire training set. A SVM is applied to each partition to build the classifier. In [16] the number of instances is reduced to train a SVM; the underlying idea is to approximate the decision boundary of SVMs, by capturing the objects near to it, using a decision tree.

Decision trees have been combined with SVMs to facilitate their application on multi-class problems. The method in [122], selects two classes at every node of a decision tree. Then, it employs the probabilistic outputs, to measure the similarity between the remaining samples. In [123], multi-class problems are simplified. They are converted into a number of two-class classification problems. The C4.5 [124] algorithm is used as a tool to generate two subsets; all the classes that are less separated by the margin are joined, and treated as one class; the class with the largest margin is considered as the opposite class. Two drawbacks in the methods shown in [123] are: (a) the margins need to be computed at each stage of a decision tree, which is costly; and (b) the method exhibits a poor performance when the number of classes is small. In [39], these drawbacks are solved by inserting information about margin measures in a list data structure. Another algorithm that faces multi-class problems was presented in [125]. It creates a tree using an SVM to split the data. The main disadvantage of this method, is that for each internal node, a separating hyperplane is computed. This strategy has the negative effect that the right, and the left sub trees are imbalanced. It is known that SVMs don't achieve a high classification accuracy on skewed data sets.

5.2 Detecting regions with support vectors

The optimal separating hyperplane of the SVMs classifiers is defined by a few examples known as the SV. In general, these objects are located close to others with an opposite class. These examples are near to the boundaries of class distribution [126] [127] [64]. The SV candidates could be selected using a brute force approach; by computing all the distances between objects and choosing the pairs of examples with the shorter distances. However, a problem with this trivial method is that it takes about $O(N^2)$ in time and space, as shown in Chapter 3. This naive approach becomes computationally expensive and, therefore, is unsuitable even for medium-size data sets.

DTFSVM avoids the bottleneck of the brute force approach, by using a different strategy. Instead of computing all the distances between objects, DTFSVM discovers low entropy regions. The majority class of each region is the most repeated class in it. DTFSVM determines all the adjacent regions whose majority class is different from the current region being analyzed. Having two regions with opposite majority class, DTFSVM applies a search-method that selects the examples with the shortest distances.

It is well known that decision trees create partitions of the input space. According to such approach, these partitions have generally a low entropy value. The partitions are discovered in $O(N \cdot d)$ time, with N being the number of examples and d the number of dimensions.

5.2.1 Computing adjacent regions

In this subsection, a method to detect adjacent regions in the input space of a given data set is presented. In order to explain the approach, this subsection is divided into two parts. The first part explains the representation of partitions discovered by decision trees (leaves), and exposes some definitions that are used later. The second subsection describes how to detect adjacent regions.

Leaves of decision trees

Decision trees separate the input space of data sets into low entropy partitions. These are represented as leaves (terminal nodes) in the tree's structure. The boundaries of leaves are constrained by hyperplanes that are parallel to the axes. The partitions can be interpreted as disjoint ortho topes or hyperboxes. Their vertices are orthogonal to the axes of the attributes.

Let $\mathcal{T}$ represent a decision tree and let $\mathcal{L}_i$ its i-th leaf. In this work, a leaf $\mathcal{L}_i$ is represented by

$$\mathcal{L}_i = \bigcap_{j=1}^{d} r_{ij}, \;\; l_{ij} \leq r_{ij} < h_{ij} \tag{5.1}$$

where:

$\mathcal{L}_i$: the $i-th$ leaf in the decision tree $\mathcal{T}$.

d: the number of dimensions of the training set.

$r_{i,j}$: a region of input space determined by boundaries $[l_{i,j}, h_{i,j})$ with $l_{i,j}, h_{i,j} \in R$. These $l_{i,j}, h_{i,j}$ are the cutting points found by an induction tree algorithm.

Figure 5.1 shows an example of a leaf, and the contained region in two dimensions. In this case, the region is a rectangle. In three dimensions, the region would be a rectangular prism, and in higher dimensions, it would be a hyperbox.

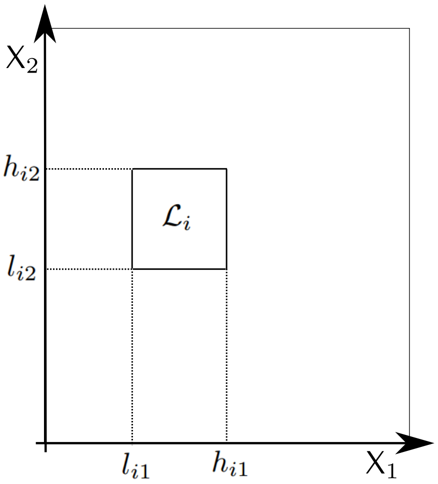


Figure 5.1: **A leaf $\mathcal{L}_i$ in two dimensions**

A candidate to be neighbor of a leaf or region $\mathcal{L}_p$ of $\mathcal{T}$, is another leaf $\mathcal{L}_o$ of $\mathcal{T}$, that is located *together* to $\mathcal{L}_p$. Definition 15 formalizes this notion.

Definition 15 (Neighbor candidates) *Two leaves $\mathcal{L}_o$ and $\mathcal{L}_p$, are candidates to be neighbors if their boundaries have been changed from eq. (5.1) to eq. (5.2).*

$$L_o = \bigcap_{j=1}^{d} r_{o,j}, \;\; l_{o,j} \leq r_{o,j} \leq h_{o,j} \tag{5.2}$$

$$L_p = \bigcap_{j=1}^{d} r_{p,j}, \;\; l_{p,j} \leq r_{p,j} \leq h_{pj}$$

Remark 15.1 Most induction tree algorithms split the input space using a rule of the form $x_i < C$ for one partition and $x_i \geq C$ for the other one. In the Definition 15, the boundaries of leaves $\mathcal{L}_o$ and $\mathcal{L}_p$ have changed to give them a chance to be connected.

Two leaves are together (they are true neighbors) if they share at least a split point. Any two points of two true neighbors can be joined by a path.

Definition 16 (True neighbors) *Two leaves $\mathcal{L}_o, \mathcal{L}_p$ which are candidates to be neighbors, become* ***true neighbors*** *if $L_o \cup L_p$ is a connected space.*

Remark 16.1 Leaves $\mathcal{L}_o$ and $\mathcal{L}_p$ are neighbors if there exists an m with $1 \leq m \leq d$, and the following are satisfied:

$$h_{o,m} = l_{p,m} \text{ or } l_{o,m} = h_{p,m} \tag{5.3}$$

$$l_{p,n} \leq l_{o,n} \leq h_{p,n} \text{ or } l_{pn} \leq h_{o,n} \leq h_{p,n} \tag{5.4}$$

with $1 \leq n \leq d$ with $n \neq m$

Explanation

If two leaves $\mathcal{L}_o$ and $\mathcal{L}_p$ are neighbors, then there exists at least one point x_n that fulfills the following.

$$x_n \in \{r_{o,k} \cap r_{p,k}\},\ for\ k = 1, ..., d \tag{5.5}$$

This means that leaves $\mathcal{L}_o$ and $\mathcal{L}_p$ must necessarily share a boundary, which is orthogonal to an axis of features, for example m. The possible options are that the lower(upper) bound of $\mathcal{L}_p$ coincides with the upper(lower) bound of $\mathcal{L}_o$, as stated in eq. (5.3).

Remark16.2 Eq. (5.3) is a necessary but insufficient condition for two leaves to be true neighbors; they need to share all their boundaries at the other dimensions. In order to be true neighbors, eq. (5.5) and eq. (5.4) must both be fulfilled.

Figure 5.2 shows an example in which the condition Eq. (5.3) is fulfilled, but Eq. (5.4) is not. Here, $h_{2,1} = l_{3,1}$ (the separating hyperplane orthogonal to dimension one is shared by leaves $\mathcal{L}_2$ and $\mathcal{L}_3$) but $\mathcal{L}_2$ and $\mathcal{L}_3$ are not neighbors. Something similar occurs with the pairs ($\mathcal{L}_2$,$\mathcal{L}_4$), ($\mathcal{L}_2$,$\mathcal{L}_7$), ($\mathcal{L}_1$,$\mathcal{L}_7$), ($\mathcal{L}_3$,$\mathcal{L}_8$), ($\mathcal{L}_4$,$\mathcal{L}_5$), ($\mathcal{L}_4$,$\mathcal{L}_7$), ($\mathcal{L}_4$,$\mathcal{L}_8$), ($\mathcal{L}_5$,$\mathcal{L}_6$), etc.

In order to implement our method, it is necessary to discover true neighbors of a leaf. In the next subsection, we explain an algorithm to compute them.

A method to find all the neighbors of a leaf

In this part, we propose a method to compute the neighbors of a leaf. The method was designed for the $C4.5$ decision tree; however, it can be extended to other trees.

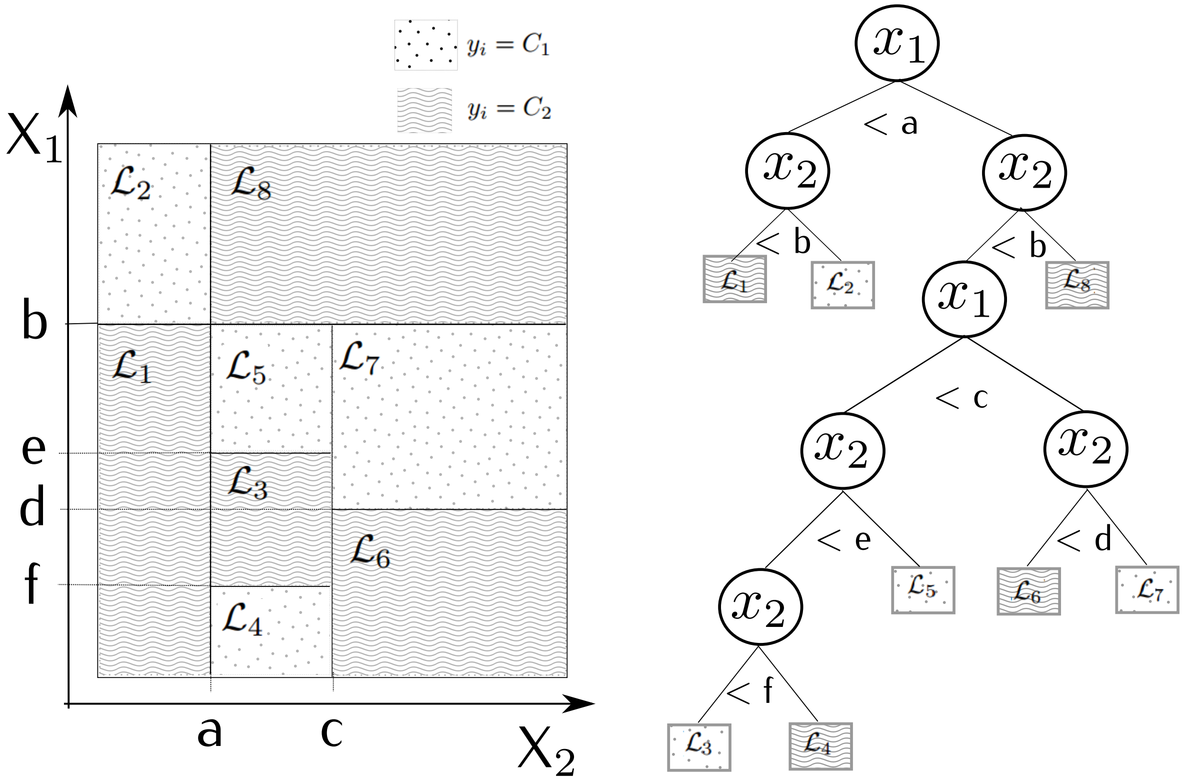


Figure 5.2: **Example of boundaries produced by an induction tree**

At the beginning, the tree is traversed to determine the boundaries of each leaf; all boundaries are stored in a matrix $M \in R^{N_L \times 2d}$, with N_L the number of leaves in the induced decision tree. The matrix M can be filled in $O(N_L h_t)$ time, with h_t the height of the tree. During the traversing of the decision tree, the split points in each internal node are used to discover the boundaries of partitions.

The $i-th$ row in the matrix M, contains all the boundaries of leaf $\mathcal{L}_i$. The columns of the matrix M represent the values l_{ij} and h_{ij}, with $j = 1, .., d$. These values are implemented in a vector B, which is initialized with the minimum and maximum value of float point variables at root node. These values are represented as $-\infty$ and $+\infty$. Algorithm 7 shows the pseudo code to fill out the matrix M.

To exemplify the results produced with Algorithm 7, Table 5.1 shows the boundaries of the leaves for the tree shown in Figure 5.2.

The neighbors of any leaf $\mathcal{L}_p$ can be quickly computed, using Algorithm 8. The set of

Table 5.1: **Matrix M computed with Algorithm 7 for the tree of Figure 5.2**

$\mathcal{L}_i$	$l_{i,1}$	$h_{i,1}$	$l_{i,2}$	$h_{i,2}$
1	$-\infty$	a	$-\infty$	b
2	$-\infty$	a	b	$+\infty$
3	a	c	f	e
4	a	c	$-\infty$	f
5	a	c	e	b
6	c	$+\infty$	$-\infty$	d
7	c	$+\infty$	d	b
8	a	$+\infty$	b	$+\infty$

Algorithm 7: Computation of leaves' boundaries

```
Input :
        $\mathcal{T}$: An induced C4.5 decision tree
Output:
        $M$: A matrix with the boundaries of all the leaves of T.
begin
    $B \in R^{2d}$, $B[1\ to\ d] \leftarrow \{[-\infty]\}$ , $B[d+1\ to\ 2d] \leftarrow \{[+\infty]\}$ ;
    call $DiscoverBoundariesRecursive(B, M)$ from root node;
    return $M$;
DiscoverBoundariesRecursive($B$,$M$)
if current node is a leaf then
    Insert vector $B$ as last row into matrix $M$;
else
    Create $B_L$ copying boundaries from B;
    $B_L$: Change $h_{ij}$ using attribute index and the split point value of current node;
    call DiscoverBoundariesRecursive($B_L$, M) on the left son of current node;
    Create $B_R$ copying boundaries from B;
    $B_R$: Change $l_{ij}$ using attribute index and the split point value of current node;
    call DiscoverBoundariesRecursive($B_R$, M) on the right son of current node;
return
```

Algorithm 8: Neighbors of a leaf

```
Input  :
        $M$ matrix of leave's boundaries
        $\mathcal{L}_p$ A leaf
Output:
        $\mathcal{N}$ The neighbors of leaf $\mathcal{L}_p$
begin
    foreach attribute $m$ in data set do
        Get all leaves that satisfy (5.3);
        foreach leaf $\mathcal{L}_o$ obtained in the previous step do
            if $\mathcal{L}_o$ satisfies (5.4) then
                Add $\mathcal{L}_o$ to list $\mathcal{N}$;
    Remove repeated elements in $\mathcal{N}$;
    return $\mathcal{N}$;
```

leaves that satisfy eq. (5.3) for attribute m are computed by searching in the matrix M. This set is explored again to find those leaves that are neighbors, namely, those that satisfy Eq. (5.4).

5.2.2 Detecting objects on boundaries

We recover the instances that are close to others with an opposite label using the detected neighbors of each leaf. We apply Fisher's linear discriminant to each pair of neighbors. Then use of Fisher's linear discriminant is based on the observation that for linearly separable cases, the linear discriminant produces similar results than SVMs [128]. The idea is to approximate two adjacent regions with opposite class as a linearly separable case. Algorithm 9 shows the implementation: Once two adjacent regions have been detected, all the points in $\mathcal{L}_j$ are projected on vector ω (eq. (2.70)), a number of $\delta \times |\mathcal{L}_j|$ examples with smaller projections are added to the reduced set X_R. Here, $|\mathcal{L}_j|$ refers to the number of examples in the leaf $\mathcal{L}_j$ of $\mathcal{T}$. δ is a parameter given by the user.

It is important to notice that high entropy regions will contain support vectors, and they do not need to be analyzed with Fisher's linear discriminant, and are included directly in X_R.

To exemplify the method, consider the toy example shown in Figure 5.3. After the training of a decision tree, the input space has been partitioned into six regions $\mathcal{L}_1$ to $\mathcal{L}_6$. Each region

Algorithm 9: General Algorithm using Decision Tree and FLD

```
Input :
        $X$: Training set
        $\delta$: Threshold
Output:
        $X_R$: $X_R \subset X$ s.t. $|X_R| \ll |X|$
begin
    Train a decision tree $\mathcal{T}$;
    $X_R \leftarrow NULL$ // $X_R$ Begins empty;
    foreach leaf $\mathcal{L}_i$ of $\mathcal{T}$ do
        foreach opposite class neighbor $\mathcal{L}_j$ do
            if entropy of $\mathcal{L}_j$ is low then
                //Select SVs candidates:
                Use $\mathcal{L}_i$ and $\mathcal{L}_j$ to build $X^+$ and $X^-$, respectively;
                Compute $\omega$ (eq. (2.70));
                Project every example in $X^-$ on $\omega$;
                Select the most separate (w.r.t. projections) pair of objects $x_i^+ \in X^+$ and
                $x_j^- \in X^-$;
                Compute distance between $x_j^+$ and every element in $X^-$, sort examples
                w.r.t. distance;
                Select the first $\sigma$% of objects in $\mathcal{L}_j$ and join them to $X_R$;
            else
                $X_R \leftarrow X_R \cup \mathcal{L}_j$// Add all the elements;
    return $X_R$
```

Table 5.2: **Partition of input space and adjacent regions**

Region	Majority class	Adjacent regions
$\mathcal{L}_1$	+1	$\mathcal{L}_2$, $\mathcal{L}_3$, $\mathcal{L}_4$
$\mathcal{L}_2$	mixed	$\mathcal{L}_1$, $\mathcal{L}_4$,$\mathcal{L}_5$
$\mathcal{L}_3$	+1	$\mathcal{L}_1$, $\mathcal{L}_4$
$\mathcal{L}_4$	-1	$\mathcal{L}_1$, $\mathcal{L}_2$, $\mathcal{L}_3$,$\mathcal{L}_5$, $\mathcal{L}_6$
$\mathcal{L}_5$	-1	$\mathcal{L}_2$, $\mathcal{L}_4$, $\mathcal{L}_6$
$\mathcal{L}_6$	+1	$\mathcal{L}_4$, $\mathcal{L}_5$

$\mathcal{L}$ is associated with a majority class $y = \{C_1, C_2\}$, with $C_1 = +1$ and $C_2 = -1$ for this example. Table 5.2 shows the discovered partitions, the majority class of each one and the adjacent regions.

For certain region $\mathcal{L}_i$, our method selects the examples that are located "close" to adjacent regions $\mathcal{L}_j$ that contain different class examples. For example, for region $\mathcal{L}_4$ ($y = -1$) the adjacent regions are $\mathcal{L}_1, \mathcal{L}_3$ and $\mathcal{L}_6$ ($y = +1$).

Regions with high entropy value, such as $\mathcal{L}_2$ in Figure 5.3, are aggregated directly to X_R.

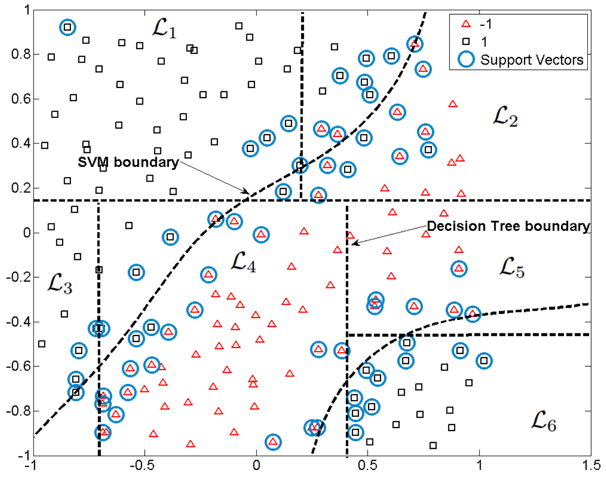


Figure 5.3: **Decision boundaries for SVM and Decision Tree classifier**

Table 5.3: **Data sets used to test DTFSVM**

Data set	Size	Dim	$\lvert y_i = +1\rvert$	$\lvert y_i = -1\rvert$
Iris-setosa	100	4	50	50
Iris-versicolor	100	4	50	50
Iris-virginica	100	4	50	50
Haberman's Survival	306	3	225	81
Ionosphere	351	34	126	225
Breast-Cancer	683	10	444	239
Diabetes	768	8	500	268
Four-class	862	2	307	555
Waveform-0	3,308	40	1,653	1,655
Waveform-1	3,347	40	1,692	1,655
Waveform-2	3,347	40	1,692	1,653
ijcnn1	35,000	22	3,415	31,585
Bank marketing	45,211	16	39,922	5,289
Cod-rna	59,535	8	19,845	36,690
Cross rotated	90,000	2	50,000	40,000
Checkerboard100K	100,000	2	50,000	50,000

5.3 Experiments and results

We compare the performance of our method with respect to that of the SMO[1] [12] and LIBSVM[2] [80]. In the experiments, SMO and LIBSVM are trained with the entire data set. DTFSVM trains a SVM using the points detected by the proposed algorithms.

The experiments were conducted on a computer with the following features: Core i7 2.2 *GHz* processor, 8.0 GB RAM and Windows 7 operating system. The algorithms are implemented in the Java language. The maximum amount of random-access memory given to the Java virtual machine is set to 2.0 *GB*. The reported results correspond to 100 runs of each experiment. The 70% of the data is used to create the training set; the remainder is used for testing.

The kernel used in all experiments is a radial basis function, the value of γ was selected using the grid search method.

[1] implementation http://wiki.pentaho.com/display/DATAMINING/SMO

[2] implementation http://www.cs.iastate.edu/ yasser/wlsvm.html

5.3.1 Data sets used in the experiments

We tested the method on eighteen data sets. Nine of them are publicly available. We modified three sets to create binary problems. In order to explore the performance of the method, one data set was produced artificially.

The public data sets are Haberman's survival, Ionosphere, Breast cancer, Diabetes, Four-class, ijcnn1, Bank marketing and cod-rna.

The data sets modified were Checkerboard [120], Iris and Waveform. The former was generated to have $100,000$ examples (Fig.4.19), the second was separated into three binary classification problems: Iris-setosa, Iris-versicolor and Iris-virginica. The third was also separated in binary classification problems: WaveformBin-0, WaveformBin-1 and WaveformBin-0. The name after the hyphen is the removed class, for example, the data set Iris-setosa consists of all elements of data set Iris minus those elements with label "'setosa"'.

A synthetic data set called Rotated cross (Fig. 4.21) was built; it is linearly inseparable; it has two features and $90,000$ instances.

Table 5.3 shows a summary of the data sets used in the experiments. The column's names and meaning are the following: *Size* is the number of examples in the data set; *Dim* is the number of features in the data set; $|y_i = +1|$ and $|y_i = -1|$ are the number of examples with label $+1$ and -1, respectively.

Table 5.5: **Performance of the DTFSVM algorithm**

Method	Data set	Training time avg (ms)	Training time std dev	Acc (%)	Acc (std dev)	Size (%)
LibSVM	Iris-setosa	3.11	0.24	94.65	0.03	-
SMO		3.01	0.23	94.70	0.02	-
DTFSVM		3.88	0.38	89.21	0.19	79.46
LibSVM	Iris-versicolor	1.51	0.23	100.00	0.00	-
SMO		1.65	0.31	100.00	0.00	-
DTFSVM		1.35	0.37	100.00	0.00	68.19
LibSVM	Iris-virginica	1.76	0.24	100.00	0.00	-
SMO		1.68	0.25	100.00	0.00	-

Continued on next page

Table 5.5 – continued from previous page

Method	Data set	Training time avg (ms)	Training time std dev	Acc (%)	Acc (std dev)	Size (%)
DTFSVM		1.43	0.40	100.00	0.00	68.04
LibSVM	Haberman	9.48	0.79	73.41	0.03	-
SMO		10.51	0.66	73.55	0.03	-
DTFSVM		4.51	0.89	73.08	0.06	80.52
LibSVM	Ionosphere	9.48	0.79	73.41	0.03	-
SMO		8.59	0.99	73.05	0.03	-
DTFSVM		4.51	0.89	73.08	0.06	80.52
LibSVM	Breast-cancer	29.36	1.95	96.63	0.01	-
SMO		35.00	1.45	96.92	0.01	-
DTFSVM		15.78	1.76	95.31	0.08	84.51
LibSVM	Diabetes	81.13	6.27	74.73	0.02	-
SMO		35.00	7.12	77.34	0.03	-
DTFSVM		15.00	4.77	73.25	0.08	84.46
LibSVM	Four-class	18.10	5.87	99.81	0.00	-
SMO		21.23	8.51	98.78	0.02	-
DTFSVM		14.06	7.14	97.55	0.04	70.49
LibSVM	waveformBin-0	1,210.40	180.69	94.39	0.01	-
SMO		1,687.00	78.96	93.99	0.02	-
DTFSVM		193.30	66.57	93.78	0.01	88.22
LibSVM	waveformBin-1	1,659.50	49.39	92.52	0.02	-
SMO		1,840.00	33.89	92.60	0.00	-
DTFSVM		316.60	68.59	91.31	0.01	64.01
LibSVM	waveformBin-2	1,667.00	42.85	92.50	0.01	-
SMO		1,859.00	30.55	92.70	0.02	-
DTFSVM		193.00	75.55	91.98	0.00	89.98
LibSVM	ijcnn1	25,875.25	3,650.04	97.92	3.65	-
SMO		48,923.31	4,605.03	96.01	1.01	-
DTFSVM		5,444.25	375.03	96.06	0.00	89.96

Continued on next page

Table 5.5 – continued from previous page

Method	Data set	Training time avg (ms)	Training time std dev	Acc (%)	Acc (std dev)	Size (%)
LibSVM	Bank marketing	program crashes				-
SMO		program crashes				-
DTFSVM		39, 102.12	6, 062.00	89.91	0.03	91.12
LibSVM	cod-rna	441, 568.70	33, 942.69	93.05	0.01	-
SMO		989, 476.00	89, 279.26	94.01	0.02	-
DTFSVM		23, 224.7	11, 866.53	92.46	0.01	87.67
LibSVM	Rotated Cross	119, 123.14	22, 368.78	94.15	0.02	-
SMO		309, 111.00	2, 078.03	93.65	0.01	-
DTFSVM		6, 726.91	2, 400.82	93.78	0.02	92.53
LibSVM	Checkerboard100K	131, 855.25	3, 116.76	98.86	5.79	-
SMO		285, 621.12	4, 036.99	98.99	0.01	-
DTFSVM		8, 740.25	412.39	97.15	0.00	89.99

5.3.2 Calibration of Parameters

The parameters of our algorithms are the minimum number of objects in each leaf of a decision tree (Min_obj) and the fraction of examples that are taken from each region (δ).

The parameter Min_obj is required by the C4.5 decision tree. A set is not partitioned if it contains fewer samples than Min_obj. Choosing a large value for this parameter, produces a fast training of decision trees, at expense of creating impure partitions.

In general, data reduction methods should retain the patterns closest to opposite class examples. As explained in the preliminary experiments, shown in section 3.5 of Chapter 3, there are different strategies for the selection of examples using the brute force approach. In our DTFSVM method, instead of choosing a constant number of examples, a percentage of each partition is used; the parameter δ is used to decide this.

In order to achieve the best classification accuracy, these two parameters need to be tuned. Table 5.4 shows the values used for each data set in the experiments. These values were selected applying a grid search method.

For large data sets, the values of the parameter Min_obj are sizable. The parameter δ also becomes large for imbalanced data sets, such as Breast-cancer and Diabetes. This is attributed to the fact that for skewed data sets decision trees produce impure partitions; therefore, more points are needed to achieve a better classification accuracy.

In Table 5.4, the regularization parameter for the QPP and the gamma for the kernel are represented by C and γ, respectively.

5.3.3 Results

The performance of the SMO algorithm, LibSVM library and our method are shown in Table 5.5. column Acc (%) is the classification accuracy. Column Size (%) is the percentage of objects deleted from the training set. It can be seen that the training time is improved in practically all cases with our method DTFSVM. For small data sets, the savings in time are not considerable; however, when the training sets are large, the improvement becomes important.

It is noticeable that the accuracy is slightly degraded, but still acceptable. This is due to the fact that some of the SV are not selected during the pre-processing step. The sizes of the training sets are reduced in about 80% and 90% for most cases; this reduction makes faster the training of SVMs.

5.4 Variant of the DTFSVM method

The use of Fisher's linear discriminant in the DTFSVM method selects the closest examples to the opposite class. This helps to improve accuracy at the expense of increasing the computational complexity of the method. In this section, we show a modification of the DTFSVM algorithm called DTDRSSVM (Decision Tree Directed Random Sampling SVM) . This method consists of replacing the linear discriminant by a random sampling. We observed that this produces better results than simple random sampling.

5.4.1 Methods based on clusters for training SVMs

Most clustering based methods to improve the training time of SVMs use centroids of detected clusters, and afterwards, they train the SVM multiple times to refine the separating hyperplane [129][130][131].

DTDRSSVM is similar to clustering methods, in the sense that it first creates clusters of examples, and then, it trains the SVM using a summarized version of the clusters.

The main difference between DTDRSSVM and the others, is that it is not necessary to guess the number of clusters in the training set, which is a drawback in some of those algorithms. The method does not use any distance measure to create the clusters, and, additionally, a SVM is trained only once.

DTDRSSVM is also somewhat similar to random sampling methods. However, a uniform distribution of examples is not assumed in this case, as simple random sampling or other algorithms do. Additionally, a SVM is not trained several times, which is an important difference to other randomized methods.

In this variant, the centroids are not used as a summary of the training sets, but a number of examples is randomly selected. The probability of an example to be chosen varies with respect to its distance to the center of the cluster. In this way, the examples located on exterior boundaries of clusters are preferred because they are probably SV, according to the observation made in [16]. This mechanism is a guided random selection and not a simple random sampling.

In general, decision trees find regions in which almost/all of the contained examples are of the same class. Each region is specified by a leaf of decision trees, it can be seen as a kind of cluster. Clustering methods such as KNN or Fuzzy C-Means, form clusters explicitly by grouping examples, using distance or density measures. In contrast, in our DTDRSSVM , a decision tree discovers pure regions, using a purity function such as the Gini index or the entropy gain. An advantage of decision trees is that they don't require specifying the number of leaves.

Figure 5.4 shows a toy example, and the decision boundaries discovered by a decision tree C4.5. The number of clusters (leaves) is four and all the clusters are not pure.

At this time, it is useful to remember that according to the KKT conditions (2.29a), the examples that are located close to decision boundaries determine the optimal separating hyperplane. Intuition says that the points close to the center of clusters are not too important and can be safely discarded. Other methods that exploit this observation in a different way are reported in [129][64][66].

This intuition can be supported by Eidelheit's separation Theorem (see Therorem 2.1), which states that given two convex sets K_1 and K_2 in a real vector space X, such that K_1 contains interior points, and K_2 contains no interior points of K_1. In other words, K_1 and K_2 lay on the opposite half-spaces determined by the hyperplane H.

Based on this observation, we develop a variant of DTFSVM. Instead of using a linear

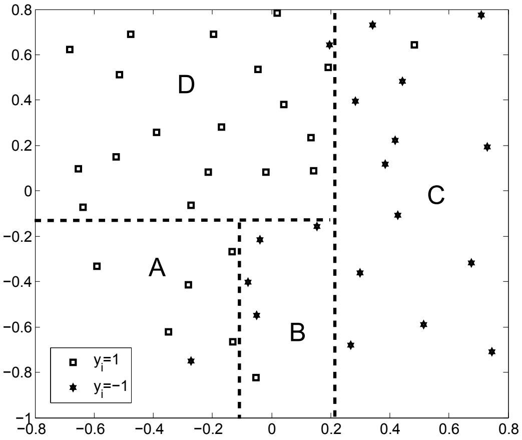


Figure 5.4: **Decision tree applied on a toy example**

discriminant, we exploit the clusters discovered by a decision tree, and execute a random selection. The method can be summarized as follows: Consider two adjacent clusters, say C and D in Figure 5.4, the minority class examples from each one can be removed, so the clusters become pure ones.

Without loss of generality, we can say that each cluster is in a convex set, and they do not intersect. The division created by a decision tree perfectly separates these two clusters, so the split is a separating boundary h_{split}, although probably not the optimal one.

It is possible to select the closest pair of examples that produce the optimal separating hyperplane for clusters C and D, by using a brute force approach; however, this is computationally expensive. Considering that this process must be repeated for each pair of clusters with different majority class, a computationally expensive approach is not a good choice.

Instead of solving this costly problem, we select randomly some examples from each cluster. However, different from simple random sampling, in this method the probability of an example to be selected from a cluster is given by

$$p(x_i) = 1 - \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(\delta)^2}{2\sigma^2}} \tag{5.6}$$

Where η is the (normalized) distance from the example to the center of the cluster and σ is the standard deviation. The greater the distance of an example with respect to the center of the cluster, the higher its probability to be chosen.

Figure 5.5 shows the probability values represented in a gray scale for the toy data set.

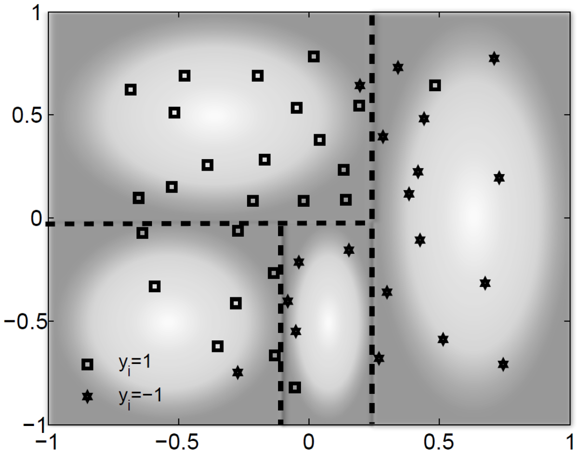


Figure 5.5: **Probabilities within clusters represented in a gray scale**

The darker a region is, the higher its probability to contain more important examples.

5.4.2 Selection using directed random sampling

DTDRSSVM is straightforward to implement, it consists of three simple steps:

1. Train a decision tree, using the whole data set. In this thesis, we use C4.5.
2. Recover all the leaves of the decision tree; these are treated as clusters with low entropy.
3. Select examples from clusters weighting their probability to be chosen as a function of the distances to the center.
4. Train the SVM with the selected instances.

Algorithm 10 shows the pseudocode for the three first steps (data selection). The decision tree used in the algorithm was the C4.5 of Weka [132], and it is called as J48 class within Weka. We modified this class to have access to some of its internal members. Another important modification to the original J48 class, was that not all examples are examined for the selection of the best division. This is because we are interested in discovering regions with low level entropy and not necessarily clusters with zero entropy.

Once clusters have been discovered, each example in a cluster is "normalized" before computing (5.6). This is implemented using

Algorithm 10: Data Selection

```
Input:      $X$ A training set
Output:
    $X_r$ A subset of $X$
Train a $C4.5$ DT on $X$
for each cluster do//Each leaf of DT is seen as a cluster
    for each $x_i$ in current cluster do
        Compute $p(x_i)$ using (5.6)
        Add $x_i$ to $X_r$ randomly according to its $p(x_i)$
    end for
end for
return $X_r$
```

$$x'_{k,j} = \frac{x_{k,j} - min\ \left\{x_{i,j}\right\}}{\left|max\ \left\{x_{i,j}\right\} - min\ \left\{x_{i,j}\right\}\right|} \tag{5.7}$$

Where k is the current example being examined,

$j = 1, ..., d$ is the j^{th} feature of k^{th} example,

$i = 1, ...N$

$min\left\{x_{i,j}\right\}$ and $max\left\{x_{i,j}\right\}$ is the minimum and maximum values of feature j in the cluster respectively.

The “distance” δ , from example k to the center of the cluster, is computed with:

$$\delta_k = \left(\frac{\sum j =_1^d \left(x'_{kj} - 0.5\right)^2}{d}\right)^{1/2} \tag{5.8}$$

with d of the training set.

Because of the previous normalization, the center of every cluster has always a value of 0.5 in each feature or dimension.

The use of d in (5.8) is to produce

$$0 \leq \eta_k \leq 1$$

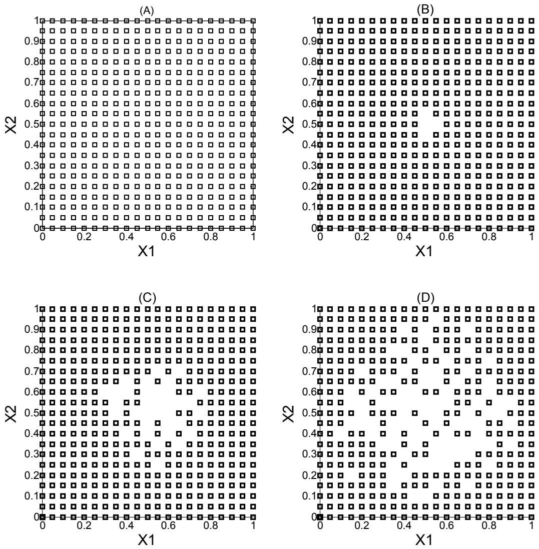


Figure 5.6: **Example of guided selection for different σ values on a uniform distribution of examples**

Parameter σ affects the number of selected examples by the method. Fig. 5.6 shows an example of guided selection for a uniformly distributed set of examples, for different values of σ: (A) Original set, (B) $\sigma = 0.10$, (C) $\sigma = 0.40$, (D) $\sigma = 0.70$,

5.5 Performance analysis

The proposed method uses the algorithm C4.5 to detect clusters. Then, it randomly selects examples from these clusters, assigning a higher probability to be chosen to those that are located on the exterior boundaries of each cluster.

The algorithm C4.5 has the following training time for non-numeric features.

$$O(d \cdot N \cdot log_2(N)))$$

where d is the number of features.

The original Weka implementation of the algorithm C4.5 does not use any discretization of continuous-valued attributes. However, for the presented method, we introduce this modification at the expense of degrading the accuracy of the classifier.

The resulting complexity of the slightly modified C4.5 is

$$O(L \cdot d \cdot m \cdot log_2(m))) \tag{5.9}$$

where L is a non negative integer number. Several values were tested, finally, the value was set to $L = 20$ for the experiments.

At each node, the decision tree tests for the best attribute, considering only L bins, without considering all the examples in the node.

Once the decision tree has been trained, its leaves are used as clusters, and some examples in them are selected. The selection is executed in linear time, so the time complexity of the method is

$$O(L \cdot d \cdot m \cdot log_2(m)) + \sum_{i=1}^{C} N_i \tag{5.10}$$

With C the number of clusters and N_i the number of examples in each cluster.

Observe that

$$\sum_{i=1}^{C} N_i = N$$

Solving the QPP to compute the optimal separating hyperplane adds up a quadratic term to the final training time of the SVM. However, since the total number of selected examples is always lower than the original size of the training set, the proposed method is more efficient than other implementations.

5.6 Experiments and results

In this section, we present the results of the DTDRSSVM method applied on nine different data sets. Given that some of the selected data sets are for multi-class problems, we created binary versions of them by selecting two classes. The total number of data sets was finally thirteen.

5.6.1 Data sets

In order to test the effectiveness of the proposed method, it was evaluated on typical data sets used for classification. Table 5.6 shows the main characteristics of them. Most of the training sets have numeric features and one of them has mixed features.

5.6.2 Setup

The method was implemented in Java, as our programming language, and Weka as our base platform. The C4.5 decision tree implemented in the J48 class of Weka was slightly modified to obtain access to some of its members.

All the experiments were run on a Laptop with Intel core $i7$ 2670QM CPU at 2.2 GHz 8 GB RAM, running the Windows 7 Operating System.

The library LibSVM [80] [3] was used to train SVM, the kernel used for SVM was the RBF function, the values for the parameter γ are shown in Table 5.4. The amount of memory given to the JVM was set to $1,200$ MB. Among all the implementations of the training algorithms for SVMs, LibSVM was selected because it outperformed the other methods in previous tests.

For each run, the training sets were randomly partitioned into two sets: the training (70%) and the testing (30%). The results presented correspond to the average of 100 runs of each experiment; these results are shown in Table 5.7.

5.6.3 Results and discussion

Table 5.7: **Performance of DTDRSSVM**

Method	Data set	Training time avg (ms)	Training time std dev	Acc (%)	Acc std dev	Size (%)
DTDRSSVM ($\eta = 0.01$)	**Iris-setosa**	3.51	2.61	88.12	0.23	70.57
Libsvm		3.11	0.24	94.65	0.03	–
DTDRSSVM ($\eta = 0.99$)	**Iris-versicolor**	1.94	0.29	100.00	0.00	67.73
Libsvm		1.51	0.23	100.00	0.00	–
DTDRSSVM ($\eta = 0.99$)	**Iris-virginica**	2.09	0.79	100.00	0.00	68.28

Continued on next page

[3] implementation http://www.cs.iastate.edu/~yasser/wlsvm.html

Table 5.7 – continued from previous page

Method	Data set	Training time avg (ms)	Training time std dev	Acc (%)	Acc std dev	Size (%)
Libsvm		1.76	0.24	100.00	0.00	-
DTDRSSVM	**Ionosphere**	9.14	1.5	73.00	0.40	59.47
($\eta = 0.70$)						
Libsvm		9.48	0.79	73.41	0.03	-
DTDRSSVM	**Diabetes**	80.17	9.23	75.0	2.1	26.02
($\eta = 0.10$)						
Libsvm		81.13	6.27	74.73	0.02	-
DTDRSSVM	**Waveform-0**	190.01	50.78	92.00	0.90	60.51
($\eta = 0.50$)						
Libsvm		1,210.40	180.69	94.39	0.01	-
DTDRSSVM	**Waveform-1**	215.62	35.37	90.98	0.57	60.17
($\eta = 0.50$)						
Libsvm		1,659.50	49.39	92.52	0.02	-
DTDRSSVM	**Waveform-2**	185.75	80.55	89.75	0.13	60.50
($\eta = 0.50$)						
Libsvm		1,667.00	42.85	92.50	0.01	-
DTDRSSVM	**ijcnn1**	4,027.16	400.00	95.13	0.23	80.33
($\eta = 0.40$)						
Libsvm		25,875.25	3, 650.04	97.92	3.65	-
DTDRSSVM	**bank-full**	4,528.12	7,085.00	86.10	0.50	79.54
($\eta = 0.60$)						
Libsvm			program crashes	program crashes		
DTDRSSVM	**svmguide3**	165.00	5.31	76.35	0.50	80.01
($\eta = 0.30$)						
Libsvm		360.00	12.09	78.82	0.33	-
DTDRSSVM	**cod-rna**	59,156.60	198.13	91.74	1.10	75.59
($\eta = 0.60$)						
Libsvm		441,568.70	33, 942.69	93.05	0.01	-

Table 5.7 shows the comparison between LibSVM and the DTDRSSVM method. It can be seen that for all the experiments, DTDRSSVM reduces the training time of SVM. The achieved accuracy is improved for some data sets. For some training sets, the accuracy is degraded, although acceptable.

For small size data sets, it is neither necessary nor useful to apply a data reduction method such as DTDRSSVM. The main motivation to apply our method on small data sets, was to explore the behavior of the proposed method.

Taking as a particular example the Iris-setosa set, the proposed method gave the worst result in accuracy. Examining both the original training set (see Figure 5.7) and the reduced version after applying the DTDRSSVM method, it can be verified that the method successfully detects examples located close to the decision boundary (see Figure 5.8) and discards those elements far from it. In general, if there are regions where classes overlap, then the accuracy is degraded. Because in practice, most data sets have this characteristic, it is necessary to adjust the parameters of the QPP solver to improve classification accuracy.

The method removes examples that have less chance to contribute to define the optimal separating hyperplane. During this process some SVs can be accidentally deleted. Although this was not used in the results reported in Table 5.7, by adjusting the parameters γ, and the penalty factor C_i in the QPP solver, the accuracy of the classifier can be improved. For example, using $\gamma = 0.85$ and $C = 2$, the achieved accuracy for Iris-setosa is 94.81%. This effect also occurs in other data reduction methods.

The parameter σ can be easily adjusted using the grid method. The values $\sigma = 0.1, 0.2, , \ldots, 0.9$ were tested to select the values that produced the best training time and accuracy. For the modified Iris data sets, this value was calibrated using a finer grid, because the training times are very short.

The results become more noticeable with the largest size data sets. The accuracy is competitive with that achieved with LibSVM using all the training examples. The cases where the accuracy is slightly degraded can be attributed to the fact that some of the SVs were not included during the selection phase.

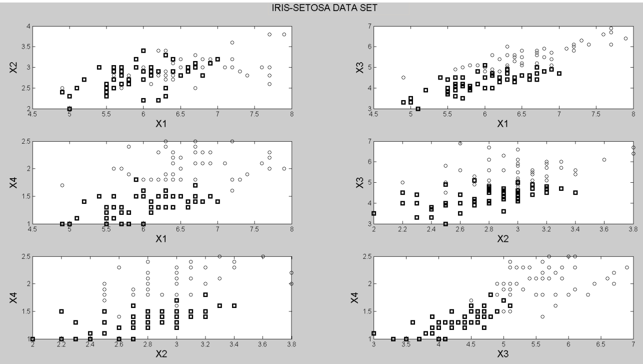


Figure 5.7: **Class Distribution for Iris-setosa Data set**

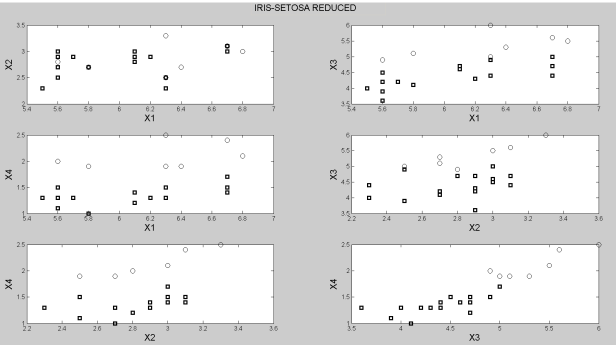


Figure 5.8: **Class Distribution for Iris-setosa reduced**

5.7 Conclusions

In this Chapter, we proposed a data reduction method to improve the training time of SVMs, called DTFSVM. This approach discovers low entropy regions which are then analyzed to detect opposite class regions in order to select examples located close to decision boundaries. The number of selected examples is a subset whose size is considerably smaller than the size of the whole training set; this subset is used to train the SVM. The training time of the SVM is improved with practically all the training sets tested in the experiments, and the accuracy is maintained slightly below the one obtained with the entire training set.

We also proposed the DTDRSSVM method, which is a variant of DTFSVM. Our proposed DTDRSSVM applies random sampling. However, unlike other methods that use simple random sampling, we guide the selection giving more chances to be selected to those examples that are on the boundaries of clusters discovered by a decision tree. Experiments on different data sets, commonly used for the classification task, show the defectiveness of the proposed variant.

The method gives better results when the training sets are large.

The difference in performance between DTFSVM and DTDRSSVM is that the former achieves higher classification accuracies and lower standard deviations.

Table 5.4: **Value of DTFSVM 's parameters used in the experiments**

Method	**Data set**	**C**	γ	δ	**Min_ obj**
			RBF Kernel	Algorithm 9	C4.5
LibSVM	Iris-setosa	3.00	3.00	–	–
DTSVM		2.00	1	0.10	2
LibSVM	Iris-versicolor	1.00	3.00	–	
DTFSVM		3	1	0.10	2
LibSVM	Iris-virginica	1.00	3.00	–	
DTFSVM		3.00	1	0.15	2
LibSVM	Haberman's survival	3.00	2.00	–	–
DTFSVM		2.00	2.00	0.30	15
LibSVM	Ionosphere	2.50	1.55	–	–
DTFSVM		3.00	2.00	0.50	2
LibSVM	Breast-cancer	1.00	$1/n$	–	–
DTFSVM		1.50	$1/n$	0.50	10
LibSVM	Diabetes	1.00	$1/n$	–	–
DTFSVM		2.5	2.00	0.15	5
LibSVM	Four-class	2.00	3.00	–	–
DTFSVM		20.00	3.50	0.15	2
LibSVM	Waveform-0	1.00	$1/n$	–	–
DTFSVM		2.50	$1/n$	0.35	35
LibSVM	Waveform-1	1.00	$1/n$	–	–
DTFSVM		2.50	$1/n$	0.35	35
LibSVM	Waveform-2	1.00	$1/n$	–	–
DTFSVM		2.50	$1/n$	0.35	35
LibSVM	ijcnn1	3.00	$1/n$	–	–
DTSVM		5.00	$1/n$	0.10	30
LibSVM	Bank marketing	3.50	2.3	–	–
DTSVM		4.30	$1/n$	0.05	40
LibSVM	cod-rna	4.50	3.5	–	–
DTFSVM		10.50	3.5	0.35	100
LibSVM	Rotated Cross	2.00	$1/n$	–	–
DTFSVM		3.00	$1/n$	0.15	20
LibSVM	Checkerboard100K	2.00	3.00	–	–
DTFSVM		20.00	3.50	0.35	2
DTFSVM		2.00	$1/n$	0.15	25

Table 5.6: **Data sets for experiments with DTDRSSVM**

Data set	Size	Dim	Class 1	Class 2	Feat. type
Iris-setosa	100	4	50	50	Numeric
Iris-versicolor	100	4	50	50	Numeric
Iris-virginica	100	4	50	50	Numeric
Ionosphere	351	34	126	225	Numeric
diabetes	768	8	500	268	Numeric
svmguide3	1, 243	22	296	947	Numeric
Waveform-0	3, 308	40	1, 653	1, 655	Numeric
Waveform-1	3, 347	40	1, 692	1, 655	Numeric
Waveform-2	3, 347	40	1, 692	1, 653	Numeric
Mushroom	8, 124	112	3, 916	4, 208	Numeric
ijcnn1	35, 000	22	3, 415	31, 585	Numeric
bank-full	45, 211	16	39, 922	5, 289	Mixed
cod-rna	59, 535	8	19, 845	36, 690	Numeric

6 Conclusions and future work

Better than a thousand hollow words, is one word
that brings peace
Siddharta Gautama

6.1 Conclusions

In this work, we developed novel data reduction methods for SVM classifiers. Our methods are applied to data sets with the purpose of removing the objects that are not SV. It this way, the size of the training sets is significantly decreased and the reduced set is used to train the SVM.

We compared our methods against other state-of-the-art methods with the following results: the training time of SVM is significantly improved; the classification accuracies are similar to those obtained with the whole training set and the standard deviations of accuracy remain low.

As a preliminary study to our methods, we explored the performance of two strategies commonly used in other data reduction algorithms: Simple random selection and detection of objects close to an opposite class. The former is the least costly and achieves good classification accuracy if at least 10% of the size of the training set is used. A disadvantage of this approach is that the standard deviation remains large. The second method is unsuitable for large data sets, because it has $O(n^2)$ time and space complexities. Furthermore, this approach obtains lower classification accuracy than random sampling. However, the standard deviations of classification accuracy are low, in addition, the latter method can discover SVs regardless of the type of kernel (i.e., linear or Gaussian) used by the SVM. The objects that present the closest distances to opposite class examples are usually also SVs. A problem the method that computes all the distances between objects, is that some SV cannot be detected, because their distances are greater than other points that are not SV; this occurs, specially, in linearly inseparable cases.

Our methods achieve better classification accuracies than these two naive methods. This is because we detect objects that characterize decision boundaries, and then we extract SV candidates. The standard deviations of classification accuracy are considerably smaller than those obtained with such naive methods. The training time of a SVM using our algorithms is better than that required by algorithms such as SMO, RCH, SCH and LibSVM.

The first method that we present is called CCH-SVM. This approach begins by mapping all examples into a grid. This makes that points located close to other ones are put in same cell of the grid. The key element in CCH-SVM is the detection of vertices of a convex-concave hull, which corresponds to the examples located on the boundaries of the data set. For the linearly separable case, the SV are the closest pair of points of convex hulls; however, for the general case this is not true. The reduction or scaling of a convex hull has two main disadvantages: it is computationally costly and, in general, the accuracy obtained is not good. Current methods that use reduced or scaled convex hulls work only with small data sets. The points that define vertices of a convex-concave hull contain the vertices of a convex hull, and most of these points are close to the edges of convex hull. In this way, our first method works in both the linearly separable and the inseparable cases. We propose a method to apply our CCH-SVM to more than two dimensions.

Unlike the naive method based on distances, CCH-SVM selects objects not only close to their opposite class, but also the distribution of examples is sparser. A disadvantage of CCH-SVM is that is unsuitable for both high dimensional data sets and small data sets. The accuracy achieved by CCH-SVM is maintained slightly lower than the classical SVM trainers that use the whole data set. The training time is not reduced with our CCH-SVM method in these two cases.

The second method that we developed is named DTFSVM. This method discovers low entropy regions to detect objects that describe the decision boundaries, i.e., SV candidates. DTFSVM uses a decision tree to discover regions where most elements are of the same class, and then it selects objects close to their opposite class regions using a linear discriminant. Similar to the naive algorithm based on distances, DTFSVM works in input space, but it is applicable to SVMs with Gaussian kernel. Unlike the simple algorithm based on distances, our method selects objects in a clever way as shown in the results of experiments. DTFSVM does not use random selection as other methods that use a similar approach. This allows to produce repeatable results, high classification accuracies and low standard deviations. A minor variant of the DTFSVM is also proposed, which, instead of using a linear discriminant, executes a random selection in the low entropy regions. This is considered as a directed random sampling and not a simple random sampling. Although the training time obtained with the variant of DTFSVM is better than that achieved with the original DTFSVM, the classification accuracy is lower. DTFSVM and its variant are suitable for mediumsÂą size and large data sets with a number of features typically used in the classification task.

6.2 Future work

We are interested in extending our research to the following problems:

1. Data selection on large and skewed data sets. When the ratio of minority and majority classes is very small, the SVM and other classifiers present a poor performance. In this case, it is necessary to design new data reduction methods not only to reduce the size of the training sets, but also to improve the accuracy of classification, sensitivity and specificity.

2. Classification task with big data. Classic classifiers and also SVMs are unsuitable to deal with sets that contain trillions of data. Classic classifiers and data reduction methods need to be updated, and implemented in parallel and distributed environments.

3. SVMs for data streams. The data stream classification has been an active research area in the last few years. There are algorithms such as VFDT, cVFDT and on demand classification for these type of applications. However, current proposals that apply SVMs are only able to deal with low-speed-rate data streams, and the accuracy that is achieved not good enough. Data reduction methods include the use of time windows or horizons.

4. SVM Classification with embedded devices. One interesting problem is to reduce the complexity of the operations involved in the solution of the quadratic programming problem, as well as reducing the power consumption, so that this type of classifier can be used in low performance devices.

References

[1] M. Hilbert and P. López, "The world' s technological capacity to store, communicate, and compute information," *Science*, pp. 60–65, Apr. 2011.

[2] J. Gantz and D. Reinsel, "Extracting value from chaos," *http://idcdocserv.com/1142*, June 2011, eMC-sponsored IDC Digital Universe study.

[3] H. I. Witten, E. Frank, and M. A. Hall, *Data Mining Practical Machine Learning Tools and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[4] D. L. Olson and D. Delen, *Advanced Data Mining Techniques*. Springer Publishing Company, Incorporated, 2008.

[5] K. J. Cios, R. W. Swiniarski, W. Pedrycz, and L. A. Kurgan, *Data Mining: A Knowledge Discovery Approach*, 1st ed., ser. New York, NY, USA. Springer, 2007.

[6] M. Bramer, *Principles of Data Mining*, ser. Undergraduate Topics in Computer Science. Springer-Verlag, 2007.

[7] K. P. Bennett and E. J. Bredensteiner, "Geometry in learning," in *In Geometry at Work*, 1997.

[8] M. E. Mavroforakis, M. Sdralis, and S. Theodoridis, "A geometric nearest point algorithm for the efficient solution of the svm classification task." *IEEE Transactions on Neural Networks*, vol. 18, no. 5, pp. 1545–1549, 2007. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TNN.2007.900237

[9] X. Peng, “Efficient geometric algorithms for support vector machine classifier,” in *Natural Computation (ICNC), 2010 Sixth International Conference on*, vol. 2, aug. 2010, pp. 875–879.

[10] X. Peng and Y. Wang, “Geometric algorithms to large margin classifier based on affine hulls,” *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 236 –246, feb. 2012.

[11] Z. Liu, J. G. Liu, C. Pan, and G. Wang, “A novel geometric approach to binary classification based on scaled convex hulls,” *Trans. Neur. Netw.*, vol. 20, no. 7, pp. 1215–1220, July 2009.

[12] J. Platt, “Fast training of support vector machines using sequential minimal optimization,” *Advances in Kernel Methods: Support Vector Machines*, pp. pp. 185–208, 1998.

[13] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, “Improvements to platt’s smo algorithm for svm classifier design,” *Neural Comput.*, vol. 13, no. 3, pp. 637–649, Mar. 2001. [Online]. Available: http://dx.doi.org/10.1162/089976601300014493

[14] R.-E. Fan, P.-H. Chen, and C.-J. Lin, “Working set selection using second order information for training support vector machines,” *J. Mach. Learn. Res.*, vol. 6, pp. 1889–1918, December 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1046920.1194907

[15] B. Antoine, S. Ertekin, W. Jason, and B. Léon, “Fast kernel classifiers with online and active learning,” *Journal of Machine Learning Research*, vol. 6, pp. 1579–1619, 2005.

[16] M. Arun Kumar and M. Gopal, “A hybrid svm based decision tree,” *Pattern Recogn.*, vol. 43, no. 12, pp. 3977–3987, Dec. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2010.06.010

[17] C. Li, K. Liu, and H. Wang, “The incremental learning algorithm with support vector machine based on hyperplane-distance,” *Appl. Intell.*, vol. 34, no. 1, pp. 19–27, 2011.

[18] J. Wang, P. Neskovic, and L. N. Cooper, “Selecting data for fast support vector machines training.” in *Trends in Neural Computation*, ser. Studies in Computational Intelligence, K. Chen and L. Wang, Eds. Springer, 2007, vol. 35, pp. 61–84. [Online]. Available: http://dblp.uni-trier.de/db/series/sci/sci35.html#WangNC07

[19] A. Shilton, "Incremental training of support vector machines," *Neural Networks, IEEE Transactions on*, vol. 16, pp. 114–131, 2005.

[20] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 1995. [Online]. Available: http://books.google.com/books?id=sna9BaxVbj8C&printsec=frontcover

[21] W.-H. Chen and J.-Y. Shih, "A study of taiwan's issuer credit rating systems using support vector machines." *Expert Systems with Applications*, vol. 30, pp. 427–435, 2006.

[22] O. Ivanciuc, "Chapter 6 applications of support vector machines in chemistry," 2007.

[23] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Identifying suspicious urls: an application of large-scale online learning," in *ICML*, 2009, p. 86.

[24] D. Sculley and G. Wachman, "Relaxed online svms for spam filtering." in *SIGIR*, W. Kraaij, A. P. de Vries, C. L. A. Clarke, N. Fuhr, and N. Kando, Eds. ACM, 2007, pp. 415–422. [Online]. Available: http://dblp.uni-trier.de/db/conf/sigir/sigir2007.html#SculleyW07

[25] Y. Zhao, H. Xi, and Z. Wang, "A fast online svm algorithm for variable-step cdma power control." in *ICNC (1)*, ser. Lecture Notes in Computer Science, L. Wang, K. Chen, and Y.-S. Ong, Eds., vol. 3610. Springer, 2005, pp. 1090–1099. [Online]. Available: http://dblp.uni-trier.de/db/conf/icnc/icnc2005-1.html#ZhaoXW05

[26] D. Cui and D. Curry, "Prediction in marketing using the support vector machine," *Marketing Science*, vol. 24, no. 4, pp. pp. 595–615, 2005. [Online]. Available: http://www.jstor.org/stable/40056988

[27] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, 1st ed. Cambridge University Press, Mar. 2000.

[28] S. Abe, *Support Vector Machines for Pattern Classification*, ser. Advances in Pattern Recognition. London: Springer-Verlag, Jul. 2005.

[29] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals Eugen.*, vol. 7, pp. 179–188, 1936.

[30] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[31] A. A. Hussein, B. Jaume, M. V. Butz, and X. Llor, "Online adaptation in learning classifier systems: Stream data mining," University of Illinois at Urbana-Champaign, Tech. Rep. 2004031, June 2004.

[32] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc. Elsevier, 2006.

[33] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

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

[35] D. G. Luenberger, *Optimization by Vector Space Methods*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1997.

[36] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[37] S. Abe, "Analysis of multiclass support vector machines," *IEEE Trans actions on Pattern Analysis and Machine Intelligence*, pp. 437–440, 2003. [Online]. Available: http://www2.kobe-u.ac.jp/~abe/pdf/cimca2003.pdf

[38] C.-W. Hsu and C.-J. Lin, "A simple decomposition method for support vector machines," *Mach. Learn.*, vol. 46, pp. 291–314, March 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1012427100071

[39] J. Chen, C. Wang, and R. Wang, "Combining support vector machines with a pairwise decision tree," *Geoscience and Remote Sensing Letters, IEEE*, vol. 5, no. 3, pp. 409 –413, july 2008.

[40] A. Christmann and I. Steinwart, *Support Vector Machines*, 1st ed., ser. Information Science and Statistics. Springer New York, Sep. 2008, vol. 1.

[41] V. Kecman, *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models*. Cambridge, MA, USA: MIT Press, 2001.

[42] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[43] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, ser. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[44] T. M. Mitchell, *Machine Learning*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[45] G. Wang, "A survey on training algorithms for support vector machine classifiers," in *Proceedings of the 2008 Fourth International Conference on Networked Computing and Advanced Information Management - Volume 01*, ser. NCM '08, vol. 1. Washington, DC, USA: IEEE Computer Society, sept. 2008, pp. 123–128. [Online]. Available: http://dx.doi.org/10.1109/NCM.2008.103

[46] M. Doumpos, "An experimental comparison of some efficient approaches for training support vector machines," *Operational Research*, vol. 4, pp. 45–56, 2004, 10.1007/BF02941095. [Online]. Available: http://dx.doi.org/10.1007/BF02941095

[47] J. L. Balcázar, Y. Dai, and O. Watanabe, "A random sampling technique for training support vector machines," in *Proceedings of the 12th International Conference on Algorithmic Learning Theory*, ser. ALT '01. London, UK, UK: Springer-Verlag, 2001, pp. 119–134. [Online]. Available: http://dl.acm.org/citation.cfm?id=647719.736065

[48] J. L. Balcazar, Y. Dai, and O. Watanabe, "Provably fast support vector regression using random sampling," Feb. 10 2002. [Online]. Available: http://citeseer.ist.psu.edu/626945.html;http://www.lsi.upc.es/~balqui/postscript/svmregr.ps

[49] J. L. Balcázar, Y. Dai, J. Tanaka, and O. Watanabe, "Provably fast training algorithms for support vector machines," *Theory Comput. Syst*, vol. 42, no. 4, pp. 568–595, 2008. [Online]. Available: http://dx.doi.org/10.1007/s00224-007-9094-6

[50] Y. jye Lee and O. L. Mangasarian, "Rsvm: Reduced support vector machines," in *Data Mining Institute, Computer Sciences Department, University of Wisconsin*, 2001, pp. 00–07.

[51] Y.-J. Lee and S.-Y. Huang, "Reduced support vector machines: A statistical theory," *IEEE Transactions on Neural Networks*, vol. 18, no. 1, pp. 1–13, 2007.

[52] L.-J. Chien, C.-C. Chang, and Y.-J. Lee, "Variant methods of reduced set selection for reduced support vector machines," *J. Inf. Sci. Eng.*, vol. 26, no. 1, pp. 183–196, 2010.

[53] Y.-J. Lee and O. L. Mangasarian, "SSVM: A smooth support vector machine," *Computational Optimization and Applications*, vol. 20, pp. 5–22, 2001, data Mining Institute, University of Wisconsin, Technical Report 99-03. ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/99-03.ps.

[54] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, ser. COLT '92. New York, NY, USA: ACM, 1992, pp. 144–152. [Online]. Available: http://doi.acm.org/10.1145/130385.130401

[55] A. Shigeo and I. Takuya, "Fast training of support vector machines by extracting boundary data," in *ICANN '01: Proceedings of the International Conference on Artificial Neural Networks*. London, UK: Springer-Verlag, 2001, pp. 308–313.

[56] Y.-G. Liu, Q. Chen, and R.-Z. Yu, "Extract candidates of support vector from training set," in *Machine Learning and Cybernetics, 2003 International Conference on*, vol. 5, nov. 2003, pp. 3199–3202 Vol.5.

[57] Z.-W. Li, J. Yang, and J.-P. Zhang, "Dynamic incremental svm learning algorithm for mining data streams," in *Proceedings of the The First International Symposium on Data, Privacy, and E-Commerce*, ser. ISDPE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 35–37. [Online]. Available: http://dx.doi.org/10.1109/ISDPE.2007.69

[58] P. E. Hart, "The condensed nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 14, pp. 515–516, 1968.

[59] K. Gowda and G. Krishna, "The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (corresp.)," *IEEE Trans. Inf. Theor.*, vol. 25, no. 4, pp. 488–490, Sep. 1979. [Online]. Available: http://dx.doi.org/10.1109/TIT.1979.1056066

[60] F. Angiulli, “Fast condensed nearest neighbor rule,” in *Proceedings of the 22nd international conference on Machine learning*, ser. ICML ’05. New York, NY, USA: ACM, 2005, pp. 25–32. [Online]. Available: http://doi.acm.org/10.1145/1102351.1102355

[61] H. Shin and S. Cho, “Pattern selection for support vector classifiers,” in *Proceedings of the Third International Conference on Intelligent Data Engineering and Automated Learning*, ser. IDEAL ’02. London, UK, UK: Springer-Verlag, 2002, pp. 469–474. [Online]. Available: http://dl.acm.org/citation.cfm?id=646288.686626

[62] ——, “Fast pattern selection for support vector classifiers,” in *Proceedings of the 7th Pacific-Asia conference on Advances in knowledge discovery and data mining*, ser. PAKDD’03. Berlin, Heidelberg: Springer-Verlag, 2003, pp. 376–387. [Online]. Available: http://dl.acm.org/citation.cfm?id=1760894.1760944

[63] ——, “How many neighbors to consider in pattern pre-selection for support vector classifiers?” in *IJCNN 2003*, 07 2003, pp. 565–570.

[64] ——, “Neighborhood property-based pattern selection for support vector machines,” *Neural Comput.*, vol. 19, no. 3, pp. 816–855, March 2007.

[65] R. Wang and S. Kwong, “Sample selection based on maximum entropy for support vector machines,” in *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on*, vol. 3, july 2010, pp. 1390–1395.

[66] X. Jiantao, H. Mingyi, W. Yuying, and F. Yan, “A fast training algorithm for support vector machine via boundary sample selection,” in *Neural Networks and Signal Processing, 2003. Proceedings of the 2003 International Conference on*, vol. 1, dec. 2003, pp. 20–22 Vol.1.

[67] B.-Z. Qiu, F. Yue, and J.-Y. Shen, “Brim: An efficient boundary points detecting algorithm,” in *Advances in Knowledge Discovery and Data Mining*, ser. Lecture Notes in Computer Science, Z.-H. Zhou, H. Li, and Q. Yang, Eds. Springer Berlin / Heidelberg, 2007, vol. 4426, pp. 761–768.

[68] C. Xia, W. Hsu, M. Lee, and B. Ooi, “Border: efficient computation of boundary points,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 18, no. 3, pp. 289–303, march 2006.

[69] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 201–212. [Online]. Available: http://doi.acm.org/10.1145/342009.335415

[70] E. Sung, Z. Yan, and L. Xuchun, "Accelerating the svm learning for very large data sets," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 2, 0-0 2006, pp. 484 –489.

[71] J. Cervantes, X. Li, and W. Yu, "Support vector machine classification based on fuzzy clustering for large data sets," in *Lectures notes on Computer Science*, A. Gelbukh and C. Reyes-Garcia, Eds., Mexican International Conference on Artificial Intelligence. Springer-Verlag Berlin Heidelberg, November 2006, pp. 572–582.

[72] J. Cervantes, X. Li, W.Yu, and K. Li, "Support vector machine classification for large data sets via minimum enclosing ball clustering," *Neurocomputing*, vol. 71, pp. 611–619, 2008.

[73] R. Fletcher, *Practical methods of optimization; (2nd ed.)*. New York, NY, USA: Wiley-Interscience, 1987.

[74] N. List and S. Hans-Ulrich, "A general convergence theorem for the decomposition method," in *COLT*, 2004, pp. 363–377.

[75] D. Luenberger and Y. Ye, *Linear and Nonlinear Programming*, ser. International Series in Operations Research & Management Science. Springer, 2008. [Online]. Available: http://books.google.com.mx/books?id=-pD62uvi9lgC

[76] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *NNSP'97*. IEEE, 1997, pp. 276–285.

[77] R. Saunders, C. Stitson, J. Weston, L. Bottou, B. Scholkopf, and A. Smola, "Support vector machine reference manual," Department of Computer Science, Royal Holloway University of London, Egham, Surry TW20 0EX, UK, Tech. Rep. CSD–TR–98–03, Oct. 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.7760

[78] T. Joachims, “Making large-scale svm learning practical,” *Advances in Kernel Methods–Support Vector Learning*, pp. 169–184, 1998.

[79] C. Campbell, “Algorithmic approaches to training support vector machines: a survey,” in *ESANN 2000, 8th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 26-28, 2000, Proceedings*, 2000, pp. 27–36.

[80] C. Chih-Chung and L. Chih-Jen, “Libsvm: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, 2011.

[81] J. A. K. Suykens and J. Vandewalle, “Least squares support vector machine classifiers,” *Neural Process. Lett.*, vol. 9, pp. 293–300, June 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=326394.326408

[82] G. Fung and O. L. Mangasarian, “Incremental support vector machine classification,” in *SDM*, 2001, pp. 77–86.

[83] H. P. Graf, E. Cosatto, L. Bottou, I. Durdanovic, and V. Vapnik, “Parallel support vector machines: The cascade svm,” in *In Advances in Neural Information Processing Systems*. MIT Press, 2005, pp. 521–528.

[84] L. Bao-Liang, W. Kai-An, and W. Yi-Min, “Comparison of parallel and cascade methods for training support vector machines on large-scale problems,” in *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, vol. 5, Aug. 2004, pp. 3056 – 3061.

[85] R. Collobert, Y. Bengio, and S. Bengio, “Scaling large learning problems with hard parallel mixtures,” *International Journal on Pattern Recognition and Artificial Intelligence (IJPRAI)*, vol. 17, no. 3, pp. 349–365, 2003.

[86] J. xiong Dong, A. Krzyzak, and C. Suen, “Fast svm training algorithm with decomposition on very large data sets,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 4, pp. 603–618, april 2005.

[87] S. Qiu and T. Lane, “Parallel computation of rbf kernels for support vector classifiers,” in *SDM*, 2005.

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

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

[90] T. Serafini, L. Zanni, and G. Zanghirati, "Some improvements to a parallel decomposition technique for training support vector machines," in *PVM/MPI*, 2005, pp. 9–17.

[91] T. Eitrich and B. Lang, "On the optimal working set size in serial and parallel support vector machine learning with the decomposition algorithm," in *AusDM '06: Proceedings of the fifth Australasian conference on Data mining and analystics*. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2006, pp. 121–128.

[92] D. DeCoste and K. Wagstaff, "Alpha seeding for support vector machines," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 345–349. [Online]. Available: http://doi.acm.org/10.1145/347090.347165

[93] D. Feng, W. Shi, H. Guo, and L. Chen, "A new alpha seeding method for support vector machine training," in *Advances in Natural Computation*, ser. Lecture Notes in Computer Science, L. Wang, K. Chen, and Y. Ong, Eds. Springer Berlin / Heidelberg, 2005, vol. 3610, pp. 418–418. [Online]. Available: http://dx.doi.org/10.1007/11539087_87

[94] D. Roobaert, "Directsvm: A fast and simple support vector machine perceptron," in *Proceeding of IEEE, International Workshop on Neural Networks for Signal Processing*, 2000, pp. 356–365.

[95] Y. Liu, Q. Chen, Y. Tang, and Q. He, "An incremental updating method for support vector machines," in *APWeb*, 2004, pp. 426–435.

[96] Z. Hao, S. Yu, X. Yang, F. Zhao, R. Hu, and Y. Liang, "Online ls-svm learning for classification problems based on incremental chunk," in *ISNN (1)*, 2004, pp. 558–564.

[97] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Advances in Neural Information Processing Systems (NIPS*2000)*, vol. 13, 2000, pp. 409–415. [Online]. Available: http://citeseer.ist.psu.edu/cauwenberghs00incremental.html

[98] F. Orabona, C. Castellini, B. Caputo, J. Luo, and G. Sandini, "On-line independent support vector machines," *Pattern Recognition*, vol. 43, no. 4, pp. 1402–1412, 2010.

[99] F. Parrella, "Online support vector regression," Master's thesis, University of Genoa, June 2007.

[100] H. Duan, X. Shao, W. Hou, G. He, and Q. Zeng, "An incremental learning algorithm for lagrangian support vector machines." *Pattern Recognition Letters*, vol. 30, no. 15, pp. 1384–1391, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/prl/prl30.html#DuanSHHZ09

[101] H. Gâlmeanu and R. Andonie, "Implementation issues of an incremental and decremental svm," in *ICANN (1)*, 2008, pp. 325–335.

[102] S. Ruping, "Incremental learning with support vector machines," *Proceedings 2001 IEEE International Conference on Data Mining*, pp. 641–642, 1999. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=989589

[103] G. Grinblat, L. Uzal, H. Ceccatto, and P. Granitto, "Solving nonstationary classification problems with coupled support vector machines," *Neural Networks, IEEE Transactions on*, vol. 22, no. 1, pp. 37–51, jan. 2011.

[104] L. Ralaivola and F. d'Alché Buc, "Incremental support vector machine learning: A local approach," in *ICANN*, 2001, pp. 322–330.

[105] A. Bordes and L. Bottou, "The Huller: a simple and efficient online SVM," in *Machine Learning: ECML 2005*. Springer Verlag, 2005, pp. 505–512, lNAI 3720.

[106] C. Sun, "Closest pairs data selection for support vector machines," in *proceedings of the 21st national conference on Artificial intelligence - Volume 2*, ser. AAAI'06. AAAI Press, 2006, pp. 1926–1927. [Online]. Available: http://dl.acm.org/citation.cfm?id=1597348.1597511

[107] D. M. J. Tax and R. P. W. Duin, "Support vector data description," *Mach. Learn.*, vol. 54, no. 1, pp. 45–66, Jan. 2004.

[108] L. Zhang, N. Ye, W. Zhou, and L. Jiao, "Support vectors pre-extracting for support vector machine based on k nearest neighbour method," in *Information and Automation, 2008. ICIA 2008. International Conference on*, june 2008, pp. 1353–1358.

[109] Z.-q. Wang, C.-t. Wang, and F. Hou, "An effective method for support vectors selection in kernel space," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 1, dec. 2008, pp. 872–875.

[110] S. Canu, L. Bottou, and S. Canu, *Training Invariant Support Vector Machines using Selective Sampling*. MIT Press, 2007. [Online]. Available: http://eprints.pascal-network.org/archive/00002954

[111] S. Vishwanathan and M. Narasimha Murty, "Geometric svm: a fast and intuitive svm algorithm," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 2, 2002, pp. 56 – 59 vol.2.

[112] ——, "Ssvm: a simple svm algorithm," in *Proceedings of the 2002 International Joint Conference on Neural Networks IJCNN02*. IEEE, 2002.

[113] Y. Ming-Hsuan and A. Narendra, "A geometric approach to train support vector machines," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 1, p. 1430, 2000.

[114] Crisp and Burges, "A geometric interpretation of υ-svm classifiers," *NIPS*, vol. 12, pp. 244–250, 2000.

[115] D. Zhou, B. Xiao, H. Zhou, and R. Dai, "Global geometry of svm classifiers," *Institute of Automation Chinese Academy of Sciences*, 2002. [Online]. Available: http://kyb.tuebingen.mpg.de/publications/pdfs/pdf2587.pdf

[116] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Inform. Process. Lett.,*, vol. 2, pp. 18–21, 1973.

[117] J. C. A. Moreira and M. Y. Santos, "Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points," in *GRAPP (GM/R)*, 2007, pp. 61–68. [Online]. Available: http://dblp.uni-trier.de

[118] K. P. Bennett and E. J. Bredensteiner, "Duality and geometry in svm classifiers," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, pp. 57–64.

[119] B. Gnedenko, I. Beliaev, A. Solovev, and R. Barlow, *Mathematical methods of reliability theory*, ser. Probability and mathematical statistics. Academic Press, 1969. [Online]. Available: http://books.google.com.mx/books?id=a7c8AAAAIAAJ

[120] T. Ho and E. Kleinberg, "Checkerboard data set," http://www.cs.wisc.edu/math-prog/mpml.html, 1996.

[121] F. Chang, C.-Y. Guo, X.-R. Lin, and C.-J. Lu, "Tree Decomposition for Large-Scale SVM Problems," *Journal of Machine Learning Research*, 2010. [Online]. Available: http://www.jmlr.org/papers/volume11/chang10b/chang10b.pdf

[122] B. Fei and J. Liu, "Binary tree of svm: a new fast multiclass training and classification algorithm," *Neural Networks, IEEE Transactions on*, vol. 17, no. 3, pp. 696 –704, may 2006.

[123] M. Lu, C. L. P. Chen, J. Huo, and X. Wang, "Multi-stage decision tree based on inter-class and inner-class margin of svm," in *Proceedings of the 2009 IEEE international conference on Systems, Man and Cybernetics*, ser. SMC'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1875–1880. [Online]. Available: http://dl.acm.org/citation.cfm?id=1732003.1732025

[124] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[125] H. Zhao, Y. Yao, and Z. Liu, "A classification method based on non-linear svm decision tree," in *Proceedings of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery - Volume 04*, ser. FSKD '07. Washington,

DC, USA: IEEE Computer Society, 2007, pp. 635–638. [Online]. Available: http://dx.doi.org/10.1109/FSKD.2007.6

[126] D. Tax and P. Laskov, "Online SVM learning: from classification to data description and back," in *Proc. NNSP*, C. e. a. Molina, Ed., 2003, pp. 499–508.

[127] S. Katagiri and S. Abe, "Selecting support vector candidates for incremental training," in *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 2, oct. 2005, pp. 1258–1263.

[128] A. Shashua, "On the relationship between the support vector machine for classification and sparsified fisher's linear discriminant," *Neural Process. Lett.*, vol. 9, no. 2, pp. 129–139, Apr. 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1018677409366

[129] D. Wang and L. Shi, "Selecting valuable training samples for svms via data structure analysis," *Neurocomput.*, vol. 71, pp. 2772–2781, August 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1401261.1401308

[130] F. Fernandez and P. Isasi, "Local feature weighting in nearest prototype classification," *Neural Networks, IEEE Transactions on*, vol. 19, no. 1, pp. 40 –53, jan. 2008.

[131] Z.-J. Chen, B. Liu, and X.-P. He, "A svc iterative learning algorithm based on sample selection for large samples," in *Machine Learning and Cybernetics, 2007 International Conference on*, vol. 6, aug. 2007, pp. 3308–3313.

[132] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009. [Online]. Available: http://dx.doi.org/10.1145/1656274.1656278