Università degli Studi di Camerino

School of Advanced Studies

Doctoral Course in

Computer Sciences and Mathematics

XXXIII cycle

Combining Kernel Functions in Supervised Learning Models

Ph.D. Student

Elisa Marcelli

Supervisor

Prof. Renato De Leone

"If a machine is expected to be infallible, it cannot also be intelligent."

Alan Turing

Abstract

The research activity has mainly dealt with supervised Machine Learning algorithms, specifically within the context of kernel methods. A kernel function is a positive definite function mapping data from the original input space into a higher dimensional Hilbert space. Differently from classical linear methods, where problems are solved seeking for a linear function separating points in the input space, kernel methods all have in common the same basic focus: original input data is mapped onto a higher dimensional feature set where new coordinates are not computed, but only the inner product of input points. In this way, kernel methods make possible to deal with non-linearly separable set of data, making use of linear models in the feature space: all the Machine Learning methods using a linear function to determine the best fitting for a set of given data. Instead of employing one single kernel function, Multiple Kernel Learning algorithms tackle the problem of selecting kernel functions by using a combination of preset base kernels. Infinite Kernel Learning further extends such idea by exploiting a combination of possibly infinite base kernels. The research activity core idea is utilize a novel complex combination of kernel functions in already existing or modified supervised Machine Learning frameworks. Specifically, we considered two frameworks: Extreme Learning Machine, having the structure of classical feedforward Neural Networks but being characterized by hidden nodes variables randomly assigned at the beginning of the algorithm; Support Vector Machine, a class of linear algorithms based on the idea of separating data with a hyperplane having as wide a margin as possible. The first proposed model extends the classical Extreme Learning Machine formulation using a combination of possibly infinitely many base kernel, presenting a two-step algorithm. The second result uses a preexisting multi-task kernel function in a novel Support Vector Machine framework. Multi-task learning defines the Machine Learning problem of solving more than one task at the same time, with the main goal of taking into account the existing multi-task relationships. To be able to use the existing multi-task kernel function, we had to construct a new framework based on the classical Support Vector Machine one, taking care of every multi-task correlation factor.

Contents

List of Symbols

Functions

$\mathcal{E}$	The squared lost function
$\mathcal{J}$	The cost function
$\mathcal{L}$	The Lagrangian function
L	The conditional likelihood function
μ	The mean function
$d(x, \tilde{x})$	Distance measure between x and $\tilde{x}$
$f(x, w)$	A function f of x parametrized by w
$f : A \longrightarrow B$	A function f with domain A and range B
L	The loss function
$r(w)$	The regularization parameter
$R_{emp}(f)$	The empirical risk

Indexing

x^i	A vector
x^i_j	Element j of vector x^i with indexing starting at 1
x_i	A scalar

Machine Learning

$\|\cdot\|_{\mathcal{H}}$	The norm defined by $\mathcal{H}$
θ	A threshold value
w	A weight vector

$\mathcal{H}$	A generic Hilbert space
ϵ	The error term
ϵ_i	The error term with respect to the i-th example pair $\{x^i, y_i\}$
$\frac{\partial f}{\partial x}$	Partial derivative of f with respect to x
$\langle \cdot, \cdot \rangle_{\mathcal{H}}$	The inner product defined by $\mathcal{H}$
$\mathcal{D}$	A generic dataset
$\nabla f(x)$	Gradient of f at input x
Φ	A feature matrix of elements $\phi(\cdot, \cdot)$
ϕ	A feature map $\phi : X \longrightarrow \mathcal{H}$
$\{x^i, y_i\}$	The i-th example pair in $\mathcal{D}$ (supervised learning)
E_x	The evaluation functional at point x
K	The Kernel matrix
$k(\cdot, \cdot)$	A positive definite kernel function
K_{ij}	Element ij of matrix K
m	Dimension of a data point x^i
n	Dimension of X
Q	The sample covariance matrix
S_B	Between the class covariance matrix
S_w	Within classes covariance matrix
X	The set of training examples
x^i	The i-th example in X
Y	The set of training labels (supervised learning)
y_i	The i-th example in Y (supervised learning)

Other Symbols

$\|u\|$	The standard norm of u
$\mathcal{X}$	A set
c	A scalar

I	The identity matrix
u	A vector
$u^{\mathrm{T}}v$	The standard dot product between u and v
v	A vector

Sets

$\mathbb{R}$	The set of real numbers
$\{-1, +1\}$	The set containing -1 and $+1$
$\{0, +1\}$	The set containing 0 and $+1$
$\{1, 2, \ldots, n\}$	The set of all integers between 0 and n

Introduction

The term Machine Learning was coined for the first time in 1959 by Arthur Samuel, who defined it as a "*Field of study that gives computers the ability to learn without being explicitly programmed*", [Samuel, 1959]. A more recent and detailed definition was given by Tom Mitchell [Mitchell, 1997] stating that, "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*"

Broadly speaking, Machine Learning is a branch of Artificial Intelligence firstly introduced in the last decades of the twentieth century, having as a core idea the goal of ensuring a machine to learn based on a given set of data.

Machine Learning gained a huge success in the last decades, becoming one of the most popular and studied branches of artificial intelligence. The process of learning is defined by the ability of modifying any behavior or performance based on previous experience and novel outside incentive. Hence, as a direct consequence the term Machine Learning defines all kinds of computer-implemented algorithms that allow machines to change their conducts in order to adapt to given stimulus and obtain better results. Based on the considered received stimula, i.e., a set of training data, and on the problem one aims to tackle, i.e., the decisions that need to be taken, different Machine Learning fields occur corresponding to the appropriate mathematical models. Nevertheless, every Machine Learning model is characterized by one main goal: based on the given dataset, obtain a general working learning rule, namely a prediction function, to be used for future predictions. Specifically the principal goal of every Machine Learning algorithm is, once the considered model has been trained with respect to a given set of data, i.e., the training set, to generate appropriate and highly accurate outputs when dealing with new unknown samples.

Machine Learning is by definition a multi-disciplinary field, combining together many research disciplines in order to generate operating models to be successfully used in real word problems. Therefore, it requires the combination of more than one academic researchers bringing together their specific expertise to achieve a common goal [Mancini et al., 2020].

Linear learning methods are among the most used Machine Learning techniques

due to their simple yet effective operational schemes. Anyway, nonlinear data not only exist in practice but they represent the majority of real life data and are essential to pattern realistic scenarios as well as to obtain successful results. Kernel methods are often exploited to overcome this issue.

Kernel methods are specific Machine Learning algorithms characterized by the use of particular functions, called kernel functions, as part of their training phase. Specifically, the term positive definite kernel function, or simply kernel function, defines a specific group of positive definite symmetric functions mapping inputs from the original space to a usually higher dimensional feature space, with no actual need of computing such new coordinates in the feature space but just requiring the calculation of the inner products of every pair of data points in the feature space. Such technique is often known under the name of kernel trick, since it allows to map non-linearly separable data from the original input set to a higher dimensional feature set, where it can be modelled in a linear way, without the actual computation of the novel coordinates.

From a specific point of view, kernel functions are closely related to the notion of Reproducing Kernel Hilbert Space: a function of space where the actual learning and estimation processes are made. In this way, plenty of algorithms originally developed for linear learning may be successfully and efficiently applied to non-linearly separable set of data. Specifically, modelling nonlinear data using kernel methods has several benefits. First, it allows to work in linear spaces, leading to computationally cost-effective learning algorithms. Moreover, kernel methods allow the use of a huge number of kernel functions, making a wide choice of feature space based on the corresponding considered data. Furthermore, no actual computation of the new coordinates is necessary, relaying on the inner products of the data in the feature space only. Finally, no condition of the original input space is required, which can be Euclidean as well as non-Euclidean.

The use of a single kernel function may in some cases limit the sake of the considered learning algorithm. In fact, the success of the treated learning method may depend on the choice of a specific kernel function as well as on the selection of the corresponding kernel parameters. Hence, it may happen to pick a specific base kernel function not really fit for the problem one aims to treat. A solution to this matter may be to employ more than one kernel function rather than considering a single preset kernel. Specifically, the term Multiple Kernel Learning refers to the Machine Learning technique of using a finite set of base kernel functions, looking for the best possible combination coefficients as well as the leading kernel parameters. Similarly, Infinite Kernel Learning defines an extension of Multiple Kernel Learning based on the core idea of operating with infinitely many base kernel functions, leaving a wide selection of possibilities.

Other that giving an in-depth description of the related theory, in this thesis work we try to identify novel connections between the concept of using more than one kernel function and existing supervised learning methods.

The rest of the thesis is organized as follows. Initially, in Chapter 1 we summarize the state of the art on Machine Learning, indicating the historical cornerstones that strongly affected its development and applications, both from a theoretical point of view as well as from a more practical perspective. Moreover, we describe the tree main branches of Machine Learning, trying to outline its basic theory and the corresponding learning rules. After that, we focus on Kernel Methods, starting from a formal definition of Hilbert space, going through the notion of positive definite kernel function as well as some popular examples, up to defining the concept of Reproducing Kernel Hilbert Space. Then, in Chapter 2 we report the existing theory on Linear Methods, describing the most known and employed algorithms with respect to supervised and unsupervised learning. In Chapter 3 we analyse the state of the art research work on kernel based models, still focusing on the general theory but also outlying some popular methods. Moreover, we describe Multiple Kernel Learning and Infinite Kernel Learning, and in both cases we report existing approaches based on Support Vector Machine framework. In Chapter 4 we describe two of the most significant achieved results. Specifically, we summarize two published papers dealing with the investigation of the use of more than one base kernel function into existing supervised learning models adapted for our problem. Finally, we draw conclusions.

Chapter 1

State of the Art

This chapter provides a general overview of modern Machine Learning (ML) techniques: describing its development into time, taking into account several pivotal moments with respect to both theoretical background and practical point of view; looking at the main existing Machine Learning methods, describing its fundamental aspects regarding supervised, unsupervised and reinforcement learning methods; explaining the notional framework at the basis of kernel methods, analysing fundamental theoretical matters together with practical knowledge concerning kernel function definition.

Machine Learning research field has the main goal of constructing computer programs with the capacity of self improving with respect to experience and time factors. Especially in the last decades, Machine Learning has deeply broaden its theory background as well as practical applications, making the related area of research continuously updated and characterized by novel factors.

Ensuring that a machine learns is not an easy task, making a machine learn as good as a human is still to be done. Anyway, Machine Learning main purpose is to find theoretically functioning and computationally feasible algorithms leading to always improving results, approximating as good as possible the related human ones.

First, the most important stages of Machine Learning are mapped out, using a sort of timeline as a bookmark. Starting from Bayes, passing through Legendre, Gauss and Laplace, up to reaching the more contemporary theories of Markov and Kolmogorov, the critical process steps at the basis of the theory of every developed Machine Learning algorithm is proposed. In addiction, the best known cases of examples of implementation of Machine Learning algorithms are presented, from the classical Support Vector Machine, to the development of the first Neural Network structure. On a more practical point of view, Machine Learning most famous goals are described: from the first self-learning game of checkers in 1952, by way of TD-Gammon program in 1992, to the most recent AlphaGo algorithm, able to beat one of the strongest Go player alive.

Moreover, this chapter provides an in detail description of the three existing Machine Learning categories. First, supervised learning algorithm is considered, taking into account general rules of both classification and regression models. After that, unsupervised learning algorithm is treated, tackling the problem of cluster analysis and dimensionality reduction as most famous and popular examples. Finally, the classical reinforcement learning framework is schematized.

After that, a theoretical description of Kernel Methods is proposed. After describing the notion of Hilbert space, a definition of kernel function is given and several examples of well-known kernel maps are presented, together with their basic properties. Finally, the classical formulation of Reproducing Kernel Hilbert Space is given, taking into account significant existing results at the basis of the theory of Kernel Methods.

1.1 Machine Learning: a brief history

For a complete view of the main concepts behind Machine Learning, we are going to describe the development of the foundations at the core of Machine Learning research field using the passage of time as a marker. Specifically, we are going to present the events that in our opinion have deeply shaken the study of methods or algorithms for the automatic creation of models from data.

The precise outset may be placed in 1763 with the publication of Thomas Bayes' "*An essay towards solving a problem in the doctrine of chances*" [Bayes, 1763]. Center of reference of capital importance of the essay is the derivation of the posterior distribution of a problem, also known as Bayes Theorem, where the probability of a specific event, given the known occurrence of another one, is described through the use of a simple one line formula. Specifically, Bayes Theorem provides an alternative prospective to classical approaches in computing parameters: instead of choosing a fixed set of values, Bayesian methods rely on the concept of randomness. Parameters are initially treated as random variables, represented as prior distribution, and eventually learned as a posterior distribution with respect to the specific set of observations.

In 1805 Adrien-Marie Legendre published the least squares method [Legendre, 1805], to approximate a regression function with respect to a given set of observations. This method is based on the minimization of the sum of the squared deviations subject to the given model function, in order to find the parameters which better describe the original data.

However, in 1809 Carl Friedrich Gauss stated to be aware of the least square method since 1895 [Gauss, 1809], starting one of the most famous mathematical disputes, and proved it using the idea of the maximum likelihood. Moreover, it should be emphasised that in his famous "*Theoria Motus*" Gauss described for the first time the idea of the normal (i.e., Gaussian) distribution of random variables in the context of the motion of celestial bodies.

As a direct consequence, Pierre-Simon Laplace in 1810 first proved the Central Limit Theorem (CLT) [Laplace, 1810], outlining in a new way a connection between the binomial distribution and the novel normal distribution, characterised by only two parameters: the mean and the standard deviation. It is important to note that, the Central Limit Theorem plays a key role in Machine Learning models by ensuring that the sample mean, i.e., the average of a specific dataset, will act as a normal distribution.

Expanding the results obtained almost a century before by Gauss, in 1860 James Clerk Maxwell observed that the normal distribution is at the basis of a wide variety of natural phenomena: starting from the movement of gas molecules, Maxwell proved that the velocity components of particles clashing into each-other follow a normal distribution [Maxwell, 1860].

An interesting concept that will be widely used in the following years was proposed by Francis Galton in 1889 [Galton, 1889]. He discovered the concept of correlation factor, dealing with the statistical relationship between variables. Galton dealt with the problem in a purely anthropological context, focusing on biological aspects of related people, stating that: "*Two variable organs are said to be co-related when the variation of the one is accompanied on the average by more or less variation of the other, and in the same direction. Thus the length of the arm is said to be correlated with that of the leg, because a person with a long arm has usually a long leg, and conversely. If the co-relation be close, then a person with a very long arm would usually have a very long leg; if it be moderately close, then the length of his leg would usually be only long, not very long; and if there were no co-relation at all then the length of his leg would on the average be mediocre. It is easy to see that co-relation must be the consequence of the variations of the two organs being partly due to common causes. If they were wholly due to common causes, the co-relation would be perfect, as is approximately the case with the symmetrically disposed parts of the body. If they were in no respect due to common causes, the co-relation would be nil. Between these two extremes are an endless number of intermediate cases, and it will be shown how the closeness of co-relation in any particular case admits of being expressed by a simple number.*"

In 1913 the Russian mathematician Andrey Markov first presented the Markov chain model [Markov, 1913]. Markov processes will be broadly used in later years for solving tricky Machine Learning problems, specifically using hidden Markov models, where the system takes on a Markov chain shape.

A novel and contemporary vision on the notion of probability was given by Andrej Nikolaevič Kolmogorov in 1933 [Kolmogorov, 1933], with the publication of the so called "*Kolmogorov axioms*", through which he first established fixed basic concepts which will be at the base of probability theory up to the present day.

On a more practical point of view, it is important to mention the 1935 book by Ronald Fisher [Fisher, 1936] as the first real example of a series of experiments to test

given hypothesis.

The year 1948 must be considered as the starting year of the Information Theory due to the publication of the paper "*A Mathematical Theory of Communication*" by Claude Shannon [Shannon, 1948]. Shannon is considered the father of digital age and his work has given way to modern information as we know it today, by first defining, among other things, the concept of entropy.

A couple of years later in 1950 another important event shocked the foundations of Artificial Intelligence (AI): the English mathematician Alan Turing suggested the use of a test to determine whether a machine may establish an intelligent behavior. Such test known as the Turing test appeared on the magazine "*Mind*" [Turing, 1950] and had the main goal to try to determine if an Artificial Intelligence is as good as it is supposed to be and, most importantly, whether or not it is distinguishable from a human being.

A few months later in 1951, the first Artificial Neural Network (ANN), known as Stochastic neural analog reinforcement calculator (SNARC), was built by Marvin Lee Minsky. The proposed network was made of 40 neurons connected with each other and with the characteristic of being equipped with both short-term and long-term memory.

While working for the International Business Machines Corporation (IBM), the American computer pioneer Arthur Samuel was the first one to ever design a self-learning algorithm. Specifically, in 1952 Samuel developed a checker program for the IBM 701 that allowed the user to acquire skills in the game of checkers, playing with a commercial computer that trains itself. Therefore, this may be considered a first step on the idea of reinforcement learning.

Always at IBM, in 1957 Frank Rosenblatt simulated for the first time the behavior of a Perceptron [Rosenblatt, 1957], a specific mechanism equipped with learning skills and based on a biological physical structure. The algorithm proposed a novel approach: at each iteration an input vector is presented to the Perceptron, which calculates the output and compares it with the desired result, accordingly updating the structure weights. The Perceptron algorithm was originally designed for binary linear classification problems, i.e., problems where data points are linearly separable and the output can take only two specific values.

From a theoretical point of view, in 1960 Bernard Widrow together with his PhD student Marcian Hoff proposed the Least Mean Squares (LMS) algorithm, also known as Widrow-Hoff learning or delta rule [Widrow and Hoff, 1960]. It is a gradient descent algorithm which, at each iteration, aims to minimize the mean square error between the given and the computed data. The Widrow-Hoff rule is at the basis of the renowned backpropagation algorithm.

A shock to the foundations of Machine Learning was given by Alexey Ivakhnenko and his research group in 1965, with the creation of the first Deep Learning network [Ivakhnenko and Lapa, 1965]. The proposed networks were trained with the use of the

so called Group Method of Data Handling (GMDH) algorithms, based on a completely automatic optimization of the structure and parameters.

The book "*Perceptrons: an introduction to computational geometry*" by Marvin Minsky and Seymour Papert written in 1969 [Minsky and Papert, 1969] may be both considered a summary and a turning point on the field of Machine Learning. The book, in fact, besides describing the robustness and qualities of the Perceptron structure introduced by Rosenblatt, discusses its limitations as well, such as pointing out the difficulty of the Perceptron to solve simple logic predicates, e.g., XOR function.

The publication of this book coincides and is partly the cause of the crisis of the Artificial Intelligence research field, also known as "AI winter". Back then, Artificial Intelligence was thought to be able to find the best fitting solution to any kind of natural problem, but Minsky and Papert showed how the Perceptron algorithm was not able to succeed when dealing with basic examples of non-linear separable problems. All this provoked a sense of despair in the scientists, who started to question and doubt Machine Learning research field.

The release in 1977 of the Expectation-Maximization (EM) algorithm by Arthur Dempster, Nan Laird, and Donald Rubin [Dempster et al., 1977] marks a very important moment in the history of Machine Learning, somehow closing the gap created in the previous decade. The Expectation-Maximization algorithm is an iterative method usually used for computing maximum likelihood estimates in the event of the absence of some data. It represents a novel approach still in use, particularly in Bayes models.

In 1980 the Japanese computer scientist Kunihiko Fukushima proposed the neocognitron: a hierarchical, multi layer, convolutional neural network [Fukushima, 1980] primarly used for classification of handwritten numbers.

One of the most important discovery in the area of Neural Networks was published by John Hopfield in 1982 and is now named after him: the Hopfield network, the first example of recurrent Neural Network [Hopfield, 1982]. It is a single layer, fully connected Neural Network for unsupervised learning, i.e., inputs are provided to the network which learns only on the basis of these, defined by the presence of loops that allow information to extend over time.

The first probabilistic graphical model was presented by Judea Pearl in 1985 [Pearl, 1985] with the development of the Bayesian networks. In a Bayesian network the variables and their conditional dependencies are represented through the use of a directed acyclic graph (DAG), i.e., a graph where vertices are connected with each other through the use of edges, defined by specific directions and characterized by the absence of directed cycles.

Always developed while working for IBM upon the lines of Samuel's checkers game, in 1992 Gerald Tesauro developed the TD-Gammon program [Tesauro, 1992], a learning program for the game of backgammon. It is based on temporal difference learning, i.e., a reinforcement learning algorithm, where the learning phase is built on the disparity

of subsequent predictions with respect to the time variable.

In 1995 Corinna Cortes and Vladimir N. Vapnik expanded the research work done by Vapnik in the previous years and proposed the soft-margin version of the well-known Support Vector Machine (SVM) [Cortes and Vladimir Vapnik, 1995]. Specifically, the paper proposes one of the most robust supervised Machine Learning algorithm for classification problems based on a specific notion: finding the maximum-margin hyperplanes dividing the two classes of data. As widely described later, with the use of kernel functions the Support Vector Machine method may be also used for non-linear classification dataset.

In the same year, Tin Kam Ho proposed the Random Forest algorithm [Ho, 1995], a supervised Machine Learning algorithm for both regression and classification problems, based on the construction of a series of decision trees. It is an ensemble method, i.e., a technique that combines predictions from multiple Machine Learning algorithms to make more accurate predictions than any single model, which combines many decision trees into one model.

The Long Short-Term Memory (LSTM) architecture was first proposed by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [Hochreiter and Schmidhuber, 1997]. It represents a peculiar recurrent Artificial Neural Network, endowed with the capacity of learning long-term associations through the use of a novel Neural Network structure that repeats itself.

Again in 1997 an important event shocked the community: for the first time a computer, specifically an IBM computer known as "Deep Blue", beat the world chess champion of that time after competing in a six game match [IBM, 1997]. The chess playing machine was capable of go beyond the limits imagined until then and solve continuous intricate calculations necessary to beat the human world champion.

After the Deep Blue match, suddenly Artificial Intelligence regained fame and became an in-style topic again. Moreover, this historical moment also coincides with the birth of a new awareness in scientists: bigger and bigger sets of data were needed to power the algorithms, in order to correctly learn in a more precise manner how the essential process of reality occurs and to be able to reproduce them in a digital way.

Based on this line of thought, in 1999 Professor Yann LeCun together with Corinna Cortes from Google and Christopher J.C. Burges from Microsoft made avaiable to the public the world famous "MNIST" dataset [LeCun et al., 1999]: a huge set of handwritten digits containing 60 000 examples in the training set and 10 000 test set images. Even after 20 years it still represents a commonly used dataset for many image processing problems.

Still in the area of big dataset, in 2009 the famous company operating in the internet distribution of films, television series and other paid entertainment contents, Netflix, started a public race to obtain the best algorithm for predicting users' appreciation to video contents based on their past views. To do that, all the Netflix data were made

available for anyone who wanted to compete [Netflix, 2009].

Also in 2009 the largest image dataset to that date was released for public use: the "ImageNet" [Deng et al., 2009]. Even today, it represents one of the largest dataset of images containing a total of 14 million images.

In terms of development and editing of algorithms, in January 2010 the first public version of the open source software Machine Learning library "scikit-learn" was published, making a braking point regarding the free use of the most popular existing Machine Learning algorithms [Pedregosa et al., 2010]. This event marks a turning point, because since then almost all the Machine Learning libraries were made open and free to every scientist willing to use them.

In the same year, the online platform "Kaggle" was released [Kaggle, 2010]. Kaggle is an online community for competitions of predictive and analytical models where Machine Learning scholars and data scientists may exchange opinions and materials as well as develop new skills.

The Watson computer was launched on 2011 [IBM, 2011], an artificial intelligence system capable of answering queries expressed in a natural language fine-tuned by IBM. The virtual assistant Watson uses data available from external sources, as official guidelines or reliable website, and was also recently used to answer citizen questions on the recent Covid-19 pandemic.

In 2012, Google scientist Jeff Dean and Professor Andren Ng created a Neural Network made of 16 000 computers [Times, 2012]. It was originally built to act in a human way but ended up training itself to being able to recognize cats on YouTube videos.

Developed in 2014 and launched for public in 2015, the "DeepFace" is a robust algorithm for facial recognition [Taigman et al., 2014]. It is a Facebook system able to recognise human faces with an accuracy of 97.35%.

On a strictly theoretical point of view, it is impossible not to mention the advent of the Generative Adversarial Networks (GANs) created by the American computer scientist Ian Goodfellow and his research group in 2014 [Goodfellow et al., 2014]. A Generative Adversarial Network is a Machine Learning architecture formed by two neural networks set against each-other with the goal of creating new digital images that can be mistaken for real by the human eye.

Finally, a recent news to point out happened in 2016 and concerns once again a computer beating a human mind [CNN, 2016]. Specifically, "AlphaGo", i.e., a software for playing the famous Chinese board game Go, defeated one of the strongest players in a five game match.

1.2 Process of Learning

The process of learning is defined as "the relatively permanent change in a person's knowledge or behavior due to experience". Hence, the act of learning is closely related to the basic idea of personal experience: new skills and knowledge are acquired through study or by the use of external instructions, triggering a permanent modification of a person's beliefs and performance. Discovering how the process of learning happens has generally involved both animals and humans, leading to the creation of specific research fields.

Ever since the first computers were created, we asked ourselves whether it is possible or not to make them independently learn simply with the use of experience. Machine Learning studies possible approaches and algorithms relating the problem of learning in machines, powered by a simple in theory, yet tricky in practice explanatory statement: make computers solve everyday problems with no solution given, namely without being explicitly programmed to do so.

In this context, in broad terms we talk about learning when a machine changes its layout in accordance with new received information or different inputs, such that its future execution will be improved by such structural modifications. As described in the previous section, Machine Learning was firstly associated with two kind of problems. On the one hand concerning computer games, namely PC games where the user could initially play against the computer itself and later on broaden to multi-players approaches, and on the other tackling Artificial Intelligence problems. The latter notion studies the issue of the development of "intelligent agents", i.e., machine devices that, based on sensory signals, could modify their actions and goals. Over time, these two fields, i.e., Artificial Intelligence and Machine Learning, started to distance from each other, with the result of Machine Learning focusing more and more on solving practical problems over Artificial Intelligence ones. As a direct consequence, today Machine Learning is considered to be a branch of Artificial Intelligence, in other words, ML $\subset$ AI.

As described later on this thesis, Machine Learning approaches often look at and try to replicate human and animal learning methods, linking these three research fields and therefore stressing out their connections. Machine Learning usually concerns tasks such as predictions, medical diagnosis, robots control, timetables planning and physical recognition and, based on the given issue, different methods are utilized.

Since until now machines are still not able to learn as fast, good and precisely as humans, one may ask why all of this interest over focusing on the use of Machine Learning. A variety of motivations exist underlying the importance of Machine Learning. First of all, many tackled approaches arising from human or animal environments may solve and explain purely biological questions and, therefore, help solving problems in the original research fields, i.e., finding out how humans and animals learn in reality.

Secondly, external frameworks change quickly and human brains are not always capable to adapt to these changes. On the contrary, machines can easily adapt to environmental changes with no use of constant redesign. In addition, the number of data available is increasing more and more over the last decade, making the human brain poorly feasible when working with them and observing the fundamental aspects and connections among them. Furthermore, such problems arise in different fields and change rapidly as time passes. Human learning is not always prepared to immediately solve such new and complex problems, while Machines Learning approaches may be capable of monitoring such changes and quickly adapt based on them.

Machine Learning approaches are usually defined by a typical structure which describes its fundamental core. When raw data is to be analysed, namely data which enter the computer system directly after being collected, which can be either through a tool or through a direct experimental observation of the analyst, the following process takes place. Data is analysed and investigated in order to extract fundamental information with the aim of extrapolating characteristics which will provide information to be used in the learning phase. Such sub-process is known as Feature Extraction. Hence, once data is analysed and features are extracted, we obtain a dataset ready to be used in the actual learning phase. Let

$$\{x^i\}_{i=1}^n$$

be a collection of data obtained after the feature extraction phase, with

$$x^i = (x_1^i, x_2^i, \cdots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m,$$

$\forall i = 1, \cdots, n$. Namely x^i is an m dimensional feature vector. As a result, the dataset used in the learning phase is an $n \times m$ matrix with elements

$$\begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_m^1 \\ x_1^2 & x_2^2 & \cdots & x_m^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^n & x_2^n & \cdots & x_m^n \end{pmatrix} = \begin{pmatrix} {x^1}^{\mathrm{T}} \\ {x^2}^{\mathrm{T}} \\ \vdots \\ {x^n}^{\mathrm{T}} \end{pmatrix},$$

with x^i the m-dimensional feature vector corresponding to the i-th component. Once useful features are obtained from the original input data, the learning phase may begin. Features are combined and analysed so that a learning function is discovered and deployed to compute output values. When the corresponding output for each element in the dataset

$$\{y_i\}_{i=1}^n$$

is given, we talk about supervised learning; otherwise, namely when the output vector is not known in advance, one speaks of unsupervised learning. Such distinction and the corresponding learning strategies are described in more detail below. Figure 1.1 shows a basic diagram representing the described Machine Learning structure.

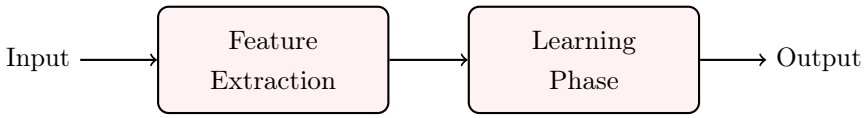


Figure 1.1: A basic Machine Learning structure. Raw data is given as input to the Feature Extraction phase, from which features are extracted to provide useful information. After that, the Learning Phase begins, leading to a computed output vector.

1.2.1 Types of learning

Broadly speaking, from a practical point of view, Machine Learning methods are usually designed to engineer a vector-valued function in order to learn specific outputs based on the data they input. Of course, the data they output ought to be connected to the problem they aim to solve, and Machine Learning algorithms try to determine the best fitting functions with respect to the considered inputs and tasks.

Based on the available data and the addressed problem to be solved, Machine Learning methods are commonly split into three different categories: supervised learning, unsupervised learning and reinforcement learning.

1.2.1.1 Supervised learning

When a dataset of labeled inputs is available, we talk about supervised learning approach. Specifically, the term labeled inputs means that example inputs together with their desired outputs are at the user's disposal, whom has the ability to use such supplementary information to build the model. Namely, in supervised learning the goal is to fit these points with a function which represents the given data in the best possible way.

In detail, let us consider a set

$$\{(x^1, y_1), (x^2, y_2), \dots, (x^n, y_n)\}$$

of n labeled data, where input value $x^i \in \mathbb{R}^m$ is paired with its corresponding label $y_i \in \mathbb{R}$, $\forall i = 1, \cdots, n$. A supervised learning algorithm looks for a function

$$f : X \longrightarrow Y,$$

where $X \subseteq \mathbb{R}^m$ is the input set and $Y \subseteq \mathbb{R}$ is the output space, such that f better represents the given labels. Specifically, using a scoring function

$$g : X \times Y \longrightarrow \mathbb{R},$$

a supervised learning algorithm seeks for the function f such that

$$f(x) = \arg\max_y g(x, y),$$

namely, such as the vector value y gives the best score.

In practice, there exist two approaches for finding the best f: the empirical risk minimization and the structural risk minimization. Both methods use the key concept of loss function. A loss function is a function

$$L : Y \times Y \longrightarrow \mathbb{R}^+,$$

analyzing how well a function f fits the given labeled data $\{(x^i, y_i)\}_{i=1}^n$. Given a generic example (x^j, y_j) and let $f(x^j) = \hat{y}_j$, the quantity $L(y_j, \hat{y}_j)$ measures how well the chosen function f works with respect to the specific dataset. The empirical risk R_{emp} [Vladimir Vapnik, 2013] is defined as the average error in the training set, namely

$$R_{emp}(f) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x^i)).$$

The empirical risk minimization strategy looks for the function f minimizing the empirical risk

$$\min_f \quad R_{emp}(f).$$

However, the use of the empirical risk model often turns out to have problems of overfitting, namely to obtain a model f too complex, which almost exactly fits the training data but that fails in modelling new inputs.

On the contrary, structural risk minimization approach has proven to be particularly suitable for a wide number of practical problems [Vapnik and A. Chervonenkis, 1979]. Structural risk minimization method may be considered an extension of the empirical risk based approach. However, in contrast to the empirical risk technique, which looks for the best predictor f over a preset family of functions, structural risk minimization method aims to balance the problem of finding a fitting predictor together with the model complexity. Specifically, let w define the model parameters. Then, structural risk model aims to solve the following minimization problem

$$\min_w \quad \frac{1}{n} \sum_{i=1}^{n} L(y_i, f_w(x^i)) + \lambda r(w),$$

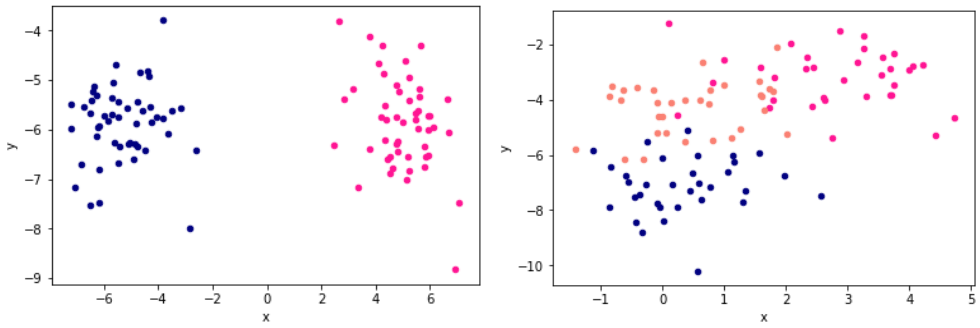
with $\lambda \geq 0$ and $r(w)$ regularization parameter.

So, in summary, classical empirical risk approach gives importance only on finding a predictor f fitting the given training data. In order to prevent overfitting problems, empirical risk minimization approach tries to find the best predictor f_w, still aiming to restrict its complexity.

In addiction, based on the type of labels, i.e., outputs, different supervised Machine Learning approaches must be employed.

Classification

In the event of discrete valued outputs, namely a finite number of numerical values, or



(a) Binary classification.

(b) Multi-class classification.

Figure 1.2: Two straightforward classification problems. Figure 1.2a shows a dataset featured with two labels, represented in blue and deep pink colors. Figure 1.2b describes data featured with three possible categories, i.e., blue, deep pink and orange.

categorical labels, i.e., outputs which may only lie in a set of categories specifying some qualitative properties, we talk about classification problems. Specifically, when the output can only take a fixed number of possible choices, no matter whether they are numerical or categorical, this requires the use of classification algorithms to correctly separate the given data into such specific categories. When only two possible choices are possible, such Machine Learning problem is referred as binary classification. On the contrary, in the case of a selection of several possibilities, we talk about multi-class classification problems. Figure 1.2a shows an example of binary classification problem, with input data taking only two possible label values. Figure 1.2b represents a multi-class classification problem, specifically with inputs belonging to three possible groups instead.

Regression

When dealing with dataset

$$\mathcal{D} = \{x^i, y_i\}_{i=1}^n$$

characterised by real valued dependent variables y_i, $\forall i = 1, \cdots, n$, we talk of regression analysis. Specifically, regression methods main goal is finding a connection between independent variables x^i and dependent variables y_i, such that cause and effect relationships in the given dataset may be estimated and the best prediction can be obtained. In particular, regression analysis tackles the problem by using a prediction function f depending on variables x^i and on a regression coefficients w such that

$$y_i = f(x^i, w) + \epsilon_i,$$

where ϵ_i is the error term specifying the difference between the actual output y_i and the computed value $f(x^i, w)$. Of course, the function f is chosen to better fit the given

data, describing the existing bond between dependent and independent variables.

When function f is selected as a linear function, we speak of linear regression; specifically, f is a linear combination of inputs x^i, $i = 1, \cdots, n$ and parameter w

$$f(x^i) = \sum_{p=0}^{m} w_p x_p^i$$
$$\Rightarrow \; y_i = \sum_{p=0}^{m} w_p x_p^i + \epsilon_i, \tag{1.1}$$

with $w = (w_0, w_1, \cdots, w_m)^{\mathrm{T}}$ and $x_0^i = 1$, for every $i = 1, \ldots, m$.

Generally, when the prediction function f is not modelled as a linear function, we have a non-linear regression. In such circumstances, for the sake of simplicity, let us consider scalar input values $x_i \in \mathbb{R}$, $\forall i = 1, \ldots, n$, that is $m = 1$. In such a case, the function f is a combination of non linear parameters and factors depending on the independent variables x_i. The easiest example of non-linear regression analysis is given by the polynomial case. The function f is shaped as the q-th degree polynomial function with respect to variables x_i and parameter w. Specifically, starting from the linear definition given by Equation (1.1), the general polynomial regression method is defined by

$$y_i = w_0 + w_1 x_i + w_2 (x_i)^2 + w_3 (x_i)^3 + \cdots + w_q (x_i)^q + \epsilon_i$$

where $w_i = (w_0, w_1, \cdots, w_q)$ is the regression coefficient vector and ϵ_i is the error term with respect to input x_i. Overall, considering the set of independent variables X and the set of dependent variables Y, non-linear regression tackles the problem of assessing the relationship between inputs X and labels Y, looking for the function f such that

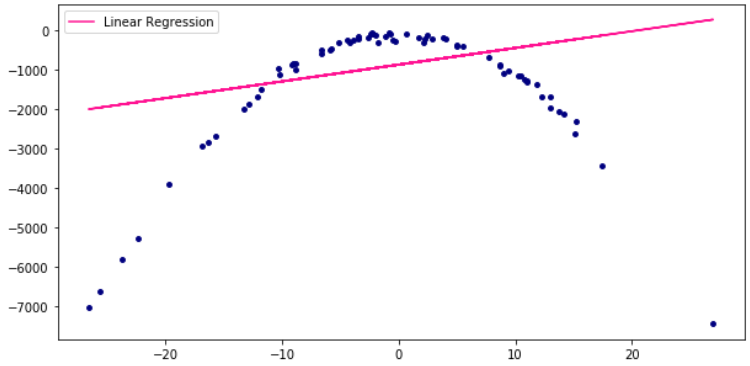
$$Y \sim f(X, w),$$

hence, looking for a function f that comes closest to the given outputs vector Y. To do so, several methods are used, all sharing one thing in common: trying to minimize the error vector $\epsilon = (\epsilon_1, \epsilon_2, \cdots, \epsilon_n)$, with $\epsilon_i = y_i - f(x_i)$ the residual with respect to the i-th input x_i.

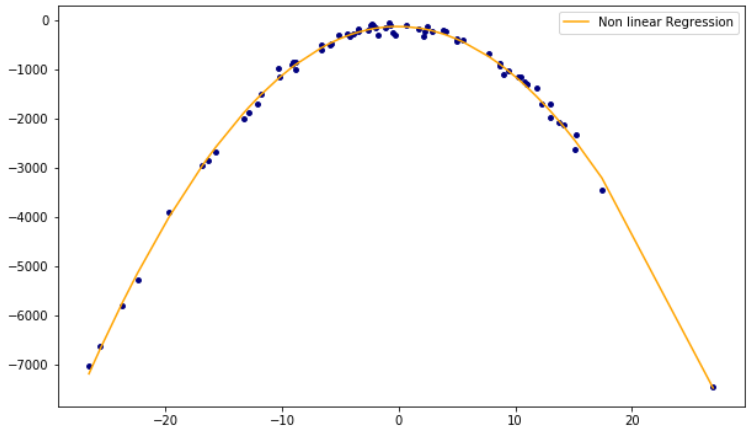
Figure 1.3 shows both linear and non linear regression functions applied to the same dataset.

1.2.1.2 Unsupervised learning

When we deal with a dataset devoid of labels, we are addressing an unsupervised learning problem. As for the supervised learning methods, we are still dealing with a training set, but with the essential difference given from the absence of labels. An unsupervised learning model operates with inputs that have not been already categorized and, therefore, it has to find some intrinsic information to extract data



(a) Linear regression.



(b) Non linear regression.

Figure 1.3: Given a generic dataset, data is fitted using a linear prediction function in Figure 1.3a and a non-linear prediction function in Figure 1.3b. In this particular case, one can easily see that the non-linear function better represents the given data compared to the linear prediction function because of the intrinsic properties of the given dataset.

connections. From learning inner data properties and analyzing such properties in each data, unsupervised learning algorithms have two main purposes: learn to divide the given dataset into specific groups (cluster analysis) based on the presence or absence of such properties; determine which feature better represent the given data (dimensionality reduction problem). Despite being characterized by lower data information with respect to supervised learning algorithms, unsupervised learning methods are investigated and commonly operated by the scientific community. The reason for this interest can easily be found simply looking at the available dataset: the vast majority of the accessible data are unlabeled, making the development and use of unsupervised learning algorithm extremely important to better understand everyday occurrences.

Clustering

Cluster analysis term describes an unsupervised learning task used for finding similarities or resemblances between unlabeled data. Dealing with data where no output or relationship between inputs is given, clustering method automatically organizes data into groups based on the likeness of data points. To do so, clustering needs some specific and predefined notions. In particular, it is necessary to specify the following concepts:

- a *proximity measure*, defining how similar or different two specific data points are with each other. Note that, for this purpose are commonly used two classical distance measures: the Euclidean distance, defined as

$$d(x, \tilde{x}) = \sqrt{\sum_{i=1}^{m} (x_i - \tilde{x}_i)^2}$$

  or the Manhattan distance, given by

$$d(x, \tilde{x}) = \sum_{i=1}^{m} |x_i - \tilde{x}_i|$$

  for $x = (x_1, x_2, \cdots, x_m)$ and $\tilde{x} = (\tilde{x}_1, \tilde{x}_2, \cdots, \tilde{x}_m)$ two generic m-dimensional input vectors;

- an *evaluation criteria* to analyse the quality of clusters. Two known methods are commonly used to evaluate a clustering division: internal evaluation and external evaluation. Using internal evaluation criteria, the level of accuracy of a cluster method is tackled on the clusters themselves, computing how similar data is into the produced clusters. To do so, several internal indices exist in literature: the silhouette index, computing how similar a data is with respect to the belonging cluster when compared with the other groups; the Calinski-Harabasz index; the Davies–Bouldin index. An external evaluation criteria, in contrast, analyses the behavior of a clustering algorithm comparing it with

existing reference parameters, usually computed in advance by human minds. Hence, such external criteria estimate how good a clustering division is, based on previously computed feature values having the characteristic of being accurate. Among the best known examples of external criteria there are:

  · F-measure, based on the notions of precision, i.e., the proportion of relevant data, that is data that is correctly measured;

  · sensitivity, namely how many relevant data were actually obtained;

  · general confusion matrix, comparing the obtained results with the correct standard ones;

  · rand index, measuring the fraction of fair decisions made by the clustering division;

- a *clustering algorithm*, computing such data groups. Clustering algorithms may be classified based on their inner structure into three possible techniques:

  i. *Hierarchical* clustering algorithms, where clusters are designed in order to obtain a hierarchical clustering structure. Strategies for hierarchical clustering are typically of two types: agglomerative, i.e., bottom-up; divisive, also known as top-down. Agglomerative algorithms start with dividing each input data into different clusters and continue with the gradual unification of clusters two by two, using an iterative approach. On the contrary, a divisive clustering algorithm begins with a single cluster containing all the initial points and gradually divides it into smaller clusters;

  ii. *Partitional* clustering algorithms, where data is divided into a set of of disjoint clusters such that the following requirements are satisfied:

    · each cluster contains at least one data point;

    · each data point belongs to one cluster only;

  iii. *Bayesian* clustering algorithms, a probabilistic model giving a posterior distribution over the whole space of clusters.

Dimensionality reduction

Dimensionality reduction defines the unsupervised learning technique for lowering the number of input variables when dealing with high dimensional sets of data. Working with big data, this may be a complicate task due to the difficulties caused by working in high dimensional spaces. Specifically, two major issues may occur: sparsity of data, leading to sparse input matrix, characterized by a predominance of zeros, and computational problems, such as complexity of calculations or computing delay brought on by the huge number of data to deal with.

Dimensionality reduction is often performed in high dimensional sets of data prior to using other Machine Learning techniques. Let us consider a generic dataset

$$\{x^i\}_{i=1}^n$$

defined by a number n of input elements. In addition, let us assume that each input data

$$x^i = (x_1^i, x_2^i, \cdots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m,$$

$\forall i = 1, \cdots, n$, namely x^i has dimension m. Hence the dataset is characterised by a set of inputs each one represented by a set of m features. In this scenario, dimensionality reduction focuses on finding a function from the original m-dimensional input space to a lower m'-dimension space, with the aim of decreasing the number of features. Dimensionality reduction is usually used for data compression and data visualization problems. Specifically, in instances where huge input data are considered, reducing the number of features solves the problem of both employing and storing such data. Moreover, dimensionality reduction may be used to plot data in order to visualize data characteristics, when the lower dimension space is chosen to be 2-D or 3-D. A key issue of dimensionality reduction lies in the correct selection of the dimension m', in order to retain the significant properties of the original input data. Moreover, in order to obtain meaningful information, when reducing the dimension of data, it is important to preserve the inter-correlation factors existing between inputs, namely the connections present between the input vectors in the original input space.

Dimensionality reduction techniques may be classified into two relevant classes, based on the problem it is intended to solve. When we wish to reduce the number of features to those which are considered the most representative, we talk about *feature selection*. In particular, suppose we are considering a set of data characterized by a large number of features where only a limited number of them are explicative and critical for the problem we wish to solve. Feature selection allows to keep those important features and dispose of those that do no give contribution to the resolution of the problem. On the other hand, when all the existing features are mapped onto a new space of smaller dimension we talk about *feature projection*. Differently from feature selection where reduction of the dataset is performed choosing only relevant features and removing the irrelevant ones, feature projection takes the whole set of features and project it onto a different dimension set, with the purpose of lowering the size of the features, while preserving the structure and information of the original dataset.

1.2.1.3 Reinforcement learning

If the connection between supervised and unsupervised learning can be easily traced by using the presence and the lacking of labels, i.e., expected given outputs paired to each input, reinforcement learning is an intermediate learning method distinguished by the essential notion of reward.

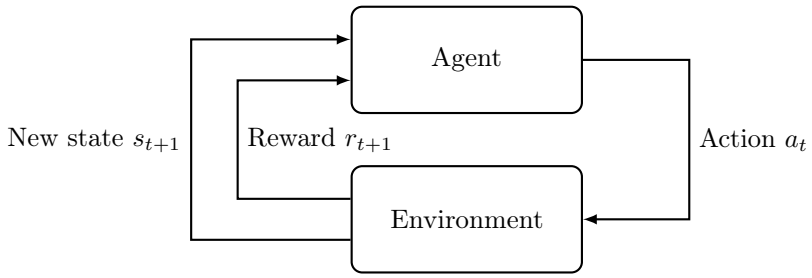


Figure 1.4: A reinforcement learning classical setting: an agent selects a specific action a_t to which the environment associates a reward r_{t+1} and a new state s_{t+1}.

A reward is nothing but a feedback provided by the environment, which is useful for the method to understand whether a specific event performed in a given phase of the algorithm is positive, namely correct, or not. In some cases, when the performed action is not proper and, hence the corresponding reward is negative, it is called penalty.

As for unsupervised learning, reinforcement learning algorithms do not need the presence of labeled data but are defined by a basic framework given by:

i. a set of *agents*, learning and making decisions;

ii. a specific *environment*, where agents learn and take decisions concerning the actions to carry;

iii. a set of *actions*, among which each agent may choose how to behave;

iv. the *state* of each agent in the environment;

v. a set of *rewards* connected to every possible action agents may pursue.

In a reinforcement learning environment, at each predefined discrete time-step, agents engage with the environment and receive a time dependent state s_t and reward r_t. Then, subsequently, agents choose a specific action a_t from the given set and send it to the environment. Consequently, the environment pairs the new action with a novel state and reward, i.e., s_{t+1} and r_{t+1}, with the main goal of maximizing the expected overall reward. Figure 1.4 shows a typical agent-environment interaction in reinforcement learning methods.

1.3 Kernel Methods: theoretical background

The term kernel method represents a research area which defines a specific group of algorithms connected by a core idea: projecting the given data from the input space into a higher dimensional space with the use of a mapping function. Specifically, when a non-linear dataset is considered, i.e., a dataset non-linearly separable in the original

input space, a kernel based algorithm allows to perform a kernel function to map the data to a feature space of higher dimension, where a linear separation may be applied. A key emblematic characteristic of this method is that only the inner products of the data needs to be computed. The concept behind kernel learning and its techniques will be deepened and refined in Chapter 3, while in this section the underlying theory of kernel methods is tackled.

One key concept about kernel methods is given by the Reproducing Kernel Hilbert Space (RKHS). First introduced by Stanislaw Zaremba in the early twentieth century [Zaremba, 1907] regarding the problem of harmonic functions, RKHS was later described and analyzed in detail by Nachman Aronszajn in 1950 [Aronszajn, 1950]. An RKHS provides a strategic framework, defined by a well planned and cost effective computation, grounds for its use in a wide selection of areas, including Machine Learning, Statistics, Group Theory and Complex Analysis. Specifically, an RKHS constructs a bijective function connecting a positive definite kernel with a Hilbert space of functions. Therefore, in order to correctly define an RKHS, it is necessary to provide the notions of Hilbert space and kernel function first.

1.3.1 Hilbert space

As the name implies, an RKHS is based on the notion of Hilbert space. A Hilbert space due its name to the German mathematician David Hilbert, whom in the early 1900s generalized the notion of Euclidean space and introduced the concept of infinite dimensional space. Such theory has been utilized, among other things, in the study of quantum mechanics and in the development of the kinetic theory of gases, other than the theory of radiation.

In order to define the notion of Hilbert space it is necessary to introduce some key definitions.

Definition 1.3.1. *Metric space.*
A metric space X is a set on which it is defined a metric function $d: \ X \times X \longrightarrow \mathbb{R}^+$, namely a function satisfying the following properties:

a) identity of indiscernibles

$$d(x, y) = 0 \iff x = y$$

b) symmetry

$$d(x, y) = d(y, x) \ \ \forall x, y \in X$$

c) triangular equality

$$d(x, z) \leq d(x, y) + d(y, z) \ \ \forall x, y, z \in X.$$

Definition 1.3.2. *Convergence sequence.*
A sequence $\{x_n\}_{n\in\mathbb{N}}$ is said to converge to, i.e., tend to, x if

$$\forall\epsilon > 0 \;\; \exists\bar{n} \in \mathbb{N} \;\; : \;\; \forall n > \bar{n} \;\; d(x_n, x) < \epsilon,$$

which is equivalent to the symbolical definition

$$\lim_{n\to\infty} x_n = x.$$

Definition 1.3.3. *Cauchy sequence.*
A sequence $\{x_n\}_{n\in\mathbb{N}}$ is said a Cauchy sequence if

$$\forall\epsilon > 0 \;\; \exists\bar{n} \in \mathbb{N} \;\; : \;\; \forall m, n \geq \bar{n} \;\; d(x_n, x_m) < \epsilon.$$

In other words, as $\bar{n}$ increases, the terms of the sequence become increasingly close to each other.

Definition 1.3.4. *Complete metric space*
A metric space in which all the Cauchy sequences are convergent is said to be complete.

Definition 1.3.5. *Real inner product.*
Consider a vector space V and the field of real numbers $\mathbb{R}$. A real inner product is a map

$$\langle\cdot,\cdot\rangle \; : \; V \times V \longrightarrow \mathbb{R}$$

assigning a real number $\langle u, v\rangle$ to every given couple of vectors u, v of the input space, such that the following conditions are satisfied:

a) symmetry

$$\langle u, v\rangle = \langle v, u\rangle$$

b) linearity in both the arguments

$$\begin{aligned}\langle a \cdot u, v\rangle &= \langle u, a \cdot v\rangle = a\langle u, v\rangle \\ \langle u + v, w\rangle &= \langle u, w\rangle + \langle u, v\rangle \\ \langle u, v + w\rangle &= \langle u, v\rangle + \langle u, w\rangle\end{aligned}$$

c) positive definiteness

$$\langle u, u\rangle \;\geq\; 0 \;\; \forall u \in V \;\; \text{and} \;\; \langle u, u\rangle = 0 \;\; \text{iff} \;\; u = 0$$

Definition 1.3.6. *Complex inner product.*
Consider a vector space V and the field of complex number $\mathbb{C}$. An inner product is a map

$$\langle\cdot,\cdot\rangle \; : \; V \times V \longrightarrow \mathbb{C}$$

assigning a complex value $\langle u, v\rangle$ to every given couple of vectors u, v of the input space, such that the following conditions are satisfied:

a) conjugate symmetry

$$\langle u, v\rangle = \overline{\langle v, u\rangle}$$

b) additivity

$$\begin{aligned}\langle a \cdot u, v\rangle &= a\langle u, v\rangle \\ \langle u, a \cdot v\rangle &= \bar{a}\langle u, v\rangle \\ \langle u + v, w\rangle &= \langle u, w\rangle + \langle u, v\rangle \\ \langle u, v + w\rangle &= \langle u, v\rangle + \langle u, w\rangle\end{aligned}$$

c) positive definiteness

$$\langle u, u\rangle \geq 0 \ \ \forall u \in V \ \text{ and } \ \langle u, u\rangle = 0 \ \text{ iff } \ u = 0$$

An inner product space is a vector space characterized by the presence of an inner product. The concept of inner product is directly linked with the notion of norm, given by

$$\|u\| := \sqrt{\langle u, u\rangle}.$$

Note that, in this thesis from now on when we consider real space inner products, for the simplicity of notation on we are going to introduce the standard dot product notation

$$\langle u, v\rangle := u^{\mathrm{T}} v,$$

where u^{T} is the transpose of u.

Definition 1.3.7. *Hilbert space*
A Hilbert space $(\mathcal{H}, \langle\cdot,\cdot\rangle_{\mathcal{H}})$ is an inner product space that is complete in the induced norm.

An incomplete space with an inner product is called *pre-Hilbert space*.

Note that, for simplicity of notation from now on we are going to consider real valued Hilbert spaces and kernel functions.

1.3.2 Kernel functions

Definition 1.3.8. *Positive definite kernel*
Let $\mathcal{X} \neq \emptyset$. A function $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ is a positive definite kernel on $\mathcal{X}$ if there exist a Hilbert space $(\mathcal{H}, \langle\cdot,\cdot\rangle_{\mathcal{H}})$ and a map $\phi : \mathcal{X} \longrightarrow \mathcal{H}$ such that $\forall x, \tilde{x} \in \mathcal{X}$

$$k(x, \tilde{x}) := \langle \phi(x), \phi(\tilde{x})\rangle_{\mathcal{H}},$$

and the following requirements are satisfied:

a) symmetry

$$k(x, \tilde{x}) = \langle \phi(x), \phi(\tilde{x})\rangle_{\mathcal{H}} = \langle \phi(\tilde{x}), \phi(x)\rangle_{\mathcal{H}} = k(\tilde{x}, x);$$

b) positive definiteness
$\forall x^1, x^2, \ldots, x^n \in \mathcal{X}$ and $c_1, c_2, \ldots, c_n \in \mathbb{R}$

$$\sum_{i,j=1}^{n} c_i c_j k(x^i, x^j) \geq 0,$$

namely, the matrix $K := (K_{ij})$ of elements $K_{ij} = k(x^i, x^j)$ is positive semi definite.

Note that, the non empty set $\mathcal{X}$ does not necessary have to be equipped with an inner product: the product $\langle \phi(x), \phi(\tilde{x}) \rangle_{\mathcal{H}}$ is the inner product defined on the Hilbert space $\mathcal{H}$.
The function ϕ is known as *feature map*, mapping points $x \in \mathcal{X}$ from the original input space to a usually higher dimensional feature space.
Given $x^1, x^2, \ldots, x^n \in \mathcal{X}$, the symmetric matrix K, with

$$K_{ij} = k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle_{\mathcal{H}},$$

is known as the *Gram matrix* or *kernel matrix*.

To prevent confusion, it is important to underline that, as specified by Definition 1.3.8, a positive definite kernel only requires the Gram matrix to be symmetric and positive semi definite (i.e., characterized by non negative eigenvalues). In the event that a kernel function is marked by a positive definite Gram matrix, namely all its eigenvalues are strictly positive, we talk about *strictly positive definite kernel*.

Definition 1.3.9. *Conditionally positive definite kernel*
Let $\mathcal{X} \neq \emptyset$. A kernel function $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ is conditionally positive definite on $\mathcal{X}$ if and only if it is symmetric and

$$\sum_{i,j=1}^{n} c_i c_j k(x^i, x^j) \geq 0$$

$\forall x^1, x^2, \ldots, x^n \in \mathcal{X}$ and $c_1, c_2, \ldots, c_n \in \mathbb{R}$ such that $\sum_{i=1}^{n} c_i = 0$.

1.3.2.1 Basic properties of kernels

Given a positive definite kernel function $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$, we have that

i. $k(x, x) \geq 0$ for every $x \in \mathcal{X}$,

ii. Cauchy-Schwarz inequality holds

$$k(x^i, x^j) \leq \sqrt{k(x^i, x^i) k(x^j, x^j)},$$

iii. given a scalar $\alpha > 0$, then αk is still a positive definite kernel.

If

$$k_i : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}, \;\; i = 1, \ldots, n$$

is a family of positive definite kernel functions. Then, the following additional properties are satisfied

iv. *Positive combination of kernels*
Given $a_1, a_2 \geq 0$, then the combination $a_1 k_1 + a_2 k_2$ is still a positive definite kernel,

v. *Product of kernels*
the product $k_1 k_2 = k_1(x, \tilde{x}) k_2(x, \tilde{x})$ is still a positive definite kernel,

vi. *Limit of kernels*
the limit

$$\lim_{i \to \infty} k_i(x, \tilde{x})$$

if exists is still a positive definite kernel.

1.3.2.2 Examples of kernels

We are now going to introduce some of the most popular kernel functions.

Let us consider a generic input set $\mathcal{X} \subseteq \mathbb{R}^m$ with corresponding inner product defined on $\mathcal{X}$ and let $x, \tilde{x} \in \mathcal{X}$ be two generic input data points.

i. *Linear kernel*
The easiest kernel map is the linear kernel. The linear kernel on x and $\tilde{x}$ is defined as

$$k(x, \tilde{x}) = x^{\mathrm{T}} \tilde{x} + c,$$

where the scalar value $c \geq 0$ is an at user's discretion constant.

ii. *Polynomial kernel*
The polynomial kernel is one of the most famous and used kernel functions, primarily for its characteristic of using features vectors from the input space as well as its combinations. Given a polynomial degree d, a constant $c \geq 0$ and a scalar parameter α, the polynomial kernel on x and $\tilde{x}$ is given by

$$k(x, \tilde{x}) = (\alpha(x^{\mathrm{T}} \tilde{x}) + c)^d.$$

iii. *Radial basis function kernel*
Given x and $\tilde{x}$, the radial basis function (RBF) kernel, also known as Gaussian kernel, is defined as

$$k(x, \tilde{x}) = \exp\left(-\frac{\|x - \tilde{x}\|^2}{2\sigma^2}\right),$$

where $\|x - \tilde{x}\|^2 = (x - \tilde{x})^{\mathrm{T}}(x - \tilde{x})$ is the squared norm of the distance of inputs x and $\tilde{x}$. When x and $\tilde{x}$ are close to each other, the value $\|x - \tilde{x}\|$ is small, otherwise for distant input points x and $\tilde{x}$, the corresponding distance $\|x - \tilde{x}\|$ will be large. The value σ is a user adjustable flexible parameter. Using the parameter $\gamma = \frac{1}{2\sigma^2}$, the radial basis function kernel is equivalent to the following formulation

$$k(x, \tilde{x}) = \exp\left(\gamma \|x - \tilde{x}\|^2\right).$$

Note that, σ (i.e., γ) is the only adaptable parameter. Therefore, the performance of the radial basis function kernel strongly depends on the best choice of it. If σ is too big (i.e., γ is too small) the kernel will act similarly to a linear function, not being able to apprehend the complexity of the dataset; otherwise, in the event that σ is too small (i.e., γ is too big), it will result in a model deficient in regularization that most likely will be heading overfitting problems.

iv. *Exponential kernel*
The exponential kernel of input values x and $\tilde{x}$ is given by

$$k(x, \tilde{x}) = \exp\left(-\frac{\|x - \tilde{x}\|}{2\sigma^2}\right).$$

It is quite similar to the radial basis function kernel structure, with only the square of the norm of the distance of inputs x and $\tilde{x}$ not present.

v. *Laplace kernel*
The Laplace kernel has a structure similar with respect to the exponential kernel defined as

$$k(x, \tilde{x}) = \exp\left(-\frac{\|x - \tilde{x}\|}{\sigma}\right),$$

with $\|x - \tilde{x}\|^2 = (x - \tilde{x})^{\mathrm{T}}(x - \tilde{x})$ the norm of the distance of x and $\tilde{x}$ as before.

vi. *Hyperbolic tangent kernel*
The Hyperbolic tangent kernel, also known as Sigmoid kernel, is defined as

$$k(x, \tilde{x}) = \tanh(\nu + \kappa(x^{\mathrm{T}}\tilde{x})),$$

with $\kappa > 0$ and $\nu < 0$.

vii. *ANOVA radial basis kernel*
Given a generic $d > 0$, the ANOVA kernel of degree d equation is

$$k(x, \tilde{x}) = \sum_{k=1}^{m} \exp(-\sigma(x_k - \tilde{x}_k)^2)^d,$$

where notations x_k and $\tilde{x}_k$ respectively identify the k-th component of input vectors x and $\tilde{x}$, $0 < k \leq m$.

viii. *Bessel kernel*

The Bessel kernel equation is given by

$$k(x,\tilde{x}) = \frac{J_{v+1}(\sigma\|x-\tilde{x}\|)}{\|x-\tilde{x}\|^{-n(v+1)}},$$

where $J(\cdot)$ is the Bessel function of the first kind defined as

$$J_\alpha(x) = \sum_{p=0}^{\infty} \frac{(-1)^p}{p!\Gamma(p+\alpha+1)}\left(\frac{x}{2}\right)^{2p+\alpha},$$

with $\Gamma(\cdot)$ being the gamma function, namely an extension of the concept of the factorial function $\Gamma(t) = (t-1)!$ to complex numbers.

ix. *Matérn kernel*

The Matérn kernel of parameter ν is

$$k(x,\tilde{x}) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left(\frac{\sqrt{2\nu}}{l}\|x-\tilde{x}\|)\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{l}\|x-\tilde{x}\|\right),$$

where $K_\nu(\cdot)$ is a modified Bessel function and $\Gamma(\cdot)$ is the Gamma function as above. Note that, the Matérn kernel strongly depends on the value of the parameter ν: for small values we get a rough approximated function while, if $\nu \to \infty$, the Matérn kernel reduces to the radial basis function kernel.

x. *Rational quadratic kernel*

Given $l, \alpha > 0$, the rational quadratic kernel formula is

$$k(x,\tilde{x}) = \sigma^2\left(1 + \frac{\|x-\tilde{x}\|^2}{2\alpha\ell^2}\right)^{-\alpha},$$

where l is a length scale parameter, α is a scale mixture parameter and σ^2 is the pooled variance, i.e., an estimate of the variance of a given samples when each sample has different mean values.

xi. *Cauchy kernel*

Generalizing the notion of Cauchy distribution we obtain the Cauchy kernel as follows

$$k(x,\tilde{x}) = \frac{1}{1+\frac{\|x.\tilde{x}\|^2}{\sigma^2}},$$

with user selected parameter σ.

xii. *Logarithmic kernel*

The logarithmic kernel function is given by

$$k(x,\tilde{x}) = -\log(\|x-\tilde{x}\|^d + 1),$$

with parameter $d > 0$. Note that, the logarithmic kernel is only conditionally positive definite (c.p.d), namely Definition 1.3.9 holds.

xiii. *Chi-square kernel*

The chi-square (χ^2) kernel, also known as intersection kernel, is a commonly used kernel function to study resemblances within images measuring their distribution with respect to specific features such as orientation, colours and so on. It is defined as

$$k(x, \tilde{x}) = 1 - \left(- \sum_{k=1}^{m} \frac{(x_k - \tilde{x}_k)^2}{\frac{1}{2}(x_k + \tilde{x}_k)} \right).$$

Note that, as above, the χ^2 kernel is only conditionally positive definite.

1.3.3 Reproducing Kernel Hilbert Space

Seamlessly from the previous section, we are going to consider Hilbert spaces over the field of real numbers $\mathbb{R}$. It is important to stress that, unless stated otherwise, all the definitions or results hold for the field of real numbers $\mathbb{R}$ as well as for the complex field $\mathbb{C}$.

Let $\mathcal{X}$ be a non empty set and let $\mathcal{F}(\mathcal{X}, \mathbb{R})$ be the set of all functions having domain $\mathcal{X}$ and codomain $\mathbb{R}$, where the following pointwise operations are defined

1. *addition*
   $\forall f, g : \mathcal{X} \longrightarrow \mathbb{R}$ and $\forall x \in \mathcal{X}$
   $$(f + g)(x) = f(x) + g(x),$$

2. *scalar multiplication*
   $\forall f : \mathcal{X} \longrightarrow \mathbb{R}$ and $\forall c \in \mathbb{R}$
   $$(c \cdot f)(x) = c \cdot f(x).$$

Definition 1.3.10. *Reproducing Kernel Hilbert Space*
A set $\mathcal{H} \subseteq \mathcal{F}(\mathcal{X}, \mathbb{R})$ is a Reproducing Kernel Hilbert Space (RKHS) on $\mathcal{X}$ if the following conditions are true.

1. $\mathcal{H}$ is a vector subspace of the function space $\mathcal{F}(\mathcal{X}, \mathbb{R})$;

2. $\mathcal{H}$ is equipped with an inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ in relation to which it is a Hilbert space, namely $\mathcal{H}$ is complete in the induced norm;

3. $\forall x \in \mathcal{X}$ the function $E_x : \mathcal{H} \longrightarrow \mathbb{R}$ defined as
   $$E_x(f) = f(x)$$
   is bounded.

The map E_x is known as *evaluation functional* at point x. The evaluation functional is always linear, namely $\forall f, g \in \mathcal{H}$ and $\forall \alpha, w \in \mathbb{R}$, we have that

$$E_x(\alpha f + wg) = (\alpha f + wg)(x) = \alpha f(x) + wg(x) = \alpha E_x(f) + wE_x(g).$$

Note that, the evaluation functional is not necessary a continuous function.

Example 1. Let us consider the Hilbert space $\mathcal{H} = L^2([0,1])$, i.e., the space of all square integrable functions $f : [0,1] \longrightarrow \mathbb{R}$, with induced norm

$$\|f_1 - f_2\|_{L^2([0,1])} = \sqrt{\int_0^1 |f_1(x) - f_2(x)|^2 \mathrm{d}x}.$$

Now, let us consider the sequence $\{s_n\}_{n=1}^{\infty}$, with $s_n = x^n$ and the evaluation functional E_1. We have that

$$\lim_{n\to\infty} \|s_n - 0\|_{L^2([0,1])} = 0,$$

namely, the sequence of functions $\{s_n\}_{n=1}^{\infty}$ converges to the zero function (i.e., the function which has always value equal to 0) with respect to the induced norm. Therefore, we have that

$$0 = E_1(\lim_{n\to\infty} s_n) \neq \lim_{n\to\infty} E_1(s_n) = \lim_{n\to\infty} 1^n = 1,$$

namely, the evaluation functional E_1 is not continuous.

Now, let us consider a Hilbert space $(\mathcal{H}, \langle\cdot,\cdot\rangle_{\mathcal{H}})$ and let $\mathcal{H}^*$ be its continuous dual space given by all bounded linear functional from $\mathcal{H}$ to $\mathbb{R}$. More precisely, let $\mathcal{H}^*$ be a set defined as

$$\mathcal{H}^* = \{f : \mathcal{H} \longrightarrow \mathbb{R} \text{ s.t. } f \text{ bounded linear functional}\}.$$

Theorem 1. *[Riesz, 1909]* ***The Riesz Representation Theorem for Hilbert Spaces***
Let $\mathcal{H}$ be a Hilbert space. For every $\phi \in \mathcal{H}^*$ there exists a unique $f \in \mathcal{H}$ such that for every $x \in \mathcal{H}$,

$$\phi(x) = \langle f, x\rangle_{\mathcal{H}}. \tag{1.2}$$

Proof. First of all, since a bounded operator is always continuous on norm-spaces, we have that $\phi(x)$ is a continuous linear functional.
If $\phi(x) = 0$ for all $x \in \mathcal{H}$, we can trivially choose $f = 0$. Otherwise, let us define the set

$$M = \{x \in \mathcal{H} : \phi(x) = 0\}.$$

The linearity of $\phi(x)$ shows that M is a subspace of $\mathcal{H}$, while the continuity of $\phi(x)$ shows that M is closed. We state that $M^{\perp} = \{y \mid \forall x \in M, \langle x, y\rangle_{\mathcal{H}} = 0\}$ is a one dimensional set. Let y_1, y_2 be two non zero vectors in $M^{\perp}$; by construction, $\phi(y_1) \neq 0$ and $\phi(y_2) \neq 0$. Moreover, since $y_1, y_2 \in M^{\perp}$, we have that $\langle x, y_1\rangle_{\mathcal{H}} = \langle x, y_2\rangle_{\mathcal{H}} = 0$, for all $x \in M$. Hence, there must exist a scalar $\lambda \neq 0$ such that $\lambda\phi(y_1) = \phi(y_2)$. Since $\lambda y_1 - y_2 \in M^{\perp}$ and $\phi(\lambda y_1 - y_2) = 0$, we have $\lambda y_1 - y_2 \in M$, namely, $\lambda y_1 - y_2 = 0$.
Let $z \in M^{\perp}$ with $\|z\| = 1$. Let us set

$$w = (\phi(x))z - (\phi(z))x.$$

By construction we have that $\phi(w) = (\phi(x))(\phi(z)) - (\phi(z))(\phi(x)) = 0$, hence $w \in M$ and $\langle w, z\rangle_{\mathcal{H}} = 0$. This leads to

$$\phi(x) = (\phi(x))\langle z, z\rangle_{\mathcal{H}} = (\phi(z))\langle x, z\rangle_{\mathcal{H}}.$$

Thus, Eq. (1.2) holds with $f = \alpha z$, $\alpha = \phi(z)$.

The element f is unique. By contradiction, let f_1, f_2 such that $\phi(x) = \langle f_1, x\rangle_{\mathcal{H}}$ and $\phi(x) = \langle f_2, x\rangle_{\mathcal{H}}$. By Cauchy-Schwarz inequality we have that

$$0 = |\phi(x) - \phi(x)| = |\langle x, f_1 - f_2\rangle_{\mathcal{H}}| \leq \|x\|_{\mathcal{H}}\|f_1 - f_2\|_{\mathcal{H}}.$$

Hence, $\|f_1 - f_2\| = 0$, which implies $f_1 = f_2$. □

As a direct consequence, if $\mathcal{H}$ is an RKHS on $\mathcal{X}$, for every $x \in \mathcal{X}$ there exists a unique vector $k_x \in \mathcal{H}$ such that, for every $f \in \mathcal{H}$

$$f(x) = E_x(f) = \langle f, k_x\rangle_{\mathcal{H}}. \tag{1.3}$$

Definition 1.3.11. *Reproducing kernel*
Let $(\mathcal{H}, \langle\cdot,\cdot\rangle_{\mathcal{H}})$ be a Hilbert space on $\mathbb{R}$-valued functions over a non empty set $\mathcal{X}$. A function $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ defined by

$$k(x, y) = k_y(x)$$

is called a reproducing kernel for $\mathcal{H}$ if the following conditions are satisfied

1. $\forall x \in \mathcal{X}$, $k(\cdot, x) = k_x \in \mathcal{H}$,
2. $\forall x \in \mathcal{X}$ and $\forall f \in \mathcal{H}$, $\langle f, k(\cdot, x)\rangle_{\mathcal{H}} = f(x)$.

The latter condition is known as the reproducing property.

Note that, for every $x \in \mathcal{X}$, the associated function $k(\cdot, x) = k_x$ is called the reproducing kernel for the point x. Moreover, we have

$$k(x, y) = k_y(x) = \langle k_y, k_x\rangle_{\mathcal{H}}. \tag{1.4}$$

The next result shows a key property of reproducing kernels.

Proposition 2. *If a reproducing kernel exists, it is unique.*

Proof. We assume ad absurdum that $\mathcal{H}$ has two reproducing kernels k_1 and k_2. By definition we have that

$$\langle f, k_1(\cdot, x) - k_2(\cdot, x)\rangle_{\mathcal{H}} = f(x) - f(x),\ \forall f \in \mathcal{H}, \forall x \in \mathcal{X}.$$

Choosing $f \coloneqq k_1(\cdot, x) - k_2(\cdot, x)$, we have $\|k_1(\cdot, x) - k_2(\cdot, x)\|^2_{\mathcal{H}} = 0$, $\forall x \in \mathcal{X}$. Hence, $k_1 = k_2$. □

Definition 1.3.11 provides a formal notion of which properties a reproducing kernel has to verify but does not establish any correlations between the two main concepts of RKHS and reproducing kernel.

Theorem 3. *[Berlinet and Thomas-Agnan, 2011]* ***Existence of reproducing kernel*** *A Hilbert space $\mathcal{H}$ is a reproducing kernel Hilbert space if and only if $\mathcal{H}$ is fitted with a reproducing kernel.*

Proof. ($\Longrightarrow$) The fact that if $\mathcal{H}$ is a reproducing kernel Hilbert then $\mathcal{H}$ has a reproducing kernel is a direct consequence of the Riesz representation Theorem for Hilbert spaces (Theorem 1).

($\Longleftarrow$) To prove the other implication, we assume that $\mathcal{H}$ is a Hilbert space equipped with a reproducing kernel k with the reproducing property

$$\langle f, k(\cdot, x)\rangle_{\mathcal{H}} = f(x).$$

Then

$$\begin{aligned} |E_x(f)| &= |f(x)| \\ &= |\langle f, k(\cdot, x)\rangle_{\mathcal{H}}| \\ &\leq \|k(\cdot, x)\|_{\mathcal{H}}\|f\|_{\mathcal{H}} \\ &= \langle k(\cdot, x), k(\cdot, x)\rangle_{\mathcal{H}}^{1/2}\|f\|_{\mathcal{H}} \\ &= k(x, x)^{1/2}\|f\|_{\mathcal{H}}, \end{aligned}$$

where in the third line we used the Cauchy-Schwarz inequality. Therefore, the function $E_x : \mathcal{H} \longrightarrow \mathbb{R}$ defined as

$$E_x(f) = f(x)$$

is a bounded linear evaluation functional. □

Lemma 4. *Let $\mathcal{H}$ be a Hilbert space with inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$, $\mathcal{X}$ a nonempty set and $\phi : \mathcal{X} \longrightarrow \mathcal{H}$. Then, $h(x, y) \coloneqq \langle\phi(x), \phi(y)\rangle_{\mathcal{H}}$ is a positive definite function.*

Proof. By definition, we have to prove that $\forall n \geq 1$, $\forall(a_1, \ldots, a_n) \in \mathbb{R}^n$, $\forall(x^1, \ldots, x^n) \in \mathcal{X}^n$, the following condition is true

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j h(x^i, x^j) \geq 0. \tag{1.5}$$

We have that

$$\begin{aligned} \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j h(x^i, x^j) &= \sum_{i=1}^{n}\sum_{j=1}^{n}\langle a_i\phi(x^i), a_j\phi(x^j)\rangle_{\mathcal{H}} \\ &= \left\langle \sum_{i=1}^{n} a_i\phi(x^i), \sum_{j=1}^{n} a_j\phi(x^j) \right\rangle_{\mathcal{H}} \\ &= \left\| \sum_{i=1}^{n} a_i\phi(x^i) \right\|_{\mathcal{H}}^2 \geq 0, \end{aligned}$$

Hence, condition (1.5) is verified. □

Corollary 5. *Reproducing kernels are positive definite.*

Proof. By Definition (1.3.11) we have that

$$k(x, y) = k_y(x) = \langle k_y, k_x \rangle_{\mathcal{H}}.$$

Therefore, to prove the assert it is enough to take $\phi : x \mapsto k_x$. □

Using Lemma 4, it is possible to identify a notion of reproducing kernel as a function which can be defined as a inner product. Note that, from now on we are going to use the most basic term kernel, cutting out the word reproducing.

Definition 1.3.12. *Kernel*
Given $\mathcal{X}$ a nonempty set, a function $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ is a kernel if there exists a real Hilbert space $(\mathcal{H}, \langle \cdot, \cdot \rangle_{\mathcal{H}})$ and a map $\phi : \mathcal{X} \longrightarrow \mathcal{H}$ such that for all $x, y \in \mathcal{H}$

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}.$$

The map ϕ is known as the *feature map* and space $\mathcal{H}$ is known as *feature space*.

It is noteworthy that, given a specific kernel function k, the feature map may not be unique but different functions ϕ and related feature spaces may exist associated to the same kernel.

Example 2. Let $\mathcal{X} = \mathbb{R}$, x and $\tilde{x}$ two generic input points and $k(x, \tilde{x}) = x\tilde{x}$. Trivially it is easy to see that function $\phi(x) = x$ is a feature map with associated feature space $\mathcal{H} = \mathbb{R}$.
Similarly, we have that

$$k(x, \tilde{x}) = \begin{bmatrix} \frac{x}{\sqrt{2}} & \frac{x}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{\tilde{x}}{\sqrt{2}} \\ \frac{\tilde{x}}{\sqrt{2}} \end{bmatrix},$$

with feature map

$$\phi'(x) = \begin{bmatrix} \frac{x}{\sqrt{2}} \\ \frac{x}{\sqrt{2}} \end{bmatrix}$$

and feature space $= \mathbb{R}^2$.

Note that, in Example 2 neither of the two possible choices of feature space are RKHS, since both $\mathcal{H}$ and $\tilde{\mathcal{H}}$ are not spaces of functions on the original space $\mathcal{X} = \mathbb{R}$.

In Corollary 5 we have proved that all kernels are positive definite functions and by Definition 1.3.12 we know that every kernel functions may be expressed as an inner product of elements lying in the feature space, pointing out a close relationship between inner products and kernel functions. Specifically, the following theorem expresses such concept stating that for every positive definite function there exists a unique Reproducing Kernel Hilbert Space.

Theorem 6. *[Aronszajn, 1950]* ***Moore-Aronszajn Theorem***
Let $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ be a positive definite function. Then, there is a unique reproducing kernel Hilbert space $\mathcal{H}$ with reproducing kernel k.

Proof. Let

$$\mathcal{H}_0 = \text{span}[\{k(\cdot, x)_{x \in \mathcal{X}}\}]$$

be the linear span of set $\{k(\cdot, x)_{x \in \mathcal{X}}\}$ with corresponding inner product

$$\langle f, g \rangle_{\mathcal{H}_0} = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i w_j k(x^i, \tilde{x}^j), \tag{1.6}$$

where

$$f = \sum_{i=1}^{n} \alpha_i k(\cdot, x^i)$$

and

$$g = \sum_{j=1}^{m} w_j k(\cdot, \tilde{x}^j),$$

which implies that

$$k(x, y) = \langle k(\cdot, x), k(\cdot, \tilde{x}) \rangle_{\mathcal{H}_0}.$$

Hence, if we define $k_x = k(\cdot, x)$, $\forall x \in \mathcal{X}$, we have that

$$k(x, \tilde{x}) = \langle k_x, k_{\tilde{x}} \rangle_{\mathcal{H}_0}.$$

First of all, (1.6) is a valid inner product on $\mathcal{H}_0$. By construction, the only condition of Definition 1.3.5 we need to verify is that

$$\langle f, f \rangle_{\mathcal{H}_0} = 0 \Longrightarrow f = 0.$$

Let

$$f = \sum_{i=1}^{n} \alpha_i k(\cdot, x^i).$$

Let fix $x \in \mathcal{X}$ and take $a_i = a\alpha_i$, $i = 1, \ldots, n$, $a_{n+1} = f(x)$ and $x^{n+1} = x$. Since k is positive definite by hypothesis, we have that

$$\begin{aligned} 0 &\leq \sum_{i=1}^{n+1} \sum_{j=1}^{n+1} a_i b_j k(x^i, \tilde{x}^j) \\ &= a^2 \langle f, f \rangle_{\mathcal{H}_0} + 2a|f(x)|^2 + |f(x)|^2 k(x, x). \end{aligned}$$

Again, since k is positive definite, we have that $|f(x)|^4 \leq |f(x)|^2 k(x, x) \langle f, f \rangle_{\mathcal{H}_0}$, which shows that if $\langle f, f \rangle_{\mathcal{H}_0} = 0$, then $f = 0$. At this step, space $\mathcal{H}_0$ with its induced norm $\langle \cdot, \cdot, \rangle_{\mathcal{H}_0}$ is a pre-Hilbert space.

Let $x \in \mathcal{X}$ and the evaluation functional E_x, for $f = \sum_{i=1}^{n} \alpha_i k(\cdot, x^i)$. We have that

$$\langle f, k(\cdot, x) \rangle_{\mathcal{H}_0} = \sum_{i=1}^{n} \alpha_i k(x, x^i) = f(x).$$

Therefore, $\forall f, g \in \mathcal{H}_0$, using the Cauchy-Schwarz inequality, the following is true

$$\begin{aligned} |E_x - E_{\tilde{x}}| &= |\langle f - g, k(\cdot, x)\rangle_{\mathcal{H}_0}| \\ &\leq \sqrt{k(x,x)}\|f - g\|_{\mathcal{H}_0}. \end{aligned}$$

Hence, the functional E_x is continuous on $\mathcal{H}_0$.

Now, we wish to show that every $\mathcal{H}_0$-Cauchy sequence $\{f_n\}$ which converges point-wise to 0, converges in $\mathcal{H}_0$-norm to 0 as well. Let $\{f_n\}$ be such $\mathcal{H}_0$-Cauchy sequence converging point-wise to 0 and take $\epsilon > 0$. Since all Cauchy sequences are bounded, there exists $S > 0$ such that $\|f_n\|_{\mathcal{H}_0} < S$, $\forall n \in \mathbb{N}$. As a direct consequence, there exists $N_1 \in \mathbb{N}$ such that $\|f_n - f_m\|_{\mathcal{H}_0} < \epsilon/2S$, for $n, m \geq N_1$ and we can write

$$f_{N_1} = \sum_{i=1}^{r} \alpha_i k(\cdot, x^i).$$

Let us take $N_2 \in \mathbb{N}$ such that for all $n \geq N_2$ we have that

$$|f_n(x^i)| < \frac{\epsilon}{2r|\alpha_i|},$$

$i = 1, \ldots, r$. Hence, for $n \geq \max(N_1, N_2)$, we have that

$$\begin{aligned} \|f_n\|^2_{\mathcal{H}_0} &\leq |\langle f_n - f_{N_1}, f_n\rangle_{\mathcal{H}_0}| + |\langle f_{N_1}, f_n\rangle_{\mathcal{H}_0}| \\ &\leq \|f_n - f_{N_1}\|_{\mathcal{H}_0}\|f_n\|_{\mathcal{H}_0} + \sum_{i=1}^{r} |\alpha_i f_n(x^i)| \\ &< \epsilon, \end{aligned} \tag{1.7}$$

hence, the $\mathcal{H}_0$-Cauchy sequence f_n converges in $\mathcal{H}_0$-norm to 0.

We now need to extend $\mathcal{H}_0$ to make it complete. First of all, note that for any Cauchy sequence $\{f_n\}$ in $\mathcal{H}_0$, $\forall x \in \mathcal{X}$, $m, n \in \mathbb{N}$ we have that

$$|f_n(x) - f_m(x)| \leq \|f_n - f_m\|_{\mathcal{H}_0}\sqrt{k(x,x)}.$$

Hence, for every $x \in \mathcal{X}$ the sequence $\{(f_n(x))\}$ is a Cauchy sequence in $\mathbb{R}$ and has a limit. Let us consider $\mathcal{H}$ as the set of functions $f : \mathcal{X} \longrightarrow \mathbb{R}$ which are point-wise limits of Cauchy sequences $\{f_n\}$ in $\mathcal{H}_0$. Namely, if $\{f_n\}$ is a Cauchy sequence in $\mathcal{H}_0$, then $\lim_{n\to\infty} f_n(x) = f(x)$ is in $\mathcal{H}$. First of all, by construction we have that $\mathcal{H}_0 \subset \mathcal{H}$.

Next, let define an inner product on $\mathcal{H}$ and let show that $\mathcal{H}$ with its induced inner product is a Reproducing Kernel Hilbert Space with reproducing kernel k. Let us consider two Cauchy sequences $\{f_n\}$ and $\{g_n\}$ in $\mathcal{H}_0$ and their corresponding functions f and g in $\mathcal{H}$. Using again the Cauchy-Schwarz inequality we have that $\forall n, m \in \mathbb{N}$

$$\begin{aligned} |\langle f_n, g_n\rangle_{\mathcal{H}_0} - \langle f_m, g_m\rangle_{\mathcal{H}_0}| &= |\langle f_n, g_n - g_m\rangle_{\mathcal{H}_0} + \langle f_n - f_m, g_n\rangle_{\mathcal{H}_0} \\ &\leq \|f_n\|_{\mathcal{H}_0}\|g_n - g_m\|_{\mathcal{H}_0} + \|f_n - f_m\|_{\mathcal{H}_0}\|g_m\|_{\mathcal{H}_0}. \end{aligned}$$

Hence, the sequence $\{\langle f_n, g_n\rangle_{\mathcal{H}_0}\}_{n\in\mathbb{N}}$ is a Cauchy sequence in $\mathbb{R}$ and the inner product $\langle f_n, g_n\rangle_{\mathcal{H}_0}$ converges. Moreover, let us us consider two Cauchy sequences $\{f'_n\}$ and $\{(g'_n)\}$ in $\mathcal{H}_0$ which converge respectively to f and g in $\mathcal{H}$ as well. Then from (1.7) we known that

$$\lim_{n\to\infty} \|f_n - f'_n\|_{\mathcal{H}_0} = 0$$

and, respectively,

$$\lim_{n\to\infty} \|g_n - g'_n\|_{\mathcal{H}_0}.$$

Moreover,

$$\begin{aligned}|\langle f_n, g_n\rangle_{\mathcal{H}_0} - \langle f'_n, g'_n\rangle_{\mathcal{H}_0}| &= |\langle f_n, g_n - g'_n\rangle_{\mathcal{H}_0} + \langle f_n - f'_n, g_n\rangle_{\mathcal{H}_0} \\ &\leq \|f_n\|_{\mathcal{H}_0}\|g_n - g'_n\|_{\mathcal{H}_0} + \|f_n - f'_n\|_{\mathcal{H}_0}\|g_n\|_{\mathcal{H}_0},\end{aligned}$$

hence $\langle f_n, g_n\rangle_{\mathcal{H}_0}$ and $\langle f'_n, g'_n\rangle_{\mathcal{H}_0}$ have the same limit only depending on the functions f and g. We can now define a inner product on $\mathcal{H}$ as

$$\langle f, g\rangle_{\mathcal{H}} = \lim_{n\to\infty} \langle f_n, g_n\rangle_{\mathcal{H}_0}.$$

By the properties of $\langle\cdot,\cdot\rangle_{\mathcal{H}_0}$, we know that $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ is a positive definite map. Moreover, given a Cauchy sequences $\{f_n\}$ in $\mathcal{H}_0$ such that $\lim_{n\to\infty}\|f_n\|_{\mathcal{H}_0} = 0$, then $f = 0$. Hence, space $\mathcal{H}$ with its induced norm $\langle\cdot,\cdot,\rangle_{\mathcal{H}}$ is a pre-Hilbert space.

We now need to show that $\mathcal{H}$ is a complete space. First of all, we note that $\mathcal{H}_0$ is dense in $\mathcal{H}$: for any $f \in \mathcal{H}$ defined as the point-wise limit of $\{f_n\}$ in $\mathcal{H}_0$, we have that $f_n \in \mathcal{H}$ for any $n \in \mathbb{N}$, and

$$\lim_{n\to\infty} \langle f, f_n\rangle_{\mathcal{H}} = \lim_{p\to\infty}\lim_{n\to\infty} \langle f_p, f_n\rangle_{\mathcal{H}_0}.$$

Let $\{f_n\}$ be a Cauchy sequence in $\mathcal{H}$. For each $n \in \mathbb{N}$ we can define $f'_n \in \mathcal{H}_0$ such that

$$\lim_{n\to\infty} \|f_n - f'_n\|_{\mathcal{H}} = 0.$$

For all $\epsilon > 0$, let $N \in \mathbb{N}$ such that $\forall n, m > N$, $\|f_n - f_m\|_{\mathcal{H}} < \epsilon/3$ and $\|f_n - f'_n\|_{\mathcal{H}} < \epsilon/3$, then

$$\begin{aligned}\|f'_n - f'_m\|_{\mathcal{H}_0} &= \|f'_n - f'_m\|_{\mathcal{H}} \\ &\leq \|f'_n - f_n\|_{\mathcal{H}} + \|f_n - f_m\|_{\mathcal{H}} + \|f_m - f'_m\|_{\mathcal{H}} \\ &\leq \frac{\epsilon}{3} + \frac{\epsilon}{3} + \frac{\epsilon}{3} \\ &\leq \epsilon.\end{aligned}$$

Hence, $\{f'_n\}$ is a Cauchy sequence in $\mathcal{H}_0$, defining a function $f \in \mathcal{H}$. Moreover,

$$\lim_{n\to\infty} \|f - f'_n\|_{\mathcal{H}} = 0,$$

hence

$$\lim_{n\to\infty} \|f - f_n\|_{\mathcal{H}} \leq \lim_{n\to\infty} \|f - f'_n\|_{\mathcal{H}} + \lim_{n\to\infty} \|f'_n - f_n\|_{\mathcal{H}}.$$

Therefore, f is the limit of the sequence $\{f_n\}$ leading to $\mathcal{H}$ complete. Hence space $\mathcal{H}$ is a Hilbert space.

Finally, k is a reproducing kernel on $\mathcal{H}$. If $f \in \mathcal{H}$, then by construction the corresponding Cauchy sequence $\{f_n\} \in \mathcal{H}_0$ converges point-wise to f. Moreover, for every $x \in \mathcal{X}$, we have that $k(\cdot, \cdot) \in \mathcal{H}_0$ and $\mathcal{H}_0 \subset \mathcal{H}$. By construction of the inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, we have that

$$\begin{aligned} \langle f, k(\cdot, x) \rangle_{\mathcal{H}} &= \lim_{n \to \infty} \langle f_n, k(\cdot, x) \rangle_{\mathcal{H}_0} \\ &= \lim_{n \to \infty} f_n(x) \\ &= f(x). \end{aligned}$$

Therefore, the function k is a reproducing kernel for $\mathcal{H}$. Considering that $\mathcal{H}_0$ is dense in $\mathcal{H}$, the space $\mathcal{H}$ is the unique Reproducing Kernel Hilbert Space containing $\mathcal{H}_0$ with reproducing kernel k. □

Basically, the Moore-Aronszajn Theorem shows that for every positive definite function k there exists a unique Reproducing Kernel Hilbert Space $\mathcal{H}$ for which the function k is a reproducing kernel. More generally, we have that every positive definite function is a (reproducing) kernel.

An alternative design and understanding of kernel function and Reproducing Kernel Hilbert Space is given by the use of Mercer's theorem, particularly using the idea of *integral operator* of a kernel k.

Definition 1.3.13. *Integral operator*
Let $\mathcal{X}$ be a compact metric space, μ a Borel measure on $\mathcal{X}$ and $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ a positive definite kernel on $\mathcal{X}$. In addition, assume that

$$\int \int_{\mathcal{X}} k(x, \tilde{x})^2 \, d\mu(x) \, d\mu(\tilde{x}) < \infty. \tag{1.8}$$

The map $L_k : L^2_\mu(\mathcal{X}) \longrightarrow L^2_\mu(\mathcal{X})$ defined by

$$\int_{\mathcal{X}} k(x, \tilde{x}) f(\tilde{x}) \, d\mu(\tilde{x})$$

is the integral operator of kernel k.

Note that, since k is symmetric, then L_k is a self-adjoint operator, namely $\forall f, g \in L^2_\mu$, then

$$\langle f, L_k g \rangle_{L^2_\mu} = \langle L_k f, g \rangle_{L^2_\mu}.$$

The positive definiteness of k leads to L_k positive operator, i.e.,

$$\langle f, L_k f \rangle_{L^2_\mu} \geq 0, \ \ \forall f \in L^2_\mu.$$

Moreover, if kernel k is continuous, L_k is a compact operator.

Hence, under the hypothesis of corresponding positive definite kernel, the integral operator L_k is a self-adjoint, positive, compact operator and the following result may be applied.

Theorem 7. *[Reed, 2012]* ***Spectral Theorem***
Let $\mathcal{H}$ be a Hilbert space and $L : \mathcal{H} \longrightarrow \mathcal{H}$ be a self-adjoint, positive, compact operator. Then, there exists an orthonormal basis $\{e_i\}$ of eigenvectors of L_k with corresponding eigenvalues $\{\lambda_i\}$ with $|\lambda_1| \geq |\lambda_2| \geq \cdots > 0$ and $\lambda_i \to 0$, such that

$$Lf = \sum_i \lambda_i \langle f, e_i \rangle_{\mathcal{H}} e_i, \quad f \in \mathcal{H}.$$

In other words, a self-adjoint, positive, compact operator can be unitarily diagonalized in an appropriate orthonormal basis.

Theorem 8. *[Mercer, 1909]* ***Mercer's Theorem***
Let $\mathcal{X}$ be a compact metric space, μ be a strictly positive finite Borel measure on $\mathcal{X}$ and let $k(\cdot, \cdot)$ be a continuous positive definite kernel function satisfying condition (1.8). Then

$$k(x, \tilde{x}) = \sum_i \lambda_i e_i(x) e_i(\tilde{x}),$$

where the series converges uniformly on $\mathcal{X} \times \mathcal{X}$ and absolutely for each pair $(x, y) \in \mathcal{X} \times \mathcal{X}$.

Note that, beyond the basic assumptions and under the further hypothesis that $\mathcal{X}$ is a compact metric space and $k : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ is a continuous positive definite kernel, Mercer's theorem shows an alternative characterization for the feature map ϕ:

$$\begin{aligned} k(x, y) &= \sum_i \lambda_i e_i(x) e_i(y) \\ &= (\sqrt{\lambda_i} e_i(x))^{\mathrm{T}} (\sqrt{\lambda_i} e_i(y)). \end{aligned}$$

Hence, we can pick the sequence space ℓ^2 as feature space with corresponding feature map of the form

$$\phi : x \mapsto \left\{ \sqrt{\lambda_i} e_i(x) \right\},$$

for every $x \in \mathcal{X}$.

Chapter 2

Linear Learning Models

Linear learning defines the Machine Learning key category concerning linear models. The term linear models includes all the Machine Learning methods using a linear function to determine the best fitting predictor for a set of given data.

As described in the next sections, linear models are undoubtedly the simplest one to be implemented and solved, since they rely on the assumption that output values may be approximated using a linear combination of the input vectors. This assumption has a different meaning when applied to classification dataset, i.e., data with corresponding output belonging to a specific binary class, or if implemented at regression dataset, i.e., with real valued outputs. Specifically, in classification problems this implies that data may be correctly separated through the use of a hyperplane.

First of all, it is important to understand if, given a specific set of points, it is possible to use a linear model to fit the data or, otherwise, if the considered dataset has a non-linear behavior and a different non-linear model should be used.

First in this chapter, a description of the most used and known linear models is presented. In the field of linear classification, the Perceptron algorithm is introduced, describing its learning algorithm in a schematized way and proving its convergence for linearly separable set of data. Then, the Support Vector Machine algorithm is described in the specific case of linearly separable two-class dataset. Finally, the logistic regression method is described, focusing on the maximum likelihood estimation method used for computing the related coefficients.

After that, the linear regression method is proposed. Specifically, first the closed form solution is presented, namely the actual mathematical expression for the optimum value; after that, the gradient descent rule applied to the least square method is described, analyzing one of the most popular indirect strategies available.

Finally, a linear learning framework approach commonly used in unsupervised problems is proposed: the Linear Discriminant Analysis. Linear Discriminant Analysis algorithm is used both for classification problems and dimensionality reduction issues,

with a final discussion on the similarities and differences between Linear Discriminant Analysis and Support Vector Machine.

2.1 Supervised Learning

2.1.1 Linear Classification

The term *linear classification* describes the supervised learning research field characterized by the general goal of classifying labeled data through the use of a linear combination of input feature variables. As described in Chapter 1, when only raw data is available, it is necessary to extract features useful in the actual learning phase. Such features are used to obtain a specific learning function, taking into account feature characteristics and input-output correlation factors. Concerning the output, in classification we tackle the problem of predicting a discrete output value. Specifically, we are focusing our attention on the task of binary classification, namely the problem of finding an output value which can take only two possible quantities.

Let us assume

$$\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n \tag{2.1}$$

to be a generic labeled dataset, where n specifies the number of elements, $x^i = (x_1^i, x_2^i, \dots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$, $i = 1, \dots, n$, is the i-th input vector of dimension m and y_i is the label corresponding to the i-th input vector x^i. Moreover, since we are describing a classification approach, we know that this implies a fundamental condition on labels which must assume discrete values. Hence, for the sake of clarity we consider labels $y_i \in \{-1, +1\}$, $i = 1, \dots, n$, i.e., a binary classification problem.

Given a generic input vector $x^i = (x_1^i, x_2^i, \dots, x_m^i)^{\mathrm{T}}$, linear classification methods tackle the problem of finding a decision making function f of x^i as a linear combination of input features together with a weight vector w. Namely, we have that

$$\hat{y}_i = f\big(w^{\mathrm{T}} x^i + \theta\big) = f\Big(\sum_{j=1}^{m} w_j x_j^i + \theta\Big) \tag{2.2}$$

with $w = (w_1, w_2, \dots, w_m)^{\mathrm{T}}$ weight vector and $\theta \in \mathbb{R}$ threshold value learned during the algorithm with respect to the given labeled dataset. As a direct consequence of Equation (2.2), we have that a linear classifier tackles the problem of separating data with the use of a straight line, for one dimensional cases, a plane in the event of dealing with bi-dimensional training dataset or a hyperplane for higher dimensional scenarios. Specifically, let us consider a linearly separable set of data such that points belonging to different classes are located onto different sections of space. Then, we can assume that there exist a vector w and a scalar θ such that

$$\begin{aligned} w^{\mathrm{T}} x^i &\geq \theta, \quad \text{if} \quad y_i = +1 \\ w^{\mathrm{T}} x^i &< \theta, \quad \text{if} \quad y_i = -1. \end{aligned} \tag{2.3}$$

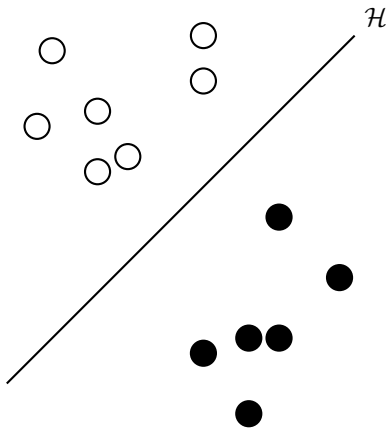


Figure 2.1: Black dots and black empty dots belong to two linearly separable different classes. Plane $\mathcal{H}$ correctly classifies the given dataset.

If Equations (2.3) are verified, vector w and scalar θ define a hyperplane separating the two classes of data points. Such hyperplane is known as *separating hyperplane* and is a key notion in linear classification techniques.

Figure 2.1 shows a basic linearly separable binary classification example, where data belonging to two linearly separable classes are separated through the use of a separating hyperplane.

2.1.1.1 Perceptron

The Perceptron algorithm was first proposed in the 50s by Frank Rosenblatt [Rosenblatt, 1957], relying its basic mechanism on the biological physical structure of the human neuron. Let us consider $\mathcal{D}$ an n-dimensional training set as defined in (2.1), with training inputs $x^i = (x_1^i, \dots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ and corresponding labels $y_i \in \{-1, +1\}$, $i = 1, \dots, n$.

The Perceptron is a classical binary classifier algorithm, based on the notion of separating hyperplane, namely the decision boundary dividing the two considered classes is a straight line. Specifically, the Perceptron algorithm uses a function f, called *activation function*, from the input space to the output space. Given a training data x^i, the function f is defined by the Heaviside step function as follows

$$f(x^i) = \begin{cases} 1, & \text{if } w^{\mathrm{T}} x^i + \theta \geq 0 \\ 0, & \text{otherwise,} \end{cases} \tag{2.4}$$

with $\theta \in \mathbb{R}$ threshold value. Equation (2.4) is also called the Perceptron classification rule. Note that, condition $w^{\mathrm{T}} x^i \geq 0$ may be also written as $\tilde{w}^{\mathrm{T}} \tilde{x}^i \geq 0$ where $\tilde{w} =$

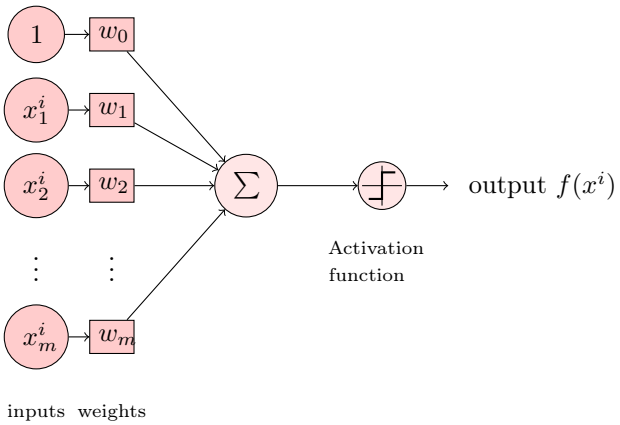


Figure 2.2: A classical Perceptron structure. Given a generic input point $x^i = (x_1^i, \ldots, x_m^i)^{\mathrm{T}}$ and a weight vector $w = (w_0, w_1, \ldots, w_m)^{\mathrm{T}}$, a linear combination of given input x^i and weight vector w is performed. After that, a specific activation function is used to obtain the computed output.

$(w_0, w_1, \ldots, w_m)^{\mathrm{T}}$ and $\tilde{x}^i = (x_o^i, x_1^i, \ldots, x_m^i)^{\mathrm{T}}$, with $w_0 = \theta$ and $x_0^i = 1$ for each $i = 1, \ldots, n$. In this section we will utilize this second formalism but, for the sake of simplicity, we will continue to use neat notations x and w.

Figure 2.2 shows the basic Perceptron framework for a generic input $x^i = (x_1^i, \ldots, x_m^i)^{\mathrm{T}}$ with output value $f(x^i) = \text{sign}\,(w^{\mathrm{T}} x^i)$. If $y_i \neq f(x^i)$, namely the computed output does not match with the corresponding label, the weight vector w is updated according to the following rule

$$w = w + \eta y_i x^i,$$

where η is the learning rate of the Perceptron, lying within the range 0 to 1, with highly volatile changes of weight vector w for larger choices of η.

The Perceptron learning algorithm is described in Algorithm 1.

Algorithm 1: Perceptron learning algorithm.

Data: A set of training data $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$, a learning rate η
Result: A separating hyperplane coefficient w^*
Initialize the weight vector $w = 0$;
repeat
  Choose a training pair (x^i, y_i);
  Compute output value $f(x^i) = \text{sign}\,(w^{\mathrm{T}} x^i)$;
  if $y_i \neq f(x^i)$ **then**
    $w = w + \eta y_i x^i$;
  end
until *convergence*;

As shown in Algorithm 1, the Perceptron learning algorithm considers one example at the time, instantly applying the weight vector w and, in case the prediction is not correct, i.e., $y_i \neq f(x^i)$, consequently updating it. In literature, applying the learning algorithm to every training example is called an *epoch*. At the end of each epoch, a weight vector is selected and, if it checks the considered convergence rule, the algorithm terminates; otherwise, a new epoch may begin.

Note that, by construction the convergence of the Perceptron model described in Algorithm 1 only occurs in cases in which the two classes are linearly separable. Specifically, if a linearly separable training set is considered, the Perceptron learning algorithm ensures that a final weight vector w^*, corresponding to a correct separating hyperplane, will be found after a finite number of epochs.

Before proving it, some assumptions are needed. First, since we are considering a finite linearly separable dataset $\mathcal{D}$, then there exists some $w^* \in \mathbb{R}^m$ correctly separating the given dataset. Moreover, we assume that there exists some $\gamma \in \mathbb{R}$ such that, for every $i \in \{1, 2, \ldots, n\}$, we have that

$$y_i(w^{\mathrm{T}} x^i) \geq \gamma.$$

Finally, we can assume without loss of generality that there exists $R \in \mathbb{R}$ such that

$$\|x^i\| \leq R,$$

for all $i \in \{1, 2, \ldots, n\}$.
Now we are ready to prove the convergence of the Perceptron algorithm.

Theorem 9. ***Perceptron convergence.***
Let

$$\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$$

be a linearly separable training dataset with training inputs $x^i = (x_1^i, \ldots, x_m^i)^T \in \mathbb{R}^m$ *and corresponding labels* $y_i \in \{-1, +1\}$, $i = 1, \ldots, n$. *Then, the Perceptron learning algorithm makes at most* $\frac{R^2}{\gamma^2}$ *updates before converging to a separating hyperplane with corresponding weight vector* w^*.

Proof. From Algorithm 1, if the Perceptron learning algorithm terminates returning a weight vector, then the corresponding separating hyperplane correctly divides points belonging to different classes. Hence, it is sufficient to show that the algorithm ends after at most $\frac{R^2}{\gamma^2}$ iterations.

Let us consider w^k as the weight vector computed after the k-th iteration on a incorrectly classified point $\{x^j, y_j\}$, namely

$$w^k = w^{k-1} + y_j x^j,$$

with learning rate $\eta = 1$. Moreover, let us assume $w^0 = 0$. Then, we have

$$\begin{aligned} {w^k}^{\mathrm{T}} w^* &= (w^{k-1} + y_j x^j)^{\mathrm{T}} w^* \\ &= {w^{k-1}}^{\mathrm{T}} w^* + y_j {x^j}^{\mathrm{T}} w^* \\ &> {w^{k-1}}^{\mathrm{T}} w^* + \gamma. \end{aligned}$$

By induction, since $w^0 = 0$, we have that

$$ {w^k}^{\mathrm{T}} w^* > k\gamma. \tag{2.5}$$

Moreover,

$$\begin{aligned} \|w^k\|^2 = {w^k}^{\mathrm{T}} w^k &= (w^{k-1} + y_j x^j)^{\mathrm{T}} (w^{k-1} + y_j x^j) \\ &= \|w^{k-1}\|^2 + 2y_j {w^{k-1}}^{\mathrm{T}} x^j + \|x^j\|^2 \\ &\leq \|w^{k-1}\|^2 + \|x^j\|^2 \\ &\leq \|w^{k-1}\|^2 + R^2, \end{aligned}$$

where we have used the fact that $y_j {w^{k-1}}^{\mathrm{T}} x^j \leq 0$. Again, since $w^0 = 0$, applying the formula above, we obtain that

$$ \|w^k\|^2 \leq kR^2. \tag{2.6}$$

Finally, combining Equations (2.5) and (2.6) we have

$$ k^2\gamma^2 \leq ({w^k}^{\mathrm{T}} w^*)^2 \leq \|w^k\|^2 \|w^*\|^2 \leq kR^2 \|w^*\|^2, $$

which implies

$$ k \leq \left(\frac{R}{\gamma}\right)^2 \|w^*\|^2. $$

Therefore, the Perceptron learning algorithm will converge after a finite number of iterations. □

In the Perceptron training phase, the loss function used is

$$ L(x, y, w) = \max\,(0, -y w^{\mathrm{T}} x). \tag{2.7}$$

Note that, we have assumed that, if w^* correctly separates the given data, then $y_i {w^*}^{\mathrm{T}} x^i \geq \gamma$ for some $\gamma \in \mathbb{R}$. Therefore, for every input x^i with corresponding label $y_i = +1$, we have that ${w^*}^{\mathrm{T}} x^i \geq \gamma$, while for those inputs x^i with negative label $y_i = -1$, then ${w^*}^{\mathrm{T}} x^i \leq \gamma$. Now, let us suppose that during epoch k some training errors occur. Then, we have the following scenario. If a data point x^i is correctly classified, then $y_i w^{\mathrm{T}} x^i \geq 0$. Otherwise, for incorrectly classified points x^i, we have that $y_i w^{\mathrm{T}} x^i < 0$. Namely the loss function described in Equation (2.7) is 0 for correctly classified points. Otherwise, if a point x^i is wrongly classified, namely it lies in the incorrect side of the considered space, then the corresponding penalty is given by the value $-y_i w^{\mathrm{T}} x^i$.

Assume now that the dataset $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$ is not linearly separable but we still wish to find a separating hyperplane with the use of a a classical Perceptron algorithm with the smallest number of incorrectly classified data. To do so, we have to consider a squared loss function. As before, let

$$\hat{y}_i = f(x^i)$$

be the computed output and y_i the actual label and let

$$e_j = \hat{y}_j - y_j$$

be the error corresponding to the j-th component of the training set $\mathcal{D}$. Then, the squared loss function is given by

$$\mathcal{E}(w) = \frac{1}{2}\sum_{i=1}^{n} e_i^2 = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2.$$

We can use the squared loss function to calculate the optimal weight vector w^*, computing the gradient of $\mathcal{E}(w)$ with respect to vector w, i.e., the derivative of the error function in the direction of the weight vector. Specifically, once the gradient of the error

$$\nabla_w \mathcal{E} = \frac{\partial \mathcal{E}}{\partial w}$$

is computed, vector w is updated using the following rule

$$w = w - \eta \nabla_w \mathcal{E}.$$

Algorithm 2 schematizes the explained learning rule.

Algorithm 2: The Perceptron learning algorithm with the use of gradient descent rule. Once a weight vector is given, the corresponding output is calculated for every input data. Then the gradient of the squared loss over the entire dataset is computed.

Data: A set of training data $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$, a learning rate η
Result: A separating hyperplane coefficient w^*
Initialize the weight vector $w = 0$;
repeat
  Compute output value $f(x^i) = \text{sign}\ (w^{\mathrm{T}} x^i)$ for every given input;
  $w = w - \eta \nabla_w \mathcal{E}$
until *convergence*;

2.1.1.2 Support Vector Machine

A Support Vector Machine algorithm was initially proposed in the first half of the sixties by Vladimir Vapnik and Alexey Chervonenkis [Vapnik and A. Y. Chervonenkis,

1964] for classifying linearly separable set of data, and, later on, broaden for the non-linear case by Vapnik himself and others. Specifically, in 1992 Vapnik together with Bernhard Boser and Isabelle Guyon [Boser et al., 1992] suggested an innovative approach for a non-linear classifier making use of kernel functions to extend the initial proposed method to non-linearly separable set of data and, in 1995, Vapnik and Corinna Cortes published the current Support Vector Machine approach [Cortes and Vladimir Vapnik, 1995] by making use of soft margins for non separable dataset.

Let us consider the linearly two-class separable case. Let $\mathcal{D}$ be a generic training dataset as specified in (2.1) with $x^i = (x_1^i, x_2^i, \ldots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ and corresponding class label $y_i \in \{-1, +1\}$, for every $i = 1, \ldots, n$. As briefly described above, Support Vector Machine algorithm was initially proposed as a linear classification method and is based on the idea of classifying the given data with the use of a separating hyperplane. Specifically, Support Vector classifiers aim to tackle the problem by finding the hyperplane with maximum margin separating the data points x^i with corresponding class label $y_i = 1$ with respect to those having label y_i equal to -1.

A generic hyperplane in $\mathbb{R}^m$ may be written as

$$w^{\mathrm{T}} x + \theta = 0, \tag{2.8}$$

with $w \in \mathbb{R}^m$ vector orthogonal to the hyperplane and $\theta \in \mathbb{R}$ threshold value. If the dataset is linearly separable, it is possible to show that there exist w and θ such that

$$\begin{aligned} w^{\mathrm{T}} x^i + \theta \geq +1, \quad \text{if} \quad y_i = +1 \\ w^{\mathrm{T}} x^i + \theta \leq -1, \quad \text{if} \quad y_i = -1. \end{aligned} \tag{2.9}$$

Note that, (2.9) is equivalent to writing

$$y_i [w^{\mathrm{T}} x^i + \theta] \geq 1,$$

for all $i = 1, \ldots, n$.

With reference to Figure 2.3, the separating hyperplane is given by $w^{\mathrm{T}} x + \theta = 0$ defining two portions of space demarcated by equations $w^{\mathrm{T}} x + \theta \geq +1$ and $w^{\mathrm{T}} x + \theta \leq -1$. If a point x^i lies in the side of space defined by $w^{\mathrm{T}} x + \theta \geq +1$, it has corresponding label y_i equal to $+1$, similarly for data lying in the section of space delineated by $w^{\mathrm{T}} x + \theta \leq -1$, the matching label is equal to -1. Still with reference to Figure 2.3, the core concept of Support Vector Machines is to look at such two parallel hyperplanes, separating the given data with respect to their corresponding class labels. Specifically, Support Vector Machine algorithms look for such two hyperplanes such that the distance between them is maximized. Such distance is known as *margin* and plays a key role.

Intuitively, given a linearly separable two-class dataset, there exist infinitely many hyperplanes correctly separating data into the given classes. While other linear classification techniques do not specifically distinguish between these separating hyperplanes,

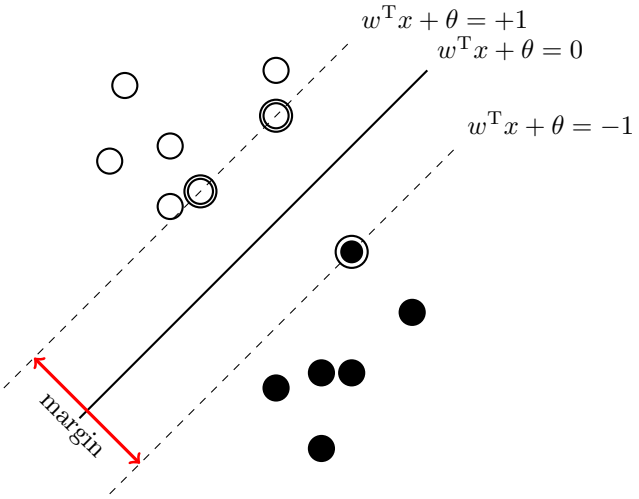


Figure 2.3: The hyperplane of equation $w^{\mathrm{T}}x + \theta = 0$ separates points belonging to the two given classes. If a point has output $y_i = +1$, then it lies on the portion of space boarded by the hyperplane of equation $w^{\mathrm{T}}x + \theta = +1$; on the contrary, if a data point has output $y_i = -1$, it is on the section of space given by the hyperplane defined by $w^{\mathrm{T}}x + \theta = -1$. The distance between such two hyperplanes takes the name of margin.

as we shall see, Support Vector Machines search for the specific hyperplane lying as far away as possible from the two given classes of data. Such distance between the two classes is known as margin of the classifier. Of course, the farther a data point is from the separating hyperplane, the more certain of the classification of such data we are.

More specifically, for a generic data point x^i with corresponding label y_i and a separating hyperplane with corresponding parameters (w, θ), the *functional margin* is defined as the quantity

$$\lambda_i = y_i(w^{\mathrm{T}}x^i + \theta). \tag{2.10}$$

It is easy to note that, the bigger the functional margin, the more correct and trusting the prediction of a certain data is. A large value for the functional margin requires a large positive value of $w^{\mathrm{T}}x^i + \theta$. Analogously, for negative values of y_i, the quantity $w^{\mathrm{T}}x^i + \theta$ has to be a large negative number for the functional margin to be large. Hence, when dealing with functional margin, we search for big values of it in order to be certain about the classifier. Anyway, it should be noted that, we can always re-scale the values of parameters (w, θ) to get large values in (2.10), since λ_i does not rely on the size of w and θ but only on its signs. For example, if we consider $(4w, 4\theta)$ instead of (w, θ), we obtain a functional margin $y_i((4w)^{\mathrm{T}}x^i + 4\theta) = 4y_i(w^{\mathrm{T}}x^i)$ which is four times bigger than (2.10) but with no actual change in the separating hyperplane. Hence, as a result, some constraint conditions on vector w have to be imposed.

To do so, we consider some geometrical notions. By construction, vector w is a

normal vector to the hyperplane defined by Equation (2.8) and $w/\|w\|$ is a unit vector pointing in the same direction as w. It is a fact that the distance between a point and a hyperplane is orthogonal to the hyperplane itself and, therefore, parallel to unit vector $w/\|w\|$. Let us consider a generic data point x^i with corresponding positive label y_i and let γ_i be the distance of such point with respect to the decision hyperplane, i.e., the *geometric margin* of point x^i. Moreover, let us take into account the closest point to x^i lying on the hyperplane. We have that such point is given by

$$x^i - \gamma_i \frac{w}{\|w\|}$$

and is located onto the hyperplane of Equation (2.8). Hence we have that

$$w^{\mathrm{T}} \left(x^i - \gamma_i \frac{w}{\|w\|} \right) = 0. \tag{2.11}$$

Solving Equation (2.11) for γ^i gives

$$\gamma_i = \frac{w^{\mathrm{T}} x^i + \theta}{\|w\|} = \frac{1}{\|w\|} w^{\mathrm{T}} x^i + \frac{\theta}{\|w\|}.$$

As a rule, for a general training data (x^i, y_i) with no knowledge on the sign of the label, the geometric margin is defined as

$$\gamma_i = y_i \left(\frac{1}{\|w\|} w^{\mathrm{T}} x^i + \frac{\theta}{\|w\|} \right). \tag{2.12}$$

Note that, if $\|w\| = 1$, Equations (2.10) and (2.12) are the same, namely the functional and the geometrical margins coincide. Moreover, as a direct consequence we have that the geometrical margin does not change when scaling of parameters (w, θ) happens.

Up to this point, we have considered the notions of functional margin and geometrical margin with respect to one single data point. Let us take a generic dataset given by $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$ as in (2.1). Based on Equation (2.10), the functional margin of $\mathcal{D}$ is defined as

$$\lambda = \min_{i=1,\ldots,n} \lambda_i.$$

Analogously, based on Equation (2.12), the geometrical margin of $\mathcal{D}$ is defined as

$$\gamma = \min_{i=1,\ldots,n} \gamma_i.$$

Note that, by construction we have that

$$\gamma = \frac{\lambda}{\|w\|}.$$

Since Support Vector Machines tackle the problem of finding a separating hyperplane by maximizing the margin between the two sets of classes, the optimal hyperplane may

be found solving the following optimization problem

$$\begin{aligned} \max_{\gamma,w,\theta} \ & \min_{i=1,\dots,n} \frac{w^{\mathrm{T}}x^i + \theta}{\|w\|} \\ \text{s.t.} \ & y_i[w^{\mathrm{T}}x^i + \theta] \geq 1, \ \ \forall i = 1,\dots,n. \end{aligned} \tag{2.13}$$

By definition of geometrical margin and structural margin of the dataset $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$ stated above, Problem (2.13) can be rewritten as

$$\begin{aligned} \max_{\lambda,w,\theta} \ & \frac{\lambda}{\|w\|} \\ \text{s.t.} \ & y_i[w^{\mathrm{T}}x^i + b] \geq 1, \ \ \forall i = 1,\dots,n. \end{aligned} \tag{2.14}$$

Since the functional margin is invariant with respect to re-scaling, we can set the functional margin $\lambda = 1$. In this way, solving the maximization Problem (2.13), i.e., Problem (2.14), is equivalent to solve the following minimization problem

$$\begin{aligned} \min_{w,\theta} \ & \frac{1}{2}\|w\|^2 \\ \text{s.t.} \ & y_i[w^{\mathrm{T}}x^i + \theta] \geq 1, \ \ \forall i = 1,\dots,n. \end{aligned} \tag{2.15}$$

Note that, Problem (2.15) is a convex quadratic optimization problem with linear constraints. Hence, it can be solved using basic quadratic programming (QP) programming libraries in an efficient way. Problem (2.15) is known in the scientific literature as *hard margin* Support Vector Machine since it takes into account specific dataset given by linearly separable two-classes data with no possibility of error occurring.

Let us now assume that the considered dataset $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$ is not linearly separable, namely the system of inequalities defined in (2.9) does not admit solutions. To extend Problem (2.15) to this scenario, let us consider slack variables ξ_i for all $i = 1,\dots,n$ in order to relax the constraint on the margin. Specifically, we now have:

$$\begin{aligned} y_i[w^{\mathrm{T}}x^i + \theta] &\geq 1 - \xi_i \\ \xi_i &\geq 0, \ \ \forall i = 1,\dots,n. \end{aligned}$$

Whenever $0 < \xi_i \leq 1$, the associated data point x^i is correctly classified and lies in the correct section of space; otherwise, when $\xi_i > 1$, the paired training data x^i enters the decision space of the opposite class and, therefore, is incorrectly classified. Hence, by definition we have that

$$\sum_{i=1}^{n} \xi_i$$

represents an upper-bound on the number of wrongly classified points in the training dataset $\mathcal{D}$. This justifies its inclusion in the objective function. Specifically, in the

event of employing slack variables $\xi_i, \forall i = 1, \dots, n$, Problem (2.15) becomes

$$\begin{aligned} \min_{w,\theta,\xi} \ & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \ & y_i[w^{\mathrm{T}}x^i + \theta] \geq 1 - \xi_i, \ \ \forall i = 1, \dots, n \\ & \xi_i \geq 0 \ \ \forall i = 1, \dots, n, \end{aligned} \tag{2.16}$$

with $C > 0$ parameter which represents some kind of degree of freedom, namely indicating how much a misclassified sample must pay for violating the margin constraint. Problem (2.16) takes the name *soft margin* Support Vector Machine.

2.1.1.3 Logistic Regression

Once again, let us consider the dataset defined in (2.1) with input vectors $x^i = (x_1^i, \dots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ and corresponding binary labels $y_i \in \{0, +1\}$ for every $i = 1, \dots, n$. Suppose we are trying to solve a classification problem, that is learning a classifier function predicting binary outcomes taking into account input-output correlation factors. Up to now, when a new input vector is considered, the classification methods we looked at, exactly estimate a precise output value belonging to the labels set considered. As part of statistical Machine Learning models, logistic regression algorithm, also known as logit regression, aims to obtain a probability output value that one of two outputs occurs, taking into consideration the relationship between input vectors and output values. The American scientist Joseph Berkson is widely recognised as the predominant figure developing logistic regression model [Berkson, 1944].

The logistic regression model has the goal of finding a conditional distribution function $P(y \mid x)$ computing the probability of a specific output y for a given input vector x.

Let $p(x) = P(y = 1)$. The logistic regression model is based on the key idea not to directly model probability function $p(x)$ but to use the *logit* function, also known as *log-odds* function, of $p(x)$, defined as

$$\text{logit}(p(x)) = \log \frac{p(x)}{1 - p(x)}. \tag{2.17}$$

Note that, by definition

$$\text{logit} : (0, 1) \longrightarrow \mathbb{R}$$

and is the inverse of the logistic curve of equation $f(x) = \frac{1}{1+e^{-x}}$. Moreover, the following results hold.

- $\text{logit}(0,5) = 0$;
- if $0 < p(x) < 0,5$ then $\text{logit}(p(x)) < 0$;
- if $0,5 < p(x) < 1$ then $\text{logit}(p(x)) > 0$.

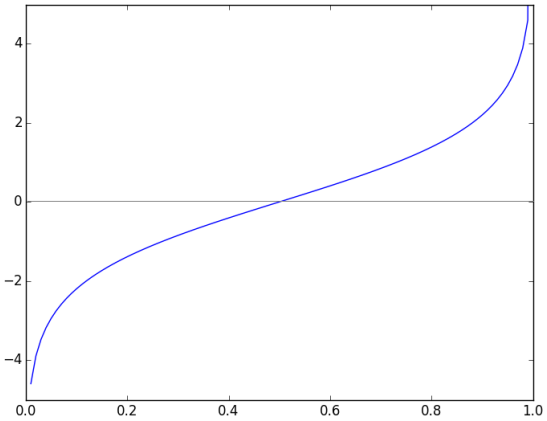


Figure 2.4: Logit function plot in its domain of definition (0,1).

Figure 2.4 shows the graph of logit function in the domain defined by values 0 and 1.

Logistic regression models the logit function of $p(x)$ as a linear function of x. Specifically, logistic regression method is formally given by

$$\log \frac{p(x)}{1-p(x)} = \theta + w^{\mathrm{T}} x. \tag{2.18}$$

Solving Equation (2.18) with respect to variable $p(x)$ we obtain

$$p(x) = \frac{e^{\theta + w^{\mathrm{T}} x}}{1 + e^{\theta + w^{\mathrm{T}} x}} = \frac{1}{1 + e^{-(\theta + w^{\mathrm{T}} x)}}. \tag{2.19}$$

Therefore, remembering that $p(x) = P(y = 1)$, if solving Equation (2.19) we get a value $p(x) \geq 0,5$, model predicts $y = 1$; otherwise, for values of $p(x) < 0,5$ we have $y = 0$. Therefore, whenever $\theta + w^{\mathrm{T}} x$ is non-negative, the related input point is classified as belonging to class $+1$, while for input vectors with corresponding negative values of $\theta + w^{\mathrm{T}} x$, we talk about points of class 0. The separating hyperplane of the two given classes is specified by equation

$$\theta + w^{\mathrm{T}} x = 0.$$

In addition to predicting the odds that an input point belongs to a specific class, logistic regression also defines class probabilities in Equation (2.19) depending on the distance of points from the separating hyperplane. Specifically, given an input vector x^i, if $\|\theta + w^{\mathrm{T}} x^i\|$ is large, then the probability class $p(x^i)$ will tend to the extreme points 0 and 1 more quickly, making the prediction more accurate and certain.

The logistic regression coefficients θ and w are usually computed through the use of the *maximum likelihood estimation* (MLE) method. From a broad point of view, let us consider a Bernoulli set of labels and let

$$p(\tilde{x}, \beta) = P(y = 1 \mid x = \tilde{x})$$

be a function defined by parameter β. Then, the conditional likelihood function is given by

$$\mathsf{L}(\beta) = \prod_{i=1}^{n} P(y = y_i \mid x = x^i) = \prod_{i=1}^{n} \Big[p(x^i, \beta)^{y_i} (1 - p(x^i, \beta))^{(1-y_i)} \Big]. \tag{2.20}$$

For a proper choice of function $p(x^i, \beta)$ depending on parameter β, the likelihood function (2.20) is a function of parameter vector β as well. Hence, we can get an estimate of parameter vector β maximizing the likelihood function. Since we are considering a two classes problem, we have probability p if $y_i = +1$ and probability $(1 - p)$ if $y_i = 0$. Hence, by Equation (2.20), we have that the likelihood function is given by

$$\mathsf{L}(\beta) = \prod_{i=1}^{n} \Big[p(x^i)^{y_i} (1 - p(x^i))^{(1-y_i)} \Big],$$

with

$$\beta = \begin{pmatrix} \theta \\ w \end{pmatrix}.$$

If we compute the natural logarithm of the likelihood function, known as the *log-likelihood*, we get

$$\begin{aligned} l(\theta, w) \coloneqq \log(\mathsf{L}(\beta)) &= \sum_{i=1}^{n} \Big[y_i \log p(x^i) + (1 - y_i) \log(1 - p(x^i)) \Big] \\ &= \sum_{i=1}^{n} \log(1 - p(x^i)) + \sum_{i=1}^{n} y_i \log \frac{p(x^i)}{1 - p(x^i)} \\ &= \sum_{i=1}^{n} \log \left(1 - \frac{e^{\theta + w^{\mathrm{T}} x^i}}{1 + e^{\theta + w^{\mathrm{T}} x^i}} \right) + \sum_{i=1}^{n} y_i (\theta + w^{\mathrm{T}} x^i) \\ &= \sum_{i=1}^{n} - \log(1 + e^{\theta + w^{\mathrm{T}} x^i}) + \sum_{i=1}^{n} y_i (\theta + w^{\mathrm{T}} x^i), \end{aligned}$$

where Equations (2.18) and (2.19) have been used. In this way we have obtained a value for the log-likelihood function $l(\theta, w)$.

To determine the best θ and w optimizing the likelihood function one option would be to compute the derivative with respect to the parameters and set it to zero. First of all, let us try to compute the partial derivative of $l(\theta, w)$ with respect to one component of w, e.g., w_j. We have

$$\begin{aligned} \frac{\partial l(\theta, w)}{w_j} &= - \sum_{i=1}^{n} \frac{1}{1 + e^{\theta + w^{\mathrm{T}} x^i}} e^{\theta + w^{\mathrm{T}} x^i} x_j^i + \sum_{i=1}^{n} y_i x_j^i \\ &= \sum_{i=1}^{n} (y_i - p(x^i)) x_j^i. \end{aligned}$$

Unfortunately, there is no close form solution for determining a stationary point. As a consequence, logistic regression method uses iterative approximation processes, e.g.,

Newton's method, to find the best fitting solution numerically, starting from a tentative initial choice of parameters θ and w.

2.1.2 Linear Regression

When we are asked to predict a continuous output value the corresponding learning problem is known as regression problem. Specifically, regression algorithms aim to find some kind of connection linking the input values with the corresponding outputs. When looking for the easiest possible connection between data, we talk about *linear regression*. Let us consider a generic dataset

$$\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n, \tag{2.21}$$

with $x^i = (x_1^i, \dots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ and corresponding continuous labels $y_i \in \mathbb{R}$. Suppose we are looking for a specific function estimating the existing relationship between given inputs x^i and related output y_i.

In linear regression the goal is to represent the approximation function as a linear combination of the input vectors, fitting the training data $\{(x^i, y_i)\}_{i=1}^n$ with the use of a line. In particular, for every given input vector $x^i = (x_1^i, \dots, x_m^i)^{\mathrm{T}}$, we look for a function f such that

$$f(x^i) = \theta + w_1 x_1^i + \cdots + w_m x_m^i, \tag{2.22}$$

with weight vector $w = (\theta, w_1, \dots, w_m)^{\mathrm{T}}$. To simplify Equation (2.22), we can introduce the notation

$$f(x^i) = \sum_{j=0}^{m} w_j x_j^i = w^{\mathrm{T}} x^i, \tag{2.23}$$

where we have introduced $x_0^i = 1$ for every $i = 1, \dots, n$. We can express the above formula in the matrix notation as

$$f = Xw \tag{2.24}$$

where

$$f = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix},$$

with $f_i = f(x^i)$ for every $i = 1, \dots, n$,

$$X = \begin{pmatrix} (x^1)^{\mathrm{T}} \\ (x^2)^{\mathrm{T}} \\ \vdots \\ (x^n)^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_m^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_m^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^n & x_2^n & \cdots & x_m^n \end{pmatrix},$$

and w is the weight vector as specified above. By Equation (2.24), it is clearly visible that linear regression algorithm entirely relies on the choice of vector w that better

represents the given dataset. The question is how to pick this vector that better describes the available data. One logic way is to consider the difference between the computed outputs $f_i = f(x^i)$ and the given labels y_i for every $i = 1, \ldots, n$. To do so, we consider the vector ϵ given by

$$\epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix},$$

where $\epsilon_i = f_i - y_i$ is the error term corresponding to i-th data point (x^i, y_i), namely, the difference between function f computed at point x^i with respect to the corresponding given label value y_i. Let $\mathcal{J}(w)$ be the cost function defied as

$$\begin{aligned} \mathcal{J}(w) &= \frac{1}{2}\sum_{i=1}^{n}(f(x^i) - y_i)^2 \\ &= \frac{1}{2}\sum_{i=1}^{n}(w^{\mathrm{T}}x^i - y_i)^2. \end{aligned} \tag{2.25}$$

Again, considering our dataset $\mathcal{D}$, one way to learn the best w is by trying to minimize the cost function $\mathcal{J}(w)$. Hence, we look for w^* such that

$$w^* \coloneqq \arg\min_w \mathcal{J}(w)$$

Such method is known as Least Mean Square (LMS) and may be solve through the use of several strategies.

First, let us consider the closed form solution. In order to compute the actual mathematical expression of w^*, let us write the cost function defined in Equation (2.25) using a matrix notation

$$\mathcal{J}(w) = \frac{1}{2}(Xw - Y)^{\mathrm{T}}(Xw - Y). \tag{2.26}$$

To find the optimum value w^* minimizing Equation (2.26), we need to compute the derivative of the cost function with respect to w. Specifically, we have

$$\begin{aligned} \frac{\partial \mathcal{J}}{\partial w} &= \frac{1}{2}\frac{\partial}{\partial w}(Xw - Y)^{\mathrm{T}}(Xw - Y) \\ &= \frac{1}{2}\frac{\partial}{\partial w}(w^{\mathrm{T}}X^{\mathrm{T}}Xw - w^{\mathrm{T}}X^{\mathrm{T}}Y - Y^{\mathrm{T}}Xw + Y^{\mathrm{T}}Y) \\ &= (X^{\mathrm{T}}Xw - X^{\mathrm{T}}Y). \end{aligned}$$

To find w^*, we set

$$\frac{\partial \mathcal{J}}{\partial w} = 0 \Longrightarrow X^{\mathrm{T}}Xw - X^{\mathrm{T}}Y = 0. \tag{2.27}$$

Hence, using Equation (2.27), we have that[1]

$$w^* = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}Y.$$

[1] We assume that $X^{\mathrm{T}}X$ is available.

When the closed form solution cannot be calculated or is too expensive to be computed, other indirect strategies need to be considered. Probably the best known is the gradient descent algorithm. First of all, let us consider an initial choice of w, say w^0. Gradient descent algorithm iteratively computes the following update rule

$$w^{t+1} \coloneqq w^t - \alpha \nabla \mathcal{J}(w^t),$$

for every $t \geq 0$. Hence, for every iteration specified by parameter t, the gradient of the cost function $\mathcal{J}(w)$ at w^t is computed. The new value w^{t+1} is updated with respect to direction opposite of the gradient. Moreover, parameter α is called learning rate of the algorithm. To have a more clear vision of the algorithm, using Equation (2.25) we can compute the gradient of the cost function $\mathcal{J}(w)$. First of all, we known that the weight vector w is a vector of $m+1$ elements of the form $w = [\theta, w_1, \ldots, w_m]^{\mathrm{T}}$. Hence the gradient $\nabla \mathcal{J}(w^t)$ takes the form

$$\nabla \mathcal{J}(w^t) = \left[\frac{\partial \mathcal{J}}{\partial \theta}, \frac{\partial \mathcal{J}}{\partial w_1}, \ldots, \frac{\partial \mathcal{J}}{\partial w_m} \right].$$

Therefore, for every $j = 0, \ldots, m$, the j-th component of the gradient vector takes the form

$$\begin{aligned}
\frac{\partial \mathcal{J}}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_{i=1}^{n} (w^{\mathrm{T}} x^i - y_i)^2 \\
&= \frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial w_j} (w^{\mathrm{T}} x^i - y_i)^2 \\
&= \frac{1}{2} \sum_{i=1}^{n} 2 (w^{\mathrm{T}} x^i - y_i) \frac{\partial}{\partial w_j} (w^{\mathrm{T}} x^i - y_i) \\
&= \frac{1}{2} \sum_{i=1}^{n} 2 (w^{\mathrm{T}} x^i - y_i)(x_j^i) \\
&= \sum_{i=1}^{n} (w^{\mathrm{T}} x^i - y_i)(x_j^i).
\end{aligned}$$

Specifically, each element of the gradient vector is equal to $\frac{\partial \mathcal{J}}{\partial w_j} = \sum_{i=1}^{n} (w^{\mathrm{T}} x^i - y_i)(x_j^i)$, $\forall j = 0, \ldots, m$. Namely, the partial derivative of the cost function with respect of the j-th component of weight vector w is computed as the sum of the error term $(w^{\mathrm{T}} x^i - y_i)$ multiplied by the j-th component of the corresponding input vector x^i. Algorithm 3 schematizes the proposed method.

Algorithm 3: Gradient descent for Least Mean Square in linear regression problems.

Data: A set of training data $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$, a learning rate α
Result: A weight vector w^*
Initialize the weight vector $w^0 = 0$;
repeat
  Compute $\nabla\mathcal{J}(w^t) = [\frac{\partial\mathcal{J}}{\partial\theta}, \frac{\partial\mathcal{J}}{\partial w_1}, \ldots, \frac{\partial\mathcal{J}}{\partial w_m}]$ gradient of cost function $\mathcal{J}(w)$ at w^t;
  Update vector w as $w^{t+1} \coloneqq w^t - \alpha\nabla\mathcal{J}(w^t)$
until *convergence*;

Note that, for small values of the learning rate α, Algorithm 3 is guaranteed to converge. As a matter of fact, even if gradient descent algorithm may be sensitive to the presence of local minima, the problem we aim to solve has only one global minimum value, since cost function $\mathcal{J}(w)$ is a convex quadratic function by construction. Hence, Algorithm 3 always converges to optimum value w^*.

In Algorithm 3 it should be underlined that the weight vector w is not updated before computing the error term $(w^{\mathrm{T}}x^i - y_i)$ for every training data $\{(x^i, y_i)\}_{i=1}^n$. An alternative solution would be not to wait until every error term is found but updating w as soon as a new error term is found. This method is known as stochastic gradient descent (i.e., incremental gradient descent) and is schematized in Algorithm 4.

Algorithm 4: Stochastic gradient descent for Least Mean Square in linear regression problems.

Data: A set of training data $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$, a learning rate α
Result: A weight vector w^*
Initialize the weight vector $w^0 = 0$;
repeat
  Consider every training pair (x^i, y_i);
  Update vector $w = (\theta, w_1, \ldots, w_m)^{\mathrm{T}}$ as
  $w_j^{t+1} \coloneqq w_j^t - \alpha(w^{\mathrm{T}}x^i - y_i)(x_j^i)$
until *convergence*;

Hence, while Algorithm 3 has to scan the entire training dataset before updating the weight vector w, stochastic gradient descent algorithm starts modifying w as soon as an error term occurs, making Algorithm 4 more convenient in terms of time and cost of operations.

2.2 Unsupervised Learning

2.2.1 Linear Discriminant Analysis

The term discriminant defines a function that, given an input vector, computes an output related to it. Specifically, the expression linear discriminant is used to describe a discriminant function using a linear combination of the input vectors. Linear Discriminant Analysis (LDA) is a linear learning method used for both classification problems and dimensionality reduction issues. In this context, linear discriminant analysis aims to map input data points into a lower dimension space, in order to obtain the best separable decision function between the given different classes.

As before, let $\mathcal{D} = \{(x^i, y_i)\}_{i=1}^n$ be a generic training set as defined in (2.1), with $x^i = (x_1^i, \dots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ and corresponding labels $y_i \in \{-1, +1\}$, for every $i = 1, \dots, n$. Linear discriminant analysis task searches for a function

$$f : \mathbb{R}^m \longrightarrow \{-1, +1\}$$

such that f is linear with respect to input vectors x^i, $i = 1, \dots, n$ and correctly classifies the given training point. Specifically, Linear Discriminant Analysis algorithm searches for a function f such that, for every x^i,

$$f(x^i) = w^{\mathrm{T}} x^i$$

with $w \in \mathbb{R}^m$ weight vector. Once a threshold value θ is chosen, if $f(x^i) \geq \theta$, then output $+1$ is assigned to input x^i. On the contrary, if $f(x^i) < \theta$, input x^i will be given output -1. Hence, vector w has a key role in Linear Discriminant Analysis. Based on the choice of w two possible outcomes are possible: we may obtain a good projection of the input vectors and, therefore, achieve a good separation between classes; in other cases, instead, we may obtain a complex projection, often leading to misclassifications. Figure 2.5 shows a two class dataset projected onto two different lines defined by weight vector w^1 and w^2.

Primarily, taking into account the two-class training dataset $\mathcal{D}$, Linear Discriminant Analysis assumes that each class may be represented with respect to its mean value and covariance function. Hence, let us denote C_1 and C_2 as the two class labels of dataset $\mathcal{D}$ and let n_1 and n_2 respectively describe the number of elements of each class. The mean value of each class is given by

$$\mu^1 = \frac{1}{n_1} \sum_{i \in C_1} x^i,$$
$$\mu^2 = \frac{1}{n_2} \sum_{i \in C_2} x^i.$$

A first idea would be to look for a weight vector w such that

$$\sum_{i=1}^m w_i = 1,$$

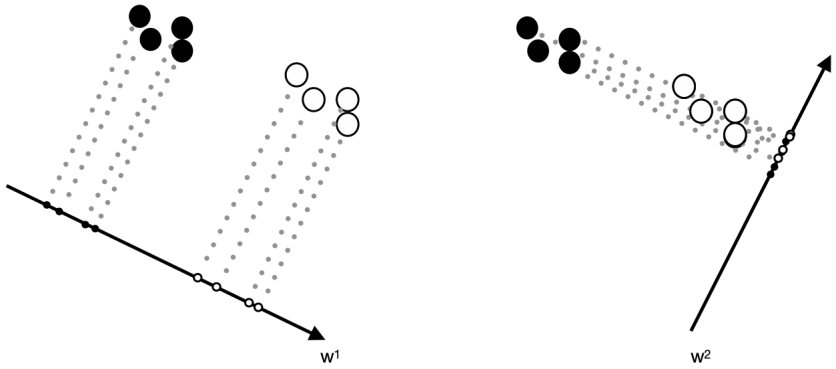


Figure 2.5: A generic two class dataset represented by the solid and empty dots. Weight vector w^1 leads to a projection of points that best separates the two given classes. On the contrary, vector w^2 generates a projection causing an inappropriate division of the considered classes.

maximizing the following quantity,

$$\frac{1}{n_1}\sum_{i\in C_1} w^{\mathrm{T}}x^i - \frac{1}{n_2}\sum_{i\in C_2} w^{\mathrm{T}}x^i = w^{\mathrm{T}}(\mu^1 - \mu^2),$$

where expressions $\frac{1}{n_1}\sum_{i\in C_1} w^{\mathrm{T}}x^i$ and $\frac{1}{n_2}\sum_{i\in C_2} w^{\mathrm{T}}x^i$ represents the projection of class mean μ^1 and μ^2 respectively. Such optimization leads to a non optimal solution when the class covariance functions assume different values with respect to each class. To solve this issue, Linear Discriminant Analysis algorithm optimizes the following function

$$J(w) = \frac{w^{\mathrm{T}}S_B w}{w^{\mathrm{T}}S_W w}, \tag{2.28}$$

where S_B is the between classes covariance matrix and S_W is the within classes covariance matrix, respectively defined as

$$S_B = (\mu^2 - \mu^1)^{\mathrm{T}}(\mu^2 - \mu^1),$$
$$S_W = \sum_j \sum_{i\in C_j} (x^i - \mu^j)(x^i - \mu^j)^{\mathrm{T}}.$$

Hence, Linear Discriminant Analysis aims to determine for a weight vector w maximizing the variance function of the projected centers S_B, namely with class means far apart, while minimizing the variance function within each class of the considered data points S_W at the same time. Computing the derivative of (2.28) and setting it to zero we obtain

$$\frac{\partial J}{\partial w} = \frac{S_B w(w^{\mathrm{T}}S_W w) - S_W w(w^{\mathrm{T}}S_B w)}{w^{\mathrm{T}}S_W w} = 0.$$

To solve the Linear Discriminant Analysis problem it is necessary to consider the following generalized eigenvalue problem

$$S_B w = J(w) S_W w.$$

Therefore, assuming matrix S_W is invertible, finding the maximum of (2.28) is equivalent to obtaining the largest eigenvectors corresponding to the largest eigenvalues of $S_W^{-1} S_B$ [Ordowski and Meyer, 2004].

From the construction of the problems, the Linear Discriminant Analysis algorithm may be associated with Support Vector Machine method. In fact, both models want to determine the optimal hyperplane separating the given data with respect to each class. However, differently from Support Vector Machine where there are no prerequisite on the considered dataset, Linear Discriminant Analysis technique starts with the assumption that every subset of the training set, corresponding to each class, is normally distributed. Specifically, Linear Discriminant Analysis needs to compute the mean value and the covariance matrix of every given class. Moreover, despite being a linear classifier, as stated above, Linear Discriminant Analysis algorithm is usually used for dimensionality reduction problems, in order to reduce the dimension of big input space with the expected desire of obtaining an easier to handle set of data.

Chapter 3

Kernel Learning Models

The key idea behind kernel learning is in the use of positive definite kernel functions. Let us consider a generic dataset $\mathcal{D}$ as specified in (2.1). Moreover, let X be the input set and Y the label set. As described in Chapter 1, given a Hilbert space $(\mathcal{H}, \langle \cdot, \cdot \rangle_{\mathcal{H}})$ and a map $\phi : X \longrightarrow \mathcal{H}$, a positive definite kernel is a function $k : X \times X \longrightarrow \mathbb{R}$ defined by

$$k(x^i, x^j) := \langle \phi(x^i), \phi(x^j) \rangle_{\mathcal{H}},$$

such that the corresponding Gram matrix $K := (K_{ij})$ where $K_{ij} = k(x^i, x^j)$ is symmetric and positive semi-definite for all $x^1, x^2, \ldots, x^n \in X$.

Differently from classical linear methods, where problems are solved seeking for a linear separation function in the original space, kernel learning algorithms all have in common the same basic focus: original input data is mapped onto a higher dimensional feature set where new coordinates are not computed but only the inner product of input points. Mapping points from a generic input space X to a Hilbert space $\mathcal{H}$ brings some well-known benefits. Specifically, the most immediate advantage in using kernel methods is the fact that it is possible to deal with non-linear set of data by mapping them into feature spaces, where they can be linearly separable. In particular, as already described beforehand, several kernel functions exist, producing a large set of possible choices with respect to the original dataset considered.

Moreover, kernel methods operate in an efficient way. In fact, let us consider a kernel function $k : X \times X \longrightarrow \mathbb{R}$. By construction, for every couple of input points x^i, x^j in the input space X, kernel k is defined as $k(x^i, x^j) = K_{ij} = \langle \phi(x^i), \phi(x^j) \rangle_{\mathcal{H}}$. As a direct consequence, it is not necessary to compute vectors $\phi(x^i)$ and $\phi(x^j)$ to obtain the corresponding kernel, but it is sufficient to calculate the value $\langle \phi(x^i), \phi(x^j) \rangle_{\mathcal{H}}$ given by the inner product of the specific input point defined with respect to the Hilbert space $\mathcal{H}$ (the kernel trick). Therefore, kernel learning represents an efficient approach: using kernel methods makes it possible to deal with non-linear dataset in a linear way mapping points to a higher dimension feature set with the addition

of not having to compute the actual coordinates of points onto such new space by the use of function $\phi(\cdot)$, but only their inner product. Moreover, kernel algorithms are a convenient approach: they allow to analyse and manipulate sets of non-linearly separable data through the use of linear functions, allowing a significant reduction of computational time.

As stated in Chapter 1, the concept of reproducing kernel was first introduced by Nachman Aronszajn in 1950 [Aronszajn, 1950]. Later in the nineties, an in-deep analysis on reproducing kernels was carried out. Specifically, Tomaso Poggio and Federico Girosi have made use of reproducing kernels in Radial Basis Function (RBF) networks, i.e., a specific neural network using radial basis function as activation function [Poggio and Girosi, 1990]. In the same year, Grace Wahba used reproducing kernels in classical approximation techniques for data analysis [Wahba, 1990]. In 1992 Bernhard Boser together with Isabelle Guyon and Vladimir Vapnik presented a maximum margin training algorithm trasforming the primal optimization problem into its dual form making use of a kernel representation [Boser et al., 1992]. From an unsupervised learning point of view, in 1998 Bernhard Schölkopf, Alexander Smola and Klaus-Robert Müller firstly used kernel functions for non-linear set of data in Principal Component Analysis [Schölkopf et al., 1998].

3.1 Supervised Learning

3.1.1 Support Vector Machine

Again, let us consider an n-dimensional generic dataset $\mathcal{D}$ as in (2.1) with input set X of elements $x^i = (x_1^i, \cdots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ for every $i = 1, \ldots, n$, and corresponding class labels set $Y = \{y_1, y_2, \ldots, y_n\}$, with $y_i \in \{-1, +1\}$, for every $i = 1, \ldots, n$. In Chapter 2, we introduced the following optimization problem corresponding to the soft margin Support Vector Machine classifier

$$\begin{aligned} \min_{w,\theta,\xi} \ & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \ & y_i[w^{\mathrm{T}}x^i + \theta] \geq 1 - \xi_i, \ \ \forall i = 1, \ldots, n \\ & \xi_i \geq 0 \ \ \forall i = 1, \ldots, n, \end{aligned} \tag{3.1}$$

with $C > 0$ training error parameter and ξ_i slack variables, for every input point $i = 1, \ldots, n$ with $\xi_i \geq 1$ corresponding to wrongly classified data points and $0 \leq \xi_i < 1$ whenever an input point x^i is correctly classified.

We can consider the Lagrangian function associated to Problem (3.1) given by

$$\mathcal{L}(w, \theta, \xi, \alpha, \beta) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left[y_i[w^{\mathrm{T}}x^i + \theta] - 1 + \xi_i\right] + \sum_{i=1}^{n}\beta_i\xi_i, \tag{3.2}$$

with $\alpha_i, \beta_i \in \mathbb{R}$ Lagrangian multipliers with $\alpha_i, \beta_i \geq 0$ for every $i = 1, \ldots, n$. In order to obtain the dual of Problem (3.1), we first compute the partial derivatives of the Lagrangian function $\mathcal{L}(w, \theta, \xi, \alpha, \beta)$ with respect to w and θ and set them to zero.

Specifically, we have

$$\frac{\partial \mathcal{L}}{\partial w}(w, \theta, \xi, \alpha, \beta) = w - \sum_{i=1}^{n} \alpha_i y_i x^i = 0,$$

implying that

$$w = \sum_{i=1}^{n} \alpha_i y_i x^i. \tag{3.3}$$

Moreover, we have that

$$\frac{\partial \mathcal{L}}{\partial \theta}(w, \theta, \xi, \alpha, \beta) = \sum_{i=1}^{n} \alpha_i y_i = 0. \tag{3.4}$$

Using Equation (3.3) into the Lagrangian function (3.2), we obtain the following expression

$$\mathcal{L}(w, \theta, \xi, \alpha, \beta) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {x^i}^{\mathrm{T}} x^j - \theta \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \xi_i (C - \alpha_i + \beta_i).$$

Now, considering Equation (3.4), we have that the Lagrangian function becomes

$$\mathcal{L}(w, \theta, \xi, \alpha, \beta) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {x^i}^{\mathrm{T}} x^j + \sum_{i=1}^{n} \xi_i (C - \alpha_i + \beta_i).$$

Moreover, we have that

$$\frac{\partial \mathcal{L}}{\partial \xi}(w, \theta, \xi, \alpha, \beta) = C - \alpha_i + \beta_i = 0. \tag{3.5}$$

Hence, the corresponding Wolfe-dual of Problem (3.1) is given by

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {x^i}^{\mathrm{T}} x^j \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \ i = 1, \ldots, n, \end{aligned} \tag{3.6}$$

where the constraints $0 \leq \alpha_i \leq C$ come from combining Lagrangian multiplier condition $\alpha_i \geq 0$, $\forall i = 1, \ldots, n$ with the result obtained using Equation (3.5). Now, since $\beta_i \geq 0$ for every $i = 1, \ldots, n$, this leads to constraint $\alpha_i \leq C$, $\forall i = 1, \ldots, n$. Note that Problem (3.6) is equivalent to the corresponding problem

$$\begin{aligned} \min_{\alpha} \quad \Gamma(\alpha) = & \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j {x^i}^{\mathrm{T}} x^j - \sum_{i=1}^{n} \alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \ i = 1, \ldots, n. \end{aligned} \tag{3.7}$$

Once Problem (3.7) is solved and a solution α^* is found minimizing the objective function $\Gamma(\alpha)$, the corresponding primal vector w^* may be computed using Equation (3.3) as

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x^i.$$

As a direct consequence, it is important to note that vector w^* is only dependent by the input vectors x^i corresponding to $\alpha_i^* > 0$. Such input points x^i are known as *support vectors* and are those closer to the maximum margin classifier. Support vectors are, in general, a small subset of the total number of input vectors, still fully controlling the margin geometrical features. Moreover, the input vectors x^i with with corresponding Lagrangian multipliers $0 < \alpha_i^* < C$ are called *free support vectors*, while those x^i with respective $\alpha_i^* = C$ are known as *bounded support vectors*, i.e. they lie inside the margin.

Once the primal solution w^* is found, we can compute the corresponding value for θ^*. Specifically, let us consider the complementary slackness conditions corresponding to Problem (3.1)

$$\alpha_i^* \Big(y_i \big[{w^*}^{\mathrm{T}} x^i + \theta^* \big] - 1 + \xi_i^* \Big) = 0 \tag{3.8a}$$

$$\beta_i^* \xi_i^* = \big(\alpha_i^* - C \big) \xi_i^* = 0 \tag{3.8b}$$

where in Equation (3.8b) we used condition (3.5). If we consider any free support vector x^i, i.e., an input vector with corresponding value of Lagrangian multiplier $0 < \alpha_i^* < C$, from Equation (3.8b) we have that $\xi_i^* = 0$. Using it in Equation (3.8a) we have that

$$y_i \big[{w^*}^{\mathrm{T}} x^i + \theta^* \big] - 1 = 0,$$

Hence, once the optimal w^* is found, we can easily obtain value θ^*.

For a new point $\bar{x}$ to be classified, the corresponding Support Vector Machine output is given by

$$f(\bar{x}) = \mathrm{sgn}({w^*}^{\mathrm{T}} \bar{x} + \theta^*) = \mathrm{sgn}\Big(\sum_{i=1}^{n} \alpha_i^* y_i {x^i}^{\mathrm{T}} \bar{x} + \theta^* \Big),$$

with

$$\mathrm{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

signum function.

Note that, the Wolfe-dual optimization Problem (3.7) may be entirely written in terms of the inner products of the input feature vectors. As a direct consequence, this makes Support Vector Machine optimization problem easily extendable to non-linear sets of data, exploiting kernel theory.

Let $\phi : X \longrightarrow \mathcal{H}$ be a feature map from the input set $X \subset \mathbb{R}^m$ to a feature space $\mathcal{H}$ of dimension greater than m. Hence, each input feature vector $x^i = (x_1^i, \cdots, x_m^i)^{\mathrm{T}}$ is mapped onto a higher dimensional feature space $\mathcal{H}$, resulting in vector $\phi(x^i)$. If we replace x^i with $\phi(x^i)$ for every $i = 1, \ldots, n$, Problem (3.7) becomes

$$\begin{aligned} \min_{\alpha} \quad & \Gamma(\alpha) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \langle \phi(x^i), \phi(x^j)\rangle_{\mathcal{H}} - \sum_{i=1}^{n}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \; i = 1, \ldots, n, \end{aligned} \tag{3.9}$$

Moreover, we have

$$w^* = \sum_{i=1}^{n}\alpha_i^* y_i \phi(x^i),$$

where w^* is the optimum solution of Problem (3.9). Using any input point such that $0 < \alpha_i^* < C$, we can compute the value of θ^* solving the following equation

$$y_i\left[{w^*}^{\mathrm{T}}\phi(x^i) + \theta^*\right] - 1 = 0.$$

Finally, for a new point $\bar{x}$ to be classified, the corresponding Support Vector Machine output is given by

$$f(\bar{x}) = \operatorname{sgn}({w^*}^{\mathrm{T}}\phi(\bar{x}) + \theta^*) = \operatorname{sgn}\left(\sum_{i=1}^{n}\alpha_i^* y_i \langle \phi(x^i), \phi(\bar{x})\rangle_{\mathcal{H}} + \theta^*\right).$$

Using the kernel definition proposed in Chapter 1, we consider a kernel function k defined as

$$k(x^i, x^j) = \langle \phi(x^i), \phi(x^j)\rangle_{\mathcal{H}},$$

with $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ inner product of the feature space $\mathcal{H}$. Problem (3.9) becomes

$$\begin{aligned} \min_{\alpha} \quad & \Gamma(\alpha) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j k(x^i, x^j) - \sum_{i=1}^{n}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \; i = 1, \ldots, n. \end{aligned} \tag{3.10}$$

The decision function described above can also be expressed in terms of the kernel function k as follows. For a novel point $\bar{x}$, the corresponding output is given by the following

$$f(\bar{x}) = \operatorname{sgn}\left(\sum_{i=1}^{n}\alpha_i^* y_i k_(x^i, \bar{x}) + \theta^*\right).$$

Note that, given any set of training data $\mathcal{D} = \{(x^i, y_i)_{i=1}^n\}$ of dimension n with $x^i = (x_1^i, \cdots, x_m^i)^{\mathrm{T}} \in X$ and class labels $Y = \{y_1, y_2, \ldots, y_n\}$, the use of a kernel function leads to an $n \times n$ kernel matrix K defined as

$$K = \big(k(x^i, x^j)\big)_{i,j=1}^n = \begin{pmatrix} k(x^1, x^1) & \cdots & k(x^1, x^n) \\ \vdots & \ldots & \vdots \\ k(x^n, x^1) & \cdots & k(x^n, x^n) \end{pmatrix},$$

symmetric and semi-positive definite by definition.

3.1.2 Gaussian Process

Gaussian Process (GP) is a supervised learning technique describing a distribution over functions. It is a kernel based algorithm which stands out from the others due to its probabilistic approach. Classical approaches make explicit use of existing relationship between input data and corresponding output, choosing between several possibilities, e.g., linear, polynomial or exponential. In these cases, where given formula are used to express such connection and where parameter size and features are set, we have parametric models. On the contrary, when such explicit hypothesis is not given, we have non-parametric models. Specifically, in this case we are not making any mathematical assumption either on the input-output relationship or on the weights, but we simply set our beliefs on the space of functions. In non-parametric models we do not design a finite set of parameters describing our dataset, leaving in this way a very high degree freedom to the possible choice of predictors. Gaussian Process belongs to the class of non-parametric model, placing a prior on the space of functions with no assumptions on the set of parameters.

From a theoretical point of view, a Gaussian Process is a stochastic process, namely a collection of random variables indexes by either time or space, where every distribution, characterised by being of finite dimension, is a multivariate Gaussian distribution, i.e., any finite choice of variables is normally distributed. If a Gaussian probability distribution is used to scalars or vectors in the form of random variables, a Gaussian Process describes the features of functions. Despite functions are defined on an infinite numbers of points, Gaussian Processes aim to deal with functions onto a specific finite set of points, that is the input dataset. Specifically, let us consider a generic dataset $\mathcal{D}$ with continuous labels as described in (2.21). Gaussian Processes describe a probability distribution over functions f such that $f(x^i)$, for every $x^i \in X$, has an n variate Gaussian distribution. Similarly to the classical Gaussian distribution, which is fully marked by a mean function and covariance function, a Gaussian Process is entirely defined by two key factors: its mean function m, defined as the expected value

$$E[f(x^i)]$$

of the entire set of variables $f(x^i)$, $\forall i = 1, \ldots, n$ of the problem, and its covariance function

$$k(x^i, x^j),$$

given by

$$k(x^i, x^j) = E[(f(x^i) - m(x^i))(f(x^j) - m(x^j))]$$

for every $i, j = 1, \ldots, n$.

Once again, it is important to note that, the use of a covariance function k on $\mathcal{D}$ directly produces an $n \times n$ covariance matrix K with elements given by $K_{ij} = k(x^i, x^j)$, $\forall i, j = 1, \ldots, n$.

In general, given a Gaussian Process $f(x)$ with corresponding mean function $m(x)$ and covariance function $k(x, x')$, for the function f the following notation is used

$$f(x) \sim \mathcal{G}(m(x), k(x, x')).$$

Note that, has described in-deep in Chapter 1, if the mean function $m(x)$ may assume any possible value, zero included, the covariance matrix K generated by the covariance function has to be positive semi-definite, namely, the corresponding covariance function $k(x, x')$ has to be positive definite. The use of Gaussian process in Machine Learning may be established in 1996 with the works of Carl Edward Rasmussen and Christopher Williams [Rasmussen, 2003] and Radford Neal [Neal, 2012].

As stated in Chapter 2, given an n-dimensional dataset $\mathcal{D}$ as in (2.21) with $X \subset \mathbb{R}^m$ input set and Y output set, in linear regression models for every input vector $x^i = (x_1^i, \ldots, x_m^i)^{\mathrm{T}} \in X$, we look for a function f such that

$$f(x^i) = \sum_{p=0}^{m} w_p x_p^i = w^{\mathrm{T}} x^i, \tag{3.11}$$

with $x_0^i = 1$ for every $i = 1, \ldots, n$ and weight vector $w = (w_0, w_1, \ldots, w_m)^{\mathrm{T}}$. Hence, introducing $\epsilon = (\epsilon_0, \epsilon_1, \ldots, \epsilon_n)^{\mathrm{T}}$ error term, we have the following relationship between $f(x^i)$ and corresponding label y_i

$$y_i = f(x^i) + \epsilon_i, \tag{3.12}$$

for every $i = 1, \ldots, n$. Namely, we assume that the computed output $f(x^i)$ differs from the given one y_i by an error term ϵ_i. We consider the error term ϵ having a Gaussian distribution, i.e., independent and identically distributed, with mean equal to zero and variance σ^2, namely

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2). \tag{3.13}$$

As a direct consequence of Equation (3.12), it follows that

$$y_i - f(x^i) \sim \mathcal{N}(0, \sigma_n^2).$$

The *likelihood*, i.e., the probability density of the outputs given the parameters, is given by

$$\begin{aligned} p(Y|X,w) =& \prod_{i=1}^{n} p(y_i|x_i,w) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{(y_i - w^{\mathrm{T}}x^i)^2}{2\sigma_n^2} \right) \\ =& \frac{1}{(2\pi\sigma_n^2)^{n/2}} \exp\left(-\frac{1}{2\sigma_n^2}|Y - X^{\mathrm{T}}w|^2 \right) = \mathcal{N}(X^{\mathrm{T}}w, \sigma_n^2 I), \end{aligned} \tag{3.14}$$

where the standard Gaussian distribution function for $p(y_i|x_i,w)$ is obtained combining the error term Gaussian distribution with the model defined by Equation (3.12). Therefore, the likelihood function

$$p(Y|X,w)$$

follows a Gaussian distribution with mean equal to $X^{\mathrm{T}}w$ and the variance of the error term ϵ as variance term.

As for every classical Machine Learning algorithm, the main goal of Gaussian Process method is, given a new input point $\bar{x}$, to determine an output value $f(\bar{x})$ based on a predictor obtained by the use of a training dataset. Let us assume that the vector w is normally distributed with zero mean function and covariance matrix equal to $S_p \in \mathbb{R}^{m\times m}$, that is

$$w \sim \mathcal{N}(0, S_p). \tag{3.15}$$

Equation (3.15) is known as *prior* with respect to the parameter vector w.
Using Bayes rule, that is

$$p(a|b) = \frac{p(b|a)p(a)}{p(b)},$$

we can obtain the *posterior* distribution of weights w as

$$p(w|Y,X) = \frac{p(Y|X,w)p(w)}{p(Y|X)}, \tag{3.16}$$

where the term $P(Y|X)$ is called *marginal likelihood* and is defined as

$$p(Y|X) = \int p(Y|X,w)p(w)\,dw. \tag{3.17}$$

Hence, once a new data $\bar{x}$ is considered, the *predictive distribution* for $\bar{f} = f(\bar{x})$ is given by

$$p(\bar{f}|\bar{x},X,y) = \int p(\bar{f}|\bar{x},w)P(w|X,Y)\,dw. \tag{3.18}$$

Equations (3.16) and (3.18) cannot, in general, be solved exactly and are often estimated with the use of approximation techniques, such as the maximum a posteriori (MAP) method. However, in the approach specified by Equations (3.12) and (3.13), also known

as Bayesian linear regression, the posterior and the predictive distribution may be computed [Rasmussen, 2003]. Specifically, we have

$$w|Y, X \sim \mathcal{N}\left(\frac{1}{\sigma_n^2} A^{-1} XY, A^{-1}\right), \tag{3.19a}$$

$$\bar{f}|\bar{x}, X, y \sim \mathcal{N}\left(\frac{1}{\sigma_n^2} \bar{x}^{-1} A^{-1} XY, \bar{x}^{-1} A^{-1} \bar{x}\right), \tag{3.19b}$$

where

$$A = \frac{1}{\sigma_n^2} XX^{\mathrm{T}} + S_p^{-1}. \tag{3.20}$$

Hence, from (3.19a) and (3.19b) both posterior and predictive distribution for Bayesian linear regression model follow a Gaussian distribution.

Again, let us consider a generic dataset $\mathcal{D}$ as specified in (2.21). Let ϕ be a feature map from the input set $X \subset \mathbb{R}^m$ to a feature space $\mathcal{H}$ of dimension m' greater than m. We can replace the original points x^i with vectors $\phi(x^i)$ for every $i = 1, \ldots, n$ and Problem (3.11) becomes

$$f(x^i) = \sum_{p=0}^{m'} w'_p \phi(x^i)_p = \langle w', \phi(x^i) \rangle_{\mathcal{H}}, \tag{3.21}$$

with novel weight vector w' of dimension m', i.e., the dimension of $\phi(x^i)$, $\forall i = 1, \ldots, n$. Analogously, considering the posterior distribution given in (3.19b), we obtain

$$\bar{f}|\bar{x}, X, y \sim \mathcal{N}\left(\frac{1}{\sigma_n^2} \phi(\bar{x})^{\mathrm{T}} A^{-1} \Phi Y, \phi(\bar{x})^{\mathrm{T}} A^{-1} \phi(\bar{x})\right), \tag{3.22}$$

where Φ is a matrix of dimension $m' \times n$ with columns given by vectors $\phi(x^i), \forall i = 1, \ldots, n$. Note that, (3.22), as well as (3.19b), contains the inverse of matrix A in both mean and covariance function. Hence, when the chosen feature space has dimension m' bigger than m, solving (3.22) would need to compute the inverse of a large matrix $m' \times m'$ A, leading to a poor cost-effective implementation. Now, remembering from (3.20) that matrix A was initially defined as $\frac{1}{\sigma_n^2} XX^{\mathrm{T}} + S_p^{-1}$, if we consider the feature map ϕ and the corresponding matrix Φ, we have that

$$A = \frac{1}{\sigma_n^2} \Phi\Phi^{\mathrm{T}} + S_p^{-1}.$$

Moreover, introducing $K = \Phi^{\mathrm{T}} S_p \Phi$ we have the following

$$\begin{aligned} AS_p\Phi =& \left(\frac{1}{\sigma_n^2} \Phi\Phi^{\mathrm{T}} + S_p^{-1}\right) S_p\Phi = \\ =& \frac{1}{\sigma_n^2} \Phi\Phi^{\mathrm{T}} S_p\Phi + \Phi I = \\ =& \frac{1}{\sigma_n^2} \Phi\left(\Phi^{\mathrm{T}} S_p\Phi + \sigma_n^2 I\right) = \\ =& \frac{1}{\sigma_n^2} \Phi\left(K + \sigma_n^2 I\right). \end{aligned} \tag{3.23}$$

Now, Equation (3.23) reduces to

$$S_p\Phi\big(K+\sigma_n^2 I\big)^{-1} = \frac{1}{\sigma_n^2}A^{-1}\Phi, \tag{3.24}$$

multiplying (3.23) by A^{-1} on the left and by $(K+\sigma_n^2 I)^{-1}$ on the right. Hence, the mean function of (3.22) becomes

$$\frac{1}{\sigma_n^2}\phi(\bar{x})^{\mathrm{T}}A^{-1}\Phi Y = \phi(\bar{x})^{\mathrm{T}}S_p\Phi\big(K+\sigma_n^2 I\big)^{-1}Y. \tag{3.25}$$

Now, let us consider the covariance function of (3.22) defined as

$$\phi(\bar{x})^{\mathrm{T}}A^{-1}\phi(\bar{x}).$$

Using the matrix inversion lemma [Henderson and Searle, 1981], we have that

$$\begin{aligned} A^{-1} &= \big(S_p^{-1}+\Phi\sigma_n^{-2}I\Phi\big)^{-1} = \\ &= S_p - S_p\Phi\big(\sigma_n^2 I+\Phi^{\mathrm{T}}S_p\Phi\big)^{-1}\Phi^{\mathrm{T}}S_p = \\ &= S_p - S_p\Phi\big(\sigma_n^2 I+K\big)^{-1}\Phi^{\mathrm{T}}S_p. \end{aligned} \tag{3.26}$$

Hence, the covariance function expression becomes

$$\phi(\bar{x})^{\mathrm{T}}A^{-1}\phi(\bar{x}) = \phi(\bar{x})^{\mathrm{T}}S_p\phi(\bar{x}) - \phi(\bar{x})^{\mathrm{T}}S_p\Phi\big(K+\sigma_n^2 I\big)^{-1}\Phi^{\mathrm{T}}S_p\phi(\bar{x}).$$

Therefore, Equation (3.22) is equivalent to

$$\begin{aligned} \bar{f}|\bar{x},X,y \sim \mathcal{N}\Big(&\phi(\bar{x})^{\mathrm{T}}S_p\Phi\big(K+\sigma_n^2 I\big)^{-1}Y, \\ &\phi(\bar{x})^{\mathrm{T}}S_p\phi(\bar{x}) - \phi(\bar{x})^{\mathrm{T}}S_p\Phi\big(K+\sigma_n^2 I\big)^{-1}\Phi^{\mathrm{T}}S_p\phi(\bar{x})\Big). \end{aligned} \tag{3.27}$$

Note that, Equation (3.27) only requires the computation of the inverse of the matrix $(K+\sigma_n^2 I)$ with $K=\Phi^{\mathrm{T}}S_p\Phi \in \mathbb{R}^{n\times n}$, where n is the number of observations of the given dataset $\mathcal{D}$. Hence, compared to Equation (3.22) where the inverse of the $m'\times m'$ matrix A needs to be computed, deriving the inverse is advantageous when $m'>n$. Moreover, it should be underlined that Equation (3.27) includes feature matrix Φ only in relationship with S_p, namely as products equal to $\phi(\bar{x})^{\mathrm{T}}S_p\Phi$, $\Phi^{\mathrm{T}}S_p\phi(\bar{x})$ and $\phi(\bar{x})^{\mathrm{T}}S_p\phi(\bar{x})$.

Let now

$$k(x^i,x^j) \coloneqq \phi(x^i)^{\mathrm{T}}S_p\phi(x^j).$$

Remembering that the covariance matrix S_p is positive semi-definite by definition, we can write

$$k(x^i,x^j) = \langle\psi(x^i),\psi(x^j)\rangle,$$

with $\psi(x) \coloneqq S_p^{1/2}\phi(x)$. Thus, substituting k in Equation (3.27), also in this case we obtain a formulation purely containing inner products with respect to the feature space.

3.2 Unsupervised Learning

3.2.1 Principal Component Analysis

Principal Component Analysis (PCA) is one of the most common and used dimensionality reduction techniques. Hence, given a generic large dataset given by a set of interconnected variables, Principal Component Analysis aims to determine the existing variable connections in order to reduce the number of attributes of the given dataset to a smaller number of unconnected variables, still containing the key information of the original set of data [Jolliffe, 2011]. As a direct consequence, Principal Component Analysis is mainly used for two specific purposes:

- To reduce the number of variables by finding those which are linearly depended with each other;
- To find the variables that are the most representative of the given set of data.

To do so, principal component analysis focuses on the concept of variance.

Let us consider a generic dataset of dimension n,

$$X = \{x^i\}_{i=1}^n, \tag{3.28}$$

with elements $x^i = (x_1^i, x_2^i, \dots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$, for every $i = 1, \dots, n$.

Firstly, as a pre-processing step, we center the given data on the mean of the given features. Namely given

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x^i = 0,$$

we set

$$x^i := x^i - \mu,$$

for every $i = 1, \dots, n$[1]. Doing so, we do not impose any change on the original data, but this will ensure that the novel dataset will be centered with respect to the new axis, also known as *principal components*.

The covariance matrix S is defined as an $n \times n$ matrix of elements s_{ij} equal to the covariance between inputs x^i and x^j for $i \neq j$ and the variance of the i-th element for $i = j$, namely for elements in the primal diagonal. Note that, when the covariance matrix is not known, a common estimator is given by the sample covariance matrix

$$Q = \frac{1}{n-1} \sum_{i=1}^{n} x^i {x^i}^{\mathrm{T}}, \tag{3.29}$$

$Q \in \mathbb{R}^{m \times m}$, with elements equal to the sample covariance, i.e., an estimate of the covariance done in a sample of the entire dataset, for $i \neq j$, and the sample variance of

[1] For the sake of simplicity, we decided to use the same symbol x^i for both the original and the centered data.

the observed values for $i = j$. Principal Component Analysis looks for a set of size $p \ll m$, of weights vectors $\alpha^k = (\alpha_1^k, \alpha_2^k, \ldots, \alpha_m^k)^{\mathrm{T}} \in \mathbb{R}^m$, $k = 1, \ldots, p$, mapping the original input dataset, namely each row of matrix X, into a Principal Component vector $z^i = (z_1^i, z_2^i, \ldots, z_p^i)^{\mathrm{T}}$. Specifically, we have that

$$z_k^i = {x^i}^{\mathrm{T}} \alpha^k,$$

for every $i = 1, \ldots, n$, $k = 1, \ldots, p$.

The first principal component z^1 is computed in such a way to have the largest variance, imposing as further constraint that the sum of squares of the components of weight α^1 is equal to 1, namely,

$${\alpha^1}^{\mathrm{T}} \alpha^1 = 1.$$

The second principal component z^2 is sought to be uncorrelated with respect to the first principal component z^1 and again to maximize the variance. This goes on until the final principal component z^p is found, with number p equivalent to the number of principal components established at the beginning of the algorithm.

Let us suppose that we are considering the $m \times m$ sample covariance matrix Q as described in (3.29). Starting from the first principal component, α^1 is chosen so that it maximizes the variance

$$\mathrm{Var}({x^i}^{\mathrm{T}} \alpha^1) = {\alpha^1}^{\mathrm{T}} Q \alpha^1$$

with respect to a normalization constraint on vector ${\alpha^1}^{\mathrm{T}} \alpha^1 = 1$.

Hence, we wish to solve the following problem

$$\begin{aligned} \max_{\alpha} \ & {\alpha^1}^{\mathrm{T}} Q \alpha^1 \\ \text{s.t.} \ & {\alpha^1}^{\mathrm{T}} \alpha^1 = 1. \end{aligned} \tag{3.30}$$

To find α^1 maximizing the variance and subject to the normalization constraint, we can consider the Lagrangian function corresponding to Problem (3.30) given by

$$\mathcal{L}(\alpha, \lambda) = {\alpha^1}^{\mathrm{T}} Q \alpha^1 - \lambda({\alpha^1}^{\mathrm{T}} \alpha^1 - 1),$$

with λ Lagrangian multiplier. We compute the partial derivative of $\mathcal{L}(\alpha, \lambda)$ with respect to λ and obtain

$$\frac{\partial \mathcal{L}}{\partial \lambda} = Q\alpha^1 - \lambda \alpha^1 = 0, \tag{3.31}$$

which is equivalent to the following expression

$$(Q - \lambda I)\alpha^1 = 0,$$

where I is the identity matrix of dimension $m \times m$. Therefore, we have that λ is an eigenvalue of matrix Q with corresponding eigenvector α^1. Moreover, note that the following condition holds

$${\alpha^1}^{\mathrm{T}} Q \alpha^1 = {\alpha^1}^{\mathrm{T}} \lambda \alpha^1 = \lambda {\alpha^1}^{\mathrm{T}} \alpha^1 = \lambda,$$

where we used the fact that $Q\alpha^1 = \lambda\alpha^1$ stated in Equation (3.31). Hence, to maximize

$$\alpha^{1^{\mathrm{T}}} Q\alpha^1,$$

is equivalent to determine the largest eigenvalue of the matrix Q.

Let us define $\lambda_1 := \alpha^{1^{\mathrm{T}}} Q\alpha^1$. As a direct consequence, the second principal component is given by $z_2^i = x^{i^{\mathrm{T}}}\alpha^2$, with $\lambda_2 = \alpha^{2^{\mathrm{T}}} Q\alpha^2$ second largest eigenvalue of Q. In this way, at the end of the algorithm we obtain the following p eigenvalues

$$\lambda_1 > \lambda_2 > \cdots > \lambda_p,$$

with corresponding eigenvectors $\alpha^1, \alpha^2, \ldots, \alpha^p$ satisfying the condition

$$Q\alpha_i = \lambda_i\alpha_i.$$

Algorithm 5 summarizes the Principal Component Analysis algorithm described above.

Algorithm 5: The Principal Component Analysis algorithm.

Data: A set of training data $X = \{(x^i)\}_{i=1}^n$ with $x^i \in \mathbb{R}^m$,
a dimension $p \ll m$
Result: A new set of variables of dimension p
Compute $\mu = \frac{1}{n}\sum_{i=1}^n x^i$;
Replace each x^i with $x^i - \mu$;
Compute the sample covariance matrix Q;
Find the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_p$ with corresponding eigenvectors
$\alpha_1, \alpha_2, \ldots, \alpha_p$ of matrix Q such that $\lambda_1 > \lambda_2 > \cdots > \lambda_p$;
For each $x^i \in X$ compute the corresponding principal component
$z^i = (z_1^i, z_2^i, \ldots, z_p^i)^{\mathrm{T}} \in \mathbb{R}^p$, with

$$z_k^i = x^{1^{\mathrm{T}}}\alpha^k,$$

for every $k = 1, 2, \ldots, p$.

Principal Component Analysis technique was first extended with the use of kernel functions during the nineties [Schölkopf et al., 1998]. Let ϕ be a feature map from the input set $X \subset \mathbb{R}^m$ to a feature space $\mathcal{H}$ of dimension m' greater than m and let $\phi(x^i)$, $\forall i = 1, \ldots, n$ be the projections of input data onto the feature space $\mathcal{H}$. Again, consider the projections to have zero mean. As a direct consequence, the matrix Q now becomes

$$Q = \frac{1}{n}\sum_{i=1}^n \phi(x^i)\phi(x^i)^{\mathrm{T}}. \tag{3.32}$$

From Equation (3.31) we have that

$$Q\alpha^k = \lambda_k\alpha^k, \quad \forall k = 1, \ldots, p, \tag{3.33}$$

with p fixed number of principal components. Using Equation (3.32) in (3.33) we obtain that

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x^i)(\phi(x^i)^{\mathrm{T}}\alpha^k) = \lambda_k\alpha^k, \tag{3.34}$$

that can be used to obtain the following expression for α^k

$$\alpha^k = \sum_{i=1}^{n} b_i^k\phi(x^i). \tag{3.35}$$

Now, combining Equations (3.34) and (3.35), we get

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x^i)\phi(x^i)^{\mathrm{T}}\sum_{j=1}^{n}b_j^k\phi(x^j) = \lambda_k\sum_{i=1}^{n}b_i^k\phi(x^i). \tag{3.36}$$

Define now

$$k(x^i, x^j) = \langle\phi(x^i), \phi(x^j)\rangle_{\mathcal{H}} = \phi(x^i)^{\mathrm{T}}\phi(x^j).$$

If we multiply Equation (3.36) from the left by $\phi(x^l)^{\mathrm{T}}$, we obtain

$$\frac{1}{n}\sum_{i=1}^{n}k(x^l, x^i)\sum_{j=1}^{n}b_j^k k(x^i, x^j) = \lambda_k\sum_{i=1}^{n}b_i^k k(x^l, x^i),$$

which can be written in the matrix form as

$$K^2b^k = \lambda_k nKb^k, \tag{3.37}$$

with K matrix with elements $K_{ij} \coloneqq k(x^i, x^j)$ and $b^k = (b_1^k, b_2^k, \ldots, b_n^k)^{\mathrm{T}} \in \mathbb{R}^n$, for every $k = 1, \ldots, p$. Moreover, as for classical Principal Component Analysis algorithm, we require $\alpha^k, \forall k = 1, \ldots, p$, to respect a normalization constraint, namely

$$\alpha^{k\,\mathrm{T}}\alpha^k = 1,$$

and hence

$$\sum_{i,j=1}^{n} b_i^k b_j^k\phi(x^i)^{\mathrm{T}}\phi(x^j) = b^{k\,\mathrm{T}}Kb^k = \lambda_k nb^{k\,\mathrm{T}}b^k = 1.$$

Therefore, the normalization constraint is equal to the following

$$||b^k|| = \sqrt{b^{k\,\mathrm{T}}b^k} = \frac{1}{\sqrt{\lambda_k n}}.$$

Using Equation (3.37), we can compute the vector b^k and the corresponding principal component following the classical scheme described before. Specifically, using Equation (3.35) we have that

$$z_k^i = \phi(x^i)^{\mathrm{T}}\alpha^k = \phi(x^i)^{\mathrm{T}}\sum_{j=1}^{n}b_j^k\phi(x^j),$$

for every $i = 1, \ldots, n$ and $k = 1, \ldots, p$.

Algorithm 6 summarizes the kernel Principal Component Analysis algorithm where we assume that the projection of the input data has zero mean.

Algorithm 6: The kernel Principal Component Analysis algorithm.

Data: A set of training data $X = \{(x^i)\}_{i=1}^n$ with $x^i \in \mathbb{R}^m$,
a dimension $p \ll m$,
a feature map ϕ and corresponding kernel matrix K
Result: A new set of variables of dimension p
Assume the projection of the input data has zero mean;
Compute the sample covariance matrix Q;
Find the p largest eigenvalues $\lambda_1, \dots, \lambda_p$ of the covariance matrix Q;
Find $b^k = (b_1^k, b_2^k, \dots, b_n^k)^{\mathrm{T}} \in \mathbb{R}^n$ solving

$$Kb_k = \lambda_k n b^k$$

for every $k = 1, \dots, p$;
For each $x^i \in X$ compute the corresponding principal component
$z^i = (z_1^i, z_2^i, \dots, z_p^i)^{\mathrm{T}} \in \mathbb{R}^p$, with

$$z_k^i = \phi(x^i)^{\mathrm{T}} \alpha^k = \phi(x^i)^{\mathrm{T}} \sum_{j=1}^n b_j^k \phi(x^j),$$

for every $k = 1, \dots, p$.

Note that, as for every other method described in this chapter, we never have to compute the value of the feature map ϕ in the input set X, but we only have to exactly compute the kernel values $K_{ij} = \phi(x^i)^{\mathrm{T}} \phi(x^j)$ for every $i, j = 1, \dots, n$, given by the inner products of the feature map computed with respect to every input data.

3.3 Multiple Kernel Learning

A fair choice of the employed kernel function is a critical issue in kernel learning algorithms. In fact, the decision of using one specific kernel function and related kernel parameters instead of others, is a key issue for the entire success or failure of an algorithm. Such decision is usually made through the use of a cross-validation method, i.e., an estimating technique splitting the dataset into training set and validation set and constructing the predicted model, using the training set, and evaluating it via the validation set. Cross-validation methods test the accuracy of implemented models with respect to unknown data, in order to obtain results which are not fully dependent and representative of the training data but may be successfully applied to novel inputs as well.

To address the problem of finding the "best" kernel function, instead of employing one single kernel function, Multiple Kernel Learning (MKL) algorithms tackle the problem of selecting kernel functions by using a combination of preset base kernels.

Let

$$X = \{x^i\}_{i=1}^n$$

be an n dimensional generic dataset with $x^i = (x_1^i, x_2^i, \ldots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ for every $i = 1, \ldots, n$. Furthermore, let

$$\{k_l(x^{i^l}, x^{j^l})\}_{l=1}^L$$

be a set of predefined kernel functions $k_l : \mathbb{R}^{m_l} \times \mathbb{R}^{m_l} \longrightarrow \mathbb{R}$, $l = 1, \ldots, L$, with $\mathbb{R}^{m_l}$ defined by the dimension l of the corresponding features x^{i^l}, x^{j^l} taken as inputs. Then, we may define

$$k_\gamma(x^i, x^j) \coloneqq f_\gamma\big(\{k_l(x^{i^l}, x^{j^l})\}_{l=1}^L\big), \tag{3.38}$$

as the combination of kernel functions k_l, $l = 1, \ldots, L$. Specifically, we have that $f_\gamma : \mathbb{R}^L \longrightarrow \mathbb{R}$ represents a combination function having as inputs all the preset kernel functions k_l, $l = 1, \ldots, L$ and with the corresponding output a kernel k_γ. Note that, nothing was said about the nature of function f_γ. Specifically, Multiple Kernel Learning might considered linear combination of kernel functions as well as non-linear ones, leading to linear or non-linear choice of function f_γ. Moreover, γ represents the combination parameters, namely the combination coefficients associated to every base kernel k_l, $l = 1, \ldots, L$, representing its weight with respect to the combination function. Just as kernel parameters, combinations parameters can be fine-tuned during the algorithm in order to obtain the best possible mix for the considered set of data. As a direct consequence, Multiple Kernel Learning main goal is to find the best combination coefficients and kernel weights so that the best possible classifier, i.e., a function mapping input data to corresponding outputs with largest accuracy value, is obtained.

As specified in [Gönen and Alpaydın, 2011], the main reasons to use Multiple Kernel Learning instead of classical kernel approaches may be outlined into two chief points. First, instead of using a single fixed structured kernel function, which may be specially suited for finding (unknown) specific data connections, multiple kernel approach allows to use different kernel functions together and, therefore, to look at different multi data bonds. In this way, when the given set of data is not entirely experienced, the user may operate using several kernels, optimizing its parameters based on data relationships and key characteristics, consequently obtaining a better classifier. In addiction, different kernels may use specific preset features of given data as inputs, allowing in this way to obtain an even more dataset specific function k_γ taking into consideration inter features connections other than inter data relationships.

With regard to function f_γ, several approaches exist. Specifically, the classical and most used way in which Multiple Kernel Learning is realized is through the use of a

linear combination of base kernel functions. In this way, Equation (3.38) becomes

$$k_\gamma(x^i, x^j) = f_\gamma\big(\{k_l(x^{i^l}, x^{j^l})\}_{l=1}^L\big) = \sum_{l=1}^{L} \gamma_l k_l(x^{i^l}, x^{j^l}), \tag{3.39}$$

with γ_l, $l = 1, \ldots, L$ combination coefficients defining the corresponding weights with respect to the associated base kernel. Usually, kernel weights are chosen such that

$$\gamma_l \geq 0, \quad \forall l = 1, \ldots, L$$

and

$$\sum_{l=1}^{L} \gamma_l = 1,$$

making (3.39) a convex combination. Since, by definition, a kernel function is expressed as

$$k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle_{\mathcal{H}},$$

where ϕ is a feature map, we can define

$$\phi_\gamma(x) = \begin{pmatrix} \sqrt{\gamma_1}\phi_1(x^1) \\ \sqrt{\gamma_2}\phi_2(x^2) \\ \vdots \\ \sqrt{\gamma_L}\phi_L(x^L) \end{pmatrix},$$

from which it follows that

$$\sum_{l=1}^{L} \gamma_l k_l(x^{i^l}, x^{j^l}) = \begin{pmatrix} \sqrt{\gamma_1}\phi_1(x^{i^1}) \\ \sqrt{\gamma_2}\phi_2(x^{i^2}) \\ \vdots \\ \sqrt{\gamma_L}\phi_L(x^{i^L}) \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \sqrt{\gamma_1}\phi_1(x^{j^1}) \\ \sqrt{\gamma_2}\phi_2(x^{j^2}) \\ \vdots \\ \sqrt{\gamma_L}\phi_L(x^{j^L}) \end{pmatrix} = \langle \phi_\gamma(x^i), \phi_\gamma(x^j) \rangle_{\mathcal{H}},$$

giving the structural dot product form we expect from a kernel function.

Moreover, the function f_γ may correspond to a non-linear combination of the base kernel functions, usually involving a much higher computational effort not always, however, leading to greater performance results [Cortes, Mohri, et al., 2009; Varma and Babu, 2009].

Regarding the learning approaches, several exploited methods exist which can be classified into the five following approaches [Gönen and Alpaydın, 2011].

- *Fixed rule* methods, where the multiple kernel function is defined at the beginning of the algorithm as a simple technique, for example sums or products of base kernels, and no combination parameter is required and, therefore, achieved during the algorithm. For example, following such method, [Kashima et al., 2009] makes use of pairwise kernels, i.e., kernels with the main goal of learning a model for pairs of input data;

· *Heuristic* methods, where the best combination of parameters is obtained through the use of some heuristic approach: parameters are individually defined relaying on single base kernel functions. Such individual kernels make either use of the kernel matrix or are tested on the training set, taking the achieved results as a measurement. Basing its algorithm on a heuristic algorithm to find the best fitting combination parameters, [Tanabe et al., 2008] obtains the following rule

$$\gamma_p = \frac{a_p - \theta}{\sum_{l=1}^{L}(a_l - \theta)},$$

with a_p accuracy value achieved making use of the single kernel k_p and θ threshold value;

· *Optimization* methods, where the best combination of parameters is obtained by solving an optimization problem. Such optimization may be incorporated in the original optimization problem, e.g., in the case of Support Vector Machines [Z. Chen, Li, and Wei, 2007], or can be carried out separately before the actual learning algorithm is performed;

· *Bayesian* methods, where combination parameters are obtained by interpreting them as random variables. Specifically, parameters are initialized through the use of prior functions. Afterwards, using information from the covariance matrix, combination parameters are eventually learned. For example, [Christoudias et al., 2009] uses a fixed rule kernel multiplied by a parametric kernel as a rule for the final multiple kernel formula;

· *Boosting* where the combination parameters are learnt through the use of boosting methods. Specifically, avoiding to use the often expensive optimization approaches, an iterative process is performed where a new base kernel is considered at every new iteration until some originally defined stop condition is achieved. As an example, [Wu et al., 2017] develops a boosting multiple kernel approach for regression problems by performing a gradient boosting method.

In particular, let us consider the multiple kernel binary classification problem with respect to the classical Support Vector Machine framework. Once again, let us consider a generic dataset $\mathcal{D}$ as in (2.1). We known that the primal soft margin Support Vector Machine classifier is given by

$$\begin{aligned} \min_{w,\theta,\xi} \ & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \ & y_i[w^{\mathrm{T}}x^i + \theta] \geq 1 - \xi_i, \ \ \forall i = 1,\dots,n \\ & \xi_i \geq 0 \ \ \forall i = 1,\dots,n, \end{aligned} \tag{3.40}$$

with $C > 0$ the training error parameter and ξ_i the slack variables corresponding to input points for every $i = 1,\dots,n$. Call that $\xi_i \geq 1$ corresponds to wrongly classified

data points x^i, while $0 \leq \xi_i < 1$ is associated to correctly classified input points x^i. Constructing the Lagrangian function $\mathcal{L}$ and setting the partial derivatives of $\mathcal{L}$ with respect to w, θ and ξ_i equal to zero, we obtain the dual expression described in Problem (3.7).

To obtain the multiple kernel Support Vector Machine formulation, we need to start from the decision function we aim to obtain. Suppose we are considering L base kernel functions given by $k_1(x, x'), k_2(x, x'), \ldots, k_L(x, x')$. As stated in [Zien and Ong, 2007], in a multiple kernel framework we are looking for a classification function of the form

$$f_{w,\theta,\gamma}(x) = \sum_{l=1}^{L} \gamma_l \langle w_l, \phi_l(x) \rangle_{\mathcal{H}} + \theta,$$

with ϕ_l associated to the l-th kernel function k_l. Therefore, Problem (3.40) becomes

$$\begin{aligned} \min_{w,\theta,\xi,\gamma} \quad & \frac{1}{2} \sum_{l=1}^{L} \gamma_l ||w_l||^2 + C \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & y_i \sum_{l=1}^{L} \gamma_l \langle w_l, \phi_l(x^i) \rangle_{\mathcal{H}} + y_i \theta \geq 1 - \xi_i, \ \ \forall i = 1, \ldots, n \\ & \xi_i \geq 0 \ \ \forall i = 1, \ldots, n, \\ & \sum_{l=1}^{L} \gamma_l = 1, \ \gamma_l \geq 0 \ \ \forall l = 1, \ldots, L. \end{aligned} \tag{3.41}$$

Note that, the formulation stated in Problem (3.41) is not in general convex because of the product term $\gamma_l ||w_l||^2$ of two of the variables to be optimized. To solve this issue, the following replacement is considered

$$v_l \coloneqq \gamma_l w_l.$$

This substitution leads to the convex minimization problem below in both γ_l and v_l variables.

$$\begin{aligned} \min_{v,\theta,\xi,\gamma} \quad & \frac{1}{2} \sum_{l=1}^{L} \frac{1}{\gamma_l} ||v_l||^2 + C \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & y_i \sum_{l=1}^{L} \langle v_l, \phi_l(x^i) \rangle_{\mathcal{H}} + y_i \theta \geq 1 - \xi_i, \ \ \forall i = 1, \ldots, n \\ & \xi_i \geq 0 \ \ \forall i = 1, \ldots, n, \\ & \sum_{l=1}^{L} \gamma_l = 1, \ \gamma_l \geq 0 \ \ \forall l = 1, \ldots, L. \end{aligned} \tag{3.42}$$

Note that, Problem (3.42) may be considered as a two-step optimization problem, with inner problem given by the optimization with respect to v, θ, ξ, and the outer given by the minimization problem with respect to γ. Following this observation, one way to

tackle Problem (3.42) consists in solving the inner minimization problem with respect to v, θ, ξ leaving vector γ fixed; then γ is updated using a descent direction algorithm this time fixing v, θ, ξ, in order to get the minimum of the considered objective function. For a more efficient approach guarantying the convergence of the algorithm, let us focus on the inner problem, namely

$$\begin{aligned} \min_{v,\theta,\xi} \quad & \frac{1}{2}\sum_{l=1}^{L}\frac{1}{\gamma_l}||v_l||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i\sum_{l=1}^{L}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} + y_i\theta \geq 1-\xi_i, \ \ \forall i = 1,\dots,n \\ & \xi_i \geq 0 \ \ \forall i = 1,\dots,n. \end{aligned} \tag{3.43}$$

The Lagrangian function of Problem (3.43) is given by

$$\mathcal{L}(v,\theta,\xi,\alpha,\nu) = \frac{1}{2}\sum_{l=1}^{L}\frac{1}{\gamma_l}||v_l||^2 + C\sum_{i=1}^{n}\xi_i + \sum_{i=1}^{n}\alpha_i\left[1-\xi_i - y_i\sum_{l=1}^{L}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} - y_i\theta\right] - \sum_{i=1}^{n}\nu_i\xi_i.$$

Setting the partial derivatives of $\mathcal{L}(v,\theta,\xi,\alpha,\nu)$ with respect to v, θ, and ξ to zero we obtain the following dual optimization problem

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \sum_{l=1}^{L}\gamma_l k_l(x^i, x^j) - \sum_{i=1}^{n}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \ \ \forall i = 1,\dots,n. \end{aligned} \tag{3.44}$$

Now, considering the outer minimization problem with respect to the linear combination parameter γ, the dual problem of (3.42) becomes

$$\begin{aligned} \min_{\gamma\in\mathbb{R}^L}\min_{\alpha} \quad & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \sum_{l=1}^{L}\gamma_l k_l(x^{i^l}, x^{j^l}) - \sum_{i=1}^{n}\alpha_i \\ \text{s.t.} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C, \ i = 1,\dots,n, \\ & \sum_{l=1}^{L}\gamma_l = 1, \\ & \gamma_l \geq 0, \ l = 1,\dots,L. \end{aligned} \tag{3.45}$$

A well-known and frequently used solution technique for solving Problem (3.45) consists in a two-step procedure. First the inner optimization problem is solved, leading to the minimization of the objective function given by

$$J(\gamma) \coloneqq \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \sum_{l=1}^{L}\gamma_l k_l(x^{i^l}, x^{j^l}) - \sum_{i=1}^{n}\alpha_i,$$

with respect to the constraints given by

$$\begin{aligned} &\sum_{i=1}^{n} \alpha_i y_i = 0, \\ &0 \leq \alpha_i \leq C,\ i = 1, \ldots, n. \end{aligned} \tag{3.46}$$

After that, the combination coefficients γ may be computed. Several approaches for finding parameter γ exist.

For example, in [Rakotomamonjy, F. Bach, et al., 2007] parameter γ is initially fixed and the classical Support Vector Machine problem (3.44) is solved assuming

$$k(x^i, x^j) = \sum_{l=1}^{L} \gamma_l k_l(x^{i^l}, x^{j^l})$$

as kernel function. Then the partial derivative of $J(\gamma)$ is computed as follows

$$\frac{\partial J(\gamma)}{\partial \gamma_m} = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k_m(x^{i^m}, x^{j^m}), \tag{3.47}$$

for every $m = 1, \ldots, L$. Hence, γ is updated using the gradient descent rule. Algorithm 7 schematizes the proposed approach.

Algorithm 7: Support Vector Machine multiple kernel approach described in [Rakotomamonjy, F. Bach, et al., 2007].

Data: A set of training data $X = \{(x^i)\}_{i=1}^n$ with $x^i \in \mathbb{R}^m$,
a set of training labels $Y = \{(y_i)\}_{i=1}^n$ with $y_i \in \{-1, +1\}$,
Result: A combination of the parameters γ_l, $l = 1, \ldots, L$ and Lagrangian multipliers α_i, $i = 1, \ldots, n$
Initialize $\gamma_l = 1/L$ for $l = 1, \ldots, L$;
while *stopping criteria is met* **do**
  Find the Lagrangian coefficients α_i, $i = 1, \ldots, n$ solving the inner classical Support Vector Machine problem with $k(x^i, x^j) = \sum_{l=1}^{L} \gamma_l k_l(x^{i^l}, x^{j^l})$ and compute the corresponding $J(\gamma)$;
  Compute the partial derivative $\frac{\partial J(\gamma)}{\partial \gamma_m}$ for every $m = 1, \ldots, L$;
  Compute descent direction $d = \nabla J(\gamma)$ and stepsize β;
  $\gamma \leftarrow \gamma + \beta d$
end

Another multiple kernel Support Vector Machine approach is proposed in [Rakotomamonjy, F. R. Bach, et al., 2008] under the name of *SimpleMKL*, where a reduced gradient algorithm [Luenberger, Ye, et al., 1984] is described. Just as in the earlier example, an iterative two step algorithm is proposed. As a first step, the minimization problem defined by (3.44) is considered, with fixed combination parameters γ_l,

$l = 1, \ldots, L$. Once the appropriate optimal Lagrangian multipliers α_i, $i = 1, \ldots, n$ are obtained with respect to the considered iteration, the outer convex optimization problem with respect to coefficients γ_l is considered. First, the partial derivative of $J(\gamma)$ with respect to γ_m, is given by Equation (3.47). Let γ_μ be a nonzero component in the vector γ. Then, the reduced gradient is equal to

$$\nabla_{\text{red}} J_m = \begin{cases} \dfrac{\partial J(\gamma)}{\partial \gamma_m} - \dfrac{\partial J(\gamma)}{\partial \gamma_\mu}, & \text{for } m \neq \mu \\ \sum_{m \neq \mu} \left(\dfrac{\partial J(\gamma)}{\partial \gamma_m} - \dfrac{\partial J(\gamma)}{\partial \gamma_\mu} \right), & \text{otherwise.} \end{cases}$$

Moreover, if there exist some indexes m such that the corresponding $\gamma_m = 0$ and $|\nabla_{\text{red}} J_m| > 0$, this would jeopardize the positivity constraint condition of combination parameters vector γ. In this case, the descent direction is automatically set to zero. Therefore, the descent direction d_m is given by

$$d_m = \begin{cases} 0 & \text{if } \gamma_m = 0 \text{ and } |\nabla_{\text{red}} J_m| > 0, \\ -\left(\dfrac{\partial J(\gamma)}{\partial \gamma_m} - \dfrac{\partial J(\gamma)}{\partial \gamma_\mu} \right) & \text{if } \gamma_m > 0 \text{ and } m \neq \mu, \\ -\sum_{m \neq \mu} \left(\dfrac{\partial J(\gamma)}{\partial \gamma_m} - \dfrac{\partial J(\gamma)}{\partial \gamma_\mu} \right) & \text{if } m = \mu. \end{cases} \tag{3.48}$$

At every new iteration, the parameter γ is updated using the vector d defined in (3.48). Algorithm 8 schematizes the proposed method.

Algorithm 8: SimpleMKL approach described in [Rakotomamonjy, F. R. Bach, et al., 2008].

Data: A set of training data $X = \{(x^i)\}_{i=1}^n$ with $x^i \in \mathbb{R}^m$, a set of training labels $Y = \{(y_i)\}_{i=1}^n$ with $y_i \in \{-1, +1\}$, stepsize β

Result: A combination of the parameters γ_l, $l = 1, \ldots, L$ and Lagrangian multipliers α_i, $i = 1, \ldots, n$

Initialize γ_l for $l = 1, \ldots, L$;

while *stopping criteria is met* **do**

- Find the Lagrangian coefficients α_i, $i = 1, \ldots, n$ solving the inner quadratic optimization problem and compute the corresponding $J(\gamma)$;
- Compute the partial derivative $\frac{\partial J(\gamma)}{\partial \gamma_m}$ for every $m = 1, \ldots, L$;
- Compute d based on (3.48);
- $\gamma \leftarrow \gamma + \beta d$

end

In [Z. Chen and Li, 2007] the problem of finding γ is tackled through the minimization of the generalization error: the error ξ_i, $i = 1, \ldots, n$ obtained using the generated solution achieved fixing parameters γ is considered in a novel minimization problem

together with coefficients γ. Specifically, through an iterative algorithm, once the convex optimization problem defined in (3.44) is solved and an early value of parameter α is computed, the following minimization problem is considered.

$$\begin{aligned} \min_{\gamma,\xi} \quad & \sum_{l=1}^{L} \gamma_l + \lambda \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \quad & y_i \Big(\sum_{j=1}^{n} \alpha_j y_j \sum_{l=1}^{L} \gamma_l k_l(x^{i^l}, x^{j^l}) + b \Big) \geq 1 - \xi_i \\ & \xi_i \geq 0 \; i = 1, \ldots, n, \\ & \gamma_l \geq 0 \; l = 1, \ldots, L, \end{aligned} \tag{3.49}$$

Algorithm 9 shows the approach described above in a schematic way.

Algorithm 9: Multiple kernel Support Vector Machine approach outlined in [Z. Chen and Li, 2007].

Data: A set of training data $X = \{(x^i)\}_{i=1}^n$ with $x^i \in \mathbb{R}^m$, a set of training labels $Y = \{(y_i)\}_{i=1}^n$ with $y_i \in \{-1, +1\}$

Result: A combination of the parameters γ_l, $l = 1, \ldots, L$ and Lagrangian multipliers α_i, $i = 1, \ldots, n$

Initialize γ_l for $l = 1, \ldots, L$;

while *stopping criteria is met* **do**

Find the Lagrangian coefficients α_i, $i = 1, \ldots, n$ solving the inner quadratic optimization problem;

Find the combination coefficients γ_l, $l = 1, \ldots, L$ solving the optimization problem defined in (3.49);

end

Note that, as in Algorithms 7 and 8, the stopping criteria is a preset condition forcing the algorithm to stop its iteration process after a finite number of iterations.

3.4 Infinite Kernel Learning

The problem of considering multiple base kernel functions may be broadly extended with the use of infinitely many kernels. In this scenario, namely where an infinite number of kernel functions is used, we talk about Infinite Kernel Learning (IKL).

As specified in the previous section, Multiple Kernel Learning uses a finite number of preset base kernel functions and combines them in order to obtain a final combination kernel. Specifically, let $\{k_l(x^{i^l}, x^{j^l})\}_{l=1}^L$ be a set of L predefined base kernel functions.

Most multiple kernel approaches use a linear combination technique, namely

$$k_\gamma(x^i, x^j) = f_\gamma\big(\{k_l(x^{i^l}, x^{j^l})\}_{l=1}^L\big) = \sum_{l=1}^{L} \gamma_l k_l(x^{i^l}, x^{j^l}),$$

with γ_l combination parameters defined by the the convex constraints, i.e.,

$$\begin{aligned} \gamma_l &\geq 0, \\ \sum_{l=1}^{L} \gamma_l &= 1. \end{aligned}$$

In this way, Multiple Kernel Learning tries to improve the obtained solution, tackling the given problem through the use of the "best" kernel function and, therefore, considering different data representation and correlation factors.

Kernel learning approach is further extended with the idea of exploiting a combination of possibly infinite base kernels. In this way, no limitation about the number of finite base kernel functions exists, leaving a broader possibility of choice other than allowing different multi data similarities. Over the last twenty years, a limited number of research works has been dealing with the idea of using an infinite base kernel set. As a starting point, [Argyriou et al., 2006] proposed an interesting Difference of Convex functions (DC) approach for learning the kernel functions starting from a predefined finite base kernel set. As a direct consequence, [Gehler and Nowozin, 2008] firstly proposed an implementation trying to solve an Infinite Kernel Learning Support Vector Machine problem based on the theoretical formulation expressed in [Argyriou et al., 2006]. Along the lines of the infinite kernel technique proposed by [Gehler and Nowozin, 2008], [Y. Liu et al., 2017] tackles the problem of Infinite Kernel Learning problems with respect to its convergence rate. Specifically, the paper proposes a new generalization error method, called Principal Eigenvalue Proportion (PEP), based on spectrum analysis, characterized by fast convergence rate with the skill of learning infinite kernel combination parameters.

Let us focus on the infinite kernel Support Vector Machine approach described in [Gehler and Nowozin, 2008]. We consider the classical n dimensional generic dataset $\mathcal{D}$ as in (2.1) with inputs $x^i \in X \subset \mathbb{R}^m$ and corresponding label set Y with $y_i \in \{-1, +1\}$. As already described in the previous section, a general primal version of Multiple Kernel Learning Support Vector Machine is given by the following formulation, convex

with respect to both γ_l and v_l.

$$\begin{aligned} \min_{v,\theta,\xi,\gamma} \quad & \frac{1}{2}\sum_{l=1}^{L}\frac{1}{\gamma_l}||v_l||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i\sum_{l=1}^{L}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} + y_i\theta \geq 1-\xi_i, \ \ \forall i = 1,\dots,n \\ & \xi_i \geq 0 \ \ \forall i = 1,\dots,n, \\ & \sum_{l=1}^{L}\gamma_l = 1, \ \gamma_l \geq 0 \ \ \forall l = 1,\dots,L, \end{aligned} \tag{3.50}$$

with $v_l \coloneqq \gamma_l w_l$ and $\phi_l : x \in \mathbb{R}^m \mapsto \phi_l(x) \in \mathbb{R}^{m'}$, l-th feature function, mapping points in the m-dimensional input set to a feature set of higher dimension m'.

Now, in Infinite Kernel Learning the kernel set $\mathcal{K}^{\text{inf}}$ given by

$$\mathcal{K}^{\text{inf}} = \left\{ \int_{\Omega} k_l \mathrm{d}p(l) \ : \ p \in \mathcal{M}(\Omega) \right\},$$

is considered, where notation k_l defines the kernel function with corresponding parameter γ_l, with l belonging to the compact index set Ω. Moreover, note that notation $\mathcal{M}(\Omega)$ means the set of all probability measures defined in set Ω [Y. Liu et al., 2017].

Then, Problem (3.50) can be extended to

$$\begin{aligned} \inf_{\Omega_f \subset \Omega} \quad \min_{v,\theta,\xi,\gamma} \quad & \frac{1}{2}\sum_{l\in\Omega_f}\frac{1}{\gamma_l}||v_l||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t.} \quad & y_i\sum_{l\in\Omega_f}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} + y_i\theta \geq 1-\xi_i, \ \ \forall i = 1,\dots,n \\ & \xi_i \geq 0 \ \ \forall i = 1,\dots,n, \\ & \sum_{l\in\Omega_f}\gamma_l = 1, \ \gamma_l \geq 0, \end{aligned} \tag{3.51}$$

where the inner problem is a classical multiple kernel Support Vector Machine framework and the outer optimization is tackled with respect to the set Ω_f, a closed and bounded finite subset of original compact index set Ω of possibly infinite size. As usual, the Wolfe-dual of Problem (3.51) is constructed. To do so, we consider the Lagrangian function of (3.51) given by

$$\begin{aligned} \mathcal{L}(v,\theta,\xi,\gamma,\alpha,\eta,\lambda,\delta) = & \frac{1}{2}\sum_{l\in\Omega_f}\frac{1}{\gamma_l}||v_l||^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\Bigg(y_i\sum_{l\in\Omega_f}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} + y_i\theta \\ & - 1 + \xi_i\Bigg) - \sum_{i=1}^{n}\eta_i\xi_i + \lambda\Bigg(\sum_{l\in\Omega_f}\gamma_l - 1\Bigg) - \sum_{l\in\Omega_f}\delta_l\gamma_l, \end{aligned} \tag{3.52}$$

with Lagrangian multipliers $\alpha_i, \eta_i, \delta_l \in \mathbb{R}^+$ for every $i = 1,\dots,n$ and $l = 1,\dots,L$ and $\lambda \in \mathbb{R}$. For the sake of simplicity, we use the truncated notation $\mathcal{L}(v,\theta,\xi,\gamma,\alpha,\eta,\lambda,\delta) = \mathcal{L}$.

First, note that (3.52) may be equivalently written in the following way

$$\mathcal{L} = -\frac{1}{2}\sum_{l\in\Omega_f}\frac{1}{\gamma_l}||v_l||^2 + \sum_{l\in\Omega_f}\frac{1}{\gamma_l}||v_l||^2 - \sum_{i=1}^{n}\alpha_i y_i\Big[\sum_{l\in\Omega_f}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}}\Big] - \sum_{i=1}^{n}\alpha_i y_i\theta$$
$$+\sum_{i=1}^{n}\alpha_i + \sum_{i=1}^{n}\xi_i\big(C-\alpha_i-\eta_i\big) - \sum_{i=1}^{n}\eta_i\xi_i + \lambda\left(\sum_{l\in\Omega_f}\gamma_l - 1\right) - \sum_{l\in\Omega_f}\delta_l\gamma_l. \tag{3.53}$$

Now, let us compute the partial derivatives of the Lagrangian function $\mathcal{L}$ with respect to variables v, θ, ξ and γ and set them to zero.

$$\frac{\partial\mathcal{L}}{\partial v_l} = \frac{1}{\gamma_l}v_l - \sum_{i=1}^{n}\alpha_i y_i\phi_l(x^i) = 0, \tag{3.54a}$$
$$\frac{\partial\mathcal{L}}{\partial\theta} = \sum_{i=1}^{n}\alpha_i y_i = 0, \tag{3.54b}$$
$$\frac{\partial\mathcal{L}}{\partial\xi_i} = C - \alpha_i - \eta_i = 0, \tag{3.54c}$$
$$\frac{\partial\mathcal{L}}{\partial\gamma_l} = -\frac{1}{2}\frac{1}{\gamma_l^2}||v_l||^2 + \lambda - \delta_l = 0. \tag{3.54d}$$

Now, substituting the values obtained through (3.54) in Equation (3.53) we obtain

$$\mathcal{L} = \sum_{l\in\Omega_f}\gamma_l\underbrace{\left[-\frac{1}{2}\frac{1}{\gamma_l^2}||v_l||^2 + \lambda - \delta_l\right]}_{=0 \text{ from (3.54d)}} + \sum_{l\in\Omega_f}v_l\underbrace{\left[\frac{1}{\gamma_l}v_l - \sum_{i=1}^{n}\alpha_i y_i\phi_l(x^i)\right]}_{=0 \text{ from (3.54a)}}$$
$$+\sum_{i=1}^{n}\xi_i\underbrace{\Big[C-\alpha_i-\eta_i\Big]}_{=0 \text{ from (3.54c)}} - \theta\underbrace{\left[\sum_{i=1}^{n}\alpha_i y_i\right]}_{=0 \text{ from (3.54b)}} + \sum_{i=1}^{n}\alpha_i - \lambda.$$

Finally, using both (3.54a) and (3.54d), we obtain the Wolfe-dual. Specifically, from Equation (3.54a) we have that

$$\frac{1}{\gamma_l}v_l = \sum_{i=1}^{n}\alpha_i y_i\phi_l(x^i).$$

Now, using it in (3.54d) we get the following result

$$\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j\langle\phi_l(x^i), \phi_l(x^j)\rangle_{\mathcal{H}} \leq \lambda,$$

where the positiveness of Lagrangian multiplier η_i, $\forall i = 1, \ldots, n$ is used.

Hence, the Wolfe-dual problem of Problem (3.51) takes the form

$$\begin{aligned} \sup_{\Omega_f \subset \Omega} \quad & \max_{\alpha,\lambda} \quad \sum_{i=1}^{n} \alpha_i - \lambda \\ & \text{s.t.} \quad 0 \leq \alpha_i \leq C, \ \ \forall i = 1, \ldots, n \\ & \qquad T(l, \alpha) \leq \lambda \ \ \forall l \in \Omega_f \\ & \qquad \alpha \in \mathbb{R}^n, \ \lambda \in \mathbb{R}, \end{aligned} \tag{3.55}$$

where

$$T(l, \alpha) \coloneqq \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \phi_l(x^i), \phi_l(x^j) \rangle_{\mathcal{H}},$$

and $\alpha = (\alpha_1, \ldots, \alpha_n)^{\mathrm{T}} \in \mathbb{R}^n$.

As for the primal formulation of the problem, the infinite kernel Wolfe dual problem given in (3.55) is defined by a two-step optimization framework. Specifically, the problem is characterized by an inner maximization problem with respect to the Lagrangian multipliers α and λ and by an outer problem tackling the matter of finding the subspace $\Omega_f \subset \Omega$ such that Ω_f is the supremum of the compact index set Ω.

Consider the inner maximization problem alone.

$$\begin{aligned} \max_{\alpha,\lambda} \quad & \sum_{i=1}^{n} \alpha_i - \lambda \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \ \ \forall i = 1, \ldots, n \\ & T(l, \alpha) \leq \lambda, \ \ \forall l \in \Omega_f \\ & \alpha \in \mathbb{R}^n, \ \lambda \in \mathbb{R}, \end{aligned} \tag{3.56}$$

Let α^* and λ^* be the optimum solution of Problem (3.56). It is important to stress that Problem (3.56) satisfies all the required conditions needed for using Theorem 4.2 of [Hettich and Kortanek, 1993]. Specifically we have that the following conditions occur.

1. Ω_f is a compact set (is a closed bounded subset of a compact set Ω);

2. The objective function

$$f : (\alpha, \lambda) \mapsto \sum_{i=1}^{n} \alpha_i - \lambda$$

   is concave (it is a linear function with respect to both variables α and λ);

3. The constraint function

$$g : (l, \alpha, \lambda) \mapsto T(l, \alpha) - \lambda$$

   is convex with respect to α and λ (it is convex with respect to variable α and linear with respect to variable λ);

4. The optimum solution
$$\sum_{i=1}^{n} \alpha_i^* - \lambda^*$$
is finite ($0 \leq \alpha_i \leq C$ for every $i = 1, \ldots, n$, and $T(l, \alpha^*) \leq \lambda^* \leq \delta_l$);

5. For every set of $n+1$ elements $l_0, \ldots, l_n$ there exist $(\tilde{\alpha}, \tilde{\lambda})$ such that
$$T(l_t, \tilde{\alpha}) - \tilde{\lambda} < 0$$
for every $t = 0, \ldots, n$ (one can take $\tilde{\alpha_i} = 0$ for every $i = 1, \ldots, n$ and $\tilde{\lambda} \in \mathbb{R}^+$).

Then, from [Hettich and Kortanek, 1993] we have that the following statement holds.

Theorem 10. *If for every $l \in \Omega$ and $\alpha \in \mathbb{R}^n$ with $0 \leq \alpha_i \leq C$, $\forall i = 1, \ldots, n$, we have that the function $T(l, \alpha)$ takes only finite values, i.e., $T(l, \alpha) < \infty$, then there exists a finite set $\Omega_f \subset \Omega$ equivalent to the optimum value of Problem (3.55) with counterpart (α^*, λ^*), and the optimal values of Problems (3.55) and (3.56) overlap.*

Proof. The proof is a direct consequence of Theorem 4.2 of [Hettich and Kortanek, 1993]. See Appendix B for further details. □

Therefore, if the function $T(l, \alpha)$ assumes only finite values for every possible choice of $l \in \Omega$ and $0 \leq \alpha_i \leq C$, $i = 1, \ldots, n$, then an optimal solution of Problem (3.55) with only a finite number of strictly positive γ_l exists. Moreover, such optimum solution corresponds to the optimal of Problem (3.56) and, hence, may be computed solving the inner maximization problem alone.

Once Problem (3.56) is solved and the corresponding optimum solution α^*, λ^* are found, using a classical Support Vector Machine framework, the classifier takes the following form
$$f(\bar{x}) = \operatorname{sgn}\left\{ \sum_{i=1}^{n} \alpha_i^* y_i \sum_{l \in \Omega_f} \gamma_l k_l(\bar{x}, x^i) + \theta \right\},$$
with $\bar{x}$ new input point.

Once again, let us consider the implementation approach proposed in [Gehler and Nowozin, 2008]. In [Gehler and Nowozin, 2008], the index set Ω_f is designed through the use of an iterative approach. The proposed approach tackles the problem by splitting the original Problem (3.56) into two sub-problems: the *restricted master* problem and the *subproblem*. Starting from an initial finite set $\Omega_0 \subset \Omega$, at each iteration t, the restricted master problem searches for parameters α and λ with respect to index set Ω_t; as a counterpart, once such optimal values for α and λ are found, the subproblem looks for new indexes to be selected. Therefore, each iteration ends by getting a new index set Ω_t such that
$$\Omega_{t-1} \subseteq \Omega_t \subseteq \Omega.$$

Algorithm 10 proposes a pseudo-code for the described approach.

Algorithm 10: Support Vector Machine infinite kernel approach described in [Gehler and Nowozin, 2008].

Data: A set of training data $X = \{(x^i)\}_{i=1}^n$ with $x^i \in \mathbb{R}^m$,
a set of training labels $Y = \{(y_i)\}_{i=1}^n$ with $y_i \in \{-1, +1\}$,
a regularization parameter C,
a kernel parameter set Ω.

Result: Combination parameters γ_l, Lagrangian multipliers α_i, $i = 1, \ldots, n$ and θ.

Select $l_v \in \Omega$ and set $\Omega_0 = \{l_v\}$;
$t \leftarrow 0$;
while *stopping criteria is not met* **do**
  Find the Lagrangian coefficients α_i, $i = 1, \ldots, n$, λ and parameters θ and γ_l solving the multiple kernel Support Vector Machine Problem (3.56) using index set Ω_t;
  Compute $l_v = \arg\max_{l \in \Omega} T(l, \alpha)$;
  if $T(l_v, \alpha) < \lambda$ **then**
    break
  end
  Set $\Omega_{t+1} = \Omega_t \cup \{l_v\}$;
  $t \leftarrow t + 1$;
end

Note that, the restricted master problem consists in tackling the multiple kernel problem described above. Such problem may be solved using any standard Support Vector Machine technique stated in the previous section. Concerning the subproblem phase, it involves the resolution of the following problem

$$l_v = \arg\max_{l \in \Omega} T(l, \alpha) = \arg\max_{l \in \Omega} \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k_l(x^i, x^j). \tag{3.57}$$

If an optimal solution $l_v = \arg\max_{l \in \Omega} T(l, \alpha)$ such that $T(l_v, \alpha) < \lambda$ is found, the control directly exits from the loop statement and the algorithm terminates. Otherwise, a new iteration begins with $\Omega_{t+1} = \Omega_t \cup \{l_v\}$, namely adding the newly computed l_v to the index set. Therefore, the convergence of the iterative process proposed in Algorithm 10 strongly relies on the resolution of the subproblem. In this context, the following theorem constitutes a fundamental result to ensure its convergence.

Theorem 11. *Assume that the subproblem defined in (3.57) may be solved. Then, one of the following statement is true.*

1. *Algorithm 10 stops after a finite number of iterations with a solution to problem*

(3.56);

2. *Algorithm 10 has at least one point of accumulation, each one of these solving problem (3.56).*

Note that, Theorem 11 is a direct application of Theorem 7.2 in [Hettich and Kortanek, 1993], where the solution of a semi-infinite programming problem, namely a specific problem where an objective function defined by a finite number of variables is subject to an infinite number of constraints, is tackled through the use of a sequence of finite programming problems, i.e., with only a finite number of constraints.

Chapter 4

Two novel Machine Learning approaches

This chapter presents two novel results: "Infinite Kernel Extreme Learning Machine" and "Multi-Kernel Covariance Terms in Multi-Output Support Vector Machines". As it will be clearer reading the next sections, the common thread of such results is the idea of utilizing a combination of kernels in already existing or modified supervised Machine Learning frameworks.

First, a novel approach combining the existing Extreme Learning Machine algorithm together with the Infinite Kernel Learning approach discussed in Chapter 3 is proposed [Marcelli and De Leone, 2019]. Specifically, an in-depth description of the supervised learning technique is given, analysing its core structure and describing the corresponding classifier. Starting from the state of the art theory on Extreme Learning Machine and Infinite Kernel Learning, a new algorithm is proposed where, the original Extreme Learning Machine formulation is extended using a combination of possibly infinite kernel functions. The algorithm is a two-step procedure and an analysis on its convergence is included. Finally, the proposed method is tested on 20 public classification dataset and the obtained accuracy, precision and recall values are listed regarding both the training set and the test set.

Secondly, a pre-existing covariance function developed for multi-task Gaussian processes is included in a new Support Vector Machine framework [Marcelli and De Leone, 2020]. In particular, first the problem of solving more than one task at the same time, i.e., Multi-Task Learning, is presented, giving a description of the problem and defining the main differences with the Multi-Output Learning subfield. After that, an original idea for a multi-task kernel matrix is outlined. These multi-task kernels are eventually used in a newly developed Support Vector Machine framework, specifically suitable for the multi-task case. The proposed model is tested on four real open-source dataset as well as on 40 synthetic collections of data and the obtained

accuracy, precision and recall values are presented.

4.1 Infinite Kernel Extreme Learning Machine

4.1.1 Extreme Learning Machine

Extreme Learning Machine (ELM) is a supervised Machine Learning algorithm first introduced in 2004 [G.-B. Huang, Zhu, et al., 2004] which may be considered a combination between feedforward Neural Networks and Support Vector Machines.

The basic idea behind Extreme Learning Machine is to implement a method that can improve the learning speed of classical Single-Hidden Layer Feedforward Neural Networks (SLFNs). The term feedforward Neural Network defines an artificial neural network framework, i.e., a collection of nodes called neurons connected with each-other, where information flows only into one specific direction, with no presence of loops allowed. Specifically, in a feedforward Neural Network, information only moves forward, from the input to the output layer. The classical and simplest example of feedforward Neural Network is given by the Perceptron, which was deeply described in Chapter 2, a linear classifier where only two layers are allowed, i.e., the input layer and the output layer. During its learning phase, a Perceptron learns a real-valued vector of weights corresponding to each input neuron. When multiple layers of neurons are considered, we have a multiple-layer Neural Network, characterized by the presence of one or more inner layers, known with the name of *hidden layers*. Figure 4.1 shows a classical multiple-layer Perceptron with five input nodes and one hidden layer. As mentioned above, connections between neurons only go in one direction, namely from the input layer to the output layer, without the presence of loops. Furthermore, each arrow is associated with a weight and each neuron with a threshold value.

When a feedforward Neural Network is used, in order to obtain the optimal parameters, all weights and biases characterizing each layer have to be tuned, making the learning algorithm quite slow. Moreover, among the most used algorithms to solve the problem we find the gradient method, which is usually characterized by a slow convergence and may be defined by local minima convergence issues.

Extreme Learning Machine has the structure of classical feedforward Neural Networks with one or multiple hidden layers, but is characterized by a fundamental feature: hidden nodes variables, i.e., parameters and weights, need not to be tuned but are randomly assigned at the beginning of the algorithm and remain fixed throughout all the computation. In this way, Extreme Learning Machine is characterized by lower computational complexity, moving from an iterative based approach to a one-step type algorithm, maintaining, however, a remarkable performance compared to classical Machine Learning methods. Extreme Learning Machine algorithm present two important features that characterize it from classical Machine Learning approaches: better

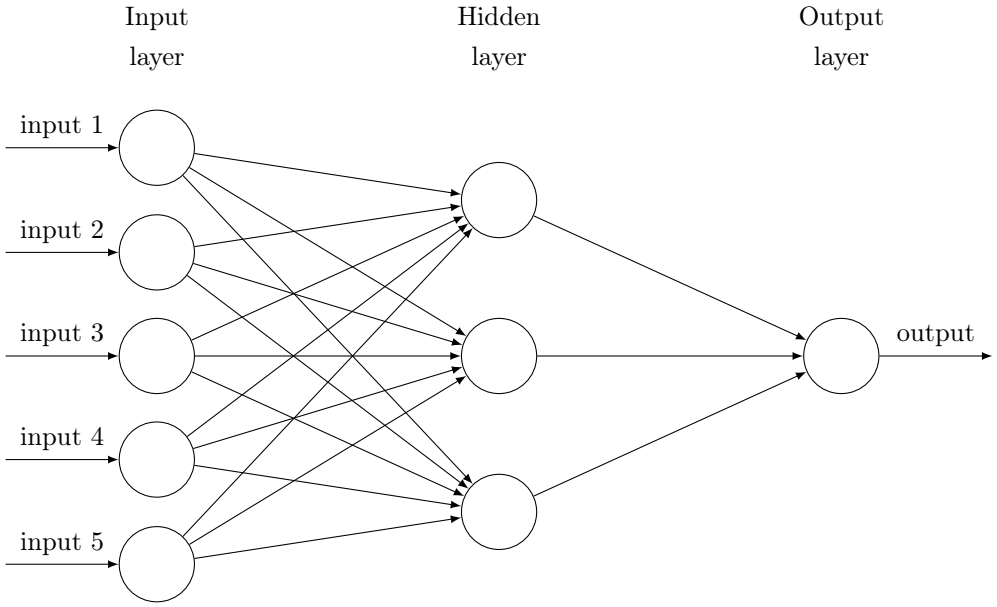


Figure 4.1: Multiple-layer Perceptron with one hidden layer.

generalization performance and faster convergence. Specifically, this method achieves the first goal since it not only aims to minimize the approximation error, given by the difference between the expected output and the computed result, but at the same time it also looks for the smallest norm weights which correctly fulfil the purpose.

Extreme Learning Machine was first introduced in the field of Single-Hidden Layer Feedforward Neural Networks with the aim of solving slow convergence phenomena and local minima convergence, developing a model which could improve the learning speed and, at the same time, reach a global optimum [G.-B. Huang, L. Chen, et al., 2006; G.-B. Huang, Zhu, et al., 2004, 2006]. Later, it was broaden to the general field of feedforward Neural Networks. The key idea behind Extreme Learning Machine affects the hidden layer: as already said, the parameters of hidden nodes need not to be learned but are randomly chosen as an initial step of the algorithm and never changed during the whole execution of the algorithm.

In particular, consider a Single-Hidden Layer Feedforward Neural Network with one hidden layer defined by Q hidden nodes and an output layer characterized by p output nodes. Given a generic input $x^i \in \mathbb{R}^m$ with corresponding generic output y_i, a classical Extreme Learning Machine architecture is mathematically modelled as

$$f(x^i) = \sum_{q=1}^{Q} w_q \phi_q(x^i), \tag{4.1}$$

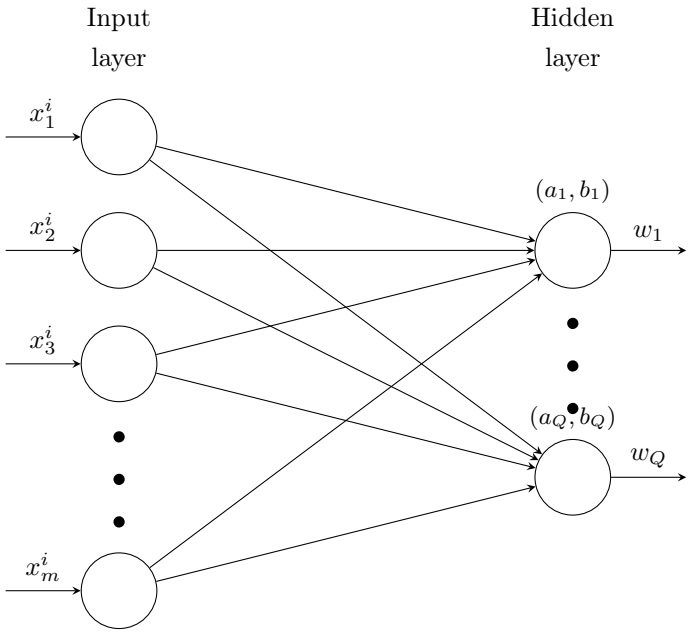


Figure 4.2: Extreme Learning Machine framework for a generic input vector $x^i \in \mathbb{R}^m$, $\forall i = 1, \ldots, n$. Hidden nodes parameters (a_q, b_q), $q = 1, \ldots, Q$, are randomly chosen at the beginning of the algorithm and never changed. Weights w_q, $q = 1, \ldots, Q$ connecting the input layer and the output layer are found solving problem (4.5).

where $w = (w_1, \ldots, w_Q)^{\mathrm{T}}$ is the output weight connecting the hidden layer with the output layer, and $\phi : \mathbb{R}^m \longrightarrow \mathbb{R}^Q$ maps data from the m-dimensional input space to the Q-dimensional hidden space.

Specifically,

$$\phi_q(x^i) = g_q(a_q, b_q, x^i) \tag{4.2}$$

defines the activation function, i.e., the output function, of the q-th hidden node with respect to input x^i with parameters (a_q, b_q). Note that, the activation function is decided and fixed at the beginning of the algorithm and may not be unique, meaning that each hidden node may have assigned a different output function.

Similarly to feedforward Neural Networks, in Extreme Learning Machine the goal is to minimize the training error but, at the same time, it aims to reach the smallest norm of the output weights w_q, $q = 1, \ldots, Q$ as well: as underlined in [Bartlett, 1998], the number of parameters does not effect the generalization performance of the network; on the contrary, the size of the weights tends to strongly affect the accuracy of the algorithm. In order to obtain such results, the concept of minimum norm least-square solution of a general linear system is used.

Let us consider a generic multi-output training set

$$\{x^i, y^i\}_{i=1}^n \tag{4.3}$$

where $x^i = (x_1^i, \dots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ and $y^i = (y_1^i, \dots, y_p^i)^{\mathrm{T}} \in \mathbb{R}^p$, with $y_j^i \in \{0, 1\}$. Specifically, $y_j^i = 1$ whether x^i belongs to the jth class, while on the contrary $y_j^i = 0$ if x^i does not belong to the jth class, for every $i = 1, \dots, n$ and $j = 1, \dots, p$. Let $\Phi \in \mathbb{R}^{n \times Q}$ and $Y \in \mathbb{R}^{n \times p}$ respectively be the hidden layer output matrix and the matrix containing all the labels y^i for $i = 1 \dots n$. Namely

$$\Phi = \begin{pmatrix} (\phi(x^1))^{\mathrm{T}} \\ \vdots \\ (\phi(x^n))^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} \phi_1(x^1) & \dots & \phi_Q(x^1) \\ \vdots & \ddots & \vdots \\ \phi_1(x^n) & \dots & \phi_Q(x^n) \end{pmatrix} \in \mathbb{R}^{n \times Q},$$

and

$$Y = \begin{pmatrix} (y^1)^{\mathrm{T}} \\ \vdots \\ (y^n)^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} y_1^1 & \dots & y_p^1 \\ \vdots & \ddots & \vdots \\ y_1^n & \dots & y_p^n \end{pmatrix} \in \mathbb{R}^{n \times p}.$$

In Extreme Learning Machine the key idea is to solve the following problem

$$\text{minimize} \begin{cases} ||\Phi w - Y||_\pi^{\sigma_1} \\ ||w||_\tau^{\sigma_2} \end{cases} \tag{4.4}$$

with $\sigma_1 > 0$, $\sigma_2 > 0$ and π and τ defining the utilized norm function. Therefore, Extreme Learning Machine aims to tackle the problem by considering the minimization of the approximation error, given by the difference between the expected output and the computed result, while at the same time looking for the smallest norm weights $w \in \mathbb{R}^{Q \times p}$. In this way, as we will see in-depth later, the method tends to have a better generalization performance with respect to classical feedforward Neural Networks, together with a faster convergence rate. In particular, observing the problem defined in (4.4), its goal is not only to reduce the gap between the given output and the computed output, but it also searches for the smallest norm weight w. The aim is to determine w to be the minimum norm least-square solution of the system $||\Phi w - Y||$, i.e., the smallest 2-norm solution among all the least-square solutions.

A similarly worded as in (4.4), Extreme Learning Machine aims is to solve the following minimization problem

$$\min_{w \in \mathbb{R}^{Q \times p}} \quad ||\Phi w - Y||^2, \tag{4.5}$$

leaving out the minimization over w in substance. The optimal solution to Problem (4.5) is given by

$$\hat{w} = \Phi^\dagger Y,$$

with $\Phi^\dagger$ the Moore-Penrose generalized pseudo-inverse of matrix Φ. Note that, it is possible to show that this special matrix $\Phi^\dagger$ is indeed the minimum norm least-square solution of Problem (4.5) [G.-B. Huang, Zhu, et al., 2004]. Moreover, the convergence speed is characterized by the fact that Extreme Learning Machine randomly assigns

values to the hidden layer, only adjusting the output weights. In this way, the classical minimization problem to be solved with iterative adjustments becomes a one step algorithm, aiming to compute the pseudo-inverse of matrix Φ, reducing in this manner the computational costs.

The minimization of the norm of the weights and of the classification error is given by

$$\begin{aligned} \min_{w \in \mathbb{R}^{Q \times p}} \quad & \frac{1}{2}||w||^2 + \frac{C}{2}\sum_{i=1}^{n}||\xi^i||^2 \\ \text{s.t} \quad & \phi(x^i)^{\mathrm{T}} w = {y^i}^{\mathrm{T}} - {\xi^i}^{\mathrm{T}} \quad i = 1, \ldots, n, \end{aligned} \tag{4.6}$$

where $\xi^i = (\xi_1^i, \ldots, \xi_p^i)^{\mathrm{T}} \in \mathbb{R}^p$ is the training error vector with respect to input x^i, for every $i = 1, \ldots, n$, and C is the regularization parameter. The Lagrangian function associated to Problem (4.6) is given by the following equation

$$\mathcal{L}_{\mathrm{ELM}}(w, \xi, \alpha) = \frac{1}{2}||w||^2 + \frac{C}{2}\sum_{i=1}^{n}||\xi^i||^2 - \sum_{i=1}^{n}\sum_{j=1}^{p}\alpha_j^i\Big(\phi(x^i)^{\mathrm{T}} w^j - y_j^i + \xi_j^i\Big) \tag{4.7}$$

where $\alpha^i = (\alpha_1^i, \ldots, \alpha_p^i)^{\mathrm{T}} \in \mathbb{R}^p$, $i = 1, \ldots, n$, are the Lagrangian parameters. The Karush-Kuhn-Tucker (KKT) optimality conditions are obtained by calculating the partial derivatives of $\mathcal{L}_{\mathrm{ELM}}(w, \xi, \alpha)$ with respect to variables w, ξ and α and set them to zero. Specifically, we have

$$\begin{aligned} &\frac{\partial \mathcal{L}_{\mathrm{ELM}}}{\partial w^j} = w^j - \sum_{i=1}^{n}\alpha_j^i \phi(x^i)^{\mathrm{T}} = 0, \\ &\frac{\partial \mathcal{L}_{\mathrm{ELM}}}{\partial \xi^i} = C\xi^i - \alpha^i = 0, \quad i = 1, \ldots, n, \\ &\frac{\partial \mathcal{L}_{\mathrm{ELM}}}{\partial \alpha^i} = \phi(x^i)^{\mathrm{T}} w - ({y^i}^{\mathrm{T}} - {\xi^i}^{\mathrm{T}}) = 0, \quad i = 1, \ldots, n, \end{aligned}$$

or equivalently

$$w = \Phi^{\mathrm{T}}\alpha, \tag{4.8a}$$

$$\alpha = C\xi, \tag{4.8b}$$

$$\Phi w - (Y - \xi) = 0. \tag{4.8c}$$

Based on the size n of the data set specified in (4.3) and on the number Q of chosen hidden neurons, different solutions of Problem (4.6) exist. Specifically, the following two scenarios may occur.

1. If $n < Q$, i.e., the number of training data is not bigger than the number of the hidden neurons, matrix Φ has more columns than rows. In this specific scenario, the following procedure ca be applied: the values of equations (4.8a) and (4.8b) are substituted in equation (4.8c) obtaining the following

$$\Phi\Phi^{\mathrm{T}}\alpha - Y + \frac{1}{C}\mathrm{I}\alpha = 0 \Rightarrow \alpha = \left(\frac{1}{C}\mathrm{I} + \Phi\Phi^{\mathrm{T}}\right)^{-1} Y, \tag{4.9}$$

where I is the identity matrix of dimension $n \times n$.
Now, combining (4.8a) and (4.9) we obtain

$$w = \Phi^{\mathrm{T}} \left(\frac{1}{C} \mathrm{I} + \Phi \Phi^{\mathrm{T}} \right)^{-1} Y. \tag{4.10}$$

Remembering Equation (4.1) and substituting the value of w defined by (4.10), the Extreme Learning Machine output function corresponding to a generic input value $\bar{x}$ can be obtained as

$$f(\bar{x}) = \sum_{q=1}^{Q} w_q \phi_q(\bar{x}) = \phi(\bar{x}) \Phi^{\mathrm{T}} \left(\frac{1}{C} \mathrm{I} + \Phi \Phi^{\mathrm{T}} \right)^{-1} Y. \tag{4.11}$$

2. If $n > Q$, namely when a very large dataset is considered, a different technique is used. Specifically, substituting (4.8b) into the Equation (4.8a), we obtain

$$w = C \Phi^{\mathrm{T}} \xi \Longrightarrow \xi = \frac{1}{C} (\Phi^{\mathrm{T}})^{\dagger} w. \tag{4.12}$$

Now, using the result for ξ obtained from Equation (4.12) into Equation (4.8c) we obtain that

$$\Phi w - Y - \frac{1}{C} (\Phi^{\mathrm{T}})^{\dagger} w = 0,$$

from which we have

$$w = \left(\frac{1}{C} \mathrm{I} + \Phi^{\mathrm{T}} \Phi \right)^{-1} \Phi^{\mathrm{T}} Y,$$

where we used the fact that $\Phi^{\mathrm{T}} (\Phi^{\mathrm{T}})^{\dagger} = \mathrm{I}$. Hence, as before, considering Equation (4.1), if $n > Q$ the Extreme Learning Machine classifier for a generic input $\bar{x}$ is now

$$f(\bar{x}) = \sum_{q=1}^{Q} w_q \phi_q(\bar{x}) = \phi(\bar{x}) \left(\frac{1}{C} \mathrm{I} + \Phi^{\mathrm{T}} \Phi \right)^{-1} \Phi^{\mathrm{T}} Y. \tag{4.13}$$

In the classical Extreme Learning Machine classifier, the hidden layer purpose is to map the given input from the original space to a higher dimensional space. In particular, in this specific scenario a mapping $\phi : \mathbb{R}^m \longrightarrow \mathbb{R}^Q$ is considered, where the generic component q is equal to $\phi_q(x^i) = g_q(a_q, b_q, x^i)$, corresponding to the activation function of the q-th hidden neuron, $q = 1, \ldots, Q$. Therefore, through the hidden layer, input data is mapped into a higher dimensional space. As a direct consequence, this novel space may be considered as a feature space, leading to the implementation of an Extreme Learning Machine kernel function.

It is important to note that, in Equation (4.11) matrix Φ only appears in the form of the product $\Phi \Phi^{\mathrm{T}}$ and, analogously, in Equation (4.13) matrix Φ shows up purely with the term $\Phi^{\mathrm{T}} \Phi$. These expressions are also known as *Extreme Learning Machine kernel matrix* with elements of the form

$$\phi(x^i)^{\mathrm{T}} \phi(x^j),$$

for every $i, j = 1, \dots, n$, corresponding to the inner product in the considered feature space (in this case $\mathbb{R}^Q$).

Differently from classical kernel methods, where the corresponding kernel function is selected based on the specific set of data and on the given problem, here the feature map and the associated kernel function strictly rely on the hidden layer. Therefore, since, by definition, weights on the arcs from the input to the hidden layer are randomly assigned at the beginning of the algorithm, the corresponding Extreme Learning Machine kernel function as described in (4.2) is randomly established and cannot be modified.

4.1.2 Related work

Our main goal is to combine Extreme Learning Machine together with the idea of Infinite Kernel Learning, i.e., to use an infinite combination of base kernels. Specifically, we will introduce an algorithm, called *Infinite Kernel Extreme Learning Machine* (IK-ELM), tackling the problem of optimizing a single feedforward Neural Network with hidden nodes randomly initiated and never changed, with the use of infinitely many base kernel functions. In particular, our starting point is the work by [X. Liu et al., 2015], where an approach called *Multiple Kernel Extreme Learning Machine* (MK-ELM) is described. The base concept there is to merge the Extreme Learning Machine structure with Multiple Kernel Learning technique: the computed optimal kernel function is a combination of predefined base kernel functions. The coefficients of such kernel combination, together with the Extreme Learning Machine parameters, are learnt during the process.

Consider the generic training set defined in (4.3). The Multiple Kernel Extreme Learning Machine problem is defined as follows:

$$\begin{aligned} \min_{\gamma} \min_{v,\xi} \ & \frac{1}{2}\sum_{l=1}^{L}\frac{1}{\gamma_l}||v_l||^2 + \frac{C}{2}\sum_{i=1}^{n}||\xi^i||^2 \\ \text{s. t.} \ & \sum_{l=1}^{L}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} = y^i - \xi^i, \ \ \forall i = 1, \dots, n \\ & \sum_{l=1}^{L}\gamma_l = 1, \ \ \gamma_l \geq 0, \ \ \forall l = 1, \dots, L, \end{aligned} \tag{4.14}$$

where $\{\phi_l(\cdot)\}_{l=1}^L$ are the L feature mapping from $\mathbb{R}^m$ to a Hilbert space $\mathcal{H}$ of undefined dimension corresponding to the predefined base kernels $\{k_l(\cdot,\cdot)\}_{l=1}^L$, $\{\gamma_l\}_{l=1}^L$ are the base kernel combination parameters, and $v_l := \sqrt{\gamma_l}w_l$ for $l = 1, \dots, L$ as for the classical Multiple Kernel Support Vector Machine framework described in Chapter 3. Moreover, $v_l \in \mathbb{R}^{|\phi_l(\cdot)|\times p}$, for every $l = 1, \dots, L$, leading to

$$v \coloneqq (v_1, \dots, v_L) \in \mathbb{R}^{(|\phi_1(\cdot)|+\dots+|\phi_L(\cdot)|)\times p}.$$

The Lagrangian function corresponding to Problem (4.14) is given by

$$\mathcal{L}_{\text{MK-ELM}}(\gamma, v, \xi, \alpha, \lambda, \delta) = \frac{1}{2}\sum_{l=1}^{L}\frac{1}{\gamma_l}||v_l||^2 + \frac{C}{2}\sum_{i=1}^{n}||\xi^i||^2 - \sum_{l=1}^{L}\delta_l\gamma_l + \lambda\left(\sum_{l=1}^{L}\gamma_l - 1\right) - \sum_{i=1}^{n}\sum_{j=1}^{p}\alpha_j^i\left(\sum_{l=1}^{L}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} - y_j^i + \xi_j^i\right).$$

In [X. Liu et al., 2015] an iterative scheme for the Multiple Kernel Extreme Learning Machine algorithm for both sparse and non-sparse case is outlined. Specifically, after computing the partial derivatives of $\mathcal{L}_{\text{MK-ELM}}$ with respect to variables v_l, ξ_j^i and α_j^i, the following matrix expression is obtained

$$\alpha = \left(K_\gamma + \frac{1}{C}\mathrm{I}\right)Y^{\mathrm{T}}, \tag{4.15}$$

where K_γ is the matrix of elements

$$k_\gamma(x^i, x^j) = \sum_{l=1}^{L}\gamma_l k_l(x^i, x^j),$$

for every $i, j = 1, \ldots, n$. Algorithm 11 schematizes the Multiple Kernel Extreme Learning Machine approach proposed by [X. Liu et al., 2015].

Algorithm 11: Multiple kernel extreme learning machine general approach described in [X. Liu et al., 2015]. Note that, depending on the specific type of dataset, i.e., sparse or non sparse, kernel combination weights γ are updated using a different rule.

Data: A set of training data $\{x^i, y^i\}_{i=1}^n$, regularization parameter C, a set of base kernel functions k_l, $l = 1, \ldots, L$
Result: Combination parameters γ_l, Lagrangian multipliers α
Initialize $\gamma \leftarrow \gamma^0$;
$t \leftarrow 0$;
while $\max\{|\gamma^{t+1} - \gamma^t|\} < 1e-4$ **do**
  Compute $k_\gamma(\cdot,\cdot) = \sum_{l=1}^{L}\gamma_l k_l(\cdot,\cdot)$;
  Find α^t solving equation (4.15);
  Update γ^{t+1};
  $t \leftarrow t+1$
end

4.1.3 The proposed model

We extend the formulation described in (4.14) using a combination of possibly infinitely many base kernel functions. From now on, for the sake of calculations, let us

consider the single output case. Specifically, we consider a classification training set $\mathcal{D}$, with

$$\mathcal{X} = \{x^i\}_{i=1}^n$$

feature set with elements $x^i = (x_1^i, x_2^i, \ldots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$, $i = 1, \ldots, n$, and

$$\mathcal{Y} = \{y_i\}_{i=1}^n$$

label set with $y_i \in \{-1, 1\}$, for every $i = 1, \ldots, n$. Note that, as in [Gehler and Nowozin, 2008], Ω_f and Ω are, respectively, a finite set and a set of undefined cardinality of kernel parameters. Moreover, the feature maps are defined as

$$\phi_l : \mathbb{R}^m \longrightarrow \mathcal{H},$$

where $\mathcal{H}$ is a Hilbert space, for every $l \in \Omega_f$. Then, the problem we wish to solve is the following

$$\begin{aligned} \inf_{\Omega_f \subset \Omega} \min_{\gamma} \min_{v,\xi} \; & \frac{1}{2} \sum_{l \in \Omega_f} \frac{1}{\gamma_l} ||v_l||^2 + \frac{C}{2} \sum_{i=1}^n ||\xi_i||^2 \\ \text{s. t.} \; & \sum_{l \in \Omega_f} \langle v_l, \phi_l(x^i) \rangle_{\mathcal{H}} = y_i - \xi_i, \;\; \forall i = 1, \ldots, n \\ & \sum_{l \in \Omega_f} \gamma_l = 1, \;\; \gamma_l \geq 0, \;\; l \in \Omega_f, \end{aligned} \tag{4.16}$$

with the set of all possible kernels theoretically containing an uncountable number of elements. In Problem (4.16) we can define an inner problem with respect to variables v, ξ and γ and an outer problem searching for the best finite subset of $\Omega_f \subset \Omega$. Furthermore, as set out in more detail below, the inner problem is subdivided once again, splitting it with respect to the two minimization processes, respectively regarding γ and v, ξ.

Consider first the inner problem

$$\begin{aligned} \min_{\gamma} \min_{v,\xi} \; & \frac{1}{2} \sum_{l \in \Omega_f} \frac{1}{\gamma_l} ||v_l||^2 + \frac{C}{2} \sum_{i=1}^n ||\xi_i||^2 \\ \text{s. t.} \; & \sum_{l \in \Omega_f} \langle v_l, \phi_l(x^i) \rangle_{\mathcal{H}} = y_i - \xi_i, \;\; \forall i = 1, \ldots, n \\ & \sum_{l \in \Omega_f} \gamma_l = 1, \;\; \gamma_l \geq 0, \;\; l \in \Omega_f. \end{aligned} \tag{4.17}$$

To build the dual problem of (4.17), we construct the Lagrangian function corresponding

to this inner problem given by the following expression.

$$\begin{aligned}\mathcal{L}_{\text{IK-ELM}}(\gamma, v, \xi, \alpha, \lambda, \delta) =& \frac{1}{2}\sum_{l\in\Omega_f}\frac{1}{\gamma_l}||v_l||^2 + \frac{C}{2}\sum_{i=1}^{n}||\xi_i||^2 \\ &- \sum_{l\in\Omega_f}\delta_l\gamma_l + \lambda\left(\sum_{l\in\Omega_f}\gamma_l - 1\right) \\ &- \sum_{i=1}^{n}\alpha_i\left(\sum_{l\in\Omega_f}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} - y_i + \xi_i\right),\end{aligned} \tag{4.18}$$

with α_i, λ and δ_l Lagrangian multipliers. Moreover the Lagrangian multipliers δ_l are non-negative for all $l \in \Omega_f$. The Karush–Kuhn–Tucker conditions corresponding to Problem (4.17) are given by the following expressions:

$$\frac{\partial\mathcal{L}_{\text{IK-ELM}}}{\partial v_l} = \frac{1}{\gamma_l}v_l - \sum_{i=1}^{n}\alpha_i\phi_l(x^i)^{\mathrm{T}} = 0, \tag{4.19a}$$

$$\frac{\partial\mathcal{L}_{\text{IK-ELM}}}{\partial\gamma_l} = -\frac{1}{2}\frac{1}{\gamma_l^2}||v_l||^2 + \lambda - \delta_l = 0, \tag{4.19b}$$

$$\frac{\partial\mathcal{L}_{\text{IK-ELM}}}{\partial\xi^i} = C\xi^i - \alpha_i = 0, \tag{4.19c}$$

$$\frac{\partial\mathcal{L}_{\text{IK-ELM}}}{\partial\alpha_i} = \sum_{\gamma_l}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} - y^i + \xi^i = 0. \tag{4.19d}$$

Substituting Equations (4.19a)-(4.19d) in Equation (4.18) we can rewrite the Lagrangian function as

$$\begin{aligned}\mathcal{L}_{\text{IK-ELM}}(\gamma, v, \xi, \alpha, \lambda, \delta) =& \sum_{l\in\Omega_f}\frac{1}{\gamma_l}||v_l||^2 + \sum_{l\in\Omega_f}\left(-\frac{1}{2}\frac{1}{\gamma_l^2}||v_l||^2 - \delta_l + \lambda\right) - \lambda \\ &+ \sum_{i=1}^{n}\alpha_i y_i + \sum_{i=1}^{n}\xi_i(C\xi_i - \alpha_i) - \frac{C}{2}\sum_{i=1}^{n}||\xi_i||^2 \\ &- \sum_{i=1}^{n}\alpha_i\left(\sum_{l\in\Omega_f}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}}\right).\end{aligned} \tag{4.20}$$

Taking into account the fact that the Lagrangian multipliers δ_l are non-negative for all $l \in \Omega_f$, we obtain the following dual form of Problem (4.17)

$$\begin{aligned}&\max_{\alpha,\lambda}\ \sum_{i=1}^{n}\alpha_i\left(y_i - \frac{\alpha_i}{2C}\right) - \lambda \\ &\text{s. t. } T(l,\alpha) \le \lambda,\ \ \forall l \in \Omega_f,\end{aligned} \tag{4.21}$$

where

$$T(l,\alpha) := \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j\langle\phi_l(x^i), \phi_l(x^j)\rangle_{\mathcal{H}}.$$

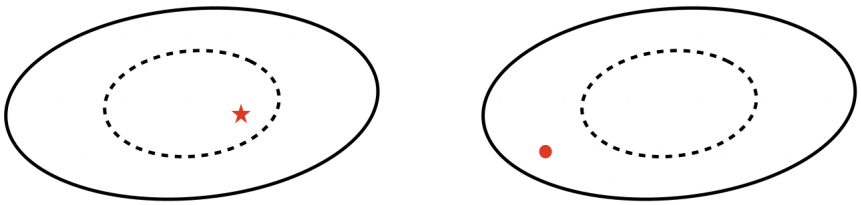


Figure 4.3: The solid line object represents the feasible region of Problem (4.21). The dotted line object shows the feasible region of Problem (4.22). The red star presents a feasible solution for both problems. The red dot shows an example of a point lying in the feasible region of Problem (4.21) but not in the feasible region of Problem (4.22).

The constraint $T(l, \alpha) \leq \lambda$ is obtained using Equation (4.19b).

Given Problem (4.21), we can construct the dual form of Problem (4.16) as follows

$$\begin{aligned} &\max_{\alpha,\lambda} \ \sum_{i=1}^{n} \alpha_i \left(y_i - \frac{\alpha_i}{2C} \right) - \lambda \\ &\text{s. t. } \ T(l, \alpha) \leq \lambda, \ \ \forall l \in \Omega. \end{aligned} \tag{4.22}$$

Note that, in this formulation, the feasible region, namely the set containing all the feasible solutions, is smaller than the one corresponding to Problem (4.21). In fact, by construction, Problem (4.21) is defined by fewer constraints, given by

$$T(l, \alpha) \leq \lambda, \ \ \forall l \in \Omega_f.$$

Hence, if (α^*, λ^*) is an optimal solution of Problem (4.21), two things may happen: either (α^*, λ^*) verifies

$$T(l, \alpha^*) \leq \lambda^*, \ \ \forall l \in \Omega,$$

namely (α^*, λ^*) is an optimal solution of Problem (4.22); either there exists some $l \in \Omega$ such that

$$T(l, \alpha^*) > \lambda^*.$$

This second scenario is equivalent to the event that (α^*, λ^*) lies in the feasible region of Problem (4.21) but is not located in the feasible region of Problem (4.22). In this case, l is added to Ω_f, namely, a new kernel is considered. Figure 4.3 shows the two described scenarios.

Moreover, it is important to point out that, in the formulation of Problem (4.22) we leave out the supremum corresponding to the infimum of Problem (4.16): if there exists a solution (α^*, λ^*) with corresponding values ξ^*, v^*, γ^*, satisfying the condition $T(l, \alpha^*) \leq \lambda^*$ for all $l \in \Omega$, then it also satisfies the condition for all the finite sets $\Omega_f \subset \Omega$. Finally, as already pointed out in Chapter 3, from Theorem 4.2 of [Hettich

and Kortanek, 1993] it follows that if Problem (4.22) admits solution, then its optimal solution is defined by only a finite number of l different from zero. Therefore, the proposed algorithm searches for a finite set Ω_f as a solution of our problem.

Based on the above calculations, we propose an iterative two-step algorithm for Infinite Kernel Extreme Learning Machine, schematized in Algorithm 12 [Marcelli and De Leone, 2019]. With reference to the inner Problem (4.17), we define the following problem given by

$$\begin{aligned} &\min_{\gamma} \ \mathrm{S}(\gamma) \\ &\text{s. t.} \ \sum_{l\in\Omega_f} \gamma_l = 1 \\ &\qquad \gamma_l \geq 0, \ \ l \in \Omega_f, \end{aligned} \tag{4.23}$$

with

$$\begin{aligned} \mathrm{S}(\gamma) := \ \min_{v,\xi} \ & \frac{1}{2}\sum_{l\in\Omega_f}\frac{1}{\gamma_l}||v_l||^2 + \frac{C}{2}\sum_{i=1}^{n}||\xi_i||^2 \\ \text{s. t.} \ & \sum_{l\in\Omega_f}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} = y_i - \xi_i, \ \ \forall i = 1,\ldots,n. \end{aligned} \tag{4.24}$$

The proposed algorithm may be summarised as follows.

Step 1. Solve Problem (4.17) with values of γ fixed, where problem $S(\gamma)$ is defined as in (4.24);

Step 2. Look for the value of $l \in \Omega$ maximizing function $T(l, \alpha)$.

Algorithm 12 summarizes the proposed approach.

Now, we are going to analyze every step of Algorithm 12 in detail, in order to determine how it may be solved.

First, it should be pointed out that the main advantage of using Extreme Learning Machine framework is that, differently form other classification approaches, it does not need the use of an iterative approach but is an exact algorithm.

We now focus on Step 1 of the proposed algorithm. It involves the search of a vector of fixed size γ, minimizing the function $\mathrm{S}(\gamma)$ over the unit simplex

$$\Delta := \{\gamma : \mathrm{e}^{\mathrm{T}}\gamma = 1, \ \gamma \geq 0\}.$$

As pointed out [De Klerk et al., 2008], the optimization of a quadratic function over a simplex may be a difficult task to perform, but well-known schemes exist in order to perform this kind of problem, e.g., Genetic Algorithm, Accelerated Projected Gradient Descent, Exponentiated Gradient Descent. Note that, once we obtain the value γ^*, we can compute λ^* using the complementarity condition

$$\sum_{l\in\Omega_f}\delta_l\gamma_l = 0$$

Algorithm 12: Infinite Kernel Extreme Learning Machine

Begin

Given Ω, select $\Omega_f \subset \Omega$;

Step 1 Solve

$$\min_{\gamma} \ \mathrm{S}(\gamma)$$

$$\text{s. t. } \mathrm{e}^{\mathrm{T}}\gamma_l = 1, \ \ \gamma_l \geq 0$$

to obtain λ^*, α^*, the dual optimal solution of Problem (4.22);

where

$$\mathrm{S}(\gamma) := \min_{v,\xi} \ \frac{1}{2}\sum_{l\in\Omega_f}\frac{1}{\gamma_l}||v_l||^2 + \frac{C}{2}\sum_{i=1}^{n}||\xi_i||^2$$

$$\text{s. t. } \sum_{l=1}^{\tilde{l}}\langle v_l, \phi_l(x^i)\rangle_{\mathcal{H}} = y_i - \xi_i, \ \ \forall i = 1,\ldots,n$$

Step 2 Compute

$$\max_{l\in\Omega} \ T(l,\alpha^*)$$

where

$$T(l,\alpha) := \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j\langle\phi_l(x^i),\phi_l(x^j)\rangle_{\mathcal{H}}$$

if $T(l,\alpha^*) \leq \lambda^*$ **then**

break;

else

add l to Ω_f and return to Step 1;

end

end

together with the corresponding Karush–Kuhn–Tucker condition given by Equation (4.19b).

Finally, with respect to Step 2, the value of the function $T(l, \alpha^*)$ plays the role of some kind of evaluation threshold, specifying whether or not a new kernel needs to be selected.

As regard to the convergence of the proposed Infinite Kernel Extreme Learning Machine algorithm, we can use a direct application of Theorem 7.2 in [Hettich and Kortanek, 1993]. Specifically, as stated in the previous chapter by Theorem 11, we know that, if the problem admits solution, either Algorithm 12 ends after a finite number of steps, leading to a solution of the problem, or it has at least one accumulation point, each one of these solving the problem. Namely, similarly to [Gehler and Nowozin, 2008], the following statement is true. Note that, there is no assurance that, given a feasible problem, Algorithm 12 would identify a finite set Ω_f adequate to render the general set Ω. Anyway, if Algorithm 12 ends after a finite number of iterations, then Infinite Kernel Extreme Learning Machine algorithm converges, obtaining the optimal solution.

4.1.4 Numerical results

In order to explore the performance of the proposed Infinite Kernel Extreme Learning Machine, we present an experimental analysis on 20 public dataset, with the specific goal of tackling the execution of our proposed model in various dataset. Since the proposed model is a binary classification algorithm by nature, we decided to focus our attention on classification dataset, yet not taking into account the number of classes of the considered dataset, that is, tackling both binary and multi-class dataset. Specifically, when dealing with multi-class dataset, we decided to select a preset class and use the classification algorithm with respect to all the others, i.e., fixing a label and training the classifier against the examples of every other class. We decided to make this choice since the main goal of the proposed implementation is not to provide the most effective algorithm but to show the feasibility of the suggested Infinite Kernel Extreme Learning Machine. In particular, rather than aiming to get ambitious and vying results, we wish to achieve proper results, showing the fairness of our method from an implementation point of view.

Table 4.1 reports the name (first column), the number of examples (second column), the number of attributes (third column) and the type of problem (forth column) of the sets of data we utilized. Moreover, footnotes report the source of each dataset.

The algorithm starts with three fixed initial base kernel functions

$$k_1(x^i, x^j) = {x^i}^{\mathrm{T}} x^j,$$
$$k_2(x^i, x^j) = ({x^i}^{\mathrm{T}} x^j)^2,$$
$$k_3(x^i, x^j) = ({x^i}^{\mathrm{T}} x^j)^3,$$

i.e., the linear kernel and two polynomial kernels, respectively of degree 2 and 3 and null free parameters. For precision, it is fair to underline that, we decided to implement our code with the possibility of using $k_1,\ k_2,\ k_3$ as initial base kernels. In practice, however, we fixed the combination parameter for kernel k_3 equal to 0, and, therefore, in our experiments, polynomial kernel functions of degree 3 were never used.

The choice of further kernel functions is made as follows. We search for novel kernel functions among Gaussian kernels of the form

$$k_\mu(x^i, x^j) = e^{-\mu||x^i - x^j||^2/m},$$

where m is the size of the input space. Specifically, the following maximization problem is solved in order to compute the novel kernel combination parameters

$$\max_{\mu} \ \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i^* \alpha_j^* e^{-\mu||x^i - x^j||^2/m}$$
$$\text{s. t. } \ \mu_1 \leq \mu \leq \mu_2,$$

with α^* the dual optimal solution of Problem (4.22). Moreover, in our implementation we decided to fix the range of parameter μ in the interval $[0.1; 700]$. In this way, as described in the previous section, the proposed model searches for a finite number of kernels from a base kernel set of infinite dimension.

The program is coded in MatLab version R2021a and executed on a laptop. Appendix A contains the main parts of the code implemented by us and used to obtain the results reported in Table 4.2. It is important to emphasise that, our MatLab implementation is based on the *Infinite and Multiple Kernel Learning algorithm for MatLab* code by Peter Gehler and Sebastian Nowozin [Gehler and Nowozin, 2008]. Specifically, as well as the source code, our implementation utilizes the Interior Point Optimizer *Ipopt*, an open source software package for large-scale nonlinear optimization. Instead, unlike the original code, we did not use the Support Vector Machine solver *libsvm* and that is because, as deeply emphasizes in the previous sections, Extreme Learning Machine tackles the problem of finding the best combination parameters by considering the minimization of the approximation error, given by the difference between the expected output and the computed result, while at the same time looking for the smallest norm weights. In this way, Extreme Learning Machine has a very similar formulation with respect to classical Support Vector Machine. However, while Support Vector Machine framework requires an optimization algorithm, Extreme Learning Machine needs only to solve a system of equations.

The obtained results are shown in Table 4.2, where the achieved performance values for both the training set and the test set are shown. Specifically, we computed the accuracy, the precision and the recall values obtained in the training phase of the algorithm, as well as using a testing subset of the dataset. As is evident from the table, we achieved very high performance measures in the training set, with lowest values for the "phishing data" dataset with accuracy value of 0.969, precision value of 0.979 and recall value of 0.970. Weaker results were obtained for the test set. Looking at the accuracy of the test set, we can see that its value is in the range specified by the lowest value of 0.710 for the "cardiovascular" dataset and by the highest value of 0.980 for the "diabetes" dataset. Concerning the precision on the test set, we can observe that its lowest value of 0.640 is obtained on the "cardiovascular" dataset, while its highest result of 0.990 is obtained with the use of the "diabetes" dataset. Finally, as regards to the recall value, its best value of 0.977 is obtained for the "diabetes" dataset, while a limited result of 0.740 is achieved for the "pima indians" dataset.

Note that, the worst results were obtained with the "cardiovascular" dataset and the "heart" dataset.

As far as the "cardiovascular" dataset is concerned, it is a binary classification dataset and it contains over $68,000$ instances characterized by 11 attributes. The aim is to find the presence of cardiovascular disease on the basis of specific patients body attributes. Considering the huge number of instances, we decided to narrow the number of examples used to only $1,000$ balanced instances, with 500 examples belonging to class 1 and 500 examples belonging to class -1. Regardless a balanced number of both classes were selected, we strongly believe that chopping the number of instances in such a substantial way, may justify the weak results obtained in comparison with the other dataset.

With respect to the "heart" dataset, as specified in Table 4.1, it is given by 304 examples made of 13 attributes. Specifically, the aim is to determine, using the 13 body attributes of each candidate, whether or not a person suffers of heart disease. Although the "heart" dataset is a binary classification dataset, we obtained not as good results as for the other dataset. However, we observed that, comparing the results obtained in three different papers [Brown, 2004; K. Huang et al., 2004; Zhou and Jiang, 2004], the accuracy has value ranging from 0.789 [Zhou and Jiang, 2004] to 0.857 [K. Huang et al., 2004], results that are not all that far from those obtained using the Infinite Kernel Extreme Learning Machine method.

In conclusion, we may say that the values specified in Table 4.2 may be considered as a very promising achievement of our proposed method. In particular, it is important to underline that, in our implementation we did not perform any fine-tuning method, namely no transfer learning approach was used in order to improve the weakest parts of our code. Moreover, we decided to use standard optimization techniques, not paying attention to operating with more optimization processes, with the aim of picking

the best performance one. Finally, no changes were made to the standard Ipopt options, leaving them fixed throughout the entire experimental analysis. We decided to make these choices because, as already mentioned above, we were more interested in investigating the feasibility of out proposed method, more than generating racing results, on a par with other existing performance values.

Dataset	**Examples**	**Attributes**	**Type of problem**
abalone[1]	4177	8	multi-class
banknote[1]	1372	4	binary
cardiovascular[2]	68684 (only 1000 used)	11	binary
cryotherapy[1]	90	6	binary
diabetes[1]	2000	8	binary
divorce[1]	170	54	binary
glass[1]	214	9	multi-class
heart[1]	304	13	binary
ionosphere[1]	351	34	binary
iris[1]	150	4	multi-class
leaf[2]	340	15	multi-class
phishing data[1]	1354	9	multi-class
pima indians[2]	768	8	binary
seeds[1]	210	7	binary
sonar[3]	208	60	binary
breast cancer[1]	570	31	binary
wine[1]	178	13	multi-class
wine quality (red)[1]	1599	11	multi-class
wine quality (white)[1]	4898	11	multi-class

Table 4.1: Name, number of examples, number of attributes and number of target values of the dataset used in the evaluations.

[1] https://archive.ics.uci.edu/ml/datasets.php
[2] https://www.kaggle.com/datasets
[3] https://datahub.io/machine-learning/sonar

	Accuracy		**Precision**		**Recall**	
Dataset	Training	Test	Training	Test	Training	Test
abalone	0.998	0.812	0.999	0.907	0.998	0.858
banknote	1	1	1	1	1	1
cardiovascular	1	0.710	1	0.648	1	0.764
cryotherapy	1	0.9444	1	0.875	1	1
diabetes	1	0.980	1	0.99	1	0.977
divorce	1	0.971	1	1	1	0.945
glass	1	0.955	1	0.900	1	0.900
heart	1	0.771	1	0.640	1	0.762
ionosphere	1	0.943	1	1	1	0.923
iris	1	1	1	1	1	1
leaf	1	0.956	1	1	1	0.956
phishing data	0.969	0.882	0.979	0.884	0.970	0.896
pima indians	1	0.753	1	0.926	1	0.740
seeds	1	1	1	1	1	1
sonar	1	0.762	1	0.870	1	0.741
speech feature	1	0.907	1	0.800	1	0.800
breast cancer	1	0.978	1	0.932	1	1
wine	1	1	1	1	1	1
winequality (red)	1	0.844	1	0.857	1	0.857
winequality (white)	1	0.783	1	0.882	1	0.812

Table 4.2: Accuracy, precision and recall values of training and test sets of the dataset specified in Table 4.1 using Infinite Kernel Extreme Learning Machine algorithm.

4.2 Multi-Kernel Covariance Terms in Multi-Output Support Vector Machines

4.2.1 Multi-task learning

The problem of simultaneously solving more than one task is known as Multi-Task Learning (MTL) and has gained a significant amount of notoriety over the last years. Unlike classical supervised Machine Learning techniques, where a function is sought to map input values to a specific appropriate output, i.e., one continuous value for regression problems or one discrete label for classification problems, Multi-Task Learning aims, given a specific input, to learn multiple outputs at once. Specifically, the goal of Multi-Task Learning is to simultaneously learn multiple tasks, while considering all the possible existing relationships as well as discrepancies and resemblances between them. Therefore, for every given input, a multidimensional output is assigned, corresponding to each task of the problem. The main hypothesis of Multi-Task Learning is that each task, or at the very least a subset of them, has inter-correlation factors, making necessary the use of an algorithm that explicitly considers this correlation. Multi-Task Learning is often mistakenly considered to be the same or improperly swapped with another Machine Learning subfield, known as Multi-Output Learning (MOL). Differently from Multi-Task Learning, Multi-Output Learning associates each input to a single vector output value. Thus, in Multi-Output Learning several possible choices of outputs are given, but only one is finally paired to each input. Moreover, even if inter-output correlation factors are not excluded, they do not constitute a key factor of Multi-Output Learning.

In literature, a fair number of review articles exists, showing how to approach Multi-Task Learning in different research areas: multi-task regression with real value outputs [Borchani et al., 2015; Melki et al., 2017]; classifications problems [Bielza et al., 2011; Vembu and Gärtner, 2010]; overall frameworks moving the view on the whole prospective [D. Xu et al., 2019].

The simplest approach to solve a generic multi-task problem would be to consider every task as independent from the others and, therefore, separately solving every problem and, finally, combining the solutions found to obtain a general result. However, in this way, since independent problems are solved, it is very likely to ignore significant multi-task correlation factors, thus resulting with improper results not taking into account relationships between different tasks [Ben-David and Schuller, 2003]. As a direct consequence, doing so, all the benefits that make Multi-Task Learning particularly suitable for describing real problems, i.e., problems where the inner task correlation factors are not only present but also significantly important, are canceled. Of course, if the given multi-task problem is defined by tasks that are not related with each other, the method described above fits the purpose correctly. An attractive approach dealing with

Multi-Task Learning makes use of Reproducing Kernel Hilbert Spaces of vector–valued functions [Micchelli and Pontil, 2005]. In this case we talk of matrix–valued kernels, an extension of the idea of kernels, where a kernel matrix is defined with respect to every input value.

Over the last years, several approaches have been proposed for solving multi-task problems based on the typical Support Vector Machine framework. An interesting approach is the multi-task Support Vector Machine [Zhang et al., 2018], that combines multi task learning with data from different backgrounds; regularization based approaches such as [Evgeniou and Pontil, 2004] using minimization of regularization functionals can be also found in literature; moreover, a sequential minimal optimization approach [Cai and Cherkassky, 2012], decomposing the problem into a set of minimal subproblems, has also been proposed. In [S. Xu et al., 2014], starting from the basic formulation, a generalized multi-task Support Vector Machine framework is formulated and solved with the use of Krylow space methods.

In Gaussian Processes the Multi-Task Learning problem is solved with the use of multi-task covariance functions. Such functions are able to solve the critical issue of multi-task correlations looking at all the possible connections between different tasks. Specifically, this approach requires the covariance function to operate on a cross-correlation factor between tasks, in addition to the classical correlation factor of each single task. Several approaches were introduced over the years, e.g., in the geostatical environment [Cressie, 1992; Wackernagel, 2013] known under the name of *cokriging*. Several cokriging methods exist: simple, ordinary, collocated, and indicator. They mainly differ from each other on the way the mean value is exploited. For example, simple cokriging employs a clearly defined beforehand mean, making it globally constant throughout the whole process. On the contrary, ordinary cokriging opts for locally constant means based on specific points. Usually, such methods use the same covariance function for different tasks. Along the lines of multi-task Gaussian process, an innovative approach is proposed in [Melkumyan and Ramos, 2011], that will be explored in more detail below.

4.2.2 Related work

Let us now outline in details the multi-task covariance function template for Gaussian Processes presented in [Melkumyan and Ramos, 2011].

Consider a generic L-tasks training set $\mathcal{D}$, with generic elements $x_l^i \in \mathcal{X}$ and $y_l^i \in \mathcal{Y}$, respectively defining the i-th input with respect to the l-th task and the i-th output for the l-th task. Each task is associated with a different kernel function and multi-task factors are modelled with the use of cross covariance terms. Specifically, the paper relies on the idea of using a convolution structure for each single task: given their basis function ϕ_l, i.e., a smoothing kernel, a single task auto covariance function k_{ll} can be

written as follows

$$k_{ll}(x^i, x^j) = \int_{-\infty}^{+\infty} \phi_i(x^i - u)\phi_i(x^j - u)\,\mathrm{d}u, \quad l = 1, \ldots, L. \tag{4.25}$$

Generalizing Equation (4.25), we obtain the following form for the covariance matrix K, given by

$$K(x_l^i, x_s^j) = \int_{-\infty}^{+\infty} \phi_l(x_l^i - u)\phi_s(x_s^j - u)\,\mathrm{d}u, \tag{4.26}$$

where x_l^i is in the l-th task and x_s^j is in the s-th task. It is important to note that, the covariance function defined in Equation (4.26) has the characteristic of being a positive semidefinite (PSD) matrix. Therefore, Equation (4.26) may be exploited to tackle the construction of cross-covariance terms, i.e., elements k_{ls} with $l \neq s$, modelling the inter task dependencies. The main challenge of computing such cross-covariance terms is finding the corresponding basis functions ϕ_l, $l = 1, \ldots, L$.

A general form to compute such basis functions is given by the following

$$g(\tau) = \frac{1}{(2\pi)^{1/4}} \mathcal{F}_{s\to\tau}^{-1}[\sqrt{\mathcal{F}_{\tau\to s}[k(\tau)]}], \tag{4.27}$$

where $\tau = x^i - x^j$ and $\mathcal{F}_{\tau\to s}[k(\tau)]$ is the Fourier transformation of an arbitrary stationary covariance function $k(\tau)$.

Moreover, using the scheme just described, in [Melkumyan and Ramos, 2011] three examples of cross covariance functions are provided: the squared exponential, the Matérn and the sparse covariance functions. The exact formulations of the described functions are given by Equations (4.28a)-(4.28c).

$$k_{SE}(r, l_{SE}) = \exp\left[-\frac{1}{2}\left(\frac{r}{l_{SE}}\right)^2\right], \tag{4.28a}$$

$$k_M(r, l_M) = \left(1 + \frac{\sqrt{3}r}{l_M}\right)\exp\left(-\frac{\sqrt{3}r}{l_M}\right), \tag{4.28b}$$

$$k_S(r, l_S) = \left[\frac{2 + \cos(2\pi r/l_S)}{3}(1 - r/l_S) + \frac{1}{2\pi}\sin(2\pi r/l_S)\right]H(l_S - r), \tag{4.28c}$$

with $r = |x^i - x^j|$ and l_M, l_{SE} and l_S respectively length scale parameters for Matérn, squared exponential and sparse covariance functions. Moreover, H is the Heaviside step function defined in Chapter 2 and the Matérn covariance function is computed with kernel parameter $\nu = 3/2$. The auto covariance, i.e., when the same covariance function is used, and the cross covariance functions, i.e., when two different basis covariance functions are considered, are calculated. Note that, as it will be clarified in the next section, since we have utilized Matérn $3/2$ and squared exponential, we are going to present only the covariance terms including such functions.

Specifically, let us consider the definition of squared exponential and Matérn covariance functions. Using Equations (4.28a) and (4.28b), auto covariance and cross

covariance terms can be constructed. Using the squared exponential kernel defined in (4.28a) and Equation (4.25), we have that the corresponding auto covariance term may be expressed as

$$k_{SE_1\times SE_2}(r, l_{SE_1}, l_{SE_2}) = \sqrt{\frac{2l_{SE_1}l_{SE_2}}{l_{SE_1}^2 + l_{SE_2}^2}}\exp\left(-\frac{r^2}{l_{SE_1}^2 + l_{SE_2}^2}\right) \tag{4.29}$$

where $r = |x^i - x^j|$ as above and l_{SE_1} and l_{SE_2} are the respective length scales.
Analogously, using (4.28b) in Equation (4.25), the Matérn 3/2 auto covariance term is given by

$$k_{M_1\times M_2}(r, l_{M_1}, l_{M_2}) = \sigma_{M_1M_2}(l_{M_1}e^{-\sqrt{3}\frac{r}{l_{M_1}}} - l_{M_2}e^{-\sqrt{3}\frac{r}{l_{M_2}}}) \tag{4.30}$$

with $\sigma_{M_1M_2} = 2\sqrt{l_{M_1}l_{M_2}}/(l_{M_1}^2 l_{M_2}^2)$ and l_{M_1}, l_{M_2} lenght scales.
Finally, we can compute the squared exponential-Matérn cross covariance term using both (4.28a) and (4.28b) into Equation (4.25), obtaining the following expression

$$\begin{aligned} k_{SE\times M}(r, l_{SE}, l_M) = \sqrt{\lambda}(\pi/2)^{1/4}e^{\lambda^2}\Bigg[&2\cosh\left(\frac{\sqrt{3}r}{l_M}\right) \\ &- e^{\frac{\sqrt{3}r}{l_M}}\operatorname{erf}\left(\lambda + \frac{r}{l_{SE}}\right) - e^{-\frac{\sqrt{3}r}{l_M}}\operatorname{erf}\left(\lambda - \frac{r}{l_{SE}}\right)\Bigg] \end{aligned} \tag{4.31}$$

where $\lambda = \frac{\sqrt{3}l_{SE}}{2l_M}$ and erf is the Gauss error function defined as

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}\,\mathrm{d}t.$$

4.2.3 The proposed model

Firstly, it is important to specify that, we have decided to name it *multi-output Support Vector Machine*. Therefore, from now on, when we talk about multi-output dataset, we are actually considering data where every given input is assigned with a corresponding multidimensional output, with the further characteristic of inter-correlated output components.

To be able to use a kernel matrix as the ones defined by Equations (4.29)-(4.31) in a Support Vector Machine framework, we need to construct a multi-output Support Vector Machine, taking care of correlation factors of every output. Let us consider a generic dataset

$$\{x^i, y^i\}_{i=1}^n,$$

with $x^i = (x_1^i, \ldots, x_m^i)^{\mathrm{T}} \in \mathbb{R}^m$ and $y^i \in \{-1, +1\}^t$. Starting from the classical Support Vector Machine structure, our goal is to define a problem with a multi-task cross factor. To do so, we propose some modifications to the classical Support Vector Machine problem. As far as the use of kernel functions, we decided to consider more than one feature map: specifically, for every output $p = 1, \ldots, t$, we consider a feature map

$$\phi_p(\cdot) \in \mathbb{R}^l,$$

of fixed dimension l. Concerning the normal vector to the hyperplane w, we substitute it with a more complex formulation: from every couple of outputs $\{i, j\}$, $i, j = 1, \dots, t$, we consider a vector w_{ij} of dimension l, i.e., the dimension of each feature map, leading to a three dimensional tensor W of dimension $t \times t \times l$ with elements w_{kpq}. As an additional condition on W, we require $w_{ij.} = w_{ji.}$, $\forall i, j, = 1, \dots, t$. For a sparse representation of Support Vector Machine, w is substituted by its expansion in terms of the dual variable. In this case the objective function contains a term imposing sparsity of the solution. In the same spirit, similarly to classical Support Vector Machine problem, we propose to minimize the sum of the square of every element present in the three dimensional tensor W.

The optimization problem is the following [Marcelli and De Leone, 2020].

$$\begin{aligned} \min_{W,\xi} \quad & \frac{1}{2} \sum_{k,p=1}^{t} \sum_{q=1}^{l} w_{kpq}^2 + C \sum_{k=1}^{t} \sum_{i1}^{n} \xi_k^i \\ \text{s. t.} \quad & y_k^i \left(\sum_{p=1}^{t} \sum_{q=1}^{l} w_{kpq} \phi_{pq}(x^i) + \theta_k \right) \geq 1 - \xi_k^i \\ & w_{kpq} = w_{pkq}, \;\; k, p = 1, \dots, t, \;\; q = 1, \dots, l \\ & \xi_k^i \geq 0 \;\; i = 1, \dots, n, \;\; k = 1, \dots, t. \end{aligned} \tag{4.32}$$

Note that,

$$\sum_{p=1}^{t} \sum_{q=1}^{l} w_{kpq} \phi_{pq}(x^i) = \sum_{p=1}^{t} w_{kp.}{}^{\mathrm{T}} \phi_p(x^i),$$

where the notation ϕ_{pq} is used to describe the q-th component of vector ϕ_p.
The Lagrangian function associated to (4.32) is

$$\begin{aligned} \mathcal{L}(w, \xi, \lambda, \mu, \beta) = & \frac{1}{2} \sum_{k,p=1}^{t} \sum_{q=1}^{l} w_{kpq}^2 + C \sum_{k=1}^{t} \sum_{i=1}^{n} \xi_k^i \\ & - \sum_{i=1}^{n} \sum_{k=1}^{t} \lambda_k^i \left[y_k^i \left(\sum_{p=1}^{t} \sum_{q=1}^{l} w_{kpq} \phi_{pq}(x^i) + \theta_k \right) - 1 + \xi_k^i \right] \\ & + \sum_{k=1}^{t} \sum_{i=1}^{n} \mu_k^i \xi_k^i + \sum_{k,p=1}^{t} \sum_{q=1}^{l} \beta_{kpq} (w_{kpq} - w_{pkq}). \end{aligned} \tag{4.33}$$

For the sake of semplicity, from now on we are going to use the bare notation $\mathcal{L} = \mathcal{L}(w, \xi, \lambda, \mu, \beta)$. The Karush–Kuhn–Tucker conditions are given by the following

$$\frac{\partial \mathcal{L}}{\partial w_{kpq}} = 0 \Longrightarrow w_{kpq} = \sum_{i=1}^{n} \lambda_k^i y_k^i \phi_{pq}(x^i) - \beta_{kpq} + \beta_{pkq}, \tag{4.34a}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = 0 \Longrightarrow \sum_{i=1}^{n} \lambda_k^i y_k^i = 0, \tag{4.34b}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_k^i} = 0 \Longrightarrow C = \lambda_k^i - \mu_k^i. \tag{4.34c}$$

Substituting (4.34a)-(4.34c) into (4.33) we can rewrite the Lagrangian function as

$$\begin{aligned}\mathcal{L} = &-\frac{1}{2}\sum_{k,p=1}^{t}\sum_{q=1}^{l} w_{kpq}^2 + \sum_{k,p=1}^{t}\sum_{q=1}^{l} w_{kpq}\left(w_{kpq} - \sum_{i=1}^{n}\lambda_k^i y_k^i \phi_{pq}(x^i) - \beta_{kpq} + \beta_{pkq}\right) \\ &+ \sum_{k=1}^{t}\sum_{i=1}^{n}\xi_k^i\left(C - \lambda_k^i + \mu_k^i\right) - \sum_{k=1}^{t}\left(\sum_{i=1}^{n}\lambda_k^i y_k^i\right) + \sum_{i=1}^{n}\sum_{k=1}^{t}\lambda_k^i = \\ &-\frac{1}{2}\sum_{k,p=1}^{t}\sum_{q=1}^{l} w_{kpq}^2 + \sum_{i=1}^{n}\sum_{k=1}^{t}\lambda_k^i.\end{aligned} \tag{4.35}$$

Now, recalling from (4.34a) that

$$w_{kpq} = \sum_{i=1}^{n}\lambda_k^i y_k^i \phi_{pq}(x^i) - \beta_{kpq} + \beta_{pkq} \tag{4.36a}$$

$$w_{pkq} = \sum_{i=1}^{n}\lambda_p^i y_p^i \phi_{kq}(x^i) - \beta_{pkq} + \beta_{kpq}, \tag{4.36b}$$

and since $w_{kpq} - w_{pkq} = 0$, we have that

$$w_{kpq} = \frac{1}{2}\left(\sum_{i=1}^{n}\lambda_k^i y_k^i \phi_{pq}(x^i) + \sum_{i=1}^{n}\lambda_p^i y_p^i \phi_{kq}(x^i)\right). \tag{4.37}$$

Substituting (4.37) into (4.33) we finally obtain the ultimate form for the Lagrangian function of Problem (4.32), given by the following

$$\begin{aligned}\mathcal{L} = &\sum_{i=1}^{n}\sum_{k=1}^{t}\lambda_k^i - \frac{1}{2}\sum_{k,p=1}^{t}\left[\frac{1}{2}\left(\sum_{i=1}^{n}\lambda_k^i y_k^i \phi_p(x^i) + \sum_{i=1}^{n}\lambda_p^i y_p^i \phi_k(x^i)\right)\right]^2 = \\ = &\sum_{i=1}^{n}\sum_{k=1}^{t}\lambda_k^i - \frac{1}{8}\sum_{k,p=1}^{t}\left[\sum_{i=1}^{n}\sum_{i'=1}^{n}\lambda_k^i\lambda_k^{i'} y_k^i y_k^{i'}\phi_p(x^i)^{\mathrm{T}}\phi_p(x^{i'}) + \right. \\ &\left.\sum_{i=1}^{n}\sum_{i'=1}^{n}\lambda_p^i\lambda_p^{i'} y_p^i y_p^{i'}\phi_k(x^i)^{\mathrm{T}}\phi_k(x^{i'}) + 2\sum_{i=1}^{n}\sum_{i'=1}^{n}\lambda_p^i\lambda_k^{i'} y_p^i y_k^{i'}\phi_k(x^i)^{\mathrm{T}}\phi_p(x^{i'})\right] = \\ = &\sum_{i=1}^{n}\sum_{k=1}^{t}\lambda_k^i - \frac{1}{4}\sum_{k,p=1}^{t}\left[\sum_{i=1}^{n}\sum_{i'=1}^{n}\lambda_k^i\lambda_k^{i'} y_k^i y_k^{i'}\phi_p(x^i)^{\mathrm{T}}\phi_p(x^{i'}) + \right. \\ &\left.\sum_{i=1}^{n}\sum_{i'=1}^{n}\lambda_p^i\lambda_k^{i'} y_p^i y_k^{i'}\phi_k(x^i)^{\mathrm{T}}\phi_p(x^{i'})\right].\end{aligned} \tag{4.38}$$

Note that, since both indices k and p depend on dimension t, which was chosen not to change throughout the process, we have that

$$\sum_{i=1}^{n}\sum_{i'=1}^{n}\lambda_p^i\lambda_p^{i'} y_p^i y_p^{i'}\phi_k(x^i)^{\mathrm{T}}\phi_k(x^{i'}) = \sum_{i=1}^{n}\sum_{i'=1}^{n}\lambda_k^i\lambda_k^{i'} y_k^i y_k^{i'}\phi_p(x^i)^{\mathrm{T}}\phi_p(x^{i'}),$$

leading to the final formulation (4.38).

The Wolfe-dual problem of (4.32) is given by

$$\begin{aligned} \min_{\lambda} \quad & -\mathcal{L} \\ \text{s. t.} \quad & 0 < \lambda_k^i < C \\ & \sum_{i=1}^{n} \lambda_k^i y_k^i = 0, \quad k = 1, \dots, t \end{aligned} \tag{4.39}$$

with $\mathcal{L}$ specified as in Equation (4.38). Note that, both cross covariance and auto covariance terms are present in (4.38). The term $\phi_k(x^i)^{\mathrm{T}}\phi_k(x^{i'})$ is relative to the same output relations, while $\phi_k(x^i)^{\mathrm{T}}\phi_p(x^{i'})$ takes into account the inter output correlation factors. Moreover, note that since we are considering feature maps $\phi_p(\cdot) \in \mathbb{R}^l$, $\forall p = 1, \dots, t$, we are using the classical inner product notation.

Now, using the notation

$$\mathcal{K}_{pk}(x^i, x^{i'}) \coloneqq \phi_p(x^i)^{\mathrm{T}}\phi_k(x^{i'}),$$

for every $p, k = 1, \dots, t$, once Problem (4.39) is solved and a finite solution λ^* is found, the classifier function for a generic given input $\hat{x}$ is computed as follows

$$\begin{aligned} f_k(\hat{x}) =& \sum_{p=1}^{t} \left[\frac{1}{2} \sum_{i=1}^{n} {\lambda^*}_k^i y_k^i \phi_p(x^i) + {\lambda^*}_k^i y_k^i \phi_k(x^i) \right]^{\mathrm{T}} \phi_p(\hat{x}) + \theta_k \\ =& \frac{1}{2} \sum_{p=1}^{t} \sum_{i=1}^{n} \left[{\lambda^*}_k^i y_k^i \mathcal{K}_{pp}(x^i, \hat{x}) + {\lambda^*}_k^i y_k^i \mathcal{K}_{kp}(x^i, \hat{x}) \right] + \theta_k. \end{aligned} \tag{4.40}$$

4.2.4 Numerical results

In order to test our proposed multi-output Support Vector Machine, we had to look for available multi-task dataset that would be appropriate for our goal. As already mentioned, despite being an interesting research field from both the theoretical and the practical point of view, only a small number of publicly available multi-task dataset are present. To overcome this deficiency, in this study, we decided to use both real and synthetic set of data, in order to investigate a fair amount of data and to obtain information as much reliable as possible.

Specifically, at first, we used four real open-source collection of data freely available online from the Java open source library Mulan [Tsoumakas et al., 2009]: “jura”, “enb”, “andro”, “sf1”. Table 4.3 lists the used dataset, specifying the number of examples, features and targets of each dataset. Primarily, we needed to modify the dataset in order to be able to use them in our model and to get valid information. The dataset listed in Table 4.3 are commonly exploited in regression frameworks, since the outputs may assume a continuous range of values. In order to use them in our classification problem, we had to modify the output values. We proceeded in the following way. For each one of the given outputs we computed the corresponding mean value. If

Dataset	Examples	Features	Targets
jura	359	15	3
enb	768	8	2
andro	49	30	6
sf1	323	10	3

Table 4.3: Description of the list of the Mulan dataset used in the experimental phase. Each row represents a specific set of data. The second column defines the number of examples; the third column describes the number of features; the forth column represents the number of targets.

the original output of a sample is grater than the mean value, then the new discrete output value assigned to it is $+1$, otherwise, i.e., if the original value is less than the mean value, we assign the value -1. We apply such process to any output leading to a novel multi-output classification dataset. Moreover, as far as the "jura" dataset is concerned, since its feature ranges are widely dissimilar, we decided to normalize the corresponding data. Specifically, we applied a min-max normalization, re-scaling the features in the range $[0; 1]$ using the following general formula

$$\text{new value} = \frac{\text{old value} - \text{min value}}{\text{max value} - \text{min value}}.$$

Once done the data pre-processing, we designed a series of experiments to evaluate the efficiency of our model. For the "jura" and "sf1" dataset, since they consist of three outputs, we considered each possible combination of the three outputs, i.e., output 1 and output 2 (y_1 & y_2), output 1 and output 3 (y_1 & y_3), output 2 and output 3 (y_2 & y_3). In this way, we derived three dataset, each one made of two output values, from the original one. The "enb" dataset has only two outputs, therefore no modification were needed. Finally, "andro" dataset consists of six outputs. Here, we decided to randomly choose two outputs and apply the proposed method to them.

Since we were somehow obliged by the circumstances to merely use "jura", "enb", "andro", "sf1" as real dataset, it could be an uncertainly to analyse our proposed model based only on the results obtained using the dataset listed in Table 4.3. Moreover, we cannot precisely determine whether or not the considered tasks are related to each other. Therefore, it is not trivial to evaluate the obtained results and give an objective assessment to the implemented model.

To somehow solve this issue and obtain a more complete picture, we decided to perform further experiments with the goal of obtaining an in-depth understanding of the proposed model. To do so, we searched for supplementary existing multi-task dataset, in order to generate a more appropriate and complete study of our model. Specifically, we decided to test our model using 40 synthetic dataset obtained from the multi-target dataset presented by [Aguiar et al., 2019]. This synthetic dataset consists of a total of

648 sets of data and was generated using the framework presented by [Mastelini et al., 2018]. In particular, the creation of such synthetic dataset is made through the use of an algorithm generating the input dataset as well as an initial uncorrelated set of features y_1, y_2, y_3. The input dataset is defined by the number of instances, features and targets, together with the generating group, i.e., a random partition of the data having the same input to output linear mapping characteristics. Starting from the uncorrelated targets the final outputs are obtained using a combination of the initial targets y_1, y_2, y_3. Such combination may be identity, quadratic or cubic functions of y_1, y_2, y_3.

Table 4.5 presents a list of the used synthetic dataset, describing the number of features, the number of targets, the generating group and the percentage of instances affected by noise. Moreover, the last three columns show how the correlated targets are generated, starting from the initial values y_1, y_2, y_3.

To perform our model, we used a kernel matrix $\mathcal{K}$ given by

$$\mathcal{K} = \begin{pmatrix} \mathcal{K}_{M\times M} & \mathcal{K}_{SE\times M} \\ \mathcal{K}_{M\times SE} & \mathcal{K}_{SE\times SE} \end{pmatrix},$$

where $\mathcal{K}_{SE\times M} = \mathcal{K}_{M\times SE}$, using Equations (4.29), (4.30) and (4.31). Therefore, when auto-covariance functions are required, $\mathcal{K}_{M\times M}$ and $\mathcal{K}_{SE\times SE}$ are used, corresponding to elements in the main diagonal. Otherwise, if cross-covariance terms are needed, the elements in the secondary diagonal are used.

The program is coded in MatLab version R2017b and executed on a laptop. Appendix A contains the main parts of the code implemented by us and used to obtain the achieved results. The dual constrained optimization Problem (4.39) is solved using the MatLab *fmincon* function. The computational cost incurred for the proposed method may be considered comparable with respect to other well known kernel methods. After the optimal solution λ^* has been calculated, classifier (4.40) is used for the final classification.

The results for the initial four real open-source collection of data listed in Table 4.3 are shown in Table 4.4, with a list of the achieved values for accuracy, precision and recall. We obtained the best results with the "enb" dataset, achieving an accuracy value of 0.99, a precision of 0.95 and a recall of 0.97. On the contrary, when we tackled the other real dataset, we obtained very different results, from a precision of 0.95 for the "jura" dataset to an accuracy of 0.40 for the "sf1". It is important to underline that, "enb" dataset is the only one having two targets from the beginning and it is the one with the biggest number of examples. In our opinion, that may explain the reason why our proposed model works so well with respect to it.

Table 4.6 describes the obtained accuracy, precision and recall results for the synthetic collection of data described in Table 4.5. It is easy to notice that, the achieved results are not homogeneous but, on the contrary, tend to have various values

Dataset		Accuracy	Precision	Recall
	y1 & y2	0.60	1	0.60
jura	y1 & y3	0.60	0.92	0.61
	y2 & y3	0.60	0.95	0.61
enb		0.99	0.95	0.97
andro		0.70	0.64	1
	y1 & y2	0.40	0.40	0.88
sf1	y1 & y3	0.63	0.60	0.94
	y2 & y3	0.63	0.62	0.98

Table 4.4: Accuracy, precision and recall values of the set of data described in Table 4.3.

from case to case. Specifically, the accuracy varies from the lowest value of 0.40 for "id 13" and "id 39" to the highest value of 0.850 for "id 4"; the precision value range is between 0.10 for "id 24" and 0.980 for "id 5"; recall value goes from worst result of 0.40 of "id 39" and best result of 0.99 of "id 32". Therefore, to get a general view on the reasons of such diversity of the results, we decided to plot the dataset having the best outcome values and the dataset with the worst corresponding results. Specifically, we focused on dataset "500s15f3t1g0.01py1-y1+y2-y1+y3", i.e., "id 4", corresponding to one of the best results, and on dataset "500s15f3t2g0.05py1y2y3", i.e., "id 24", that presents one of the lowest values. It is evident that, in Figure 4.4(a) points are very much correlated with each other, lying almost along a straight line. On the contrary, Figure 4.4(b) shows mostly uncorrelated or weakly correlated points, located randomly in the plane. Our proposed model seems to obtain good results when dealing with highly correlated inputs; when uncorrelated data are considered, the proposed multi-output Support Vector Machine would appear not to perform well.

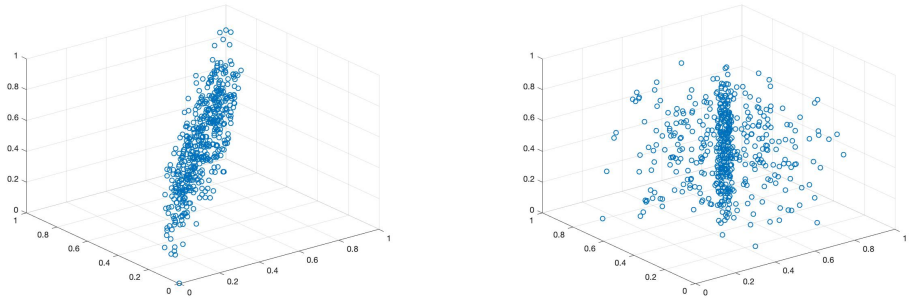
It is important to stress that, during the implementation phase of our method, we struggled finding existing multi-task classification dataset. Precisely for this reason, we were forced to use only four set of real data, i.e., "jura", "enb", "andro", "sf1", that do not exactly fit the purposes we aimed to tackle, and 40 synthetic dataset. Therefore, as we specified above, we had to carry on several pre-processing techniques in order to be able to use such sets of data in our proposed model. Moreover, it should be noted that, our main goal was to analyse the proposed method for inter correlated multi-task dataset. Hence, it is fair to emphasize that, the aim of the implementation phase was to demonstrate the feasibility of the method rather than obtain competitive and ambitious results.

Dataset	Features	Group	Noise	Target 1	Target 2	Target 3
id 1	15	2	10%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 2	15	2	1%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 3	30	2	1%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 4	15	1	1%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 5	15	1	10%	y_1	y_1^2	y_1^3
id 6	15	1	1%	y_1	y_1^2	y_1^3
id 7	15	1	10%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 8	15	1	1%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 9	15	1	10%	y_1	y_2	y_3
id 10	15	1	5%	y_1	y_2	y_3
id 11	15	1	5%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 12	15	1	5%	y_1	y_1^2	y_1^3
id 13	15	1	5%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 14	15	1	5%	y_1	y_2	y_3
id 15	15	2	1%	y_1	y_1^2	y_1^3
id 16	15	2	10%	y_1	y_1^2	y_1^3
id 17	15	2	1%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 18	15	2	10%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 19	15	2	1%	y_1	y_2	y_3
id 20	15	2	10%	y_1	y_2	y_3
id 21	15	2	5%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 22	15	2	5%	y_1	y_1^2	y_1^3
id 23	15	2	5%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 24	15	2	5%	y_1	y_2	y_3
id 25	30	1	1%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 26	30	1	10%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 27	30	1	1%	y_1	y_1^2	y_1^3
id 28	30	1	10%	y_1	y_1^2	y_1^3
id 29	30	1	1%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 30	30	1	10%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 31	30	1	1%	y_1	y_2	y_3
id 32	30	1	10%	y_1	y_2	y_3
id 33	30	1	5%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 34	30	1	5%	y_1	y_1^2	y_1^3
id 35	30	1	5%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$
id 36	30	1	5%	y_1	y_2	y_3
id 37	30	2	1%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 38	30	2	10%	y_1	$y_1 + y_2$	$y_1 + y_3$
id 39	30	2	1%	y_1	y_1^2	y_1^3
id 40	30	2	1%	y_1	$y_1^2 + y_2^2$	$y_1^2 + y_3^2$

Table 4.5: A general overview of the proposed synthetic dataset consisting of 40 cases, each one containing 500 examples. Each row represents a specific set of data. The second column defines the number of features; the third column represents the generating group; the forth column provides the percentage of instances effected by noise; the last three columns describe how the given targets are built. Every considered dataset is defined by three target values.

		Accuracy			Precision			Recall		
Id	**Dataset**	y1 & y2	y1 & y3	y2 & y3	y1 & y2	y1 & y3	y2 & y3	y1 & y2	y1 & y3	y2 & y3
id 1	500s15f3t2g0.1py1-y1+y2-y1+y3	0.715	0.640	0.640	0.665	0.590	0.275	0.760	0.655	0.894
id 2	500s15f3t2g0.01py1-y1+y2-y1+y3	0.810	0.825	0.660	0.737	0.845	0.435	0.789	0.782	0.790
id 3	500s15f3t1g0.1py1-y1+y2-y1+y3	0.620	0.575	0.650	0.790	0.460	0.895	0.650	0.540	0.680
id 4	500s15f3t1g0.01py1-y1+y2-y1+y3	0.850	0.825	0.795	0.730	0.695	0.812	0.960	0.945	0.825
id 5	500s15f3t1g0.1py1-y1e2-y1e3	0.610	0.780	0.60	0.635	0.845	0.980	0.660	0.656	0.540
id 6	500s15f3t1g0.01py1-y1e2-y1e3	0.710	0.600	0.500	0.806	0.300	0.105	0.695	0.940	0.625
id 7	500s15f3t1g0.1py1-y1e2+y2e2-y1e2+y3e2	0.500	0.640	0.580	0.200	0.680	0.925	0.990	0.710	0.560
id 8	500s15f3t1g0.01py1-y1e2+y2e2-y1e2+y3e2	0.740	0.630	0.450	0.690	0.625	0.280	0.765	0.820	0.740
id 9	500s15f3t1g0.1py1y2y3	0.775	0.710	0.750	0.700	0.80	0.835	0.895	0.700	0.715
id 10	500s15f3t1g0.01py1y2y3	0.665	0.615	0.800	0.400	0.515	0.715	0.940	0.830	0.840
id 11	500s15f3t1g0.05py1-y1+y2-y1+y3	0.895	0.730	0.720	0.925	0.990	0.450	0.875	0.645	0.905
id 12	500s15f3t1g0.05py1-y1e2-y1e3	0.570	0.792	0.680	0.555	0.810	0.660	0.655	0.735	0.700
id 13	500s15f3t1g0.05py1-y1e2+y2e2-y1e2+y3e2	0.710	0.750	0.400	0.700	0.855	0.130	0.76	0.745	0.700
id 14	500s15f3t1g0.05py1y2y3	0.710	0.635	0.660	0.800	0.470	0.565	0.800	0.895	0.835
id 15	500s15f3t2g0.1py1-y1e2-y1e3	0.590	0.635	0.640	0.610	0.930	0.990	0.690	0.575	0.595
id 16	500s15f3t2g0.01py1-y1e2-y1e3	0.550	0.530	0.700	0.430	0.595	0.760	0.655	0.715	0.795
id 17	500s15f3t2g0.1py1-y1e2+y2e2-y1e2+y3e2	0.480	0.658	0.655	0.500	0.615	0.925	0.520	0.660	0.670
id 18	500s15f3t2g0.01py1-y1e2+y2e2-y1e2+y3e2	0.520	0.67	0.615	0.700	0.695	0.755	0.520	0.620	0.700
id 19	500s15f3t2g0.1py1y2y3	0.675	0.695	0.630	0.725	0.340	0.605	0.625	0.835	0.760
id 20	500s15f3t2g0.01py1y2y3	0.660	0.605	0.640	0.545	0.970	0.690	0.655	0.560	0.635
id 21	500s15f3t2g0.05py1-y1+y2-y1+y3	0.600	0.555	0.685	0.840	0.605	0.460	0.595	0.655	0.870
id 22	500s15f3t2g0.05py1-y1e2-y1e3	0.520	0.635	0.550	0.410	0.830	0.550	0.715	0.665	0.775
id 23	500s15f3t2g0.05py1-y1e2+y2e2-y1e2+y3e2	0.640	0.650	0.410	0.500	0.760	0.280	0.790	0.690	0.750
id 24	500s15f3t2g0.05py1y2y3	0.410	0.670	0.560	0.100	0.550	0.560	0.800	0.790	0.745
id 25	500s30f3t1g0.1py1-y1+y2-y1+y3	0.670	0.745	0.810	0.400	0.740	0.790	0.945	0.810	0.845
id 26	500s30f3t1g0.01py1-y1+y2-y1+y3	0.830	0.755	0.710	0.885	0.865	0.460	0.815	0.795	0.435
id 27	500s30f3t1g0.1py1-y1e2-y1e3	0.720	0.775	0.730	0.850	0.725	0.840	0.705	0.825	0.750
id 28	500s30f3t1g0.01py1-y1e2-y1e3	0.510	0.815	0.730	0.250	0.735	0.875	0.650	0.790	0.645
id 29	500s30f3t1g0.1py1-y1e2+y2e2-y1e2+y3e2	0.665	0.650	0.600	0.810	0.570	0.700	0.675	0.810	0.670
id 30	500s30f3t1g0.01py1-y1e2+y2e2-y1e2+y3e2	0.735	0.545	0.470	0.680	0.425	0.400	0.865	0.750	0.800
id 31	500s30f3t1g0.1py1y2y3	0.760	0.780	0.750	0.835	0.660	0.785	0.750	0.815	0.770
id 32	500s30f3t1g0.01py1y2y3	0.650	0.790	0.840	0.300	0.600	0.830	0.990	0.800	0.890
id 33	500s30f3t1g0.05py1-y1+y2-y1+y3	0.690	0.810	0.805	0.980	0.740	0.875	0.625	0.865	0.760
id 34	500s30f3t1g0.05py1-y1e2-y1e3	0.590	0.605	0.605	0.435	0.905	0.905	0.800	0.525	0.525
id 35	500s30f3t1g0.05py1-y1e2+y2e2-y1e2+y3e2	0.640	0.555	0.630	0.750	0.455	0.640	0.655	0.820	0.670
id 36	500s30f3t1g0.05py1y2y3	0.705	0.760	0.695	0.815	0.930	0.505	0.755	0.690	0.910
id 37	500s30f3t2g0.1py1-y1+y2-y1+y3	0.635	0.680	0.575	0.810	0.655	0.825	0.670	0.725	0.520
id 38	500s30f3t2g0.01py1-y1+y2-y1+y3	0.715	0.600	0.745	0.930	0.855	0.750	0.670	0.590	0.730
id 39	500s30f3t2g0.1py1-y1e2-y1e3	0.410	0.500	0.400	0.300	0.640	0.580	0.650	0.450	0.400
id 40	500s30f3t2g0.1py1-y1e2+y2e2-y1e2+y3e2	0.570	0.470	0.490	0.400	0.270	0.425	0.740	0.620	0.700

Table 4.6: Accuracy, precision and recall values of the set of data described in Table 4.5.



(a) Plot of the dataset 500s15f3t1g0.01py1-y1+y2-y1+y3

(b) Plot of the dataset 500s15f3t2g0.05py1y2y3

Figure 4.4: Figure (a) plots dataset "500s15f3t1g0.01py1-y1+y2-y1+y3", i.e., "id 4", corresponding to one of the best obtained results. Figure (b) plots dataset "500s15f3t2g0.05py1y2y3", i.e., "id 24", corresponding to one of the worst obtained results. As is evident, in (a) a very strong correlation between points is present; on the contrary, (b) shows points scattered in the plan, leading to strongly uncorrelated inputs.

Conclusions

This thesis work had the primary objective to tackle and report the theory behind kernel learning, particularly in relation to supervised learning algorithms.

To this end, we started defining what is meant by the term Machine Learning and how it developed in the course of time, while paying strong attention to the pivotal events which are still a key feature of such research field. After that, basic properties and learning rules at the basis of the three main Machine Learning types of learning are considered, with a specific focus on the motivation and background concepts at their base. Subsequently, particular attention is given to the common thread of this work: kernel methods. Kernel methods define a set of Machine Learning techniques all having in common the use of positive definite kernel functions. In this way, initially defined linear models may be employed in tackling non-linearly separable dataset, making use of a Reproducing Kernel Hilbert Space, consequently without forsaking the many advantages linked to the use of linear estimators, both from analytic and computational points of view. Since, by definition, kernel methods are opposed to linear models, a good portion of this thesis work was dedicated to outline both these Machine Learning key categories, paying special attention to their respective most popular and known algorithms. Moreover, we defined the theory behind two extensions of classical kernel based methods: Multiple Kernel Learning and Infinite Kernel Learning. Specifically, we started from the hypothesis of each of the two approaches and, later, we proposed two practical examples in the context of Support Vector Machines, respectively in the case of using a finite set of predefined base kernel functions and possibly infinitely many kernels. Finally, this thesis concluded with a thorough description of two novel results, both based on the idea of employing more than one kernel function to improve the model obtained.

Chapter 1 has set the stage for everything that follows. We started describing a timeline of the most important Machine Learning related events, both from a theoretical and a more practical point of view. After that, we described the three classical Machine Learning methods, i.e., supervised learning, unsupervised learning and reinforcement learning, analyzing the different learning strategies each corresponding method must follow. In addiction, an in-depth analysis of kernel methods is given: starting from

the notion of Hilbert space, the formal definition of positive definite kernel is given, together with its basic properties and a substantial number of well-known and used examples. After that, the Moore-Aronszajn Theorem is presented, showing that each positive definite kernel function is uniquely connected to a Reproducing Kernel Hilbert Space. Finally, the Mercer's theorem is introduced, showing an alternative way to describe feature maps, based on the notion of integral operator of a kernel function.

In Chapter 2 linear learning algorithms are tackled. As in the previous chapter, a division between the three main Machine Learning categories was made. Specifically, first linear classification methods were described: starting from the notion of separating hyperplane, a description of the most popular and longer used linear classification algorithms was given. In particular, Perceptron, Support Vector Machine and Logistic Regression algorithms were tackled, describing their learning rule. After that, classical Linear Regression method was considered and the least mean square method was used to obtain the best regression parameters. Finally, the Linear Discriminant Analysis method was considered in the context of linear unsupervised learning, underlying the similarities and differences existing between such method and Support Vector Machine algorithm.

Chapter 3 outlined various kernel learning methods together with their computational techniques. As regard to supervised learning algorithms, Support Vector Machines and Gaussian Processes were considered, describing the corresponding kernel based problems thoroughly. Moreover, the kernel Principal Component Analysis algorithm was considered, extending the well-known unsupervised technique to kernel learning. After that, ample space was given to the problem of considering more than one kernel function. Specifically, Multiple Kernel Learning was described, dwelling on a Multiple Kernel Support Vector Machine formulation. Then, Infinite Kernel Learning algorithm was considered, still giving a detailed description of the corresponding Infinite Kernel Support Vector Machine.

Finally, Chapter 4 presented two of the most significant obtained results. First, a novel method called "Infinite Kernel Extreme Learning Machine" was described. The concept behind it was to merge together a pre-existing supervised learning algorithm, called Extreme Learning Machine, together with the idea of using a combination of possibly infinitely many kernel function. Specifically, after a description of the original method was given, an overview on the original paper that inspired us on the implementation of the proposed method was proposed. Finally, the proposed model was tackled, leading to an original two-step algorithm. The method was tested on 20 dataset and the achieved outcomes were outlined in a table, describing the obtained accuracy, precision and recall values with respect to both the training and the test set. Then, the second result called "Multi-Kernel Covariance Terms in Multi-Output Support Vector Machines" was proposed. Here, the basic idea was to use an existing multi-task kernel structure implemented for Gaussian Processes in a novel multi-task Support Vector

Machine framework. Again, first special attention was given to a description of the paper that inspired us. After that, the proposed result was described, with particular care on the necessary calculations leading to the obtained novel framework. Lastly, our proposed model was tested on four real open-source dataset and on 20 synthetic dataset. The obtained results were described, analysing the achieved accuracy, precision and recall values, stressing out the strengths as well as the limitations of our proposed method.

Ringraziamenti

Dal punto di vista didattico, desidero ringraziare il professor Renato De Leone, non solo per avermi dato la possibilità di intraprendere questo nuovo percorso di studi ma per i suoi preziosi consigli, la chiarezza e la pazienza che mi hanno aiutata e sostenuta nel corso di questi tre anni.

Ringrazio mia madre e mio padre, per aver sempre creduto in me e nei miei sogni e per il sostegno e l'incoraggiamento costante di cui non sembro aver bisogno ma che si è rivelato fondamentale.

Ringrazio i miei amici, preziosi compagni di mille avventure e cenette indimenticabili, senza le quali tutto sarebbe stato più grigio e incolore.

Per ultimo, ma non per importanza, ringrazio Laura, che non ha mai smesso di credere in me. Grazie per essermi sempre stata accanto e per avermi aiutato a vedere le cose nella giusta prospettiva.

Appendix A

MatLab codes

Infinite Kernel ELM

Infinite Kernel Learning algorithm

```
function [Ks, beta, mu, alpha, theta, lambda] = IKELMmain(X, y, C, opts)
    %
    % Infinite Kernel Learning algorithm
    %
    % Usage:
    %
    % [Ks, beta, mu, alpha, lambda] = IKELMmain(X, y, C, opts)
    %
    % Input:
    %
    % X                    trainig set
   %  y                     training labels (vector of dimension L)
    %  subproblem_func function handle for the subproblem
    % C                    regularing constant
    %
    % Optional Input:
    %
    %     opts - struct with the following possible fields (with
    %     default values)
    %    .maxiter (100) - Maximum number of iterations. One
    %    iteration is the evaluation of the subproblem and
    %    the masterproblem
    %    .maxiter_ipopt (100) - Maximum number of iterations of
```

```
    %    IPopt to solve the master problem
    %    .mkl_eps (1e-4) - Precision for the MKL objective
    %    .svm_eps (1e-8) - Precision of the SVM solver
    %    .decay_eps (1e-5) - Weights below this value are
    %    pruned away
    %    .objective_eps (1e-5) - Convergence threshold. If
    %    the relative decrease of the objective value is
    %    below this threshold, the program is stopped
    %
    % Output:
    %
    %    Ks              the set of kernels
    %    beta            coefficients for the kernels. The final kernels
    %                    is a linear combination of the kernels in Ks with
    %                    coefficients beta
    %    alpha, lambda optimal alpha dual variables for the subproblem
    %
    % Compute distances in each dimension separately
    %
    D = zeros(size(X,1),size(X,1),size(X,2));
    for k = 1:size(X,2)
        D(:,:,k) = dist_euclid(X(:,k),X(:,k));
    end
    %
    % Calculate an initial starting point
    %
    Ks = [ ];
    [Ks, mu] = Kernel_Init(X, D, opts);
    %
    % Set initial values for beta
    %
    beta = ones(length(mu)+3,1)/(length(mu)+3);
    %
    % Determine best kernels and their best combination
    %
    [Ks, beta, mu, alpha, theta, lambda] = Inf_Kernel_Problem(X, y, K, beta,
        mu, D, C, opts);
end
```

Initial Kernels

```
function [K, mu] = Kernel_Init(X, D, opts)
%
%  Construct initial  kernels
%  There are 3 polynomial kernels and Gaussian kernels with parameters mu
%
K = [ ];
K(:,:,1)  =  X*X';
K(:,:,2)  = (X*X').^2;
K(:,:,3)  = (X*X').^3;
non_exp = 3;
scales  = linspace(opts.range_mu(1),opts.range_mu(2)/10,opts.exp_kern);
nFact = size(D,3);
for k = 1:opts.exp_kern
    K(:,:, non_exp+k) = exp(-scales(k)*sum(D,3)/nFact);
end
mu = scales';
end
```

Infinite Kernel problem

```
function [Ks, beta, mu_values, alpha, theta, lambda] = Inf_Kernel_Problem(X,
    y, Ks, beta_init, mu_values, D, C, opts)
    %
    %   Infinite  Kernel problem solver
    %
    exit  = -1;
    outer_iter = 0;

    while exit < 0
        fprintf('  beta = ');
        disp(beta_init');
        fprintf('  mu = ');
        disp(mu_values');

        %
        % Increase outer  iterations  count
        %
        outer_iter = outer_iter + 1;
```

```
%
% Determine best values for beta multipliers
%
[beta, alpha, theta, lambda] = Best_Convex_Combination(X, y, Ks,
    beta_init, C, opts);

%
% Set up the auxiliary data
%
options.auxdata = {alpha, D};

%
% Define functions
%
funcs.objective         = @outer_prob_objective;
funcs.gradient          = @outer_prob_gradient;
funcs.hessian           = @outer_prob_Hessian;
funcs.hessianstructure  = @outer_prob_Hessian_struct;
funcs.constraints       = @outer_prob_constraints;
funcs.jacobian          = @outer_prob_jacobian;
funcs.jacobianstructure = @outer_prob_jac_struct;

%
% Set options
%
options.lb = opts.range_mu(1);
options.ub = opts.range_mu(2);
options.ipopt.mu_strategy      = 'adaptive';
options.ipopt.print_level      = 0;
options.ipopt.tol              = opts.outer_tol;
options.ipopt.max_iter         = opts.outer_max_iter;
options.ipopt.derivative_test  = 'second-order';

%
% Call ipopt
%
mu_init = mu_values(length(mu_values));
f_init  = -outer_prob_objective(mu_init, options.auxdata);
[mu,info] = ipopt_auxdata(mu_init, funcs, options);
```

```
        f_final   = -outer_prob_objective(mu, options.auxdata);
        fprintf(' *** outer problem initial value = %g \t final value = %g %g\n',
              f_init, f_final, f_final-lambda);
        fprintf('   new value of mu = %f \n', mu);

        %
        % Check outer loop convergence or add new Kernels
        %
        if outer_iter > opts.outer_max_iter
            exit = 1;
        end
        if f_final - lambda < opts.outer_tol
           fprintf(" Outer loop convergence %g \n", f_final - lambda);
           exit = 0;
        end

        if exit < 0
            %
            % Calculate new Ks
            %
            F = size(beta,1);
            nFact = size(D, 3);
            Ks (:,:, F+1) = exp(-mu*sum(D, 3)/nFact);
            mu_values = [mu_values;mu];
            beta_init = beta;
            beta_init(F+1) = 0;
        end
    end
    fprintf(' beta = ');
    disp(beta');
    fprintf(' mu = ');
    disp(mu_values');
end

function f = outer_prob_objective(mu, auxdata)
    %
    % Objective function for outer problem
    %
    [alpha,D] = deal(auxdata{:});
    f = outer_function(mu, alpha, D);
```

```
end

function df = outer_prob_gradient(mu, auxdata)
    %
    %  Gradient function for outer problem
    %
    [alpha,D] = deal(auxdata{:});
    [foo, df] = outer_function(mu, alpha, D);
end

function H = outer_prob_Hessian(mu, sigma, lambda, auxdata)
    %
    % Hessian function for outer problem
    %
    [alpha,D] = deal(auxdata{:});
    [foo, dfoo, H] = outer_function(mu, alpha, D);
    H = sparse(H);
end

function H = outer_prob_Hessian_struct(auxdata)
    %
    %  Hessian structure for outer problem
    %
    H = sparse(1);
end

function [f, df, H] = outer_function(mu, alpha, D)
    %
    %  Calculate outer objective function, gradient and Hessian
    %
    %  Calculate objective function
    %
    nFact = size(D, 3);
    K1 = exp(-mu*sum(D, 3)/nFact);
    f = -0.5*alpha'*K1*alpha;
    if nargout > 1
       %
       % Calculate gradient with respect to mu
       %
       K2 = exp(-mu*sum(D, 3)/nFact).*(-sum(D, 3)/nFact);
```

```
        df = -0.5*alpha'*K2*alpha;
        if nargout > 2
            %
            % Calculate Hessian
            %
            K3 = exp(-mu*sum(D, 3)/nFact).*(sum(D, 3)/nFact).^2;
            H = -0.5*alpha'*K3*alpha;
        end
    end
end
```

Multi-Kernel Covariance Terms in Multi-Output SVM

Kernel matrix function

```
function [K] = KernelMatrix(X)
%
% Compute the Kernel matrices according to Melkumyan, Ramos Multi-Kernel
    Gaussian Processes
%
% Input values:
%    X the training set
%
% Output Value:
%    K the tensor of kernels
%        K(.,., p,k) gives the element in position (p,k) of the kernel matrix K

%
% Compute norm(x^i, x^j), i, j = 1, .. l
%
l = size(X,1);
for i=1:l
    D(i, i) = 0.0;
    for j=i+1:l
        D(i, j) = norm(X(i,:)-X(j,:));
        D(j, i) = D(i, j);
    end
end
```

```
K = [ ];

for i=1:2
    for j=1:2
        %
        % Compute Kernel Squared Exponential - Squared Exponential
        %   formula (21) in Melkumyan, Ramos Multi-Kernel Gaussian Processes
        %
        if i==1 && j==1
            l1 = 2;
            l2 = 2;
            den    = l1*l1+l2*l2;
            factor = sqrt((2*l1*l2)/den);
            K(:,:,i,j)=zeros(size(D));
            K(:,:,i,j)= factor*exp(-D.^2/den);
        end
        %
        % Compute Kernel Matern - Matern
        %   formula (15) in Melkumyan, Ramos Multi-Kernel Gaussian Processes
        %
        if i==2 && j==2
            l1=1.1;
            l2=2;
            sigma=2*sqrt(l1*l2)/(l1^2-l2^2);
            factor1=-sqrt(3)*D/l1;
            factor2=-sqrt(3)*D/l2;
            K(:,:,i,j)=sigma*(l1*exp(factor1)-l2*exp(factor2));
        end
        %
        % Compute Kernel Squared Exponential - Matern
        %   formula (14) in Melkumyan, Ramos Multi-Kernel Gaussian Processes
        %
        if i~=j
            lm     = 1.1;
            lse    = 1.5;
            lambda = sqrt(3)*lse/(2*lm);
            factor = sqrt(lambda)*(pi/2)^(1/4)*exp(lambda*lambda);
            K(:,:,i,j) = factor*(2*cosh(sqrt(3)*D/lm) - exp(sqrt(3)*D/lm).*
                erf(lambda+D/lse) ...
                    - exp(-sqrt(3)*D/lm).*erf(lambda-D/lse));
```

```
        end
    end
end
end
```

Constraint Matrix function

```
function [Aeq] = ConstraintMatrix(Y)
%
% Compute the Linear Constraint matrix
%
% Input values:
%     Y training labels
%
% Output Value:
%     Aeq Linear Constraint matrix
%

m = size(Y,2);
N= size(Y,1);
Aeq= [ ];
d=zeros(1,N);
for i=1:m
    if i==1
        Aeq(i,:)=[Y(:,i)' zeros(1,m*N-N)];
    elseif i==m
        Aeq(i,:)=[zeros(1,m*N-N) Y(:,i)'];
    else
        Aeq(i,:)=[repmat(d,1,i-1) Y(:,i)' repmat(d,1,N*m-i*N)];
    end
end
end
```

Objective function

```
function [f, gradf]= objectiveFunction(theta, K, Y)
%
```

```
% Compute the objective function according to notes
%
% Input Values:
%     theta is a column vector given by all the rows of lambda, lambda(i,k)==
    theta((k-1)*N+i)
%     K kernel matrix
%     Y training labels
%
% Output Values:
%     f Lagrangian function to be optimized
%     gradf Gradient of f
%

%N number of data equal to the number of rows (= columns) of K
N=size(K,1);

%m number of outputs, theta is a column vector of N*m components
m=size(theta,1)/N;

s=0;
t=0;

for k=1:m
    for p=1:m
        for i=1:N
            for j=1:N
            %case for k=p => auto covarinace terms
            %s=s+(lambda(i,k)*lambda(j,k)*Y(i,k)*Y(j,k))*K(i,j,1);
            s=s+(theta((k-1)*N+i)*theta((k-1)*N+j)*Y(i,k)*Y(j,k))*K(i,j,p,p);
            end
        end
    end
end

for k=1:m
    for p=1:m
        for i=1:N
            for j=1:N
                %case where k=p => cross covariance terms
                %t=t+(lambda(i,p)*lambda(j,k)*Y(i,p)*Y(j,k))*K(i,j,2);
```

```
                t=t+(theta((p-1)*N+i)*theta((k-1)*N+j)*Y(i,p)*Y(j,k))*K(i,j,k,
                    p);
            end
        end
    end
end

v=sum(theta);
%
%Negative of the Lagrangian function to be minimized
%
f=1/4*(s+t)-v;
%
%Gradient of the objective function
%
if nargout>1
    gradf= zeros(N,m);
    for i=1:N
        for k=1:m
            factor1=0;
            factor2=0;
            factor4=0;
            factor5=0;
            %lambda(i,k)==theta((k-1)*N+i)
            for j=setdiff(1:N,i)
                for p=1:m
                factor1=factor1+theta((k-1)*N+j)*Y(i,k)*Y(j,k)*K(i,j,p,p);
                end
            end
            for j=1:N
                for p=setdiff(1:m,k)
                factor2=factor2+theta((p-1)*N+j)*Y(j,p)*Y(i,k)*K(j,i,k,p);
                end
            end
            factor3=theta((k-1)*N+i)*(Y(i,k)^2)*K(i,i,k,k);
            for p=1:m
                factor4=factor4+theta((k-1)*N+i)*(Y(i,k)^2)*K(i,i,p,p);
            end
            for j=setdiff(1:N,i)
               factor5=factor5+theta((k-1)*N+j)*Y(j,k)*Y(i,k)*K(j,i,k,k);
```

```
            end
            gradf(i,k)=1/2*factor1+1/2*factor2+1/2*factor3+1/2*factor4+1/2*
                factor5-1;
        end
    end
    gradf=gradf(:);
end
end
```

Appendix B

Semi-infinite programming reduction to finite problems

Let us consider the problem of solving the general optimization problem given by the following formulation

$$\begin{aligned} \max_{x \in X} \quad & f(x) \\ \text{s.t.} \quad & g(x, y) \leq 0, \ \ \forall y \in Y, \end{aligned} \tag{B.1}$$

with $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^m$. Hence, $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is the function to be optimized subject to a system of constraint expressed by function $g : \mathbb{R}^n \times \mathbb{R}^m \longrightarrow \mathbb{R}$. Problem (B.1) takes the name of *semi-infinite programming* (SIP) deriving it from the fact that X indicates a finite number of variables $x = (x_1, \ldots, x_n)^{\mathrm{T}} \in X$, while set Y is an infinite set, corresponding to an infinite number of constraints.

Let

$$\mathcal{F} = \{x \mid g(x, y) \leq 0, \ y \in Y\}$$

denote the feasible set of Problem (B.1). Let $\bar{Y} \subset Y$ be a subset of finite dimension of the set Y, i.e., $|\bar{Y}| < \infty$, and let us consider the following problem achieved by replacing feasible set $\mathcal{F}$ with

$$\bar{\mathcal{F}} = \{x \mid g(x, y) \leq 0, \ y \in \bar{Y}\}$$

in (B.1)

$$\begin{aligned} \max_{x \in X} \quad & f(x) \\ \text{s.t.} \quad & x \in \bar{\mathcal{F}}. \end{aligned} \tag{B.2}$$

Namely, we wish to approximate the semi-infinite programming problem described in (B.1) by imposing only finitely many constraints. As a direct consequence, we wish to

understand whether or not Problems (B.1) and (B.2) are comparable with each other, specifically if they share the same optimum solution.

As described in [Hettich and Kortanek, 1993], this is not always the case, and, therefore, Problem (B.1) may not generally be approximated by its finite formulation given in (B.2). The following result holds.

Theorem 12. *Consider the problem*

$$\begin{aligned} \max_{x \in X} \quad & f(x) \\ s.t. \quad & g(x,y) \leq 0, \ \ \forall y \in Y, \end{aligned} \tag{B.3}$$

with $X \subset \mathbb{R}^n$ *and* $Y \subset \mathbb{R}^m$*. For convenience let us indicate*

$$\begin{aligned} v(P) = \max\{f(x), \ \textit{such that } x \in Y^P\} \ \ \textit{with} \\ Y^P = \{x, \ \textit{such that } g(x,y) < 0, \ y \in Y\} \subset \mathbb{R}^n. \end{aligned}$$

Assume the following conditions are satisfied:

1. *Y is a compact set;*
2. *The objective function $f : x \mapsto f(x)$ is concave;*
3. *Constraint function $g : (x,y) \mapsto g(x,y)$ is convex with respect to x (f and $g(\cdot, y)$ are finite over $\mathbb{R}^n$);*
4. *Optimum solution f^* of Problem (B.3) is finite;*
5. *For every set of $n+1$ elements $y_0, y_1, \ldots, y_n \in Y$ there exists a vector $\tilde{x} \in X$ such that $g(\tilde{x}, y_i) < 0$, for every $i = 0, \ldots, n$.*

Then, a set $T_n = \{y_1, \ldots, y_n\} \subset Y$ exists, such that the two following conditions are true.

(a) $v(P) = v(P(T_n))$*;*

(b) Multipliers $\mu_i \geq 0$, $i = 1, \ldots, n$, exist such that the following condition holds

$$v(P) = \sup \left\{ f(x) - \sum_{i=1}^{n} \mu_i g(x, y_i), x \in \mathbb{R}^n \right\};$$

where $P(T_n)$ defines the approximate problem given by

$$\begin{aligned} v(P(T_n)) = \max\{f(x), \ \textit{such that } x \in P(T_n)\} \ \ \textit{with} \\ P(T_n) = \{x, \ \textit{such that } g(x,y) < 0, \ y \in T_n\} \subset \mathbb{R}^n. \end{aligned}$$

Appendix C

Duality

Consider the following nonlinear minimization problem

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0, \\ & h(x) = 0, \end{aligned} \tag{C.1}$$

with $f : \mathbb{R}^n \longrightarrow \mathbb{R}$, $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ and $h : \mathbb{R}^n \longrightarrow \mathbb{R}^p$.

The Lagrangian associated with Problem (C.1) is a function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \longrightarrow \mathbb{R}$ defined as

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^{\mathrm{T}} g(x) + \mu^{\mathrm{T}} h(x), \tag{C.2}$$

with vectors λ and μ called Lagrangian multipliers. The dual function is given by

$$\theta(\lambda, \mu) = \min_{x\in\mathbb{R}^n} \mathcal{L}(x, \lambda, \mu),$$

and the corresponding dual problem is defined as

$$\begin{aligned} \max_{\lambda,\mu} \quad & \theta(\lambda, \mu) \\ \text{s.t.} \quad & \lambda \geq 0. \end{aligned} \tag{C.3}$$

Theorem 13. ***Weak Duality***
Let us consider the pair of problems described in (C.1) and (C.3). Let $\bar{x}$ be a feasible solution for the primal minimization Problem (C.1) and $\bar{\lambda}, \bar{\mu}$ be a feasible solution for the dual maximization Problem (C.3). Then

$$\theta(\bar{\lambda}, \bar{\mu}) \leq f(\bar{x}).$$

Proof. We have that

$$\begin{aligned} \theta(\bar{\lambda}, \bar{\mu}) &= \min_{x\in\mathbb{R}^n} \left\{ f(x) + \bar{\lambda}^{\mathrm{T}} g(x) + \bar{\mu}^{\mathrm{T}} h(x) \right\} \\ &\leq f(\bar{x}) + \bar{\lambda}^{\mathrm{T}} g(\bar{x}) + \bar{\mu}^{\mathrm{T}} h(\bar{x}) \\ &\leq f(\bar{x}), \end{aligned}$$

where the equality follows from the definition of the Lagrangian dual function and the second inequality is a direct consequence of the feasibility of point $\bar{x}$. □

Therefore, from the Weak Duality Theorem, the objective value at any feasible solution to the dual Problem (C.3) is a lower bound for the objective value at any feasible solution to the primal Problem (C.1).

Now, let us define

$$p^* = \min_x f(x)$$

and

$$d^* = \max_{\lambda,\mu} \theta(\lambda, \mu).$$

We call *duality gap* the non-negative number given by $p^* - d^*$.

Definition C.0.1. Slater's condition
Given the convex problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & g(x) \leq 0, \\ & h(x) = 0, \end{aligned} \tag{C.4}$$

with $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ convex and continuously differentiable, $g : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ convex and continuously differentiable and $h : \mathbb{R}^n \longrightarrow \mathbb{R}^p$ linear functions. Problem (C.4) satisfies Slater's condition if there exists a strictly feasible point, namely if the following condition is verified.

$$\exists \ \hat{x} \in \mathcal{D} : g(\hat{x}) < 0, \ \ h(\hat{x}) = 0,$$

where $\mathcal{D}$ is the domain of the problem, i.e., the intersection of the domains of all the functions involved.

We have than that the following result holds.

Theorem 14. ***Strong Duality***
Let us consider the convex primal problem defined in (C.4). If it satisfies the Slater's condition, i.e., if there exists an $\hat{x}$ such that $\hat{x}$ is strictly feasible, then the duality gap is zero, namely

$$p^* = d^*.$$

Again, let us consider the convex minimization problem (C.4). The corresponding Karush-Kuhn-Tucker (KKT) conditions are

1. $\nabla f(x) + \sum_{i=1}^m \lambda_i \nabla g_i(x) + \sum_{j=1}^p \mu_j \nabla h_j(x) = 0$ (stationarity condition);
2. $g_i(x) \leq 0$, for all $i = 1, \ldots, m$ and $h_j(x) = 0$, for every $j = 1, \ldots, p$ (primal feasibility condition);

3. $\lambda_i \geq 0$, for all $i = 1, \ldots, m$ (dual feasibility condition);

4. $\sum_{i=1}^{m} \lambda_i g_i(x) = 0$ (complementary slackness condition).

Theorem 15. *Let us consider the convex Problem (C.4) and assume that the strong duality condition holds, i.e., the duality gap is zero. Then x^*, λ^* and μ^* are respectively primal and dual solutions if and only if x^*, λ^* and μ^* satisfy the Karush-Kuhn-Tucker conditions.*

Proof. ($\Rightarrow$) Let x^*, λ^* and μ^* be respectively primal and dual solutions of Problems (C.4) and (C.3) with duality gap equal to zero. Clearly primal and dual feasibility conditions hold. Moreover, we have

$$\begin{aligned} f(x^*) &= \theta(\lambda^*, \mu^*) \\ &= \min_x \left\{ f(x) + \sum_{i=1}^{m} \lambda_i^* g_i(x) + \sum_{j=1}^{p} \mu_j^* h_j(x) \right\} \\ &\leq f(x^*) + \sum_{i=1}^{m} \lambda_i^* g_i(x^*) + \sum_{j=1}^{p} \mu_j^* h_j(x^*) \\ &\leq f(x^*). \end{aligned} \tag{C.5}$$

Hence, all the inequalities stated in Equation (C.5) are actually equalities. Therefore, the following conditions are verified.

· Point x^* minimizes Lagrangian function $\mathcal{L}(x, \lambda^*, \mu^*)$, which leads to

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{j=1}^{p} \mu_j^* \nabla h_j(x^*) = 0;$$

· $\sum_{i=1}^{m} \lambda_i^* g_i(x^*) = 0$, namely

$$\lambda_i^* g_i(x^*) = 0, \quad \forall i = 1, \ldots, m.$$

Hence, x^*, λ^* and μ^* satisfy the Karush-Kuhn-Tucker conditions.
($\Leftarrow$) Let us suppose that there exist x^*, λ^* and μ^* satisfying the Karush-Kuhn-Tucker conditions. Then, we have

$$\begin{aligned} \theta(\lambda^*, \mu^*) &= f(x^*) + \sum_{i=1}^{m} \lambda_i^* g_i(x^*) + \sum_{j=1}^{p} \mu_j^* h_j(x^*) \\ &= f(x^*), \end{aligned}$$

where the first equality is a consequence of stationary condition, since function $f(x) + \sum_{i=1}^{m} \lambda_i^* g_i(x) + \sum_{j=1}^{p} \mu_j^* h_j(x)$ is convex so any stationary point corresponds to a minimum point. The second equality comes from complementary slackness condition. Hence x^*, λ^* and μ^* are respectively primal and dual solutions. □

Again, let us consider the convex Problem (C.4) and the corresponding Lagrangian function

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^{\mathrm{T}} g(x) + \mu^{\mathrm{T}} h(x).$$

Then, the Wolfe dual problem corresponding to (C.4) is given by

$$\begin{aligned} \max_{x,\lambda,\mu} \quad & \mathcal{L}(x, \lambda, \mu) \\ \text{s.t.} \quad & \nabla_x \mathcal{L}(x, \lambda, \mu) = 0, \\ & \lambda \geq 0. \end{aligned} \tag{C.6}$$

Proposition 16. *Let x^* be a global solution for primal Problem (C.4) with corresponding Lagrangian multipliers equal to λ^* and μ^*. Then x^*, λ^* and μ^* are a global solution for Problem (C.6) and the duality gap is zero.*

Proof. By Theorem 15, point (x^*, λ^*, μ^*) satisfies the Karush-Kuhn-Tucker condition. As a direct consequence, (x^*, λ^*, μ^*) is a feasible point for Problem (C.6). Moreover,

$$\begin{aligned} \mathcal{L}(x^*, \lambda^*, \mu^*) &= f(x^*) + (\lambda^*)^{\mathrm{T}} g(x^*) + (\mu^*)^{\mathrm{T}} h(x^*) \\ &= f(x^*), \end{aligned}$$

since $g(x^*)^{\mathrm{T}}(\lambda^*) = 0$ by complementary slackness condition and $h(x^*) = 0$ by feasibility condition. Therefore, the duality gap is zero. Moreover, let us consider generic $\lambda \geq 0$ and $\mu \in \mathbb{R}^p$. As a consequence of the feasibility of the point x^*, we have that

$$\begin{aligned} \mathcal{L}(x^*, \lambda^*, \mu^*) = f(x^*) &\geq f(x^*) + \lambda^{\mathrm{T}} g(x^*) + \mu^{\mathrm{T}} h(x^*) \\ &= \mathcal{L}(x^*, \lambda, \mu). \end{aligned} \tag{C.7}$$

Since f and g are convex, h is linear by assumption and λ is non-negative, $\mathcal{L}(\cdot, \lambda, \mu)$ is convex in x and, for any feasible choice of (x, λ, μ) we have

$$\begin{aligned} \mathcal{L}(x^*, \lambda, \mu) &\geq \mathcal{L}(x, \lambda, \mu) + \nabla_x \mathcal{L}(x, \lambda, \mu)^{\mathrm{T}} (x^* - x) \\ &= \mathcal{L}(x, \lambda, \mu), \end{aligned} \tag{C.8}$$

where the equality comes from the first constraint of Problem (C.6). By combining Equations (C.7) and (C.8), we obtain

$$\mathcal{L}(x^*, \lambda^*, \mu^*) \geq \mathcal{L}(x, \lambda, \mu),$$

for every feasible choice of (x, λ, μ) for Problem (C.6). Therefore, (x^*, λ^*, μ^*) is a global solution of Problem (C.6). □

Bibliography

Aguiar, Gabriel J, Everton José Santana, Saulo M Mastelini, Rafael G Mantovani, and Sylvio Barbon Júnior

2019 "Towards meta-learning for multi-target regression problems", in *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, IEEE, pp. 377-382. (Cit. on p. 119.)

Argyriou, Andreas, Raphael Hauser, Charles A Micchelli, and Massimiliano Pontil

2006 "A DC-programming algorithm for kernel selection", in *Proceedings of the 23rd international conference on Machine learning*, pp. 41-48. (Cit. on p. 86.)

Aronszajn, Nachman

1950 "Theory of reproducing kernels", *Transactions of the American mathematical society*, 68, 3, pp. 337-404. (Cit. on pp. 23, 35, 64.)

Bartlett, Peter L

1998 "The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network", *IEEE transactions on Information Theory*, 44, 2, pp. 525-536. (Cit. on p. 96.)

Bayes, Thomas

1763 "LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, FRS communicated by Mr. Price, in a letter to John Canton, AMFR S", *Philosophical transactions of the Royal Society of London*, 53, pp. 370-418. (Cit. on p. 6.)

Ben-David, Shai and Reba Schuller

2003 "Exploiting task relatedness for multiple task learning", in *Learning Theory and Kernel Machines*, Springer, pp. 567-580. (Cit. on p. 112.)

Berkson, Joseph

1944 "Application of the logistic function to bio-assay", *Journal of the American statistical association*, 39, 227, pp. 357-365. (Cit. on p. 52.)

Berlinet, Alain and Christine Thomas-Agnan

2011 *Reproducing kernel Hilbert spaces in probability and statistics*, Springer Science & Business Media. (Cit. on p. 33.)

Bielza, Concha, Guangdi Li, and Pedro Larranaga

2011 "Multi-dimensional classification with Bayesian networks", *International Journal of Approximate Reasoning*, 52, 6, pp. 705-727. (Cit. on p. 112.)

Borchani, Hanen, Gherardo Varando, Concha Bielza, and Pedro Larrañaga

2015 "A survey on multi-output regression", *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5, 5, pp. 216-233. (Cit. on p. 112.)

Boser, Bernhard E, Isabelle M Guyon, and Vladimir N Vapnik

1992 "A training algorithm for optimal margin classifiers", in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144-152. (Cit. on pp. 48, 64.)

Brown, Gavin

2004 *Diversity in neural network ensembles*, PhD thesis, Citeseer. (Cit. on p. 109.)

Cai, Feng and Vladimir Cherkassky

2012 "Generalized SMO algorithm for SVM-based multitask learning", *IEEE transactions on neural networks and learning systems*, 23, 6, pp. 997-1003. (Cit. on p. 113.)

Chen, Zhenyu and Jianping Li

2007 "A multiple kernel support vector machine scheme for simultaneous feature selection and rule-based classification", in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, pp. 441-448. (Cit. on pp. 84, 85.)

Chen, Zhenyu, Jianping Li, and Liwei Wei

2007 "A multiple kernel support vector machine scheme for feature selection and rule extraction from gene expression data of cancer tissue", *Artificial Intelligence in Medicine*, 41, 2, pp. 161-175. (Cit. on p. 80.)

Christoudias, Mario, Raquel Urtasun, Trevor Darrell, et al.

2009 "Bayesian localized multiple kernel learning", *Univ. California Berkeley, Berkeley, CA*. (Cit. on p. 80.)

CNN

2016 *Computer scores big win against humans in ancient game of Go*, `https://money.cnn.com/2016/01/28/technology/google-computer-program-beats-human-at-go/index.html`. (Cit. on p. 11.)

Cortes, Corinna, Mehryar Mohri, and Afshin Rostamizadeh

2009 "Learning non-linear combinations of kernels", in *Advances in neural information processing systems*, pp. 396-404. (Cit. on p. 79.)

Cortes, Corinna and Vladimir Vapnik

1995 "Support-vector networks", *Machine learning*, 20, 3, pp. 273-297. (Cit. on pp. 10, 48.)

Cressie, Noel

1992 "Statistics for spatial data", *Terra Nova*, 4, 5, pp. 613-617. (Cit. on p. 113.)

De Klerk, Etienne, Dick Den Hertog, and G Elabwabi

2008 "On the complexity of optimization over the standard simplex", *European journal of operational research*, 191, 3, pp. 773-785. (Cit. on p. 105.)

Dempster, Arthur P, Nan M Laird, and Donald B Rubin

1977 "Maximum likelihood from incomplete data via the EM algorithm", *Journal of the Royal Statistical Society: Series B (Methodological)*, 39, 1, pp. 1-22. (Cit. on p. 9.)

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei

2009 "ImageNet: A Large-Scale Hierarchical Image Database", in *CVPR09*. (Cit. on p. 11.)

Evgeniou, Theodoros and Massimiliano Pontil

2004 "Regularized multi–task learning", in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 109-117. (Cit. on p. 113.)

Fisher, Ronald Aylmer

1936 "Design of experiments", *Br Med J*, 1, 3923, pp. 554-554. (Cit. on p. 7.)

Fukushima, Kunihiko

1980 "A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biol. Cybern.*, 36, pp. 193-202. (Cit. on p. 9.)

Galton, Francis

1889 "I. Co-relations and their measurement, chiefly from anthropometric data", *Proceedings of the Royal Society of London*, 45, 273-279, pp. 135-145. (Cit. on p. 7.)

Gauss, Carl Friedrich

1809 "Theoria motus corporum coelestum", *Werke*. (Cit. on p. 6.)

Gehler, PV. and S. Nowozin

2008 *Infinite Kernel Learning*, tech. rep. 178, Max-Planck Institute for Biological Cybernetics, Tübingen, Germany. (Cit. on pp. 86, 90, 91, 102, 107, 108.)

Gönen, Mehmet and Ethem Alpaydın

2011 "Multiple kernel learning algorithms", *The Journal of Machine Learning Research*, 12, pp. 2211-2268. (Cit. on pp. 78, 79.)

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio

2014 "Generative adversarial nets", in *Advances in neural information processing systems*, pp. 2672-2680. (Cit. on p. 11.)

Henderson, Harold V and Shayle R Searle

1981 "On deriving the inverse of a sum of matrices", *Siam Review*, 23, 1, pp. 53-60. (Cit. on p. 72.)

Hettich, Rainer and Kenneth O Kortanek

1993 "Semi-infinite programming: theory, methods, and applications", *SIAM review*, 35, 3, pp. 380-429. (Cit. on pp. 89, 90, 92, 104, 107, 144.)

Ho, Tin Kam

1995 "Random decision forests", in *Proceedings of 3rd international conference on document analysis and recognition*, IEEE, vol. 1, pp. 278-282. (Cit. on p. 10.)

Hochreiter, Sepp and Jürgen Schmidhuber

1997 "Long short-term memory", *Neural computation*, 9, 8, pp. 1735-1780. (Cit. on p. 10.)

Hopfield, John J

1982 "Neural networks and physical systems with emergent collective computational abilities", *Proceedings of the national academy of sciences*, 79, 8, pp. 2554-2558. (Cit. on p. 9.)

Huang, Guang-Bin, Lei Chen, Chee Kheong Siew, et al.

2006 "Universal approximation using incremental constructive feedforward networks with random hidden nodes", *IEEE Trans. Neural Networks*, 17, 4, pp. 879-892. (Cit. on p. 95.)

Huang, Guang-Bin, Qin-Yu Zhu, and Chee-Kheong Siew

2004 "Extreme learning machine: a new learning scheme of feedforward neural networks", in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, IEEE, vol. 2, pp. 985-990. (Cit. on pp. 94, 95, 97.)

2006 "Extreme learning machine: theory and applications", *Neurocomputing*, 70, 1-3, pp. 489-501. (Cit. on p. 95.)

Huang, Kaizhu, Haiqin Yang, Irwin King, Michael R Lyu, and Laiwan Chan

2004 "Biased Minimax Probability Machine for Medical Diagnosis.", in *ISAIM*. (Cit. on p. 109.)

IBM

1997 *Deep Blue*, `https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/`. (Cit. on p. 10.)

2011 *IBM Watson*, `https://www.research.ibm.com/deepqa/faq.shtml`. (Cit. on p. 11.)

Ivakhnenko, Alekseui Grigorevich and Valentin Grigor'evich Lapa

1965 *Cybernetic predicting devices*, CCM Information Corporation. (Cit. on p. 8.)

Jolliffe, Ian

2011 "Principal Component Analysis", in *International Encyclopedia of Statistical Science*, ed. by Miodrag Lovric, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1094-1096, ISBN: 978-3-642-04898-2, DOI: `10.1007/978-3-642-04898-2_455`, `https://doi.org/10.1007/978-3-642-04898-2_455`. (Cit. on p. 73.)

Kaggle

2010 *Kaggle.com*, `https://www.kaggle.com/`. (Cit. on p. 11.)

Kashima, Hisashi, Satoshi Oyama, Yoshihiro Yamanishi, and Koji Tsuda

2009 "On pairwise kernels: An efficient alternative and generalization analysis", in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, pp. 1030-1037. (Cit. on p. 79.)

Kolmogorov, Andrey

1933 "Sulla determinazione empirica di una lgge di distribuzione", *Inst. Ital. Attuari, Giorn.*, 4, pp. 83-91. (Cit. on p. 7.)

Laplace, Pierre-Simon

1810 "Mémoire sur les approximations des formules qui sont fonctions de très grands nombres et sur leur applications aux probabilités", *Memoires de l'Academie des Sciences de Paris*. (Cit. on p. 7.)

LeCun, Yann, Corinna Cortez, and Christopher C.J. Burges

1999 *The MNIST Handwritten Digit Database*, `http://yann.lecun.com/exdb/mnist/`. (Cit. on p. 10.)

Legendre, Adrien Marie

1805 *Nouvelles méthodes pour la détermination des orbites des comètes*, F. Didot. (Cit. on p. 6.)

Liu, Xinwang, Lei Wang, Guang-Bin Huang, Jian Zhang, and Jianping Yin

2015 "Multiple kernel extreme learning machine", *Neurocomputing*, 149, pp. 253-264. (Cit. on pp. 100, 101.)

Liu, Yong, Shizhong Liao, Hailun Lin, Yinliang Yue, and Weiping Wang

2017 "Infinite kernel learning: generalization bounds and algorithms", in *Thirty-First AAAI Conference on Artificial Intelligence*. (Cit. on pp. 86, 87.)

Luenberger, David G, Yinyu Ye, et al.

1984 *Linear and nonlinear programming*, Springer, vol. 2. (Cit. on p. 83.)

Mancini, Alessio, Leonardo Vito, Elisa Marcelli, Marco Piangerelli, Renato De Leone, Sandra Pucciarelli, and Emanuela Merelli

2020 "Machine learning models predicting multidrug resistant urinary tract infections using "DsaaS"", *BMC bioinformatics*, 21, 10, pp. 1-12. (Cit. on p. 1.)

Marcelli, Elisa and Renato De Leone

2019 "Infinite Kernel Extreme Learning Machine", in *Advances in Optimization and Decision Science for Society, Services and Enterprises*, Springer, pp. 95-105. (Cit. on pp. 93, 105.)

2020 "Multi-Kernel Covariance Terms in Multi-Output Support Vector Machines", in *Machine Learning, Optimization, and Data Science*, Springer, pp. 714-725. (Cit. on pp. 93, 116.)

Markov, Andrei A

1913 "Primer statisticheskogo issledovanija nad tekstomEvgenija Onegina'illjustrirujuschij svjaz'ispytanij v tsep (An example of statistical study on the text ofEugene Onegin'illustrating the linking of events to a chain)", *Izvestija Imp. Akademii nauk*, 6, 3, pp. 153-162. (Cit. on p. 7.)

Mastelini, Saulo Martiello, Everton Jose Santana, Victor Guilherme Turrisi da Costa, and Sylvio Barbon

2018 "Benchmarking multi-target regression methods", in *2018 7th Brazilian Conference on Intelligent Systems (BRACIS)*, IEEE, pp. 396-401. (Cit. on p. 120.)

Maxwell, James Clerk

1860 "V. Illustrations of the dynamical theory of gases.—Part I. On the motions and collisions of perfectly elastic spheres", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 19, 124, pp. 19-32. (Cit. on p. 7.)

Melki, Gabriella, Alberto Cano, Vojislav Kecman, and Sebastián Ventura

2017 "Multi-target support vector regression via correlation regressor chains", *Information Sciences*, 415, pp. 53-69. (Cit. on p. 112.)

Melkumyan, Arman and Fabio Ramos

2011 "Multi-kernel Gaussian processes", in *Twenty-second international joint conference on artificial intelligence*. (Cit. on pp. 113, 114.)

Mercer, James

1909 "Xvi. functions of positive and negative type, and their connection the theory of integral equations", *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209, 441-458, pp. 415-446. (Cit. on p. 39.)

Micchelli, Charles A and Massimiliano Pontil

2005 "Kernels for Multi–task Learning", in *Advances in neural information processing systems*, pp. 921-928. (Cit. on p. 113.)

Minsky, Marvin and Seymour A Papert

1969 *Perceptrons: An introduction to computational geometry*, MIT press. (Cit. on p. 9.)

Mitchell, Tom M.

1997 *Machine Learning*, McGraw-Hill. (Cit. on p. 1.)

Neal, Radford M

2012 *Bayesian learning for neural networks*, Springer Science & Business Media, vol. 118. (Cit. on p. 69.)

Netflix

2009 *The Netflix Prize*, `https://www.netflixprize.com/`. (Cit. on p. 11.)

Ordowski, Mark and Gerard GL Meyer

2004 "Geometric linear discriminant analysis for pattern recognition", *Pattern Recognition*, 37, 3, pp. 421-428. (Cit. on p. 61.)

Pearl, Judea

1985 "Bayesian netwcrks: A model cf self-activated memory for evidential reasoning", in *Proceedings of the 7th Conference of the Cognitive Science Society, University of California, Irvine, CA, USA*, pp. 15-17. (Cit. on p. 9.)

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay

2010 "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, 12, pp. 2825-2830. (Cit. on p. 11.)

Poggio, Tomaso and Federico Girosi

1990 "Networks for approximation and learning", *Proceedings of the IEEE*, 78, 9, pp. 1481-1497. (Cit. on p. 64.)

Rakotomamonjy, Alain, Francis R Bach, Stéphane Canu, and Yves Grandvalet

2008 "SimpleMKL", *Journal of Machine Learning Research*, 9, Nov, pp. 2491-2521. (Cit. on pp. 83, 84.)

Rakotomamonjy, Alain, Francis Bach, Stéphane Canu, and Yves Grandvalet

2007 "More efficiency in multiple kernel learning", in *Proceedings of the 24th international conference on Machine learning*, pp. 775-782. (Cit. on p. 83.)

Rasmussen, Carl Edward

2003 "Gaussian processes in machine learning", in *Summer School on Machine Learning*, Springer, pp. 63-71. (Cit. on pp. 69, 71.)

Reed, Michael

2012 *Methods of modern mathematical physics: Functional analysis*, Elsevier. (Cit. on p. 39.)

Riesz, Marcel

1909 *Sur les opérations fonctionnelles linéaires*, C. R. Acad. Sci. Paris, 149: 974-977. (Cit. on p. 31.)

Rosenblatt, Frank

1957 *The perceptron, a perceiving and recognizing automaton Project Para*, Cornell Aeronautical Laboratory. (Cit. on pp. 8, 43.)

Samuel, Arthur L

1959 "Some studies in machine learning using the game of checkers", *IBM Journal of research and development*, 3, 3, pp. 210-229. (Cit. on p. 1.)

Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Müller

1998 "Nonlinear component analysis as a kernel eigenvalue problem", *Neural computation*, 10, 5, pp. 1299-1319. (Cit. on pp. 64, 75.)

Shannon, C. E.

1948 "A mathematical theory of communication", *The Bell System Technical Journal*, 27, 3, pp. 379-423, DOI: 10.1002/j.1538-7305.1948.tb01338.x. (Cit. on p. 8.)

Taigman, Yaniv, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf

2014 "Deepface: Closing the gap to human-level performance in face verification", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701-1708. (Cit. on p. 11.)

Tanabe, Hiroaki, Tu Bao Ho, Canh Hao Nguyen, and Saori Kawasaki

2008 "Simple but effective methods for combining kernels in computational biology", in *2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*, IEEE, pp. 71-78. (Cit. on p. 80.)

Tesauro, Gerald

1992 "Temporal difference learning of backgammon strategy", in *Machine Learning Proceedings 1992*, Elsevier, pp. 451-457. (Cit. on p. 9.)

Times, New York

2012 *How Many Computers to Identify a Cat? 16,000*, https://www.nytimes.com/2012/06/26/technology/in-a-big-network-of-computers-evidence-of-machine-learning.html?pagewanted=all. (Cit. on p. 11.)

Tsoumakas, Grigorios, Ioannis Katakis, and Ioannis Vlahavas

2009 "Mining multi-label data", in *Data mining and knowledge discovery handbook*, Springer, pp. 667-685. (Cit. on p. 118.)

Turing, Alan

1950 "Computing machinery and intelligence", *Mind*, 59, 236, p. 433. (Cit. on p. 8.)

Vapnik, V and A Chervonenkis

1979 "Theory of Pattern Recognition Nauka, Moscow", *German translation: Theorie der Zeichenerkennung, Akademie Verlag, Berlin.* (Cit. on p. 15.)

Vapnik, V and A Ya Chervonenkis

1964 "A class of algorithms for pattern recognition learning", *Avtomat. i Telemekh*, 25, 6, pp. 937-945. (Cit. on p. 47.)

Vapnik, Vladimir

2013 *The nature of statistical learning theory*, Springer science & business media. (Cit. on p. 15.)

Varma, Manik and Bodla Rakesh Babu

2009 "More generality in efficient multiple kernel learning", in *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1065-1072. (Cit. on p. 79.)

Vembu, Shankar and Thomas Gärtner

2010 "Label ranking algorithms: A survey", in *Preference learning*, Springer, pp. 45-64. (Cit. on p. 112.)

Wackernagel, Hans

2013 *Multivariate geostatistics: an introduction with applications*, Springer Science & Business Media. (Cit. on p. 113.)

Wahba, Grace

1990 *Spline models for observational data*, SIAM. (Cit. on p. 64.)

Widrow, Bernard and Marcian E Hoff

1960 *Adaptive switching circuits*, tech. rep., Stanford Univ Ca Stanford Electronics Labs. (Cit. on p. 8.)

Wu, Di, Boyu Wang, Doina Precup, and Benoit Boulet

2017 "Boosting based multiple kernel learning and transfer regression for electricity load forecasting", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, pp. 39-51. (Cit. on p. 80.)

Xu, Donna, Yaxin Shi, Ivor W Tsang, Yew-Soon Ong, Chen Gong, and Xiaobo Shen

2019 "Survey on Multi-Output Learning", *IEEE transactions on neural networks and learning systems.* (Cit. on p. 112.)

Xu, Shuo, Xin An, Xiaodong Qiao, and Lijun Zhu

2014 "Multi-task least-squares support vector machines", *Multimedia tools and applications*, 71, 2, pp. 699-715. (Cit. on p. 113.)

Zaremba, Stanislaw

1907 *Léquation biharmonique et une classe remarquable de fonctions fondamentales harmoniques*, Imprimerie de l'Universite. (Cit. on p. 23.)

Zhang, Jiashuai, Yiwei He, and Jingjing Tang

2018 "Multi-view multi-task support vector machine", in *International Conference on Computational Science*, Springer, pp. 419-428. (Cit. on p. 113.)

Zhou, Zhi-Hua and Yuan Jiang

2004 "NeC4. 5: neural ensemble based C4. 5", *IEEE Transactions on Knowledge and Data Engineering*, 16, 6, pp. 770-773. (Cit. on p. 109.)

Zien, Alexander and Cheng Soon Ong

2007 "Multiclass multiple kernel learning", in *Proceedings of the 24th international conference on Machine learning*, pp. 1191-1198. (Cit. on p. 81.)