UNIVERSIDAD AUTÓNOMA DE MADRID

A Practical View of Large-Scale Classification: Feature Selection and Real-Time Classification

by

Irene Rodríguez Luján

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Escuela Politécnica Superior
Departamento de Ingeniería Informática

Under the supervision of Carlos Santa Cruz Fernández

May 2012

"We can do that in some philosophy seminar, but in the real world there're real cases that ought to concern us"

Noam Chomsky

Abstract

The increasing volume of data during last years together with the emergence of classification systems requiring real-time responses have given rise to a new school of thought in machine learning owing to the impossibility of applying many of the classification methods that, traditionally, had been successful. This impossibility can be due to hardware limitations, the data cannot be completely stored in memory as required by many of the existing algorithms, or requirements concerning the training and/or classification times that traditional classifiers are not able to fulfill. In any case, it is mandatory to adapt the current machine learning solutions to the new scenario which, inevitably, leads to the design of simple and easily scalable algorithms. In particular, this thesis proposes two complementary solutions that make it possible to face up those classification problems requiring real-time predictions.

The first of these contributions is a new feature selection algorithm independent of the classifier and capable of reducing its computational cost in the classification and training phases. The proposed method can be categorized into the multivariate filters group and it yields similar classification rates that its state-of-the-art counterparts while reducing their computational complexity. In addition, the new algorithm is reformulated in a higher dimensional space induced by a kernel turning out to be equivalent to the well-known Kernel Fisher Discriminant Analysis (KFDA). This equivalence is theoretically proven providing new insights into the KFDA.

The second contribution of this work is focused on the design of an algorithm capable of classifying patterns in few milliseconds. Motivated by the difficulty of applying nonlinear Support Vector Machines (SVMs) to real-time classification domains, the new method attempts to approximate the nonlinear decision boundaries by means of piecewise linear functions while locally preserving the maximum margin criteria. The results presented in this thesis show how the proposed algorithm can bridge the gap between the simplicity but low accuracy of linear SVMs and the effectiveness but sophistication of nonlinear SVMs in real-time classification systems.

In conclusion, the large amount of data makes it sometimes necessary to leave aside the precision of the most complex models in favor of approximate solutions fulfilling the requirements of the classification system.

Resumen

El incremento del volumen de datos durante los últimos años junto con la aparición de sistemas de clasificación que requieren respuestas en tiempo real han provocado una nueva corriente dentro del campo del aprendizaje automático debido a la imposibilidad de aplicar muchos de los métodos de clasificación tradicionalmente exitosos. Esta imposibilidad puede venir dada por limitaciones en el hardware (los datos no pueden ser almacenados completamente en la memoria tal y como requerían la mayoría de los algoritmos existentes), o por requerimientos sobre el tiempo de aprendizaje y/o clasificación que los clasificadores tradicionales no son capaces de satisfacer. En cualquiera de los casos, es necesario adaptar las soluciones del aprendizaje automático a este nuevo escenario lo que conduce, inevitablemente, al diseño de algoritmos relativamente sencillos y fácilmente escalables. En particular, esta tesis plantea dos soluciones complementarias que permiten abordar satisfactoriamente aquellos problemas de clasificación que requieran predicciones en tiempo real.

La primera de estas contribuciones consiste en un nuevo método de selección de variables independiente del clasificador y capaz de reducir la carga computacional de las fases de predicción y aprendizaje. El algoritmo propuesto se enmarca dentro de las técnicas de filtro multivariable y es capaz de igualar las tasas de acierto de los métodos del estado del arte reduciendo, a la vez, su complejidad. Adicionalmente, este nuevo método se reformula en un espacio inducido por un núcleo y se demuestra su equivalencia con el popular discriminante de Fisher para núcleos, dando lugar a nuevas interpretaciones sobre el mismo.

La segunda de las contribuciones se centra en el diseño de un algoritmo capaz de emitir predicciones en pocos milisegundos. Motivado por la difícil aplicación de las Máquinas de Vectores Soporte no lineales a problemas de predicción en tiempo real, el nuevo algoritmo trata de aproximar las fronteras de decisión no lineales mediante funciones lineales a trozos manteniendo localmente el criterio de máximo margen. Los resultados presentados en esta tesis muestran como el algoritmo propuesto permite salvar las diferencias entre la simplicidad pero bajo acierto de los modelos lineales y la efectividad pero sofisticación de los modelos no lineales en sistemas de clasificación en tiempo real.

En conclusión, la gran cantidad de datos hace necesario, en ocasiones, prescindir de la exactitud de los modelos más complejos en favor de soluciones aproximadas que garanticen el cumplimiento de los requisitos del sistema de clasificación.

Acknowledgements

First of all, I would like to thank my advisor Carlos Santa Cruz for teaching me to *learn from data* and introducing me in the "real-world" problems beyond the mathematical formalism. I really appreciate his initiative and enthusiasm to start this thesis being his patience and confidence essential for the development and completion of this work. In second place, but not less important, thanks Ramón. Thanks for your welcome in San Diego, your brilliant ideas and your teaching, your optimism and your valuable guidance; in short, thanks for all your dedication to this thesis. Thanks Professor Charles Elkan for your commitment and for involving me in the Artificial Intelligence group activities.

Y lo más importante: gracias mamá y papá porque esta tesis es el resultado de vuestros esfuerzos por darme la mejor educación pero sobre todo, gracias por ayudarme a enfrentar los problemas con fortaleza, valentía y responsabilidad. Gracias por levantarme en los momentos más difíciles. Gracias Luisete por tu comprensión infinita, por hacerme sonreir cuando me ponía *marilloros* y por ayudarme tantas veces a relativizar las cosas. And thanks Jaime, thanks for your support and patience, especially in the final stage. In short, thanks for making things easier. Thanks, thanks, thanks!

Thanks Teresita and Mariaje for being always there, you are simply great. Thanks my SAFA friends Javi, Noe, Lauri, Álvaros and Natalia for the *cañas y trinas* to take a break and to sort out the world. Thanks my *Infomates* friends Edu, Jorge, Juanda, Fran, María, Lucy, Jesús and Elena for asking me about my thesis title every time we met ;). Thanks my colleagues at Instituto de Ingeniería del Conocimiento (IIC) for all the unforgettable moments living there during all these years. Especially thanks Álvaro, Jorge and Sandra. Thanks José and Eduardo, thanks for your wise advice and guidance during all these university years. I am so grateful to you. Thanks Julia Díaz for your help. Thanks my colleagues at GB2S group for your support in the final stage.

And finally, I would like to express my gratitude to the grants that have supported my research: the FPU grant kindly funded by the Universidad Autónoma de Madrid and the pre-doctoral grant given by "Cátedra IIC de análisis de patrones de comportamiento". I should also mention the institutions where I had developed this thesis: the Computer Science Department of Universidad Autónoma de Madrid, the Instituto de Ingeniería del Conocimiento at the same university and the BioCircuits Institute and Computer Science and Engineering Department of University of California San Diego (UCSD).

Contents

Algorithms and Methods Acronyms

CART	**C**lassification **A**nd **R**egression **T**rees
CFS	**C**orrelation based **F**eature **S**election
(K)**CV**	(K-fold) **C**ross-**V**alidation
FDA	**F**isher **D**iscriminant **A**nalysis
H-LSVM	**H**ierarchical **L**inear **S**upport **V**ector **M**achine
KFDA	**K**ernel **F**isher **D**iscriminant **A**nalysis
KQPFS	**K**ernel **Q**uadratic **P**rogramming **F**eature **S**election
MaxDep	**Max**imal **Dep**endency
MaxRel	**Max**imal **Rel**evance
MID	**M**utual **I**nformation **D**ifference
MIQ	**M**utual **I**nformation **Q**uotient
mRMR	**m**inimun **R**edundancy **M**aximum **R**elevance
PCA	**P**rincipal **C**omponent **A**nalysis
QPFS	**Q**uadratic **P**rogramming **F**eature **S**election
RFE-SVM	**R**ecursive **F**eature **E**limination **S**upport **V**ector **M**achine
SFS	**S**treamwise **F**eature **S**election
SGD	**S**tochastic (sub)**G**radient **D**escent
SMO	**S**equential **M**inimal **O**ptimization
SVM	**S**upport **V**ector **M**achine
VC	**V**apnik-**C**hervonenkis

To my family and friends

Chapter 1

Introduction

This thesis is about **machine learning**, a discipline of **artificial intelligence** based on the *discovery* of relationships between objects without needing a lot of expert knowledge. As the name suggests, the aim of machine learning is to learn a solution of a problem instead of understanding the underlying model which can be, in most of the cases, extremely complex. For example, in a fraud detection system the machine learning approach would try to answer the question *Is this transaction fraud or not?* leaving aside the explanation of what is the cause of the verdict. From this point of view, machine learning can solve problems from different areas because their solutions are not focused on the specific domain in which they are going to be applied. The challenge is to *make a machine learn a solution* from past data. The adoption of a machine learning solution can indeed improve the expert knowledge by finding out relationships that the experts, even knowing them, are not able to explain or describe. In addition, the machine can handle with volumes of data unmanageable by the experts and it also *learns* faster.

The assumption behind any machine learning system is that some general rules about the relationships between objects can be inferred from the behavior of few examples, known as **training samples (patterns or examples)**, and thus *learning* involves an inductive inference from the particular examples to general rules. These rules can be understood as functions f, known as **classifiers** or **predictors**, transforming input examples from the data domain $\mathcal{X}$ to output objects (**predictions**) in the space $\mathcal{Y}$:

$$f : \mathcal{X} \longrightarrow \mathcal{Y}.$$

The input and output spaces are determined by the nature of the problem in question; for example, in the credit card fraud detection system the space of the inputs $\mathcal{X}$ would consist of information about credit card transactions such as temporal information or the amount of the transaction whereas the output space $\mathcal{Y}$ would be formed by a discrete variable taking two possible values: *fraud* or *not fraud*.

The characteristics of the input and output spaces divide machine learning algorithms into different categories being the most widely applied supervised and unsupervised learning. **Supervised learning** can be regarded as *learning with teacher* in the sense that the model has a feedback about its learning process. Training samples consist of pairs formed by the input example and the desired output value and the objective of the classifier is to predict the output for patterns with unknown label. In turn, the properties of the output space $\mathcal{Y}$ categorize supervised learning as **classification**, the possible outputs (**labels**, **targets** or **classes**) are a set of discrete values, or as **regression**, the output takes continuous values. By contrast, **unsupervised learning** (or **clustering**) *learns without teacher* since it does not make use of the labels information and it attempts to group the training data into subsets (**clusters**) in such a way that patterns belonging to the same cluster would be similar according to certain criterion. Among other approaches, **semisupervised learning** can be considered as a kind of supervised learning but not having all the training patterns labeled. Finally, **reinforcement learning** learns how an agent has to take decisions in an environment in order to maximize some notion of long-term reward. The main difference with supervised learning is that the pairs input-output are not presented to the agent beforehand and hence, the decisions are adopted in an online manner by weighting up the exploration and the exploitation of the knowledge.

In any of the scenarios, the machine learning approach can be explained as a sequence of steps, namely: collect the data, choose the features and the classifier and finally, train and evaluate the classifier as presented in Figure 1.1. Although the election and the training of the classifier represent the essence of the machine learning task, other phases such as finding a proper representation of data and performing a fair evaluation of the

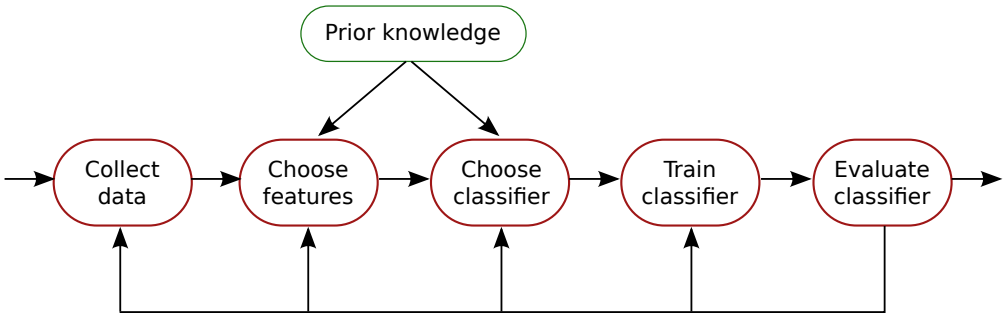


FIGURE 1.1: Diagram of the design of a machine learning system. Figure adapted from [1].

classifier should not be pushed into the background since the success of the final model also depends on them.

The starting point corresponds to the **collection of data** from one or several sources. Intuitively, the higher amount of examples available, the easier inference of general rules – in fact, Chapter 2 will support this idea on the basis of statistical learning theory. In spite of the apparent benefits of having a large number of training samples, this situation can burden the applicability of conventional machine learning classifiers as it will be discussed further down in Section 1.1. On the other hand, those problems with few training patterns require special attention in order to guarantee reliable models. Unfortunately, the samples shortage can not be solved immediately by the system designer as the data collection usually represents an independent step previous to the design process.

Each of the samples gathered consists of a set of **attributes** (**variables** or **features**) containing information about different aspects of the problem. While some of them might be uninformative for the learning process, there could exist relevant relationships among the original features that are not explicitly represented by a single attribute. Therefore, **choosing relevant features** from data entails two stages (i) feature construction to bring out new attributes and (ii) feature selection to keep only those variables which are relevant to explain the data. Both processes can be conducted by an expert who introduces some prior knowledge about the problem and/or they can adopt a machine learning strategy which infers dependences among features from a subset of samples. Besides the preparation of the inputs for the classification process strictly speaking, dimensionality reduction methods can accelerate the training and prediction processes, they can also reduce the storage requirements and save resources for the next data collection, they may imply a performance improvement especially in those models sensitive

to noisy data and they can improve the data understanding. The importance of this phase is reflected in the amount of effort devoted in machine learning to design effective and efficient algorithms capable of detecting the most relevant attributes as it will be shown in Section 2.2. Indeed, feature selection constitutes a research field in machine learning by itself.

Once a good representation of the data has been found, it is time to **choose the model** to learn the data which can be regarded as choosing the family of functions which the classifier f belongs to. The selection of an appropriate model is vital for achieving good performance as it assumes certain characteristics in the data which, in case of being wrong, can degrade the classifier's reliability. At this point, some expert knowledge can be useful to filter some classifiers among all the possibilities. Furthermore, the evaluation of the suitability of each candidate might consider different criteria depending on the final purpose. Despite the most natural choice seems to be selecting that classifier that *better learns* the data, other factors such as the time needed to learn and to predict or the simplicity and understandability of the result can also play an important role. Moreover, diverse classifiers (hypothesis) might yield similar accuracies in which case, the known **Occam's Razor** establishes that the simplest models are preferred as they should provide better stability and generalization.

The selection of a family of functions to model the data has stated a set of hypothesis about the data distribution but it is necessary to adjust this general description to the particular problem to solve. This adaptation is called **training** and it is the basis of *learning* from the training data. Chapter 2 will show how this learning process can be formulated as the adjustment of the parameters of certain mathematical function according to some criterion. That makes the *machine learning* possible. In the same way, the selection of the model has implied the choice of a family of mathematical functions which can go from simple hyperplanes to the most complex nonlinear mappings.

And to finish, the **evaluation of the trained model** is mandatory in order to check if the original goals of the system have been fulfilled. The simplest way to measure the success of the classifier is to determine how well it has executed its task (classification, clustering, semisupervised classification or reinforcement learning) in the training samples. However, it is not a good practice as the classifier might have learn the training patterns *by heart* and it has not been capable of inferring general rules from them. In

this case, the evaluation of the classifier would be extremely positive but its performance for unseen patterns could be really poor. A fairer qualification is obtained in practice by evaluating the classifier in a set of examples not seeing during the training process. To do that, two of the most popular techniques are the **hold-out** method and the **k-fold cross validation (*k*cv)** technique. Hold-out approach consists in dividing the available samples into two disjoint subsets: the training and **validation sets** using the first one to train the model and the second one to evaluate it. On the other hand, the k-fold validation strategy randomly generates k partitions of the available data using one of the partitions to evaluate the model and the rest form the training set. This procedure is repeated the k times corresponding to consider each of the partitions individually as the test set and the overall performance is given by the average over each partition. As aforementioned, even though the ultimate aim of any machine learning system is to discover relationships from data, other factors can guide the decision about the goodness of the classifier and especially those which have been decisive in the selection of the model such as the training or prediction time, to name just a few. If the final evaluation is not positive, it may be necessary to reconsider one or several of the decisions taken in the previous steps. Once the final model is achieved, it is a good practice to carry out an external evaluation of its performance by means of a set of patterns not used in neither the training nor the validation phases and popularly known as **test set**.

1.1 Thesis Scope

The goals and contributions of this thesis are focused on the improvement of machine learning solutions when they are applied to large-scale classification problems.

Why classification? Because many problems in the real world can be understood as a supervised classification task and they cover areas as diverse as those presented in Table 1.1.

And why in large-scale domains? Because the scope of machine learning solutions has changed dramatically in the last years with the emergence of "large-scale" applications in domains like web, finance or biology, among others. The Internet expansion has caused such explosion in the volume of information available that web applications have become the main consumer of large-scale machine learning solutions. Examples of

Domain	**Application**	**Input space**	**Output space**	**Ex.**
Bioinformatics	Sequence analysis	DNA / protein sequence	Types of genes	[2]
Document classification	Internet search	Text document (bag of words)	Categories (business, sports, etc.)	[3]
Document image analysis	Reading machine	Document image	Alphanumeric characters, words	[4]
Biometrics	Personal identification	Fingerprints, iris, sign, etc.	Identity accepted/rejected	[5]
Speech recognition	Automatic transcription of spoken language into readable text	Speech waveform	Transcribed words	[6]
Fraud detection	Credit card transactions	Information of the use of cards by different customers and its legality	Fraud / not fraud	[7]

TABLE 1.1: Examples of supervised classification applications. Table adapted from [8] including some related works (*Ex.*).

the amount of data traveling on Internet coming from sources as diverse as documents, social networks, images and videos or on-line shopping are presented in Figure 1.2.

At a first glance, having many training samples might help to infer more general models due to the variety of situations described in the input space. But this apparent appealing contrasts with the difficulties associated with handling such amount of data. The initial concerns in machine learning – and Computer Science in general – about optimizing the algorithms' complexity to deal with the hardware limitations faded with the development of more powerful machines capable of working with the available volume of data. However, the interest in designing and implementing efficient algorithms have reemerged lately due to the hardware progress has not followed the exponential growth of data. Aspects like the **training and test times** as well as **memory requirements** are these days extremely relevant in the design of a large-scale classification system. Furthermore, it is worth reflecting on the real usefulness of this extra training data and its involvement in the performance of the model.

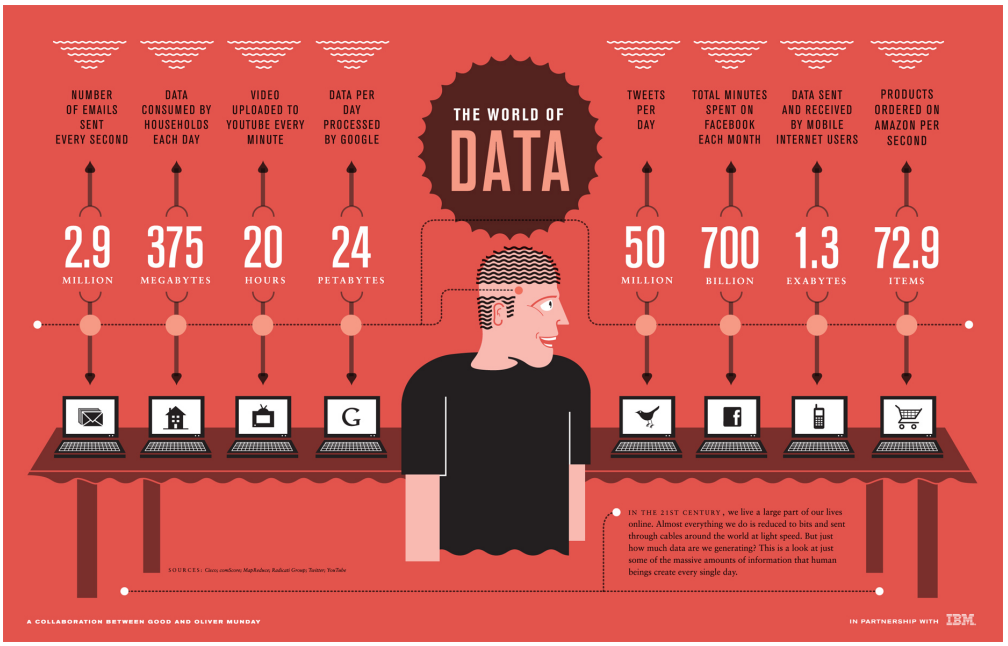


FIGURE 1.2: Massive amount of data created everyday on Internet. Infographic extracted from the article *"The World of Data We're Creating on the Internet"* published in *Good Magazine* (October 2010).

But what can be considered "large-scale"? Unfortunately, it does not exist a single answer to this question. The definition of large-scale is not as simple as provide a threshold for the number of training samples and/or dimensions since it depends on the requirements and characteristics of each classification problem. Mainly, two different definitions can be raised: a problem can be categorized as "large-scale" if (i) it does not fit in RAM and then, the conventional machine learning algorithms cannot be straightforwardly used, or (ii) in spite of having all the information loaded in memory, the classifier cannot be applied on a single machine in *reasonable* time because it does not fulfill the time requirements associated with the training or classification phases. These limitations make it necessary to revisit traditional models and algorithms since the solution is not as simple as "throw more machines": algorithms that were outstanding not many years ago, such as polynomial time ones, nowadays might turn into an unfeasible task as the number of patterns increases. This phenomenon affects not only the training costs but also the classification response times: methods providing acceptable training times in large-scale scenarios might become useless because they are not fast enough to classify a new patterns in few milliseconds. Therefore, another dimension comes into play in the large-scale scenario: the **scalability**. Actually, the challenge goes one step further as it requires to make a more transcendental decision about the result of the machine learning process: the necessity of exact solutions in

weaker models versus the validity of approximate solutions in stronger models. Recent advances point out that the second approach has won the battle since in some cases the effectiveness of the exact but weak solutions is insufficient and what is more, the computation or the evaluation of the exact solution might be prohibitive even for a linear model.

The scalability challenge has been tackled from different points of view:

- **Subsampling the data**. Probably the simplest approach: only a subset of the training samples are used with conventional methods. Although this alternative can be a good point of reference and it would accelerate the classification in those cases in which the resulting model depends somehow on the number of training samples, hiding some information to the classifier can degrade its accuracy. Truthfully, it does not represent a real solution to large-scale problems since it solves a simplified problem.

- **MapReduce** consists in partitioning the training data into several groups and distributing them to multiple machines. This way, each machine solves a subproblem and the process ends with the combination of the individual results. this approach can be also adopted for speeding up the classification by dividing the evaluation of the model into several tasks. In spite of the attractiveness of this massive parallelization, many of the machine learning algorithms cannot be adapted to this architecture.

- **Hardware solution: Graphics Processing Unit (GPU)**. GPUs are dedicated to graphical processing and float-point operations in order to lighten the CPU workload. They are usually arranged in a parallel architecture pursuing to run many concurrent threads rather than one fast thread as in CPUs. They have been successfully applied to many machine learning methods in which they can reduce the training time from weeks to hours [9, 10] or make their applicability to real-time operation systems possible [11, 12]. Unfortunately, not all the machine learning tasks can be parallelized at operation level and there still exists a dependence on the CPU which might be a bottleneck.

- **Adaptation of existing algorithms**. The *purest approach* from the point of view of machine learning. While all the preceding solutions are concerned about

reducing the amount of data and/or parallelizing the algorithms' operations, the adaptation of existing algorithms to large-scale domains requires a deep knowledge of their basis in order to make them scalable either in the training or in the classification phases.

1.2 Thesis Contributions

The research work developed in this thesis is in line with the last approach focused on algorithms and tools to make large-scale supervised classification problems possible. This issue can be raised from complementary points of view in accordance with the machine learning design cycle presented in Figure 1.1. As stated above, the simplest way to tackle large-scale problems would be modifying the data collection step in order to reduce the amount of available information but, it might imply a significant worsening in the classification accuracies. The contributions of this thesis cover three complementary phases in which there are still room of improvement in large-scale domains: feature selection and classifier choice and training. On the one hand, **feature selection**, working as a preprocessing step, can be applied to many learning algorithms and it improves their scalability when their complexity depends somehow on the dimension of the patterns. In large-scale problems is crucial to evaluate the success of the feature selection not only in terms of the dimensionality reduction but also as a function of its scalability. On the other hand, the classifier selection and training are more domain-dependent tasks since most of the times they should consider the final purpose of the algorithm beyond the classification accuracy. For example, medical domains might be concerned about the interpretability of the model whereas web applications might desire low response times disregarding the classifier understanding. Among all the possible criteria, the second part of the contributions will deal with large-scale classifiers requiring predictions in few milliseconds that is, **real-time classification**. To summarize, the contributions of this thesis are presented in the following points:

1. A new feature selection algorithm for large-scale and high-dimensional datasets named **Quadratic Programming Feature Selection (QPFS)** given rise to the following publication:

I. Rodriguez-Lujan, R. Huerta, C. Elkan, and C. Santa Cruz. Quadratic programming feature selection. Journal of Machine Learning Research, 99:1491–1516, 2010.

The new algorithm can be categorized as multivariate filter (Section 2.2.1) as it takes into account not only the relevance between each feature with the class but also possible redundancies between attributes. QPFS allows to use more powerful but more computationally intensive classification algorithms by reducing the dimensionality of the patterns.

2. The reformulation of the QPFS algorithm to generate new attributes. The new algorithm called **Kernel Quadratic Programming Feature Selection (KQPFS)** makes use of the popular kernel trick (Section 2.3.2.1) to implicitly introduce non-linearities in the data and it is shown the equivalence with the well-known Kernel Fisher Discriminant Analysis (KFDA). These results are published in:

   I. Rodriguez-Lujan, C. Santa Cruz, and R. Huerta. On the equivalence of Kernel Fisher Discriminant Analysis and Kernel Quadratic Programming Feature Selection. Pattern Recognition Letters, 32(11):1567 – 1571, 2011.

3. A new learning algorithm for binary classification problems named **Hierarchical Linear Support Vector Machine (H-LSVM)** which has given rise to a manuscript which is currently under review:

   I. Rodriguez-Lujan, C. Santa Cruz, and R. Huerta. Hierarchical Linear Support Vector Machine. Pattern Recognition. Under second review.

   The new algorithm attempts to be a solution to the high classification cost in large-scale domains of one of the most successful algorithms nowadays: Support Vector Machines (SVMs) (Section 2.3.2). The new algorithm enables to classify new patterns in few milliseconds by means of piecewise linear decision functions while locally preserve the generalization properties the SVM models.

All these contributions will be presented in separate chapters including a thorough analysis of the strengths and weaknesses of the state-of-the-art solutions, as well as experimental results showing the practical performance and utility of the algorithms here proposed. Software implementations of the QPFS and H-LSVM algorithms are publicly available as additional material:

`http://sites.google.com/site/irenerodriguezlujan/documents/QPFS-1.0.zip`

`https://sites.google.com/site/irenerodriguezlujan/HLSVM-1.1.zip` .

1.3 Thesis Structure

The rest of chapters in this thesis are organized as follows. Chapter 2 introduces the concepts necessary for the comprehension of the contents developed throughout this thesis. Specifically, a brief review of statistical learning theory provides a mathematical statement of essential concepts such as risk minimization, underfitting, overfitting, generalization, model complexity or Vapnik-Chervonenkis (VC) dimension, among others. Immediately afterwards, a general taxonomy of feature selection algorithms and a review of the most promising approaches for large-scale problems are provided. The chapter closes with a brief description of three learning algorithms highly related to the contents of this thesis namely: Fisher Discriminant Analysis (FDA), Support Vector Machines and decision trees. The three following chapters correspond to the three contributions of this work introduced in the previous section. Chapter 3 presents a new feature selection algorithm including an analysis of the scalability of existing approaches, the derivation of the error bound of the new method and a lengthly experimental section. As an extension, Chapter 4 reformulates the algorithm proposed in Chapter 3 in a kernel space. Theoretical and empirical proofs of its eivalence with the Kernel Fisher Discriminant Analysis are given besides a comparison between the computational complexity of the standard KFDA solution against the KQPFS one. Focusing now on real-time prediction systems, Chapter 5 starts with some insights into the works addressing the acceleration of the prediction time of nonlinear SVMs. After that, the new algorithm is presented together with a theoretical analysis of generalization error bounds. Then, the experimental setup places the new method with respect to linear and nonlinear SVMs, compares it against some of the techniques introduced at the beginning of the chapter and provides a practical intuition about the resulting error bounds. Embracing the solutions provided by Chapters 3 and 5, Chapter 6 analyzes the strengths and weaknesses of a machine learning system combining these two approaches. To finish, Chapter 7 states the conclusions derived from this thesis and points out some possible lines of further work. Additional material is provided in the appendices: Appendix A summarizes the notation following along this work, Appendix B gives some insights into the Nyström matrix

diagonalization, a keystone in the algorithm proposed in Chapter 3, and Appendix C establishes the convergence properties of the algorithm introduced in Chapter 5.

Chapter 2

Supervised Classification

This chapter is a general review of the notions necessary to the understanding of the algorithms presented in Chapters 3-5 focused on dimensionality reduction and real-time classification. The chapter starts with the mathematical statement of the supervised learning problem and the introduction of the risk minimization and classifier's complexity concepts which allow to enshrine the contributions of this thesis in the generalization framework. The following section establishes the basis of the algorithm proposed in Chapters 3 and 4 providing a taxonomy of feature selection techniques as well as a description of some of the most known feature selection algorithms. Finally, a survey of different approaches for supervised classification is addressed at the end of the chapter, specifically the Fisher Discriminant Analysis, that will play an essential role in Chapter 4, and Support Vector Machines and decision trees, whose combination is the basis of the method introduced in Chapter 5.

2.1 Statistical Learning Theory

Chapter 1 has described intuitively machine learning as a discipline oriented to *learn from the data* in the sense that it tries to infer the most adequate model to describe relationships from a given **training set** S. The training set is usually formed by a finite number of training samples, N, in a M-dimensional space:

$$S = \{(\boldsymbol{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} \mid i = 1, \ldots, N\}.$$

Furthermore, each sample $\boldsymbol{x}_i \in \mathcal{X}$ is characterized by M features being x_i^j the j-th feature of the i-th pattern. The random variable corresponding to the j-th feature is represented as X_j and then, $\{X_1, X_2, \ldots, X_M\}$ is the set of all the features in $\mathcal{X}$. In the same way, the random variable associated with the labels is denoted as Y.

Chapter 1 has categorized machine learning algorithms as a function of the structure of $\mathcal{Y}$. If information about $\mathcal{Y}$ is not considered during the training phase, the problem is called unsupervised learning. On the contrary, if the pair $(\boldsymbol{x}_i, y_i)$ is taking into account in the construction of the model, the task is called supervised learning. In turn, supervised learning can be divided into regression problems ($\mathcal{Y} = \mathbb{R}$) or classification problems ($|\mathcal{Y}|$ is finite). Commonly, the classification problem is called binary if $|\mathcal{Y}| = 2$ and muti-class, otherwise.

According to the scope of this thesis, **supervised classification problems** are assumed in what follows. In this case, the challenge of a learning algorithm is to determine a function $f : \mathcal{X} \longrightarrow \mathcal{Y}$ which transforms objects from the input space $\mathcal{X}$ to the set of possible discrete labels $\mathcal{Y}$. However, the complexity of many machine learning problems prevents the existence of a deterministic one-to-one function f making necessary the introduction of a probabilistic framework. Assuming that the process to be modeled has been generated by an unknown and fixed probability distribution $P(X, Y)$ over the space $\mathcal{X} \times \mathcal{Y}$ and supposing that the training patterns have been drawn independently and identically from P, the probability of observing a particular label $y \in \mathcal{Y}$ subject to the observation $\boldsymbol{x} \in \mathcal{X}$ is given by the Bayes' law,

$$P(Y|X) = \frac{P(X, Y)}{P(X)} = \frac{P(X|Y)P(Y)}{P(X)}.$$

Then, knowing the distribution $P(Y|X)$, the **Bayes' decision rule** of the **Bayes' optimal solution** to assign a label y to the pattern $\boldsymbol{x}$ is given by

$$f^*(\boldsymbol{x}) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} P(Y = y | X = x). \tag{2.1}$$

Apart from its natural origin, it can be shown that the Bayes' decision rule is optimal by introducing the loss function concept. The loss function, $l : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}^+$, quantifies what is the cost of predicting $f(\boldsymbol{x})$ when the real label of $\boldsymbol{x}$ is y. In classification problems, an intuitive way to define the loss function is [1]

$$l(f(\boldsymbol{x}), y) = \begin{cases} 0 & f(\boldsymbol{x}) = y \\ 1 & f(\boldsymbol{x}) \neq y \end{cases}. \tag{2.2}$$

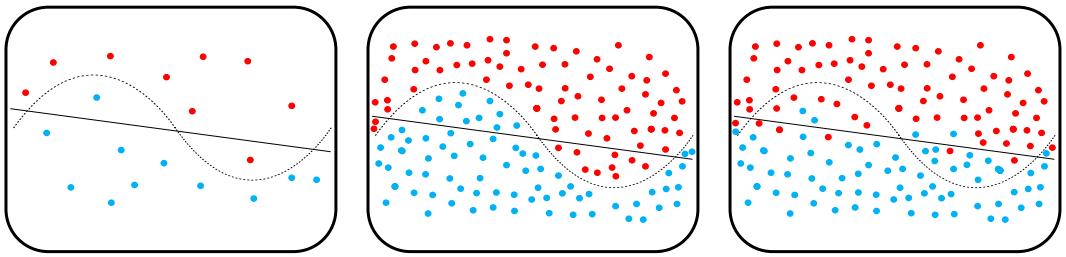
The aim of many learning algorithms is to find the function f minimizing the **expected error or risk** defined as,

$$R(f) = \mathbb{E}_P \left[l(f(\boldsymbol{x}), y) \right],$$

where $\mathbb{E}_P$ is the expectation with respect to the joint distribution $P(X, Y)$. The Bayes' decision rule defined in Equation 2.1 minimizes $R(f)$ for reasonably defined loss functions as the one presented in Equation 2.2, and thus the classification problem can be solved in an optimal way. Unfortunately, the probability $P(X, Y)$ is usually unknown in practice. One possible solution would be to estimate it but it entails several difficulties such as making assumptions about the underlying distribution or compiling enough data to have a reliable approximation. As alternative, the expected risk is replaced by the **empirical error or risk** computed as the average errors over the training set:

$$R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^{N} l(f(\boldsymbol{x}_i), y_i). \tag{2.3}$$

[1] Although other penalties are possible, especially in those asymmetric problems such as medical diagnosis or fraud detection, this is the lost function considered through this thesis.



(a) (b) (c)

FIGURE 2.1: The overfitting and underfitting phenomena. In Figure 2.1(a) the small number of training patterns makes it impossible to determine which of the models is closer to the real distribution of the data. Conversely, in Figure 2.1(b) the large size of the training size provides a good approximation of the distribution of data. In this case, the dashed model is closer to the ground truth while the solid one underfits the data. Figure 2.1(c) shows the opposite scenario: the solid boundary seems correct whereas the dashed model overfits the training data. Figure adapted from [13].

Consequently, learning algorithms will try to find a decision function f which minimizes $R_{\text{emp}}(f)$. However, measuring the empirical error on the training set can yield models with poor **generalization** on unseen data, even when the empirical risk on the training set is small. This circumstance is known as **overfitting**: the model is complex enough to learn the training patterns *by heart* but it is not able to generalize to unseen patterns. The other extreme, **underfitting**, can also happen: the model is too coarse to learn anything from particular data. A simple example of the underfitting and overfitting dilemmas is presented in Figure 2.1. As a balance between overfitting and underfitting, it would be desirable that as the number of training patterns increases ($N \to \infty$), the decision function f founded by the learning algorithm converges to the function with the lowest expected error. In other words, it seems reasonable to impose **consistency** in machine learning algorithms which can be achieved by controlling the model complexity in order to protect it against overfitting and underfitting. The model complexity is directly connected to the family of functions $\mathcal{U}$ which are potential solutions and intuitively, it is determined by the number of different predictions that can be obtained when choosing functions from this family.

2.1.1 The Vapnik-Chervonenkis Dimension

One possible way to quantify the complexity of a family of functions is the **Vapnik-Chervonenkis dimension** which, broadly speaking, counts the maximum number of points that can be shattered for all possible labellings using functions of the class $\mathcal{U}$. A

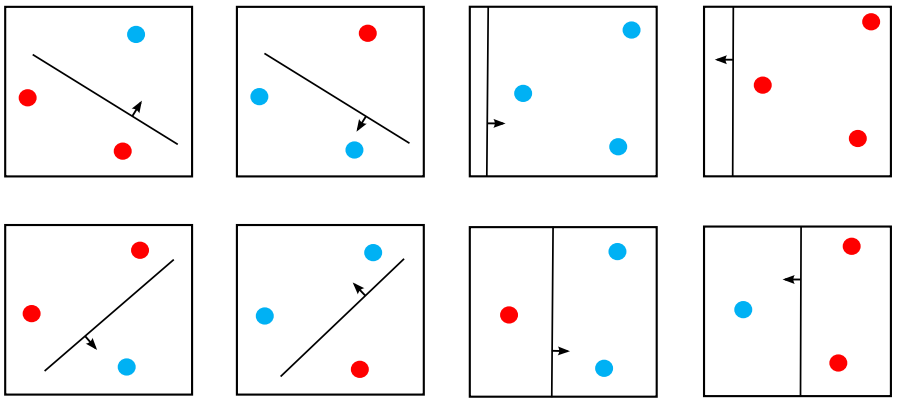


FIGURE 2.2: All the possible dichotomies of three points in $\mathbb{R}^2$ shattered by oriented hyperplanes.

detailed analysis of the Vapnik-Chervonenkis theory is out of the scope of this thesis; however, the VC dimension of an hyperplane in the space $\mathcal{X}^M$ is studied as an example and starting point of the generalization error bound of the algorithm presented in Chapter 5 which assumes a two class classification problem with $\mathcal{Y} = \{-1, 1\}$ and a solution of the form $f(\boldsymbol{x}) = (\boldsymbol{w} \cdot \boldsymbol{x}) + b$. The family of functions containing all the possible hyperplanes defined by $\boldsymbol{w}$ and b can be expressed as $\mathcal{U} = \{f : \mathcal{X} \longrightarrow \mathbb{R} \mid f(\boldsymbol{x}) = (\boldsymbol{w} \cdot \boldsymbol{x}) + b\}$. As stated above, the VC dimension of $\mathcal{U}$ is the maximal number of points, U, such that there exists $f \in \mathcal{U}$ verifying $f(\boldsymbol{x}_i) = y_i$ for arbitrary $y_i \in \{-1, 1\}$. In this case, the maximum number of M-dimensional points with an arbitrary labeling that can be completely separated by an hyperplane is $M + 1$ ($\text{VCdim}(\mathcal{U}) = M + 1$) since, for any configuration of $M + 1$ points one of them can be chosen as origin in $\mathcal{R}^{\mathcal{M}}$ such that the remaining M points are linearly independent[2]. This reasoning cannot be extended for $M + 2$ points because, fixing one point as the origin, the remaining $M + 1$ points in a M-dimensional space are not linearly independent. Figure 2.2 shows how all dichotomies of three not collinear points in $\mathbb{R}^2$ can be separated by an hyperplane in $\mathbb{R}^2$.

2.1.2 Complexity Control and Regularization

The VC dimension is relevant because it provides bounds on the expected risk as a function of the empirical risk and the number samples N. Paying attention to the aim of this thesis, the following bound is applied to supervised classification problems using the 0-1 loss function (Equation 2.2).

[2]Assuming that the $M + 1$ points are not collinear.

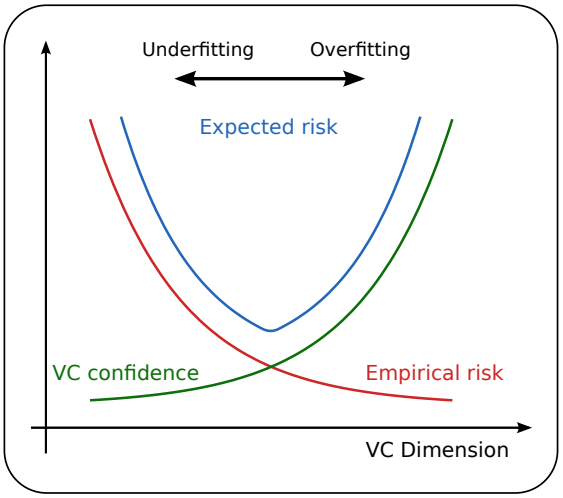


FIGURE 2.3: Intuition behind the Vapnik-Chervonenkis bound (Theorem 2.1). The optimal model is found as a balance between the empirical risk and the VC confidence. Figure adapted from [13].

Theorem 2.1. *[14] Let $VCdim(\mathcal{U})$ the VC dimension of the family of functions $\mathcal{U}$ and let R_{emp} the empirical error as defined in Equation 2.3 using the 0-1 loss function. For all $\delta > 0$ and $f \in \mathcal{U}$, with probability at least $1 - \delta$ the following bound holds:*

$$R(f) \leq R_{emp}(f) + \sqrt{\frac{VCdim(\mathcal{U})\left(\log \frac{2N}{VCdim(\mathcal{U})} + 1\right) - \log(\frac{\delta}{4})}{N}} \tag{2.4}$$

for $N > VCdim(\mathcal{U})$ training samples randomly drawn from $P(X, Y)$.

The second term on the right hand side is called the **VC confidence** and it becomes smaller as the ratio $\frac{N}{\text{VCdim}(\mathcal{U})}$ gets larger. This fact is consistent with the idea discussed above and presented in Figure 2.1 that the empirical risk minimization is only a good approximation when sufficient data is available. Regarding the discussion of the VC bound, two extremes arise from Equation 2.4: the simplest models make the VC confidence small but the empirical risk might be notably high whereas, families of functions extremely complex reduce the training error at the price of large square root term. However, the best classifier is usually in the middle (Figure 2.3): it minimizes the empirical risk – to learn from data– while keeping relatively simple models – to prevent overfitting –.

To summarize, the practical implications of the preceding analysis for the construction of a machine learning algorithm are:

1. Find a function with small empirical error.
2. Try to keep the CV confidence small by means of (i) reduce VC dimension of the classifier and (ii) use as many training patterns as possible.

In spite of its elegance, the VC theory can be hardly applied in practice because the upper bound in Theorem 2.1 might be higher than one, the VC dimension cannot be easily computed in some cases (for example, in neural networks) or even, it can be infinite (as in the case of the nearest neighbor) [13]. Nevertheless, VC bounds provide helpful insights into the machine learning problems that should be taken into account in the design of an efficient classifier.

A practical approach to the VC bound is to consider, besides the empirical risk, a term which penalizes the complexity of the model. The new term is so-called **regularization** term and the learning algorithm minimizes the **regularized error** or risk defined as,

$$R_{\text{reg}}(f) = \lambda\Omega(f) + R_{\text{emp}}(f). \tag{2.5}$$

Where $\Omega : \mathcal{U} \to \mathbb{R}^+$ is the **regularizer** which quantifies somehow the complexity of a classifier f. The regularization constant $\lambda > 0$ establishes a trade-off between minimizing the empirical error and minimizing the regularizer. It is expected that appropriate choices of λ and Ω will provide a classifier f that also minimizes the VC bound. In practice, the regularizer Ω usually is fixed beforehand and λ is adjusted in the validation phase. Even though regularization was firstly introduced by Tikhonov and Arsenin to solve ill-posed problems [15], it has been widely used in different fields of machine learning to control the complexity of the model and to protect it against overfitting or underfitting. Indeed, Section 2.3.2 will show how the regularized risk functional can be successfully applied to linear models leading to the well-known Support Vector Machines.

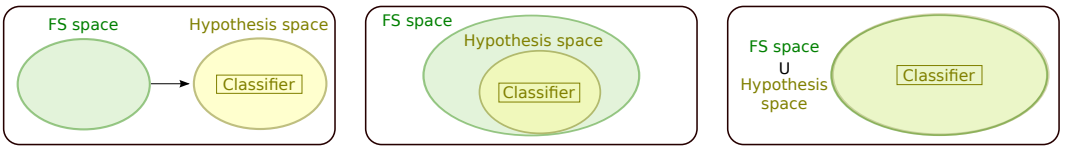
The statistical theory developed further up establishes a group of guidelines to have in mind in the design cycle of a classifier and it suggests that accurate and simple models should provide stability and good generalization. This point of view brings up other benefits derived from the contributions presented in this thesis beyond the computational time savings. From one hand, algorithms proposed in Chapters 3 and 4 attempt to reduce the dimension of the original input space by removing noisy features which can perturb the learning process and hence, they can be very helpful to achieve high accuracy and good generalization; on the other hand, the new classification method presented in Chapter 5 is based on the local minimization of the regularized risk in order to reach high level of generalization. Therefore, the three approaches group together under the umbrella of generalization.

2.2 Feature Selection Fundamentals

Increasing the dimensionality of patterns without control is not a good practice in machine learning because it may end in an unmanageable situation for the learning algorithm in which the relevant information is immersed in a sea of possibly irrelevant, noise and redundant features (attributes or variables). Feature selection faces this problem trying to answer when a feature is informative in order to select a subset of relevant features to construct the final model. The incorporation of a successful feature selection technique in a machine learning cycle can draw three main benefits:

- Substantial **gain in computational efficiency**, especially important for any application that requires classifier execution in real-time as the one presented in Chapter 5.
- **Scientific discovery** by determining which features are most correlated with the class labels, which may in turn reveal unknown relationships among features.
- **Reduction of the risk of overfitting** especially if too few training instances are available. This problem is particularly serious in situations with high dimensionality relative to training set sizes.

Document categorization [16], prosthesis control [17, 18], cardiac arrhythmia classification [19], fMRI analysis, gene selection from microarray data [2, 20–22], real-time



(a) (b) (c)

FIGURE 2.4: Interaction between the feature selection method and the classifier in for each feature selection technique: 2.4(a) filter, 2.4(b) wrapper and 2.4(c) embedded. Figure adapted from [21].

identification of polymers [23], and credit card fraud detection [24] are some real-life domains where these gains are especially meaningful.

Before going into details, it is worth distinguishing feature selection methods from other attempts to reduce the dimension of the data like **projection techniques** or **compression procedures**. Among the first group, well-known algorithms like Principal Component Analysis (PCA) [1] or Fisher Discriminant Analysis (FDA) (Section 2.3.1), extracting new features as linear combinations of the original ones, can be found; the second category is composed of those algorithms that simply reduce the storage requirements of the data ignoring the ultimate objective of classification. In any case, feature selection algorithms are a completely different approach since they preserve the original representation of the attributes to favor the understandability of the final model.

The unquestionable importance of the feature selection paradigm has caused such barrage of methods during the last fifteen years that an exhaustive description of all of them is unapproachable. Hence, the aim of what follows is to supply a general perspective as the starting point of the methods proposed in Chapters 3 and 4. Feature selection methods can be categorized into three groups depending on the interaction between the feature selection process and the classifier. Figure 2.4 illustrates how the search in the feature space coexists with the learning model in each of the categories of this taxonomy: filter, wrapper and embedded techniques.

2.2.1 Filter Methods

Feature selection only considers the intrinsic properties of the data and it is performed independently of the classifier as shown in Figure 2.4(a). Many filter techniques rank features according to their *correlation* with the class label (**relevance**) ignoring possible dependences among features (**redundancy**), hence their name: **univariate filter**

methods. This way, features with highest scores are preferable and they will be used to train the final classifier. Two of the most representative metrics to score variables are described below: the Pearson correlation coefficient and mutual information.

The Pearson correlation coefficient is simple and has it has been shown to be effective in a wide variety of feature selection methods, including Correlation based Feature Selection (CFS) [25] described below and Principal Component Analysis [1]. Formally, the Pearson correlation coefficient between two random variables X_i and X_j is defined as

$$\rho_{ij} = \frac{\text{cov}(X_i, X_j)}{\sqrt{\text{var}(X_i)\text{var}(X_j)}}$$

where *cov* is the covariance of variables and *var* is the variance of each variable. The **sample correlation** is calculated as

$$\hat{\rho}_{ij} = \frac{\sum_{k=1}^{N} (\boldsymbol{x_k^i} - \bar{X}_i)(\boldsymbol{x_k^j} - \bar{X}_j)}{\sqrt{\sum_{k=1}^{N} (\boldsymbol{x_k^i} - \bar{X}_i)^2 \sum_{k=1}^{N} (\boldsymbol{x_k^j} - \bar{X}_j)^2}} \tag{2.6}$$

where, as before, N is the number of samples, $\boldsymbol{x_k^i}$ is the k-th sample of random variable X_i, and $\bar{X}_i$ is the sample mean of the random variable X_i.

Mutual Information is a nonlinear statistic which can capture nonlinear dependencies between variables unlike the Pearson correlation coefficient which only measures *linear* relationships between two random variables (see Figure 2.5). Intuitively, given two random variables X_i and X_j, mutual information measures, on average, how much information X_i and X_j share that is, how much the knowledge of one of the random variables can reduce the uncertainty about the other; by contrast, Pearson correlation

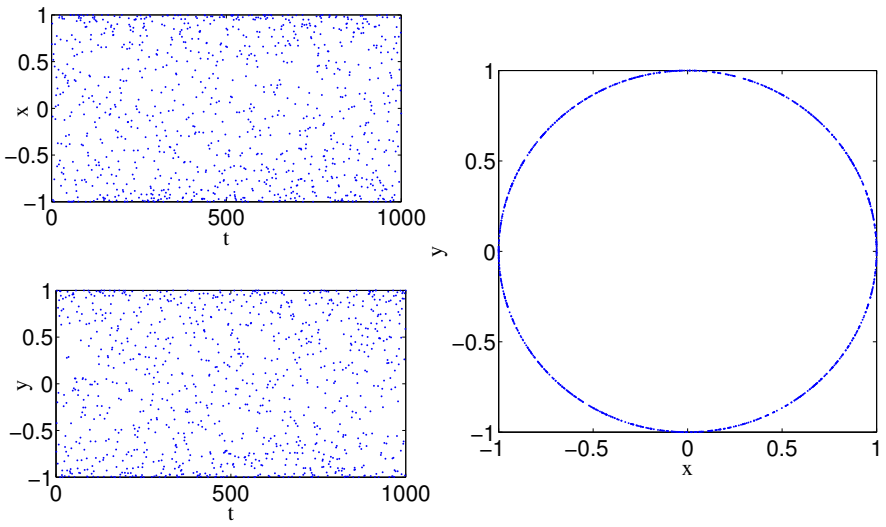


FIGURE 2.5: An example where the Pearson correlation does not capture the nonlinear relation among features. Features x and y were generated as a function of t uniformly distributed in the interval $(-1, 1)$ and according to $x = \sin(2\pi t)$ and $y = \cos(2\pi t)$. Although knowing x gives valuable information about y, the correlation between these two features is -0.014.

indicates how much variance in X_i can be explained by X_j. Formally, the mutual information between the random variables X_i and X_j is defined as

$$I(X_i; X_j) = \int \int P(X_i, X_j) \log \frac{P(X_i, X_j)}{P(X_i)P(X_j)} dX_i dX_j \ . \tag{2.7}$$

Mutual information properties are derived from the definition above:

$$\begin{aligned} I(X_i; X_j) &\geq 0 \\ I(X_i; X_j) &= I(X_j; X_i). \end{aligned}$$

If X_i and X_j are independent, knowing X_i does not provide any information about X_j being their mutual information equals to zero – this implication is also true in the other direction: mutual information is zero only for independent random variables

[26] –. Conversely, if both variables are identically distributed, knowing X_i determines completely X_j, and vice versa, and mutual information is maximum.

Computing mutual information is based on estimating the probability distributions $P(X_i)$, $P(X_j)$ and $P(X_i, X_j)$. For discrete features, Equation 2.7 can be expressed as a summation and the mutual information calculation is straightforward,

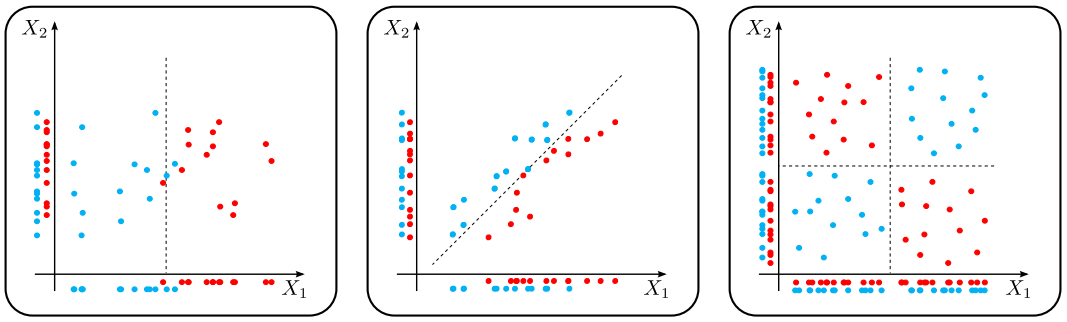
$$I(X_i; X_j) = \sum_{w_i} \sum_{w_j} P(X_i = w_i, X_j = w_j) \log \frac{P(X_i = w_i, X_j = w_j)}{P(X_i = w_i)P(X_j = w_j)}. \tag{2.8}$$

By contrast, estimating $P(X_i)$, $P(X_j)$ and $P(X_i, X_j)$ for continuous variables can be highly complicated. The continuous distributions can be discretized simplifying Equation 2.7 to Equation 2.8, but sometimes find out the appropriated discretization can be extremely costly being a possible alternative to apply other density estimation methods like maximum likelihood, bayesian estimation or non-parametric techniques [1].

Since univariate filter algorithms score variables separately from each other, they do not achieve the goal of finding combinations of variables that give the best classification performance. It has been shown that simply combining good variables does not necessary lead to good classification accuracy [8, 27, 28] namely:

- Features that are not relevant by themselves may become relevant in the presence of others.
- Features that are individually relevant may not be so useful in the presence of others because of possible redundancies.

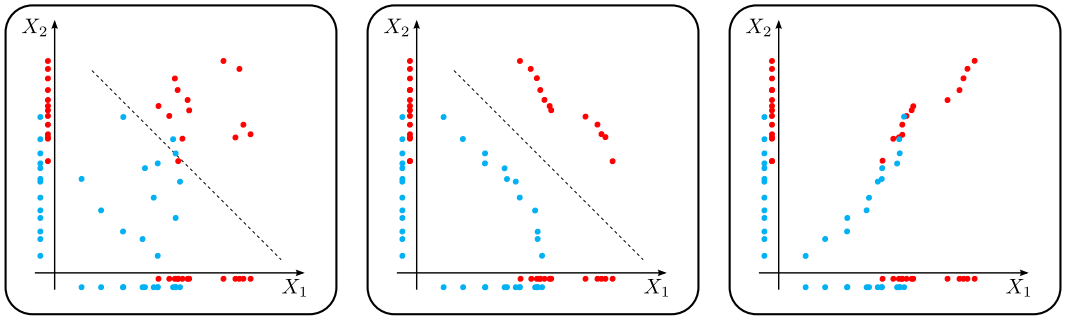
An example showing a variety of situations in which the univariate filters can fail is given in Figure 2.6 [29]. While in Figure 2.6(a) feature X_1 contains all the information to separate both classes and variable X_2 is uninformative, simply a 45 degree rotation of the input space yields Figure 2.6(d) in which both features are informative. Figure 2.6(b) shows an example in which a helpful feature may be irrelevant by itself: trying to separate the problem only taking into account the informative feature X_2 would not produce as good performance as the oblique hyperplane in the bidimensional space being



(a) Vertical separation with feature X_2 uninformative.

(b) Oblique separation with feature X_2 uninformative.

(c) Chessboard pattern. None of the projections are informative.



(d) 45 degree rotation of 2.6(a). Both features are informative.

(e) Same projections than 2.6(d).

(f) Same projections than 2.6(d).

FIGURE 2.6: Binary and bidimensional classification examples to illustrate some deficiencies of univariate filter methods. Red and blue points represents each of the classes. The projections of the classes on the axes are also shown.

the projection of feature X_1 uninformative. Going one step further, in Figure 2.6(c) the projections of both features are uninformative whereas the problem can be easily separated in the bidimensional space. Finally, Figures 2.6(e) and 2.6(f) illustrate that correlation does not imply redundancy. The projections of the classes on the axes are the same as in Figure 2.6(d) and, while in Figure 2.6(f) the features are correlated and are indeed redundant (the elimination of one of the features does not imply a worsening in the classification accuracy), in Figure 2.6(e) both features are anti-correlated but needed for a perfect separation.

One common improvement direction for univariate filter algorithms is to consider dependencies among variables leading to **multivariate filter methods**. These approaches tend to achieve better classification results than univariate solutions because their assumptions about feature independence are not so strict. Some of the most popular algorithms falling into this group are described in what follows.

Correlation based Feature Selection was proposed by Hall [25] as a method to rank feature subsets rather than individual features. The heuristic is based on the hypothesis that good feature subsets are those consisting of features highly correlated with the class, yet uncorrelated with each other. The heuristic for a subset S of size $\tilde{M}$ can be formalized as,

$$\text{Merit}_S = \frac{\tilde{M}\ \bar{\rho}_{cf}}{\sqrt{\tilde{M} + \tilde{M}(\tilde{M}-1)\bar{\rho}_{ff}}} \tag{2.9}$$

where $\bar{\rho}_{cf}$ is the average feature-class correlation and $\bar{\rho}_{ff}$ is the average feature-feature correlation. The numerator in Equation 2.9 can be interpreted as the relevance term whereas the denominator represents the redundancy factor. Although the relevance class-feature is measured independently for each feature, it has been shown that CFS can identify useful subsets of features with moderate levels of redundancy.

A crucial point of the CFS algorithm is how to select the candidate subsets to be the optimal. In view of the impossibility to evaluate all the possible 2^n subsets, CFS searches in the feature subset space using a best first search starting from an empty set of features. At each step of the algorithm, all possible single features expansions are considered and that with the highest evaluation (Equation 2.9) is chosen. If at some point the expansion does not entail any improvement, the search draws back and the next best subset is expanded. Obviously, the best first search will examine all the possibilities if no criterion is imposed to limit the search, thus in practice the number of returns is bounded by a parameter of the algorithm.

Streamwise Feature Selection (SFS) [30] selects a feature if the *benefit* of adding it to the model is greater than the increase in the model complexity. SFS can be used in a statistical setting using a t or F statistic yielding the α-investing procedure. Intuitively, this approach dynamically modifies the threshold on the error reduction required to add a new feature in order to reduce the risk of overfitting. Mathematically, this idea is implemented by controlling the False Discovery Rate (FDR) that is, the fraction of features that, when added to the model, reduce its predictive accuracy. This way,

α-investing adds as many features as possible subject to the FDR bound giving the minimum out-of-sample error. One of the main advantages of this algorithm is that it considers features sequentially for addition to a model, making unnecessary to know all the features in advance. Furthermore, as it will be seen in the experimental section of Chapter 3, it is able to reduce the dimension of the problem drastically while getting acceptable accuracies.

minimum Redundancy Maximum Relevance criterion (mRMR) [31] has been a significant advance in multivariate feature selection methods based on mutual information. The aim of any feature selection algorithm based on mutual information is to find a subset of features $\tilde{X} = \{\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_{\tilde{M}}\}$ of size $\tilde{M} < M$ which jointly have the largest dependency on the target class Y:

$$\max D(\tilde{X}, Y) \quad , \quad D = I\left(\left\{\tilde{X}_i, i = 1, \ldots, \tilde{M}\right\}; Y\right).$$

This criterion, **so-called Maximal Dependency (MaxDep)**, is theoretically optimum but it is often unfeasible to compute the joint density functions of all features and of all features with the class since (i) the number of training samples is usually not enough and (ii) the high computational load. Therefore, the use of MaxDep is restricted to problems with many patterns and few dimensions.

One alternative to making the MaxDep approach practical is **Maximal Relevance (MaxRel)** feature selection [31]. This approach, categorized into the univariate filer methods, selects those features that have the highest relevance (mutual information) with the target class. As explained above, the main drawback of univariate filters is not accounting for redundancy among features. The mRMR criterion overcomes this limitation by choosing a subset of features $\tilde{X}$ with both, minimum redundancy (approximated as the mean value of the mutual information between each pair of variables in the subset) and maximum relevance (estimated as the mean value of the mutual information between each feature and the target class):

$$\max_{\tilde{X}} \left\{ \frac{1}{|\tilde{X}|} \sum_{\tilde{X}_i \in \tilde{X}} I(X_i; Y) - \frac{1}{|\tilde{X}|^2} \sum_{\tilde{X}_i, \tilde{X}_j \in \tilde{X}} I(\tilde{X}_i; \tilde{X}_j) \right\}. \tag{2.10}$$

Again, the evaluation of Equation 2.10 for all the possible subsets of features is unfeasible. The mRMR authors propose a greedy algorithm that, assuming that the optimal subset of size $m-1$, $\tilde{X}_{m-1}$, has be found, forms the subset of size m, $\tilde{X}_m$, according to

$$\max_{X_j \in X \setminus \tilde{X}_{m-1}} \left\{ I(X_j; Y) - \frac{1}{m-1} \sum_{\tilde{X}_i \in \tilde{X}_{m-1}} I(\tilde{X}_i; X_j) \right\}.$$

In the first iteration of the algorithm, the feature with the highest relevance with the class is chosen. The previous formulation of the mRMR criterion is also known as the **Mutual Information Difference (MID)** and it is the most used in practice. Nevertheless, the quotient of the relevance and redundancy term, so-called **Mutual Information Quotient (MIQ)**, has also been successfully applied to microarray expression data [20].

ReliefF [32] evaluates the quality of a feature according to how well it distinguishes between instances that are near to each other. Specifically, the algorithm starts by identifying the K closest samples of the same class (nearest hits) and the K closest samples of a different class (nearest misses) of each training pattern $\boldsymbol{x_i}$. Then, only considering the j-th feature, the sum of distances between the examples and their nearest misses is compared to the sum of distances of their nearest hits as follows,

$$C(X_j) = \frac{\sum_{i=1}^{N} \sum_{k=1}^{K} |x_i^j - x_{M_k(i)}^j|}{\sum_{i=1}^{N} \sum_{k=1}^{K} |x_i^j - x_{H_k(i)}^j|}$$

where $M_k(i)$ and $H_k(i)$ are the k-the nearest miss and hit of pattern $\boldsymbol{x_i}$, respectively. Therefore, features with the highest scores are those that keep close samples in the same class while they get away from samples of different classes.

2.2.2 Wrapper Methods

Wrappers use search techniques to select candidate subsets of variables and evaluate their fitness based on classification accuracy as presented in Figure 2.4(b). Wrapper framework requires to specify how the search in the space of all possible feature subsets is conducted, the underlying classifier and how the misclassification error will be used to guide the search. Wrapper methods are usually criticized because they seem a brute-force approaches involving high computational load though several search strategies have been proposed to reduce their complexity; best-first [33], simulated annealing [34] or genetic algorithms [35, 36] among others. Regardless of the algorithm, the search can be performed either adding iteratively the most relevant features (**forward**) to an initially empty subset, or taking the whole set of features as starting point and removing sequentially those variables less informative (**backward**).

2.2.3 Embedded Methods

Feature selection is accomplished together with the training of the classifier by incorporating the feature selection step in the classifier's objective function (or algorithm) as illustrated in Figure 2.4(c). Examples per excellence of embedded feature selection techniques are decision trees with axis-parallel splits (Section 2.3.3). However, one of the simplest and more efficient methods in line with embedded methods and Support Vector Machines, studied more in depth in Section 2.3.2, is the Recursive Feature Elimination Support Vector Machine algorithm (RFE-SVM) proposed by Guyon et al. [37] and successfully applied to problems of gene selection for microarray data [37–39].

Recursive Feature Elimination Support Vector Machine. Given a classification problem in a M-dimensional space, the number of final dimensions $\tilde{M} < M$ should be specified at the beginning of the algorithm. The aim of the algorithm is to choose the

subset of $\tilde{M}$ features which leads to the largest margin in the SVM classifier. The search among all the possible subsets of size $\tilde{M}$ is a combinatorial problem extremely costly, especially when M is high. Thus, the RFE-SVM method greedily constructs the subset of size $\tilde{M}$ using a backward procedure: at each iteration in the SVM training, the input dimension that reduces the margin the least is removed until only $\tilde{M}$ dimensions remain. A pseudocode of this method is presented in Algorithm 1. An alternative to speed up the execution time of the algorithm is to remove more than one feature at each iteration (step 4). A study of the SVMs model reveals that, while finding out the feature reducing the least the margin is straightforward for linear SVMs, this discovery is not so easy in the nonlinear case being necessary some simplifications about the SVM model [37].

Algorithm 1 Recursive Feature Elimination Support Vector Machine.

1: **Inputs**: Subset of training patterns $\mathcal{S}$ in a M-dimensional space, new dimension $\tilde{M}$.
2: **repeat**
3: Training a linear/nonlinear SVM.
4: Remove the feature that reduces the margin the least.
5: **until** $\tilde{M}$ features are left
6: **Outputs**: subset of $\tilde{M}$ features.

The strengths and weaknesses of each of the three feature selection approaches are summarized in Table 2.1 adapted from [21]. These characteristics can be decisive when feature selection wants to be applied to certain domains. Among large-scale and high-dimensionality problems, filter approaches usually are preferred due to their simplicity and good scalability. In particular, univariate strategies seem the perfect candidate for large-scale problems and, in fact, this is true but their little competitiveness in terms of classification accuracy makes it necessary to resort to multivariate filters. The algorithm proposed in Chapter 3 is in line with multivariate filter techniques and it is especially suitable for large-scale domains.

2.3 Learning Algorithms

The diversity of classification models with different assumptions about the characteristics of the data can be kept in the **non free lunch theorem** context. Quoting Ho and

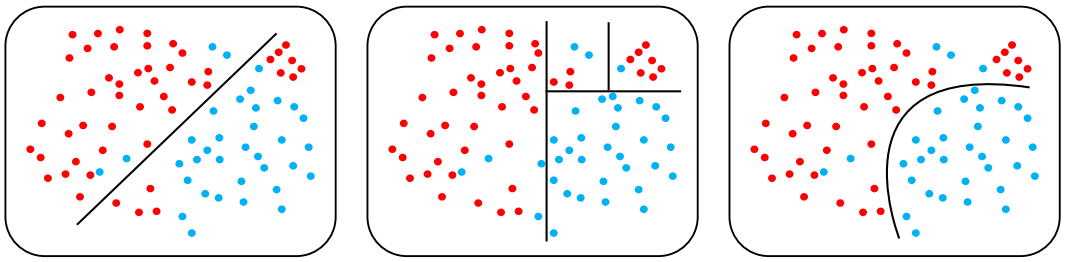
Group	Advantages	Disadvantages
Filter univariate	· Fast · Scalable · Independent of the classifier	· Ignores feature dependences · Ignores interaction with the classifier
Filter multivariate	· Models feature dependences · Independent of the classifier · Lower computational complexity than wrapper methods	· Slower than univariate techniques · Less scalable than univariate techniques · Ignores interaction with the classifier
Wrapper	· Interacts with the classifier · Models feature dependencies	· Risk of overfitting · Classifier dependent · Computationally expensive
Embedded	· Interacts with the classifier · Lower computational complexity than wrapper methods · Models feature dependencies	· Classifier dependent

TABLE 2.1: Main advantages and disadvantages of the different feature selection approaches.

Pepyne: *"A general-purpose universal optimization strategy is theoretically impossible, and the only way one strategy can outperform another is if it is specialized to the specific problem under consideration"*[3]. A common strategy used in practice is to choose the best classifier in terms of classification accuracy among a set of candidates. Other features such as simplicity or interpretability can be also taking into account in order to evaluate the goodness of the model as discussed in Chapter 1.

A taxonomy of the classification models can be formed as a function of the kind of borders used to distinguish different regions in the input space, also known as **decision boundaries**. Figure 2.7 shows three of the most popular decision boundaries: linear, axis-parallel splitting and nonlinear. According to this categorization, it is said that a classification problem is **linearly separable** if it can be completely separated by a hyperplane, being **nonlinearly separable** otherwise. While the classification borders

[3] "Simple Explanation of the No-Free-Lunch Theorem and Its Implications", 2002.



(a) (b) (c)

FIGURE 2.7: An example of different decision boundaries for a binary classification task. Figure 2.7(a) linear; Figure 2.7(b) axis-parallel split; Figure 2.7(c) nonlinear.

parallel to the axes of the feature space are characteristic of decision trees, many different learning algorithms yield linear and nonlinear decision boundaries. A description of all these methods is beyond the scope of this thesis and therefore, only a brief analysis of those classifiers used throughout this work are presented namely: Fisher Linear Discriminant (Section 2.3.1), Support Vector Machines (Section 2.3.2) and decision trees (Section 2.3.3).

For simplicity, in what follows the term of classification problem will refer to binary classification problems ($\mathcal{Y} = \{-1, 1\}$) although the Fisher Linear Discriminant and decision trees admit a direct multiclass formulation. Multiclass Support Vector Machines can be implemented following the *one-against-others* [40] or the *one-against-one* [41] strategies.

2.3.1 Fisher Discriminant Analysis

Fisher Discriminant Analysis belongs to the family of Linear discriminants, which produce linear decision boundaries by modeling the output y of the classifier as a linear combination of the input patterns $\boldsymbol{x}$,

$$\hat{y} = f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b \tag{2.11}$$

being $\boldsymbol{w}$ the **weight vector** and $(-b)$ the **threshold** or **bias**. This way, the classification criterion divides the space into two regions corresponding to the side of the hyperplane in which the pattern $\boldsymbol{x}$ falls. That is, if $f(\boldsymbol{x}) \geq 0$ the pattern $\boldsymbol{x}$ is assigned to the positive class, being labeled as negative otherwise. The linear discriminant function defined by

$\boldsymbol{w}$ and b can be determined in different ways. In particular, the Fisher Discriminant Analysis (also known as **Fisher Linear Discriminant Analysis**) tries to maximize the separation between the projected means of each class, $\tilde{\boldsymbol{\mu}_+}$ and $\tilde{\boldsymbol{\mu}}_-$, while minimizing the variance within each projected class, $\tilde{s}_+^2$ and $\tilde{s}_-^2$. This idea can be formally expressed as,

$$\max J(\boldsymbol{w}) = \frac{\|\tilde{\boldsymbol{\mu}}_+ - \tilde{\boldsymbol{\mu}}_-\|^2}{\tilde{s}_+^2 + \tilde{s}_-^2}, \tag{2.12}$$

where the means, the projected means and the variance within each projected class are defined as,

$$\begin{aligned}
\boldsymbol{\mu}_+ &= \tfrac{1}{N_+} \textstyle\sum_{\boldsymbol{x}\in S_+} \boldsymbol{x} & \boldsymbol{\mu}_- &= \tfrac{1}{N_-} \textstyle\sum_{\boldsymbol{x}\in S_-} \boldsymbol{x} \\
\tilde{\boldsymbol{\mu}}_+ &= \tfrac{1}{N_+} \textstyle\sum_{\boldsymbol{x}\in S_+} \boldsymbol{w}\cdot\boldsymbol{x} & \tilde{\boldsymbol{\mu}}_- &= \tfrac{1}{N_-} \textstyle\sum_{\boldsymbol{x}\in S_-} \boldsymbol{w}\cdot\boldsymbol{x} \\
\tilde{\boldsymbol{s}}_+^{\boldsymbol{2}} &= \tfrac{1}{N_+} \textstyle\sum_{\boldsymbol{x}\in S_+} (\boldsymbol{w}\cdot\boldsymbol{x} - \tilde{\boldsymbol{\mu}}_+)^2 & \tilde{\boldsymbol{s}}_-^{\boldsymbol{2}} &= \tfrac{1}{N_-} \textstyle\sum_{\boldsymbol{x}\in S_-} (\boldsymbol{w}\cdot\boldsymbol{x} - \tilde{\boldsymbol{\mu}}_-)^2
\end{aligned}$$

being S_+ and S_- the subsets of training patterns corresponding to the positive and negative labels, respectively. Writing Equation 2.12 in terms of $\boldsymbol{w}$,

$$\max J(\boldsymbol{w}) = \frac{\boldsymbol{w}^T S_B \boldsymbol{w}}{\boldsymbol{w}^T S_W \boldsymbol{w}} \tag{2.13}$$

being the matrices S_B and S_W the Between and Within scatter matrices:

$$S_B = (\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)^T \tag{2.14}$$

$$S_W = \sum_{\boldsymbol{x}\in S_+} (\boldsymbol{x} - \boldsymbol{\mu}_+)(\boldsymbol{x} - \boldsymbol{\mu}_+)^T + \sum_{\boldsymbol{x}\in S_-} (\boldsymbol{x} - \boldsymbol{\mu}_-)(\boldsymbol{x} - \boldsymbol{\mu}_-)^T. \tag{2.15}$$

It can be shown that the optimal solution $\boldsymbol{w}^{\text{FDA}}$ of Equation 2.13 is obtained either as the leading eigenvector of $S_W^{-1} S_B$ or in a closed form as $\boldsymbol{w}^{\text{FDA}} = S_W^{-1}(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$ [42].

As the rank of matrix S_B is at most one, the direction $\boldsymbol{w}^{\text{FDA}}$ projects the data into a 1-dimensional space.

Up to this point, the Fisher direction $\boldsymbol{w}^{\text{FDA}}$ can be interpreted as a way to reduce the dimensionality of the original space but it does not represent a discriminatory function by itself since Fisher Discriminant's assumption is that the discriminatory information is in the mean of the data, the decision criterion consists in, once the unlabeled pattern has been projected, assigning the class of the closest projected mean.

A remarkable fact about the Fisher Discriminant is that it is the Bayes' optimal solution if the two classes are gaussian distributions with equal covariance matrix [42].

2.3.2 Support Vector Machines (SVMs)

Support Vector Machines have become one of the most popular classifiers nowadays due to their good performance in many domains besides their solid mathematical basis. Introduced by Boser et al. [43], SVMs were initially designed to solve binary classification problems but these days the SVM *framewok* is widespread in domains like novelty detection [44], clustering [45] or feature selection and extraction [44], among others. Indeed, Chapter 4 deals with the adaptation of classical feature selection and extraction methods like PCA [1] or the FDA (Section 2.3.1) to the SVM structure.

Conceptually, SVMs are linear classifiers acting over special features and based on the statistical learning theory addressed in Section 2.1. As linear model, the understanding of the SVM's grounds requires to go back to the perceptron model introduced by Rosenblatt [1]. Thought based on the imitation of the synaptic behavior of a neuron, from the machine learning perspective the perceptron attempts to find an hyperplane that perfectly separates all the points in the training set. Therefore, following a linear model $f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$ and assuming that the patterns are labeled as $+1$ or -1, zero classification error rate implies,

$$y_i \left(\boldsymbol{w} \cdot \boldsymbol{x_i} + b\right) \geq 0 \ \ \forall i.$$

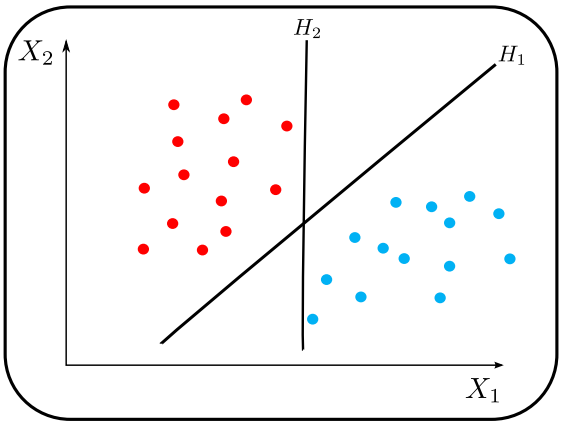


FIGURE 2.8: An example of two separating hyperplanes valid for the perceptron model.

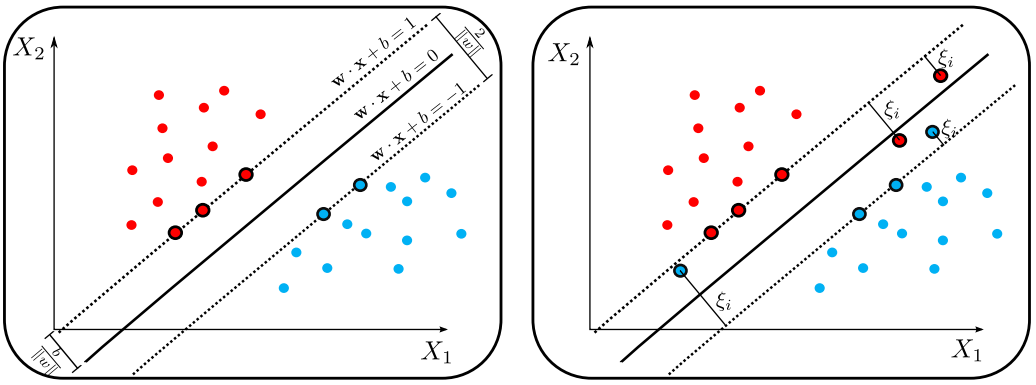
The derivation of SVMs comes from the observation that two (or more) hyperplanes separating correctly all the training points are equally good for the perceptron criterion, not matter how they divide the input space. However, in practice not all the models have the same generalization capability when applied to out-of-training samples. A clear example of this situation is presented in Figure 2.8 in which both hyperplanes, H_1 and H_2, classify correctly all the training samples but H_1 seems more stable than H_2: the presence of noise in the training data or even small changes in the distribution of unseen patterns can provoke a degradation in the test performance of H_2 as there are some training samples close to the hyperplane. On the contrary, the hyperplane H_1 is more robust in these circumstances.

Then, it is reasonable to define the optimal hyperplane as the one separating the classes with maximal distance between the hyperplane and the nearest data point. This distance is known as **margin** and can be written as,

$$m = \min_i \frac{|\boldsymbol{w} \cdot \boldsymbol{x_i} + b|}{\|\boldsymbol{w}\|_2} \quad i = 1, \ldots, N.$$

Assuming a linearly separable problem, above equation is equivalent to

$$m = \min_i \frac{y_i \left(\boldsymbol{w} \cdot \boldsymbol{x_i} + b\right)}{\|\boldsymbol{w}\|_2} \quad i = 1, \ldots, N,$$



(a) (b)

FIGURE 2.9: Figure 2.9(a) Maximum margin hyperplane for a binary classification problems. Figure 2.9(b) Maximum margin hyperplane and slack variables for the soft SVM formulation.

and the idea of maximizing the margin can be expressed as **max-min optimization problem**,

$$\max_{\boldsymbol{w},b} \min_{i} \frac{y_i\left(\boldsymbol{w}\cdot\boldsymbol{x_i}+b\right)}{\|\boldsymbol{w}\|_2} \quad i=1,\ldots,N. \tag{2.16}$$

Without loss of generality and to ensure uniqueness, $\boldsymbol{w}$ and b can be rescaled enforcing that the closest point to the optimal hyperplane lies in one of the hyperplanes $\boldsymbol{w}\cdot\boldsymbol{x}+b=\pm 1$. Consequently, every training point $\boldsymbol{x_i}$ verifies $y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)\geq 1$ and the margin is equal to $\frac{2}{\|\boldsymbol{w}\|_2}$; that is, maximizing the margin means to minimize $\|\boldsymbol{w}\|_2$. Under these assumptions, Equation 2.16 can be restated as a more tractable optimization problem:

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 \tag{2.17}$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)\geq 1 \quad i=1,\ldots,N. \tag{2.18}$$

Figure 2.9(a) shows an example of the maximal margin hyperplane obtained by the SVM model. Training points on the margin have been stressed in the illustration. These samples, called **support vectors**, are especially relevant for the SVM model because they define the separating hyperplane and they are the most difficult to classify.

Thus far, only linearly separable problems have been tackled and strict margin conditions have been imposed, restricting the applicability of the model to non-realistic problems

as the one presented in Figure 2.9(a).

In those cases in which the underlying model is linear but the training set is disturbed by some noise, the above model fails. The incorporation of certain flexibility in the margin requirements was suggested by Cortes and Vapnik [46] to alleviate this problem. Each training pattern $\boldsymbol{x_i}$ has associated a scalar variable ξ_i, called **slack variable**, which, in a nutshell, quantifies how much $\boldsymbol{x_i}$ violates the margin requirement (see Figure 2.9(b)). The model proposed by Cortes and Vapnik is known as **soft margin SVM** in contrast to the **hard margin SVM** stated by Equation 2.18. Soft margin SVM can be formalized as,

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i \tag{2.19}$$

$$\text{s.t.} \quad y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b) \geq 1-\xi_i \tag{2.20}$$

$$\xi_i \geq 0 \quad i=1,\ldots,N. \tag{2.21}$$

The above equation can be compacted as

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N}|1-y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)|_+ \tag{2.22}$$

where $|1-y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)|_+$ takes the value of $1-y_i(\boldsymbol{w}\cdot\boldsymbol{x_i}+b)$ only when it is positive and zero otherwise.

The presence of the slack variables provides room to handle the noisy data but only up to point. Moreover, the new parameter $C \in [0,\infty)$ establishes a trade-off between the two objectives of the model: maximizing the margin while classifying correctly as many training patterns as possible. Large values of C focus the attention on reducing the violations whereas small values of C overweight the large-margin to the detriment of the classification effectiveness. Obviously, these extreme values of C can be identified, respectively, with the overfitting and underfitting phenomena. To conclude, it is worth noting how Equation 2.22 matches with the regularization risk minimization raised in Section 2.1.2 (Equation 2.5).

Finally, learning the SVM model set out in Equation 2.21 requires, inevitably, to solve an optimization problem. From this point of view, one of the most attractive properties of the SVM problem is that it can be categorized into the group of **convex optimization problems** widely studied in the literature [47, 48]. The convexity of the SVM formulation, derived from the convexity of the objective function and the linearity of the constraints, allows not only to guarantee the existence of a global minimum but also to solve an alternative problem in the dual space [4]. To do so, the Lagrangian is constructed,

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N}\xi_i + \sum_{i=1}^{N}\alpha_i\left(1 - \xi_i - y_i(\boldsymbol{w}\cdot\boldsymbol{x_i} + b)\right) - \sum_{i=1}^{N}\beta_i\xi_i, \quad (2.23)$$

where $\alpha_i \geq 0$ and $\beta_i \geq 0$ are the Lagrange multipliers. The optimal primal variables $(\boldsymbol{w^*}, b^*, \boldsymbol{\xi^*})$ minimize the Lagrangian by making their gradients equals to zero,

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w^*} - \sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x_i} = 0 \quad (2.24)$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{N}\alpha_i y_i = 0 \quad (2.25)$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \beta_i = 0. \quad (2.26)$$

After some substitutions of Equations 2.24-2.26 into Equation 2.23, the SVM optimization problem is transformed to the following convex dual optimization problem,

$$\max_{\boldsymbol{\alpha}} \quad -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{x_i}\cdot\boldsymbol{x_j} + \sum_{i=1}^{N}\alpha_i \quad (2.27)$$

$$\text{s.t.} \quad \begin{cases} 0 \leq \alpha_i \leq C & \forall i \\ \sum_{i=1}^{N}\alpha_i y_i = 0 \end{cases}.$$

And the primal variables $\boldsymbol{w^*}$ and b^* can be worked out

[4] The optimization notions laying behind these results are out of the scope of this thesis, a detailed review of convexity optimization theory can be found in [49].

$$\boldsymbol{w^*} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x_i}, \tag{2.28}$$

$$b^* = (y_i - \boldsymbol{w^*} \cdot \boldsymbol{x_i}) \text{ for some } \boldsymbol{x_i}, \tag{2.29}$$

where $\boldsymbol{w^*}$ has been directly obtained from Equation 2.24 and b^* is recovered by observing that α_i's greater than zero are precisely the support vectors lying in the margins $y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b^*) = 1$. Any pattern $\boldsymbol{x_i}$ with $\alpha_i > 0$ can be used to compute b^*, though in practice the average over the support vectors is considered: $b^* = \frac{1}{\#\{\alpha_i > 0\}} \sum_{\alpha_i > 0} (y_i - \boldsymbol{w^*} \cdot \boldsymbol{x_i})$.

2.3.2.1 Introducing Kernels: The Kernel Trick

The soft margin SVMs have solved the problem of noisy linear datasets but it is still a linear model that probably fails when the data is nonlinearly separable by nature. As solution, Boser et al. suggested a way to extend the SVM framework to nonlinear classifiers with slight modifications into the linear SVM formulation via the **kernel trick**. Intuitively, the kernel trick maps the input samples into a higher-dimensional space where the maximal separating hyperplane described above is calculated. Therefore, linear classification in the new space may be equivalent to non-linear classification in the original space if the mapping function is nonlinear. A naive example of how the kernel trick can work in non-linearly separable problems is presented in Figure 2.10.

The general scheme of the kernel trick applied to SVMs is shown in Figure 2.11 (adapted from [50]) and it consists of two steps:

1. A non-linear mapping $\Phi(x)$ from the input space $\mathcal{X}$ to the high dimensional feature space $\mathcal{F}$ that is hidden for both sides (input and output).

2. The construction of the large-margin hyperplane in the feature space $\mathcal{F}$.

The theoretical justification of the first step finds its place in the **Cover's theorem**[5] and, the second step is based on the structural risk minimization of SVMs which plays

[5] *"A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated."* Cover, T.M. , Geometrical and Statistical properties of systems of linear inequalities with applications in pattern recognition, 1965.

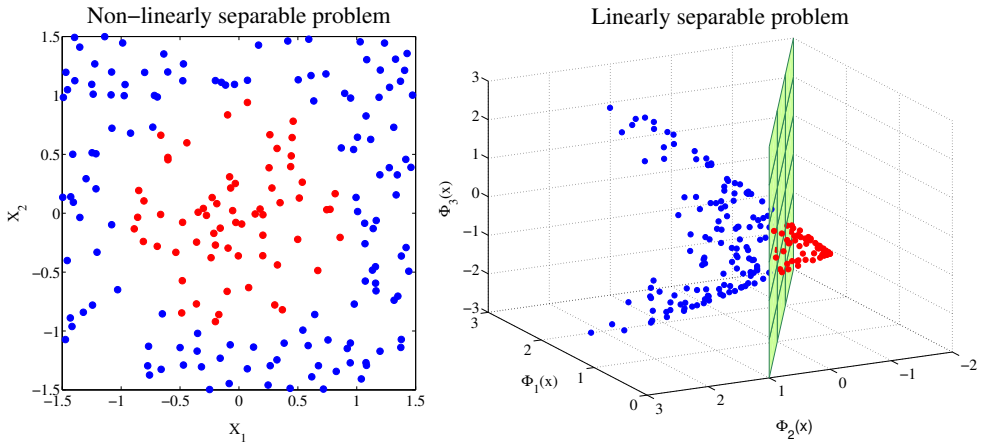


FIGURE 2.10: An example of how mapping the nonlinear data into a higher dimensional space can make the problem linearly separable. The original problem (left) is a bidimensional problem in which the positive class is formed by those points verifying $X_1^2 + X_2^2 \leq 1$ and the negative class are the points located outside the circle. This problem is clearly nonlinear but, if the two dimensional input space defined by (X_1, X_2) is transformed to the three-dimensional space given by $\Phi_1(\boldsymbol{x}) = X_1^2, \Phi_2(\boldsymbol{x}) = X_2^2, \Phi_3(\boldsymbol{x}) = \sqrt{2}X_1X_2)$, the problem turns out to be linearly separable in the larger space (right).

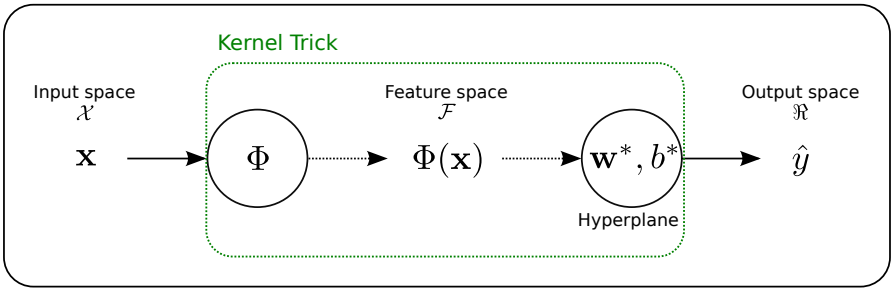


FIGURE 2.11: Scheme of the kernel trick applied to the SVM problem.

a vital role in this scheme to avoid overfitting in those cases in which the dimension of the feature space is extremely high or even infinite.

The main advantage of the kenel trick is that there is no need to compute the mapping function Φ explicitly but it requires to define the inner product in the feature space $\mathcal{F}$ by means of a **kernel function** K. Given two patterns $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ in the input space $\mathcal{X}$, the kernel function K of $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ is defined as,

$$\begin{aligned} K_{ij} &: \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R} \\ K_{ij} &= K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \Phi(\boldsymbol{x_i}) \cdot \Phi(\boldsymbol{x_j}). \end{aligned}$$

To guarantee the effectiveness of the kernel trick, K_{ij} has to be expressed in terms of $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ disregarding the implicit dependence on Φ. In addition, the kernel function must satisfy the Mercer's theorem conditions to guarantee that it defines an inner product in $\mathcal{F}$ [51]. In practice, it is not necessary to have a depth knowledge about the mathematical grounds around Mercer's theorem, it is enough to assure that the **kernel matrix** K formed by the inner product between every pair of training samples

$$K = \begin{pmatrix} K_{11} & K_{12} & \dots & K_{1N} \\ K_{21} & K_{22} & \dots & K_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ K_{N1} & K_{N2} & \dots & K_{NN} \end{pmatrix}$$

is positive semi-definite and its decomposability into an inner product of mappings is supported by the Mercer's theorem.

Going into details, the kernel trick can be straightforwardly applied to SVMs observing the dual optimization problem raised in Equation 2.27: the dot product $\boldsymbol{x_i} \cdot \boldsymbol{x_j}$ can be replaced by a kernel function K_{ij}. The new dual optimization problem is,

$$\max_{\boldsymbol{\alpha}} \quad -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K_{ij} + \sum_{i=1}^{N}\alpha_i \tag{2.30}$$

with the same box constraints. Now the optimal hyperplane $\boldsymbol{w^*}$ can be expressed as,

$$\boldsymbol{w^*} \quad = \quad \sum_{i=1}^{N}\alpha_i y_i \Phi(\boldsymbol{x_i}). \tag{2.31}$$

Unfortunately, as the solution $\boldsymbol{w^*}$ depends on the generally unknown mapping function $\Phi(\boldsymbol{x_i})$ it can not be explicitly computed as in the linear model. In any case, what it is really important is the decision function and it can be expressed only in terms of inner products in the feature space:

Kernel Function	**Parameters**	**Formula**
Linear	None	$x_i \cdot x_j$
Polynomial	d: polynomial degree	$(x_i \cdot x_j)^d$
Gaussian	σ: kernel width	$\exp\left\{-\frac{\|x_i - x_j\|_2^2}{2\sigma^2}\right\}$

TABLE 2.2: Most used kernel functions.

$$\begin{aligned} \hat{y} = f(\boldsymbol{x}) &= \operatorname{sgn}\left(\left(\sum_{i=1}^{N} \alpha_i y_i \Phi(\boldsymbol{x_i})\right) \cdot \Phi(\boldsymbol{x}) + b^*\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{N} \alpha_i y_i K(\boldsymbol{x_i}, \boldsymbol{x}) + b^*\right). \end{aligned} \tag{2.32}$$

At this moment, it is worth mentioning that the kernel trick can be similarly applied to any learning algorithm which can be expressed in terms of dot products in the feature space. The framework in Figure 2.11 holds replacing the second step corresponding to the large-margin hyperplane by the method to be kernelized.

The most known kernel functions are given in Table 2.2. Certainly, the decision border of the SVM classifier depends on the parameters used but, especially, on the kernel type. The boundary can be linear as in Figure 2.7(a) or highly nonlinear as presented in Figure 2.7(c) thus, a careful search of the proper SVMs model parameters must be accomplished in the model selection phase (Figure 1.1). Of course, the use of the linear kernel yields the linear soft margin SVMs introduced in Section 2.3.2, but in practice the Gaussian kernel is the most used function due to its versatility.

2.3.2.2 SVM solvers

It has been seen how the SVM model can be formulated as a convex optimization problem, either in the primal or in the dual space. Each of these approaches entails certain advantages and drawbacks that should be considered when working with a particular problem. On the one hand, the dual problem has simplified the constraints of the primal version but, on the other hand, it has made the quadratic term of the objective function more complicated.

Although SVM classifiers will be used in Chapters 3 and 5, the underlying optimization solver is irrelevant for the contents developed throughout this thesis. In Chapter 3 a linear SVM will be used as a black box receiving samples and returning the predicted classification label after applying the proposed feature selection algorithm. Regarding the algorithm presented in Chapter 5, the understanding of its motivation as well as the interpretation of its results only require certain notions about the SVM model expounded in the preceding sections. Therefore, the standard software for SVMs implementing two of the state-of-the-art methods will be used, namely: the **LIBLINEAR library** [52] for linear SVMs and the **LIBSVM package** [53] for nonlinear kernels. Both algorithms solve the dual optimization problem, LIBLINEAR through dual coordinate descent [54] and LIBSVM uses the Sequential Minimal Optimization (SMO) strategy proposed by Platt [55] and consisting of the decomposition of the quadratic problem in smaller subproblems which can be efficiently solved and pieced together. Furthermore, the LIBSVM package incorporates some improvements to reduce the number of kernel evaluations and to accelerate the convergence. These binary SVM solvers have also been adapted to multiclass classification problems using the *one-against-others* strategy [40] in the case of LIBLINEAR and the *one-against-one* scheme [41] in the case of LIBSVM.

Another outstanding algorithm appearing in the last years and resulting very efficient for large-scale data is the **Pegasos algorithm** introduced by Shalev-Schwartz et al. [56] which solves the linear SVM problem in the primal space. It is one of the keystones of the algorithm suggested in Chapter 5 and more details about it are given in what follows. The Pegasos algorithm is an efficient and practical method for training linear SVMs in large-scale datasets via **stochastic subgradient descent (SGD)** method. The Pegasos algorithm minimizes the objective function of a linear SVM in the primal space,

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{N}\sum_{(\boldsymbol{x},y)\in S} \mathcal{L}\left(\boldsymbol{w};(\boldsymbol{x},y)\right) \tag{2.33}$$

where $\mathcal{L}\left(\boldsymbol{w};(\boldsymbol{x},y)\right)$ represents the loss function previously used in Equation 2.22,

$$\mathcal{L}\left(\boldsymbol{w};(\boldsymbol{x},y)\right) = |1 - y_i(\boldsymbol{w}\cdot\boldsymbol{x_i})|_+. \tag{2.34}$$

Note that in the objective function of Pegasos the parameter C –used by LIBLINEAR and LIBSVM software– has been replaced by λ; the equivalence between these two parameters is immediate: $C = \frac{1}{\lambda N}$. To solve the problem in Equations 2.33 and 2.34, the Pegasos algorithm given in Algorithm 2 alternates between stochastic gradient descent steps and projection steps:

- **Stochastic gradient descent**. On iteration t of the algorithm, a set $A_t \subset S$ of size k is chosen. Then, the objective function 2.33 is approximated by,

$$\min_{\boldsymbol{w}} f(\boldsymbol{w};A_t) = \min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{k}\sum_{(\boldsymbol{x},y)\in A_t} \mathcal{L}\left(\boldsymbol{w};(\boldsymbol{x},y).\right) \tag{2.35}$$

  The update of the $\boldsymbol{w}$ based on the gradient descent method is given by $\boldsymbol{w_{t+\frac{1}{2}}} = \boldsymbol{w_t} - \eta_t \boldsymbol{\nabla_t^w}$, where $\eta_t = \frac{1}{\lambda t}$ is the learning-rate and $\boldsymbol{\nabla_t^w}$ is the subgradient of $f(\boldsymbol{w};A_t)$ with respect to $\boldsymbol{w}$ on the iteration t,

$$\boldsymbol{\nabla_t^w} \;=\; \lambda \boldsymbol{w_t} - \frac{1}{|A_t|}\sum_{(\boldsymbol{x},y)\in A_t^+} y\boldsymbol{x} \tag{2.36}$$

  being A_t^+ the set of samples in A_t with non-zero loss that is, $A_t^+ = \{(\boldsymbol{x},y)\in A_t \mid y(\boldsymbol{w_t}\cdot\boldsymbol{x}) < 1\}$.

- **Projection step**. Projection of $\boldsymbol{w_{t+\frac{1}{2}}}$ onto the set $B = \left\{\boldsymbol{w} \mid \|\boldsymbol{w}\| \leq \frac{1}{\sqrt{\lambda}}\right\}$ since it can be shown that the optimal solution of SVM is in the set B [56].

Finally, the bias term can be estimated following one of the three different approaches suggested by the Pegasos' authors:

1. **Add one more feature to each pattern taking always the same value** (generally 1). Although this approach does not require any modifications in Algorithm 2, it is solving a slightly different optimization problem that also regularizes the bias term.

Algorithm 2 The Pegasos Algorithm.

Inputs: S, λ, T, k
Intialization: Choose $\boldsymbol{w_1}$ s.t. $\|\boldsymbol{w_1}\| \leq 1/\sqrt{\lambda}$
$t = 1$
while $t \leq T$ **do**
  Choose $A_t \subseteq S$, where $|A_t| = k$
  Set $A_t^+ = \{(\boldsymbol{x}, y) \in A_t : y(\boldsymbol{w_t} \cdot \boldsymbol{x}) < 1\}$
  Set $\eta_t = \frac{1}{\lambda t}$
  Set $\boldsymbol{w_{t+\frac{1}{2}}} = (1 - \eta_t \lambda)\boldsymbol{w_t} + \frac{\eta_t}{k} \sum_{(\boldsymbol{x},y) \in A_t^+} y\boldsymbol{x}$
  Set $\boldsymbol{w_{t+1}} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\boldsymbol{w_{t+\frac{1}{2}}}\|}\right\} \boldsymbol{w_{t+\frac{1}{2}}}$
end while
Outputs: $\boldsymbol{w_{T+1}}$

2. **Incorporate the bias b only in the loss function**, $|1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b)|_+$, and compute the subgradient with respect to b –the subgradient with respect to $\boldsymbol{w}$ remain intact–. This strategy is also very simple but the cost function is no longer strongly convex, which leads to a slower convergence rate.

3. **Reformulate the loss term as an optimal value function in whose value at each $\boldsymbol{w}$ depends on a minimization problem over b.** The same update rule can be used for $\boldsymbol{w}$. The bias term b is also obtained while the subgradient of $\boldsymbol{w}$ is calculated. Despite being the most desirable update rule, this approach leaves aside the simplicity of the preceding solutions being computationally much more expensive. Its application for large values of k is limited.

Alternatively, the standard SGD packages (`http://leon.bottou.org/projects/sgd`) implement an heuristic in which the bias term is updated via subgradient descent and using a smaller learning rate (scaled by the heuristically chosen parameter τ) as it is updated more frequently than weights. At each epoch t, not only the stochastic gradient descent is applied to the $\boldsymbol{w}$ vector but also to the bias term b: $b_{t+1} = b_t - \tau\eta_t\nabla_t^b$. The subgradient of the bias is given by $\nabla_t^b = -\frac{1}{|A_t|}\sum_{(\boldsymbol{x},y)\in A_t^+} y$.

2.3.3 Decision Trees

Decision trees classify patterns according to a sequence of questions organized hierarchically in such way that the next question depends on the answers to the preceding questions. This sequence of questions is represented as a decision tree in which each

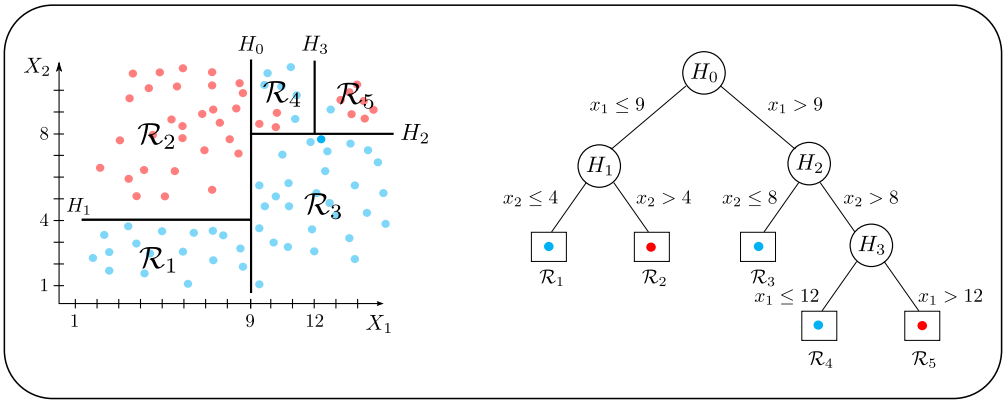


FIGURE 2.12: An example of decision tree model. The disjoint division of the input space can be represented as a decision tree with a rule associated to internal nodes (circles). The leaves (squares) determine the label assigned to each pattern.

question is associated to an **internal node** H_k and the initial question corresponds to the **root** of the tree. Tree growing algorithm starts by the whole training set as the root node and they recursively it divides the input space into disjoint subregions determined by the decision rules of the internal nodes. The process finishes when a predefined stopping criterion is reached and then, class assignment is performed by the terminal nodes also known as **leaves**. Each leaf in the tree corresponds to one of the disjoints regions in which the input space has been divided. An example of this architecture is presented in Figure 2.12. Internal nodes are indicated as circles while leaves are displayed as squares. H_0, H_1, H_2 and H_3 are the questions corresponding to the internal nodes being H_0 the root of the tree. The five disjoints subregions $\mathcal{R}_1$,$\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$ and $\mathcal{R}_5$ are associated to one leaf in the tree.

One of the most noticeable drawbacks of decision tree models is their instability under small changes on the training set which can produce extremely different decision trees. In spite of this, decision trees fill a privileged position among machine learning classifiers due to their simplicity, their interpretability and their speed classifying unlabeled samples. The last point is especially remarkable in the scope of this thesis, pointing out decision trees as a good candidate for those applications requiring real-time predictions [57].

Typically, the questions associated with internal nodes consist in logical conditions each of which depends on a single feature. It produces **axis-parallel splits** of the feature space as illustrated in Figures 2.7(b) and 2.12. The well-known Classification And Regression Trees (CART) [58] and C4.5 [59] algorithms can be categorized as axis-parallel

models. Decision trees with rules based on linear combinations of the features are called **oblique decision trees** and they were proposed to enhance the expressiveness of the splits and to favor a better adjustment of the distribution of data. Unfortunately, these models are prone to overfit the training samples and they are computationally more expensive. Examples of oblique decision tree algorithms are CART with Linear Combinations (CART-LC) [58] and the Oblique Classifier 1 (OC1) method [60]. Apart from the parallel/oblique splitting, the number of links descending from a node (**branching factor**) is another important parameter to take into account. While algorithms like CART establish dichotomic answers (true/false), other kind of decision trees like C4.5 can have more than two descendant per node. In any case, a decision tree is defined by the following five elements [58]:

1. The **criterion for assigning the class label** to a pattern when it reaches a leaf of the tree.
2. The **goodness of the node split** which needs to be evaluated in each node of the tree.
3. The **type of test or splitting criterion** carried out in each node of the tree to determine the descendant of each pattern.
4. The **stop-splitting rule** to decide when the recursive expansion is finished.
5. The **prunning algorithm** to improve the generalization capability of the model.

Keeping with the algorithm that will be presented in Chapter 5, the subsequent analysis assumes binary classification problems and dichotomic decision rules. In addition, the notation presented in Table 2.3 will be used throughout this section. The probabilities $P(+|H_k)$, $P(-|H_k)$, $P(H_k^l)$ and $P(H_k^r)$ are estimated using the training data set as follows,

$$\begin{aligned} P(+|H_k) &= \frac{N_{H_k}^+}{N_{H_k}} & P(-|H_k) &= \frac{N_{H_k}^-}{N_{H_k}} \\ P(H_k^l) &= \frac{N_{H_k}{}^l}{N_{H_k}} & P(H_k^r) &= \frac{N_{H_k}{}^r}{N_{H_k}}. \end{aligned}$$

Notation	Description
T	Decision tree
$\tilde{T}$	Set of leaves of T
$\lvert\tilde{T}\rvert$	Number of leaves in T
H_k	Node in T
H_k^l	Left child of the node H_k
H_k^r	Right child of the node H_k
S_{H_k}	Subset of samples in the node H_k
$S_{H_k}^+$	Subset of positive samples in the node H_k
$S_{H_k}^-$	Subset of negative samples in the node H_k
N_{H_k}	Number of samples in the node H_k
$N_{H_k}^+$	Number of positive samples in the node H_k
$N_{H_k}^-$	Number of negative samples in the node H_k
$\mathcal{C}(H_k)$	Class assigned to the node H_k
$R_{\text{emp}}(T)$	Empirical error rate of T
$P(+\lvert H_k)$	Probability that a positive sample falls into H_k
$P(-\lvert H_k)$	Probability that a negative sample falls into H_k
$P(H_k^l)$	Probability that a sample in H_k falls into H_k^l
$P(H_k^r)$	Probability that a sample in H_k falls into H_k^r

TABLE 2.3: Decision tree terminology.

Class Assignment Criterion. Once a pattern reaches a leaf of the decision tree, $H_k \in \tilde{T}$, it is assigned to the majority class in such leaf:

$$\mathcal{C}(H_k) = \text{argmax } \{P(+|H_k), P(-|H_k)\}\,. \tag{2.37}$$

This class assignment rule minimizes the misclassification cost of the training set.

Splitting Goodness. The definition of the splitting goodness is based on the impurity function concept for multiclass classification problems. In the particular case of binary

classification problems, the impurity function of a node H_k, $I(H_k) = I(P(+|H_k), P(-|H_k))$ is a multivariate function in a bidimensional space satisfying the following properties [58]:

1. I has a unique maximum at $\left(\frac{1}{2}, \frac{1}{2}\right)$.
2. I takes its minimum values only at the points $(1, 0)$ and $(0, 1)$.
3. I is symmetric of $P(+|H_k)$ and $P(-|H_k)$.

Then, the splitting goodness $\Delta I(H_k)$ is defined as,

$$\Delta I(H_k) = I(H_k) - P(H_k^l)\ I(H_k^l) - P(H_k^r)\ I(H_k^r). \tag{2.38}$$

As the aim of the decision tree is to minimize the overall misclassification rate, it would be natural to think in the classification error as impurity measure. Then, those splits reducing most the misclassification rate would be preferred and the impurity function of the node H_k would be defined as,

$$I(H_k) = 1 - \max\left\{P(+|H_k), P(-|H_k)\right\}.$$

This impurity function verifies all the properties given further up. However, as pointed out by Breiman et al. [58, Chapter 4], this measure presents two important limitations:

1. The improvement in the impurity can be zero for all the splits in S_{H_k}:

$$\begin{aligned}
\Delta I(H_k) &= I(H_k) - P(H_k^l)\ I(H_k^l) - P(H_k^r)\ I(H_k^r) \\
&= \sum_{h \neq \mathcal{C}(H_k)} P(h|H_k) - P(H_k^l)\ \min_i \sum_{h \neq i} P(h|H_k^l) - P(H_k^r)\ \min_i \sum_{h \neq i} P(h|H_k^r) \\
&= \sum_{h \neq \mathcal{C}(H_k)} \left[P(h|H_k^l)\ P(H_k^l) + P(h|H_k^r)\ P(H_k^r)\right] \\
&\quad -P(H_k^l)\ \min_i \sum_{h \neq i} P(h|H_k^l) - P(H_k^r)\ \min_i \sum_{h \neq i} P(h|H_k^r)
\end{aligned}$$

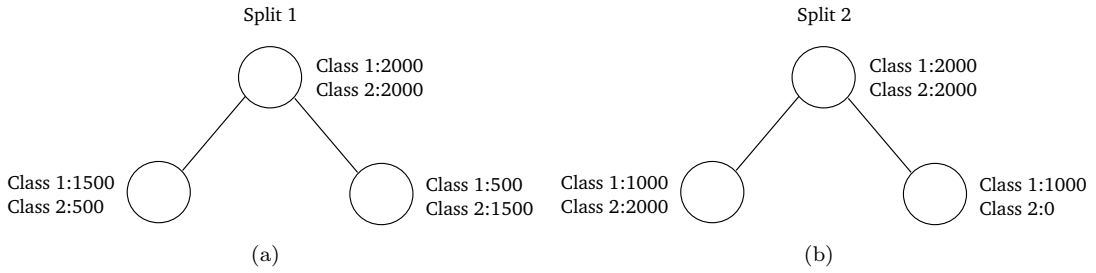


FIGURE 2.13: An example of two different splits in a decision tree. If classification error is used as impurity measure, both splits misclassified 1000 samples. Nevertheless, the second split seems more desirable for the future expansion of the tree.

and grouping terms,

$$\begin{aligned}\Delta I(H_k) &= P(H_k^l)\left[\sum_{h\neq\mathcal{C}(H_k)} P(h|H_k^l) - \min_i \sum_{h\neq i} P(h|H_k^l)\right] \\ &+ P(H_k^r)\left[\sum_{h\neq\mathcal{C}(H_k)} P(h|H_k^r) - \min_i \sum_{h\neq i} P(h|H_k^r)\right].\end{aligned}$$

The expressions in brackets are equals zero when $\mathcal{C}(H_k) = \mathcal{C}(H_k^l) = \mathcal{C}(H_k^r)$ which leads to undesirable situations; for example, suppose a binary classification problem with predominance of one of the classes in a node H_k, it is conceivable that the majority class in H_k would be the predominance class for every split of H_k. Thus, any split in H_k would have $\Delta I(H_k) = 0$ making it impossible to decide which split is the best.

2. The inadequacy of misclassification rate as impurity function for an iterative-split decision tree method (see Figure 2.13).

Having dismissed the misclassification rate as a measure of splitting goodness, the most known decision tree algorithms have adopted different splitting criteria. For example, CART can use the impurity measured called **Gini index**,

$$I(H_k) = P(+|H_k)\ P(-|H_k)$$

but CART can also use the **gain criterion** based on the entropy concept coming from the information theory field:

$$I(H_k) = -P(+|H_k)\ \log_2\left(P(+|H_k)\right) - P(-|H_k)\ \log_2\left(P(-|H_k)\right).$$

In any case, it has been shown that all the popular splitting criteria generate decision trees with similar generalization capabilities [58]. The main difference between using one measure or other lies in the size of the resulting tree. The gain criterion usually yields more compact decision trees [61] which favors its application in large-scale domains.

Splitting-Stop Criterion. A split can be stopped either when certain criterion is reached (for example, the improvement in the impurity measure or the number of patterns in the node are not large enough for successive splittings), or when all the patterns in the node belong to the same class (**homogeneous node**). All these rules set to stop the expansion of the tree before reaching homogeneous leaves can be considered as **preprunning** approaches trying to avoid overfitting. However, these criteria are seldom applied in practice because it is difficult to define a criterion which yield results close to optimal. Stopping tree growth can save much time, but makes optimum solutions less probable. A common practice is to build overfitted trees and prune them using a **pruning algorithm** as the one described in the following paragraph.

2.3.3.1 Cost-Complexity Pruning

Generally, including a pruning algorithm at the end of the training of a decision tree is mandatory, especially in those cases in which the stopping criterion does not incorporate *pre-pruning* rules. Most of the existing pruning methods remove the deepest nodes of the tree whose validity is in doubt as they can have been generated with few samples. Among them, the Cost-Complexity pruning algorithm proposed by Breiman [58] is described in what follows since it is a key element of the algorithm presented in Chapter 5.

As it name suggests, the **Cost-Complexity objective function** establishes a trade-off between the misclassification cost of the tree and its complexity, approximated by the number of leaves in the tree. Formally, the cost-complexity objective function for a tree T with a set of leaves $\tilde{T}$ is given by

$$R^{\alpha}_{\text{emp}}(T) = \alpha|\tilde{T}| + R_{\text{emp}}(T) \ , \tag{2.39}$$

where $\alpha \in [0, \infty)$ is a scalar parameter overweighting the complexity and misclassification terms. As α varies in the interval $[0, \infty)$, a set of decreasing-size subtrees of the original tree is obtained. In fact, when $\alpha = 0$ the complexity term in Equation 2.39 is not considered and therefore, the best tree is that with the lowest misclassification rate. On the other hand, there exists a large enough value for α which produces a decision tree formed exclusively by the root. It can be shown that for every value of α, there exists a tree $T(\alpha)$ minimizing Equation 2.39 [58, Chapter 10].

The main idea of the Cost-Complexity method is to construct a set of **decreasing-size subtrees** by means of variations in the α parameter. Although α can take infinite values in the interval $[0, \infty)$, the set of subtrees of the original tree is finite and thus, the problem is not so arduous at it looks at first glance. It works as follows, suppose that $T(\alpha_k)$ is the **minimizing tree** of Equation 2.39 given α_k then, this tree is still the minimizing tree in Equation 2.39 until a jump point α_{k+1} is reached. Now, $T(\alpha_{k+1})$ becomes the new minimizing tree until achieving the next jump point α_{k+2} and so on. The result of this process is a sequence of subtrees $T = T(0) > T(\alpha_1) > \ldots > T(\alpha_K) = \text{root}(T)$.

The search of the subtree minimizing the Cost-Complexity function among all the possible subtrees in T is unfeasible, being mandatory the implementation of an efficient pruning algorithm. The starting point is not the original tree T but T_0, the smallest subtree of T verifying $R(T_0) = R(T)$. The subtree T_0 can be easily obtained by removing the leaves of the original tree T which do not improve the misclassification error. The heart of the Cost-Complexity pruning lies in finding the **weakest-link cutting** based on the following proposition [58]:

Proposition 2.2. *For t any nonterminal node of T_0, T_t the subtree of T with root t and $\tilde{T}_t$ the set of terminal nodes of T_t,*

$$R_{emp}(t) > R_{emp}(T_t) = \sum_{t' \in \tilde{T}_t} R_{emp}(t') \ . \tag{2.40}$$

Then, if $\{t\}$ is the subbranch of T_t consisting of the single node t, the cost-complexity functions for $\{t\}$ and T_t can be written as,

$$\begin{aligned} R^{\alpha}_{\text{emp}}(\{t\}) &= R_{\text{emp}}(t) + \alpha \\ R^{\alpha}_{\text{emp}}(T_t) &= R_{\text{emp}}(T_t) + \alpha|\tilde{T}_t|. \end{aligned}$$

At some critical value of α, $R^{\alpha}_{\text{emp}}(\{t\})$ and $R^{\alpha}_{\text{emp}}(T_t)$ met. Working out α in $R_{\text{emp}}(t)+\alpha = R_{\text{emp}}(T_t) + \alpha|\tilde{T}_t|$:

$$\alpha = \frac{R_{\text{emp}}(t) - R_{\text{emp}}(T_t)}{|\tilde{T}_t| - 1}.$$

Regarding Equation 2.40, α is positive and it can be interpreted as the value which makes the subbranch $\{t\}$ preferable to T_t. Writing the previous expression as a function of the set of internal nodes t in T_0,

$$g_0(t) = \begin{cases} \frac{R_{\text{emp}}(t) - R_{\text{emp}}(T_t)}{|\tilde{T}_t| - 1} & \text{if } t \notin \tilde{T}_0; \\ +\infty & \text{if } t \in \tilde{T}_0. \end{cases}$$

The weakest link $\bar{t}_0$ in T_0 is the node minimizing $g_0(t)$. Intuitively, $\bar{t}_0$ is the first node that as α increases, $R^{\alpha}_{\text{emp}}(\{\bar{t}_0\})$ becomes equal to $R^{\alpha}_{\text{emp}}(T_{\bar{t}_0})$ and therefore, $\bar{t}_0$ is preferable to $T_{\bar{t}_0}$. The critical value for α associated to the weakest link is $\alpha_1 = g_0(\bar{t}_0)$. The process is repeated starting from $T_1 = T_0 - T_{\bar{t}_0}$ instead of T_0. If at any step there are several candidates to minimize $g_k(t)$ (for example $g_k(\bar{t}_k) = g_k(\bar{t}'_k)$), the next subtree in the sequence is defined as $T_{k+1} = T_k - T_{\bar{t}_k} - T_{\bar{t}'_k}$. The algorithm finishes when only the root is left, having obtained the complete sequence of subtrees. The only point left to address is how to select one of these subtrees as the **optimum-size tree**. As

the training misclassification rate is a biased estimation of $R_{\text{emp}}(T_k)$, an independent **pruning set**, made up randomly or via cross-validation, is used to evaluate the goodness of each subtree and select that with the highest classification accuracy.

Other pruning methods have been developed in parallel to decision tree algorithms. For example, the C4.5 algorithm tackles the pruning task from other perspective, making pessimistic assumptions about the error rate in each node of the tree. A thorough analysis of the CART and C4.5 pruning techniques was carried out by Esposito et al. [62] concluding that, while C4.5 pruning tends to prune less than necessary, the Cost-Complexity method usually yields subtrees smaller than the optimal. Bearing in mind all these factors, the Cost-Complexity algorithm seems more suitable for large-scale machine learning systems trying to reduce the classification cost.

Table 2.4 gathers some of the advantages and limitations of the previous classifiers. In accordance with the goal of Chapter 5 of constructing a classifier capable of classifying new patterns in few milliseconds, the nonlinear SVMs do not look as the best option in spite of their effectiveness and robust mathematical background. However, the high prediction speed of linear SVMs (besides their good generalization properties) and decision trees together with the easy combination of decision trees with other classification algorithms, points to the hybrid architecture of decision trees and linear SVMs presented in Chapter 5 as an appealing solution to approximate nonlinear SVMs by piecewise linear functions with low classification cost.

Classifier	Advantages	Disadvantages
Linear	· Simplicity · Good scalability · Fast training algorithms in general · Fast classification	· Poor performance in non-linear domains
SVM	· Robust mathematical basis · Flexible method for creating nonlinear models · Good generalization · Good accuracy levels	· Lack of expressiveness and comprehensibility · Poor scalability of training algorithms (except for the linear kernel)
Decision trees	· Simplicity · Fast classification · Graphical representation · Relatively faster learning speed than other classification methods · Can be combined with other decision techniques	· Instability and high risk of overfitting · Higher training cost than linear models · The structure of decision boundaries limits its application · Oblique decision trees are not convex and computationally expensive

TABLE 2.4: Main advantages and disadvantages of some of the most popular learning algorithms.

Chapter 3

Quadratic Programming Feature Selection (QPFS)

Chapters 1 and 2 have highlighted how the incorporation of an effective feature selection phase in a large-scale classification system is highly recommendable in order to, not only reduce the computational cost of the learning algorithm, but also to favor the generalization and the understandability of the final model. In this respect, this chapter presents one of the contributions of this thesis: a new feature selection method for very large and high dimensionality multiclass classification problems [63]. The new method is in line with **filter feature selection techniques** introduced in Section 2.2.1 and, in particular, it is inspired by the mRMR algorithm which not only takes into account the relevance of each feature with the target class but also dependences among variables. The proposed method, named **Quadratic Programming Feature Selection**, reduces the feature selection task to a **quadratic optimization problem** simplifying the optimization process thanks to the **Nyström method** (Appendix C). The chapter presents the QPFS algorithm, including the Nyström approximation, error estimation and theoretical complexity. Finally, experiments comparing the proposed algorithm with state of the art filter techniques show that the new QPFS+Nyström method is a competitive and efficient filter-type feature selection algorithm for large-scale and high-dimensional supervised classification problems. The new method is superior in terms of computational efficiency to other successful feature selection algorithms while it maintains their classification accuracy.

3.1 Scalability of Feature Selection Methods

As discussed in Section 2.2, feature selection methods can be categorized into three groups as a function of their dependence on the classifier: filter methods perform feature selection regardless of the classifier, wrapper methods use search techniques to select candidate subsets of variables whose goodness is based on their classification accuracy, and embedded methods incorporate feature selection in the classifier objective function or algorithm. Section 2.2 has analyzed the most prominent advantages and disadvantages of each category, revealing their differences in terms of computational load as reflected in Figure 3.1. Among these groups, filter methods are often preferable to other selection methods for high dimensionality problems because of (i) their scalability, favored by their computational speed and their simplicity [64, 65]; and (ii) their usability with alternative classifiers, which means evaluate the costly feature selection process only once and then analyze different classifiers. A common disadvantage of filter methods is that most proposed techniques are univariate ignoring dependences among features. To overcome this problem, multivariate filter techniques taking into account dependences among features were proposed. Multivariate filters improve significantly the classification accuracy of univariate approaches but increase the computational effort. The applicability of wrapper methods in high-dimensional problems is particularly critical in those cases where the number of features is large compared to the number of training samples (e.g., 10,000 features and 100 samples); a common case in gene selection domains [21]. In this situations, wrapper methods not only can be much slower, especially in learning the classifier, but also can prone to *overfitting*. Finally, embedded techniques can be located in an intermediate point between univariate filter and wrapper methods, being far less computationally demanding than wrapper methods but more expensive than univariate filter approaches. Regarding the multivariate filters, the most influential embedded techniques have a computational cost similar that of the multivariate filters [65]. However, as pointed out in Chapter 1, large-scale classifiers can be focused on different aspects like accelerating the training time, improving the classification cost or reducing the memory requirements thus, multivariate filters are versatile enough to use them whatever is the final purpose. Moreover, their use as a preprocessing step can lighten the computational cost of the classifier and overcome overfitting at the same time [64].

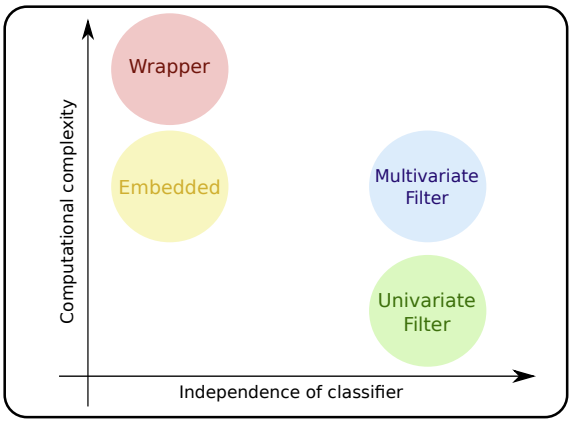


FIGURE 3.1: Dependence between the computational cost and the independence with the classifier of the main feature selection algorithms: univariate filter, multivariate filter, wrapper and embedded methods.

Leaving aside the computationally intensive wrapper methods, Table 3.1 shows the time charge of some of the most influential filter and embedded techniques aforementioned in Section 2.2. Note that these computational costs correspond to ranking features according to certain criterion but they do not include the cost of determining the optimal number of features to be selected. As expected, the univariate filter feature selection methods (**MaxRel**) has the lowest computational cost but it does not takes into account dependences among features, which limits its effectiveness. The **SFS** method captures linear dependences as well but it is not simply a feature selection technique as it is able to generate new features. However, its cost is cubic in the number of features which makes SFS hardly scalable. In spite of the computational efficiency of multivariate feature selection methods based on linear similarities – like **CFS** –, multivariate algorithms, such as **mRMR** or **ReliefF**, with reasonable computational requirements and based on nonlinear similarities are often preferred as they are not reduced to linear dependences [28, 66]. Specifically, mRMR has a quadratic dependence on the dimension of the classification problem while ReliefF scales at least quadratically with the number of patterns. However, the mRMR provides in general better classification rates than ReliefF as it will be shown later in Section 3.3. Finally, the **RFE-SVM** algorithm introduced in Section 2.2.3 is a good example of embedded methods because of its classification effectiveness and its leadership in terms of computational cost. RFE-SVM performs backward feature elimination and its cost is totally dependent on the criterion used to remove features. The most popular techniques are (i) deleting features by a factor of 2 (that is, reduce the number of remaining features by half at each iteration)

Model		**Examples**	**Complexity**
Filter	*Univariate*	MaxRel	NM
	Multivariate	CFS	NM^2
		SFS	NM^3
		mRMR	NM^2
		ReliefF	N^2M
Embedded		RFE-SVM	$\max(N, M)N^2$ (Features removed by a factor of 2)
			N^2M^2 (Features removed one by one)

TABLE 3.1: Frequently used filter and embedded feature selection methods. N is the number of training patterns and M the number of features.

or (ii) removing features one by one.

3.2 The QPFS Algorithm

A depth insight of state-of-the-art feature selection methods points at filter techniques as a good approach to handle large-scale and high-dimensional domains. The univariate filter algorithms are the best choice in terms of computational efficiency but their classification performance is not competitive enough, making multivariate approach almost mandatory. However, the scalability of these methods for high dimensional problems is still compromised. The algorithm presented in this chapter is motivated by the possibility of accelerating the most successful multivariate filters while maintaining their classification effectiveness. The aim of the **QPFS method** is dealing with large datasets with high dimensionality providing a time complexity improvement respect to current multivariate filter methods. To achieve this goal, the QPFS method proposes a novel formulation of the feature selection task based on **quadratic programming**. The quadratic programming methodology has previously been successful for machine learning –Support Vector Machines introduced in Section 2.3.2 have been its representative par excellence during last years – and for a broad range of other quite different applications [67].

Suppose a classifier learning problem involving N training samples and M variables. A quadratic programming problem is to minimize a multivariate quadratic function subject

to linear constraints as follows:

$$\min_{\boldsymbol{w}} \left\{ \frac{1}{2} \boldsymbol{w}^T Q \boldsymbol{w} - F^T \boldsymbol{w} \right\} . \tag{3.1}$$

Above, $\boldsymbol{w}$ is an M-dimensional vector, $Q \in \mathbb{R}^{M \times M}$ is a symmetric positive semidefinite matrix, and F is a vector in $\mathbb{R}^M$ with non-negative entries. Applied to the feature selection task, the QPFS method assumes that Q represents the similarity among variables (**redundancy**), F measures how correlated each feature is with the target class (**relevance**) and the optimal value for $\boldsymbol{w}$ is used for **feature ranking** by interpreting the components of $\boldsymbol{w}$ as the weight (or importance) of each feature. Thus, features with higher weights are better variables to use for subsequent classifier training. Since w_i represents the weight of each variable, it is reasonable to enforce the following constraints:

$$w_i \geqslant 0 \text{ for all } i = 1, \ldots, M \tag{3.2}$$

$$\sum_{i=1}^{M} w_i = 1 . \tag{3.3}$$

In order to provide some flexibility to the model, let introduce a **scalar parameter** $\alpha \in [0, 1]$ which, depending on the learning problem, overweights the linear and the quadratic terms as follows,

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \left\{ \tfrac{1}{2}(1-\alpha) \boldsymbol{w}^T Q \boldsymbol{w} - \alpha F^T \boldsymbol{w} \right\} \\ \text{s.t.} \quad & w_i \geq 0 \qquad \forall i = 1 \ldots M \\ & \|\boldsymbol{w}\|_1 = 1. \end{aligned} \tag{3.4}$$

If $\alpha = 1$, only relevance is considered; the quadratic programming problem becomes linear and equivalent to classical univariate filter methods in which features are ranked according to their relevance with the class; among these approaches, criteria based in Pearson correlation or mutual information (MaxRel) as the ones presented in Section 2.2.1 or decision trees (Section 2.3.3) can be found. On the contrary, if $\alpha = 0$ only independence between features is considered and features with higher weights are

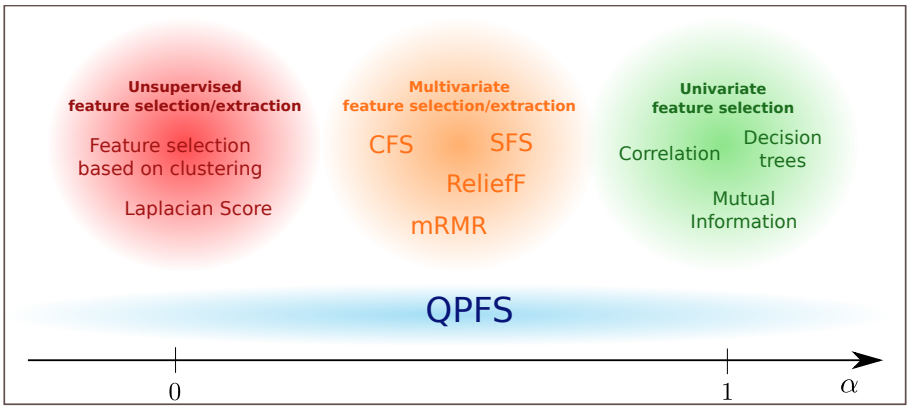


FIGURE 3.2: Relation between the α parameter of QPFS and other feature selection and extraction techniques

those having the lowest similarity coefficients with the rest of features. In this case, the QPFS algorithm can be categorized into unsupervised feature selection algorithms like those based on clustering algorithms [68, 69] or based on the Laplacian Score [70]. However, values of α in the interval $(0, 1)$ establish a trade-off between dependences among features which leads QPFS to be considered as a multivariate feature selection method like the ones introduced in Section 2.2.1 and whose scalability have been analyzed at the beginning of this chapter: CFS, MBF, SFS, mRMR or ReliefF. To sum up, the α parameter provides QPFS with a flexible model capable of represent the main categories of feature selection techniques as shown in Figure 3.2.

Every dataset has its best choice of α to extract the minimum number of features for classification purposes. The best methodology for determining an appropriate value for α would be to use a validation subset. However, that approach requires evaluating the accuracy of the underlying classifier for each point in a grid of values for α in which case, QPFS would become a wrapper feature selection method instead of a filter method because it would depend on the classifier accuracy to determine the proper value of α. Unfortunately, the evaluation of a classifier for each α value makes the QPFS extremely costly for high dimensional datasets, making necessary an **heuristic approximation** for α providing competitive classification results and less computational load. A reasonable choice of α must balance the linear and quadratic terms of Equation 3.4 in order to ensure that both redundancy and relevance are taking into account. If features are only slightly redundant, i.e. they have low correlation with each other, then the linear term in Equation 3.1 is dominant: $\bar{f} \gg \bar{q}$. Making α small reduces this dominance. On the other hand, if the features have a high level of redundancy relative to relevance ($\bar{q} \gg \bar{f}$),

then the quadratic term in Equation 3.1 can dominate the linear one. In this case, overweighting the linear term (α close to 1) makes the objective function be balanced. Thus, estimating the mean value $\bar{q}$ of the elements of the matrix Q and the mean value $\bar{f}$ of the elements of the vector F as

$$\bar{q} = \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} q_{ij} \tag{3.5}$$

$$\bar{f} = \frac{1}{M}\sum_{i=1}^{M} f_i \,, \tag{3.6}$$

the relevance and redundancy terms in Equation 3.4 are balanced when $(1-\hat{\alpha})\,\bar{q} = \hat{\alpha}\bar{f}$. Then, a reasonable initial estimate of α is

$$\hat{\alpha} = \frac{\bar{q}}{\bar{q}+\bar{f}} \,. \tag{3.7}$$

As it will be shown in practice (Section 3.3), this α heuristic leads to good results in terms of classification accuracy and number of selected features.

Regarding that the quadratic programming problem given by Equation 3.4 is convex if the matrix Q is positive semidefinite and strictly convex if matrix Q is positive definite [71]. Hence, one immediate advantage of the QPFS formulation is that it is sufficiently general to permit the use of any symmetric similarity measure as long as Q verifies these conditions in Q. This thesis suggests the **Pearson correlation coefficient** and **mutual information** as similarity measures because they are well-known measurements of dependences between random variables (Section 2.2.1). Nevertheless, the QPFS algorithm has also been successfully applied by Sousa et al. with other similarity criterion [72].

When correlation is used, each matrix element q_{ij} is defined to be the absolute value of the Pearson correlation coefficient of the pair of variables X_i and X_j, i.e. $q_{ij} = |\hat{\rho}_{ij}|$. Suppose a classifier learning problem with C classes, the relevance weight of variable X_i, F_i, is computed using a modified correlation coefficient [73] which is an extension to the C-class classification scenario:

$$F_i = \sum_{k=1}^{C} \hat{P}(Y = k)|\hat{\rho}_{iY_k}|$$

where Y is the target class variable, Y_k is a binary variable taking the value 1 when $Y = k$ and 0 otherwise, $\hat{P}(Y = k)$ is the empirical prior probability of class k, and $\hat{\rho}_{iY_k}$ is the sample correlation between the feature X_i and the binary variable Y_k, computed according to Equation 2.6.

As discussed in Section 2.2.1, since the correlation coefficient only measures *linear* relationships between random variables it may not be suitable for some classification problems, being necessary the use of nonlinear similarity measures as mutual information. In this case, the QPFS quadratic term is $q_{ij} = I(X_i, X_j)$ and the linear one is $F_i = I(X_i, Y)$. QPFS using mutual information as its similarity measure resembles mRMR, but there is an important difference: while the mRMR method selects features greedily, as a function of features chosen in previous steps, QPFS is not greedy and provides a ranking of features that takes into account simultaneously the mutual information between all pairs of features and the relevance of each feature to the class label.

The quadratic programming formulation of the feature selection problem is elegant and provides insight but the formulation by itself does not significantly reduce the computational complexity of well established feature selection methods like mRMR. Nevertheless, in high-dimensional domains, it is likely that the feature space is redundant. If so, the symmetric matrix Q is singular and Equation 3.4 can then be simplified and solved in a space of dimension less than M, thus reducing the computational cost.

Given the diagonalization $Q = U\Lambda U^T$ in decreasing order of eigenvalues, Equation 3.4 is equivalent to

$$\min_{\boldsymbol{w}} \left\{ \frac{1}{2}(1-\alpha)\boldsymbol{w}^T U\Lambda U^T \boldsymbol{w} - \alpha F^T \boldsymbol{w} \right\} . \tag{3.8}$$

If the rank of Q is $k \ll M$, then the diagonalization $Q = U\Lambda U^T$ can be written as $Q = \bar{U}\bar{\Lambda}\bar{U}^T$, where $\bar{\Lambda}$ is a diagonal square matrix consisting of the highest k eigenvalues of Q in decreasing order and $\bar{U}$ is a $M \times k$ matrix consisting of the first k eigenvectors of Q. Then, Equation 3.8 can be rewritten as

$$\min_{\boldsymbol{w}} \left\{ \frac{1}{2}(1-\alpha)\boldsymbol{w}^T \bar{U}\bar{\Lambda}\bar{U}^T\boldsymbol{w} - \alpha F^T\boldsymbol{w} \right\} .$$

Let $z = \bar{U}^T\boldsymbol{w}$ be a vector in $\mathbb{R}^k$. The optimization problem is reduced to minimizing a derived quadratic function in a k-dimensional space:

$$\min_{\boldsymbol{z}} \left\{ \frac{1}{2}(1-\alpha)\boldsymbol{z}^T \bar{\Lambda}\boldsymbol{z} - \alpha F^T\bar{U}\boldsymbol{z} \right\}$$

under $M+1$ constraints:

$$\begin{aligned} \bar{U}\boldsymbol{z} &\geq \overrightarrow{0} \\ \sum_{i=1}^{M}\sum_{j=1}^{k} \bar{u}_{ij} z_j &= 1. \end{aligned}$$

And the original vector $\boldsymbol{w}$ can be approximated as $\boldsymbol{w} \approx \bar{U}\boldsymbol{z}$.

Nonetheless, this scenario is not practical: the matrix Q is seldom precisely singular for real world datasets. However, Q can normally be reasonably approximated by a **low-rank matrix** formed from its $\tilde{k}$ eigenvectors whose eigenvalues are greater than a fixed threshold $\delta > 0$ [74]. More precisely, let $\tilde{Q} = U\Gamma U^T$ be the $\tilde{k}$-rank approximation of Q, where $\Gamma \in \mathbb{R}^{M\times M}$ is a diagonal matrix consisting of the $\tilde{k}$ highest eigenvalues of Q and the rest of diagonal entries are zero. Then, the approximate quadratic programming problem is formulated as

$$\min_{\boldsymbol{w}} \left\{ \frac{1}{2}(1-\alpha)\boldsymbol{w}^T U\Gamma U^T\boldsymbol{w} - \alpha F^T\boldsymbol{w} \right\} .$$

Equivalently,

$$\min_{\boldsymbol{z}} \left\{ \frac{1}{2}(1-\alpha)\boldsymbol{z}^T \widetilde{\Gamma}\boldsymbol{z} - \alpha F^T\widetilde{U}\boldsymbol{z} \right\} \tag{3.9}$$

where $\boldsymbol{z} = \widetilde{U}^T\boldsymbol{w} \in \mathbb{R}^{\tilde{k}}$, $\widetilde{\Gamma} \in \mathbb{R}^{\tilde{k}\times\tilde{k}}$ is a diagonal matrix with the nonzero eigenvalues of Γ and $\widetilde{U} \in \mathbb{R}^{M\times\tilde{k}}$ are the first $\tilde{k}$ eigenvectors of U. The $M+1$ constraints of the

optimization problem are now defined as

$$\begin{aligned} \widetilde{U}\boldsymbol{z} &\geq \overrightarrow{0} \\ \sum_{i=1}^{M}\sum_{j=1}^{\tilde{k}} \tilde{u}_{ij} z_j &= 1\ . \end{aligned}$$

A diagram of the proposed approximation of the QPFS model in a lower-dimensional space is shown in Figure 3.3(a). It must be remarked that the model given by Equation 3.9 is an approximation of the original model (Equation 3.4), and thus the optimal solutions found by both methods are not necessarily the same. An estimation of the error due to this approximation is given later in Section 3.2.2.

3.2.1 QPFS + Nyström: Approximate Solution of the Quadratic Programming Problem

Although the QPFS approximation (Equation 3.9) reduces the quadratic programming problem to a lower dimensional space, the bottleneck of QPFS falls now on the diagonalization of the $Q \in \mathbb{R}^{M\times M}$ matrix, whose cubic cost, $O(M^3)$, compromises the QPFS scalability.

However, taking advantage of the redundancy that typically makes the matrix Q almost singular, it is possible to speed up the diagonalization through the **Nyström approximation**. When the feature space is highly redundant, the rank of Q is much smaller than M and the Nyström method can approximate eigenvalues and eigenvectors of Q by solving a **smaller eigenproblem** using only a subset of rows and columns of Q [1]. Suppose that $k < M$ is the rank of Q which is represented as

$$Q = \begin{pmatrix} A & B \\ B^T & E \end{pmatrix} \tag{3.10}$$

where $A \in \mathbb{R}^{k\times k}$, $B \in \mathbb{R}^{k\times(M-k)}$, $E \in \mathbb{R}^{(M-k)\times(M-k)}$, and the rows of $[A\ B]$ are independent. Then, the eigenvalues and eigenvectors of Q can be calculated exactly

[1] A more detailed description of this method as well a brief discussion on Nyström sampling techniques are given in Appendix C.

from the submatrix $[A\ B]$ and the diagonalization of A. Let $S = A + A^{-\frac{1}{2}} B B^T A^{-\frac{1}{2}}$ and its diagonalization $S = R\hat{\Sigma}R^T$ then, the highest k eigenvalues of Q are given by $\tilde{\Lambda} = \hat{\Sigma}$ and its associated eigenvectors $\tilde{U}$ are calculated as,

$$\tilde{U} = \begin{pmatrix} A \\ B^T \end{pmatrix} A^{-\frac{1}{2}} R \hat{\Sigma}^{-\frac{1}{2}} \ .$$

The application of the Nyström method entails some practical issues:

1. A prior knowledge of the rank k of Q is, in general, unfeasible and it is necessary to estimate the number of subsamples r to be used in the Nyström approximation.

2. The r rows of $[A\ B]$ should be, ideally, linearly independent. If the rank of Q is greater than r or the rows of $[A\ B]$ are not linearly independent, an approximation of the diagonalization of Q is obtained whose error can be quantified, in general, as $\|E - B^T A^{-1} B\|$.

Although the Nyström approximation is not error-free, if the redundancy of the feature space is large enough, then good approximations can be achieved, as shown later on. When **QPFS+Nyström** is used, not only the diagonalization of the redundancy matrix can be accelerated but also the calculation of the entries of such matrix. As explained above, the Nyström approximation just needs to know the submatrix $[A\ B]$ whose calculation requires $k \times M$ operations in contrast with the calculation of the whole Q matrix consisting of $M \times M$ entries (in fact, only $\frac{M^2}{2}$ entries should be computed due to the symmetry in Q). In this case, it would be inefficient to compute all the entries of Q only to estimate the value of the α parameter so, a slightly different rule for setting the value of α is suggested. The Nyström approximation $\hat{Q}$ of the original matrix Q is defined as,

$$\hat{Q} = (\hat{q_{ij}}) = \begin{pmatrix} A & B \\ B^T & B^T A^{-1} B \end{pmatrix} \ .$$

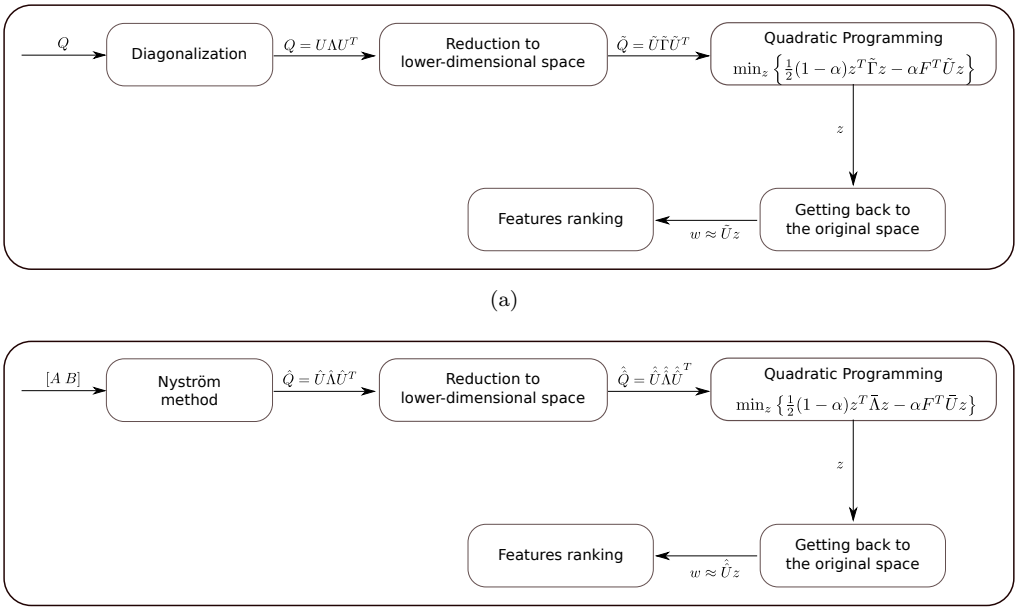


FIGURE 3.3: 3.3(a) Diagram of the QPFS algorithm in a lower-dimensional space. 3.3(b) Diagram of the QPFS algorithm using the Nyström method. $[A\ B]$ is the upper $r \times M$ submatrix of Q.

An intuitive way to extend the α rule given by Equation 3.7 to the QPFS+Nsyström method would be to compute the mean value of $\hat{Q}$. Unfortunately, it requires to obtain the submatrix $B^T A^{-1} B$ with a computational cost of $O(k^3 + (M-k)^2 k^2)$, which is still expensive[2]. Therefore, the mean value of $\hat{Q}$ is approximated as the mean value of the entries of the submatrix $[A\ B]$ as follows,

$$\bar{\hat{q}} = \frac{1}{kM} \sum_{i=1}^{k} \sum_{j=1}^{M} \hat{q_{ij}}.$$

The mean value $\bar{f}$ of the vector F is still calculated using Equation 3.6, since QPFS+Nyström needs to know all the coordinates of F. To sum up, the α heuristic for the QPFS+Nyström method is

$$\hat{\alpha} = \frac{\bar{\hat{q}}}{\bar{\hat{q}} + \bar{f}}. \tag{3.11}$$

[2]The cubic cost k^3 corresponds to the inversion of matrix A and the cost $(M-k)^2k^2$ is associated to the matrix multiplication $B^T A^{-1} B$.

A diagram of the proposed QPFS+Nyström method is shown in Figure 3.3(b) and it can be summarized as follows:

1. Compute the F vector representing the dependence of each variable with the class.
2. Choose r rows of Q according to some criterion (typically, uniform sampling without replacement[3]).Arrange the Q matrix so that these r rows are the first ones. Define the $[A\ B]$ matrix to be the first r rows of Q.
3. Set the value of the α parameter according to Equation 3.11.
4. Apply the Nyström method knowing $[A\ B]$. Obtain an approximation of the eigenvalues and eigenvectors of Q, $\hat{Q} = \hat{U}\hat{\Lambda}\hat{U}^T$.
5. Formulate the quadratic programming problem in the lower dimensional space $\hat{\hat{Q}} = \hat{\hat{U}}\hat{\hat{\Lambda}}\hat{\hat{U}}^T$.
6. Solve the quadratic programming problem in the subspace to obtain the solution vector $\boldsymbol{z}$.
7. Return to the original space via $\boldsymbol{w} = \hat{\hat{U}}\boldsymbol{z}$.
8. Rank the variables according to the coefficients of vector $\boldsymbol{w}$. In case of equal coefficients, rank them by decreasing relevance to the class.

3.2.2 Error Estimation

Any of the previous QPFS formulations reducing the quadratic programming problem to a lower-dimensional space are subject to an approximation error whenever the original space is dimensionality higher than the reduced space where the quadratic program is going to be solved. Moreover, the application of the Nyström method entails considering the error owing to the approximation of the eigenproblem.

Firstly, a bound for the QPFS approximation error using the exact matrix diagonalization (Figure 3.3(a)) is provided. Given the solutions $\boldsymbol{w}^*$ of Equation 3.4 and $\tilde{\boldsymbol{w}}^*$ of Equation 3.9, the error of the approximation can be estimated using the following theorem by Fine et al. [74],

[3]See Appendix C.

Theorem 3.1. *Given $\tilde{Q}$ a $\tilde{k}$-rank approximation of Q, if $(Q-\tilde{Q})$ is positive semidefinite and $tr(Q-\tilde{Q}) \leq \epsilon$ then the optimal value of the original problem is larger than the optimal objective value of the perturbed problem and their difference is bounded by*

$$\tilde{g}(\tilde{\boldsymbol{w}}^*) \leq g(\boldsymbol{w}^*) \leq \tilde{g}(\tilde{\boldsymbol{w}}^*) + \frac{d^2 l\epsilon}{2} \tag{3.12}$$

where l is the number of active constraints in the perturbed problem and d is an upper bound for the $\boldsymbol{w}$'s coefficients of the original solution.

In the QPFS case, $0 \leq w_i \leq 1$ and $d = 1$. The matrix $(Q - \tilde{Q})$ is positive semidefinite since $(Q - \tilde{Q}) = U(\Lambda - \Gamma)U^T$ and $(\Lambda - \Gamma)$ is a diagonal matrix with positive eigenvalues upper bounded by δ. Moreover $\epsilon \leq (M - \tilde{k})\delta$ and $l \leq M + 1$, so

$$g(\boldsymbol{w}^*) - \tilde{g}(\tilde{\boldsymbol{w}}^*) \leq \frac{l(M-\tilde{k})\delta}{2} \leq \frac{(M+1)(M-\tilde{k})\delta}{2} = \gamma \tag{3.13}$$

where $g(\boldsymbol{w})$ and $\tilde{g}(\boldsymbol{w})$ are defined as

$$g(\boldsymbol{w}) = \frac{1}{2}(1-\alpha)\boldsymbol{w}^T Q \boldsymbol{w} - \alpha F^T \boldsymbol{w} \text{ for } \boldsymbol{w} \in \mathbb{R}^M \tag{3.14}$$

$$\tilde{g}(\boldsymbol{w}) = \frac{1}{2}(1-\alpha)\boldsymbol{w}^T \tilde{\Gamma} \boldsymbol{w} - \alpha F^T \tilde{U} \boldsymbol{w} \text{ for } \boldsymbol{w} \in \mathbb{R}^{\tilde{k}} \,. \tag{3.15}$$

The analysis of the QPFS+Nyström error is more arduous as it has two levels of approximation:

1. The first level is to approximate the eigenvalues and eigenvectors of the original matrix Q based on only a subset of rows, applying the Nyström method: $\hat{Q} = \hat{U}\hat{\Lambda}\hat{U}^T$. One of the critical issues with the Nyström method is how to choose the subset of rows to use [75]. Ideally, the number of linearly independent rows of $[A\ B]$ should be the rank of Q. Uniform sampling without replacement has been used to select the rows of $[A\ B]$ submatrix because of (i) its successful results in other applications and (ii) the existence of theoretical performance bounds which show that this technique not only is efficient in terms of time and space but also

it produces more effective approximations [76]. The following theorem by Kumar et al. [76] is used,

Theorem 3.2. *Let $Q \in \mathbb{R}^{M\times M}$ be a symmetric positive semidefinite Gram (or kernel) matrix. Assume that r columns of Q are sampled uniformly at random without replacement ($r > k$), let $\hat{Q}$ be the rank-k Nyström approximation to Q, and let $\tilde{Q}$ the best rank-k approximation to Q. For $\epsilon > 0$, if $r \geq \frac{64k}{\epsilon^4}$, then*

$$\mathbb{E}\left[\|Q - \hat{Q}\|_F\right] \leq \|Q - \tilde{Q}\|_F + \epsilon \left[\left(\frac{M}{r}\sum_{i\in D(r)} Q_{ii}\right)\sqrt{M\sum_{i=1}^{M} Q_{ii}^2}\right]^{\frac{1}{2}} \tag{3.16}$$

where $\sum_{i\in D(r)} Q_{ii}$ is the sum of the largest r diagonal entries of Q and $\|\cdot\|_F$ represents the Frobenius norm.

The best rank-$\tilde{k}$ approximation to Q is $\tilde{Q} = U\Gamma U^T$ as given in Section 3.2. As expected, Equation 3.16 shows that the error in the Nyström approximation decreases with the number of sampled rows, r.

2. The second level of approximation corresponds to solve the quadratic programming problem using the Nyström approximation: only eigenvalues higher than a fixed threshold $\delta > 0$ are considered in the rank of matrix $\hat{Q}$. Then, these top $\tilde{k}$ eigenvalues of matrix $\hat{Q}$ are taken to make up a diagonal matrix $\hat{\hat{\Lambda}} \in \mathbb{R}^{\tilde{k}\times\tilde{k}}$ whose eigenvectors are the columns of matrix $\hat{\hat{U}} \in \mathbb{R}^{M\times\tilde{k}}$. Therefore, the QPFS+Nyström method approximates Q by $\hat{\hat{Q}} = \hat{\hat{U}}\hat{\hat{\Lambda}}\hat{\hat{U}}^T$ and the quadratic programming problem is defined as,

$$\hat{\hat{g}}(\boldsymbol{w}) \quad = \quad \frac{1}{2}(1-\alpha)\boldsymbol{w}^T\hat{\hat{Q}}\boldsymbol{w} - \alpha F^T\boldsymbol{w} \text{ for } \boldsymbol{w} \in \mathbb{R}^M.$$

By denoting the optimal solution of $\hat{\hat{g}}(\boldsymbol{w})$ as $\hat{\hat{\boldsymbol{w}}}^*$ and defining $g(\boldsymbol{w})$ and $\tilde{g}(\boldsymbol{w})$ as in Equations 3.14 and 3.15, respectively, the total error in the QPFS+Nyström approximation is obtained as follows,

$$\begin{aligned}\mathbb{E}\left[\left|g(\boldsymbol{w}^*) - \hat{\hat{g}}(\hat{\hat{\boldsymbol{w}}}^*)\right|\right] &\leq \mathbb{E}\left[\left|g(\hat{\hat{\boldsymbol{w}}}^*) - \hat{\hat{g}}(\hat{\hat{\boldsymbol{w}}}^*)\right|\right] \\ &= \frac{1}{2}(1-\alpha)\mathbb{E}\left[\left|(\hat{\hat{\boldsymbol{w}}}^*)^T\left(Q - \hat{\hat{Q}}\right)(\hat{\hat{\boldsymbol{w}}}^*)\right|\right] \\ &= \frac{1}{2}(1-\alpha)\mathbb{E}\left[\left\|(\hat{\hat{\boldsymbol{w}}}^*)^T\left(Q - \hat{\hat{Q}}\right)(\hat{\hat{\boldsymbol{w}}}^*)\right\|_2\right].\end{aligned}$$

The α parameter of QPFS belongs to the interval $[0, 1]$ which implies,

$$\frac{1}{2}(1-\alpha)\mathbb{E}\left[\left\|(\hat{\hat{\boldsymbol{w}}}^*)^T\left(Q - \hat{\hat{Q}}\right)(\hat{\hat{\boldsymbol{w}}}^*)\right\|_2\right] \leq \frac{1}{2}\mathbb{E}\left[\left\|(\hat{\hat{\boldsymbol{w}}}^*)^T\left(Q - \hat{\hat{Q}}\right)(\hat{\hat{\boldsymbol{w}}}^*)\right\|_2\right].$$

And applying the consistency of the L_2-norm for matrices,

$$\begin{aligned}\frac{1}{2}\mathbb{E}\left[\left\|(\hat{\hat{\boldsymbol{w}}}^*)^T\left(Q - \hat{\hat{Q}}\right)(\hat{\hat{\boldsymbol{w}}}^*)\right\|_2\right] &\leq \frac{1}{2}\mathbb{E}\left[\|Q - \hat{\hat{Q}}\|_2\|\hat{\hat{\boldsymbol{w}}}^*\|_2^2\right] \\ &\leq \frac{1}{2}(M+1)\,\mathbb{E}\left[\|Q - \hat{\hat{Q}}\|_F\right].\end{aligned}$$

The last inequality is obtained from the constraints $0 \leq w_i \leq 1$ which must be satisfied by the optimum value $\hat{\hat{\boldsymbol{w}}}^*$. Applying the bound for the Nyström method with uniform sampling without replacement (Equation 3.16) and noting that $\|Q - \tilde{Q}\|_F \leq \text{trace}\left(Q - \tilde{Q}\right) \leq (M - \tilde{k})\delta$ by construction, the above expression can be upper bounded as follows,

$$\begin{aligned}\mathbb{E}\left[\left|g(\boldsymbol{w}^*) - \hat{\hat{g}}(\hat{\hat{\boldsymbol{w}}}^*)\right|\right] &\leq \frac{1}{2}(M+1)\left((M-\tilde{k})\delta + \epsilon\left[\left(\frac{M}{r}\sum_{i\in D(r)} Q_{ii}\right)\sqrt{M\sum_{i=1}^{M} Q_{ii}^2}\right]^{\frac{1}{2}}\right) \\ &\leq \gamma + \frac{\epsilon}{2}(M+1)\left[\left(\frac{M}{r}\sum_{i\in D(r)} Q_{ii}\right)\sqrt{M\sum_{i=1}^{M} Q_{ii}^2}\right]^{\frac{1}{2}}.\end{aligned}$$

That is, the total error is the sum of the error γ obtained from the approximation of the quadratic programming problem in a subspace (Equation 3.13) and the error due to the Nyström method.

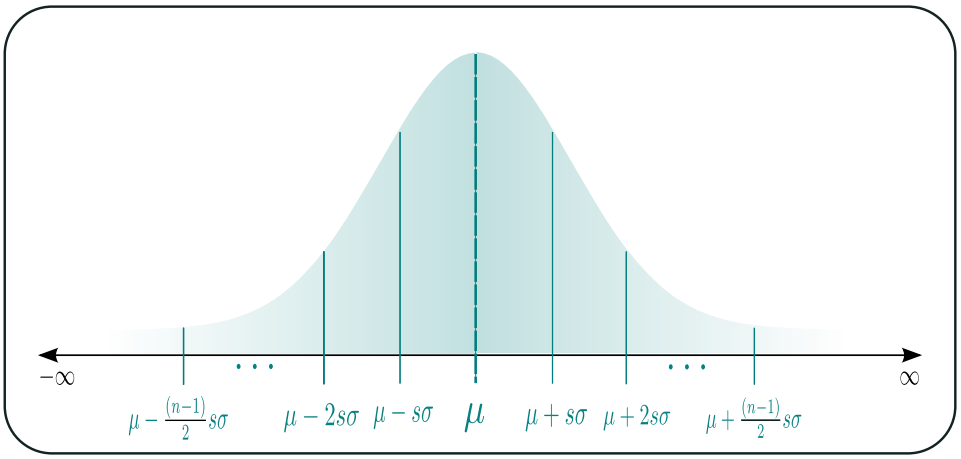


FIGURE 3.4: QPFS variable discretization for mutual information estimation. The variable is discretized in an odd number of segments, n, covering s standard deviations. μ is the sample mean of the training data and σ the standard deviation.

3.2.3 Theoretical Complexity

As already mentioned, the mRMR method is one of the most successful previous methods for multivariate filter feature selection. However, its scalability for high-dimensional data is compromised due to its computational cost depends quadratically on the dimension of the classification problem. In this section, an analysis of the theoretical complexity of mRMR and QPFS is carried out to conclude that the QPFS+Nyström method is able to improve the time complexity of the mRMR algorithm.

The time complexities of mRMR and QPFS both have two components, the time needed to compute the matrices Q and F *(Similarity)*, and the time needed to perform variable ranking *(Rank)*. The computational cost of calculating the correlation among a pair of variables is $O(N)$. The theoretical complexity of mutual information depends directly on how to estimate the density functions for continuous variables. For simplicity and following the mRMR methodology [31], each variable is discretized in an odd number of segments, n, covering s standard deviations as illustrated in Figure 3.4. In this case, mutual information can be approximated in a simple way using Equation 2.8. The cost of the discretization of each variable is $O(N)$ – corresponding to calculate its mean and its standard deviation. Once the attributes have been discretized, the marginal and joint probabilities for each pair of variables are obtained with cost $O(N)$.

In the case of mRMR and when the complete redundancy matrix Q is used by QPFS, the correlation or mutual information between each pair of variables and between each

variable with the target class are computed with cost $O(NM^2)$ and $O(NM)$ respectively, being the dominant term $O(NM^2)$. The QPFS+Nyström approximation needs also the vector F, $O(NM)$ but it only requires to know the submatrix $[A\ B]$ in Equation 3.10. Let $p \in (0, 1]$ the proportion of features randomly chosen by the Nyström method and $k = pM$ the number of subsampled features then, the submatrix $[A\ B]$ has k rows and M columns and thus, pM^2 entries need to be computed with a total cost of $O(NpM^2)$. Summing up, the predominant cost in the QPFS+Nyström similarity phase is given by $O(NpM^2)$.

The variable ranking is the process in which QPFS+Nyström advantages mRMR. The mRMR algorithm selects iteratively the *best* feature taking into account the subset of variables chosen in previous steps. Therefore, a linear search is performed at each iteration and M iterations are needed to provided a complete permutation of the variables, which means a total cost of $O(M^2)$. In the case of QPFS, the feature ranking implies the solution of a quadratic programming problem whose time cost is cubic in the feature space $O(M^3)$ [77]. Nevertheless, the cubic cost is an upper bound for the quadratic programming problem and it will be seldom achieved by the proposed model because (i) the QPFS algorithm generally solves the quadratic programming problem in a lower dimensional space and (ii) the matrix in the quadratic term is diagonal. The diagonalization of matrix Q can be performed by well-known algorithms like SVD or Jacobi [78] with cost $O(M^3)$. Finally, the cost of the Nyström diagonalization with k subsamples is $O(k^2M) = O(p^2M^3)$ [77] and then, the solution of the quadratic programming problem of rank k is, at the most, $O(k^3) = O(p^3M^3)$. Obviously, $p^2M^3 > p^3M^3$ thus, the ranking cost in QPFS+Nyström is $O(p^2M^3)$.

Note that the predominant cost in the ranking phase is the one associated with the Q diagonalization – regardless if the Nyström approach is used or not – which means that it is not worth trying to speed up the quadratic programing solver.

Table 3.2 summarizes the time cost of the three algorithms mRMR, QPFS and QPFS + Nyström indicating the most expensive phase, Similarity or Rank, in boldface. It can be concluded, thus, that the QPFS time complexity is greater than or similar to that of mRMR. However, in the case of the QPFS+Nyström its time complexity is the lowest when $N \gg pM$ and $N \gg M$. For $N \ll pM$, QPFS+Nyström outperforms mRMR provided that $p^2M^3 \ll NM^2$ or, equivalently, $N \gg p^2M$. For example, if $p = 10^{-1}$ then

	mRMR		**QPFS**		**QPFS+Nyström**	
	Similarity	Rank	Similarity	Rank	Similarity	Rank
M large $N \ll pM$	$\mathbf{O(NM^2)}$	$O(M^2)$	$O(NM^2)$	$\mathbf{O(M^3)}$	$O(NpM^2)$	$\mathbf{O(p^2M^3)}$
M medium $N \gg pM$	$\mathbf{O(NM^2)}$	$O(M^2)$	$O(NM^2)$	$\mathbf{O(M^3)}$	$\mathbf{O(NpM^2)}$	$O(p^2M^3)$
M small $N \gg M$	$\mathbf{O(NM^2)}$	$O(M^2)$	$\mathbf{O(NM^2)}$	$O(M^3)$	$\mathbf{O(NpM^2)}$	$O(p^2M^3)$

TABLE 3.2: Time complexity of algorithms as a function of training set size N, number of variables M,and Nyström sampling rate p. The predominant cost term is indicated in boldface.

QPFS+Nyström is more efficient than mRMR when $N \gg 10^{-2}M$ i.e. if the size of the training set is greater than 10^{-2} times the number of variables.

3.3 Experimental Results

Experimental results described in this section provide an extensive analysis of the QPFS performance divided into two subsections according to the twofold aim of the experiments: first, to compare classification accuracy achieved using QPFS versus other well-known filter techniques, especially mRMR; and second, to compare their computational cost.

The datasets used for all the experiments are shown in Table 3.3 and they were chosen because they are representative of multiple types of classification problems with respect to the number of samples, the number of features, and the achievable classification accuracy. Moreover, these datasets have been used in other research on the feature selection task [2, 31, 79–83].

In order to estimate classification accuracy, for the ARR, NCI60, SRBCT and GCM datasets 10-fold cross-validation (10CV) and 100 runs were used. Mean error rates are comparable to the results reported in [2, 31, 82, 83]. In the case of the RAT dataset, 120 training samples (61 for test) and 300 runs were used, following [79]. The MNIST dataset was divided into training and testing subsets as proposed by [53], with 60,000 and 10,000 patterns, respectively.

Dataset	N	M	C	Baseline Error Rate	References
ARR	422	278	2	21.81%	[31, 82]
NCI60	60	1,123	9	38.67%	[2, 82, 83]
SRBCT	83	2,308	4	0.22%	[2, 83]
GCM	198	16,063	14	33.85%	[2, 82, 83]
RAT	181	8,460	2	8.61%	[79]
MNIST	60,000	780	10	6.02%	[80, 81]

TABLE 3.3: Description of the datasets used in experiments. N is the number of examples, M is the number of variables, and C is the number of classes. Baseline error rate is the rate obtained taking into account all variables. The last column cites papers where the datasets have been used.

mRMR and QPFS were implemented in C using *LAPACK* for matrix operations [84]. Quadratic optimization is performed by the Goldfarb and Idnani algorithm implemented in Fortran and used in the R *quadprog* package [71]. A publicly available toolbox of QPFS can be found at

`http://sites.google.com/site/irenerodriguezlujan/documents/QPFS-1.0.zip`.

3.3.1 Classification Accuracy Results: Comparison with other filter feature selection methods

The aim of the experiments described in this section is to compare classification accuracy achieved by some state-of-the-art filter feature selection methods and by QPFS, with and without the Nyström approximation. The classification error is measured as a function of the number of features in order to determine the efficiency of the different feature selection approaches. Firstly, the two proposed similarity measures, correlation and mutual information, will be considered to conclude that, generally, mutual information outperforms correlation since its capability detecting nonlinear relationships among features. Then, the experiments will be focused on comparing QPFS with the greedy filter-type method mRMR in its difference form (MID), which also takes into account the difference between redundancy and relevance, and with the MaxRel algorithm that basically ranks features according to their similarity with the target class. Finally, other filter-type feature selection methods will be also evaluated in order to point out QPFS as a promising multivariate filter feature selection method.

Any of the versions of the QPFS algorithm requires to adjust the α parameter that weights the importance of the redundancy and relevance in the final model. The heuristics proposed in Equations 3.7 and 3.11 to avoid any dependence on the underlying classifier yield the α values shown in Table 3.4. Notice that high values of α are better for datasets with high redundancy among variables. On the other hand, if there is low redundancy then small α should yield better results. Other values of the α parameter, $\alpha \in \{0.0, 0.1, 0.3, 0.5, 0.7, 0.9\}$, were considered in all experiments in order to verify that the proposed method of setting α provides good results. Besides the α parameter, the QPFS+Nyström method needs to determine the Nyström sampling rate p; it was chosen as large as possible while still yielding a reasonable running time, since larger values reduce error in the approximation of the Q matrix (Equation 3.16).

Dataset	**p**	$\bar{\mathbf{q}}$	$\bar{\mathbf{f}}$	$\hat{\alpha}$
ARR (cor)	-	0.0889	0.0958	0.481
NCI60 (cor)	-	0.267	0.165	0.618
ARR (MI)	-	0.0106	0.0152	0.411
NCI60 (MI)	-	0.0703	0.253	0.217
SRBCT (MI)	-	0.0188	0.0861	0.179
GCM (MI)	0.05	0.0284	0.158	0.152
RAT (MI)	0.1	0.0346	0.0187	0.649
MNIST (MI)	-	0.0454	0.0515	0.469

TABLE 3.4: Values of the α parameter for each dataset. Correlation (cor) and mutual information (MI) were used as similarity measures for ARR and NCI60 datasets. Only mutual information was used for SRBCT, GCM, RAT and MNIST datasets. p is the subsampling rate in the Nyström method, $\bar{q}$ is the mean value of the elements of the matrix Q (similarity among each pair of features), and $\bar{f}$ is the mean value of the elements of the F vector (similarity of each feature with the target class). For the MNIST dataset only nonzero values have been considered for the statistics due to the high level of sparsity of its features (80.78% sparsity in average).

As stated in Section 2.2, filter feature selection methods can be used with any classifier thus, the learning algorithm can be considered as a *metaparameter* for any filter method. In all the experiments described in this section, the linear SVM model introduced in Section 2.3.2 and provided by the *LIBSVM* package [53] was used as underlying classifier. Of course, any other classification learning algorithm can be used instead of linear SVMs but results here presented are expected to be representative enough. The linear kernel has been chosen to reduce the number of SVM parameters, thus making meaningful results easier to obtain. In particular, the cost parameter C of the linear SVM needs to be adjusted. The influence of the value of the parameter on the comparison between mRMR and QPFS can be analyzed from Figure 3.5, where the performance of both

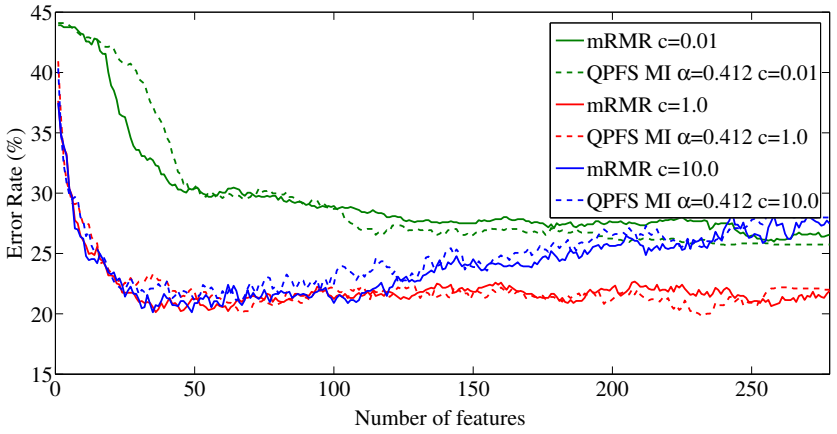
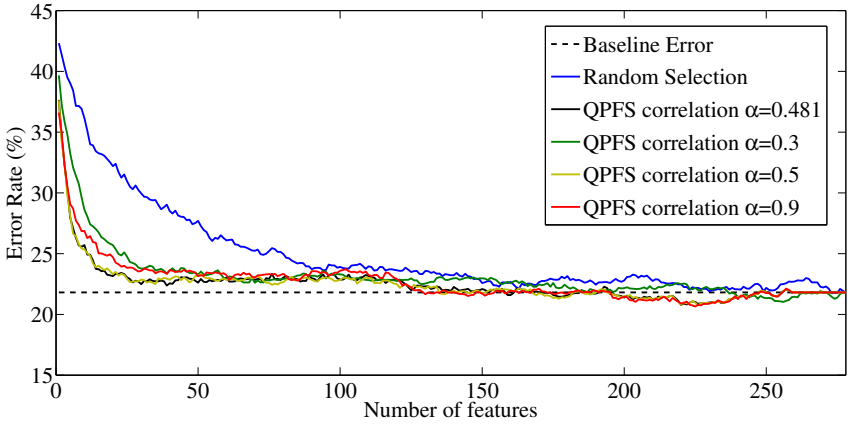


FIGURE 3.5: Classification error as a function of the number of features for the ARR dataset and different regularization parameter values C in linear SVM. The figure shows that for $C = 0.01$, the SVM is too regularized. The effect when $C = 10.0$ is the opposite and the SVM overfits the training data. A value of $C = 1.0$ is a good tradeoff.

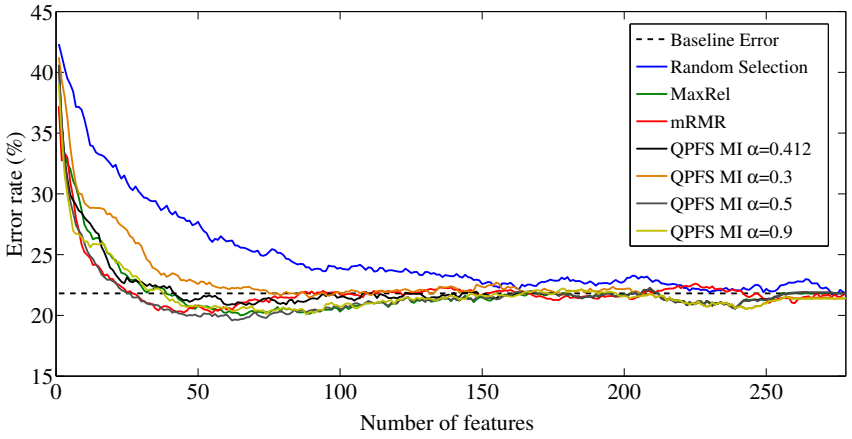
methods for the ARR dataset and different C values is shown. From Figure 3.5 it can be concluded that the C value does not affect in the comparison between filter feature selection models and therefore, C is set to 1.0 in all experiments.

The definition of the QPFS model admits any symmetric similarity measure as discussed in Section 3.2 and, depending on the context in which the feature selection method is going to be used, certain measures may be better than others. In this thesis, correlation and mutual information are used due to they are well established statistical measures of dependence. The mutual information was proposed as an alternative to the correlation to solve some deficiencies of the last one in detecting nonlinear relationships between features. In the first experiment, an empirical comparison between the performance of both similarity measures in the ARR dataset is accomplished. Figure 3.6 shows the average classification error rate for the ARR dataset as a function of the number of features. In Figure 3.6(a), correlation is the similarity measure while mutual information (MI) is applied in Figure 3.6(b). As a secondary objective, results from a non-informative feature selection method which does random selection of features are also provided in order to determine the absolute advantage of using any feature selection method. Several conclusions can be derived from the experimental results obtained in Figure3.6:

1. α **parameter**. In both cases the best accuracy is obtained with $\alpha = 0.5$, which means that an equal tradeoff between relevance and redundancy is better. However, accuracies using the values of α specified by the heuristic (Table 3.4) are



(a)



(b)

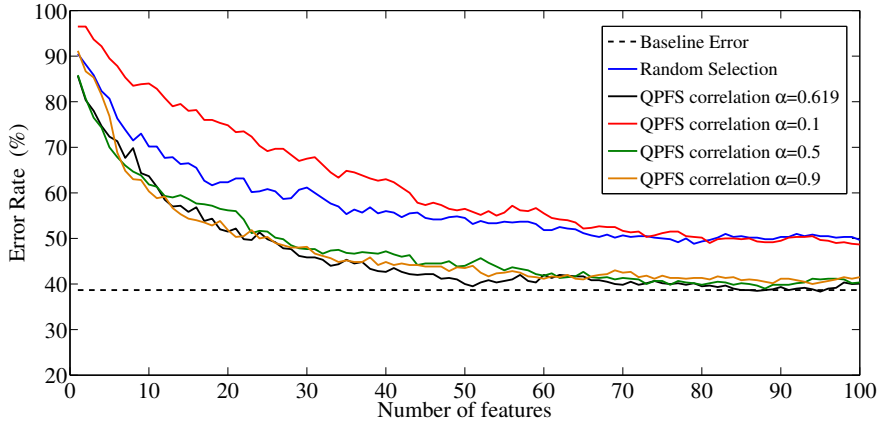
FIGURE 3.6: Classification error as a function of the number of features for the ARR dataset. 3.6(a) QPFS results using correlation as similarity measure with different α values. 3.6(b) MaxRel, mRMR and QPFS results using mutual information as similarity measure and different values of α for QPFS.

similar.

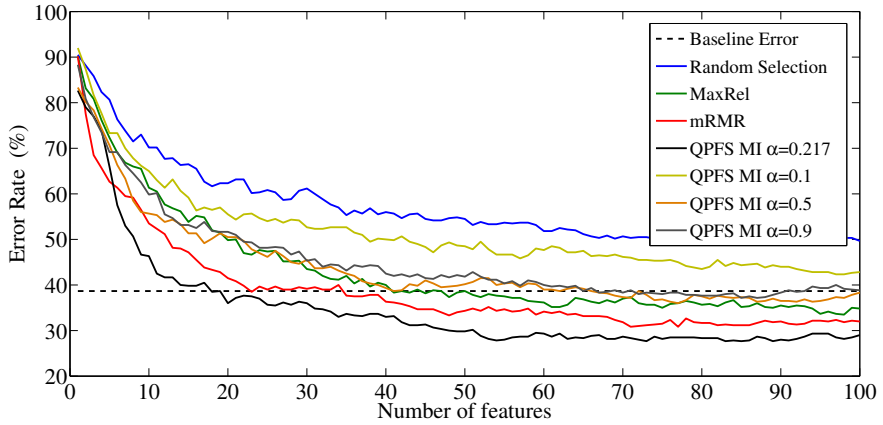
2. **Similarity measures**. The use of MI as similarity measure improves significantly the QPFS accuracy compared to the correlation performance.

3. **Comparison with mRMR**. When MI is used (Figure 3.6(b)) the error rate curve for $\alpha = 0.5$ is similar to that obtained with mRMR.

4. **Effectiveness of feature selection**. The random selection method yields results significantly worse than those obtained with the other algorithms, especially up to about 150 features.



(a)



(b)

FIGURE 3.7: Classification error as a function of number of features for the NCI60 dataset. 3.7(a) QPFS results using correlation as similarity measure with different α values. 3.7(b) MaxRel, mRMR and QPFS results using mutual information as similarity measure and different values of α for QPFS.

In order to check the truthfulness of these four statements, the previous experiment was reproduced for the NCI60 dataset (Figure 3.7). In this case, the best accuracy is obtained when mutual information is used (Figure 3.7(b)) and α is set to 0.217 according to Table 3.4; what is more, this accuracy is slightly better than the accuracy of mRMR. The value of α close to zero indicates that it is appropriate to give more weight to the quadratic term in QPFS. When correlation is used (Figure 3.7(a)), the best accuracy is also obtained when α is set according to the heuristic. Again, the performance of the random feature selection is poor compared with the best results of mRMR and QPFS.

Generally, MI as similarity measure leads to better accuracy than correlation. This finding is reasonable given that MI can capture nonlinear relationships between variables.

Therefore, MI is used in the experiments described in the remainder of this section. Furthermore, it is clear that information guided procedures are essential for the success of the feature selection task and hence, random feature selection is not included in the following experiments.

Up to now, the experiments have analyzed the performance of QPFS without the Nyström approximation but, as stated in Section 3.2.1, in despite of the novelty of the QPFS formulation, it does not represent any improvement in terms of computational time. The aim of the following experiments is to point out the influence of the Nyström method in the QPFS algorithm starting with the SRBCT dataset which illustrates not only the competitiveness of QPFS+Nyström in terms of classification accuracy, but also the dependence between the Nyström subsampling rate, p, and the approximate model. The average error for the SRBCT dataset and different sampling rates as a function of the number of selected features is shown in Figure 3.8. In order to simplify the figure, only results for the best α value in the grid $\{0, 0.1, 0.3, 0.5, 0.7, 0.9\}$, $\alpha = 0.1$, and the estimated $\hat{\alpha} = 0.179$ are shown providing both values similar results in terms of classification accuracy and comparable to those of mRMR. The fact that a low value of α is best indicates low redundancy among variables compared to their relevance with the target class. This scenario favors the hypotheses that the redundancy matrix can be approximated by a low-rank matrix efficiently computed by the Nyström method. Indeed, the QPFS+Nyström method gives a classification accuracy similar to that of QPFS when $p > 0.1$ which means computing only the redundancy among 10% of the variables with all the rest and solving the optimization problem in a subspace 10 times lower. Moreover, a remarkable result in Figure 3.8 is the empirical verification of the Nyström error bound given in Equation 3.16: the higher parameter p, the closer the Nyström approximation is to the complete diagonalization and thus to the QPFS solution.

Once the relationship between the subsampling rate and the effectiveness QPFS+Nyström has been verified, the computational savings of QPFS+Nyström in the high dimensional datasets GCM and RAT are presented. Figure 3.9 shows error rates for the GCM dataset using the algorithms MaxRel, mRMR, and QPFS+Nyström with $\alpha = 0.1$ and $\hat{\alpha} = 0.152$. When the number of features is over 60, accuracy achieved with QPFS+Nyström is better than with mRMR. A sampling rate of 3% is adequate for this dataset, which represents a major time complexity reduction given a feature space of $16,063$ variables. Looking at Table 3.2, the GCM dataset can be classified in the group of problems with M

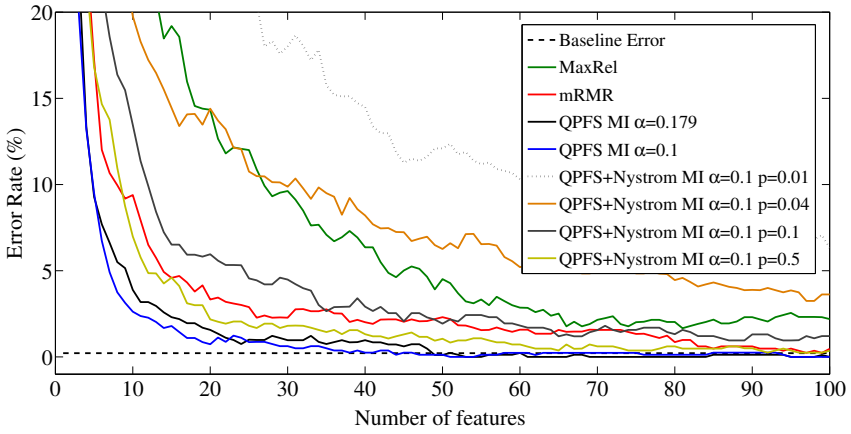


FIGURE 3.8: Error rates using MaxRel, mRMR and QPFS+Nyström methods, with mutual information as similarity measure for the SRBCT dataset.

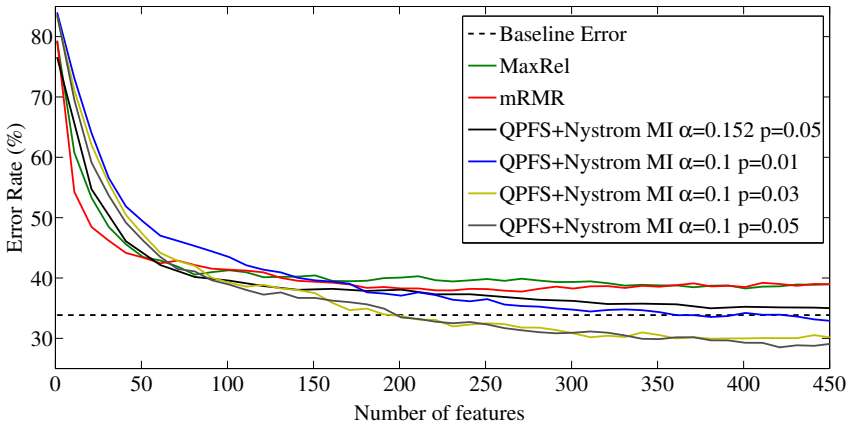


FIGURE 3.9: Error rates using MaxRel, mRMR and QPFS+Nyström methods, with mutual information as similarity measure for the GCM dataset.

large ($N \ll pM$) in which case mRMR has a cost of $NM^2 \approx 2 \cdot 10^2 (1.6 \cdot 10^4)^2 \approx 5.1 \cdot 10^{10}$ while the cost associated to QPFS+Nyström ($p = 0.03$) is one order of magnitude lower: $p^2M^3 \approx (3 \cdot 10^{-2})^2 (1.6 \cdot 10^4)^3 \approx 3.7 \cdot 10^9$. In the case of the RAT dataset (Figure 3.10), QPFS+Nyström gives classification accuracy similar to that of mRMR when the subset size is over 80 and the sampling rate is 10%. Given the good performance of the MaxRel algorithm for this dataset, it is not surprising that a large α value $\alpha = 0.9$ or $\hat{\alpha} = 0.649$ is best, considering also that QPFS with $\alpha = 1.0$ is equivalent to MaxRel. Again, the RAT dataset can be categorized in the first row of Table 3.2 and QPFS+Nyström accelerates mRMR: $p^2M^3 \approx 6.1 \cdot 10^9$ versus $NM^2 \approx 1.3 \cdot 10^{10}$. However, the good performance of the univariate filter points to MaxRel as the best option for this dataset.

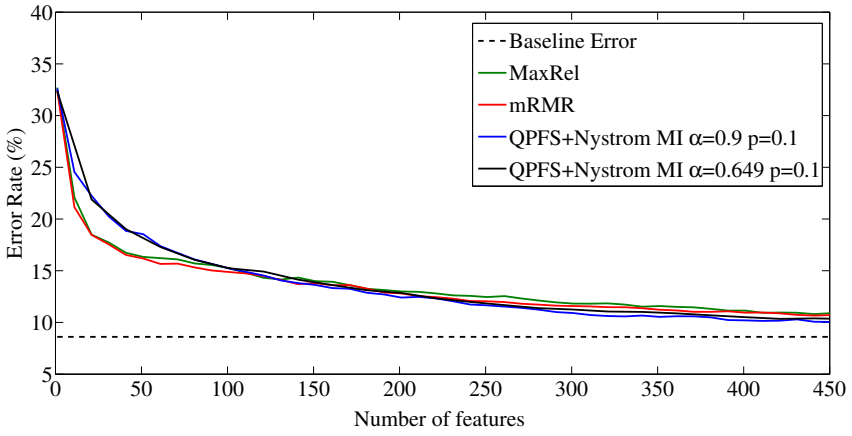


FIGURE 3.10: Error rates using MaxRel, mRMR and QPFS+Nyström methods, with mutual information as similarity measure for the RAT dataset.

Finally, an example in which the result of the feature selection can be visualized and can be easily interpreted is presented. The MNIST dataset has a high number of training examples $(60,000)$ each ofwhich represents a handwriting digit (0 to 9) in a grid of 780 pixels (features). Results for MNIST are shown in Figure 3.11 for QPFS with $\alpha = 0.3$, the estimation $\hat{\alpha} = 0.469$ and QPFS+Nyström with $\hat{\alpha}$ and $p \in \{0.1, 0.2, 0.5\}$. The error rate for all algorithms reaches a minimum when about 350 features are selected. This is not a surprising fact: analyzing the sparsity of the MNIST features, approximately 400 of them have a level of sparsity higher than 70%. But if the feature space needs a greater reduction, significant differences appears between the studied methods as shown in Figure 3.11. mRMR and QPFS with $\hat{\alpha} = 0.469$ have similar performance and close to the best results obtained by QPFS with $\alpha = 0.3$. For this dataset, the number of samples is much greater than the number of features, $N \gg M$, and therefore , the time complexity of mRMR and QPFS is the same: $O(NM^2)$. When QPFS+Nyström is applied with $p = 0.2$, the error rate is competitive and the MNIST provides an example of the ability of QPFS+Nyström to handle large datasets reducing the computational cost of mRMR and QPFS by a factor of 5. Note that the error rates shown for the MNIST dataset are obtained using a linear kernel. The radial basis function kernel for SVM classifiers is known to lead to lower error rates for the full MNIST dataset, but the choice of kernel is an issue separate from feature selection, which is the focus of this chapter.

To illustrate graphically the effects of QPFS feature selection, Figure 3.12 shows a grid of 780 pixels arrayed in the same way as the images in the MNIST datasets. A pixel is

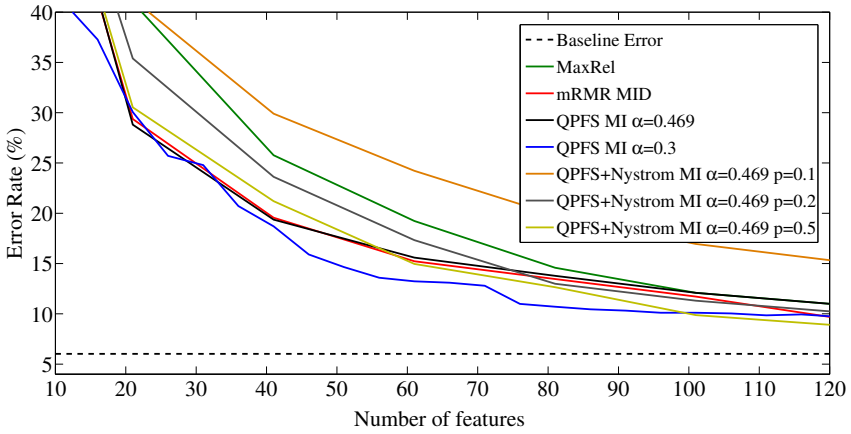
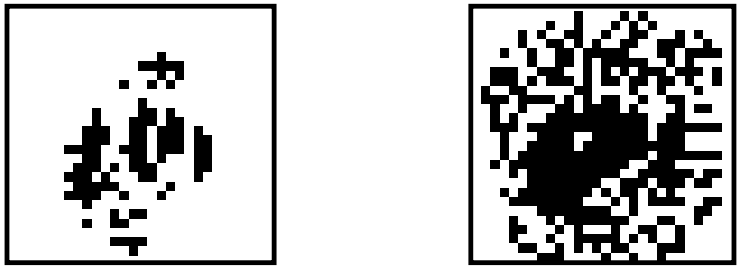


FIGURE 3.11: Error rates using MaxRel, mRMR and QPFS+Nyström methods, with mutual information as similarity measure for the MNIST dataset.



(a) (b)

FIGURE 3.12: First 3.12(a) 100 and 3.12(b) 350 features selected by QPFS+Nyström ($\hat{\alpha} = 0.469$ and $p = 0.5$) for the MNIST dataset (black pixels).

black if it corresponds to one of the top 100 (Figure 3.12(a)) and 350 (Figure 3.12(b)) selected features, and white otherwise. Black pixels are more dense towards the middle of the grid, because that is where the most informative features are. Pixels sometimes appear in a black/white/black checkerboard pattern, because neighboring pixels tend to make each other redundant.

In order to compare in-depth the mRMR and QPFS classification errors, the statistical significance of error rate differences is evaluated in Table 3.5. For each dataset, 100 classifiers were trained using the stated number M of selected features. The 100 classifiers arise from 10 repetitions of 10-fold cross-validation, so which M features are used may be different for each classifier. The one-tailed paired t-test for equal means is applied to the two sets of error rates, one set for mRMR and one set for QPFS. The test is one-tailed because the null hypothesis is that the mRMR method is as good or better than the QPFS method. The test is paired because both methods were applied to the

same 100 dataset versions. Results of the test are given in the row labeled *significant?*.

For the NCI60 and SRBCT datasets, the best result is obtained when QPFS is used and it is statistically significantly better than mRMR. When 200 to 400 variables are considered, mRMR and QPFS are not statistically significantly different but the accuracy is not as good as in the case of 100 features, probably due to overfitting. In the case of the GCM dataset, the mRMR method is statistically significantly better when fewer than 50 variables are considered. If the number of features is over 100, the accuracy with QPFS is significantly better than with mRMR, and the best performance is obtained in this case. For the ARR dataset, mRMR is statistically significantly better than QPFS if fewer than 10 features are considered but the error rate obtained can be improved if more features are taken into account. When more than 50 features are selected, the two methods are not statistically significantly different. The RAT dataset behavior is quite similar. When fewer than 100 features are used, the mRMR algorithm is statistically better than QPFS, but the error rate can be reduced adding more features. The two algorithms are not statistically significantly different in the other cases, except if more than 400 features are involved in which case QPFS is statistically significantly better than mRMR. Note that the error rates shown for QPFS are obtained with the proposed estimation of $\hat{\alpha}$. In some cases, as shown in Figures 3.6 to 3.10, this α value cannot be the best choice.

Beyond simple binary statistical significance, Table 3.5 indicates that the QPFS method is statistically significantly better when the value of $\hat{\alpha}$ is small. A possible explanation for this finding is the following. When $\hat{\alpha}$ is small, features are highly correlated with the label ($\bar{f} \gg \bar{q}$). The mRMR method is greedy, and only takes into account redundancy among features selected in previous iterations. When features are highly correlated with the label, then mRMR selects features with high relevance and mostly ignores redundancy. In contrast, QPFS evaluates all variables simultaneously, and always balances relevance and redundancy.

Once QPFS algorithm has been successfully compared against mRMR, it seems natural to extend the comparison to other other multivariate filter methods. The feature selection review developed in Section 2.2 point to mRMR quotient form (called MIQ), reliefF and SFS as some of the most satisfactory feature selection algorithms independent of the classifier. Again, this independence makes the applicability of any learning

	M				
	10	50	100	200	400
RAT					
mRMR	21.15 ± 0.31	16.18 ± 0.27	14.88 ± 0.24	12.81 ± 0.25	10.95 ± 0.23
QPFS $\hat{\alpha} = 0.65$	27.13 ± 0.33	18.16 ± 0.29	15.24 ± 0.27	12.85 ± 0.26	10.51 ± 0.21
significant?	no	no	no	no	yes
p value	1.00	1.00	0.89	0.56	1.7×10^{-2}
ARR					
mRMR	25.19 ± 0.65	20.76 ± 0.63	21.71 ± 0.61	21.64 ± 0.61	-
QPFS $\hat{\alpha} = 0.41$	28.05 ± 0.65	21.30 ± 0.65	21.52 ± 0.65	21.76 ± 0.58	-
significant?	no	no	no	no	-
p value	1.00	0.96	0.69	0.39	-
NCI60					
mRMR	53.50 ± 2.17	34.33 ± 1.74	32.00 ± 1.93	32.83 ± 1.84	33.64 ± 1.80
QPFS $\hat{\alpha} = 0.22$	46.33 ± 2.19	29.83 ± 1.68	29.00 ± 1.83	34.67 ± 1.81	35.17 ± 1.95
significant?	yes	yes	yes	no	no
p value	1.6×10^{-3}	7.5×10^{-3}	3.3×10^{-2}	0.95	0.96
SRBCT					
mRMR	9.38 ± 1.06	2.31 ± 0.51	0.47 ± 0.23	0.24 ± 0.17	0.49 ± 0.30
QPFS $\hat{\alpha} = 0.18$	3.89 ± 0.75	0.11 ± 0.11	0.05 ± 0.11	0.11 ± 0.11	0.35 ± 0.25
significant?	yes	yes	yes	no	no
p value	5.4×10^{-9}	5.6×10^{-5}	2.3×10^{-2}	0.27	0.36
GCM					
mRMR	54.26 ± 1.19	43.38 ± 1.18	41.38 ± 1.08	38.26 ± 1.06	38.50 ± 1.10
QPFS $\hat{\alpha} = 0.15$	65.66 ± 1.03	44.11 ± 1.11	39.57 ± 1.24	38.06 ± 1.16	35.23 ± 1.17
significant?	no	no	yes	no	yes
p value	1.00	0.81	0.037	0.40	1.42×10^{-4}

TABLE 3.5: Average error rates using the mRMR and QPFS methods, for classifiers based on M features. The parameter $\hat{\alpha}$ of the QPFS method is indicated; rows are ordered according to this value. The Nyström approximation was used for the GCM and RAT datasets.

algorithm possible but, from analogy with the previous experiments, linear SVMs with cost parameter $C = 1.0$ were chosen. Furthermore, ReliefF and SFS are feature selection methods which need to establish the value of some parameters. In ReliefF, all instances were considered (not random subsampling) and the number of neighbors was adjusted for all datasets resulting to be 3 for all datasets, except for MNIST in which 10 neighbors were taking into account. The ReliefF implementation provided by the LIBGS package [85] was used. In the case of the SFS algorithm, a Matlab implementation [30] with the default parameters was used.

Average error rates for MaxRel, mRMR (MID), mRMR (MIQ), reliefF and QPFS and

different number of features are shown in Table 3.6. The error rate for SFS is shown separately (Table 3.7) since SFS has been formulated for binary datasets being only suitable for the ARR and RAT datasets. In spite of the SFS original formulation includes a feature generation step [30], it has not been considered in the experiments to provide a fair comparison with the rest of methods. Table 3.6 shows that for ARR, NCI60, SRBCT and GCM datasets, the best selector is mRMR or QPFS (the statistical study of the performance of both methods has been given in Table 3.5). In the case of the RAT dataset, the best methods are MaxRel and reliefF. The fact that the best results are obtained with methods which only consider relevance with the target class fits in with the analysis of Figure 3.10. Finally, for the MNIST dataset the best choice is the mRMR (MIQ) algorithm. Nevertheless, the performance of MIQ in some datasets is not competitive (see, for instance, the ARR and NCI60 results). The accuracy of QPFS+Nyström ($p = 0.2$) is good if a high enough number of features is used, and it has lower computational cost than mRMR and QPFS. Regarding SFS, Table 3.7 shows that SFS provides a competitive error rate for the ARR dataset with few features (around 11) but its effectiveness in the RAT dataset is improved by other feature selection algorithms when more than 6 attributes are considered. It is noticeable the efficiency of SFS getting acceptable accuracies using a small number of features.

3.3.2 Computational Complexity Results

Since the previous section has established the superiority of mRMR and QPFS in terms of classification performance, it is useful now to compare mRMR and QPFS experimentally with respect to time complexity. Time complexity is measured as a function of the training set size (N), the dimensionality (M), and the Nyström sampling rate ($p = \frac{r}{M}$). In all cases, times are obtained as the average over 50 runs. In order to measure time complexity as a function of the training set size, the number of SRBCT examples was artificially increased 4 times ($N = 332$) and dimensionality reduced to $M = 140$. Time complexity as a function of the dimensionality and the Nyström sample rate was measured using the original SRBCT dataset. The results obtained from the threefold comparison are shown in Figure 3.13 and can be analyzed as follows.

- **Computational cost as a function of the number of patterns** N. As stated in Section 3.2.3, the running times of mRMR and QPFS with and without

Dataset	Method	M						
		10	20	40	50	100	200	400
ARR	MaxRel	27.48	24.68	21.70	20.82	**20.31**	21.73	-
	MID	**25.19**	**22.99**	**20.64**	**20.76**	21.71	**21.64**	-
	MIQ	29.79	27.78	23.89	23.32	21.53	21.74	-
	reliefF	30.64	24.48	21.54	21.34	20.90	21.66	-
	QPFS	28.05	23.72	22.39	21.30	21.52	21.76	-
NCI60	MaxRel	61.33	49.83	40.00	38.67	34.83	35.50	34.17
	MID	53.50	41.50	36.33	34.33	32.00	32.83	**33.67**
	MIQ	56.50	47.50	38.83	38.17	32.83	35.50	35.17
	reliefF	56.93	54.17	48.49	48.49	38.07	**32.13**	34.36
	QPFS	**46.33**	**36.00**	**33.00**	**29.83**	**29.00**	34.67	35.17
SRBCT	MaxRel	21.58	14.33	6.36	4.51	2.19	0.24	**0.13**
	MID	9.39	3.33	2.01	2.31	0.47	0.24	0.49
	MIQ	10.11	2.18	**0.47**	0.72	0.24	0.25	0.72
	reliefF	6.38	4.18	1.65	1.79	0.96	0.40	0.40
	QPFS	**3.89**	**1.57**	0.97	**0.11**	**0.05**	**0.11**	0.35
GCM	MaxRel	79.32	60.78	48.46	45.58	40.98	39.98	38.77
	MID	**54.26**	**48.45**	**44.16**	**43.38**	41.38	38.26	35.50
	MIQ	79.32	56.48	46.64	43.96	41.80	38.46	38.05
	reliefF	61.25	51.61	46.36	43.83	**39.35**	39.75	37.08
	QPFS+N $p = 0.05$	65.66	54.72	46.09	44.11	39.57	**38.06**	**35.26**
RAT	MaxRel	**19.95**	**17.32**	**15.40**	**15.16**	14.34	13.54	11.97
	MID	21.15	18.46	16.53	16.18	14.88	12.81	10.95
	MIQ	23.69	19.62	17.23	16.61	15.07	12.46	10.96
	reliefF	22.16	20.40	17.44	16.45	**13.68**	**11.43**	**9.85**
	QPFS+N $p = 0.1$	27.13	21.89	19.02	18.16	15.24	12.85	10.51
MNIST	MaxRel	59.19	40.98	25.77	22.5	12.09	7.64	6.72
	MID	53.39	29.37	19.56	17.40	11.72	7.55	6.66
	MIQ	51.69	**25.98**	**11.79**	**10.87**	**7.78**	**6.90**	**6.33**
	reliefF	**50.91**	40.20	23.81	19.56	12.31	8.47	6.86
	QPFS+N $p = 0.2$	57.00	35.39	23.62	20.48	11.31	7.71	6.54

TABLE 3.6: Error rates for different feature selection methods and Linear SVM. The best result in each case is marked in bold. QPFS+N indicates that the Nyström approximation is used in the QPFS method and p represents the subsampling rate in Nyström method. In all cases, the α parameter of QPFS is set to $\hat{\alpha}$.

Dataset	Number of Selected Features (average)	Error rate (%)
ARR	10.75 ± 0.155	23.34 ± 0.63
RAT	6.12 ± 0.13	22.87 ± 0.33

TABLE 3.7: Streamwise Feature Selection error rates.

Nyström all depend linearly on N when M and p are fixed. This theoretical dependence is confirmed in Figure 3.13(a), in which mRMR and QPFS show a

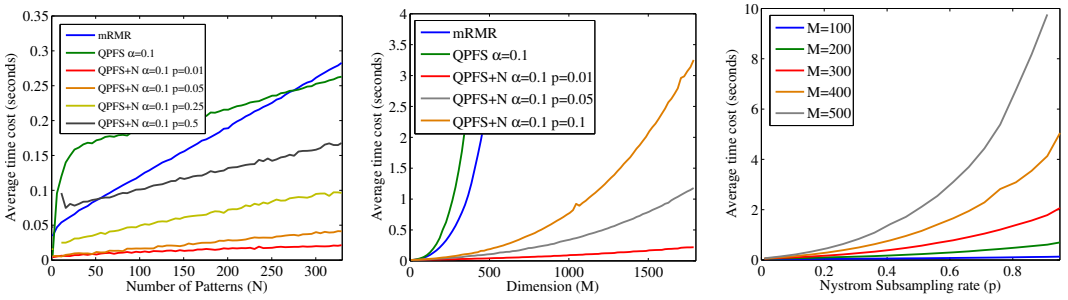
linear dependence on the number of patterns. For QPFS+Nyström, Table 3.2 shows that the slope of this linear dependence is proportional to the sampling rate p. Over the range $p = 0.01$ to $p = 0.5$, a decrease in p leads to a decrease in the slope of the linear dependence on N. Therefore, although all algorithms are linearly dependent on N, the QPFS+Nyström is computationally the most efficient. The time cost advantage increases with increasing number of training examples because the slope is greater for mRMR than for QPFS.

- **Computational cost as a function of the original dimension** M. Table 3.2 shows that mRMR and QPFS have quadratic and cubic dependence on M, respectively. However, the QPFS+Nyström cubic coefficient is proportional to the square of the sampling rate. When small value of p are sufficient,which is the typical case, the cubic terms are not dominant. These results are illustrated in the experiments shown in Figure 3.13(b). As expected from Table 3.2, mRMR and QPFS empirically show quadratic and cubic dependence on problem dimension. QPFS+Nyström shows only quadratic dependence on problem dimension, with a decreasing coefficient for decreasing p values. In all cases, a t-test has been used to verify the order of the polynomial that best fits each curve by least-squares fitting [86]. Overall, for small Nyström sampling rates, QPFS+Nyström is computationally the most efficient.

- **Computational cost as a function of the Nyström subsmapling rate** p. Table 3.2 reveals that there should be a quadratic dependence on sampling rate for the QPFS+Nyström algorithm. Figure 3.13(c) shows the empirical average time cost for the SRBCT dataset as a function of the sampling rate p. As expected, there is quadratic dependence on p and cubic dependence on the problem dimension M.

3.4 Discussion

Among machine learning tasks, identifying a subset of features with a competitive classification accuracy is a problem of growing importance because of the increasing size and dimensionality of real-world datasets. This responsibility fall on feature selection methods which can be divided into three groups according to their dependence with



(a) (b) (c)

FIGURE 3.13: Time cost in seconds for mRMR and QPFS as a function of: 3.13(a) the number of patterns, N; 3.13(b) the dimension, M; and 3.13(c) the sampling rate, p. QPFS+N indicates that the Nyström approximation is used in the QPFS method.

the classifier algorithm. Among them, filter approaches are often preferred in large-scale contexts due to its scalability and independence of the underlying classifier. The most scalable filter-type approaches are the univariate methods that only consider the discriminative power of variables individually. Unfortunately, their classification rates are generally poor, being necessary to resort to multivariate filters which also detect redundancies among features.

The chapter has started with the analysis of the scalability of the most important state-of-the-art multivariate filter algorithms to show that computational complexity improvements are still desirable in high-dimensional and large-scale domains. In this regard, a new multivariate filter feature selection method for multiclass classification problems has been presented and studied. The new method, named Quadratic Programming Feature Selection (QPFS), is based on the optimization of a quadratic function that is reformulated in a lower-dimensional space using the Nyström approximation (QPFS+Nyström).

Experimental results show that QPFS classification accuracy is similar to those of mRMR when mutual information used, and it yields slightly better results if there is high redundancy. Both methods outperform several well-known filter feature techniques considered in this chapter. In all experiments, QPFS using mutual information obtains better classification accuracy than correlation, presumably because mutual information better captures nonlinear dependencies among attributes. With respect to the computational complexity, small sampling rates in the Nyström method still lead to reasonable approximations of exact matrix diagonalization, sharply reducing the time complexity of QPFS. In fact, the QPFS+Nyström method using either Pearson correlation coefficient

or mutual information as similarity measure is computationally more efficient than the leading previous method mRMR.

In summary, the new QPFS+Nyström method presented in this chapter establishes a new formulation of the feature selection task capable of reducing the computational cost of state-of-the-art multivariate filter-type algorithms while maintaining their classification effectiveness.

Chapter 4

Kernel Quadratic Programming Feature Selection (KQPFS)

Feature selection techniques presented in Chapter 3 do not perform well if the original features do not contain explicitly the discriminatory information. In these cases, finding a proper representation of data is fundamental for the success of the feature selection task. However, obtaining a good representation entails not only removing irrelevant features, but also deriving new attributes as well. To tackle this problem, several well-known feature selection and extraction methods has been extended to perform the **feature selection/extraction in a kernel space** using a specially designed mapping function which implicitly generates nonlinear combinations of the original features –in a similar way as it was done in the context of Support Vector Machines in Section 2.3.2.1–. Following this approach, this chapter presents the second contribution of this thesis: the reformulation of the feature selection method proposed in Chapter 3 in a kernel space resulting the **Kernel Quadratic Programming Feature Selection algorithm** [87]. The chapter starts with a general overview about the application of the kernel trick to feature extraction algorithms and the case of the Fisher Discriminant Analysis introduced in Section 2.3.1 is studied in depth. Afterwards, the kernelized version of the QPFS method is formulated to find the projection direction which maximizes the quadratic objective function of QPFS in the kernel space. This direction turns out to be the well-known **Kernel Fisher vector** and the theoretical proof of such equivalence is given. The subsequent analysis of the computational complexity of the standard

Kernel Fisher Discriminant Analysis solution and the new solution derived from KQPFS reveals that the latest can be more efficient for highly unbalanced datasets. Finally, the experiments carried out at the end of the chapter corroborate the preceding theoretical results.

4.1 Feature Selection in a Kernel Space

Feature selection methods choose subsets of features in the original feature space and, as shown in Chapter 3, they can achieve good results in several domains. However, there are some cases in which it is not enough to detect relationships between the original features and the class and the dimensionality reduction problem requires to go one step further. As example, Figure 4.1 shows a synthetic dataset in which the class of each point is determined by its euclidean norm: a point belongs to the blue class if its distance to the origin $(0, 0)$ is lower than 0.5; otherwise, it is assigned to the red class. The symmetry of the problem makes that any reasonable feature selection algorithm considers any of the original features, X_1 and X_2, equally important and thus, dimensionality reduction maintaining the separability of the original data is not possible. Nevertheless, with a single variable $Z = X_1^2 + X_2^2$ both classes could be distinguished perfectly. The limited scope of feature selection methods is extended by feature extraction techniques. Among those, Principal Component Analysis (PCA) [1] or FDA (Section 2.3.1) are widely known in machine learning but they fail in cases as the one illustrated in Figure 4.1 as they only take into account linear combinations of the original features. Nonlinear feature extraction methods are more general approaches since they are capable of generating nonlinear combinations of the original features.

Nonlinear feature extraction can be tackled from two opposite points of view (Figure 4.2). The first approach consist in introducing a **set of nonlinear functions acting directly on the samples** and then, applying a feature selection/extraction method. In turn, the nonlinear functions can be established beforehand, for example as radial basis functions (RBF), or they can be extracted from certain grammar established by the user [88, 89]. The grammar makes it possible to add some prior information in the feature extraction process and it favors the understanding of the new attributes. However, grammatical evolution techniques are computationally expensive in terms of run time and memory allocation because they require to generate *explicitly* the new features. Therefore, its

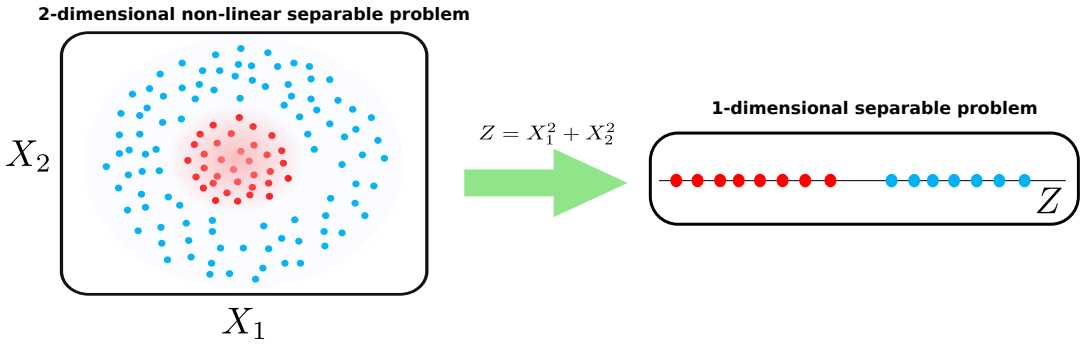


FIGURE 4.1: Bidimensional (x_1, x_2) synthetic dataset where feature selection methods performs poorly while nonlinear feature extraction methods can make the problem separable with a single feature $Z = X_1^2 + X_2^2$.

direct application to large-scale and high-dimensionality datasets is unfeasible and more effort is needed to make this alternative viable.

As a second approach, kernel methods for feature extraction **map the data from an original space to a kernel space** (by means the nonlinear function Φ generally not known) in which the feature selection is carried out corresponding to a nonlinear function in the input space. Section 2.3.2.1 presented the use of the kernel trick in the context of SVMs yet its application goes beyond and the most popular feature extraction methods like PCA and FDA have been reformulated in a kernel space. However, the kernelization of traditional feature selection techniques is not as straightforward as in the case of feature extraction methods since it entails several difficulties:

- **The mapping function Φ is generally not known** which makes it impossible to have an explicit list of the new features and select them using any of the *traditional* feature selection methods studied in Chapters 2 and 3.
- **The dimensionality of the feature space can be potentially infinite** thus, it makes no sense to rank features according to some criterion.
- In those cases in which the dimensionality of the feature space is not infinite and the new features could be obtained explicitly (i.e., polynomial kernel), **the dimension of the new feature space is so high** that even the most efficient feature selection algorithms become impracticable.
- The most commonly used similarity measures as correlation or mutual information can not be expressed as dot-products in the kernel space, forcing the **search of**

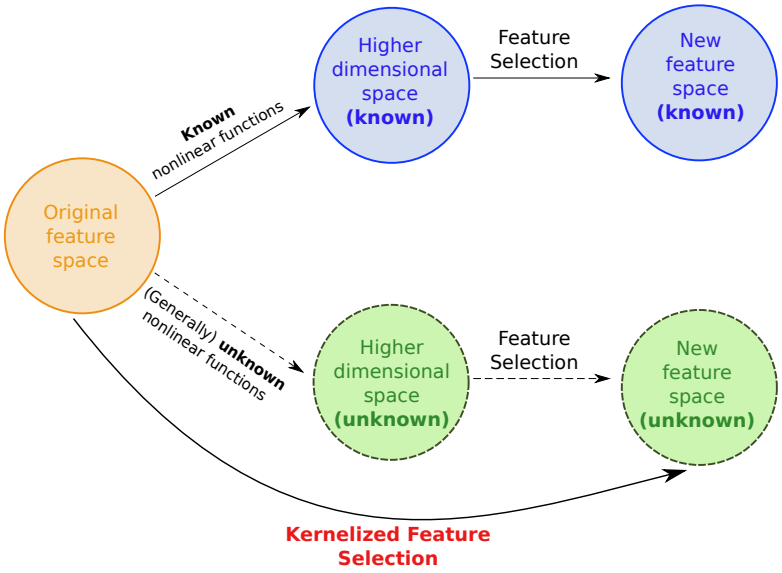


FIGURE 4.2: Differences between feature selection in the space generated either by a set of known nonlinear function or by a kernel.

a basis set in the kernel space. But, when the data is not sufficient or the dimension of the kernel space is infinite, not all the basis in the kernel space can be expressed by combinations of the data. Kernel Gram-Schmidt [90] or Kernel PCA [91] has been used to find such basis leading to an increment in the computational cost of the algorithm.

In despite of these limitations, several of the filter feature selection methods enumerated in Table 3.1 have their kernelized version. Those algorithms based on the computation of correlation or mutual information as similarity measure (MaxRel, CFS, mRMR) can be reformulated in the kernel space by finding a basis in the kernel space. ReliefF has also been adapted to the kernel space in combination with the Kernel Gram-Schmidt algorithm [90, 92]. In the case of wrapper methods, the kernelization must be done in the underlying classifier used to evaluate the performance of every feature subset. Basically, this group is formed by any backward or forward algorithm using the nonlinear SVM performance as quality measure. Again, for large-scale and high dimensional domains, the evaluation of a nonlinear SVM for every candidate subset is too expensive. Finally, the kernelization of embedded feature selection methods has been focused on feature selection in SVMs. This topic has been widely studied in the literature and several approaches has been suggested [93, 94]. In particular, Wu et al. proposed Sparse Large Margin Classifiers which find sparse SVMs classifiers consisting in finding sparse SVMs

that usually have better generalization ability than SVMs models in which sparsity is not enforced. As any embedded method, the main drawback of these models is the lack of versatility to be used with other classifiers.

4.1.1 Kernel Fisher Discriminant Analysis

The Fisher Discriminant Analysis expounded in Section 2.3.1 is probably the most popular supervised feature extraction techniques in machine learning. Although relying on heavy assumptions which are not true in many applications, Fisher Discriminant Analysis has proven to be very powerful and it is a point of reference in many machine learning systems due to its appealing properties:

1. The existence of a global solution and the absence of a local minima.
2. The solution has a closed form.
3. Its clear interpretation.

However, the analysis carried out in the previous section still holds here: for most real-world data, a linear discriminant is not complex enough. To tackle this problem, two different approaches can be considered:

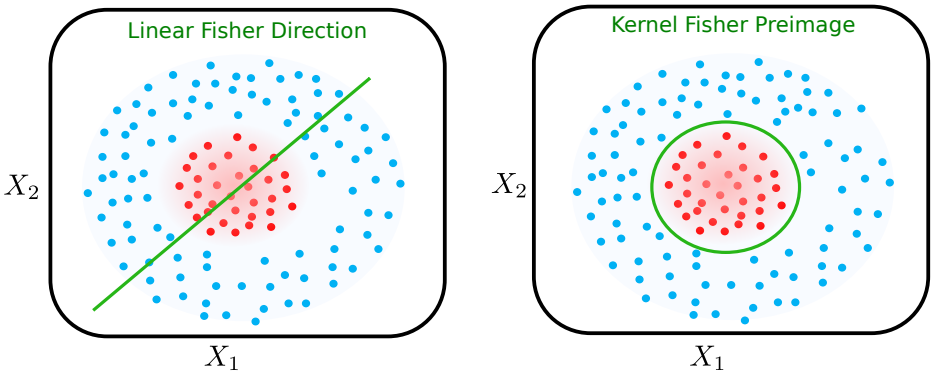
1. **Use more sophisticated distributions –instead of gaussian distributions– to model the data**. Assuming general distributions often sacrifices the simple and closed form and it requires more complex solvers, which is not desirable for large-scale problems.
2. **Look for nonlinear directions**. These nonlinearities can be in turn introduced *explicitly* or *implicitly*.
   - **Introducing explicit nonlinearities**. A set $\mathcal{D}$ of predefined nonlinear functions $\Phi_i(\boldsymbol{x})$ acting directly on the samples is considered: $\mathcal{D} = \{\Phi_i : \mathcal{X} \to \mathbb{R}\}$. Then, the optimal discriminant is defined as,

$$f(\boldsymbol{x}) = \sum_{\Phi_i(\boldsymbol{x}) \in \mathcal{D}} w_i \Phi_i(\boldsymbol{x}) + b. \tag{4.1}$$

These models produce nonlinear functions and hence, they are known as generalized linear discriminants. Taking advantage of the linearity of f with respect to the parameters $\boldsymbol{w}$ and b, Equation 4.1 can be optimized via least squares fitting. Well-known examples following this approach are radial-basis function (RBF) networks [95], where nonlinear functions have the form $\Phi_i(\boldsymbol{x}) = \exp(-\|\boldsymbol{x_i} - \boldsymbol{x}\|^2/\sigma)$ with $\boldsymbol{x_i}$ a fixed pattern and $\sigma \geq 0$. If σ parameter is also optimized, the function f is not linear anymore and methods like gradient descent should be applied. Many other works extending the classical linear discriminant framework can be found in the literature. These methods are based on the introduction of special nonlinearities, the imposition of certain types of regularization and the use of special optimization algorithms [42]. While an appropriate choice of functions $\mathcal{D}$ could approximate any function with an arbitrary precision, the presence of parameters to be tuned in the basis functions (multilayer neural networks) complicates the optimization problem, which is often prone to local minimal and lack for an intuitive interpretation.

- **Introducing implicit nonlinearities**. Instead of hand-crafting a nonlinear preprocessing, kernel functions are introduced. The idea introduced by Mika et al. [96] suggests an efficient way to increase the expressiveness of the discriminant via nonlinear directions applying the kernel-trick (Section 2.3.2.1): first map the data nonlinearly into some feature space $\mathcal{F}$ and compute the Fisher Discriminant Analysis there, which yields a nonlinear discriminant in the original input space. Being still the solutions hardly interpretable, this new formulation is able to find closed form solutions while maintaining the theoretical beauty of Fisher discriminant providing the model with flexibility by means of the use of different kernels. The rest of this section is directed to explain in detail the **kernelization of the Fisher discriminant**, closely related with the QPFS kernelization.

Formalizing the idea given by Mika et al. [96], let $S_+ = \{\boldsymbol{x_1^+}, \ldots, \boldsymbol{x_{N_1}^+}\}$ and $S_- = \{\boldsymbol{x_1^-}, \ldots, \boldsymbol{x_{N_2}^-}\}$ be samples from two different classes, $\boldsymbol{x_i} \in \mathbb{R}^M$ and $S = S_+ \cup S_-$ the complete set of N ($N = N_+ + N_-$) training samples. And let $\boldsymbol{y} \in \{-1, 1\}^N$ be the vector with the corresponding labels. Let $\Phi : \mathbb{R}^M \longrightarrow \mathcal{F}$ be the mapping function to the kernel space and $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \Phi(\boldsymbol{x_i}) \cdot \Phi(\boldsymbol{x_j})$ the Mercer kernel which defines the dot-product



(a) (b)

FIGURE 4.3: Illustrating the performance of Fisher Discriminant in a nonlinear synthetic example. Figure 4.3(a) shows the projection direction of the Fisher discriminant in the original space. Figure 4.3(b) shows the preimage in the original space of the Kernel Fisher Discriminant.

in $\mathcal{F}$. To find the Fisher Discriminant Analysis in $\mathcal{F}$, the following quantity must be maximized[1],

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w^T} S_B^{\Phi} \boldsymbol{w}}{\boldsymbol{w^T} S_W^{\Phi} \boldsymbol{w}} \tag{4.2}$$

where $\boldsymbol{w} \in \mathcal{F}$ and S_B^{Φ} and S_W^{Φ} are the corresponding between and within scatter matrices in $\mathcal{F}$, i.e.

$$\begin{aligned} S_B^{\Phi} &= (\boldsymbol{\mu}_+^{\boldsymbol{\Phi}} - \boldsymbol{\mu}_-^{\boldsymbol{\Phi}})(\boldsymbol{\mu}_+^{\boldsymbol{\Phi}} - \boldsymbol{\mu}_-^{\boldsymbol{\Phi}})^T \\ S_W^{\Phi} &= \sum_{\boldsymbol{x} \in S_+} (\Phi(\boldsymbol{x}) - \boldsymbol{\mu}_+^{\boldsymbol{\Phi}})(\Phi(\boldsymbol{x}) - \boldsymbol{\mu}_+^{\boldsymbol{\Phi}})^T + \sum_{\boldsymbol{x} \in S_-} (\Phi(\boldsymbol{x}) - \boldsymbol{\mu}_-^{\boldsymbol{\Phi}})(\Phi(\boldsymbol{x}) - \boldsymbol{\mu}_-^{\boldsymbol{\Phi}})^T \end{aligned}$$

with $\boldsymbol{\mu}_+^{\boldsymbol{\Phi}} = \frac{1}{N_+} \sum_{\boldsymbol{x} \in S_+} \Phi(\boldsymbol{x})$ and $\boldsymbol{\mu}_-^{\boldsymbol{\Phi}} = \frac{1}{N_-} \sum_{\boldsymbol{x} \in S_-} \Phi(\boldsymbol{x})$. Finding a solution to Equation 4.2 in the kernel space $\mathcal{F}$ requires to reformulate it in terms of only dot products of the input patterns. Every solution $w \in \mathcal{F}$ can be written as an expansion in terms of the mapped training data $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \Phi(\boldsymbol{x_i})$ making the things tractable. Consider the symmetric operators $\mathcal{S}$ on the finite-dimensional subspace spanned by the $\Phi(\boldsymbol{x_i})$ in a feature space $\mathcal{F}$ –possibly infinite–. For example, any matrix S constructed as a linear combination of $\{\Phi(\boldsymbol{x_i})\}$ as matrices S_B and S_W. Writing $\boldsymbol{w} = \boldsymbol{v_1} + \boldsymbol{v_2}$ as the decomposition of the part $\boldsymbol{v_1}$ lying in the subspace spanned by the training data and

[1]Observe the similarity between the following expressions and the objective function of Fisher Discriminant Analysis given by Equations 2.13–2.15.

$\boldsymbol{v_2}$ lying in its orthogonal subspace, it can be shown that it is sufficient to consider the part of the quadratic form $\boldsymbol{w^T} S\boldsymbol{w}$ which lies in the span of the examples,

$$\langle \boldsymbol{w}, S\boldsymbol{w} \rangle = \langle (\boldsymbol{v_1} + \boldsymbol{v_2}), S(\boldsymbol{v_1} + \boldsymbol{v_2}) \rangle = \langle (\boldsymbol{v_1} + \boldsymbol{v_2}), S\boldsymbol{v_1} \rangle = \langle \boldsymbol{v_1}, S\boldsymbol{v_1} \rangle.$$

These equalities use the properties $S\boldsymbol{v_2} = 0$ and $\langle \boldsymbol{v}, S\boldsymbol{v} \rangle$. Moreover, as $\boldsymbol{v_1}$ lies in the span of $\{\Phi(\boldsymbol{x_i})\}$ and S, by construction, only operates in this subspace, $S\boldsymbol{v_1}$ lies in the subspace spanned by $\{\Phi(\boldsymbol{x_i})\}$ as well.

The formulation of $\boldsymbol{w}$ as the linear combination of the mapped patterns, $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \Phi(\boldsymbol{x_i})$ makes it possible to write Equation 4.2 in terms of dot products in the kernel space as follows,

$$J(\boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha^T} B \boldsymbol{\alpha}}{\boldsymbol{\alpha^T} W \boldsymbol{\alpha}} \tag{4.3}$$

being

$$B = (B_+ - B_-)(B_+ - B_-)^T \qquad W = K_+(I_{N_+} - 1_{N_+})K_+^T + K_-(I_{N_-} - 1_{N_-})K_-^T$$

$$(B_+)_j = \frac{1}{N_+} \sum_{k=1}^{N_+} K(\boldsymbol{x_j}, \boldsymbol{x_k^+}) \qquad (B_-)_j = \frac{1}{N_-} \sum_{k=1}^{N_-} K(\boldsymbol{x_j}, \boldsymbol{x_k^-})$$

where K_+ is a $N \times N_+$ matrix with $(K_+)_{nm} = K(\boldsymbol{x_n}, \boldsymbol{x_m^+})$, I_{N_+} is the identity matrix in $\mathbb{R}^{N_+ \times N_+}$ and 1_{N_+} is a square matrix in $\mathbb{R}^{N_+ \times N_+}$ with all entries equal to $\frac{1}{N_+}$. Definitions for the negative class are analogous. Similarly to how the linear Fisher direction is obtained in the Fisher Discriminant Analysis, $\boldsymbol{\alpha}^{\text{KFDA}}$ can be worked out by finding the leading eigenvector of $W^{-1}B$ or by computing $\boldsymbol{\alpha}^{\text{KFDA}} = W^{-1}(B_- - B_+)$. However, the proposed model is **ill-posed** [15]: the N-dimensional covariance matrix has been estimated with N patterns which cause matrix W not to be positive[2]. Besides the

[2]Let $W = KDK^T$ with $D = I - \boldsymbol{v_1}\boldsymbol{v_1^T} - \boldsymbol{v_2}\boldsymbol{v_2^T}$, $\boldsymbol{v_j} \in \mathbf{R}^N$ and $(v_+)_i = \frac{1}{\sqrt{N_+}}$ if the example i belongs to the positive class and zero otherwise. The same applies for negative samples. $\text{Rank}(W) \leq \min\{\text{Rank}(K), \text{Rank}(D)\}$. Matrix K is usually full rank in practice but anyway, the rank of matrix D is $N - 2$ spanning $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ the null space of D.

numerical problems, successful learning requires to control the size and complexity of the classification model as stated in Section 2.1. While the Fisher discriminant is rather simple, its reformulation in a kernel space admits a wide variety of nonlinear solutions and the incorporation of a **regularization term** improves the generalization capability of the model. Regularization functions as $\|\boldsymbol{\alpha}\|^2$, $\|\boldsymbol{w}\|^2$ and others have been proposed in [96–98]. The KFDA authors propose to approximate matrix W by $W_\mu = W + \mu_W I_N$ with $\mu_W \geq 0$ and, as it will be shown later, this regularization is equivalent to regularize $\|\boldsymbol{\alpha}\|^2$. This kind of regularization can be viewed from different perspectives:

- If μ_W is large enough (ideally, the minimum value which makes W positive definite), the problem in Equation 4.3 is feasible and numerically more stable.
- As μ_W is increased, the variance of the W estimation decreases. In fact, when $\mu_W \to \infty$ the solution lie in the direction of $(B_- - B_+)$.
- The regularization $\|\boldsymbol{\alpha}\|^2$ is a l_2-regularization which favors solutions with small expansion coefficients.

Regularization on $\|\boldsymbol{w}\|^2$ is achieved by adding a multiple of the kernel matrix to W: $W_\mu = W + \mu_K K$ $(\mu_K \geq 0)$. This regularization resembles Support Vector Machines studied in Chapter 2 (Section 2.3.2) and indeed it can be formulated as a Least Squares SVM problem [99].

The question now is how to project a point into the direction given by the coefficients $\boldsymbol{\alpha}^{\text{KFDA}}$ if $\boldsymbol{w}^{\text{KFDA}}$ is not explicitly known (recall that it has been assumed that $\boldsymbol{w}^{\text{KFDA}} = \sum_{i=1}^{N} \alpha_i^{\text{KFDA}} \Phi(\boldsymbol{x_i})$). The dependence on Φ must be removed and the projection of a point need to be expressed as a function of the kernel matrix. More precisely, let $\tilde{\boldsymbol{x}}$ a point in the original input space, its projection according to the KFDA optimal direction is a one-dimensional feature given by,

$$\begin{aligned} \tilde{z} &= \boldsymbol{w}^{\text{KFDA}} \cdot \Phi(\tilde{\boldsymbol{x}}) = \left(\sum_{i=1}^{N} \alpha_i^{\text{KFDA}} \Phi(\boldsymbol{x_i}) \right) \cdot \Phi(\tilde{\boldsymbol{x}}) = \sum_{i=1}^{N} \alpha_i^{\text{KFDA}} \left(\Phi(\boldsymbol{x_i}) \cdot \Phi(\tilde{\boldsymbol{x}}) \right) \\ &= \sum_{i=1}^{N} \alpha_i^{\text{KFDA}} K(\boldsymbol{x_i}, \tilde{\boldsymbol{x}}). \end{aligned}$$

Regarding the synthetic example given in Figure 4.1, Figure 4.3(a) shows the projection direction computed via FDA which does not improve the discriminatory information in the data. By contrast, the KFDA boundary shown in Figure 4.3(b) separates linearly the input data with a single feature. Then, why not use always KFDA to ensure the most general model? The answer is simple, while the projection of a point to the linear Fisher direction only requires to evaluate one dot product in the input space, the projection of a point in the kernel space requires as many kernel evaluations as training samples and weighting each evaluation by $\boldsymbol{\alpha}^{\mathrm{KFDA}}$. In short, the evaluation of the kernel Fisher discriminant is so computationally intensive that its use is only justified in those cases in which linear feature extraction is insufficient. In addition to its high computational cost and unlike its linear counterpart, when KFDA is used it may not be possible to find an exact preimage of a reconstructed pattern. And, as in FDA, KFDA for binary problems project the data into one dimensional space which may be not enough for complex classification problems.

4.2 The QPFS Kernelization

Having the kernelization of popular methods like PCA and FDA in mind, it seems natural to try to adapt QPFS to a kernel space in order to extract nonlinear features which can reveal the discriminatory information. Despite the difficulty –even the impossibility– of getting the preimage of the nonlinear extracted features, the kernelization of PCA and FDA is able to maintain the theoretical basis of the original linear methods while, at the same time, to provide enough flexibility with the use of different kernels. Recall the **QPFS method** proposed in Chapter 3 consists in minimizing a multivariate quadratic function subjected to linear constraints as follows

$$\begin{aligned} \min_{\boldsymbol{w}} \quad & \tfrac{1}{2}\boldsymbol{w}^T Q \boldsymbol{w} - F^T \boldsymbol{w} \\ \text{s.t.} \quad & w_i \geq 0 \qquad \forall i = 1 \dots M \\ & \|\boldsymbol{w}\|_1 = 1, \end{aligned} \tag{4.4}$$

where $\boldsymbol{w}$ is an M-dimensional vector, $Q \in \mathbb{R}^{M \times M}$ is a symmetric positive semidefinite matrix, and F is a vector in $\mathbb{R}^M$ with non-negative entries. Q represents the similarity among variables (redundancy), and F measures how correlated each feature is with the target class (relevance). The components of the solution vector $\boldsymbol{w}^*$ represents the weight of each feature which are normalized to be in the interval $[0, 1]$ and to sum up to one. Thus, the aim of Equation 4.4 is to select those features which provide a good tradeoff between relevance and redundancy for the classification task. For the time being, the α parameter weighting redundancy-relevance in QPFS (Equation 3.4) is set to $\alpha = 0.5$. The effect of this parameter in the following reasoning will be analyzed at the end of this section.

Intuitively, the **kernelization of QPFS** would try to perform QPFS in a kernel space so that the new features would be as much independent as possible and highly correlated with the target class. Furthermore, the redundancy reduction gain importance in the feature space where the dimension of the mapped data can be significantly higher than the number of training samples. However, the formulation of Equation 4.4 in a kernel space suffers from the same drawbacks previously enumerated for feature selection methods: for some kernels, it is not possible to give a weight to each feature in the kernel space due to its potential infinite dimension and what is more, the mapping function is not generally known. Maintaining still the goal of redundancy minimization of the features and relevance maximization of each feature with the target class, Equation 4.4 can be adapted to find an optimal direction $\boldsymbol{w}$ to project the data into the kernel space. Let Φ be the nonlinear mapping to the feature space $\mathcal{F}$ then, the **adapted QPFS objective function** is defined as,

$$\min_{\boldsymbol{w}} \frac{1}{2} \boldsymbol{w}^{\boldsymbol{T}} Q^{\Phi} \boldsymbol{w} - \left(F^{\Phi}\right)^{T} \boldsymbol{w} \tag{4.5}$$

where Q^{Φ} is the **redundancy** among features in the kernel space and F^{Φ} is the **relevance** of each feature with the target class in the kernel space. Thus, Equation 4.5 represents a **feature extraction method**, KQPFS, instead of a feature selection technique as the original QPFS.

In the original QPFS approach, correlation and mutual information were considered as similarity measures of redundancy and relevance. Mutual information was proposed to handle nonlinear relationships in the data, but now a linear dependence –like correlation– must be applied since $\boldsymbol{w}$ induces a linear projection of the data. To adapt correlation to the kernel space, the undesirable search of a basis in the kernel space is enforced. If instead of correlation the **covariance** is used as similarity measure, the KQPFS formulation does not require the presence of an explicit basis in the kernel space. More precisely, the Q^{Φ} and F^{Φ} matrices are defined as follows,

$$\begin{aligned} Q^{\Phi} &= \sum_{x \in S} \left(\Phi(\boldsymbol{x}) - \boldsymbol{\mu^{\Phi}}\right) \left(\Phi(\boldsymbol{x}) - \boldsymbol{\mu^{\Phi}}\right)^T \\ F^{\Phi} &= \sum_{x \in S} (\boldsymbol{y} - \boldsymbol{\mu^{y}}) \left(\Phi(\boldsymbol{x}) - \boldsymbol{\mu^{\Phi}}\right) \end{aligned}$$

where $\boldsymbol{y}$ is a N-dimensional vector with the labels of the training samples while $\boldsymbol{\mu^{\Phi}}$ and $\boldsymbol{\mu^{y}}$ are the mean value of the training samples and the training labels, respectively. That is,

$$\begin{aligned} \boldsymbol{\mu^{\Phi}} &= \frac{1}{N} \sum_{i=1}^{N} \Phi(\boldsymbol{x_i}) \\ \boldsymbol{\mu^{y}} &= \frac{1}{N} \sum_{i=1}^{N} y_i \ . \end{aligned}$$

But a formulation of Equation 4.5 in terms of only dot products of the input patterns is needed. Following the same reasoning that in Section 4.1.1, it can be shown that the solution vector $\boldsymbol{w}$ can be written as a linear combination of the mapped samples. The quadratic term in Equation 4.5 lies in the subspace spanned by the mapped training patterns $\{\Phi(\boldsymbol{x_i})\}$ since Q^{Φ} also lies in the subspace of $\{\Phi(\boldsymbol{x_i})\}$ by construction. Regarding the linear term, the operator F^{Φ} is formed by a linear combination of $\{\Phi(\boldsymbol{x_i})\}$, $(\boldsymbol{y} - \boldsymbol{\mu^{y}}) \in \mathbb{R}$, and thus when it is applied to $\boldsymbol{w} = \boldsymbol{v_1} + \boldsymbol{v_2}$ ($\boldsymbol{v_1} \in \text{Span}\{\Phi(\boldsymbol{x_i})\}$ and $\boldsymbol{v_2} \perp \text{Span}\{\Phi(\boldsymbol{x_i})\}$) only $F^{\Phi}\boldsymbol{v_1}$ is non zero. Therefore, writing $\boldsymbol{w}$ as $\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i \Phi(\boldsymbol{x_i})$, Equation 4.5 can be formulated as the minimization of function $G(\boldsymbol{\alpha})$ defined as

$$G(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^{\boldsymbol{T}} K\left(I_N - 1_N\right) K\boldsymbol{\alpha} - \boldsymbol{y}^{\boldsymbol{T}}\left(I_N - 1_N\right) K\boldsymbol{\alpha}, \tag{4.6}$$

where I_N is the N-dimensional identity matrix and 1_N is a N-dimensional square matrix with all entries $\frac{1}{N}$.

Defining $Q_K = K\left(I_N - 1_N\right) K$ and $F_K = K\left(I_N - 1_N\right)\boldsymbol{y}$, the optimal value of $\boldsymbol{\alpha}^{\text{KQPFS}}$ is obtained by making the gradient of $G(\boldsymbol{\alpha})$ equals to zero,

$$\boldsymbol{\alpha}^{\text{KQPFS}} = \left(Q_K\right)^{-1} F_K.$$

If the Q_K matrix is invertible, the formulation of the optimal direction is straightforward,

$$\begin{aligned}
\alpha^*_{\text{KQPFS}} &= \left(Q_K\right)^{-1} F_K \\
&= K^{-1}\left(I_N - 1_N\right)^{-1} K^{-1} K\left(I_N - 1_N\right)\boldsymbol{y} \\
&= K^{-1}\boldsymbol{y}.
\end{aligned}$$

Unfortunately, the matrix Q_K is always singular because its rank is upper-bounded by the rank of matrix $(I_N - 1_N)$ which is $N - 1$. As in the KFDA case, some regularizer is needed and following [96], a multiple of the identity matrix is added to Q_K matrix: $Q_\mu = Q_K + \mu_Q I_N$ with $\mu_Q \geq 0$. Ideally, μ_Q is the minimum value which makes Q_μ positive definite and a process to estimate its value is needed.

Replacing Q_K by Q_μ in Equation 4.6, we obtain the **regularized version of KQPFS**,

$$G_\mu(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^{\boldsymbol{T}}\left(Q_K + \mu_Q I_N\right)\boldsymbol{\alpha} - F_K^T\boldsymbol{\alpha}$$

which is equivalent to,

$$G_\mu(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^{\boldsymbol{T}} Q_K\boldsymbol{\alpha} - F_K^T\boldsymbol{\alpha} + \frac{\mu_Q}{2}\|\boldsymbol{\alpha}\|^2. \tag{4.7}$$

And the **regularized KQPFS direction** is given by,

$$\boldsymbol{\alpha}^{\text{KQPFS}} = \left(Q_K + \mu_Q I_N\right)^{-1} F_K \tag{4.8}$$

The KQPFS solution raised by Equation 4.8 projects the input data to one-dimensional space and it represents the projection direction which minimizes the covariance among features and maximizes the covariance of each feature with the target class in the kernel space. Note that such direction depends only on the kernel matrix K and the class labels $\boldsymbol{y}$.

Up to now, the parameter α of QPFS was assumed to be $\alpha = 0.5$ and indeed, the value of the parameter has no effect in the solution given in Equation 4.8. Denoting this parameter by β in order to become notation clearer, the KQPFS equation incorporating the β parameter can be written as follows,

$$\frac{1}{2}(1-\beta)\boldsymbol{w}^{\boldsymbol{T}} Q_\mu^\Phi \boldsymbol{w} - \beta (F^\Phi)^T \boldsymbol{w}. \tag{4.9}$$

Making the first derivative equal to zero and assuming $\beta \in (0, 1)$, the optimal direction $\boldsymbol{w}_{\boldsymbol{\beta}}^{\text{KQPFS}}$ can be expressed as $\boldsymbol{w}_{\boldsymbol{\beta}}^{\text{KQPFS}} = \frac{\beta}{1-\beta} Q_\mu^{-1} F$. It is straightforward to see that this direction is the same as the one obtained above except by the scaling factor $\frac{\beta}{1-\beta}$ that has not effect in the projection. The extreme cases $\beta = 0$ and $\beta = 1$ make KQPFS objective function meaningless. When $\beta = 0$, the relevance term is not considered and the KQPFS direction is $\boldsymbol{w}_{\boldsymbol{\beta}}^{\text{KQPFS}} = \mathbf{0}$ under the assumption that matrix Q^Φ is positive semidefinite. On the other hand, if $\beta = 1$, the redundancy term is removed from Equation 4.9 which turns out to be an unconstrained linear problem in which the coordinates of $\boldsymbol{w}_{\boldsymbol{\beta}}^{\text{KQPFS}}$ are equal to ∞ or $-\infty$ when the corresponding entries in F are positive or negative, respectively.

4.3 Equivalence of KFDA and KQPFS

This section will show that the apparent different approaches which encouraged KFDA and KQPFS converge to the same solution in the kernel space. Without loss of generality,

the regularization approaches defined in Sections 4.1.1 and 4.2 will be considered. It is straightforward to show that the following proof is also valid for other regularization functions.

Firstly, let put forward the reasons which lead Mika et al. [100] to formulate **KFDA as a quadratic programming problem**. While KFDA avoids working explicitly in the high dimensional feature space induced by the kernel matrix, the closed solution initially proposed by KFDA requires (i) a memory allocation of the order of N^2 doubles corresponding to matrices B_+, B_- and W and (ii) the calculation of either inverse of the matrix W or the greater eigenvalue of $W^{-1}(B_- - B_+)$ entails a computational cost of $O(N^3)$. In spite of the increasing memory storage of modern computers and the existence of several off-the-selve efficient eigensolvers or Cholesky packages, dealing with these matrices in large-scale domains is still unpleasant. Moreover, the obtained solutions are in general non sparse – which would speed up the projection of new points in the feature space– and it is not obvious how to induce sparsity in the model raised by Equation 4.3. However, writing the Kernel Fisher discriminant as a convex quadratic programming avoids the matrix inversion or diagonalization by making use of the most efficient quadratic programming solvers for large-scale domains. Regarding the sparsity, it can be enforced easily in the quadratic optimization model in a similar way as it was introduced for SVMs in Section 2.3.2.

The initial version of the casting of KFDA into a quadratic program exploits the fact that the matrix B has rank 1 ($B = (B_+ - B_-)(B_+ - B_-)^T$, being B_i N-dimensional vectors) and thus, $\boldsymbol{\alpha}^{\boldsymbol{T}} B \boldsymbol{\alpha}$ can be rewritten as $(\boldsymbol{\alpha}^{\boldsymbol{T}}(B_- - B_+))^2$. Noting that any non-zero multiple of $\boldsymbol{\alpha}$ is also solution for Equation 4.3, $(\boldsymbol{\alpha}^{\boldsymbol{T}}(B_- - B_+))^2$ can be equal to any non-zero number, let say 2. And all that is left is to minimize $\boldsymbol{\alpha}^{\boldsymbol{T}} W \boldsymbol{\alpha}$ resulting the following optimization problem,

$$\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^{\boldsymbol{T}} W \boldsymbol{\alpha} + CP(\boldsymbol{\alpha}) \tag{4.10}$$

Subject to:

$$\boldsymbol{\alpha}^{\boldsymbol{T}}(B_+ - B_-) \;=\; 2 \tag{4.11}$$

where $P(\boldsymbol{\alpha})$ is a regularization term which makes the W regularization raised in Section 4.1.1 explicit and $C \in \mathbb{R}$ is the regularization constant. The previous formulation has an appealing interpretation: the constraint ensures that the distance between the projected means of each class is 2 (corresponding to the distance between labels ± 1) while the intra-class variance is minimized in the objective function; that is, the average margin is being maximized. It can be shown[3] that solving the problem given in Equations 4.10 and 4.11 is equivalent to optimize

$$\min_{\boldsymbol{\alpha},b,\boldsymbol{\xi}} \|\boldsymbol{\xi}\|^2 + CP(\boldsymbol{\alpha}) \tag{4.12}$$

Subject to:

$$K\boldsymbol{\alpha} + \mathbf{1}b = \boldsymbol{y} + \boldsymbol{\xi} \tag{4.13}$$

$$\mathbf{1}_{\boldsymbol{i}}^{\boldsymbol{T}}\boldsymbol{\xi} = 0 \text{ for } i = \{-1, 1\} \tag{4.14}$$

being $\mathbf{1} \in \mathbb{R}^N$ a vector with all entries 1 and $\mathbf{1_i} \in \mathbb{R}^N$ are binary vectors with j-th entry equals to 1 if the j-th sample belongs to class i and 0 otherwise. The quadratic optimization problem can be understood as a least squares minimization in the kernel space, being ξ the error in the model prediction as stated in the first constraint. The second constraint represents the maximization of the distance between the average outputs for each class.

Besides the computational advantages derived from the reformulation of KFDA as a quadratic programming, the regularization has an extra motivation: if the number of patterns is higher than the dimension of the problem, the system of equalities is over-determined; conversely, if the number of samples is lower than the dimension of the feature space –a typical situation in kernel spaces–, the system is under-determined existing one or more solutions fitting the labels and the solution is prone to overfitting.

[3]The equivalence is obtained straightforwardly verifying that (i) the feasible sets of problems 4.10–4.11 and 4.12–4.14 are the same with respect to $\boldsymbol{\alpha}$ and (ii) that the objective functions coincide. For more details, consult [100].

Once KFDA has been expressed as a quadratic optimization problem, the **equivalence between KQPFS and KFDA** can be easily obtained. Replacing W by W_μ in Equation 4.10, the regularization term $P(\boldsymbol{\alpha})$ is equal to $\|\boldsymbol{\alpha}\|^2$, the regularization constant C is μ_W and the regularized quadratic problem in Equations 4.12-4.14 can be written as,

$$\min_{\boldsymbol{\alpha},b,\boldsymbol{\xi}} \|\boldsymbol{\xi}\|^2 + \mu_W \|\boldsymbol{\alpha}\|^2. \tag{4.15}$$

Subject to:

$$K\boldsymbol{\alpha} + \mathbf{1}b = \boldsymbol{y} + \boldsymbol{\xi} \tag{4.16}$$

$$\mathbf{1}_i^{\boldsymbol{T}} \boldsymbol{\xi} = 0 \text{ for } i = \{-1, 1\}. \tag{4.17}$$

The following proposition establishes the equivalence between KFDA and KQPFS by means of the quadratic programming above and the assumption that both methods use the same regularization criterion.

Proposition 4.1. *Given $\mu_W \in \mathbb{R}$ and let $\mu_W = \mu_Q$, any optimal solution $(\boldsymbol{\alpha}^*, b^*, \boldsymbol{\xi}^*)$ to the optimization problem (4.15-4.17) is also optimal for the regularized KQPFS (4.7) and vice versa.*

Proof. Working out $\boldsymbol{\xi}$ in the constraint given in Equation 4.16 leads to

$$\boldsymbol{\xi}\,(\boldsymbol{\alpha}, b) = K\boldsymbol{\alpha} + \mathbf{1}b - \boldsymbol{y}\ .$$

By expanding $\|\boldsymbol{\xi}\,(\boldsymbol{\alpha}, b)\,\|^2$ the optimization problem of Equation 4.15 is reformulated as

$$\min_{\boldsymbol{\alpha},b} \{\boldsymbol{\alpha}^{\boldsymbol{T}} K K \boldsymbol{\alpha} - Nb^2 - 2\boldsymbol{y}^{\boldsymbol{T}} K \boldsymbol{\alpha} + \boldsymbol{y}^{\boldsymbol{T}} \boldsymbol{y} + \mu_W \|\boldsymbol{\alpha}\|^2\}$$

subject to:

$$\mathbf{1}_i^{\boldsymbol{T}} \boldsymbol{\xi}(\boldsymbol{\alpha}, b) = 0 \text{ for } i = \{-1, 1\}.$$

The value of b can be expressed as a function of $\boldsymbol{\alpha}$ using the remaining constraint:

$$b(\boldsymbol{\alpha}) = -\frac{1}{N} 1_N K \boldsymbol{\alpha} + 1_N \boldsymbol{y} \; . \tag{4.18}$$

The resulting optimization problem has no constraints:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \{\boldsymbol{\alpha}^{\boldsymbol{T}} K K \boldsymbol{\alpha} - N \left(b(\boldsymbol{\alpha})\right)^2 \\ & -2\boldsymbol{y}^{\boldsymbol{T}} K \boldsymbol{\alpha} + \boldsymbol{y}^{\boldsymbol{T}} \boldsymbol{y} + \mu_W \|\boldsymbol{\alpha}\|^2\}. \end{aligned} \tag{4.19}$$

Then, substituting $b(\boldsymbol{\alpha})$ in Equation 4.19 by the value obtained in Equation 4.18,

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \{\boldsymbol{\alpha}^{\boldsymbol{T}} K \left(I_N - 1_N\right) K \boldsymbol{\alpha} \\ & -2\boldsymbol{y}^{\boldsymbol{T}} \left(I_N - 1_N\right) K \boldsymbol{\alpha} + \frac{\mu_W}{2} \|\boldsymbol{\alpha}\|^2 + D\} \end{aligned} \tag{4.20}$$

with D being a constant. It follows that the minimum value of Equation 4.20 is the same as the obtained for the objective function of the regularized KQPFS (Equation 4.7) when $\mu_W = \mu_Q$.

□

This equivalence provides a new solution of the Fisher direction which does not depend explicitly on the unintuitive kernelized within scatter matrix W (Equation 4.8). Moreover, the Fisher solution has a simple interpretation as the direction which minimizes the covariance among features and maximizes the covariance of each feature with the target class. Going one step further, the equivalence between KFDA and KQPFS swells the list of equivalences among KFDA and other machine learning methods. Particularly beneficial for large-scale problems is the Keerthi et al. work [101], which shows that the KFDA solution using as regularizer $\|\boldsymbol{w}\|^2$ can be derived by rescaling the solution of a Least Squares Support Vector Machines [99]. It makes possible the use of the efficient Sequential Minimal Optimization algorithm [55] to solve it and, in fact, recent advances in the SMO algorithm [49] are also applicable to KFDA. The most impressive advantages of SMO are that it does not require to storage the entire kernel matrix K at any time and it can be computationally faster for large-scale datasets than the KFDA closed formulations. Unfortunately, the design of a SMO-style algorithm is not straightforward

when regularization in $\boldsymbol{\alpha}$ is desired and costlier greedy approaches need to be applied in this case [42].

Finally, the use of the linear kernel which turns KFDA – and hence KQPFS – into the Fisher linear discriminant (regularized) could lead to believe that FDA is equivalent to QPFS in this case. Nothing further from reality – the differences between KQPFS and QPFS are noticeable: (i) QPFS is not a feature extraction method and (ii) the optimization constraints on QPFS were removed in the KQPFS formulation.

4.3.1 Theoretical Complexity Comparison

A reasonable question now is to determine whether it is possible to get any computational advantage from the new KFDA formulation as the kernelization of QPFS. Even though several algorithms have been proposed to speed up KFDA [102–104], the interest is focused on analyzing an equivalent problem to the KQPFS one given in Equation 4.4 and thus, the regularized KFDA solution $\boldsymbol{\alpha}^{\text{KFDA}} = (W_\mu)^{-1}(B_- - B_+)$ from Section 4.1.1 will be used.

Algorithms 3 and 4 show the MATLAB code for the KFDA and KQPFS methods, respectively. The number of float-point operations needed by KFDA is $4N$ (lines 2-5), $N_+^2 + N_-^2 + N^2 + 2N(N_+^2 + N_-^2) + 3N^2$ (line 6), $N^2 + 3N$ (line 7) and $O(N^3)$ (line 8) which makes a total cost of $O(N^3) + 2N(N_+^2 + N_-^2) + 5N^2 + N_+^2 + N_-^2 + 7N$ operations. In the case of the KQPFS algorithm, $N^2 + N^3$ operations are needed in line 2, $2N^2 + N^3$ in line 3, N^2 in line 4 and $O(N^3)$ in line 5 that is, a total cost of $O(N^3) + 2N^3 + 4N^2$ float-point operations. As the instruction in line 8 of KFDA and line 5 of KQPFS work with dimensionality equivalent matrices, it is supposed that the cost of these lines is the same in both cases and therefore, KQPFS is computationally faster than the proposed version of KFDA if $(N_+^2 + N_-^2)(2N + 1) + 5N^2 + 7N \gg 2N^3 + 4N^2$. The inequality is satisfied when the prior distributions of the class labels are highly unbalanced i.e., when $N_+ \to N$ or $N_- \to N$. Summing up, the KFDA cost depends on the prior distribution of classes while KQPFS does not and KQPFS is more efficient for highly unbalanced classification problems.

Algorithm 3 Kernel Fisher Discriminant Analysis

1: **Inputs**: N, K, y, μ_W
2: pos = (y==1);
3: neg = (y==-1);
4: Npos = sum(pos);
5: Nneg= sum(neg);
6: W = K(:,pos)*(eye(Npos)-(1/Npos)*ones(Npos))*(K(:,pos))'+
K(:,neg)*(eye(Nneg)-(1/Nneg)*ones(Nneg))*(K(:,neg))'+
diag(μ_W*ones(N,1));
7: B = ((1/Npos)*(sum(K(:,pos),2))) - ((1/Nneg)*(sum(K(:,neg),2)));
8: α_{KFDA} = W \ B;
9: **Output**: α_{KFDA}

Algorithm 4 Kernel Quadratic Programming Feature Selection

1: **Inputs**: N, K, y, μ_Q
2: A=K*(eye(N)-((1/N)*ones(N));
3: Q=A*K+diag(μ_Q*ones(N,1));
4: B = A*y;
5: $\alpha_{\text{KQPFS}} = Q \setminus B$
6: **Outputs**: α_{KQPFS}

4.4 Experimental Results

Experimental results described in this section are divided into two subsections according to the twofold aim of the experiments: in the first place, to verify that the projection direction obtained by KFDA and KQPFS in thirteen datasets is the same for both methods; and secondly, to compare their computational cost.

4.4.1 Empirical Equivalence between KFDA and KQPFS

Proposition 4.1 provides a theoretical proof of the equivalence between KFDA and KQPFS and it is time to corroborate that the numerical solutions given by KFDA and KQPFS coincide with the theoretical results.

Part of the experimental setup described by the KFDA's authors [96] was followed: Gaussian kernel (Table 2.2) and the regularized matrices W_μ and Q_μ described in Sections 4.1.1 and 4.2, respectively were used. Thirteen artificial and real world datasets were considered from the Rätsch benchmark repository[4]. Some of these datasets were

[4]The datasets are available at `http://ftp.tuebingen.mpg.de/pub/fml/raetsch-lab/benchmarks/`

not binary so they were transformed into two-classes problems and all of them were partitioned into 100 pairs of training and test sets (about 60%:40%). The KFDA and KQPFS algorithms were implemented in MATLAB according to the codes given in Algorithms 3 and 4. The experiments require to estimate two parameters: the width of the Gaussian kernel and the regularization parameter μ_W of the within class scatter matrix W in KFDA. The procedure to estimate these parameters consists in running 5-fold cross validation on the first five realizations of the training sets and taking the model parameter to be the median over the five estimates. The value of these parameters is also provided by the KFDA's authors [96]. Since the equivalence of KQPFS and KFDA holds when the same regularization form and regularization constant is applied in both cases, there is no need to adjust the KQPFS regularization parameter μ_Q.

The empirical equivalence of KFDA and KQPFS can be demonstrated showing that the directions $\boldsymbol{w}^{\text{KFDA}}$ and $\boldsymbol{w}^{\text{KQPFS}}$ obtained by these algorithms are parallel. Enforcing parallel directions implies that the vectors' coordinates are equal except by a real factor $\rho \neq 0$. More precisely, $\boldsymbol{w}^{\text{KFDA}} = \rho \boldsymbol{w}^{\text{KQPFS}}$. A sufficient condition to guarantee parallelism is to impose the parallelism in $\boldsymbol{\alpha}^{\text{KFDA}}$ and $\boldsymbol{\alpha}^{\text{KQPFS}}$, which are the outputs of the Algorithms 3 and 4. Therefore, the empirical equivalence of KFDA and KQPFS has been confirmed measuring the cosine between the solutions $\boldsymbol{\alpha}^{\text{KFDA}}$ and $\boldsymbol{\alpha}^{\text{KQPFS}}$, which ideally should be close to 1 or to -1. In all the datasets, the cosine of both directions was 1 for every training set.

4.4.2 Time Complexity Results

Once the numerical equivalence between KFDA and KQPFS has been demonstrated, the runtime of the KFDA and KQPFS algorithms is presented in order to show some advantages derived from the new formulation in certain scenarios. As discussed in Section 4.3.1, the KFDA complexity depends on the prior distribution of the classes in the binary classification problem and thus, the experimental setup consists in modifying the prior probability of one of the classes and then comparing the runtime of KFDA and KQPFS codes (Algorithms 3 and 4).

The regression dataset *cadata* available in the LIBSVM repository [53] was used. The dataset, formed by $20,640$ samples (N) in a 8-dimensional space, was modified in order

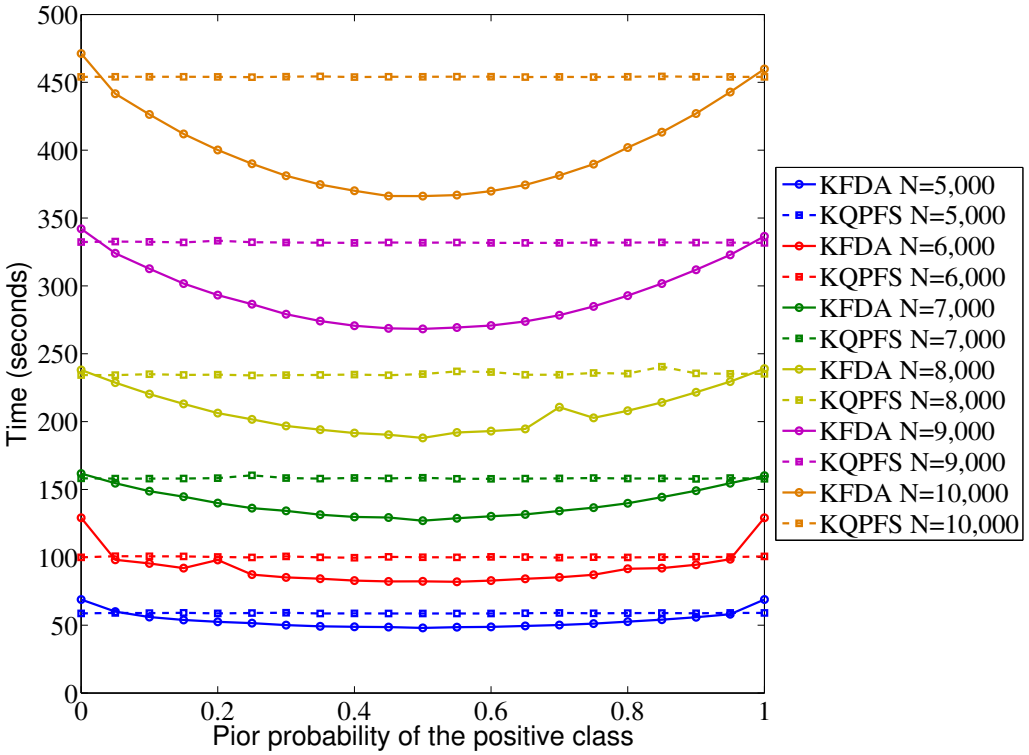


FIGURE 4.4: Cadata. Training time in seconds for the KFDA and KQPFS algorithms.

to analyze the performance of both algorithms as a function of the prior distribution of classes and the number of training samples.

- **Prior distribution of classes**. The samples were arranged in ascending order according to the regression variable. The prior probability of the positive class p_+ was modified from 0 to 1 with a stepwise of 0.05, assigning the positive label to those samples among the first p_+N patterns in the ranking.

- **Number of training samples (N)**. The first $5,000$, $6,000$, $7,000$, $8,000$, $9,000$ and $10,000$ samples were considered.

Figure 4.4 shows the runtime in training as a function of the prior probability of the positive class. As expected, the KFDA cost is dependent on the class prior probabilities, being faster than KQPFS except when the class distributions are highly unbalanced. Furthermore, the KQPFS complexity is independent of the prior distributions.

4.5 Discussion

Motivated by the possible lack of expressiveness of feature selection methods introduced in Chapters 2 and 3, this chapter has extended the feature selection method proposed in Chapter 3 to a kernel space giving rise to a nonlinear feature extraction algorithm: Kernel Quadratic Programming Feature Selection (KQPFS). The kernelization of the QPFS algorithm gives as a result an optimal projection direction in the kernel space, which maximizes the relevance between the features and the labels and minimizes redundancies among features as stated in Chapter 3. However, the similarity criteria used in the previous chapter namely, Pearson correlation and mutual information, cannot be easily extended to the kernel space since they require to compute a basis there. The obtaining of such basis not only affects to the performance of the feature selection method, but also increases its computational load, which makes this approach unfeasible for large-scale domains. Fortunately, if covariance is used instead of the previous similarity measures, the reformulation of QPFS in a kernel space can be done without computing any basis.

The use of the covariance as similarity measure in the KQPFS algorithm yields the main contribution of this Chapter: a theoretical proof of the equivalence between the KQPFS and KFDA projection directions. Such relation leads to a new interpretation of the KFDA vector as the direction which minimizes the covariance among features and maximizes the covariance of each feature with the target class in the kernel space. In addition, the new KFDA solution disregards the explicitly dependence on the kernelized between and within scatter matrices used traditionally in the Fisher algorithm and it allows a more efficient computation of the Kernel Fisher direction when classes are highly unbalanced.

In conclusion, the new KQPFS method introduced in this chapter establishes a new formulation of the Kernel Fisher Discriminant Analysis which gives new insights into the Kernel Fisher direction.

Chapter 5

Hierarchical Linear Support Vector Machine

According to the large-scale data mining process described in Chapter 2, Chapters 3 and 4 have been focused on improving the efficiency of large-scale systems in the preprocessing phase. In this chapter the attention is going to be centered in the development of a fast algorithm in the prediction phase, which represents the third contribution of this thesis [105]. The new algorithm is motivated by the necessity of designing efficient large-scale algorithms not only in the training but also in the prediction phases. Among classification problems, **Support Vector Machines** studied in Section 2.3.2 have been widely used due to its effectiveness but its application to large-scale domains requiring real-time predictions is compromised. An overview of the existing techniques focused on the acceleration of the SVMs prediction phase makes it possible to conclude that more effort is needed for **large-scale and real-time classification** problems. In this line, the **Hierarchical Linear Support Vector Machine algorithm** is presented together with an analysis of its training and prediction complexities. In addition, the derivation of an upper bound of the generalization error allows to quantify the complexity of the H-LSVM model. The chapter closes with a group of experiments pointing to H-LSVM as a compromise solution for real-time classification systems where the applicability of the nonlinear SVMs is extremely limited and linear models are too simple. The experimental results also confirm the validity of the generalization error bound in practice.

5.1 Accelerating SVM Classification

The undeniable appeal of SVMs contrasts with their high computational cost [8]. That is the reason that has led machine learning community to suggest a wide range of solutions to reduce the SVM complexity. Although many of these efforts have been focused on speeding up the SVM training, the emergence of applications requiring great classification speed (like a credit card fraud detection system with response times lower than 10 milliseconds) makes necessary the design of new algorithms maintaining as much as possible the effectiveness of nonlinear SVM and improving its classification complexity at the same time. Specifically, Equation 2.32 showed that the computational complexity of testing a pattern using a nonlinear SVM optimized in the dual space is:

$$\boxed{O\left(n_{SV} \times M \times n_K\right)}, \tag{5.1}$$

where n_{SV} is the number of support vectors, M is the dimension of the samples and n_K is the cost of evaluating the kernel function.

The excessive computational load of this expression becomes more pronounced as the number of training samples increases, since the number of support vectors (n_{SV}) scales linearly with the number of training patterns [106].

In accordance with Equation 5.1, several approaches –not exclusive– can help in the reduction of the classification cost of a machine learning system based on SVMs (Figure 5.1),

1. **Choose features** using dimensionality reduction techniques as the one proposed in Chapter 3.
2. **Choose the SVM model** to reduce the complexity of the kernel function n_K.
3. **Reduce the number of support vectors** n_{SV}. Several techniques have been proposed in order to reduce the complexity of the SVM model using either numerical approximations or data-reduction approaches; at the same time, they can be performed before (**preprocess**), during (**embedded**) or after (**postprocess**) the training process.

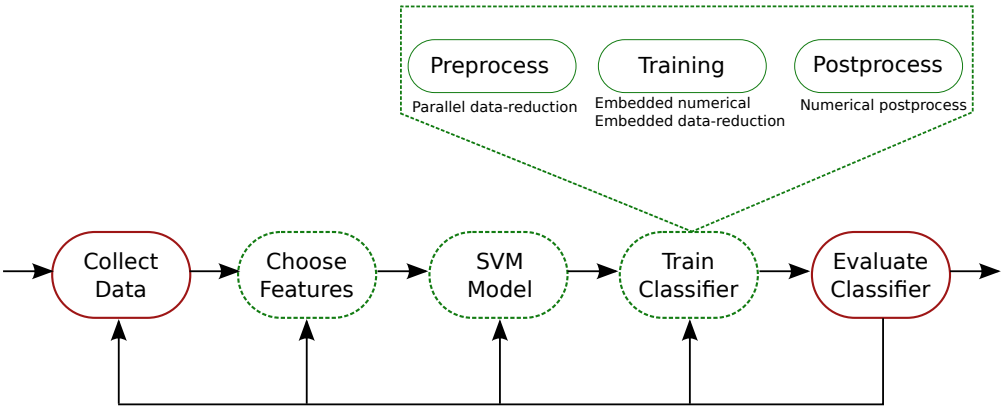


FIGURE 5.1: Machine learning system based on SVM model. The reduction of the prediction cost can be performed in different phases: choose features, choose SVM model and training the SVM model.

Linear SVMs are the best alternative for fast execution because the cost of computing the kernel function is M and their decision boundary is formed by a single hyperplane ($n_{SV} = 1$). However, their performance for nonlinear problems is uncompetitive and a compromise between performance and classification speed is needed. Therefore, methods to reduce the number of support vectors while maintaining the nonlinear kernel have been proposed in the literature. They can be divided into two groups as suggested by Keerthi et al. [107]:

- **Numerical techniques** find a reduced set of basis functions needed to classify a pattern. These algorithms usually consider all the training patterns and find a sparse representation of the support vectors. It is worth noting that the resulting model is an approximation of the exact solution of the SVM optimization problem.

  - **Embedded numerical methods**. Sparseness is imposed in the SVM model during the training process. This sparseness can be induced either in the Lagrange multipliers of the SVM dual problem or in the primal space. However, the formulation in the primal is often preferable because it is straightforward to control the quality of the obtained solution while there is no guarantee that a good approximation in the dual space yields a good approximation in the primal space [108]. In order to simplify the model, the SVM hyperplane $\boldsymbol{w}$ is written as as a linear combination of $\tilde{N}$ basis functions $\Phi(\tilde{\boldsymbol{x}}_{\boldsymbol{i}})$ as $\boldsymbol{w} = \sum_{i=1}^{\tilde{N}} \beta_i \Phi(\tilde{\boldsymbol{x}}_{\boldsymbol{i}})$ trying to have as few non-zero β_i's as possible [93, 107, 109–111]. These works suggest different methods to determine the

subset of basis functions ($\tilde{N} \leq N$), being all of them computationally intensive. The prediction cost is considerably reduced while having a competitive classification accuracy but in some datasets the number of basis functions needed to achieve a good classification rate is still high for efficient prediction phases [107].

  - **Post-processing numerical methods**. The support vector reduction can be performed as a post-processing phase after training the SVM model. These techniques reduce the number of support vectors once the SVM model has been trained [112, 113]. Therefore, they still depend on the standard SVM training which can be extremely costly in large-scale problems.

- **Data-reduction methods** reduce the number of training patterns by dividing the original training set into one or several smaller subsets and a SVM is trained in each partition. The subsets can be formed before (preprocessing) or during (embedded) the training process and the individual classifiers are combined using some of the classical classifier ensembles.

  - **Preprocessing data-reduction methods**. Parallel combinations of classifiers like bagging [114] or parallel mixture of SVMs [115] can be categorized into this group.

  - **Hierarchical data-reduction methods**. The subsets are organized hierarchically like in SVM-cascade (the classifiers are sequentially invoked until a pattern is classified) [116] or decision tree approaches (a decision tree decomposes the input space in small subregions modeled by a SVM classifier) [117, 118]. In both cases, the subset selection takes place during the training of the model.

5.1.1 Approximating nonlinear SVMs by linear SVMs

In despite of the significant improvement in terms of classification speed reached by the numerical and data-reduction approaches, their application to real-time and large-scale domains is still compromised if they depend somehow on nonlinear SVMs. Restricting the application of the approximations presented further up to linear SVMs, the numerical methods are still insufficient for the most complex task as they would generate

approximate linear models. Regarding the data-reduction techniques, in those cases in which the decision boundary is approximately linear and the data is linearly separable in a small region, the low classification cost of linear SVMs can be well-spent by building piecewise linear models. The main contributions published in the literature in this line can be categorized according to the different architectures adopted by ensemble learning,

- **Parallel combination of linear SVMs**. Each linear classifier is trained independently using the whole set or a subset of the training patterns.
  - **Linear SVMs on the manifold coordinates**. Research in this area [119–121] is based on manifold learning algorithms, also called local coding methods. The idea is to approximate any point $\boldsymbol{x}$ on the manifold as the linear combination of a set C of anchor points $\boldsymbol{v}$; thus, $\boldsymbol{x}$ can be written as follows

$$\boldsymbol{x} \approx \sum_{\boldsymbol{v} \in C} \gamma_v(\boldsymbol{x}) \boldsymbol{v},$$

    where $\gamma_{\boldsymbol{v}}(\boldsymbol{x})$ are the coefficients of the combination. Moreover, it is imposed that $\sum_{\boldsymbol{v} \in C} \gamma_{\boldsymbol{v}}(\boldsymbol{x}) = 1$ to guarantee invariance to Euclidean transformations of the data. The coefficients $\gamma_{\boldsymbol{v}}(\boldsymbol{x})$ –also called local coordinates– can be either obtained as a function of the distance of $\boldsymbol{x}$ to each anchor point [119], or as the minimization of the projection error using some regularization term inducing properties such as sparsity or locality [120]. The set of anchor points can be obtained by using standard vector quantization methods [119] or by minimizing the sum of the projection errors over the training set [120]. In any case, the local coding defined above can be used to approximate any Lipschitz function $f(\boldsymbol{x})$ as the linear combination of the values of the function in the anchor points: $f(\boldsymbol{x}) \approx \sum_{\boldsymbol{v} \in C} \gamma_{\boldsymbol{v}}(\boldsymbol{x}) f(\boldsymbol{v})$; and the quality of this approximation can be quantified using the bounds provided by Yu et al. [120]. An example of the application of these ideas is the method proposed by Ladický and Torr [121] in which the anchor points are chosen via the k-means algorithm [1] and the local coordinates are obtained as the inverse of the Euclidean distance to the k nearest neighbors.

  - **Mixture of Linear SVMs (MLSVM)**. The method proposed by Fu and Robles-Kelly [122] is based on a mixture of linear SVMs defining an underlying probabilistic model which implicitly selects the linear SVMs to be used to classify each pattern. A test sample is classified by the weighted average over the mixture of classifiers.

- **Hierarchical combination of linear SVMs**. As before, the classifiers can be grouped forming a cascade or a decision tree.

  - **Cascade of Linear SVMs**. This approach is the one followed by Zapién et al. [123] (Figure 5.2(a)) assuming that each split in the tree is able to classify correctly all the patterns belonging to the left child. Other algorithms proposed by Fehr et al. [124] and Sun et al. [125] represent an extension to the Zapién's model, being the split of each node a linear SVM and having nonlinear SVMs in the leaves of the tree (Figure 5.2(b)). These models still depend on nonlinear SVMs which means a large number of support vector evaluations to classify a test sample.

  - **Decision Tree of Linear SVMs**. This architecture is very flexible and it uses a specialized classifier in each subregion of the input space. However, the design of the tree should be carefully guided as the tree can easily incur in overfitting as pointed out in Section 2.3.3. To the best of the knowledge of the author of this thesis, any algorithm based on the construction of a decision tree with linear SVMs as node splits has been proposed in the literature. Although Bennet et al. [126] pointed out that enlarging the margin in decision trees improves their generalization ability, their methods are based on the refinement of the decision tree obtained by the OC1 algorithm [60] instead of using directly linear SVMs. The main limitations of Bennet's methods falls on (i) the non-convexity of the OC1 objective function which produces non-deterministic solutions since the algorithm can stuck in local-minimal, (ii) their dependence of several parameters that have to be tuned with extreme precision to achieve satisfactory results and (iii) their high training cost for large-scale domains.

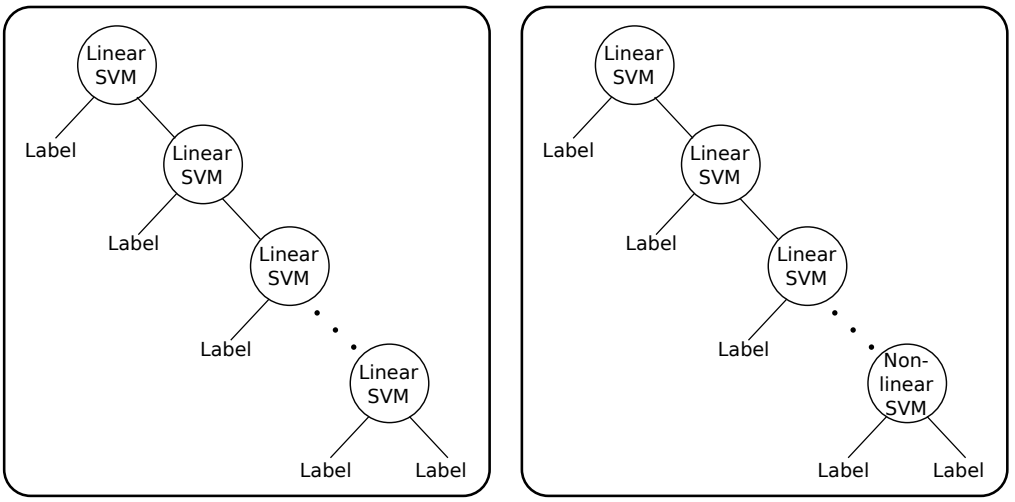


FIGURE 5.2: Architecture of different algorithms combining hierarchically SVMs. Figure 5.2(a) Zapién et al. model [123]. Figure 5.2(b) Fehr et al. [124] and Sun et al. [125] models.

5.2 The Hierarchical Linear Support Vector Machine Algorithm (H-LSVM)

In the preceding section, several approaches to speed up the prediction phase of nonlinear SVMs have been presented being the **combination of linear SVMs and decision trees** one of the most attractive solutions due to the demonstrated prediction speed of decision trees [57] and the effectiveness and generalization capabilities of linear SVMs as linear models [8]. However, the particular structure of the algorithms based on the combination of SVMs and decision trees illustrated in Figures 5.2(a)–5.2(b) suggests two possible improvements in the design of the decision tree: (i) remove the dependence on the nonlinear SVMs to avoid their high prediction complexity for large-scale settings and (ii) generalize the cascade structure adopted by the Zapién and Fehr methods. Taking into account these two points, the proposed algorithm named **Hierarchical Linear Support Vector Machine** is based on the construction of a ***complete* binary decision tree** (Figure 5.3) in which both children of each node can be expanded and the dependence on the costly nonlinear models is removed.

Following the notation used throughout this thesis, suppose a classifier learning problem involving N training samples, M variables and the training set $S = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^{N}$, where

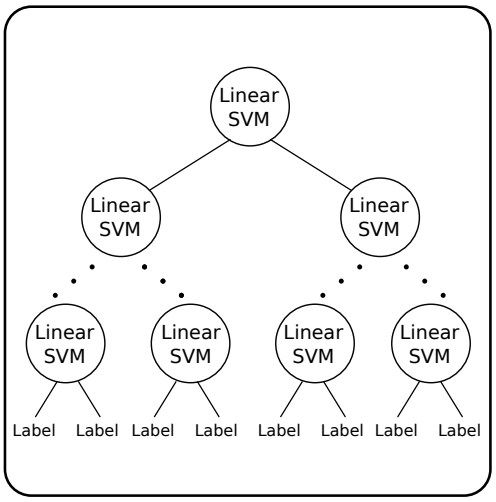


FIGURE 5.3: H-LSVM decision tree architecture.

$\boldsymbol{x_i} \in \mathbb{R}^M$ and $y_i \in \{+1, -1\}$. Decision tree terminology will be the same as the one used in Section 2.3.3 and summarized in Table 2.3.

From the point of view of decision tree algorithms (Section 2.3.3), the novelty of the H-LSVM algorithm stems from the splitting criterion consisting on linear SVMs trained with a modified version of the Pegasos algorithm (Section 2.3.2.2) with weighted patterns. For the rest of elements, well-known techniques and criteria have been used. Moreover, once the complete tree is trained, a pruning step improves the generalization capability of the H-LSVM model. The four key elements for the construction of the H-LSVM decision tree with the pruning algorithm are described below.

Splitting Goodness. The **entropy** was chosen as impurity function because it is one of the most common impurity functions in recent methods [127, 128]. Recalling the concept introduced in Section 2.3.3, the entropy of a node H_k in a binary decision tree is formulated as follows,

$$I(H_k) = -P(+|H_k)\ \log_2\left(P(+|H_k)\right) - P(-|H_k)\ \log_2\left(P(-|H_k)\right). \tag{5.2}$$

Splitting Criterion. The H-LSVM algorithm uses a **linear SVM** as splitting criterion because a single hyperplane vector $\boldsymbol{w}$ is obtained as a result of the training process which makes prediction much more efficient. Furthermore, the **Pegasos algorithm** has been demonstrated to be an efficient method for training linear SVMs in large-scale datasets [56, 129]. However, in the original formulation of the primal SVM objective function given in Section 2.3.2.2, the misclassification cost for each pattern is the same and independent of the class. This scheme can give classifiers that assign the majority class to all patterns [130], which can be optimum for the overall performance but undesirable for H-LSVM's interests in separating classes with successive splits. Going one step further, the proposed model generates a piecewise linear decision functions that divide the input space into disjoint regions in which the proportion of patterns of each class can be unbalanced and can not necessarily be the same as in the original problem. In addition, some classification problems, like fraud detection [131] or medical diagnosis [132], are unbalanced by nature. To overcome the imbalance, the H-LSVM method computes the weight $\nu_i^{H_k}$ of the sample $\boldsymbol{x_i}$ in the node H_k according to,

$$\nu_i^{H_k} = \begin{cases} \frac{1}{2N_{H_k}^+} & \text{if } \boldsymbol{x_i} \in S_{H_k}^+ \\ \frac{1}{2N_{H_k}^-} & \text{if } \boldsymbol{x_i} \in S_{H_k}^- \end{cases} \tag{5.3}$$

and verifying $\sum_{i=1}^{N_{H_k}} \nu_i^{H_k} = 1$ for all H_k. Now, the objective function of the **Weighted-Pegasos algorithm** incorporates the sample weight in the loss term,

$$\begin{aligned} &\min_{\boldsymbol{w}} f(\boldsymbol{w}; A_t) = \\ &\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \frac{1}{k} \sum_{(\nu,\boldsymbol{x},y)\in A_t} \nu \max\{0, 1 - y(\boldsymbol{w}\cdot\boldsymbol{x})\} \end{aligned} \tag{5.4}$$

and the subgradient of Equation 5.4 with respect to $\boldsymbol{w}$ on the iteration t is given by,

$$\boldsymbol{\nabla_t^w} = \lambda \boldsymbol{w_t} - \frac{1}{|A_t|} \sum_{(\nu,\boldsymbol{x},y)\in A_t^+} \nu y \boldsymbol{x}. \tag{5.5}$$

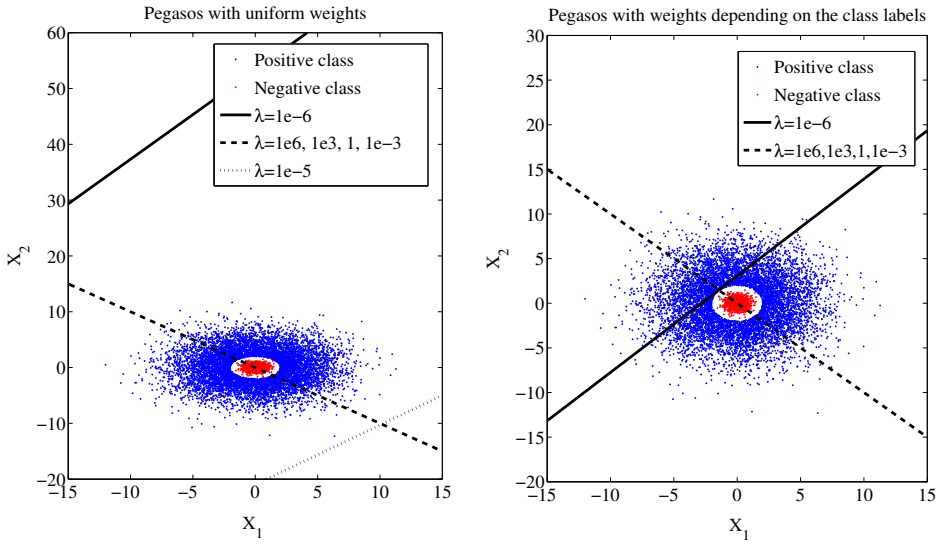


FIGURE 5.4: An example of the application of weighted patterns scheme in a nonlinear binary classification problem where patterns are unbalanced. The positive class contains $8,000$ samples while the negative class is formed by $1,000$ patterns. For different values of the λ parameter, the left figure shows the hyperplanes obtained by uniform weighting strategy while the right figure depicts the resulting hyperplanes of the weighting scheme proposed in Equation 5.3.

Figure 5.4 shows an example of the effect of weighting patterns in an unbalanced binary classification problem consisting of $9,000$ samples ($8,000$ belonging to the positive class and $1,000$ corresponding to the negative class). The hyperplanes obtained by the uniform weights for different values of λ parameter are shown on the left. It can be seen that large values of λ produce a symmetric separation of the input space while small values for λ overweight the cost term, classifying all patterns as positive. This last situation is undesirable for the H-LSVM algorithm as it does not represent any advance in the splitting process. On the other hand, the right figure shows the resulting hyperplanes for different values of λ according to the weighting scheme given in Equation 5.3. As expected, when the regularization parameter λ takes the largest values, the behavior of the model is the same as the one with uniform weights since the optimization is focused on the regularization term and the effects of the new weights are negligible. However, in the most complex cases in which more effort needs to be invested in the correct classification of the patterns, the new weights successfully solve the problem of assigning all samples to the majority class by providing a good hyperplane for successive splits.

It is worth noting that this reassignment of the penalization does not implies an improvement in the overall classification accuracy. In the example in Figure 5.4, the overall misclassification rate for w_A is $\frac{1}{11}$ whereas the rate is $\frac{5}{11}$ for the w_B hyperplane.

It can be shown that the Weighted-Pegasos algorithm verifies that the norm of the optimum of Equation 5.4 is upper bounded by $\frac{1}{\sqrt{N\lambda}}$ unlike the original version, whose solution was upper bounded by $\frac{1}{\sqrt{\lambda}}$ [56]. Nevertheless, the number of iterations required for achieving a solution of accuracy ϵ is the same in both cases: $\tilde{O}\left(\frac{R^2}{\lambda\epsilon}\right)$, where R is the radius of the ball containing all the training samples. A more detailed derivation of the convergence properties of the Weighted-Pegasos algorithm is given in Appendix C.

The pseudocode of the Weighted-Pegasos algorithm is presented in Algorithm 5. Obviously, it is based on the Pegasos algorithm introduced in Section 2.3.2.2 but some implementation issues besides the incorporation of the new bound in the projection step must be tackled. In Section 2.3.2.2, several alternatives to calculate the bias term in the Pegasos algorithm were analyzed. The simplest ones did not solve the original SVM problem and the best solution from the point of view of optimization was too expensive for large-scale problems. The H-LSVM method opts for the implementation followed by one of the standard SGD packages[1], which updates the bias term via subgradient descent and using a smaller learning rate scaled by a factor τ heuristically chosen. In addition, the number of iterations is reduced by incorporating an allowable tolerance ϵ_{PEG} for the norm of the difference between two consecutive $\boldsymbol{w}$ vectors. Thus, the training stops whenever the maximum number of iterations T is reached or the tolerance condition is satisfied.

Splitting-Stop Criterion. A node split is stopped when it does not represent an improvement in the impurity measure or when the rate of training samples associated to this node is lower than a parameter δ. If δ value is too large, the tree could be not expanded enough. It is preferable small values of δ which yield an overfitted tree because such tree will be pruned later. That is why, δ was set to $10^{-i}, i = \lfloor log_{10}N \rfloor$ in the experiments carried out further down in Section 5.3.

[1] `http://leon.bottou.org/projects/sgd`

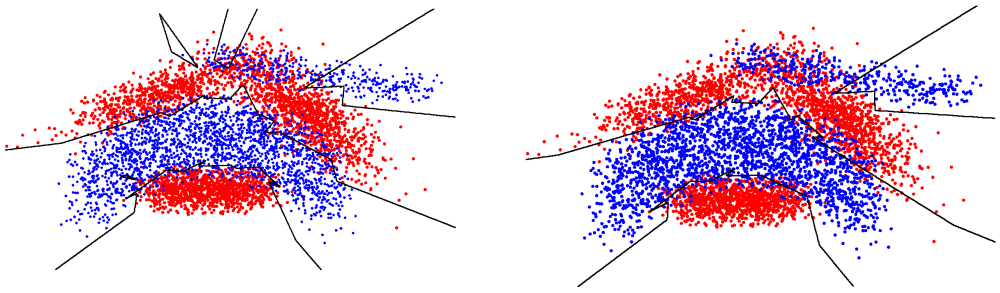
Algorithm 5 The Weighted-Pegasos Algorithm.

Inputs: $S, \lambda, T, k, \tau, \epsilon_{\text{PEG}}$
Intialization: Choose $\boldsymbol{w_1}$ s.t. $\|\boldsymbol{w_1}\| \leq 1/\sqrt{N\lambda}$
$t = 1$
while $t \leq T$ AND $\|\boldsymbol{w_t} - \boldsymbol{w_{t-1}}\| > \epsilon_{\text{PEG}}$ **do**
  Choose $A_t \subseteq S$, where $|A_t| = k$
  Set $A_t^+ = \{(\nu, \boldsymbol{x}, y) \in A_t : y(\boldsymbol{w_t} \cdot \boldsymbol{x}) < 1\}$
  Set $\eta_t = \frac{1}{\lambda t}$
  Set $\boldsymbol{w_{t+\frac{1}{2}}} = (1 - \eta_t\lambda)\boldsymbol{w_t} + \frac{\eta_t}{k} \sum_{(\nu,\boldsymbol{x},y)\in A_t^+} \nu y \boldsymbol{x}$
  Set $\boldsymbol{w_{t+1}} = \min\left\{1, \frac{1/\sqrt{N\lambda}}{\|\boldsymbol{w_{t+\frac{1}{2}}}\|}\right\} \boldsymbol{w_{t+\frac{1}{2}}}$
  Set $b_{t+1} = b_t + \tau \frac{\eta_t}{k} \sum_{(\nu,\vec{x},y)\in A_t^+} \nu y$
end while
Outputs: $\boldsymbol{w_{T+1}}, b_{T+1}$

Class Assignment Criterion. As justified in Section 2.3.3, once a pattern reaches a leaf of the decision tree, it is assigned to the majority class in such leaf because this rule minimizes the misclassification cost in the training set.

Pruning. Incorporating a pruning process in a decision tree algorithm reduces the risk of having an overfitted model [133, 134]. Although the SVM formulation already incorporates a regularization term which favors the generalization capability of the optimal hyperplane, a small value for δ in the splitting-stop criterion might imply an overfitted model. This point can be solved setting different δ values and evaluating the performance of the model in a validation step. However, this approach is computationally costlier than using a small value for δ –that is, making the tree grows as much as possible– and then applying a pruning algorithm. The latest approach is used by H-LSVM and it uses the **Cost-Complexity pruning algorithm** proposed by Breiman et al. [58] and described in Section 2.3.3.1. The pruning algorithm needs a subset of patters not seen during the construction of the tree to evaluate the goodness of the pruning. The H-LSVM algorithm keeps away a proportion ρ of the input patterns to be used by the Cost-Complexity method. The value of ρ is a parameter of the model and it must be tuned in the validation phase.

Figure 5.5 presents an example of application of the H-LSVM model. The bidimensional synthetic banana dataset [135] is used. The H-LSVM parameters were $\lambda = 10^{-5}$ and



(a) (b)

FIGURE 5.5: An example of application of H-LSVM on the banana dataset.Figure 5.5(a) shows the decision boundary when no pruning is applied. Figure 5.5(b) shows the decision boundary after a pruning process with $\rho = 0.1$.

$\delta = 10^{-3}$. Blue and red points correspond to positive and negative samples, respectively. The problem is clearly nonlinearly separable: the linear SVM yields a classification accuracy of 54.44% while the Gaussian Kernel SVM achieves a classification rate of 90.60% with 1152 support vectors. Figure 5.5(a) shows the H-LSVM decision boundary when no pruning is applied ($\rho = 0.0$). The model is clearly overfitted. Figure 5.5(b) shows the H-LSVM decision boundary when pruning is applied ($\rho = 0.1$). This model only needs to evaluate at most 12 hyperplanes to classify a new pattern achieving a classification rate of 90.60%. In this case, H-LSVM gets the same classification accuracy that the nonlinear SVM but with a classification time two orders of magnitude lower.

Once the design aspects of the H-LSVM algorithm have been established, the H-LSVM training procedure can be summarized in the following steps assuming that the model parameters $T, k, \tau, \epsilon_{\text{PEG}}, \delta$ have been fixed and the parameters λ and ρ has been estimated in the validation phase.

1. Select randomly $(1-\rho)N$ samples from the initial training set S to form the subset S_0. The remaining ρN samples, subset P, is used by the pruning algorithm.

2. Initialize the weight of each pattern in S_0 as described in Equation 5.3.

3. Train recursively the **H-LSVM Tree** following the steps given in Algorithm 6.

4. As a result of the H-LSVM tree construction, a set of K hyperplanes $\{\boldsymbol{w_k}, b_k\}_{k=1}^{K}$ is obtained.

5. **Pruning step**: If $\rho > 0$ apply the Cost-Complexity pruning algorithm over the set P to get $\left\{\left(\tilde{\boldsymbol{w_k}}, \tilde{b_k}\right)\right\}_{k=1}^{\tilde{K}}$ where $\tilde{K} \leq K$; otherwise, set $\tilde{\boldsymbol{w_k}} = \boldsymbol{w_k}$ and $\tilde{b_k} = b_k$ for all $1 \leq k \leq K$.

6. **Prediction step**: Let $\tilde{\boldsymbol{x}}$ a new sample and the H-LSVM tree defined by $\left\{\left(\tilde{\boldsymbol{w_k}}, \tilde{b_k}\right)\right\}_{k=1}^{\tilde{K}}$. The target $\tilde{y}$ of the pattern $\tilde{\boldsymbol{x}}$ is calculated as the majority class in the leaf node of the tree associated to $\tilde{\boldsymbol{x}}$.

Algorithm 6 H-LSVM Tree Construction.

Inputs: $S_0, T, k, \tau, \epsilon_{\text{PEG}}, \delta, \lambda, \rho$
I_0=I(H_0)
if $I_0 = 0$ **then**
  Finish {Homogeneous node}
end if
if $(\frac{|S_0|}{N} > \delta)$ **then**
  $\{\boldsymbol{w}, b\}$=Weighted-Pegasos($S_0, \lambda, T, k, \tau, \epsilon_{\text{PEG}}$)
else
  Finish {There are not enough number of patterns.}
end if
if $I(\boldsymbol{w}, b) \geq I_0$ **then**
  Finish {Cannot find any split}
end if
$S_{H^l} = \{\boldsymbol{x} \in S \mid \boldsymbol{w} \cdot \boldsymbol{x} + b \leq 0\}$
$S_{H^r} = \{\boldsymbol{x} \in S \mid \boldsymbol{w} \cdot \boldsymbol{x} + b > 0\}$
if $|S_{H^l}| > 0$ **then**
  Compute the weight of each pattern in S_{H^l} using Equation 5.3 where $H_k = H$
  H-LSVM_Tree($S_{H^l}, \lambda, T, \delta, \tau$)
end if
if $|S_{H^r}| > 0$ **then**
  Compute the weight of each pattern in S_{H^r} using Equation 5.3 where $H_k = H$
  H-LSVM_Tree($S_{H^r}, \lambda, T, \delta, \tau$)
end if
Outputs: $\{(\boldsymbol{w_k}, b_k)\}_{k=1}^{K}$

To finish, note that the H-LSVM learning algorithm always converges and produces a decision tree as final model. The number of nodes to generate is finite and upper bounded by the number of training samples because of the stopping criterion commonly used in learning decision tree schemes: the tree expansion is finished when there is not improvement in the impurity measure or when there are not enough number of patterns in a node. The convergence properties of the model can be obtained considering each

node separately and applying the Weighted-Pegasos convergence bounds provided in Appendix C.

5.2.1 Training and Prediction Complexity

As already mentioned, the main advantage of the H-LSVM method is speeding up the prediction phase of nonlinear SVMs. SVMs have very good results in performance in off-line problems, but when they are placed in real time operation, they are not viable. Thus, the focal attention has to be placed on prediction complexity of the linear/nonlinear SVMs and H-LSVM, though training complexity is also provided for completeness. Regarding the convexity of the SVM objective function presented in Section 2.3.2, the solution of the optimization problem is the same independently of the solver used as long as it finds the exact optimum. Therefore, the analysis of the prediction complexity carried out in what follows is valid for any SVM exact solver. However, the training complexity depends on the SVM solver used and that is why the methods used for training the SVMs are specified in what follows.

The linear SVMs were trained using the popular LIBLINEAR classification package [52]. The algorithm behind LIBLINEAR [52] is coordinate descent on the dual SVM formulation [54]. The nonlinear SVMs have been trained using the SMO algorithm [55] implemented in the LIBSVM package [53]. Finally, the H-LSVM cost is that of training as many linear SVMs as nodes in the H-LSVM tree via the Weighted-Pegasos algorithm. More precisely, if the H-LSVM decision tree has N_H internal nodes and n_i training samples reach the i-th node, the training complexity is given by the cost of training N_H linear SVMs with the Weighted-Pegasos algorithm given in Algorithm 5. Then, considering that the number of iterations needed by the Weighted-Pegasos algorithm to achieve a solution with tolerance ϵ is $\tilde{O}\left(\frac{R^2}{\lambda\epsilon}\right)$ – being R the radius of the ball containing all the training samples– and the cost per iteration is $O\left(kM\right)$, the total cost of H-LSVM is $O\left(\frac{N_H kM}{\lambda\epsilon}\right)$. For simplicity, the tolerance ϵ is fixed for every node in the tree, but as suggested by Shalev-Shwartz and Srebo [129], it could be adapted as a function of the number of training samples n_i to get some fixed generalization error in each node.

The training complexities of LIBLINEAR, LIBSVM and Pegasos algorithms are analyzed in depth by Menon [136]. Table 5.1 (column *Training*) shows the training time complexities of the three algorithms LIBLINEAR, SVM-SMO and H-LSVM. H-LSVM cost is

highly dependent on each dataset as it is determined by the structure of the tree (N_H). As expected, the lowest training cost corresponds to the linear SVM. The comparison between the training times of the nonlinear SVM and H-LSVM is not straightforward as it depends on the H-LSVM tree architecture and the λ and ϵ parameters. H-LSVM would be faster than SMO-SVM in the training phase if $\frac{N_H k}{\lambda \epsilon} \ll N^2$.

The cost to classify a new pattern $\boldsymbol{x} \in \mathbb{R}^M$ by a linear SVM is the cost of computing the dot product between the resulting hyperplane and the pattern to be classified: $O(M)$. In the case of nonlinear SVMs, the classification of the pattern $\boldsymbol{x}$ is performed according to: $\sum_{i=1}^{n_{SV}} \alpha_i \times K(\boldsymbol{x_i}, \boldsymbol{x})$, being n_{SV} the number of support vectors. If the number of operations needed to compute $K(\boldsymbol{x_i}, \boldsymbol{x})$ is expressed as $n_K \times M$, the SVM prediction complexity is $n_{SV} \times M \times n_K$. The proposed H-LSVM algorithm needs to find the leaf of the tree for the pattern $\boldsymbol{x}$ which leads to $N_H^P(\boldsymbol{x}) \times M$ operations, being $N_H^P(\boldsymbol{x})$ the number of internal nodes –oblique hyperplanes– evaluated by the algorithm until the pattern $\boldsymbol{x}$ reaches a leaf in the tree. The summary of the number of operations needed by each algorithm to classify a new pattern $\boldsymbol{x}$ is given in Table 5.1 (column *Classification*).

Obviously the lowest classification cost corresponds to the linear SVM but the linear model is usually not competitive enough for real-world datasets. Regarding the nonlinear models, it is reasonable to assume that the number of kernel operations n_K is at least 1. In that case, H-LSVM has the lowest cost if the number of node evaluations needed to classify the pattern $\boldsymbol{x}$, $N_H^P(\boldsymbol{x})$, is lower than the number of support vectors encountered by SVM, n_{SV}. The values of n_{SV} and $N_H^P(\boldsymbol{x})$ for real-world datasets are given in the following experiments and it is shown that in practice the number of operations needed by H-LSVM is indeed several orders of magnitude lower than the number of evaluations required by the nonlinear SVMs.

5.2.2 Generalization Error Bound

An interesting point of analysis in the H-LSVM algorithm is to determine the generalization capability of the model as well as to quantify somehow the complexity of the resulting tree.

The generalization error bound derived in this section is obtained from the results given by Golea et al. [137]. In this work, the bounds depend on the effective number of leaves

Algorithm	Training	**Classification**
Linear SVM	$NM \log\left(\frac{1}{\epsilon}\right)$	M
SMO-SVM	$N^2 \times M$	$n_{SV} \times M \times n_K$
H-LSVM	$\frac{MkN_H}{\lambda\epsilon}$	$N_H^P(\boldsymbol{x}) \times M$

TABLE 5.1: Number of operations needed to train a set S of N patterns in a M-dimensional space (*Training* column) and to classify a new pattern (*Classification* column) by Linear SVM, SVM-SMO and the H-LSVM algorithm. λ: regularization parameter in Pegasos formulation. ϵ: optimization tolerance. n_{SV}: number of support vectors of the nonlinear SVM model. n_K: operations are needed to compute the kernel between each support vector and the test pattern. N_H: total number of internal nodes in the H-LSVM tree. n_i: number of training samples which reach the node i in the H-LSVM tree. k: size of the random subset at each iteration of the Weighted-Pegasos algorithm. $N_H^P(\boldsymbol{x})$: number of nodes encountered by pattern $\boldsymbol{x}$ in the H-LSVM tree. Computational costs of linear SVM, SMO-SVM and Pegasos are extracted from [136].

L_{eff}, a data-dependent quantity which reflects how uniformly the training data covers the tree's leaves. L_{eff} can be considerably smaller than the total number of leaves in the tree (L) [138] and it makes this bound different from the Vapnik−Chervonenkis one, dependent on L [139, 140].

Suppose a two-class decision tree T whose internal decision nodes are labeled with boolean functions from some class $\mathcal{U}$ and whose leaves are labeled as -1 or 1. Formally, let $P = (p_1, \ldots, p_L)$ the probability vector which represents the probability that a pattern $\vec{x}$ reaches leaf i for $i = 1 \ldots L$. Then, the quadratic distance between the probability vector P and the uniform probability vector $U = (1/L, \ldots, 1/L)$ is given by $d(P, U) = \sum_{i=1}^{L} (p_i - 1/L)^2$ and the effective number of leaves in the tree is defined by $L_{\text{eff}} \equiv L(1-d(P,U))$. A bound of misclassification probability under certain distribution $\mathcal{D}$, $P_{\mathcal{D}}[T(\boldsymbol{x}) \neq y]$, can be estimated using the following theorem [137]:

Theorem 5.1. *For a fixed $\xi > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{-1, +1\}$. Consider the class of decision trees of depth up to D, with decision functions in $\mathcal{U}$. With probability at least $1 - \xi$ over the training set S (of size N), every decision tree T that is consistent with S has*

$$P_{\mathcal{D}}[T(\boldsymbol{x}) \neq y] \leq c \left(\frac{L_{\mathit{eff}}\ VCdim(\mathcal{U})\ log^2 N\ log\ D}{N} \right)^{\frac{1}{2}}$$

where $VCdim$ is the Vapnik-Chevonenkis dimension.

The H-LSVM algorithm is in line with this framework identifying the class $\mathcal{U}$ with the linear SVM. As discussed in Section 2.1.1, the Vapnik-Chervonenkis dimension of an hyperplane in a M-dimensional space is $(M+1)$ and therefore, the error bound for the H-LSVM method is reformulated as,

Lemma 5.2. *For a fixed $\xi > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{-1, +1\}$. Consider the class of decision trees of depth up to D, with H-LSVM decision functions. With probability at least $1-\xi$ over the training set S (of size N), every decision tree T that is consistent with S has*

$$P_{\mathcal{D}}\left[T(\boldsymbol{x}) \neq y\right] \leq c\left(\frac{L_{\mathit{eff}}\,(M+1)\;log^2 N\;log\;D}{N}\right)^{\frac{1}{2}}. \tag{5.6}$$

In practice it is quite difficult to have a consistent tree with the training data S. In that case, a bound of the misclassification probability can be obtained as a function of the misclassification probability in S, $P_S\left[T(\boldsymbol{x}) \neq y\right]$. Now, the probability vector is reformulated according to the training set as

$$P'_i = \frac{p_i P_S\left[T(\boldsymbol{x}) = y \mid \boldsymbol{x} \text{ reaches leaf } i\right]}{P_S\left[T(\boldsymbol{x}) = y\right]}.$$

Applying the adapted version of theorem 5.1 to not consistent trees [137], the following result is obtained for the H-LSVM tree,

Lemma 5.3. *For a fixed $\xi > 0$, there is a constant c that satisfies the following. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \{-1, +1\}$. Consider the class of decision trees of depth up to D with H-LSVM internal node decision functions. With probability at least $1-\xi$ over the training set S (of size N), every decision tree T has*

$$P_{\mathcal{D}}\left[T(\boldsymbol{x}) \neq y\right] \leq P_S\left[T(\boldsymbol{x}) \neq y\right] + c\left(\frac{L'_{\mathit{eff}}\,(M+1)\;log^2 N\;log\;D}{N}\right)^{\frac{1}{3}} \tag{5.7}$$

where c is a universal constant, and $L'_{eff} = L(1 - d(P', U))$ is the empirical effective number of leaves of T.

Therefore, the parameters of the tree which determine the error bound for the H-LSVM algorithm are the **depth D of the tree** and the **effective number of leaves L_{eff}**: the lower these parameters are, the better generalization error. The value for these parameters and the subsequent estimation of the model complexity according to Equation 5.7 are provided and analyzed in the following section.

5.3 Experimental Results

The experiments developed in this section provide an analysis of the H-LSVM model divided into three subsections corresponding to the threefold aim of the experiments:

- Compare H-LSVM with linear SVMs and nonlinear SVMs in terms of classification accuracy and prediction complexity.
- Compare H-LSVM with Zapién's algorithm described at the beginning of this Chapter (Section 5.1.1) and illustrated in Figure 5.2(a) in terms of classification accuracy and prediction complexity. To the best of the knowledge of the author of this thesis, this is the only existing approach combining exclusively linear SVMs and decision trees.
- Analyze numerically the H-LSVM error bounds derived in Section 5.2.2.

Following the aim of the H-LSVM algorithm, the experiments have been conducted in binary classification problems in which nonlinear SVMs produce a large number of support vectors. Table 5.2 presents the number of training and test patterns patterns, the number of features and the public repository for each dataset. The *Shuttle* dataset has been converted to a binary classification problem by differentiating class 1 from the rest. In the same way, the *Vehicle* dataset has been reformulated as a binary classification task consisting of differentiating class 3 from the rest. The *M3VO* and *M3VOm8* datasets correspond to differentiate digit 3 from all the other digits in the *MNIST* and *MNIST8m* problems, respectively. Finally, the *Covtype* dataset has been

	# Train	# Test	# Feat.	Repository
IJCNN	49,990	91,701	22	LIBSVM [53]
Shuttle	43,500	14,500	9	LIBSVM [53]
M3VO	60,000	10,000	780	LIBSVM [53]
M3VOm8	810,000	7,290,000	784	LIBSVM [53]
Vehicle	78,823	19,705	100	LIBSVM [53]
Faces	8,525	4,263	576	Peter Carbonetto [142]
Covtype	522,910	58,102	54	LIBSVM [53]

TABLE 5.2: Binary datasets used to compare H-LSVM with linear SVMs and nonlinear SMVs.

transformed into a binary classification problem consisting of differentiating class 2 from the rest.

In most of the datasets (*IJCNN*, *Shuttle*, *M3VO* and *Vehicle*), the training and test subsets are given beforehand. In *Faces* dataset, the experimental setup described by Zapién et al. [124] was followed using two thirds of the observations for the training and the rest as testing set. Moreover, data was normalized to minimum and maximum feature values. The experiments were run over 10 different randomly chosen training-test partitions of the dataset. In the case of the *M3VOm8* and *Covtype* datasets, it has been tried to use as many as training patterns as possible in order to simulate a large-scale system with a large number of support vectors. Then, the first $810,000$ patterns in the *M3VOm8* dataset were used for training and the remaining samples for test[2]. In the *Covtype* dataset, according to the experiments carried out in [115, 141], 9/10 of the samples for training and the remaining patterns for test and the experiments were run over 10 different randomly chosen training-test partitions of the dataset.

In all the experiments and according to the scenario described in the complexity cost analysis (Section 5.2.1), linear SVMs and nonlinear SVMs implemented in LIBLINEAR [52] and LIBSVM [53] packages were used, respectively. In the case of nonlinear SVMs, the Gaussian kernel, $k(x_i, x_j) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$, was considered. The H-LSVM has been implemented in C language and the code is publicly available at

`https://sites.google.com/site/irenerodriguezlujan/HLSVM-1.1.zip`.

[2]LIBSVM for the M3VOm8 dataset did not finish in reasonable time when training with all the available patterns.

	LIBLINEAR	**LIBSVM**		**H-LSVM**	
	C	C	γ	λ	ρ
IJCNN	10^1	10^1	10^0	10^{-5}	0
Shuttle	10^2	10^6	10^0	10^{-7}	0.2
M3VO	10^0	10^2	10^{-2}	10^{-4}	0.2
M3VOm8	10^0	10^2	10^{-2}	10^{-5}	0.2
Vehicle	10^{-1}	10^1	10^{-1}	10^{-6}	0.1
Faces	10^{-1}	10^1	10^{-2}	10^{-5}	0.1
Covtype	10^0	10^0	0.346	10^{-7}	0.0

TABLE 5.3: Parameters used in the linear SVM, nonlinear SVM and H-LSVM models for each binary dataset.

Linear SVMs, nonlinear SVMs and H-LSVM need to determine the values of a few parameters. In all datasets, except *Covtype*, the hyperparameter selection has been done using 5-fold cross validation over the training set. The cost parameter C in linear SVMs and nonlinear SVMs were selected from the grid $10^i, i = -6, \ldots, 6$. The γ parameter of the Gaussian kernel was taken from the range $10^i, i = -3, \ldots, 3$. Finally, for the H-LSVM model, the maximum number of Weighted-Pegasos iterations was fixed to $T = 10^7$, the allowable tolerance was set to $\epsilon_{\text{PEG}} = 10^{-4}$ and the minimum proportion of patterns needed to split a node δ was chosen as 10^{-i} with $i = \lfloor \log_{10} N \rfloor$ to guarantee that the H-LSVM grows to sufficient size (pruning is applied if necessary). The regularization parameter λ was chosen from the grid $\frac{10^i}{N}, i = -6, \ldots, 6$ being N the number of training samples. The grid was obtained from the equivalence $\lambda = \frac{1}{CN}$ between the LIBLINEAR and LIBSVM cost parameter C and the λ regularizer in H-LSVM. The prune rate ρ took values in $[0.0, 0.1, 0.2]$. Unfortunately, applying this hyperparameter procedure is unfeasible due to the size of the *Covtype* dataset and the number of support vectors in the resulting model; then, the nonlinear SVM hyperparameters provided in [115] were applied. The resulting parameters for each dataset and each model are given in Table 5.3.

5.3.1 Results: Comparison with linear and nonlinear SVMs

Regarding the aims described at the beginning of this Chapter, the goals of the H-LSVM algorithm were to develop an algorithm with classification effectiveness close to the nonlinear SVM but with much lower prediction complexity favoring its application in real-time classification settings. In order to quantify the efficiency of the algorithm taking as a point of reference the linear and nonlinear SVMs, the quantities *Relative Error (RE)* and *Relative Complexity (RC)* are defined as follows,

$$RE = \frac{e_{\text{LSVM}} - e}{e_{\text{LSVM}} - e_{\text{SVM}}} \tag{5.8}$$

$$RC = \frac{Hyp - 1}{n_{SV} - 1}, \tag{5.9}$$

where e represents the classification error rate. A value equals 0 in these magnitudes *RE/RC* indicate that the classification accuracy/complexity is the same as that of the linear SVM while a value of 1 represents the equivalence with the nonlinear case. Therefore, it would be desirable to have a *Relative Error* close to 1 and a *Relative Complexity* close to 0.

The results in terms of classification error (*Error (%)*), classification cost and the relative magnitudes *RE/RC* are shown in Table 5.4. In the case of the linear SVM, the number of hyperplane evaluations (*Hyp*) is shown whereas the number of support vectors (n_{SV}) is indicated for the nonlinear SVM. While the classification cost of linear/nonlinear SVMs is independent of the test sample, the H-LSVM prediction cost depends on the path of the pattern in the H-LSVM tree. Thus, the mean number of H-LSVM hyperplanes encountered per test sample and also the maximum number of H-LSVM hyperplane evaluations written in parentheses are shown. In those cases in which there were several training/test partitions, the average and standard deviation over the 10 runs of the experiment are indicated.

As expected, the classification results of the nonlinear SVMs are superior than those of the linear SVM and H-LSVM. However, the classification accuracy of H-LSVM is better than that of the linear model in all cases. These results are not surprising because the proposed H-LSVM method is simpler than the nonlinear SVM but more sophisticated

IJCNN

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	7.82	1.01	2.36
n_{SV} *or Hyp*	1	$3,154$	7.28 (16)
RE / RC	0 / 0	1 / 1	0.80 / $2.0 \cdot 10^{-3}$

Shuttle

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	2.21	0.062	0.10
n_{SV} *or Hyp*	1	66	5.18 (12)
RE / RC	0 / 0	1 / 1	0.98 / $6.43 \cdot 10^{-2}$

M3VO

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	2.09	0.33	1.79
n_{SV} *or Hyp*	1	$2,873$	3.15 (8)
RE / RC	0 / 0	1 / 1	0.17 / $7.49 \cdot 10^{-4}$

M3VOm8

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	3.90	0.03	1.43 ± 0.02
n_{SV} *or Hyp*	1	$13,471$	4.73 ± 0.003 (11.00 ± 0.00)
RE / RC	0 / 0	1 / 1	0.64 / $2.77 \cdot 10^{-4}$

Vehicle

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	14.18	11.88	12.61
n_{SV} *or Hyp*	1	$23,642$	2.84 (10)
RE / RC	0 / 0	1 / 1	0.68 / $7.78 \cdot 10^{-5}$

Faces

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	8.81 ± 0.35	2.97 ± 0.24	6.39 ± 0.43
n_{SV} *or Hyp*	1	$1,260.3 \pm 14.81$	2.59 ± 0.06 (5.30 ± 0.15)
RE / RC	0 / 0	1 / 1	0.41 / $1.26 \cdot 10^{-3}$

Covtype

	Linear SVM	*Non-linear SVM*	*H-LSVM*
Class. Err (%)	23.66 ± 0.21	18.57 ± 0.20	11.39 ± 0.08
n_{SV} *or Hyp*	1	$245,687.2 \pm 167.8$	12.93 ± 0.087 (44.00 ± 1.08)
RE / RC	0 / 0	1 / 1	2.41 / $4.86 \cdot 10^{-5}$

TABLE 5.4: Test error rate (*Class. Err (%)*) and classification complexity (n_{SV} *or Hyp*) of Linear SVMs, nonlinear SVMs and H-LSVM. The mean number of hyperplane evaluations per test sample is indicated for linear SVMs and H-LSVM. The maximum number of H-LSVM hyperplane evaluations is shown in parentheses. In the case of nonlinear SVMs, the number of support vectors (n_{SV}) is shown. The reference measures RE and RC (Equations 5.8 and 5.9) are also provided.

than linear SVMs. The classification error of H-LSVM is closer to that of the nonlinear SVM in most of the cases except in the *Faces* and *M3VO* datasets. Nevertheless, in the case of the *Faces* dataset the H-LSVM model represents an improvement of 41% respect to the linear SVM and, as it will be shown later, that outperforms significantly the results of the Zapién's algorithm. It seems that the proposed method cannot approximate nonlinear SVMs in certain domains; probably because the assumption that the decision boundary is approximately linear and the data is linearly separable in the small regions generated by the H-LSVM tree does not hold in such cases. It is worth pointing out that H-LSVM is superior to the nonlinear SVM in the *Covtype* dataset. Although, the classification error obtained for the nonlinear SVM is comparable to the results reported in [115], a thorough search of the nonlinear SVM parameters might provide better results.

Nevertheless, the main interest of the H-LSVM algorithm is not having the best classification error rates but providing a method capable of classifying a pattern in few milliseconds while having competitive performance. In this respect, the nonlinear SVM needs the largest number of operations in prediction while the lowest cost is that of the linear SVM. However, the performance of the linear SVM can be extremely poor as in the *IJCNN* or *Covtype* datasets. The classification complexity of H-LSVM is between these two models: it is higher than that of the linear SVM –in the worst case it increases the cost of the linear model in one order of magnitude– but much lower than the cost of the nonlinear SVM –H-LSVM can accelerate the prediction cost of the nonlinear SVM even by a factor of 10^4 as in the case of the *M3VOm8* and *Covtype* datasets–. In fact, the *Relative Complexity* is lower than 10^{-1} in all cases.

In order to visualize the trade-off between the misclassification error versus classification cost, Figure 5.6 shows the dependence between these two magnitudes for the linear SVM, nonlinear SVM and H-LSVM. The x-axis represents the number of support vectors or hyperplanes encountered by each method in logarithmic scale. The y-axis shows the classification error rate. Each dataset is represented by a color according to the legend: circles, squares and diamonds represent the linear SVM, nonlinear SVM and H-LSVM models, respectively. The lower left-hand area is associated to the best scenario: the lowest classification error and the lowest classification complexity. In this figure, three clusters can be easily identified according to the underlying classifier (circles, squares and diamonds). Clearly, the nonlinear SVMs have the highest classification complexity

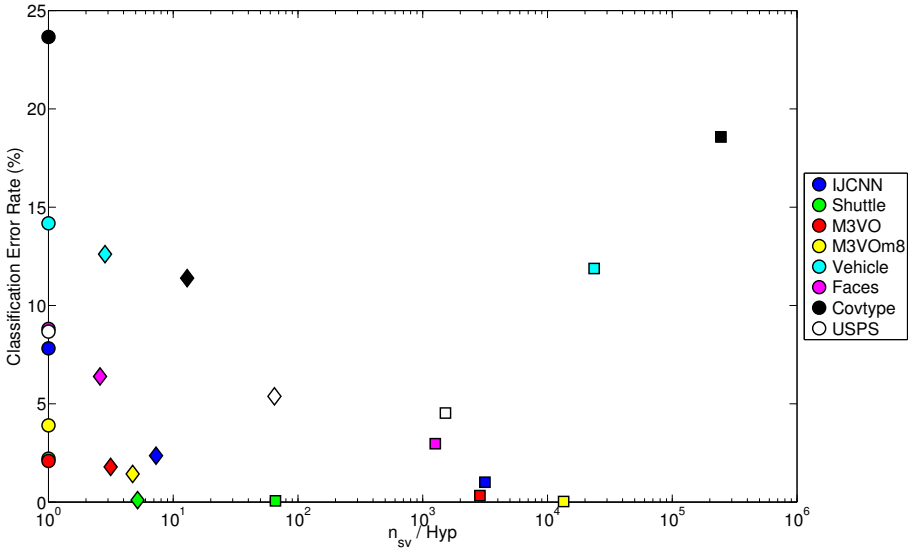


FIGURE 5.6: Classification complexity (n_{SV} / *Hyp*) versus classification error rate for different datasets. • Linear SVM ■ Nonlinear SVM ♦ H-LSVM.

while the H-LSVM cost is closer to the linear one. Looking at the classification error, in all cases the nonlinear SVM is superior –except the *Covtype* dataset– and the H-LSVM effectiveness is greater than that of the linear model.

Finally, to give an idea of the goodness of the H-LSVM algorithm with regard to the prediction time, Table 5.5 shows the time in seconds needed by a linear SVM, a nonlinear SVM and H-LSVM to classify a new pattern in an Intel(R) Core(TM) i7 CPU 920 at 2.67GHz. The training time is also included for completeness. As expected, the lowest training and testing times corresponds to the linear SVM. Regarding the training cost discussed in Section 5.2.1 , the differences between the training cost of the nonlinear SVM and H-LSVM are given by the structure of the H-LSVM tree. Therefore, depending on the dataset either the nonlinear SVM or H-LSVM is faster in the training phase. Focusing on the aim of speeding up the nonlinear SVM prediction cost, the H-LSVM classification time is always in the order of tenths of milliseconds at most and significantly lower than those of the nonlinear SVM.

Taking a glance at the methods oriented to find a reduced subset of basis functions of the nonlinear SVM hyperplane, the results obtained in this section for the *IJCNN*, *Shuttle*, *M3VO* and *Vehicle* datasets are comparable to those of the SpSVM method proposed by Keerthi et al. [107]. The SpSVM formulation gets closer to the error rate of nonlinear SVMs as the number of basis functions increases turning out to be the exact

	Linear SVM		**Nonlinear SVM**		**H-LSVM**	
	Training	*Test*	*Training*	*Test*	*Training*	*Test*
IJCNN	$4.00 \cdot 10^{-1}$	$1.22 \cdot 10^{-7}$	$2.54 \cdot 10^{1}$	$2.17 \cdot 10^{-4}$	$1.44 \cdot 10^{3}$	$2.21 \cdot 10^{-6}$
Shuttle	$4.77 \cdot 10^{-1}$	$1.09 \cdot 10^{-7}$	$4.83 \cdot 10^{0}$	$3.91 \cdot 10^{-6}$	$2.37 \cdot 10^{4}$	$2.76 \cdot 10^{-6}$
M3VO	$4.76 \cdot 10^{0}$	$6.47 \cdot 10^{-7}$	$6.40 \cdot 10^{3}$	$3.27 \cdot 10^{-3}$	$6.25 \cdot 10^{3}$	$3.61 \cdot 10^{-5}$
M3VOm8	$3.38 \cdot 10^{2}$	$3.00 \cdot 10^{-6}$ $\pm 1.28 \cdot 10^{-8}$	$7.71 \cdot 10^{4}$	$2.77 \cdot 10^{-2}$ $\pm 5.88 \cdot 10^{-6}$	$7.19 \cdot 10^{4}$	$2.88 \cdot 10^{-5}$ $\pm 4.90 \cdot 10^{-8}$
Vehicle	$2.82 \cdot 10^{0}$	$4.83 \cdot 10^{-7}$	$1.85 \cdot 10^{3}$	$9.02 \cdot 10^{-3}$	$2.99 \cdot 10^{4}$	$2.67 \cdot 10^{-6}$
Faces	$1.04 \cdot 10^{0}$ $\pm 5.28 \cdot 10^{-2}$	$2.41 \cdot 10^{-6}$ $\pm 5.38 \cdot 10^{-9}$	$2.39 \cdot 10^{1}$ $\pm 9.61 \cdot 10^{-2}$	$2.52 \cdot 10^{-3}$ $\pm 1.66 \cdot 10^{-5}$	$4.70 \cdot 10^{3}$ $\pm 5.98 \cdot 10^{1}$	$1.36 \cdot 10^{-5}$ $\pm 4.71 \cdot 10^{-7}$
Covtype	$6.65 \cdot 10^{1}$ $\pm 1.05 \cdot 10^{-1}$	$1.30 \cdot 10^{-7}$ $\pm 1.76 \cdot 10^{-9}$	$2.10 \cdot 10^{4}$ $\pm 1.31 \cdot 10^{2}$	$1.95 \cdot 10^{-2}$ $\pm 7.43 \cdot 10^{-5}$	$3.13 \cdot 10^{4}$ $\pm 8.47 \cdot 10^{1}$	$6.83 \cdot 10^{-6}$ $\pm 3.86 \cdot 10^{-8}$
USPS	$5.46 \cdot 10^{0}$	$3.58 \cdot 10^{-6}$	$5.21 \cdot 10^{0}$	$1.17 \cdot 10^{-3}$	$5.88 \cdot 10^{3}$	$1.59 \cdot 10^{-4}$

TABLE 5.5: Training and testing times in seconds required by LIBLINEAR, LIBSVM and H-LSVM.

	Classification Cost	**SpSVM error rate**	**H-LSVM error rate**
IJCNN	10^1	$\approx 8.5\%$	2.36%
Shuttle	10^1	$> 0.3\%$	0.14%
M3VO	10^0	$> 3.4\%$	1.87%
Vehicle	10^1	$\approx 14.4\%$	12.98%

TABLE 5.6: Classification error rate for the SpSVM method [107] and the H-LSVM algorithm at the level of H-LSVM prediction complexity. The SpSVM results are extracted from figures in [107].

solution of nonlinear SVM when the number of basis functions is equal to the number of support vectors. However, when the number of basis functions is in the order of the H-LSVM prediction complexity, SpSVM classification error rates are significantly higher than those of H-LSVM as shown in Table 5.6. These results corroborates the suitability of the H-LSVM method when fast classification is required.

5.3.2 Results: Comparison with SVM Trees Algorithm

Once H-LSVM have been located respect to the SVMs reference models, now its performance is compared with that of the Zapién's decision tree (Figure 5.2(a)) also denominated SVM Trees algorithm. Unfortunately, only one of the binary classification problems used in Zapién's works [123, 124] is nonlinear (*Faces*). Despite H-LSVM has been designed for binary classification problems, the performance of the model in the multiclass *USPS* dataset was measured. The *USPS* dataset for handwritten text recognition is available in the LIBSVM Repository [53]. It consists of $7,291$ training samples and $2,007$ test samples and each example is described by 256 features. Following the methodology described in the experiments reported by the authors, the data was normalized to minimum and maximum feature values and the *one against one* approach (1A1) was used for the multiclass problem. The 1A1 strategy consists on training a classifier for every pair of classes and classifying a new pattern based on majority voting. The hyperparameters were chosen using 5-fold cross validation as described for the preceding experiments. The selected parameters were $C = 1$ for the linear SVM, $C = 10^1$ and $\gamma = 10^{-2}$ for the nonlinear SVM with Gaussian kernel and $\lambda = 10^{-5}$ and $\rho = 0$ for the H-LSVM algorithm. The results in terms of the misclassification error, classification cost and the relative measurements *RE/RC* for SVM Trees and H-LSVM methods are presented in Table 5.7 in which the Zapién's method performance was extracted from [123, 124]. In both cases H-LSVM is superior in terms of classification accuracy whereas the classification cost is in the same order of magnitude. Specifically, their classification complexity is quite similar in the *Faces* dataset but SVM Trees algorithm is slightly faster for the *USPS* database.

In summary, the H-LSVM decision tree expanding both children of each node and weighting patterns in the linear SVM provides advantages with respect to the Zapién's method in terms of classification accuracy while maintaining its classification cost. It is also worth noting that in all cases the maximum depth of the tree is lower than the number of internal nodes (the number of linear SVMs), which means that the structure of the tree is far from being a cascade of classifiers as in the models shown in Figures 5.2(a) and 5.2(b).

Regarding other techniques based on the use of linear SVMs to accelerate the nonlinear

Faces

	Linear SVM	*Non-linear SVM*	*SVM Trees*	*H-LSVM*
Class. Err (%)	8.81	2.97	8.99	6.39
n_{SV} *or Hyp*	1	1260.3	4	2.59 (5.30)
RE / RC	0 / 0	1 / 1	−0.03 / 0.002	0.41 / 0.001

USPS

	Linear SVM	*Non-linear SVM*	*SVM Trees*	*H-LSVM*
Class. Err (%)	8.67	4.53	6.24	5.38
n_{SV} *or Hyp*	1	1521	49	64.77 (117)
RE / RC	0 / 0	1 / 1	0.59 / 0.03	0.79 / 0.04

TABLE 5.7: Comparison of the SVM Trees method by Zapién et al. [123, 124] and H-LSVM. The misclassification error (*Class. Err (%)*) and the mean number of hyperplane evaluations per test sample (*Hyp*) are shown for both methods and for the linear and non-linear SVMs (n_{SV}). The maximum number of H-LSVM hyperplane evaluations is indicated in parentheses. The number of hyperplane evaluations was computed as the sum of the hyperplanes evaluated in every binary classifier. The *Relative Error (RE)* and the *Relative Complexity (RC)* of the SVM Trees method and H-LSVM are also given.

SVMs prediction phase (Section 5.1.1), those based on the manifold coordinates have come out recently making a big impact. In particular, the Locally Linear SVM (LL-SVM) model proposed by Ladicky et al. [121] reports results for the *USPS* dataset with a slightly lower classification accuracy. As stated in Section 5.1.1, the prediction cost falls on the computation of the distance to the k-means centroids whose number amount to 100 in this case whereas H-LSVM evaluates 64.77 hyperplanes in average (maximum 117). Therefore, both methods are comparable in terms of classification accuracy and prediction complexity.

5.3.3 Numerical Analysis of H-LSVM Generalization Error Bound

The generalization error bound provided by Lemma 5.3 establishes a linear dependence between the complexity of the tree and the difference between the misclassification probability under certain distribution $\mathcal{D}$ and the misclassification rate in the training set S. More precisely, let $T_{\text{comp}}(L'_{\text{eff}}, M, N, D)$ the complexity of the decision tree T defined as a function of some data-dependent parameters as follows

$$T_{\text{comp}}(L'_{\text{eff}}, M, N, D) = \left(\frac{L'_{\text{eff}} \, (M+1) \, log^2 N \, log \, D}{N} \right)^{\frac{1}{3}}.$$

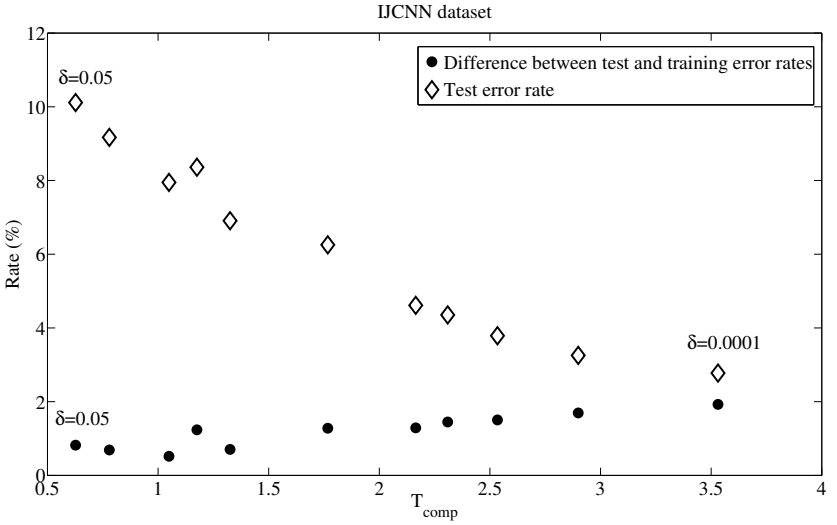
The error bound can be rewritten as,

$$P_{\mathcal{D}} \left[T(\boldsymbol{x}) \neq y \right] \leq P_S \left[T(\boldsymbol{x}) \neq y \right] + c \, T_{\text{comp}}(L'_{\text{eff}}, M, N, D).$$

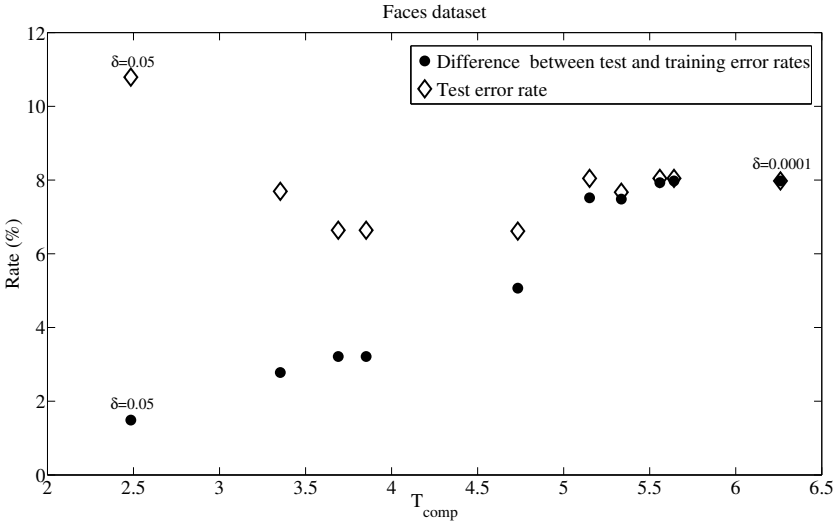
The aim of the experiments described in what follows is to determine whether this bound holds in practice. The misclassification probability under a distribution $\mathcal{D}$, $P_{\mathcal{D}} \left[T(\boldsymbol{x}) \neq y \right]$ has been approximated by the error rate in the test set and it is denoted as $\hat{P}_{\mathcal{D}} \left[T(\boldsymbol{x}) \neq y \right]$. The range of values of the complexity measure depends on the characteristics of each dataset making useless the comparison among different datasets. However, an interesting point of analysis is to determine if, in practice, there exists such linear correlation between the difference of the test and training error rates and the complexity of the model. This relation is analyzed for the *IJCNN* and *Faces* datasets used above varying the values of the δ parameter to obtain the values for T_{comp}, $P_S \left[T(\boldsymbol{x}) \neq y \right]$ and $\hat{P}_{\mathcal{D}} \left[T(\boldsymbol{x}) \neq y \right]$. The δ parameter allows to measure and control the complexity of the model: if δ takes values in the grid $\{\delta_1 > \delta_2 > \ldots > \delta_K\}$, the resulting trees T_{δ_m} verify $T_{\delta_1} \preceq T_{\delta_2} \preceq \ldots \preceq T_{\delta_K}$. In particular, the δ grid in these experiments varies from $\delta = 0.05$ to $\delta = 0.0001$. The prune rate was fixed to $\rho = 0.0$ in both datasets and the λ parameter was selected as in Table 5.3.

The linear correlation between the H-LSVM tree complexity and the difference between the test and training error rates is 0.91 for the *IJCNN* dataset and it is 0.97 for the *Faces* dataset. These high correlations reveal that the generalization error bound given in Lemma 5.3 is valid in practice.

Finally, it is interesting to see how the underfitting and overfitting phenomena introduced in Section 2.1 are reflected in Figure 5.7. In the case of the *IJCNN* dataset, the differences between the test and training error rates are small for the largest values of δ while the test error rate is the worst. It seems that the model is too coarse to fit the data. On the other hand, the lowest values for δ have slightly larger differences between



(a)



(b)

FIGURE 5.7: Difference between the test and training error rates and the test error rate as a function of the complexity T_{comp} of the H-LSVM model. Figure 5.7(a) *IJCNN* dataset; Figure 5.7(b) *Faces* dataset.

the test and training error rates but the test error rate is the lowest. This scenario is preferable to that with large values of δ because the model have the lowest classification error rate without incurring in overfitting.

In the *Faces* dataset underfitting/overfitting are clearly reflected for large/small δ values, respectively. In the first case, when δ takes large values, the difference between the training and test errors is small but the test error is the highest which means that the model is underfitted. In the second case, δ taking low values implies that the difference between the training and the test error increases whereas the test error starts

to becoming worse for the lowest values of δ; that is, the H-LSVM decision tree is generalizing poorly in the test set because it has overfitted the training data. Regarding that the δ parameter was chosen in the experiments small to guarantee that the depth of the tree is large enough to learn the data, it makes sense that the optimal pruning rate for the *Faces* dataset was $\rho = 0.1$ in order to improve the generalization capabilities of the model and avoid overfitting.

5.4 Discussion

This chapter has presented and analyzed a new classification method, named Hierarchical Linear Support Vector Machine, for medium and large-scale datasets which tackles the problem of classification of large-scale databases. The new method was motivated by the impossibility of applying nonlinear SVMs to large-scale problems because they generate a large number of support vectors. Although other methods have been proposed in the literature to speed up SVMs by finding numerical approximations of the exact model or by decomposing the data space in subregions, most of them still depend somehow on nonlinear SVMs with an unmanageable number of support vectors. The success of the new algorithm falls on the demonstrated efficiency in training and prediction of linear SVMs and the arrangement of these models in a decision tree which allows the construction of a piecewise linear model to approximate nonlinear decision boundaries. In this way, the H-LSVM algorithm is based on the construction of an oblique decision tree in which the node split is obtained as a linear SVM trained with a modified version of the Pegasos algorithm with weighted patterns. The H-LSVM algorithm represents a very simple and efficient model in training but mainly in prediction for large-scale datasets: only a few hyperplanes need to be evaluated in the prediction step, no kernel computation is required and the tree structure makes parallelization possible [143]. In experiments with medium and large datasets, the H-LSVM is able to classify a pattern in few milliseconds speeding up the prediction phase of SVMs several orders of magnitude while maintaining a classification accuracy close to that of the nonlinear SVMs. Compared with the existing methods based on the construction of a decision tree with linear SVMs as splitting criterion, the H-LSVM model is superior in terms of classification accuracy while maintaining a classification complexity of the same order of magnitude. In addition, the derivation of a generalization error bound and its subsequent empirical

analysis have shown not only the validity of this estimate, but also the importance of a careful search of the H-LSVM parameters as well as the undeniable necessity of the pruning postprocess carried out.

Complementing the methods proposed in Chapters 3 and 4 oriented to improve the efficiency of large-scale machine learning systems, the new algorithm intends to be a solution to the problem of applying SVM technology into industrial settings with high loads in real-time classification in which the accuracy worsening is bearable when compared to the runtime savings.

Chapter 6

A Global View

Chapters 3 and 5 have proposed two complementary approaches to improve the efficiency of large-scale classification systems. Specifically, Chapter 3 presented a scalable feature selection method (QPFS) applicable to any classification algorithm and able to reduce considerably the computational cost of the classifier in both the training and prediction phases. Chapter 5, for its part, was focused on a new classification algorithm (H-LSVM) capable of guaranteeing classification response times of few milliseconds that the successful nonlinear SVMs are unable to achieve. Undoubtedly, both approaches can be easily combined and the aim of this chapter is to show empirically the advantages derived from this synergy in a large-scale classification problem.

6.1 Experimental Setup

The experiments were in the ***M3VOm8* dataset** already used in Chapter 5 and consisting of **$810,000$ training patterns** (N) and **$7,290,000$ test samples** of **784 dimensions** (M). Recall that this dataset is an extension of the MNIST database used in Chapter 3 formed by 28×28 pixel images of handwriting digits 0-9 and it is available at the LIBSVM repository [53]. The dataset was converted into a binary classification problem to distinguish digit 3 from all the rest. This dataset was chosen because (i) it represents a large-scale domain in which nonlinear SVMs are hardly applicable when real-time predictions are needed, (ii) the linear SVMs classification rate is far from that of the nonlinear SVMs and (iii) the QPFS method has demonstrated its good performance

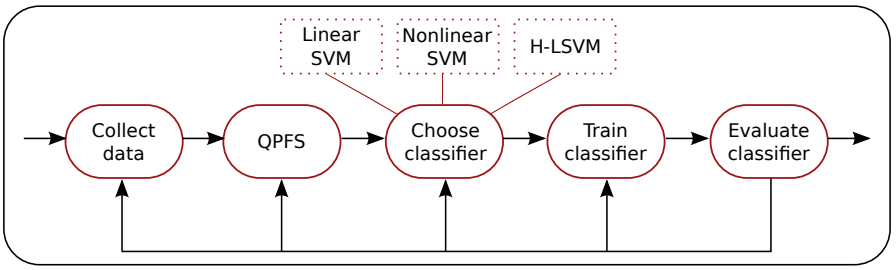


FIGURE 6.1: Classification system incorporating the QPFS and H-LSVM algorithms. Linear and nonlinear SVMs are also taking into account as baseline methods.

in the multiclass MNIST dataset. A diagram of the classification system to evaluate is presented in Figure 6.1: the QPFS will be used as feature selection method and the three different classification models introduced in Chapter 2 will be considered.

As in the preceding chapters, **publicly available software** was used to implement the algorithms in Figure 6.1 namely, **LIBLINEAR package** [52] for linear SVMs, **LIBSVM software** [53] for nonlinear SVMs and the **implementations of QPFS and H-LSVM** provided together with this thesis. All these methods entails to adjust some parameters. In the case of QPFS, the parameter α weighting the relevance and redundancy terms (Equation 3.4) was set according to the heuristic proposed in Chapter 3 and the Nyström approximation was used to alleviate considerably the computational load of the feature selection algorithm. In particular, the Nyström subsampling rate was set $p = 0.5$ which means to halve the computational cost of the original QPFS. In the case of the classification learners, the parameters was set following the methodology described in Chapter 5: the hyperparameter selection has been done using 5-fold cross validation over the training set. The cost parameters in linear SVMs and nonlinear SVMs were selected from the grid $10^i, i = -6, \ldots, 6$. The γ parameter of the Gaussian kernel was taken from the range $10^i, i = -3, \ldots, 3$. Finally, for the H-LSVM model, the maximum number of Pegasos iterations was fixed to $T = 10^7$ with a tolerance of $\epsilon_P = 10^{-4}$ and the minimum proportion of patterns needed to split a node δ was chosen as 10^{-5}. The regularization parameter λ was chosen from the grid $\frac{10^i}{N}, i = -6, \ldots, 6$ being N the number of training samples. The grid was obtained from the equivalence $\lambda = \frac{1}{CN}$ between the LIBLINEAR and LIBSVM cost parameter C and the λ regularizer in H-LSVM. The prune rate ρ took values in $[0.0, 0.1, 0.2]$. The resulting values for these parameters are shown in Table 6.1.

Finally, all the experiments were run in a Intel(R) Core(TM) i7 CPU 920 at 2.67GHz.

Algorithm	**Parameters**
QPFS	$\hat{\alpha} = 0.49 \quad p = 0.5$
LIBLINEAR	$C = 10^0$
LIBSVM	$C = 10^2 \quad \gamma = 10^{-2}$
H-LSVM	$\lambda = 10^{-5} \quad \rho = 0.2$

TABLE 6.1: Parameters used in the QPFS, linear SVM, nonlinear SVM and H-LSVM models for the *M3VOm8* dataset.

6.2 Experimental Results

According to the aim of this thesis, the results obtained from the experiments will be analyzed in terms of the classification error rate and classification cost. The linear and nonlinear SVMs without feature selection are considered as reference models and they are denoted as *BS-LSVM* and *BS-NLSVM*, respectively. The proximity of the SVMs and H-LSVM using QPFS to the baseline models is quantified with the following measures:

- **Relative Error (RE)**. e represents the classification error rate of the model to evaluate.

$$RE = \frac{e_{\text{BS-LSVM}} - e}{e_{\text{BS-LSVM}} - e_{\text{SVM}}}.$$

- **Relative Classification Complexity (RCC)**. *eval* represents the number of operations required to classify a test pattern. According to Table 5.1 in Chapter 5, the linear SVM complexity scales linearly with the dimension of the patterns needing $O(M)$ operations, the nonlinear SVM evaluates the sample with a cost of $O(n_{\text{SV}} \times M \times n_K)$ and H-LSVM requires $O(N_H^P(\boldsymbol{x}) \times M)$ calculations. In the case of nonlinear SVMs, the number of kernel evaluations is supposed to be $n_K = 1$, which is an optimistic approximation. Unlike the SVMs, the H-LSVM classification complexity is not deterministic as it depends on the path of each test samples in the H-LSVM decision tree. Thus, the RCC measure will use the mean number of hyperplanes encountered by each test sample but, the maximum number of internal nodes evaluated during the H-LSVM test phase will be also provided in what follows to show the worst scenario in the H-LSVM classification procedure.

$$RCC = \frac{eval - eval_{\text{BS-LSVM}}}{eval_{\text{BS-NLSVM}} - eval_{\text{BS-LSVM}}}.$$

- **Relative Classification Time (RCT)**. t^{class} represents the time in seconds spent in the classification phase. Analogously to the RCC measure, the classification time of H-LSVM depends on the path followed by each test pattern in the H-LSVM decision tree and t^{class} will be approximated by the mean value over all test samples.

$$RCT = \frac{t^{\text{class}} - t^{\text{class}}_{\text{BS-LSVM}}}{t^{\text{class}}_{\text{BS-NLSVM}} - t^{\text{class}}_{\text{BS-LSVM}}}.$$

- **Relative Training Time (RTT)**. Although the scope of this thesis has been focused on the prediction time of the classifiers, the relative training time of the different architectures presented in Figure 6.1 are also given for completeness. In this case, t^{train} represents the time (in seconds) spent in the training phase.

$$RTT = \frac{t^{\text{train}} - t^{\text{train}}_{\text{BS-LSVM}}}{t^{\text{train}}_{\text{BS-NLSVM}} - t^{\text{train}}_{\text{BS-LSVM}}}.$$

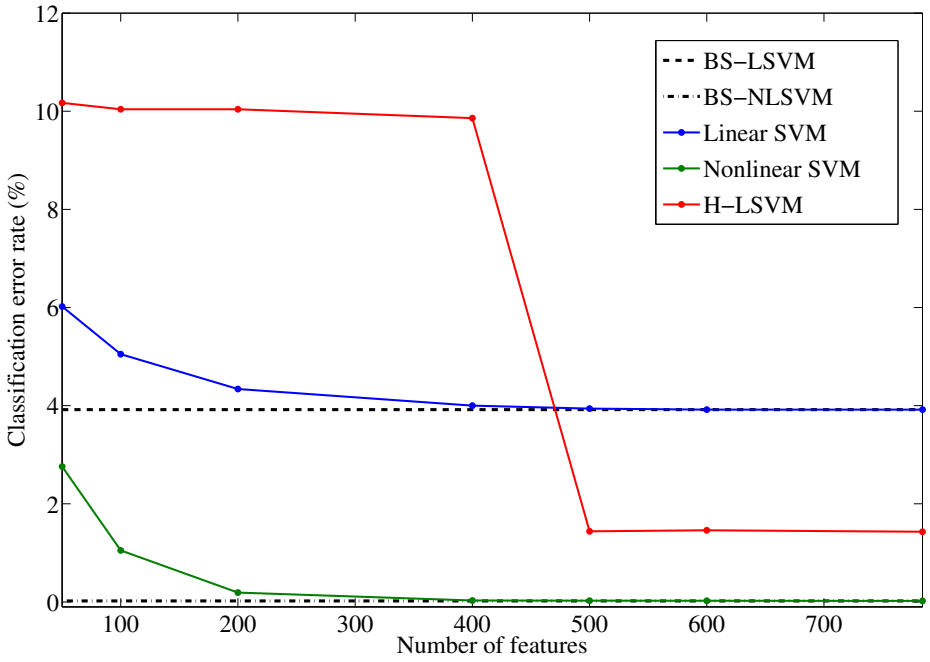
In all cases, a value equals 0 in these magnitudes indicates that the classification accuracy, classification complexity or training/classification times are the same as those of the linear SVM without feature selection. Conversely, a value of 1 represents the equivalence with the nonlinear case. Therefore, it would be desirable to have RE close to 1 and a RCC, RCT (and RTT) close to 0. Table 6.2 shows the results in terms of classification error rate and number of hyperplanes/support vectors for the three classifiers considered in Figure 6.1 which take into account different number of features according to the QPFS algorithm as well as the relative measures RE and RCC. Besides the classification complexity measure –which is independent of the characteristics of the computer in which the classification system will work–, Table 6.3 shows the time in seconds in the training and classification as well as the values of the relative measures RCT and RTT in order to have an idea of the real computational time required by all these algorithms (recall that the experiments were run in an Intel(R) Core(TM) i7 CPU 920 at 2.67GHz). At this point, it is worth mentioning that it took $2.80 \cdot 10^4$ seconds (468 minutes) for QPFS (+Nyström) to obtain the single rank of features used by all

the algorithms. Additionally, graphical representations of the results given by Tables 6.2 and 6.3 are given in Figures 6.2, 6.3, 6.4 and 6.5.

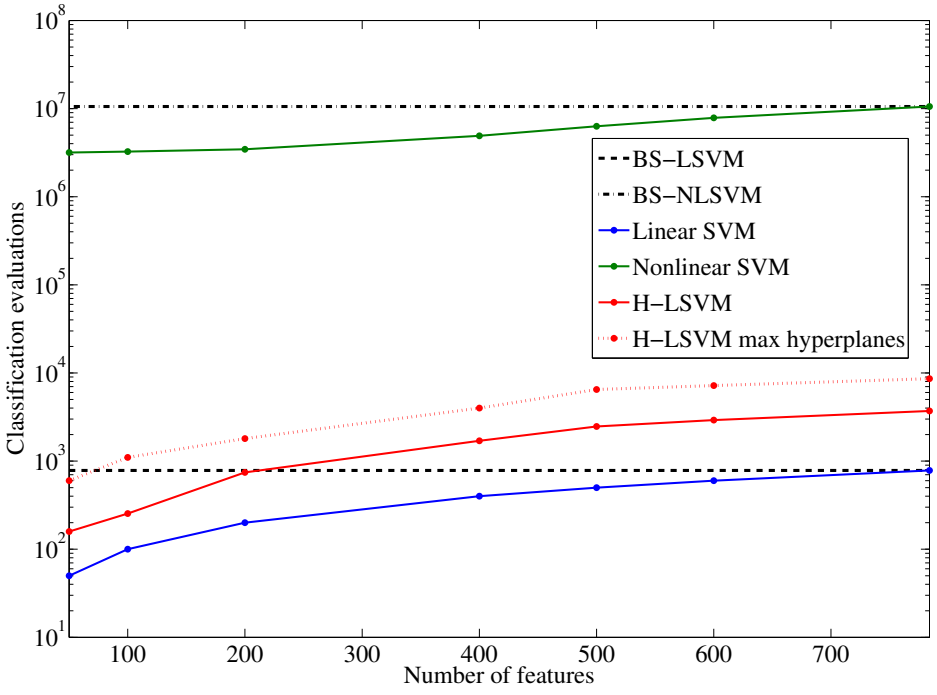
Similarly to Figure 5.6 in Chapter 5, Figure 6.6 attempts to give a visual representation of the relative position of the different scenarios proposed in Figure 6.1 that is, it shows the dependence between the misclassification error and the classification complexity for the linear SVM, nonlinear SVM and H-LSVM with and without QPFS feature selection. The number of features preserved in each case was chosen trying to maintain as much as possible the effectiveness of the model without feature selection; that is, 400 features were considered for SVMs and 500 attributes were taking into account by H-LSVM. The x-axis represents in logarithmic scale the number of operations needed by each method to classify a sample, the y-axis shows the classification error rate and the lower left-hand area is associated to the best scenario: the lowest classification error and the lowest classification complexity. The maximum classification cost of H-LSVM is also depicted.

Feat.	**LINEAR SVM**				**NONLINEAR SVM**				**H-LSVM**			
	Error(%)	*Hyp*	*RE*	*RCC*	*Error(%)*	n_{SV}	*RE*	*RCC*	*Error(%)*	*Hyp*	*RE*	*RCC*
50	6.02	1	-0.54	$-7.00 \cdot 10^{-5}$	2.76	63,490	0.30	0.30	10.17	3.17(12)	-1.60	$-5.90 \cdot 10^{-5}$
100	5.05	1	-0.29	$-0.65 \cdot 10^{-4}$	1.05	32,544	0.74	0.31	10.04	2.54(11)	-1.57	$-5.00 \cdot 10^{-5}$
200	4.34	1	-0.11	$-0.55 \cdot 10^{-4}$	0.19	17,282	0.96	0.33	10.04	3.72(9)	-1.57	$-3.8 \cdot 10^{-6}$
400	4.00	1	-0.02	$-3.6 \cdot 10^{-5}$	0.029	12,291	1.00	0.47	9.86	4.25(10)	-1.52	$8.7 \cdot 10^{-5}$
500	3.94	1	-0.005	$-2.7 \cdot 10^{-5}$	0.025	12602	1.00	0.60	1.44	4.94(13)	0.64	$1.6 \cdot 10^{-4}$
600	3.92	1	0	$-1.7 \cdot 10^{-5}$	0.024	13080	1.00	0.74	1.46	4.87(12)	0.64	$2.00 \cdot 10^{-4}$
784	3.92	1	0	0	0.020	13477	1	1	1.43	4.73(11)	0.64	$2.8 \cdot 10^{-4}$

TABLE 6.2: Classification error rate (*Error(%)*), number of hyperplanes (*hyp*) or support vectors (n_{SV}) and *Relative Error (RE)* and *Relative Classification Complexity (RCC)* for the machine learning system presented in Figure 6.1 consisting of QPFS feature selection combined with linear SVM, nonlinear SVM or H-LSVM classifiers. For H-LSVM the maximum number of hyperplane encountered by a test sample is indicated in brackets.

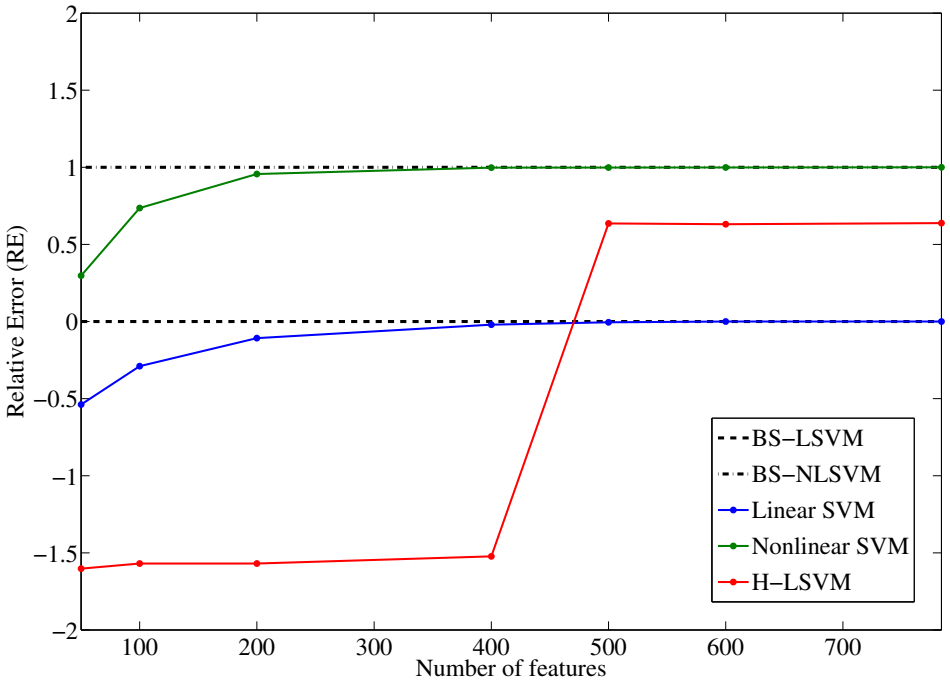


(a)

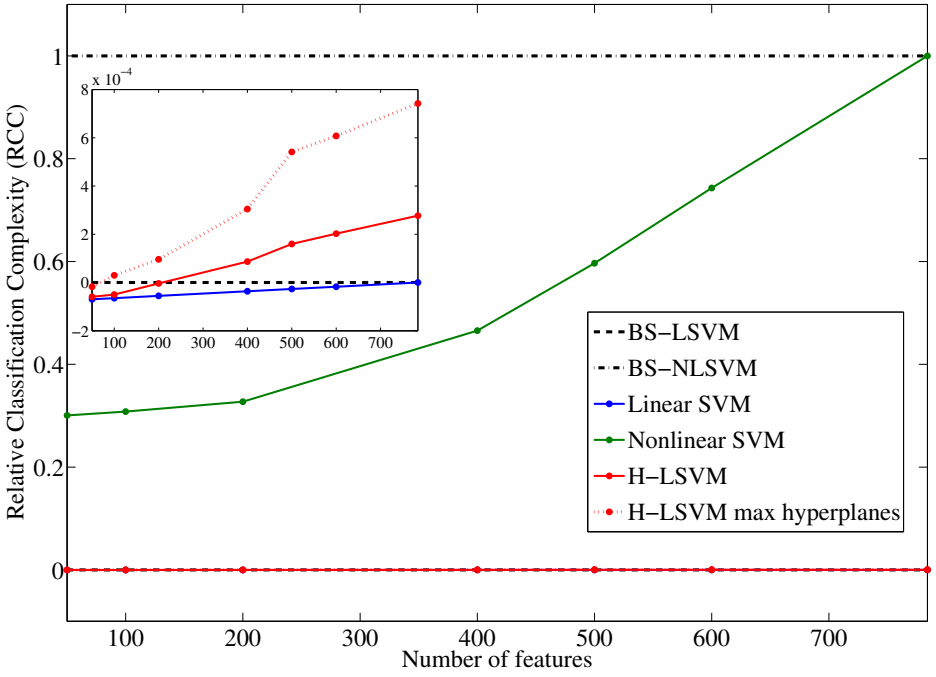


(b)

FIGURE 6.2: Classification error rate (Figure 6.2(a)) and number of operations needed to classify a test pattern (Figure 6.2(b)) corresponding to the baseline linear SVM with all features (BS-LSVM), the baseline nonlinear SVM with all features (BS-NLSVM) and the three architectures presented in Figure 6.1 combining QPFS with SVMs and H-LSVM. For H-LSVM, the maximum number of evaluations in classification is also provided. More detail in Table 6.2.



(a)

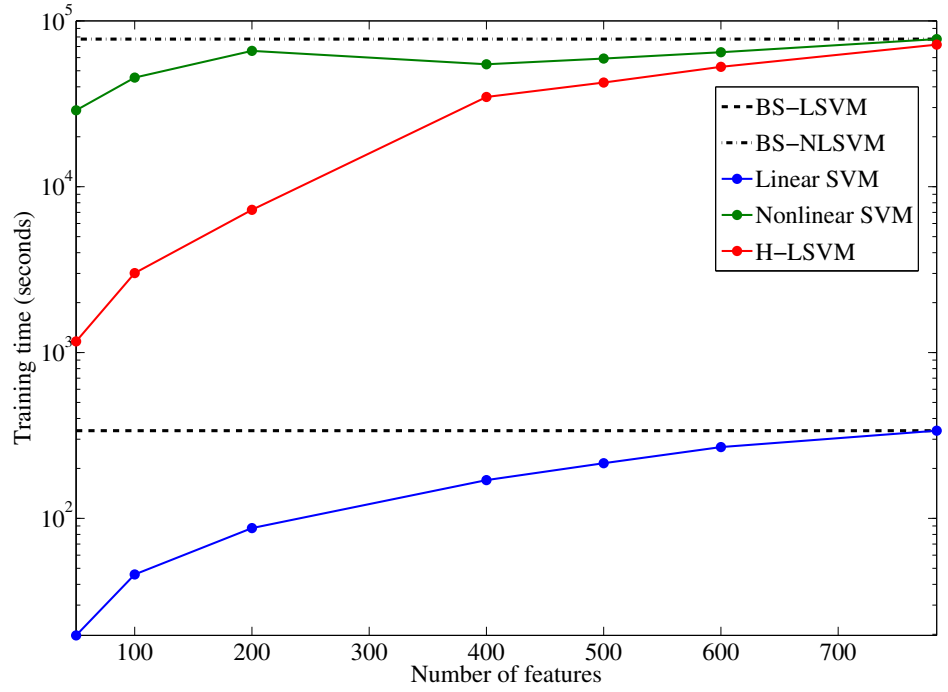


(b)

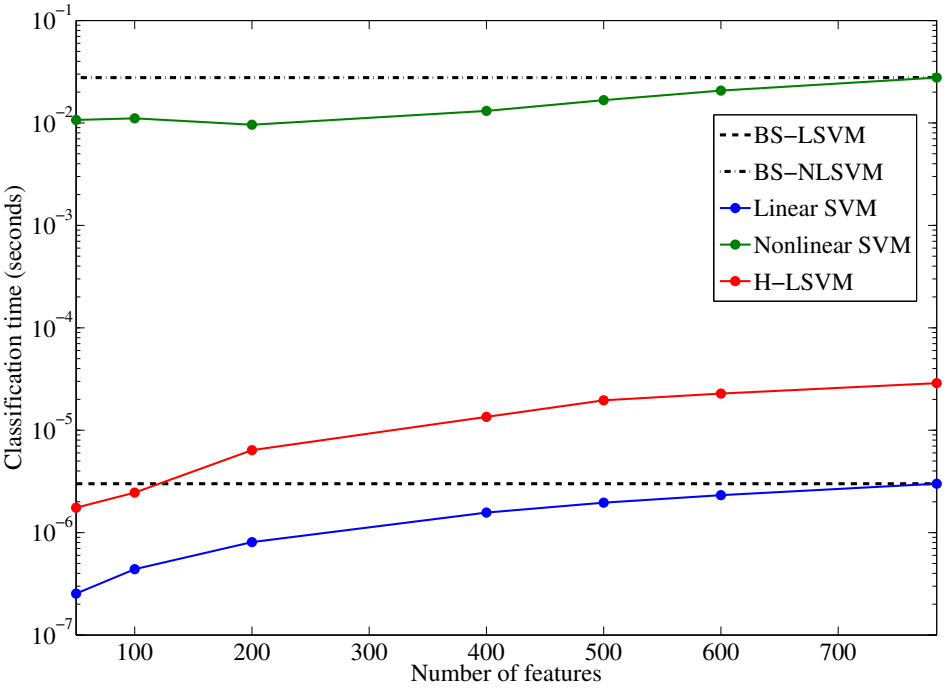
FIGURE 6.3: Relative Error (Figure 6.3(a) and Relative Classification Complexity (Figure 6.3(b)) corresponding to the baseline linear SVM with all features (BS-LSVM), the baseline nonlinear SVM with all features (BS-NLSVM) and the three architectures presented in Figure 6.1 combining QPFS with SVMs and H-LSVM. For H-LSVM, the maximum number of evaluations in classification is also provided. Note that values of these relative measures close to 0 reveal the equivalence of the model under consideration and the baseline linear SVM whereas values close to 1 show its proximity to the baseline nonlinear SVM. More detail in Table 6.2.

Feat.	**LINEAR SVM**				**NONLINEAR SVM**				**H-LSVM**			
	Training	*Test*	*RTT*	*RCT*	*Training*	*Test*	*RTT*	*RCT*	*Training*	*Test*	*RTT*	*RCT*
50	$1.97 \cdot 10^{1}$	$2.54 \cdot 10^{-7}$	$-4.1 \cdot 10^{-3}$	$9.91 \cdot 10^{-5}$	$2.89 \cdot 10^{4}$	$1.07 \cdot 10^{-2}$	$3.69 \cdot 10^{-1}$	$3.86 \cdot 10^{-1}$	$1.17 \cdot 10^{3}$	$1.76 \cdot 10^{-6}$	$1.07 \cdot 10^{-2}$	$-4.5 \cdot 10^{-5}$
100	$4.59 \cdot 10^{1}$	$4.40 \cdot 10^{-7}$	$-3.8 \cdot 10^{-3}$	$-9.24 \cdot 10^{-5}$	$4.55 \cdot 10^{4}$	$1.11 \cdot 10^{-2}$	$5.84 \cdot 10^{-1}$	$4.00 \cdot 10^{-1}$	$3.01 \cdot 10^{3}$	$2.46 \cdot 10^{-6}$	$3.46 \cdot 10^{-2}$	$-1.95 \cdot 10^{-5}$
200	$8.74 \cdot 10^{1}$	$8.08 \cdot 10^{-7}$	$-3.2 \cdot 10^{-3}$	$-7.91 \cdot 10^{-5}$	$6.60 \cdot 10^{4}$	$9.61 \cdot 10^{-3}$	$8.49 \cdot 10^{-1}$	$3.47 \cdot 10^{-1}$	$7.25 \cdot 10^{3}$	$6.38 \cdot 10^{-6}$	$8.94 \cdot 10^{-2}$	$1.22 \cdot 10^{-4}$
400	$1.70 \cdot 10^{2}$	$1.57 \cdot 10^{-6}$	$-2.2 \cdot 10^{-3}$	$-5.16 \cdot 10^{-5}$	$5.47 \cdot 10^{4}$	$1.31 \cdot 10^{-2}$	$7.04 \cdot 10^{-1}$	$4.73 \cdot 10^{-1}$	$3.48 \cdot 10^{4}$	$1.35 \cdot 10^{-5}$	$4.45 \cdot 10^{-1}$	$3.79 \cdot 10^{-4}$
500	$2.15 \cdot 10^{2}$	$1.96 \cdot 10^{-6}$	$-1.6 \cdot 10^{-3}$	$-3.76 \cdot 10^{-5}$	$5.93 \cdot 10^{4}$	$1.67 \cdot 10^{-2}$	$7.62 \cdot 10^{-1}$	$6.03 \cdot 10^{-1}$	$4.25 \cdot 10^{4}$	$1.96 \cdot 10^{-5}$	$5.45 \cdot 10^{-1}$	$5.99 \cdot 10^{-4}$
600	$2.69 \cdot 10^{2}$	$2.32 \cdot 10^{-6}$	$-9.00 \cdot 10^{-4}$	$-2.46 \cdot 10^{-5}$	$6.47 \cdot 10^{4}$	$2.07 \cdot 10^{-2}$	$8.33 \cdot 10^{-1}$	$7.47 \cdot 10^{-1}$	$5.28 \cdot 10^{4}$	$2.28 \cdot 10^{-5}$	$6.79 \cdot 10^{-1}$	$7.15 \cdot 10^{-4}$
784	$3.38 \cdot 10^{2}$	$3.00 \cdot 10^{-6}$	0	0	$7.77 \cdot 10^{4}$	$2.77 \cdot 10^{-2}$	1	1	$7.19 \cdot 10^{4}$	$2.88 \cdot 10^{-5}$	$9.25 \cdot 10^{-1}$	$9.32 \cdot 10^{-4}$

TABLE 6.3: Training and testing times (in seconds),*Relative Training Time (RTT)* and *Relative Classification Time (RCT)* for the machine learning system presented in Figure 6.1 consisting of QPFS feature selection combined with linear SVM, nonlinear SVM and H-LSVM classifiers.

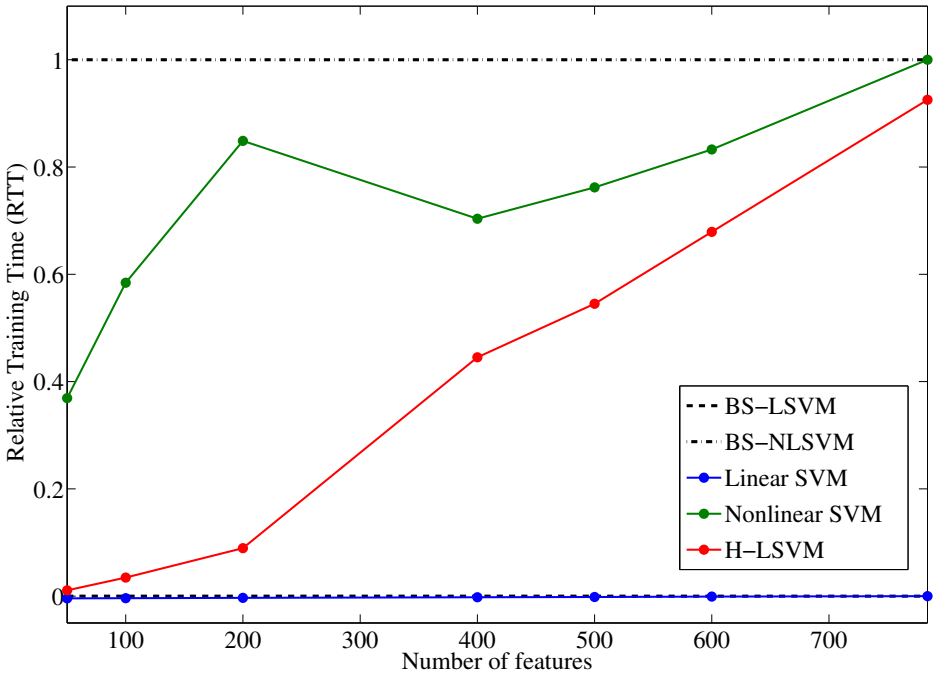


(a)

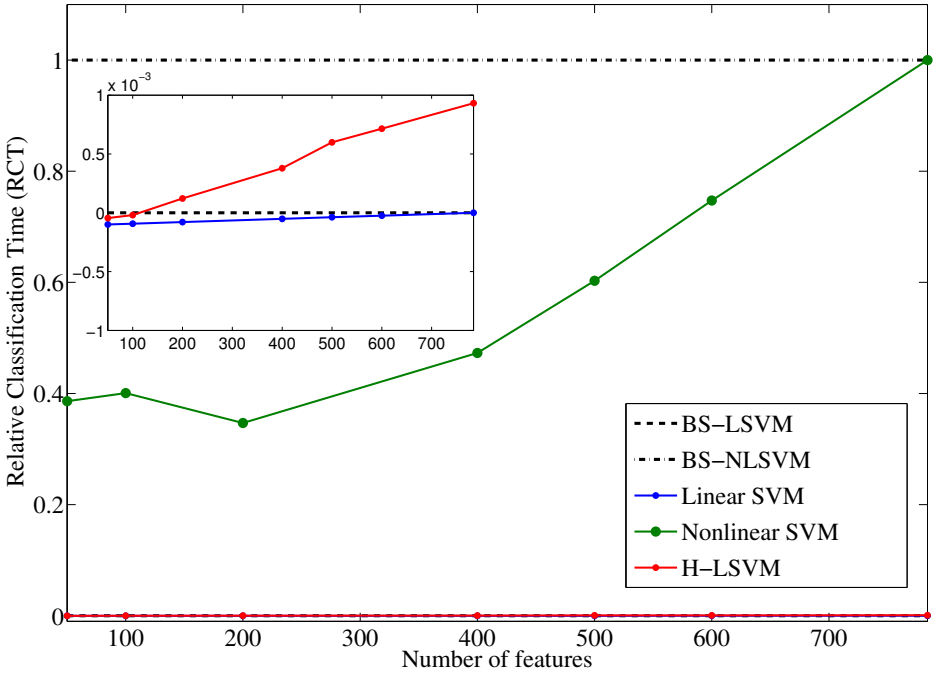


(b)

FIGURE 6.4: Training time (Figure 6.4(a)) and classification time (Figure 6.4(b)) in seconds corresponding to the baseline linear SVM with all features (BS-LSVM), the baseline nonlinear SVM with all features (BS-NLSVM) and the three architectures presented in Figure 6.1 combining QPFS with SVMs and H-LSVM. For H-LSVM, the maximum number of evaluations in classification is also provided. More detail in Table 6.3.



(a)



(b)

FIGURE 6.5: Relative Training Time (Figure 6.5(a)) and Relative Classification Time (Figure 6.5(b)) corresponding to the baseline linear SVM with all features (BS-LSVM), the baseline nonlinear SVM with all features (BS-NLSVM) and the three architectures presented in Figure 6.1 combining QPFS with SVMs and H-LSVM. Note that values of these relative measures close to 0 reveal the equivalence of the model under consideration and the baseline linear SVM whereas values close to 1 show its proximity to the baseline nonlinear SVM. More detail in Table 6.3.

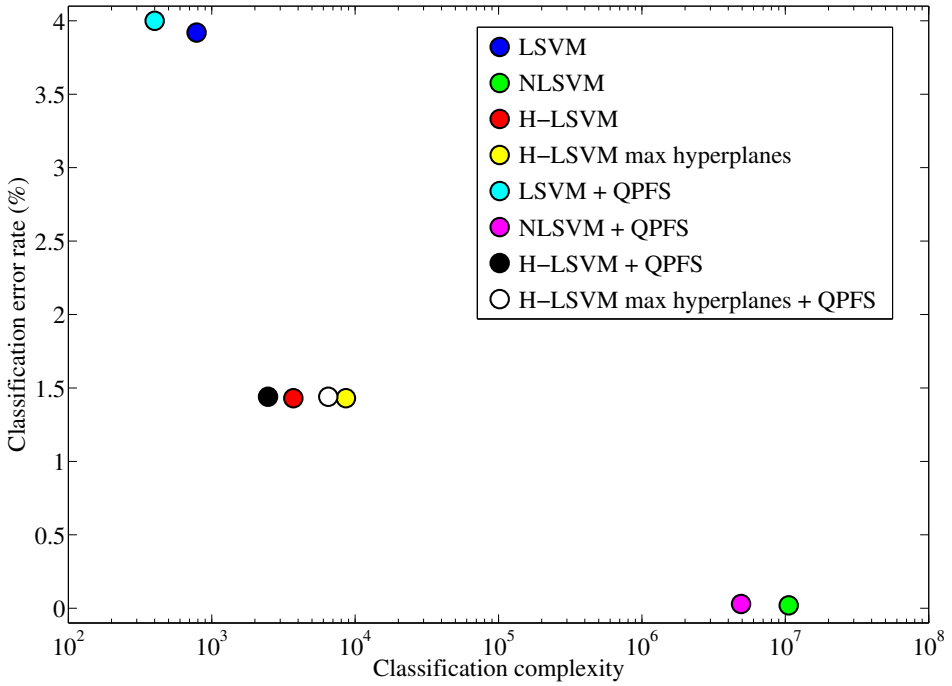


FIGURE 6.6: Classification complexity versus classification error rate for the different classifiers presented in Figure 6.1 linear SVM (LSVM), nonlinear SVM (NLSVM) and H-LSVM with QPFS feature selection. In addition, the linear SVM (BS-LSVM), nonlinear SVM (BS-LSVM) and H-LSVM performances without feature selection are also shown as well as the maximum number of hyperplanes encountered by the H-LSVM models in the classification phase.

6.3 Discussion

The preceding results give empirical evidence of the power of the QPFS+H-LSVM architecture and they also corroborate the conclusions derived in Chapters 3 and 5 about the behavior of QPFS and H-LSVM when acting separately. More precisely and following an incremental reasoning, a numerical analysis of the results reported in Tables 6.2–6.3 and shown in Figures 6.2–6.6 reveals that:

1. **QPFS** can lead to significant improvements in the complexity of the learning algorithm whenever the input space contains redundant or uninformative dimensions. In the *M3VOm8* dataset, both linear and nonlinear SVMs considering the first 400 features achieve the same classification error rate as their counterparts without feature selection (Figure 6.2(a)); the same figure shows that the H-LSVM algorithm with 500 features is enough to provide the same effectiveness as the model trained

with all the attributes. According to the results given in Table 6.3, the incorporation of QPFS yields a relative improvement in the classification speed of 48% in the linear SVM, 53% in the case of nonlinear SVM and 32% for H-LSVM whereas the relative acceleration in the training times of the three algorithms amount to 50%, 30% and 41%, respectively. These computational savings correspond to select in each case the minimum number of features maintaining the classification accuracy of the reference model with all features; that is, 400 variables for SVMs and 500 attributes for H-LSVM. Of course, if more decrease in the computation load are necessary, the number of features can be reduced until obtain an acceptable balance accuracy-complexity.

2. **H-LSVM** yields classification rates close to the nonlinear model with much lower classification cost. In absolute terms, the H-LSVM model trained with all features diminishes in 1.41% the accuracy of the nonlinear SVM model (Figure 6.2(a)) in return for a classification speed 10^3 times faster (Figure 6.2(b)). It can also be seen in Figure 6.2(a) that H-LSVM represents a plausible alternative capable of improving the performance of the linear model in 2.49% in absolute terms while providing the response times required by real-time classification systems (each test sample is classified in less than 0.1 milliseconds according to Figure 6.4(b)).

3. **QPFS+H-LSVM** preserving the first 500 features leads to a degradation in the performance of the nonlinear SVM without feature selection of 1.42% in absolute terms (Figure 6.2(a)). Nevertheless, the improvements in the classification speed are not inconsiderable: Table 6.3 shows that QPFS+H-LSVM is $1.4 \cdot 10^3$ times faster than the nonlinear SVM with all the features and it entails a classification complexity $4 \cdot 10^3$ times lower (Table 6.2). Compared against the performance of the nonlinear SVM with 400 features shown in Figure 6.2(a), the difference between misclassification rates holds while QPFS+H-LSVM has a classification complexity $2 \cdot 10^3$ times lower according to the results given in Table 6.2 ($6.7 \cdot 10^2$ times lower in terms of classification time looking at Table 6.3). The insufficiency of the linear model is reflected in its classification error rate close to 4% (Figure 6.2(a)). With not much effort in the classification process QPFS+H-LSVM can reduce this rate by more than a half. Figure 6.6 highlights the efficiency of QPFS+H-LSVM relative to the trade-off between accuracy and classification cost.

With regard to the relative measures shown in Figures 6.3 and 6.5, their behavior is exactly as expected: in the case of SVMs, they approach their reference values 0 (for the linear) and 1 (for the nonlinear) as the number of features increases; similarly, in the case of H-LSVM the more features used, the larger values for the relative measures. In addition, when more than 500 variables are considered by QPFS+H-LSVM, the algorithm achieves a relative error closer to the nonlinear SVM (Figure 6.3(a)) whilst the relative classification complexity (Figure 6.3(b)) and relative classification time (Figure 6.5(b)) are near to 0.

Finally, the training times of QPFS+H-LSVM are always below those of the nonlinear SVM as shown in Figures 6.4(a) and 6.5(a). Unfortunately, this behavior can not be considered a general rule applicable to any classification problem as it has been discussed in Section 5.3.

Chapter 7

Conclusions

This thesis has been focused on the design of new algorithms to classify datasets with large number of samples and/or high dimensions. The task has been tackled from two complementary points of view, each of them leading to improvements in the design of large-scale machine learning systems. On the one hand, as step preceding the classifier selection and training, dimensionality reduction can alleviate considerably the computational load of the underlying classifier in both the training and prediction phases. Nevertheless, the scalability of the feature selection algorithm is essential to guarantee its utility in high-dimensional domains. At the same time, the temporal restrictions of some of these large-scale systems make it necessary to revisit the existing algorithms and raise new solutions specially adapted to the new requirements. Traditionally, the efforts to improve the scalability of the classification algorithms have opted for creating fast algorithms in the training phase; however, several machine learning applications require prediction times of few milliseconds. Motivated by these ideas, this thesis have obtained the following results

- A new feature selection method named Quadratic Programming Feature Selection (QPFS) has been proposed in line with multivariate filter techniques. The new algorithm formulates the feature selection task as a quadratic problem in which the linear term represents the similarity between each feature and the class while the quadratic term takes into account dependences between each pair of variables. The QPFS algorithm admits any positive and symmetric similarity measure. Although

this work has limited the QPFS use to the Pearson correlation and mutual information, other authors have successfully applied QPFS considering other similarity measures [72].

- The improvement of the QPFS computational cost through the reformulation of the optimization problem in a lower dimensional space. The dimensionality reduction is performed by the Nyström method for matrix diagonalization which efficiently obtains an approximation of the eigenvectors and eigenvalues of the matrix in the QPFS quadratic term reducing the intensive computational cost of diagonalization. The reformulation of the quadratic programming in a lower dimensional space obtained through the Nyström diagonalization places QPFS among the most scalable multivariate filters.

- Experimental results show that the use of mutual information as similarity measure yields better classification rates than the Pearson correlation. Generally speaking, QPFS with mutual information provides similar results in terms of classification accuracy than state-of-the-art multivariate filters being slightly superior in domains with high levels of redundancy. Additionally, the incorporation of the Nyström method is able to maintain competitive classification rates while reducing at the same time the QPFS complexity. This fact puts QPFS at the top of multivariate filter techniques.

- The posing of a feature extraction method Kernel Quadratic Programming Feature Selection (KQPFS) to generate nonlinear features. The new method is arisen from the reformulation of the QPFS algorithm in a higher dimensional space induced by a kernel; hence the name. A theoretical proof of the equivalence between KQPFS and the popular Kernel Fisher Discriminant Analysis has been given and it shows a new understanding of KFDA and a new formulation of KFDA direction.

- Experimental results show the empirical equivalence of both methods as well as the computational superiority of KQPFS in classification problems with highly unbalanced classes.

- A new real-time classification algorithm, Hierarchical Linear Support Vector Machine (H-LSVM), consisting of a decision tree with linea SVM as splitting criterion in the nodes which takes advantage of the fast classification of both methods. This architecture generates piecewise linear decision functions and reduces significantly

the classification cost of the nonlinear SVMs. Based on the results of Golea et al. [137], a theoretical generalization error bound as a function of the tree structure and the Vapnik-Chervonenkis dimension of the SVMs hyperplanes is established.

- Experimental results show how the new algorithm obtains classification rates between the linear SVMs and the nonlinear ones, being in general closer to the nonlinear models. However, in terms of classification complexity H-LSVM achieves response times of few milliseconds outperforming the nonlinear SVM's speed in all cases and being even four orders of magnitude faster in some classification problems. Compared against other approaches based on the combination of decision trees and linear SVMs, the proposed algorithm reaches better classification results while keeping the same prediction complexity.

- An empirical analysis of the generalization error bound confirms its validity and establishes a connection between the complexity of the H-LSVM tree and the difference of the classification error in training and test sets. Furthermore, these results have shown the necessity of pruning the decision tree, as it was done, in order to prevent the overfitting phenomenon.

- An empirical proof of the suitability of the combination of QPFS and H-LSVM in a case in which the nonlinear SVM is too expensive in the classification process and a linear model is too simple.

- Public software implementations of the QPFS and H-LSVM algorithms. The QPFS software includes the Nyström approximation as well as mutual information and Pearson correlation as similarity measures. These codes are available at `https://sites.google.com/site/irenerodriguezlujan/documents/QPFS-1.0.zip` `https://sites.google.com/site/irenerodriguezlujan/HLSVM-1.1.zip`.

An overall analysis of the state-of-the-art methods, the proposed algorithms and the experimental results obtained reveals the need of including a new dimension in the evaluation of a large-scale system: the scalability. Until few years ago, many of the machine learning algorithms were concern about finding exact solutions for relatively complex models having as a result computationally expensive algorithms. Conversely, the results here presented suggest that sometimes it is necessary to sacrifice the accuracy of the model to a certain extend in favor of simple and scalable approximations, which not

only reduce the computational cost of the algorithms, but also improve their generalization abilities.

The results obtained illustrate that the use of simplifications of the data, as feature selection, the application of methods for numerical approximations, as the Nyström method, or the combination of simple methods and ideas to form more complex models, as in the case of H-LSVM, make it possible to design algorithms that, even not being exact, are able to face some problems of machine learning which were unapproachable by some of the conventional techniques.

7.1 Further Work

This thesis represents one step in a long way. There is still a lot to *learn* in large-scale classification since datasets which today are considered as *large-scale* will probably turn into the general trend in the near future. Therefore, it is worth investing more efforts in making large-scale classification practical and the ideas and advances presented in this work give rise to new approaches pursuing this goal:

- The QPFS has presented a new elegant and efficient solution to the feature selection task. However, the decision about the number of features to conform the final subset still depends on an extern process. It would be desirable to provide a feature selection algorithm that instead of ranking the features, provides a closed subset of them. This way, the entries of the solution vector to the QPFS quadratic programming would take binary values $\{0, 1\}$ indicating the membership of each feature to the final subset. Although there exists a branch in optimization theory exclusively dedicated to the optimization of $0 - 1$ quadratic programming problems [144, 145], these solutions tends to be computationally expensive and more efficient algorithms are needed for large-scale domains.

- The analysis of the QPFS computational complexity reveals that the main cost falls on the computation of the similarity matrix between each pair of features when the number of samples is large compared to the number of dimensions – a common situation in large-scale problems –. In the same way as the Nyström approximation has alleviated the diagonalization cost, it would be interesting to consider some

numerical approaches or other similarity measures in order to improve the QPFS scalability.

- Chapter 4 started with an analysis of different techniques to generate new features from the original ones. Two opposite approaches can be distinguished according to the process used to extract new attributes. On the one hand, new variables can be created in a implicit manner by means of the reformulation of the feature selection algorithm in a kernel space – as in KFDA and KQPFS –. Conversely, new attributes can be explicitly generated to apply afterwards the original feature selection method in the expanded space. From the point of view of large-scale classification, the latest option is really appealing because the explicit knowledge of the feature space makes its use as input of a simple linear model possible. One alternative to explicitly increase the number of variables without expert knowledge is to generate them randomly according to certain rules based on Grammatical Evolution techniques [89]. Unfortunately, handling such amount of features is inaccesible even for the most scalable feature selection methods. Therefore, it would be of interest to combine QPFS with the grammatical feature selection process controlling the dimensionality explosion somehow. For example, generate the new features in a sequential manner or divide the feature space into subgroups to distribute the QPFS computation could be plausible solutions.

- The first option discussed in the preceding point and consisting in reformulating the feature selection algorithm in a kernel space is the alternative adopted in Chapter 4 leading to KQPFS. Lamentably, this approach turns out to have the same scalability drawbacks that nonlinear SVMs in large-scale problems: (i) expensive training and test processes if many Lagrange multipliers are not zero and (ii) the kernel matrix storage. If instead of applying regularization to the Lagrange multipliers it is imposed to the hyperplane vector $\boldsymbol{w}$, the problem is shown to be equivalent to Least Squares Support Vector Machines [99] and thus, efficient SMO-style methods can be used to solve it [49] although the slow prediction phase is still an obstacle. However, the use of SMO-style solvers with regularization in the Lagrange multipliers is not so straightforward and more efforts continuing the work started by Mika [42] are necessary.

- Even though the aim of the H-LSVM algorithm was focused on speeding up the prediction phase of nonlinear SVMs, it would be interesting to make an effort to

parallelize the training algorithm taking advantage of the decision tree structure of the model.

- It is known that the value of the regularization parameter affects critically the performance of SVMs. In this regard, the H-LSVM algorithm divides the input space into disjoint regions using the same regularization parameter in all the splits. Nevertheless, even being drawn from the same statistical distribution, each of these regions might have different properties being interesting to explore procedures to set different regularization parameters in the nodes of the tree. This task must be carefully guided as solutions evaluating each possible combination of parameters in the tree are absolutely unfeasible.

7.2 Conclusiones

Esta tesis se ha centrado en el diseño de nuevos algoritmos para la clasificación de bases de datos con gran número de patrones y/o dimensiones desde dos puntos de vista complementarios y que constituyen sendas mejoras en el diseño de un sistema de clasificación a gran escala. Por una parte, como paso previo a la selección y entrenamiento de un algoritmo de aprendizaje automático, la reducción de las dimensiones de los patrones puede aliviar considerablemente la carga computacional del clasificador subyacente, tanto en entrenamiento como en predicción. No obstante, es fundamental que el algoritmo de selección de variables sea escalable para garantizar su utilidad en dominios de altas dimensiones. Al mismo tiempo, las limitaciones temporales de algunos de estos sistemas a gran escala requieren revisar los algoritmos existentes y plantear nuevas soluciones adaptadas a los nuevos requisitos. Tradicionalmente, gran parte de los avances realizados en cuanto a la escalabilidad de los sistemas se han centrado en producir algoritmos rápidos en la fase de entrenamiento. Sin embargo, también deben considerarse otras aplicaciones del aprendizaje automático que precisan tiempos de predicción de pocos milisegundos. Motivada por estas ideas, esta tesis ha obtenido los siguientes resultados:

- Propuesta de un nuevo método de selección de variables, Quadratic Programming Feature Selection (QPFS), enmarcado dentro de los métodos de filtro multivariable. El nuevo algoritmo formula la selección de variables como la optimización de un problema cuadrático donde el término lineal representa la similitud de cada

variable con la clase y el término cuadrático modela las dependencias entre pares de variables. El algoritmo QPFS admite cualquier medida de similitud positiva y simétrica, restringiéndose este trabajo a la correlación de Pearson y la información mutua. No obstante, ha sido aplicado con éxito por otros autores utilizando medidas de similitud distintas a las aquí propuestas [72].

- Mejora del coste computacional del algoritmo QPFS mediante la reformulación del problema de optimización en un subespacio de menor dimensión. La reducción de dimensión se hace mediante el método Nyström para la diagonalización de matrices que permite obtener eficientemente una solución aproximada de los autovalores y autovectores de la matriz del término cuadrático, reduciendo así el coste computacional de la diagonalización. La resolución del problema cuadrático en un subespacio de menor dimensión obtenido mediante la diagonalización de Nyström situa al QPFS como uno de los métodos de filtro multivariantes con menor coste computacional.

- Resultados experimentales muestran que el uso de QPFS con información mutua produce mejores aciertos en clasificación que la version con correlaciones. En términos generales, iguala el estado del arte en métodos de filtro multivariantes siendo ligeramente mejor en problemas con alto nivel de redundancia entre variables. Además, la incorporación del método Nyström es capaz de mantener los resultados competitivos del algoritmo inicial reduciendo, a su vez, su coste computacional y situándolo a la cabeza de las técnicas de filtro multivariables.

- Reformulación del método QPFS en un espacio de dimensión mayor inducido por un núcleo (kernel) para generar atributos no lineales, dando lugar al algoritmo de generación de variables Kernel Quadratic Programming Feature Selection (KQPFS). Se ha dado una demostración teórica de la equivalencia entre este nuevo método y el popular discriminante lineal de Fisher para núcleos que arroja una nueva interpretación para el discriminante de Fisher además de una nueva forma de cálculo.

- Resultados experimentales demuestran la equivalencia empírica de ambos métodos así como la superioridad computacional de la nueva solución para problemas de clasificación con clases altamente desbalanceadas.

- Propuesta de un nuevo algoritmo para clasificación en tiempo real, Hierarchical Linear Support Vector Machine (H-LSVM), formado por un árbol de decisión con Máquinas de Vectores Soporte (SVM) lineales en los nodos, aprovechando así la velocidad de predicción de ambas técnicas. Esta arquitectura genera fronteras de decisión lineales a trozos (no lineales) pero con un coste en clasificación significativente inferior al de las SVMs no lineales. Basándose en los resultados de Golea et al. [137] se deduce, además, una cota teórica del error de generalización del nuevo método en función de la estructura del árbol y la dimensión de Vapnik-Chervonenkis de los hiperplanos.

- Resultados experimentales muestran como el nuevo algoritmo obtiene aciertos de clasificación situados entre las SVMs lineales y no lineales, siendo por lo general, más cercanos a áquellos obtenidos por los modelos no lineales. Sin embargo, en términos de coste en clasificación, H-LSVM es capaz de emitir una predicción en pocos milisegundos acelerando el tiempo de las SVMs no lineales en todos los casos y consiguiendo aceleraciones de incluso cuatro órdenes de magnitud para algunos problemas. Comparado con otras aproximaciones basadas en la combinación de árboles de decisión y SVMs lineales, el algoritmo propuesto obtiene mejores resultados en clasificación manteniendo la misma velocidad de predicción.

- El análisis empírico de la cota del error de generalización propuesta confirma su validez y establece una relación entre la complejidad del árbol H-LSVM y la diferencia del error de clasificación en entrenamiento y test. Adicionalmente, ha quedado patente la necesidad de un proceso de poda del árbol, tal y como se ha realizado, para evitar el sobreajuste de los datos de entrenamiento.

- Una prueba empírica de la idoneidad de la combinación de QPFS y H-LSVM en un caso en el que la SVM no lineal es demasiado costosa en la fase de clasificación y un modelo lineal es demasiado simple.

- Implementación de software público de los algoritmos QPFS y H-LSVM. En el caso de QPFS, el programa incluye la aproximación de Nyström así como la información mutua y la correlación de Pearson como medidas de similitud. Estos códigos están disponibles en:
  `https://sites.google.com/site/irenerodriguezlujan/documents/QPFS-1.0.zip`
  `https://sites.google.com/site/irenerodriguezlujan/HLSVM-1.1.zip`.

Un análisis en conjunto de los métodos que conforman el estado del arte, los algoritmos propuestos y los resultados experimentales obtenidos refleja la necesidad de incluir una nueva dimensión en la evaluación de los modelos: la escalabilidad. Hasta hace unos años, gran parte del aprendizaje automático se había centrado en la consecución de soluciones exactas para modelos relativamente complejos lo que desencadenaba en algoritmos computacionalmente costosos. Sin embargo, los resultados derivados de esta tesis sugieren que en ciertas ocasiones es preciso sacrificar hasta cierto punto la tasa de acierto del modelo en favor de aproximaciones simples y escalables que además de reducir el coste computacional de los algoritmos también mejoren su capacidad de generalización.

Los resultados aquí obtenidos ilustran que el uso de simplificaciones de los datos, como en el caso de la selección de variables, la aplicación de métodos de aproximación numérica, como el caso del método Nyström, o la combinación de métodos e ideas simples para formar modelos más complejos, como en el caso de H-LSVM, permiten diseñar algoritmos que, aunque no exactos, en la práctica afrontan satisfactoriamente problemas de aprendizaje automático inabordables por algunas de las tecnicas convencionales.

Appendix A

Notation

N	Number of the training samples
M	Dimension of the training samples
S	Training set
S_+	Subset of samples labeled as positive in the training set
S_-	Subset of samples labeled as negative in the training set
N_+	Number of samples labeled as positive in the training set
N_-	Number of samples labeled as negative in the training set
$(\boldsymbol{x_i}, y)$	Pair sample-label for the i-th sample in the training set
$\mathcal{X}$	Input space
$\mathcal{Y}$	Output space
x_i^j	j-th feature in the i-th training sample
X_j	Random variable corresponding to the j-th feature
Y	Random variable representing the supervised classification targets
$l(f(\boldsymbol{x}, y)$	Loss function for the supervised method f
$R(f)$	Expected error or risk
$R_{\text{emp}}(f)$	Empirical error or risk
$R_{\text{reg}}(f)$	Regularized error or risk
$\mathbb{E}_P$	Expectation with respect to the distribution P
$VCdim$	Vapnik-Chervonenkis dimension

Appendix B

The Nyström Method for matrix diagonalization

The Nyström method is an efficient technique to generate low-rank matrix approximations and it has been widely used in large-scale learning applications [75, 77, 146]. The method approximates the eigenvalues and eigenvectors of a M-dimensional **Gram matrix** diagonalizing a $k \times k$ matrix with $k \ll M$ by means of the Nyström extension.

Initially the Nyström method was used to find numerical approximations to **eigenfunction problems** of the form

$$\int_a^b K(x,y)\phi(y)dy = \lambda\phi(x). \tag{B.1}$$

Dividing the interval $[a,b]$ up to n evenly spaced points $\xi_1, \xi_2, \ldots, \xi_n$ and approximating the above integral by the simple quadrature rule [78], the following expression approximates Equation B.1,

$$\frac{(b-a)}{n}\sum_{j=1}^{n} K(x,\xi_j)\hat{\phi}(\xi_j) = \lambda\hat{\phi}(x) \tag{B.2}$$

being $\hat{\phi}(x)$ an approximation of $\Phi(x)$ and making $x = \xi_i$ in Equation B.2,

$$\frac{(b-a)}{n}\sum_{j=1}^{n} K(\xi_i,\xi_j)\hat{\phi}(\xi_j) = \lambda\hat{\phi}(\xi_i) \text{ , for } i = 1,\ldots,n. \tag{B.3}$$

Without loss of generality, let $a = 0$ and $b = 1$ then, the system described in Equation B.3 can be rewritten as,

$$K\hat{\Phi} = n\hat{\Phi}\Lambda \tag{B.4}$$

where $K_{ij} = K(\xi_i,\xi_j)$ is a Gram matrix, Λ a diagonal matrix with the approximated eigenvalues $\{\lambda_1,\lambda_2,\ldots,\lambda_n\}$ of K and $\Phi = [\phi_1,\phi_2,\ldots,\phi_n]$ the associated eigenvectors. Plugging the previous equality in Equation B.2 yields the Nyström extension for each $\hat{\phi}_i$,

$$\hat{\phi}_i(x) = \frac{1}{n\lambda_i}\sum_{j=1}^{n} K(x,\xi_j)\hat{\phi}_i(\xi_j). \tag{B.5}$$

The **Nyström method for matrix diagonalization** can be formulated starting from the previous result. Let $Q \in \mathbb{R}^{M\times M}$ a symmetric matrix decomposed as,

$$Q = \begin{pmatrix} A & B \\ B^T & E \end{pmatrix} \tag{B.6}$$

where $A \in \mathbb{R}^{k\times k}$, $B \in \mathbb{R}^{k\times m}$, and $E \in \mathbb{R}^{m\times m}$ being $k \ll m$ and $k + m = M$. As Q is a Gram matrix, it can be rewritten as the inner product of a matrix Z by itself: $Q = Z^T Z$. If Q is of rank k and the rows of the submatrix $[A \quad B]$ are linearly independent, Z can be defined as a function of the submatrices A and B. Let $Z = [X \quad Y]$ with $X \in \mathbb{R}^{M\times k}$ and $Y \in \mathbb{R}^{M\times m}$, Q can be reformulated as follows

$$Q = Z^T Z = \begin{pmatrix} X^T X & X^T Y \\ Y^T X & Y^T Y \end{pmatrix}. \tag{B.7}$$

Making equal each block in Equations B.6 and B.7 yields $A = X^T X$ and $B = X^T Y$. Let $A = U \Lambda U^T$ the diagonalization of matrix A with $U^T U = I_M$ and where I_M is the M-dimensional identity matrix. By defining

$$\bar{X} = \Lambda^{\frac{1}{2}} U^T \tag{B.8}$$

$$\hat{Y} = (\bar{X}^T)^{-1} B = \Lambda^{-\frac{1}{2}} U^T B \tag{B.9}$$

$$\hat{Z} = [\bar{X} \quad \hat{Y}], \tag{B.10}$$

with $\bar{X} \in \mathbb{R}^{k \times k}$, $\hat{Y} \in \mathbb{R}^{k \times m}$ and $\hat{Z} \in \mathbb{R}^{k \times M}$. Then, matrix $\hat{Q}$ can be written as

$$\hat{Q} = \hat{Z}^T \hat{Z} = \begin{pmatrix} \bar{X}^T \bar{X} & \bar{X}^T \hat{Y} \\ \hat{Y}^T \bar{X} & \hat{Y}^T \hat{Y} \end{pmatrix} \tag{B.11}$$

$$= \begin{pmatrix} \bar{X}^T \bar{X} & \bar{X}^T \Lambda^{-\frac{1}{2}} U^T B \\ (\Lambda^{-\frac{1}{2}} U^T B)^T \bar{X} & (\Lambda^{-\frac{1}{2}} U^T B)^T \Lambda^{-\frac{1}{2}} U^T B \end{pmatrix} \tag{B.12}$$

$$= \begin{pmatrix} A & B \\ B^T & B^T A^{-1} B \end{pmatrix}. \tag{B.13}$$

If the rank of matrix Q is larger than k or the rows of submatrix $[A \quad B]$ are not linearly independent, $\hat{Q}$ is an approximation of matrix Q whose quality can be quantified as $\|E - B^T A^{-1} B\|$.

Given the previous approximation of Q, $\hat{Q}$ and the A matrix diagonalization, $A = U \Lambda U^T$, it is possible to calculate the eigenvectors of $\hat{Q}$, $\hat{Q} = \bar{U} \Lambda \bar{U}^T$, by means of the Nyström extension for matrix diagonalization; the jth coordinate of the ith eigenvector of $\hat{Q}$ is given by,

$$\bar{U}_{ji} = \frac{1}{\lambda_i} \sum_{n=1}^{k} Q_{jn} U_{ni}, \tag{B.14}$$

being $\lambda_1, \lambda_2, \ldots, \lambda_k$ the eigenvalues of A, Q_{jn} the element of Q in the j-th row and the n-th column and U_{ni} the n-th coordinate of the i-th eigenvector of A. Then, the i-th eigenvector of Q is given by,

$$\bar{U}_i = \begin{pmatrix} \frac{1}{\lambda_i} \sum_{n=1}^{k} Q_{1n} U_{ni} \\ \vdots \\ \frac{1}{\lambda_i} \sum_{n=1}^{k} Q_{Mn} U_{ni} \end{pmatrix} = \frac{1}{\lambda_i} \begin{pmatrix} A \\ B^T \end{pmatrix} U. \tag{B.15}$$

And thus,

$$\bar{U} = \begin{pmatrix} A \\ B^T \end{pmatrix} U\Lambda^{-1} = \begin{pmatrix} AU\Lambda^{-1} \\ B^T U\Lambda^{-1} \end{pmatrix} = \begin{pmatrix} U \\ B^T U\Lambda^{-1} \end{pmatrix}. \tag{B.16}$$

Finally, the columns of matrix $\bar{U}$ are orthogonalized. This is address as follows, let $A^{\frac{1}{2}}$ the symmetric positive definite square root of A, let $S = A + A^{-\frac{1}{2}} B B^T A^{-\frac{1}{2}}$ and its diagonalization $S = R\hat{\Lambda}R^T$. Defining the matrix $\hat{V}$ as

$$\hat{V} = \begin{pmatrix} A \\ B^T \end{pmatrix} A^{-\frac{1}{2}} R \hat{\Lambda}^{-\frac{1}{2}}, \tag{B.17}$$

it can be shown that $\hat{V}$ and $\hat{\Lambda}$ diagonalize $\hat{Q}$, i.e. $\hat{Q} = \hat{V}\hat{\Lambda}\hat{V}^T$ and $\hat{V}^T\hat{V} = I_k$.

B.1 Nyström Sampling Techniques

One of the critical points in the Nyström approximation is to determine the sampling criterion to select the rows forming the submatrix $[A \quad B]$ since the quality of the approximation $\hat{Q}$ is quantified as $\|E - B^T A^{-1} B\|$,depending directly on the choice of the $[A \quad B]$ submatrix. In the work developed by Kumar et al. [76], different sampling techniques for the Nyström method including theoretical and empirical points of view are analyzed in depth. The sampling methods considered are fixed throughout the subsampling process leaving aside those adaptive techniques which modify their selection criteria in function of previous choices. Although adaptive methods yield in general better results, they are quite expensive in practice [82, 147]. Several non-adaptive sampling techniques have been studied in the literature and they can be divided into two groups,

- **Uniform sampling without replacement**. This is the sampling method proposed in the paper which introduced the Nyström method for machine learning applications [77] and the most commonly used in practice [75, 148–150].
- **Non-uniform sampling**. A weight is associated to each row in such a way that rows with the highest weights are more likely to be selected. For instance, these weights can be estimated as the L2 norm of the row (column-norm sampling) or the corresponding diagonal element of the Nyström matrix $\hat{Q}$ (diagonal sampling) [146, 151].

Kumar results suggest that uniform sampling without replacement apart from being more efficient in both time and space, produces more accurate approximations.

Appendix C

Convergence Properties of Weighted-Pegasos Algorithm

The aim of this appendix is to show that the bound for the **norm of the optimal solution** of the Weighted-Pegasos algorithm presented in Chapter 5 is different than that of the standard Pegasos algorithm [56]. However, this modification has no effect in the number of iterations needed for convergence which remains as in the original version. A deep analysis of the mathematical results used further down is beyond the scope of this thesis, more details can be found in [56, 152].

The analysis of the convergence of the Pegasos algorithm relies on the notion of **strongly convex functions** [152]. Recall the approximate objective function of the Weighted-Pegasos algorithm:

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}; A_t) = \\ \min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \frac{1}{k}\sum_{(\nu,\boldsymbol{x},y)\in A_t} \nu \max\{0, 1 - y(\boldsymbol{w}\cdot\boldsymbol{x})\}.$$

According to Lemma 1 in [152], the above function is λ-strongly convex because it can be represented as $g = \lambda\|\boldsymbol{w}\|_2^2 + h(\boldsymbol{w})$ with $\|\boldsymbol{w}\|_2^2$ differentiable and convex and $h(\boldsymbol{w}) = \sum_{(\nu,\boldsymbol{x},y)\in A_t} \nu \max\{0, 1 - y(\boldsymbol{w}\cdot\boldsymbol{x})\}$ convex with respect to $\boldsymbol{w}$. The h convexity is guaranteed by the convexity of each term in the summation as shown in Figure C.1.

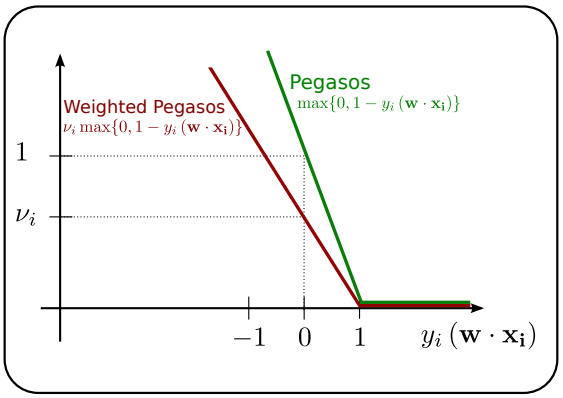


FIGURE C.1: Convexity of the loss functions for the Pegasos and the weighted Pegasos algorithms.

Therefore, all the convergence theory applied by the Pegasos' authors holds here. However, the weights in the loss term slightly change the bound of the $\boldsymbol{w}$ norm and hence, the projection step.

Lemma C.1. *The optimal weight vector* $\boldsymbol{w}^*$ *that minimizes the primal weighted SVM problem satisfies* $\|\boldsymbol{w}\|^2 \leq \frac{1}{\sqrt{N\lambda}}$.

Proof. The strong duality theorem states that in the minimization of a convex function subject to a set of linear constraints, the value of the optimal primal and dual solutions meet. The optimization problem in of Weighted-Pegasos:

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 + \frac{1}{N}\sum_{i=1}^{N} \nu_i \max\{0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x_i})\} \tag{C.1}$$

satisfies these conditions. The dual problem of Equation C.1 can be written as:

$$\begin{aligned} \max_{\alpha \in [0, \frac{\nu_i}{N}]^N} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2\lambda}\|\alpha_i y_i \boldsymbol{x_i}\|_2^2 \\ \text{s.t.} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0. \end{aligned}$$

If $\boldsymbol{w}^*$ and $\boldsymbol{\alpha}^*$ are the optimal primal and dual solutions respectively, the following equality holds:

$$\frac{\lambda}{2}\|\boldsymbol{w}^*\|_2^2 + \frac{1}{N}\sum_{i=1}^{N}\nu_i \max\{0, 1 - y_i(\boldsymbol{w}^* \cdot \boldsymbol{x_i})\} \quad = \quad \sum_{i=1}^{N}\alpha_i^* - \frac{1}{2\lambda}\|\alpha_i^* y_i \boldsymbol{x_i}\|^2. \quad \text{(C.2)}$$

At the optimum $\boldsymbol{w}^* = \frac{1}{\lambda}\sum_{i=1}^{N}\alpha_i^* y_i \boldsymbol{x_i}$ by imposing that the gradient of the Lagrangian with respect to $\boldsymbol{w}^*$ must be equals zero. Plugging this equality into the Equation C.2 and rearranging terms:

$$\frac{\lambda}{2}\|\boldsymbol{w}^*\|_2^2 + \frac{1}{N}\sum_{i=1}^{N}\nu_i \max\{0, 1 - y_i(\boldsymbol{w}^* \cdot \boldsymbol{x_i})\} \quad = \quad \sum_{i=1}^{N}\alpha_i^* - \frac{\lambda}{2}\|\boldsymbol{w}^*\|_2^2.$$

Working out $\|\boldsymbol{w}^*\|_2^2$:

$$\|\boldsymbol{w}^*\|_2^2 = \frac{1}{\lambda}\left(\sum_{i=1}^{N}\alpha_i^* - \frac{1}{N}\sum_{i=1}^{N}\nu_i \max\{0, 1 - y_i(\boldsymbol{w}^* \cdot \boldsymbol{x_i})\}\right).$$

Taking into account that the term $\frac{1}{N}\sum_{i=1}^{N}\nu_i \max\{0, 1 - y_i(\boldsymbol{w}^* \cdot \boldsymbol{x_i})\}$ is positive and the Lagrange multipliers are upper bounded by $\frac{\nu_i}{N}$:

$$\|\boldsymbol{w}^*\|^2 \leq \frac{1}{\lambda}\sum_{i=1}^{N}\frac{\nu_i}{N}.$$

Finally, according to the weighting scheme proposed in Chapter 5 (Equation 5.3):

$$\nu_i = \begin{cases} \frac{1}{2N^+} & \text{if } \boldsymbol{x_i} \in S_+ \\ \frac{1}{2N^-} & \text{if } \boldsymbol{x_i} \in S_- \end{cases}$$

the desired bound is obtained,

$$\|\boldsymbol{w}^*\|_2^2 \leq \frac{1}{N\lambda}\left(\sum_{(\boldsymbol{x_i}, y_i)\in S_+}\frac{1}{2N_+} + \sum_{(\boldsymbol{x_i}, y_i)\in S_-}\frac{1}{2N_-}\right) = \frac{1}{N\lambda}$$

□

It is worth noting that the above bound is more restrictive than those of the standard Pegasos algorithm: $\|\boldsymbol{w}^*\|^2 \leq \frac{1}{\lambda}$ [56]. According to the bound of Lemma C.1, the theorem establishing the convergence rate in expectation for the Pegasos algorithm ([56, Theorem 2]) can be adapted as follows,

Theorem C.2. *Assume that for all* $(\boldsymbol{x}, y) \in S$ *the norm of* $\boldsymbol{x}$ *is at most* R *and for all* t, A_t *is chosen identically and independently distributed from* S. *Let* r *an integer picked uniformly at a random from* $[1, 2, \ldots, T]$. *Then,*

$$\mathbb{E}_{A_1, A_2, \ldots, A_T} \mathbb{E}_r[f(\boldsymbol{w_r})] \leq f(\boldsymbol{w^*}) + \frac{c \log(T)}{\lambda T}$$

where $c = \left(\sqrt{\frac{\lambda}{N}} + R\right)^2$.

Then, the number of iterations requires for achieving a solution of accuracy $\epsilon = f(\boldsymbol{w}) - f(\boldsymbol{w}^*)$ is $\tilde{O}\left(\frac{R^2}{\lambda\epsilon}\right)$ where $\tilde{O}$ denotes the stochastic component of the bound. Although the value of the c constant is different from that of the standard Pegasos algorithm ($c = (\sqrt{\lambda} + R)^2$), the bound in terms of $\tilde{O}$ is the same in both cases.

Bibliography

[1] R. O. Duda, P.E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[2] T. Li, C. Zhang, and M. Ogihara. A comparative study of feature selection and multiclass classification methods for tissue classification based on gene expression. *Bioinformatics*, 20:2429–2437, 2004.

[3] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1, 2002.

[4] S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7):1029–1058, 1992.

[5] J. Fiérrez-Aguilar, J. Ortega-Garcia, D. Ramos, and J. Gonzalez-Rodriguez. HMM-based on-line signature verification: Feature extraction and signature modeling. *Pattern Recognition Letters*, 28(16):2325–2334, 2007.

[6] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of IEEE*, volume 77, pages 257–286, 1989.

[7] J. R. Dorronsoro, F. Ginel, C. Sgnchez, and C. S. Cruz. Neural fraud detection in credit card operations. *Neural Networks, IEEE Transactions on*, 8(4):827–834, 1997.

[8] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(1):4–37, 2000.

[9] K. Oh and K. Jung. GPU implementation of neural networks. *Pattern Recognition*, 37(6):1311–1314, 2004.

[10] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *ICML '09: Proceedings of the 26th international conference on Machine learning*, volume 382, page 110, 2009.

[11] K.D. Lillywhite, D-J. Lee, and D. Zhang. Real-time human detection using histograms of oriented gradients on a GPU. In *Workshop on Applications of Computer Vision (WACV)*, pages 1–6. IEEE Computer Society, 2009.

[12] J. Platoš and P. Gajdoš. Large data real-time classification with non-negative matrix factorization and self-organizing maps on GPU. In *International Conference on Computer Information Systems and Industrial Management Applications (CISIM)*, pages 176–181, 2010.

[13] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. In *IEEE Neural Networks*, volume 12, pages 181–201, 2001.

[14] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition.* Nauka, Moscow, 1974. (in Russian).

[15] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-posed problems.* W.H. Winston, 1977.

[16] G. Forman. BNS feature scaling: an improved representation over TF-IDF for SVM text classification. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge mining*, pages 263–270, 2008.

[17] K. Momen, S. Krishnan, and T. Chau. Real-time classification of forearm electromyographic signals corresponding to user-selected intentional movements for multifunction prosthesis control. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 15(4):535–542, 2007.

[18] P. Shenoy, K. J. Miller, B. Crawford, and R. N. Rao. Online electromyographic control of a robotic prosthesis. *IEEE Transactions on Biomedical Engineering*, 55 (3):1128–1135, 2008.

[19] J. Rodriguez, A. Goni, and A. Illarramendi. Real-time classification of ECGs on a PDA. *IEEE Transactions on Information Technology in Biomedicine*, 9(1):23–34, 2005.

[20] C. Ding and H. Peng. Minimum redundancy feature selection from microarray gene expression data. *Journal Bioinformatics and Computational Biology*, 3(2): 185–205, April 2005.

[21] Y. Saeys, I. Inza, and P. Larrañaga. A review of feature selection techniques in bioinformatics. *Bioinformatics*, 23(19):2507–2517, 2007.

[22] Y. Zhang, C. Ding, and T. Li. Gene selection algorithm by combining reliefF and mRMR. *BMC Genomics*, 9(Suppl 2), 2008.

[23] R. Leitner, H. Mairer, and A. Kercek. Real-time classification of polymers with NIR spectral imaging and blob analysis. *Real-Time Imaging*, 9:245–251, 2003.

[24] C. Phua, V. C. S. Lee, K. Smith-Miles, and R. W. Gayler. A comprehensive survey of data mining-based fraud detection research. *CoRR*, 2010.

[25] M. A. Hall. *Correlation-based feature selection for machine learning.* PhD thesis, Department of Computer Science, Waikato University, New Zealand, 1999.

[26] D. J. C. MacKay. *Information theory, inference, and learning algorithms.* Cambridge University Press, 2003.

[27] T.M. Cover. The best two independent measurements are not the two best. *IEEE Transactions on Systems, Man, and Cybernetics*, 4:116–117, 1974.

[28] T. M. Cover and J. A. Thomas. *Elements of information theory.* Wiley-Interscience, 1991.

[29] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature Extraction, Foundations and Applications.* Series Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer, 2006.

[30] J. Zhou, D. P. Foster, R. A. Stine, and L. H. Ungar. Streamwise feature selection. *Journal of Machine Learning Research*, 7:1861–1885, 2006.

[31] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:1226–1238, 2005. Software available at `http://research.janelia.org/peng/proj/mRMR/`.

[32] K. Kira and L. A. Rendell. A practical approach to feature selection. In *International Congress of Machine Learning*, pages 249–256, 1992.

[33] L. Xu, P. Yan, and T. Chang. Best first strategy for feature selection. In *9th International Conference on Pattern Recognition*, volume 2, pages 706–708, 1988.

[34] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[35] J. Bala, J. Huang, H. Vafaie, K. Dejong, and H. Wechsler. Hybrid learning using genetic algorithms and decision trees for pattern classification. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 719–724, 1995.

[36] H. Vafaie and K. De Jong. Genetic algorithms as a tool for feature selection in machine learning. In *IEEE International Conference on Tools with Artificial Intelligence*, pages 200–204, 1992.

[37] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using Support Vector Machines. *Machine Learning*, 46(1-3):389–422, 2002.

[38] J. Weston, A. Elisseeff, B. Schölkopf, and M. E. Tipping. Use of the zero-norm with linear models and kernel methods. *Journal of Machine Learning Research*, 3:1439–1461, 2003.

[39] A. Rakotomamonjy. Variable selection using SVM–based criteria. *Journal of Machine Learning Research*, 3:1357–1370, 2003.

[40] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. *Machine Learning*, 47(2-3):201–233, 2002.

[41] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: A stepwise procedure for building and training a neural network. In *Neurocomputing: Algorithms, Architectures and Applications*, volume F68, pages 41–50. 1990.

[42] S. Mika. *Kernel Fisher Discriminants*. PhD thesis, Elektrotechnik und Informatik der Technischen Universität Berlin, 2002.

[43] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory (COLT '92)*, pages 144–152, 1992.

[44] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. 2001.

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

[46] C. Cortes and V. Vapnik. Support Vector Networks. *Machine Learning*, 20:273–297, 1995.

[47] R.T. Rockafellar. *Convex analysis*. Princeton University Press, 1970.

[48] S.P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[49] A. Barbero. *Efficient Optimization Methods for Regularized Learning: Support Vector Machines and Total–Variation Regularization*. PhD thesis, Computer Science Department, Universidad Autónoma de Madrid, 2011.

[50] F. Gorunescu. *Data Mining: Concepts, Models and Techniques*. Intelligent Systems Reference Library. Springer, 2011.

[51] J. Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society, London*, 209:415–446, 1909.

[52] R. Fan, K. Chang, C. Hsieh, X. Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9: 1871–1874, 2008. Software available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

[53] C. Chang and C. Lin. *LIBSVM: a Library for Support Vector Machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[54] C. Hsieh, K. Chang, C. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, volume 307, pages 408–415, 2008.

[55] John C. Platt. Fast training of Support Vector Machines using Sequential Minimal Optimization. In *Advances in Kernel Methods: Support Vector Learning*, pages 185–208, Cambridge, MA, 1998. MIT Press.

[56] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for Svm. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, volume 227, pages 807–814, 2007.

[57] M. Arun Kumar and M. Gopal. A hybrid SVM based decision tree. *Pattern Recognition*, 43:3977–3987, 2010.

[58] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, NY, 1984.

[59] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[60] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal Artificial Intelligence Research (JAIR)*, 2:1–32, 1994.

[61] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3:319–342, 1989.

[62] F. Esposito, D. Malerba, and G. Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analalysis and Machine Intelligence*, 19:476–491, 1997.

[63] I. Rodriguez-Lujan, R. Huerta, C. Elkan, and C. Santa Cruz. Quadratic Programming Feature Selection. *Journal of Machine Learning Research*, 99:1491–1516, 2010.

[64] I. Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[65] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML '03: Proceedings of the 20th international conference on Machine learning*, 2003.

[66] I. Guyon. *Practical Feature Selection: from correlation to causality*, volume 19 of *NATO Science for Peace and Security Series D: Information and Communication Security*, pages 27–43. IOS Press, 2008.

[67] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1999.

[68] M. Dash, K. Choi, P. Scheuermann, and H. Liu. Feature selection for clustering - a filter solution. In *In Proceedings of the Second International Conference on Data Mining*, pages 115–122, 2002.

[69] C. Boutsidis, M.W. Mahoney, and P. Drineas. Unsupervised feature selection for Principal Components Analysis. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–69, 2008.

[70] X. He, D. Cai, and P. Niyogi. Laplacian score for feature selection. In *Proceedings of Nineteenth Annual Conference on Neural Information Processing Systems (NIPS)*, 2005.

[71] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical Programming*, 27(1):1–33, 1983.

[72] R. Sousa, H. Oliveira, and J. S. Cardoso. Feature selection with complexity measure in a quadratic programming setting. In *Proceedings of Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA)*, 2011.

[73] M. A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the International Conference on Machine Learning*, pages 359–366, 2000.

[74] S. Fine, K. Scheinberg, N. Cristianini, J. Shawe-Taylor, and B. Williamson. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2001.

[75] C. Fowlkes, S. Belongie, and J. Malik. Efficient spatiotemporal grouping using the Nyström method. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 231–238, 2001.

[76] S. Kumar, M. Mohri, and A. Talwalkar. Sampling techniques for the Nyström method. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2009.

[77] C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*, pages 682–688, 2001.

[78] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[79] J. Hua, W. D. Tembe, and E. R. Dougherty. Performance of feature-selection methods in the classification of high-dimension data. *Pattern Recognition*, 42(3): 409–424, 2009.

[80] F. Lauer, C. Y. Suen, and G. Bloch. A trainable feature extractor for handwritten digit recognition. *Pattern Recognition*, 40(6):1816–1824, 2007.

[81] Y. Lecun, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351, 2001.

[82] K. Zhang, I. W. Tsang, and J. T. Kwok. Improved Nyström low-rank approximation and error analysis. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 1232–1239, 2008.

[83] S. Zhu, D. Wang, and T. Li. Feature selection for gene expression using model-based entropy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 99(1), 2008.

[84] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. H. Bischof, and D. C. Sorensen. LAPACK: a portable linear algebra library for high-performance computers. In *Proceeding of Supercomputing*, pages 2–11, 1990.

[85] Y. Zhang, D. Wang, and T. Li. LIBGS: A MATLAB software package for gene selection. *International Journal of Data Mining and Bioinformatics (IJDMB)*, 4 (3):348–355, 2010.

[86] J. Neter and W. Wasserman. *Applied Linear Statistical Models*. Richard D. Irwin, INC., 1974.

[87] I. Rodriguez-Lujan, C. Santa Cruz, and R. Huerta. On the equivalence of Kernel Fisher Discriminant Analysis and Kernel Quadratic Programming Feature Selection. *Pattern Recognition Letters*, 32(11):1567–1571, 2011.

[88] A. Brabazon and M. O'Neill. *Biologically Inspired Algorithms for Financial Modelling*. Natural Computing Series. Springer, 2006.

[89] D. Gavrilis, I. G. Tsoulos, and E. Dermatas. Selecting and constructing features using grammatical evolution. *Pattern Recognition Letters*, 29(9):1358–1365, 2008.

[90] B. Cao, D. Shen, J. Sun, Q. Yang, and Z. Chen. Feature selection in a kernel space. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, volume 227, pages 121–128, 2007.

[91] Bernhard Schölkopf, Alex J. Smola, and Klaus-Robert Müller. Kernel Principal Component Analysis. In *International Conference on Artificial Neural Networks (ICANN)*, volume 1327, pages 583–588, 1997.

[92] I. Guyon. Multivariate nonlinear feature selection with kernel multiplicative updates and Gram-Schmidt Relief. In *2003 BISC FLINT-CIBI International Joint Workshop on Soft Computing for Internet and Bioinformatics Frontiers*, 2003.

[93] M. Wu, B. Schölkopf, and G. H. Bakir. Building sparse large margin classifiers. In *ICML '05: Proceedings of the 22th international conference on Machine learning*, volume 119, pages 996–1003, 2005.

[94] M. Tan, L. Wang, and I. W. Tsang. Learning sparse SVM for feature selection on very high dimensional datasets. In *ICML '10: Proceedings of the 27th international conference on Machine learning*, pages 1047–1054, 2010.

[95] J. Moody and C.J. Darken. Fast learning in networks of locally-tuned processing units. *Neural computation*, 1(2):281–294, 1989.

[96] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, and K.R. Mullers. Fisher Discriminant Analysis with kernels. In *Neural Networks for Signal Processing IX, 1999. Proceedings of the 1999 IEEE Signal Processing Society Workshop*, pages 41–48, 1999.

[97] J. H. Friedman. Regularized Discriminant Analysis. *Journal of the American Statistical Association*, 84(405):165–175, 1989.

[98] T. J. Hastie, R. J. Tibshirani, and A. Buja. Flexible Discriminant Analysis by Optimal Scoring. Technical report, AT&T Bell Laboratories, 1993.

[99] J. A. K. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[100] S. Mika, G. Rätsch, and K. Müller. A mathematical programming approach to the Kernel Fisher Algorithm. In *Proceedings of Neural Information Processing Systems 13 (NIPS)*, pages 591–597, 2001.

[101] S. S. Keerthi and S. K. Shevade. SMO algorithm for least-squares SVM formulation. *Neural Computation*, 15(2):487–507, 2003.

[102] D. Cai, X. He, and J. Han. Efficient kernel discriminant analysis via spectral regression. In *Proceedings of the IEEE International Conference on Data Mining 2007 (ICDM'07)*, pages 427–432, 2007.

[103] S. Mika. An improved training algorithm for Kernel Fisher Discriminants. In *Proceedings of the 8th International Workshop on Artificial Intelligence and Statistics (AISTATS 2001)*, pages 98–104, 2001.

[104] T. Xiong, J. Ye, Q. Li, R. Janardan, and V. Cherkassky. Efficient kernel discriminant analysis via QR decomposition. In *Proceedings of Neural Information Processing Systems 17 (NIPS)*, volume 17, pages 1529–1536, 2004.

[105] I. Rodriguez-Lujan, C. Santa Cruz, and R. Huerta. Hierarchical Linear Support Vector Machine. *Pattern Recognition*. Under review.

[106] I. Steinwart. Sparseness of Support Vector Machines—some asymptotically sharp bounds. In *Advances in Neural Information Processing Systems 16 (NIPS)*, volume 16, pages 1069–1076, 2004.

[107] S. S. Keerthi, O. Chapelle, and D. DeCoste. Building Support Vector Machines with reduced classifier complexity. *Journal of Machine Learning Research*, 7:1493–1515, 2006.

[108] O. Chapelle. Training a Support Vector Machine in the primal. *Neural Computation*, 19(5):1155–1178, 2007.

[109] Y. Lee and O. L. Mangasarian. RSVM: Reduced Support Vector Machines, 2001.

[110] K. Lin and C. Lin. A study on Reduced Support Vector Machines. *IEEE Transactions on Neural Networks*, 14(6):1449–1459, 2003.

[111] E. Osuna and F. Girosi. Reducing the run-time complexity of Support Vector Machines, 1998.

[112] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of Support Vector Learning Machines. In *Advances in Neural Information Processing Systems 9*, pages 375–381, 1997.

[113] T. Downs, K. E. Gates, and A. Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, 2002.

[114] H. Kim, S. Pang, H. Je, D. Kim, and S.Y. Bang. Support Vector Machine Ensemble with Bagging. In *Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines*, pages 397–407, 2002.

[115] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002.

[116] 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*, pages 521–528, 2005.

[117] V. Menkovski, I.T. Christou, and S. Efremidis. Oblique decision trees using embedded Support Vector Machines in classifier ensembles. In *7th IEEE International Conference on Cybernetic Intelligent Systems, 2008. CIS 2008.*, pages 1–6, 2008.

[118] C. Guo X. Lin F.Chang and C. Lu. Tree decomposition for large-scale SVM problems. *Journal of Machine Learning Research*, 11:2935–2972, 2010.

[119] X. Zhou, N. Cui, Z. Li, F. Liang, and T.S. Huang. Hierarchical gaussianization for image classification. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1971–1977, 2009.

[120] K. Yu, T. Zhang, and Y. Gong. Nonlinear learning using local coordinate coding. In *Advances in Neural Information Processing Systems 22*, pages 2223–2231. 2009.

[121] L. Ladicky and P. Torr. Locally Linear Support Vector Machines. In *ICML '11: Proceedings of the 28th international conference on Machine learning*, pages 985–992, 2011.

[122] Z. Fu, A. Robles-Kelly, and J. Zhou. Mixing linear SVMs for nonlinear classification. *IEEE Transactions on Neural Networks*, 21(12):1963–1975, 2010.

[123] K. Zapien Arreola, J. Fehr, and H. Burkhardt. Fast Support Vector Machine classification using linear SVMs. In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR '06)*, volume 3, pages 366–369, 2006.

[124] Janis Fehr, Karina Zapien Arreola, and Hans Burkhardt. Fast Support Vector Machine classification of very large datasets. Technical report, University of Freiburg, Department of Computer, 2007.

[125] Jian qing Sun, Gong gui Wang, Qiong Hu, and Shou yi Li. A novel SVM decision tree and its application to face detection. In *Proceedings of the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'07)*, volume 1, pages 385–389, 2007.

[126] K. P. Bennett, N. Cristianini, J. Shawe-Taylor, and D. Wu. Enlarging the margins in perceptron decision trees. *Machine Learning*, 41:295–313, 2000.

[127] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[128] J. Shotton, A.W. Fitzgibbon, M.. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *24th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1297–1304, 2011.

[129] S. Shalev-Shwartz and N. Srebro. SVM optimization: inverse dependence on training set size. In *Proceedings of the 25th International Conference on Machine Learning (ICML'08)*, volume 307, pages 928–935, 2008.

[130] Y. Huang and S. Du. Weighted Support Vector Machine for classification with uneven training class sizes. In *Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, volume 7, pages 4365–4369, 2005.

[131] S. Viaene, R. A. Derrig, and G. Dedene. Cost-sensitive learning and decision making for Massachusetts pip claim fraud data. *International Journal of Intelligent Systems*, 19(12):1197–1215, 2004.

[132] S.B. Park, S. Hwang, and B.T. Zhang. Mining the risk types of human papillomavirus (HPB) by AdaCost. In *International Conference on Database and expert Systems Applications*, pages 403–412, 2003.

[133] J.R Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[134] W. W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the Joint Conference on Artificial Intelligence (IJCAI)*, pages 988–994, 1993.

[135] G. Rätsch. *Benchmark Repository*, 2000. Datasets available at `http://www.fml.tuebingen.mpg.de/Members/raetsch/benchmark`.

[136] A. K. Menon. Large-scale Support Vector Machines: Algorithms and theory. Technical report, University of California, San Diego, 2009.

[137] M. Golea, P. L. Bartlett, W. S. Lee, and L. Mason. Generalization in decision trees and DNF: Does size matter? In *Advances in Neural Information Processing Systems*, pages 259–265, 1997.

[138] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–90, 1993.

[139] Usama M. Fayyad and Keki B. Irani. What should be minimized in a decision tree? In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 749–754, 1990.

[140] A. Ehrenfeucht, D. Haussler, and M. Kearns. Learning decision trees from random examples needed for learning. *Information and Computation*, 82:231–246, 1989.

[141] I. W. Tsang, J. T. Kwok, and P. Cheung. Core Vector Machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[142] P. Carbonetto. Peter carbonetto face detection data. URL `http://www.cs.ubc.ca/~pcarbo/#data`.

[143] Y. Ben-Haim and E. Tom-Tov. A streaming parallel decision tree algorithm. *Journal of Machine Learning Research*, 11:849–872, 2010.

[144] Kazuo Murota. Algorithms in discrete convex analysis. *Math. Programming*, 83: 313–371, 2000.

[145] C. Olsson, A. P. Eriksson, and F. Kahl. Solving large–scale binary quadratic problems: Spectral methods vs. semidefinite programming. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 07)*, pages 1–8, 2007.

[146] P. Drineas and M. W. Mahoney. On the Nyström method for approximating a Gram matrix for improved kernel-based learning. *Journal of Machine Learning Research*, 6:2153–2175, 2005.

[147] A. Deshpande, L. Rademacher, S. Vempala, and G. Wang. Matrix approximation and projective clustering via volume sampling. *Theory of Computing*, 2(1):225–247, 2006.

[148] V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Advances in Neural Information Processing Systems 15*, pages 705–712, 2002.

[149] J. C. Platt. Fast embedding of sparse similarity graphs. In *Proceedings of the 17th Annual Conference on Neural Information Processing Systems 2003*, 2003.

[150] A. Talwalkar, S. Kumar, and H. A. Rowley. Large-scale manifold learning. In *IEEE Conference on Computer Vision and Pattern Recognition, 2008 (CVPR'08)*. IEEE Computer Society, 2008.

[151] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal of Computing*, 36(1):158–183, 2006.

[152] S. Shalev-Shwartz and Y. Singer. Logarithmic regret algorithms for strongly convex repeated games. Technical report, The Hebrew University, 2007.