Kernels Methods in Machine Learning

Trabajo Fin de Grado
Dpto. de Estadística e Investigación Operativa

Universidad de Sevilla
Doble Grado en Física y Matemáticas
24 de Junio 2022

Enrique Naranjo Bejarano
Tutor: Emilio J. Carrizosa Priego

Contents

1 Introduction. Statistical Learning Theory. 2

1.1 Statistical Learning Theory . . . 3

1.1.1 The formal setup . . . 3

1.1.2 The VC Dimension . . . 6

1.1.3 Structural Risk Minimization and Regularization . . . 8

1.1.4 Rademacher Complexity . . . 9

2 Introduction to Kernels 11

2.1 Reproducing Kernel Hilbert Space . . . 11

2.2 The Reproducing Kernel Feature Map and The Feature Space . . . 16

2.3 Kernels in Rademacher Complexity Theory . . . 18

2.4 Construction of kernels . . . 20

2.4.1 Polynomial kernels . . . 21

2.4.2 Gaussian kernel . . . 22

3 Kernels in Supervised Learning 23

3.1 Support Vector Machine . . . 23

3.1.1 Hard-margin SVM . . . 24

3.1.2 Soft margin classifier . . . 27

3.1.3 ν-support vector machine . . . 29

3.1.4 2-norm soft margin SVM . . . 30

3.2 Support vector machines for regression . . . 31

3.2.1 ε-insensitive regression . . . 32

3.2.2 ν-support vector regression . . . 34

3.3 The perceptron . . . 35

4 Kernels in Unsupervised Learning 37

4.1 Kernel Principal Component Analysis . . . 37

4.1.1 Principal Component Analysis . . . 37

4.1.2 Kernel principal component analysis . . . 39

4.2 Kernel Cluster Analysis . . . 40

4.2.1 Greedy solutions: k-means . . . 43

4.2.2 Relaxed solution: spectral methods . . . 44

4.3 Data visualisation . . . 45

4.3.1 Multidimensional scaling . . . 46

5 Conclusions 48

References 50

1 Introduction. Statistical Learning Theory.

In this final degree thesis, we are interested in understanding the role of positive-definite kernels in the field of Machine Learning. First introduced by James Mercer in 1909 [1] in the context of integral equations, the concept of a positive-definite kernel would be further developed to become a central concept in Machine Learning. It was not until 1995, when Vladimir Vapnik [12] introduced the Support Vector Machine algorithm, that the theory of positive-definite kernels would be used in practical applications. This breakthrough was possible due to the development between 1960 and 1990 of Statistical Learning Theory(SLT) by Vapnik [14]. This framework is the most natural for introducing concepts that are central to Machine Learning, such as loss function, regularization, and, most notably for this thesis, kernels. Statistical Learning Theory tries to define the notion of learning by an algorithm in a mathematical sense. Thus, its main focus is related to supervised learning. Supervised learning is the collection of machine learning algorithms that work with labeled data. They are concerned with classification and regression problems. The main difference between the two is that in classification problems, the data we want to predict is discrete, and in regression, it is continuous. Although SLT comprises a natural framework for understanding supervised learning algorithms, it does not generalize well to unsupervised learning algorithms. Unsupervised learning tries to find underlying relationships in data. Thus, this kind of algorithm does not 'learn' anything, but rather, they help the person working with the data to understand it better. Although they are not the primary concern in machine learning, unsupervised learning is a necessary tool in any machine learning project pipeline as they help select the number of features that best fit the learning algorithm of choice. For these reasons, the kernel method that was proven useful for supervised learning algorithms was extended to unsupervised learning. We will see that any machine learning algorithm that can be expressed only in terms of the dot products of the data is suitable for a kernelized form.

During this thesis, we will follow a constructive approach. We begin by introducing the main concepts of Statistical Learning Theory. This incremental process will help us understand the main concepts underlying machine learning. After this, the notion of a semi-positive kernel will come naturally. We will introduce the three main components of the theory of semi-positive kernels: Reproducing kernel Hilbert Space, Mercer's Theorem, and the Representation Theorem. Once we have constructed a working theory of semi-positive kernels in Machine Learning, we will be concerned with how to implement algorithms that take advantage of this theory. Also, we will want to construct a series of valuable kernels. Following a historical thread, we introduce support-Vector Machines (SVM) for classification and regression problems. Then, we introduce another powerful algorithm: perceptrons. The perceptron algorithm is crucial because it is the foundation for Neural Networks, one of the most popular algorithms nowadays. Then, we will follow by introducing some kernelized algorithms in Unsupervised Learning. These algorithms are Kernel Principal Component Analysis, Kernel Cluster Analysis, and Kernel Multidimensional Scaling. As

Machine Learning is more an applied field than a theoretical one, along with each theoretical development of the algorithm, we will make comments about the computational complexity of the algorithms. Along these lines, we will discover that the main obstacle to implementing kernel methods is that they are not efficient enough for large-scale applications. This inefficiency is why this method fell out of fashion in the second half of the 00'. Although kernel methods are no longer considered to be state-of-the-art, they are an essential piece in the field of Machine Learning, and they still play a central role in many Machine Learning projects. Thus, this thesis will help us get a fine grip on all the main concepts of Machine Learning, and along the way, we will flawlessly define the concept of a kernel.

1.1 Statistical Learning Theory

In this section, we are interested in introducing the main concepts of Statistical Learning Theory as they were first presented by Vapnik [14]. This abstract learning theory introduced more general ideas than those discussed in classical statistical paradigms. The understanding of these new conditions helped in the development of new algorithms. Once we have taken care of the underlying statistical nature of the data, we will be more concerned with problems related to functional analysis. Thus, Statistical Learning Theory comprises a framework that unifies the classical theories of statistics and functional analysis. According to Vapnik [14], Fisher's paradigm to answer the learning question: *"What must one know a priori about an unknown functional dependency to estimate it based on observations"* was very restrictive. As suggested, one must know everything, i.e., understand the desired dependency up to the values of a finite number of parameters. Vapnik's new paradigm indicated that it is sufficient to know some general properties of the set of functions to which the unknown dependency belongs. To achieve this objective, Vapnik developed these subjects in his theory: determination of the general conditions under which estimating the unspecified dependence is possible, finding the inductive principles that lead to finding the best approximation to the unknown dependency, and, once all these steps are accomplished, developing algorithms for its implementation. Thus, under this framework, we have a systematic method for developing algorithms such as Support Vector Machines (SVM) and Neural Networks (NN). Furthermore, we will see that the notion of kernel follows naturally.

1.1.1 The formal setup

In Supervised Learning we will work with an *input space* $\mathbf{X}$ and an *output space* $\mathbf{Y}$. Our main objective will be to estimate a functional relationship of the form $f : \mathbf{X} \to \mathbf{Y}$. Such a relationship f is called a *classifier*. The interesting part is that we don't have access to all the points in $\mathbf{X} \times \mathbf{Y}$, but rather to a subset of *training points* $S = \{(X_1, Y_1), \dots, (X_n, Y_n)\} \subset \mathbf{X} \times \mathbf{Y}$. Our goal is to find a *classification algorithm* that produces a classifier f for our training data. If this classifier f adjusts well enough (we will see what this exactly means later) to the true underlying relationship, then we say the algorithm has *generalized.* Although no assumption is made on the spaces $\mathbf{X}$ and $\mathbf{Y}$, we will assume that each point is generated *independently* from a *joint probability distribution* P on $\mathbf{X}$ and $\mathbf{Y}$. It is important to notice that the probability distribution also affects the labels $\mathbf{Y}$. Consequently, the labels Y_i are not a deterministic function of the objects X_i. We assume this to include the possibility that the labels are suggested to *noise.* Thus, we consider that we may be training with incorrect labels. This is a somewhat realistic assumption as the labeling process is usually made by hand and is prone to errors. Nevertheless, we will always assume that this type of error is small. Another critical case covered under this assumption is that of *overlapping classes.* For example, if we want to find the gender of a person according to height, the height $X = 1.80m$ may correspond to a girl and a boy. In both cases, the essential condition to take under consideration is the conditional likelihood of the labels:

$$\eta(x) := P(Y = 1|X = x).$$

In the case of slight little noise, the conditional probability will be close to 1. For significant label noise, it will be close to 0.5. In the latter case, learning becomes more complex, and it becomes unavoidable that the classifier makes a relatively large number of errors.

Another essential assumption is that the samples are drawn *independent and identically distributed* (iid). Iid data is a relatively strong but necessary assumption. Thus, the way the data get sampled is essential to satisfy this assumption. Getting iid data will not always be possible, as is the case in *active learning* in which the users get to select the points that they want to get labeled actively. Or the subject of *time series analysis.* Analyzing data from a sample that is not iid is an unresolved field of study and is heavily researched.

An essential difference between classical statistics and SLT is that *SLT does not make any assumption on P.* Thus, statistical learning theory works in an agnostic setting. This agnostic setting is different from classical statistics, which assume that the probability distribution belongs to a probability distribution, and the goal is to estimate the parameters of the distribution. Moreover, *the distribution P is unknown at the time of learning*, which is one of the essential assumptions in SLT. Learning would be trivial if we knew P as we could write a formula for the best classifier. Writing these closed types of formulas is what the Statisticians in the first half of the 20th century tried to achieve. However, as Vapnik noted in [14], the density estimation problem is a hard and *ill-posed* problem. Then, we only have access to P through the training points. The more training points we have, the more prone the algorithm will be to generalize. Statistical Learning Theory allows us to make formal statements about the rate of convergence of the learning algorithm. Furthermore, it gives us bounds abound the number of observations the algorithm needs to generalize. We will see an important trade-off between the 'complexity' of the classes of function that we choose to fit our data and the algorithm's capacity to generalize. Moreover, the *distribution P will be fixed*. Indeed, classical SLT does not work with probability distributions that change over time. Thus, we will not be able to work with *time series data*. This strong assumption leaves an increasingly important set of problems behind.

In this way, learning consists on finding a classifier $f : \mathbf{X} \to \mathbf{Y}$ from a sample data set that approximates $\mathbb{E}[y|\mathbf{x}]$. We will look for such an estimator in a set of possible candidates. This set is called the *hypotheses space* $\mathcal{F}$. We also need an error criterion to find the best possible candidate from the hypothesis space $\mathcal{F}$, which is defined as a *risk functional* $I : \mathcal{F} \to \mathbb{R}$. This functional gives the *expected risk*:

$$I[f] = \iint_{\mathbf{X}\times\mathbf{Y}} V(y, f(\mathbf{x}))p(\mathbf{x}, y)\ d\mathbf{x}dy, \tag{1.1}$$

where $V(y, f(\mathbf{x}))$ is called the *loss function* as it measures the error made when y is predicted by $f(\mathbf{x})$. Then, a critical step in the learning process is to define the error function $V(y, f(\mathbf{x}))$. Popular options for the loss function are:

1. The L_2 norm or squared loss function: $V(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$.
2. The L_1 norm loss function: $V(y, f(\mathbf{x})) = |y - f(\mathbf{x})|$.
3. The Vapnik epsilon-insensitive loss function:
$$V(y, f(\mathbf{x})) = \begin{cases} |y - f(\mathbf{x})| & \text{if} \quad |y - f(\mathbf{x})| \geq \epsilon \\ 0 & \text{if} \quad |y - f(\mathbf{x})| < \epsilon \end{cases}$$
4. The soft margin loss function:
$$V(y, f(\mathbf{x})) = \begin{cases} y - f(\mathbf{x}) & \text{if } y \geq f(\mathbf{x}) \\ 0 & \text{if } y < f(\mathbf{x}) \end{cases}$$
5. The Huber Loss function:
$$V(y, f(\mathbf{x})) = \begin{cases} \epsilon|y - f(\mathbf{x})| - \frac{\epsilon^2}{2} & \text{if} \quad |y - f(\mathbf{x})| \geq \epsilon \\ \frac{1}{2}(y - f(\mathbf{x}))^2 & \text{if} \quad |y - f(\mathbf{x})| < \epsilon \end{cases}$$

We will mainly use the squared loss function. Once the loss functions is specified, we look for a classifier $f \in \mathcal{F}$ that minimizes the expected risk:

$$f^* = \arg\min_{f\in\mathcal{F}} I[f].$$

The minimized function f^* is called the *target function*. The joint probability $P(\mathbf{x}, y)$ is unknown from our assumptions. And the only information available about joint probability is the training set $\mathcal{S}$. Then, we can not compute the expected risk directly; consequently, we can not compute the target function. To overcome this problem, Vapnik [14] proposed that it is enough to minimize the so-called *empirical risk*:

$$I_{emp}[f,\mathcal{S}] = \frac{1}{N}\sum_{i=1}^{N} V(y_i, f(\mathbf{x_i})). \tag{1.2}$$

Finding the optimal hypothesis that minimizes the functional over the hypothesis space is called the *empirical risk minimization.* Vapnik proved [14] that the problem of empirical risk minimization is statistically consistent with the minimization of the expected risk. In order to be consistent, the following law of large numbers is a necessary and sufficient condition:

$$\lim_{N\to\infty} P\left(\sup_{f\in\mathcal{F}}(I[f] - I_{emp}[f,\mathcal{S}]) > \varepsilon\right) = 0, \quad \forall\varepsilon > 0. \tag{1.3}$$

Thus, the objective now is to minimize the empirical risk. However, the only points available to solve that minimization problem are the points from the training set. We may find a function that minimizes the empirical risk but does not minimize the expected risk. In this case, we say that the function *overfit* the data and that the algorithm is not able to *generalize.* Then, the problem of empirical risk minimization is ill-posed. To solve this, we limit the set of hypotheses $\mathcal{F}$ when minimizing the empirical risk $I_{emp}[f,\mathcal{S}]$. This restriction on the hypotheses spaces $\mathcal{F}$ is the key for the theory of semi-positive kernels as we will force this set to be a bounded convex subset of a *Reproducing Kernel Hilbert Space* $\mathcal{H}$ (RKHS). We will see later that this RKHS can be represented by a kernel k. Thus, characterising the space in which we are looking for a solution.

From the previous discussion, we have seen a constant trade-off between the number of points in the training set and the "capacity" of the functional space $\mathcal{H}$ where the empirical risk is minimized. Statistical Learning Theory provides probabilistic bounds between the expected and the empirical risk based on the size of the training set N and the "capacity" h of the hypothesis space. These bounds give a combinatorial measure for the model complexity. The problem is how to measure the capacity h of a hypothesis spaces. Vapnik introduced [14] the Vapnik-Chervonenkis (VC) dimension as a way to measure the capacity of a functional space. The VC dimension is one of the most general capacity measures but it is only defined for spaces of binary functions. Several generalizations have been made for spaces of non-binary functions. One of them is the Rademacher complexity. This measure provides more insight into kernel methods that the VC dimension while providing similar bounds. For this reason, we will use the Rademacher complexity during this thesis. We begin by introducing the VC dimension.

1.1.2 The VC Dimension

In order to define the VC dimension of a binary classification model, we introduce the VC dimension of a set family. For this purpose, the concept of a shattered set is introduced.

Definition 1.1 (Shattered set [19])**.** Let be H a set of sets and C be a set. Their intersection is defined as:

$$H \cap C := \{h \cap C \mid h \in H\}. \tag{1.4}$$

We say that the C is *shattered* by H if $H \cap C$ contains all the subsets of C.

From this definition we can define the VC dimension of a set family as:

Definition 1.2 (VC dimension of a family set [19])**.** Let H be a family set. The VC dimension D of H is the largest cardinality of sets shattered by H. If arbitrarily large subsets can be shattered, the VC dimension is ∞.

Thus, we can define what it means for a binary classification model $f(\theta)$ to shatter a set of data points.

Definition 1.3 (Model f shattering [19])**.** A **binary** classification model $f(\theta)$ that depends of some parameter vector θ is said to *shatter* a set of data points $S = \{x_1, x_2, \ldots, x_n\}$ if, for all assignments of labels to those points, there exists a θ such that the model f classify correctly all points in S.

Then, the VC dimension of a binary classification model is defined as:

Definition 1.4 (VC dimension of a model $f(\theta)$ [19])**.** The VC dimension of a model $f(\theta)$ is the maximal cardinal D such that some data point set of cardinality D can be shattered by f.

Then, for example, for a straight line classification model on points in a two-dimensional plane. Its VC dimension is three, as any three points not in a line can be classified with a line, but no line can shatter any set of four points. If we have the set of linear functions but in $\mathbb{R}^d$ the VC-dimension is $D = d + 1$ since is not possible to separate more that $d + 1$ points by a linear hyperplane in $\mathbb{R}^d$.

We call the VC dimension D of a classification model $f(\theta)$ the capacity h of the model. For a given capacity h, the bound of the expected risk is given in general with a probability of at least δ as:

$$I[f] \leq I_{emp}[f, \mathcal{S}] + \eta \left(\sqrt{\frac{h}{N}}, \delta \right), \tag{1.5}$$

where η is an increasing function. The second part in the right side of the inequality is called the *Structural risk.* This relationship is vital in the theory of Statistical Learning. From our previous discussion, the capacity h of a model gives a measure of how well the model can adjust to a set of points. If the model has a high capacity, it adjusts well to the data, and the empirical error is small. However, in this case, the right hand of the previous expression increase as η is an increasing function of $\frac{h}{N}$. Then, the discrepancy between the empirical and expected risk increases. The only way to reduce this discrepancy is to increase the number of data points N for $\frac{h}{N}$ to decrease, but then, if the model is fixed, the empirical error increase. For these reasons, there is a constant trade-off between the empirical risk, the number of points N, and the hypothesis space's capacity h.

1.1.3 Structural Risk Minimization and Regularization

To solve this problem, Vapnik [14] introduced the *Structural Risk Minimization* principle. In order to control the capacity h of the hypotheses space, we will work with a nested sequence of hypotheses spaces $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \dots \mathcal{H}_q$ where each hypothesis space $\mathcal{H}_p$, $p = 1, \dots, q$ has a finite capacity h_p larger than all the previous spaces: $h_1 \leq h_2 \leq \dots \leq h_q$. Then we can control the right side of (1.5) by choice of an appropriate $\mathcal{H}_p$. We are looking for an optimal trade-off between the empirical and the structural risk. In the theory of kernel models, the spaces $\mathcal{H}_p$ are bounded convex subsets of a Reproducing Kernel Hilbert Space $\mathcal{H}$ as induced by a kernel k. If we denote by $\|\cdot\|_k$ the norm in the RKHS and supposing that all functions are bounded by $r \in \mathbb{R}$ the empirical risk minimization is given by:

$$\min_{f \in \mathcal{H}, \|f\|_k^2 \leq r^2} \frac{1}{N} \sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)). \tag{1.6}$$

The fact that the space $\mathcal{H}$ is bounded results in a regularized empirical risk minimization known as *Ivanov Regularization.* If we take into account the Structural Risk Minimization principle then we have to solve the following problem:

$$\min_{f \in \mathcal{H}, \|f\|_k^2 \leq a_p^2} \frac{1}{N} \sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)). \tag{1.7}$$

where the sequence $\{a_p\}_{p=1}^{q} \subset \mathbb{R}$ is a monotonically increasing sequence of real constants. It can be proven [14] that optimizing this formulation of the Ivanov Regularization problem is equivalent to minimizing:

$$\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)) + \gamma_p \left(\|f\|_k^2 - a_p^2 \right), \tag{1.8}$$

with respect to $f \in \mathcal{H}$ and maximizing with respect to $\gamma_p \geq 0$. Moreover, according to the structural minimization principle, this should be done for each a_p. However, this is a computationally intensive task. For this reason, the problem is reformulated in a relaxed form known as *Tikhonov regularization problem*:

$$\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)) + \frac{1}{2\gamma} \|f\|_k^2. \tag{1.9}$$

Where γ is a regularization parameter, thus, the Tikhonov regularization proposes a trade-off between the structural risk $\|f\|_k^2$ and the empirical risk. This parameter is not known beforehand. To select one value of γ, various values γ are chosen, and we pick the optimal one using a technique such as *cross-validation.* The parameter γ is called an *hyperparameter* of the model.

1.1.4 Rademacher Complexity

Although the VC dimension provides an intuitive approach to defining the capacity of a hypothesis space, it also comes with many theoretical limitations. For these reasons, a more appropriate measure of the capacity was introduced: the Rademacher Complexity. We begin by introducing the *Rademacher distribution*:

Definition 1.5 (Rademacher Distribution [19])**.** The *Rademacher distribution* is a discrete probability distribution whose probability mass function is:

$$f(k) = \begin{cases} 1/2 & \text{if } k = -1 \\ 1/2 & \text{if } k = +1 \\ 0 & \text{otherwise} \end{cases} \tag{1.10}$$

Thus, if σ is a random variable drawn from a Rademacher distribution: $P(\sigma = -1) = P(\sigma = 1) = 0.5$.

Given a space Z and a fixed distribution D_Z, let $S = \{z_1, \dots, z_N\}$ be a set of examples drawn iid from D_Z. Furthermore, let $\mathcal{F}$ be a class of functions $f : Z \to \mathbb{R}$.

Definition 1.6 (Empirical Rademacher complexity [19])**.** The *empirical Rademacher complexity* of $\mathcal{F}$ is defined to be:

$$[[19]]\hat{R}_N(\mathcal{F}) = \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{F}} \left(\frac{1}{N} \sum_{i=1}^{N} \sigma_i f(z_i) \right) \right] \tag{1.11}$$

where $\sigma_1, \dots, \sigma_m$ are independent random variables over the Rademacher distribution.

Definition 1.7 (Rademacher complexity [19])**.** The Rademacher complexity of $\mathcal{F}$ is defined as

$$R_N(\mathcal{F}) = \mathbb{E}_D[\hat{R}_N(\mathcal{F})] \tag{1.12}$$

In this expression, the supremum measures, for a given set S and the Rademacher vector σ, the maximum correlation between $f(z_i)$ and σ_i over all $f \in \mathcal{F}$. After taking the expectation over σ, the empirical Rademacher complexity measures the ability of functions from $\mathcal{F}$ to fit random noise. Then, the Rademacher complexity of $\mathcal{F}$ measures the expected noise-fitting-ability of $\mathcal{F}$ over all data sets $S \in Z^m$ that could be drawn according to the distribution D_Z.

We present a uniform convergence result for any class of (bounded) real-value functions.

Theorem 1.8 ([18]). *Fix distribution D_Z and parameter $\sigma \in (0,1)$. If $\mathcal{F} \subset \{f : Z \to [a, a+1]\}$ and $S = \{z_1, \dots, z_n\}$ is draw iid from D_Z then with probability greater or equal than $1-\delta$ over the draw of S, for every function $f \in \mathcal{F}$,*

$$\mathrm{E}_D[f(z)] \leq \hat{\mathrm{E}}_S[f(z)] + 2R_N(\mathcal{F}) + \sqrt{\frac{\ln(1/\delta)}{N}}.$$

In addition, with probability $\geq 1-\delta$, for every function $f \in \mathcal{F}$,

$$\mathrm{E}_D[f(z)] \leq \hat{\mathrm{E}}_S[f(z)] + 2\hat{R}_N(\mathcal{F}) + 3\sqrt{\frac{\ln(2/\delta)}{N}}$$

Proof. [18] □

At last, we present the following result that connects the loss function and the expected and empirical risk. This bound depends on the class of function we are working with, and we restrict our presentation to the class of linear functions.

Theorem 1.9 ([18]). *Let $X = \mathbb{R}^d$, $Y = \{-1, 1\}$, and $Z = X \times Y$. For a concept class $\mathcal{H} \subset \{h : X \to Y\}$ we can let $L(\mathcal{H}) = \{V_h \mid h \in \mathcal{H}\}$ where $V_h : Z \to \mathbb{R}$ is a loss function corresponding to the classifier h. If we let $\mathcal{H}$ be the class of linear separators and $L(\mathcal{H})$ be the corresponding class of $0-1$ loss functions, i.e. $l_h(z) = l_h(x,y) = 1_{h(x)\neq y}$ for each $h \in H$, then:*

$$\mathbb{E}_D[l_h(z)] = I[h] \tag{1.13}$$

and

$$\hat{\mathbb{E}}_S[l_h(z)] = \frac{1}{N}\sum_{i=1}^{N} 1_{h(x_i)\neq y_i} = I_{emp}(h) \tag{1.14}$$

Taking $\mathcal{F} = L(\mathcal{H})$, then:

$$I[h] \leq I_{emp}[h] + 2R_N(L(\mathcal{H})) + \sqrt{\frac{\ln 1/\delta}{N}} \tag{1.15}$$

Moreover, the relationship $R_N(L(\mathcal{H})) = \frac{1}{2}R_N(\mathcal{H})$ holds, and the above expression can be written as:

$$I[h] \leq I_{emp}[h] + R_N(\mathcal{H}) + \sqrt{\frac{\ln 1/\delta}{N}} \tag{1.16}$$

Proof. [18] □

The above theorem gives a bound of the generalization error of a hypothesis in terms of its empirical error and the Rademacher complexity of the class of loss functions. After this result, we are ready to introduce the kernel concept. Once we fully understand the concept of a kernel and its role in Statistical Learning Theory, we will incorporate it into the Rademacher complexity theory.

2 Introduction to Kernels

We are ready to introduce Kernel theory's main ingredients: Reproducing Kernel Hilbert Spaces (RKHS), Mercer Kernels, and the Reproducing theorem. As stated in the previous section, we are looking for solutions in a hypothesis space $\mathcal{H}$ that, for our convenience, will be an RKHS. We will see that an RKHS is uniquely determined by a kernel and, thanks to Mercer's theorem, a Mercer kernel defines an RKHS. We will see how this is related to the implementation of the structural risk minimization principle. Then, we will show that a definite positive kernel k uniquely determines a so-called *Reproducing Feature Space* $\mathcal{F}_r$. The Kernel k represents the inner product between the elements of the feature space. Thus, a kernel can be seen as a function that gives the inner product of data points in a Feature Space. This Feature Space can be even infinite-dimensional. Thus, kernels allow us to calculate the inner product of points in infinite-dimensional spaces computationally. This vision is the most popular interpretation of kernels and the one we will use to derive the kernelized versions of linear algorithms. For this reason, when implementing the kernelized version on an algorithm, our objective will be to write it in a form such that only the inner products between the data are needed. This condition is not restrictive at all. When implementing an algorithm in data analysis, the only information that we know of is the distance between the data points. Thus, it is natural to write an algorithm that only depends on the distance of its points. When writing the algorithm is written in this form, we will say that the algorithm is in *dual form.* The dual version of an algorithm only depends on the inner products of the data points. As we can express the inner products as a kernel, we can couple the algorithm with any kernel function we choose. Thus, naturally restricting the hypotheses space. Moreover, we have achieved a modularity design in which the choice of Kernel and algorithms are separated. In this way, we can turn linear algorithms into non-linear ones. Thanks to this vision, we can also write an Unsupervised learning algorithm in kernel form. At last, we will show how the Rademacher complexity of a family of functions depends on kernels.

2.1 Reproducing Kernel Hilbert Space

We begin this section by introducing the concept of a Reproducing Kernel Hilbert Space:

Definition 2.1 (Reproducing Kernel Hilbert Space)**.** Let $\mathcal{H}$ be a functional Hilbert Space whose functions are defined over a bounded domain $\mathcal{X} \subset \mathbb{R}^d$. $\mathcal{H}$ is a Reproducing Kernel Hilbert Space if the evaluation functional $\mathcal{F}_{\mathbf{x}}$, defined as:

$$\mathcal{F}_{\mathbf{x}}[f] = f(\mathbf{x}), \quad \forall f \in \mathcal{H}, \tag{2.1}$$

is continuous for every $\mathbf{x} \in \mathcal{X}$. This is equivalent to being a linear and bounded operator in $\mathcal{H}$, i.e. there exists some $M_{\mathbf{x}} > 0$ such that

$$|\mathcal{F}_{\mathbf{x}}[f]| := |f(\mathbf{x})| \leq M_{\mathbf{x}} \left\| f \right\|_{\mathcal{H}}, \quad \forall f \in \mathcal{H}. \tag{2.2}$$

Where $\|f\|_{\mathcal{H}}$ is the norm defined by the inner product $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ in the Reproducing Hilbert Space. By the *Riesz representation theorem* [11], it can be proven that for each RKHS there exists an unique *positive definite function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that $k(\mathbf{x}, \cdot) \in \mathcal{H},\ \forall \mathbf{x} \in \mathcal{X}$ has the *reproducing property*:

$$f(\mathbf{x}) = \langle f(\cdot), k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} \tag{2.3}$$

Since $k(\mathbf{x}, \cdot) \in \mathcal{H}$:

$$k(\mathbf{x}, \mathbf{y}) = \mathcal{F}_{\mathbf{y}}[k(\mathbf{x}, \cdot)] = \langle k(\mathbf{x}, \cdot), k(\mathbf{y}, \cdot)\rangle_{\mathcal{H}} \tag{2.4}$$

Then $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called the *reproducing kernel* of $\mathcal{H}$.

Definition 2.2 ((Strictly) Positive Definite Function)**.** Let $a_1, a_2, \dots, a_n \in \mathbb{R}$, and $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \in \mathcal{X}$. The function $f : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is said to be *positive definite* if:

$$\sum_{i,j=1}^{n} a_i a_j f(\mathbf{x}_i, \mathbf{x}_j) \geq 0. \tag{2.5}$$

If this inequality is strictly positive, the function is called *strictly positive definitive.*

According to this definition the kernel k of a RKHS is positive definitive:

$$\sum_{i,j=1}^{n} a_i a_j k(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \sum_{i=1}^{n} a_i k(\mathbf{x}_i, \cdot), \sum_{j=1}^{n} a_j k(\mathbf{x}_j, \cdot) \right\rangle_{\mathcal{H}} = \left\| \sum_{i=1}^{n} a_i k(\mathbf{x}_i, \cdot) \right\|_{\mathcal{H}}^2 \geq 0. \tag{2.6}$$

This property is an important one because, according to the Moore-Aronszajn theorem [3], every symmetric, positive definite kernel defines a unique Reproducing Kernel Hilbert Space.

Theorem 2.3 ([3])**.** *Suppose k is a symmetric, positive definite kernel on a set $\mathcal{X}$. Then there is a unique Hilbert space of function on $\mathcal{X}$ for which k is a reproducing kernel.*

In this way, a RKHS is uniquely determined by its reproducing kernel k.

Another important class of kernels are *Mercer kernels* which are defined using Mercer's theorem

Theorem 2.4 (Mercer's theorem [1])**.** *Let X be a compact subset of $\mathbb{R}^d$. Suppose κ is a continuous symmetric function such that the integral operator $T_\kappa : L^2(\mathcal{X}) \to L^2(\mathcal{X})$*

$$(T_K f)(\cdot) = \int_X \kappa(\cdot, \mathbf{x}) f(\mathbf{x}) d\mathbf{x},$$

is positive, that is

$$\int_{\mathcal{X}\times\mathcal{X}} \kappa(\mathbf{x}, \mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0,$$

for all $f \in L^2(\mathcal{X})$. Then, we can expand $\kappa(\mathbf{x},\mathbf{z})$ in a uniformly convergent series (on $\mathcal{X} \times \mathcal{X}$) in terms of functions ϕ_j:

$$\kappa(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{d_{\mathcal{H}}} \lambda_j \phi_j(\mathbf{x})\phi_j(\mathbf{z}).$$

where $\{\phi_i\}_{i=1}^{d_{\mathcal{H}}} \in L^2(\mathcal{X})$ is an orthogonal set of normalized eigenfunctions of the integral operator T_k, i.e. $\|\phi_i\|_{L^2} = 1$. Correspondingly, the $\{\lambda_i\}_{i=1}^{d_{\mathcal{H}}}$ are the positive associated eigenvalues of the integral operator T_k and $d_{\mathcal{H}}$, the dimension of the Hilbert space, is either $d_{\mathcal{H}\in\mathbb{N}}$ or $d_{\mathcal{H}} = \infty$. Furthermore, the series $\sum_{i=1}^{\infty} \|\phi_i\|^2_{L^2(\mathcal{X})}$ is convergent.

Proof. Page 64 [18]. $\square$

The requirements defined in this theorem are called *Mercer conditions*, and the kernel *Mercer Kernel.* Mercer Kernels and the reproducing kernel of a RKHS are related. In the first place, it can be prove that Mercer Kernels are also positive definite. Intuitively this can be seen by taking a weighting sum of delta functions in the Mercer condition of the positiveness of the integral operator. Moreover, for a given *Mercer kernel k* defined over the domain $\mathcal{X} \subset \mathbb{R}^d$, there exists an RKHS $\mathcal{H}$ of functions defined over $\mathcal{X}$ for which k is the reproducing kernel.

Theorem 2.5 ([19])**.** *Given a Mercer kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ where $\mathcal{X} \subseteq \mathbb{R}$ is bounded. Then, there exists an RKHS $\mathcal{H}$ of functions defined over $\mathcal{X}$ for which the Mercer Kernel k is the reproducing kernel.*

Proof. We already now that the Mercer Kernel k is positive definite. Our aim is to construct a RKHS where k is the reproducing kernel. Let H be a Hilbert Space of functions of the form:

$$f(\cdot) = \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i(\cdot), \tag{2.7}$$

where $\{\phi_i\}_{i=1}^{d_{\mathcal{H}}}$ and $\{\lambda_i\}_{i}^{d_{\mathcal{H}}}$ are the eigenfunctions and eigenvalues of the integral operator corresponding to the Mercerl Kernel. The coefficients $c_i \in \mathbb{R}$ are arbitrary.

The inner product in this Hilbert space is defined to be:

$$\langle f(\cdot), g(\cdot)\rangle_{\mathcal{H}} = \left\langle \sum_{i=1}^{d_{\mathcal{H}}} c_i\phi_i(\cdot), \sum_{i=1}^{d_{\mathcal{H}}} d_i\phi_i(\cdot) \right\rangle := \sum_{i=1}^{d_{\mathcal{H}}} \frac{c_i d_i}{\lambda_i} \tag{2.8}$$

This Hilbert space is an RKHS with the reproducing kernel the Mercer kernel.

$$
\begin{aligned}
\langle f, k(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} &= \left\langle \sum_{i=1}^{d_{\mathcal{H}}} c_i \phi_i(\cdot), \sum_{j=1}^{d_{\mathcal{H}}} \lambda_j \phi_j(\mathbf{x}) \phi_j(\cdot) \right\rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{d_{\mathcal{H}}} \sum_{j=1}^{d_{\mathcal{H}}} c_i \lambda_j \phi_j(\mathbf{x}) \left\langle \phi_i(\cdot), \phi_j(\cdot) \right\rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{d_{\mathcal{H}}} \frac{c_i \lambda_i \phi_i(\mathbf{x})}{\lambda_i} \\
&= f(\mathbf{x}).
\end{aligned}
\tag{2.9}
$$

From which follows that the Mercer kernel k is a reproducing kernel in this Hilbert space. □

The opposite is also true.

Theorem 2.6 ([19]). *For a given RKHS $\mathcal{H}$ its corresponding reproducing kernel is also a Mercer kernel with associate eigenfunctions and eigenvalues $\{\phi_i\}_{i=1}^{d_{\mathcal{H}}}$ and $\{\lambda_i\}_i^{d_{\mathcal{H}}}$.*

Proof. See [15]. □

Then the concepts of a Mercer kernel and a reproducing kernel are equivalent:

$$
\text{RKHS } \mathcal{H} \iff \text{Mercer kernel} \iff \text{Reproducing kernel} \tag{2.10}
$$

Thus, for any RKHS there is a unique kernel k and corresponding λ_n and ϕ_n that uniquely define the space. Moreover, the norm in that RKHS is given by:

$$
\|f\|_k^2 = \sum_{i=1}^{d_{\mathcal{H}}} \frac{c_i^2}{\lambda_i}. \tag{2.11}
$$

where the notation $\|\cdot\|_k$ is used to stress the fact that the norm only depends on the kernel k of the RKHS. As we introduced previously, our goal is to use the structural risk minimization principle to derive learning machines. According to this principle, we need to define a nested sequence of hypothesis spaces $\mathcal{H}_1 \subset \cdots \subset \mathcal{H}_n$ with $\mathcal{H}_m$ being the set of functions f in the RKHS $\mathcal{H}$. It can be proven [16] that the capacity of the set of functions $\{f \in \mathcal{H} \mid \|f\|_k^2 \leq a\}$ depends on $a \in \mathbb{R}^+$. Therefore, it is enough to search for functions such as:

$$
\|f\|_k^2 \leq a_m^2, \tag{2.12}
$$

where a_m is a monotonically increasing sequence of positive constants. Then, for each m we need to solve the following constrained minimization problem:

$$
\begin{array}{r}
\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{\ell} V\left(y_i, f\left(\mathbf{x}_i\right)\right) \\
\|f\|_K^2 \leq A_m^2
\end{array}
\tag{2.13}
$$

This type of learning machines are called *kernel machines*. Learning using kernel machines of this form leads to using the Lagrange multiplier λ_m and to minimizing:

$$\frac{1}{N}\sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)) + \lambda_m(\|f\|_k^2 - A_m^2), \tag{2.14}$$

with respect to $f \in \mathcal{H}$ and maximizing with respect to $\lambda_m \geq 0$ for each element of the structure. We can choose the optimal $m^*(N)$, and the associated $\lambda^*(N)$, and get the optimal solution $\hat{f}_{m^*}(N)$.

This is the same solution we get if we solve the problem:

$$\min_{f\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)) + \lambda^*(N)\,\|f\|_k^2 . \tag{2.15}$$

Therefore, a *kernel machine* can also be defined as the solution of the following more general problem:

$$\min_{f\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)) + \lambda\,\|f\|_k^2 . \tag{2.16}$$

The following theorem is called *the Representer Theorem* and it is the last important theorem in the theory of kernels.

Theorem 2.7 (Representer Theorem [19])**.** *Consider a positive definite real valued kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ on a non-empty bounded set $\mathcal{X}$ with a corresponding Reproducing Kernel Hilbert Space $\mathcal{H}_k$. Let $V : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ be a differentiable loss function. Then, the solution to the problem:*

$$\min_{f\in\mathcal{H}} \frac{1}{N}\sum_{i=1}^{N} V(y_i, f(\mathbf{x}_i)) + \lambda\,\|f\|_k^2 , \tag{2.17}$$

has the form:

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^{N} c_i k(\mathbf{x}, \mathbf{x}_i), \tag{2.18}$$

with the coefficients c_i found by solving the minimization problem.

Proof. [19]. □

Thus, in order to define a family of learning functions it is enough to specify the loss function V. For example:

1. $V(y, f(\mathbf{x})) = (y = f(\mathbf{x}))^2$, the squared loss function, results in Regularization Networks.

2. $V(y, f(\mathbf{x})) = |y - f(\mathbf{x}|_\varepsilon$, the Vapnil's $\varepsilon-$insensitive loss function, results in SVM regression.

3. $V(y, f(\mathbf{x})) = |1 - yf(\mathbf{x})|_+$, the soft margin loss function, results in SVM classification.

After defining the kernel function of a RKHS we are ready to introduce the *Feature Space.*

2.2 The Reproducing Kernel Feature Map and The Feature Space

Given a positive definite kernel k, it define a *Reproducing Feature Space* $\mathcal{F}_r$. The kernel k represents an inner product between its elements in this space. The *Reproducing Kernel Feature Map* is defined as follows:

Definition 2.8 (The Reproducing Kernel Feature Map and The Feature Space [19])**.** Given a positive definite kernel k, the *Reproducing Kernel Feature Map* is a function $\Phi_r : \mathcal{X} \to \mathcal{F}_r$ such that:

$$\Phi(x) = k(\cdot, x), \tag{2.19}$$

the space $\mathcal{F}_r$ is called the *Feature Space.*

The feature space $\mathcal{F}_r$ is the image of the space $\mathcal{X}$ under the mapping Φ_r, and it is equal to:

$$\mathcal{F}_r = \text{span}(\{\Phi_r(\mathbf{x}) \ : \ \mathbf{x} \in \mathcal{X}\}) = \{f(\cdot) = \sum_{i=1}^{n} a_i k(\cdot, \mathbf{x}_i) : n \in \mathbb{N}, \mathbf{x} \in \mathcal{X}, a_i \in \mathbb{R}\} \tag{2.20}$$

For any two functions $f(\cdot) = \sum_i c_i k(\cdot, u_i)$ and $g(\cdot) = \sum_i d_i k(\cdot, v_i)$, the inner product in this space is define as:

$$\langle f, g \rangle_{\mathcal{F}_r} = \sum_{i,j} c_i d_j k(u_i, v_j). \tag{2.21}$$

It can be proven that this is in fact an inner product and that the space $\mathcal{F}_r$ equipped with this product can be completed to be a Hilbert space. This Hilbert space has the reproducing property:

$$\langle f, k(\cdot, \mathbf{x}) \rangle = \sum_i c_i k(\mathbf{x}, \mathbf{u}_i) = f(\mathbf{x}) \tag{2.22}$$

an the Hilbert Space is equal to:

$$\mathcal{F}_r = \overline{\text{span}(\{k(\cdot, \mathbf{x}) \ : \ \mathbf{x} \in \mathcal{X}\})} \tag{2.23}$$

Therefore, it also holds that:

$$\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle_{\mathcal{F}_r} = k(\mathbf{x}, \mathbf{x}'). \tag{2.24}$$

In this feature space, $\mathcal{F}_r$ the reproducing kernel can be seen as an inner product of the elements in the Hilbert space.

Another way to highlight the relation between kernels and inner products is through the Mercer theorem. As was shown earlier, there is a set of orthogonal eigenfunctions $\{\phi_i\}_{i=1}^{d_\mathcal{H}}$ and eigenvalues $\{\lambda_i\}_{i=1}^{d_\mathcal{H}}$ associated to every positive definite kernel. Now, we define the *Mercer Kernel Map* $\Phi_m : \mathcal{X} \subseteq \mathbb{R}^d \to \mathcal{F}_m$ where the target space $\mathcal{F}_m$ is called the *Mercer Kernel Feature Space* as:

$$\Phi_m(\mathbf{x}) = \left[\sqrt{\lambda_1}\phi_1(x), \ldots, \sqrt{\lambda_\mathcal{H}}\phi_{d_\mathcal{H}}(\mathbf{x})\right]^T, \tag{2.25}$$

where the dimension $d_\mathcal{H}$ is the dimension of the feature space, and it is determined by the choice of kernel. In the same way, an inner product can be defined in this space:

$$\langle \Phi_m(\mathbf{x}), \Phi_m(\mathbf{x}') \rangle_{\mathcal{F}_m} = \sum_{i=1}^{d_\mathcal{H}} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}'). \tag{2.26}$$

It can be shown that the space $\mathcal{F}_m$ equipped with this inner product is a Hilbert space. Moreover, the kernels reflect the inner product of the mapping of two vectors in this feature space $\mathcal{F}_m$. Thus, the positive definite kernel k can be interpreted as an inner product in another vector space. The importance of this relation is that in the feature space, the mapped data is of dimension $d_\mathcal{H}$, which can be infinite-dimensional. Kernels allow us to compute the inner product of two mapped data points $\phi(\mathbf{x})$ and $\phi(\mathbf{x})'$ that reside in a space of any possible dimension. Thus, any algorithm that only depends on inner products can be expressed in terms of kernels, and this kernel represents the inner products of the points in a feature space.

It is important to notice that the *Mercer Kernel Feature Space* $\mathcal{F}_m$ and the *Reproducing Feature Space* $\mathcal{F}_r$ are not the same space. However, it can be proven [17], that there exists an isometric isomorphism $\beta : \mathcal{F}_m \to \mathcal{F}_r$ between the two spaces. For this reason, we call such spaces the *Feature Space.*

We have shown how to construct a Feature space from a kernel. Now, we will show how a mapping of the data, $\Phi : \mathcal{X} \to \mathcal{F}$, to a Feature space $\mathcal{F}$ also defines a kernel k. This mapping can be done if the function is positive and definite.

Theorem 2.9 (Characterisation of kernels [18])**.** *A function*

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R},$$

which is either continuous or has a finite domain, can be decomposed

$$\kappa(x, z) = \langle \phi(x), \phi(z) \rangle$$

into a feature map Φ into a Hilbert space $\mathcal{F}$ applied to both its arguments followed by the evaluation of the inner product in $\mathcal{F}$ if and only if it satisfies the positive definite property.

Proof. Page 61 [18]. □

Therefore, giving a positive definite kernel, it implicitly defines a feature space $\mathcal{F}$ and an associated map Φ. Therefore, given a kernel, we can forget altogether about the feature map Φ. This fact is known as the *Kernel Trick* [17]: "*Given a model or algorithm which is formulated in terms of a positive definite kernel k, one can construct an alternative algorithm by replacing k by another positive definite kernel k'.* In this way, a algorithm working with the input data set $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathcal{X}$ with respect to a kernel k can be understood as operating on the mapped data $\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \ldots, \phi(\mathbf{x}_N) \in \mathcal{F}_1$. If this kernel k is replaced by another kernel k', this is understood as working on another set of mapped vectorial data $\phi'(\mathbf{x}_1), \ldots, \phi'(\mathbf{x_N}) \in \mathcal{F}_2$. In this way, the more convenient way to think about kernels is a function that defines a similarity measure between the data points. By changing the kernel, we change how we measure the similarity between the data points. Moreover, by the *Representer Theorem* any solution to the regularized empirical risk minimization problem can be expressed in terms of the kernel. Thus, any algorithm we create will only depend on the kernel between the points. For this reason, the *Gram matrix* or *kernel matrix* is the matrix whose entries are equal to:

$$G_{ij} = \langle \phi(x_i), \phi(x_j) \rangle = \kappa(x_i, x_j). \tag{2.27}$$

This matrix is symmetric and contains all pairs of inner products between the training data points. From our previous discussion, calculating the Gram matrix is the first step for any kernel-based algorithm. Thus, the Gram matrix serves as an *information bottleneck* between the data set and the machine learning algorithm. For this reason, kernel methods have a computational complexity of at least $O(N^2)$. This complexity is one of the biggest inconveniences of kernel methods: they are not scalable to large datasets. The other problem is the selection of the kernel. The kernel must recognize the underlying structure of the data set. In order to make this selection, we must encode our prior expectations about the possible functions we may be expected to learn. Therefore, kernel methods are only valid if designed by a person that already has an intuition about the underlying structure of the dataset. Because of that, to use kernel methods, it is necessary to study the properties of each kernel, how they are created, how they can be adapted, and how well they are matched to the task being addressed. These inconveniences are the main reasons kernel methods ceased to be popular in the mid 00' and, instead, were replaced with Neural Networks as the method of choice for machine learning experts.

At last, we will show how to include kernels in the Rademacher Complexity theory and how to construct the most popular classes of kernels.

2.3 Kernels in Rademacher Complexity Theory

In this section we present the main results to include kernels methods in the theory of the Rademacher Complexity. We will restrict our discussion to bounding the Rademacher complexity of bounded linear functions in a kernel-defined feature space.

$$\left\{\sum_{i=1}^{N}\alpha_i k(\mathbf{x}_i,\cdot)|\alpha'\mathbf{K}\alpha \leq B^2\right\} \subseteq \{\langle \mathbf{w},\phi(\cdot)\rangle|\,\|\mathbf{w}\|\leq B\} = \mathcal{F}_B, \tag{2.28}$$

where ϕ is the feature mapping corresponding to the kernel k and $\mathbf{K}$ is the kernel matrix on the sample S. In this expression we have used that the function: $f(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x})\rangle = \sum_{i=1}^{N}\alpha_i k(\mathbf{x}_i,\mathbf{x})$. Moreover:

$$\begin{aligned}
\|\mathbf{w}\|^2 = \langle \mathbf{w},\mathbf{w}\rangle &= \left\langle \sum_{i=1}^{n}\alpha_i\phi(\mathbf{x}_i), \sum_{j=1}^{n}\alpha_j\phi(\mathbf{x}_j)\right\rangle \\
&= \sum_{i,j=1}^{N}\alpha_i\alpha_j\langle\phi(\mathbf{x}_i),\phi(\mathbf{x}_j)\rangle = \sum_{i,j=1}^{N}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j) \\
&= \alpha'\mathbf{K}\alpha.
\end{aligned}$$

We present the following theorem that bound the empirical Rademacher complexity of the class $\mathcal{F}_B$:

Theorem 2.10 ([18])**.** *If $k : \mathcal{X}\times\mathcal{X}\to\mathbb{R}$ is a kernel, and $S = \{\mathbf{x}_1,\ldots,\mathbf{x}_l\}$ is a sample of points from $\mathcal{X}$, then the empirical Rademacher complexity of the class $\mathcal{F}_B$ satisfies:*

$$\hat{R}_l(\mathcal{F}_B) \leq \frac{2B}{N}\sqrt{\sum_{i=1}^{N}k(\mathbf{x}_i,\mathbf{x}_i)} = \frac{2B}{N}\sqrt{tr(\mathbf{K})} \tag{2.29}$$

Proof. Page 100 [18]. □

In order to perform a kernel machine learning algorithm, the dual representation α of the weight vector is first obtained. The corresponding norm B of the weight vector is then $\alpha'\mathbf{K}\alpha$ where $\mathbf{K}$ is the kernel matrix. The quantity $\alpha'\mathbf{K}\alpha$ is related to the complexity of the corresponding functions class. By controlling its size, we can control the capacity of the class of functions and hence improve the statistical stability of the pattern.

At last, we present an example of an application of this bound. We choose the case of classification. For this, we must first introduce the notion of the *margin* for a binary classification problem. For this problem, the *margin* is the amount by which the real value is on the right side of the threshold.

Definition 2.11 ((Functional) margin and slack variable [18])**.** For a function $g : \mathcal{X}\to\mathbb{R}$, we define its *margin* on an example $(\mathbf{x}, y)$ to be $yg(\mathbf{x})$. The *functional margin* of a training set $S = \{(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_N,y_N)\}$, is defined to be

$$m(S,g) = \min_{1\leq i\leq N} y_i g(\mathbf{x}_i). \tag{2.30}$$

Given a function g and a dedired margin γ we denote by $\xi_i = \xi((\mathbf{x}_i,y_i),\gamma,g)$ the amount by which the function g fails to achieve the margin γ for the example $(\mathbf{x}_i,y_i)$.

Using this definition, we can derive the following theorem.

Theorem 2.12 ([18]). *Fix $\gamma > 0$ and let $\mathcal{F}$ be the class of functions mapping from $Z = X \times Y$ to $\mathbb{R}$ given by $f(\mathbf{x}, y) = -yg(\mathbf{x})$, where g is a linear function in a kernel-defined feature space with norm at most 1. Let*

$$S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\} \tag{2.31}$$

be drawn iid with respect to a prability distribution $\mathcal{D}$ and fix $\delta \in (0,1))$. Then with probability at least $1-\delta$ over samples of size N we have

$$I[f] = \mathbb{E}_{\mathcal{D}}[\mathcal{H}(-yg(\mathbf{x}))] \leq \frac{1}{N\gamma}\sum_{i=1}^{N}\xi_i + \frac{4}{N\gamma}\sqrt{tr(\mathbf{K})} + 3\sqrt{\frac{\ln 2/\delta}{2N}}, \tag{2.32}$$

where $\mathbf{K}$ is the kernel matrix for the training set, $\xi_i = \xi((\mathbf{x}_i, y_i), \gamma, g)$ and $\mathcal{H}(\cdot)$ is the Heaviside function *that returns 1 if its argument is greater than 0 and zero otherwise. The loss function can be expressed as:*

$$V(y, f(\mathbf{x})) = \mathbf{H}(-yf(\mathbf{x})) \tag{2.33}$$

Proof. [18]. □

With this theorem, we have seen a typical bound of the risk with respect to the kernel of the space. Now that we know how to work with kernels, we will present some of the most used and important kernels.

2.4 Construction of kernels

Our objectives is to specify a working theory to construct kernels that generalize well. Since we have fully characterized the kernel functions as functions that satisfy the finitely positive definite property to generate new kernels from previous ones, we have to find operations that preserve this property.

Proposition 2.13 (Closure properties [18]). *Let κ_1 and κ_2 be kernels over $X \times X$, $X \subseteq \mathbb{R}^n$, $a \in \mathbb{R}^+$, $f(\cdot)$ a real-valued function on X, $\phi : X \to \mathbb{R}^N$ with κ_3 a kernel over $\mathbb{R}^N \times \mathbb{R}^N$, $p(x)$ a polynomial with positive coefficients, and $\mathbf{B}$ a symmetric positive semi-definite $n \times n$ matrix. Then the following functions are kernels:*

1. $\kappa(x,z) = \kappa_1(x,z) + \kappa_1(x,z)$.
2. $\kappa(x,z) = a\kappa_1(x,z)$.
3. $\kappa(x,z) = \kappa_1(x,z)\kappa_2(x,z)$.
4. $\kappa(x,z) = f(x)f(z)$.
5. $\kappa(x,z) = \kappa_3(\phi(x), \phi(z))$.
6. $\kappa(x,z) = x^T Bz$.

7. $\kappa(x, z) = p(\kappa_1(x, z))$.

8. $\kappa(x, z) = \exp(\kappa_1(x, z)$.

9. $\kappa(x, z) = \exp(-\|x - z\|^2 /(2\sigma^2))$.

Proof. See page 75 [18]. □

The last kernel is known as the *Gaussian kernel.* It captures radial patterns in the underlying data. Using this proposition we can create new kernels from existing ones using a number of simple operations. Moreover, it is sufficient to verify that the a function is finitely positive semi-definite to demonstrate that it is a kernel. All these operations are performed on the kernel function, but we can also apply operations to the kernel matrix as long as the kernel matrix keeps being positive, semi-definite, and symmetric. Such operations are:

1. Adding a constant to all entries in the kernel matrix. This operation correspond to adding an extra constant feature.
2. Adding a constant to the diagonal corresponds to enhancing the independence of all the inputs.
3. Centering the data by moving the origin of the feature space to the center of mass of the training examples.
4. Subspace projection. This can be an effective method of de-noising the data.

We explore in more detail two of the more important kernels: the polynomial kernel and the gaussian kernel.

2.4.1 Polynomial kernels

We introduce the most basic kernel one can use: *the polynomial kernel.* We have previously seen that the space of valid kernels is closed under applying polynomials with positive coefficients. A *polynomial kernel* is defined as follows:

Definition 2.14. Polynomial kernels[[18]] The derived *polynomial kernel* for a kernel κ_1 is defined as

$$\kappa(\mathbf{x}, \mathbf{z}) = p(\kappa_1(\mathbf{x}, \mathbf{z})),$$

where $p(\cdot)$ is any polynomial with positive coefficients. Frequently, it also refers to the special case

$$\kappa_d(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + R)^d,$$

defined over a vector space X of dimension n, where R and d are parameters.

Now we would like to know the dimension of the feature space associated with a polynomial kernel with the previous form. The following proposition gives the response to this question.

Proposition 2.15 ([18])**.** *The dimension of the feature space for the polynomial kernel* $\kappa_d(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + R)^d$ *is*

$$\binom{n+d}{d}.$$

Proof. [18]. □

Now if we expand the polynomial kernel we obtain

$$\kappa_d(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} \binom{d}{s} R^{d-s} \langle \mathbf{x}, \mathbf{z} \rangle^s.$$

Hence we have obtained a reweighting of the features of the polynomial kernels

$$\hat{\kappa}_s(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^s, \text{ for } s = 0, \dots, d.$$

Moreover, the introduction of the parameter R grants of some controls of the relative weightings of the different degree monomials as we can write

$$\kappa_d(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} \binom{d}{s} R^{d-s} \hat{\kappa}_s(\mathbf{x}, \mathbf{z}) = \sum_{s=0}^{d} a_s \hat{\kappa}_s(\mathbf{x}, \mathbf{z})$$

Therefore, increasing R decreases the relative weighting of the higher order polynomials.

2.4.2 Gaussian kernel

The next most used type of kernel is the Gaussian kernel. We define the *Gaussian kernel* as follows

Definition 2.16 (Gaussian kernel [18])**.** For $\sigma > 0$, the *Gaussian kernel* is defined by

$$\kappa(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right).$$

From the definition of the Gaussian kernel we deduce that all points have norm 1 in the resulting feature space as $\kappa(\mathbf{x}, \mathbf{x}) = \exp(0) = 1$. Also, the parameter σ controls the flexibility of the kernel similarly to the degree d in the polynomial kernel. Small values of σ correspond to large values of d since they allow classifiers to fit any label, risking overfitting. In such cases, the kernel matrix becomes close to the identity matrix. On the other hand, large values of σ gradually reduce the kernel to a constant function, making it impossible to learn any non-trivial classifier. The feature space has infinite dimension for every value of σ, but the weight decays very fast on the higher-order features for large values. In other words, although the rank of the kernel matrix will be full for all practical purposes, the points lie in a low-dimensional subspace of the feature space.

3 Kernels in Supervised Learning

3.1 Support Vector Machine

Support Vector Machine (SVM) is a popular machine learning algorithm for classification and regression. It consists of finding a hyperplane that separate the two types of points that we want to classify. The objective of the SVM algorithm is to find the hyperplane that, to the best degree, separates data points of one class from those of another. "Best" is the hyperplane with the most significant margin between the two classes. Margin means the maximum slab width parallel to the hyperplane with no interior data points. Only for linearly separable problems can the algorithm find such a hyperplane; for most practical problems, the algorithm maximizes the soft margin allowing a small number of misclassification. Training a support vector machine corresponds to solving a quadratic optimization problem to fit a hyperplane that minimizes the soft margin between the classes. The number of support vectors determines the number of transformed features. Kernels enter the complete picture to make SVMs more flexible, thus making them able to handle nonlinear problems. In general, we want our algorithms to be robust and stable. In order to check the robustness of the algorithm, the following bound on the expected risk of a linear function $g(x)$ with norm 1 in kernel-defined feature space for a binary classification problem is given [18]:

$$I[f] = P_D(y \neq g(x)) \leq \frac{1}{N\gamma}\sum_{i=1}^{N}\xi_i + \frac{4}{N\gamma}\sqrt{tr(K)} + 3\sqrt{\frac{\ln 2/\delta}{2N}}.$$

This bound is used to choose the linear function returned by the learning algorithm. We will first introduce the hard-margin separating hyperplane. This algorithm does not allow any point to cross the separating hyperplane, which is a very restrictive condition. For this reason, we want to relax the conditions to find a solution that best adjusts to the underlying data. The soft-margin algorithm and the v-soft algorithm are introduced. Thanks to the framework we developed in the previous chapters, we will see that adjusting to each algorithm is enough to select the appropriate loss function and kernel. Once this is done, we will obtain the dual form of the algorithm. Moreover, to fully understand SVM, we have to grasp the following concepts: (i) the separating hyperplane, (ii) the maximum-margin hyperplane, and (iii) the soft margin hyperplane. When developing a machine learning algorithm, two methods can be used. First, we have to choose criteria to look up for a pattern function. In our case, we will try to find a hyperplane that separates the two kinds of points. We will say that the data points are *separable* when possible. Usually, this is not the case as real-world data contain much noise. Therefore we will have to develop a soft-margin approach in which not all data points are included in the separation. We will use stability analysis of the corresponding task to find the desired pattern function. All will come down to a convex optimization problem. We will use Lagrange's optimization theorem to derive the dual form of the pattern function. From this point, we can write the necessary condition in dual form from which an application of kernel functions will be natural.

3.1.1 Hard-margin SVM

We start by analyzing hard-margin SVM. This algorithm will try to find a hyperplane that separates the two types of points that we want to classify. This separation will not always be possible. Therefore, although it is a stable algorithm, it will not be robust because any additional point can make the set non-separable.

We consider a given training set

$$S = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$$

our objective is to find a norm 1 linear function function:

$$g(\mathbf{x}) = \langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b$$

where $\mathbf{w}$ is a weight vector and b is the threshold. Furthermore, we look for a *margin* $\gamma > 0$, such that no point is nearer to the hyperplane by that amount

$$\xi_i = (\gamma - y_i g(\mathbf{x_i}))_+ = 0, \quad 1 \leq i \leq N$$

It is important to notice that the expression $y_i(g(\mathbf{x_i}) = y_i\langle \mathbf{w}, \phi(\mathbf{x_i})\rangle + b)$ measures how far is the point $\phi(\mathbf{x_i})$ from the boundary hyperplane. Moreover, as ξ_i is the loss function for this model, the empirical risk will be equal to zero. If we define the margin to be

$$m(S, g) = \min_{1 \leq i \leq N} y_i g(\mathbf{x_i})$$

then it must satisfies that $m(S, g) \geq \gamma$. A set will be called *consistent* if it correctly classifies all of the training set. Therefore the computation problem that we want to solve is the following:

Computation 3.1 (Hard margin SVM [18])**.** The Hard margin SVM is obtained by solving the following optimisation problem:

$$\begin{aligned} \max_{\mathbf{w}, b, \gamma} \quad & \gamma \\ \text{subject to} \quad & m(S, g) \geq \gamma \\ & \|\mathbf{w}\|^2 = 1 \end{aligned}$$

Now we want to arrive at the dual optimization problem. To do so, we will derive a Lagrangian

$$L(\mathbf{w}, b, \gamma, \alpha, \lambda) = -\gamma - \sum_{i=1}^{N} \alpha_i \left(y_i(\langle \mathbf{w}, \phi(\mathbf{x_i})\rangle + b) - \gamma\right) + \lambda\left(\|\mathbf{w}\|^2 - 1\right).$$

Now, we want to find the equilibrium points of this Lagragian with respect to their parameters:

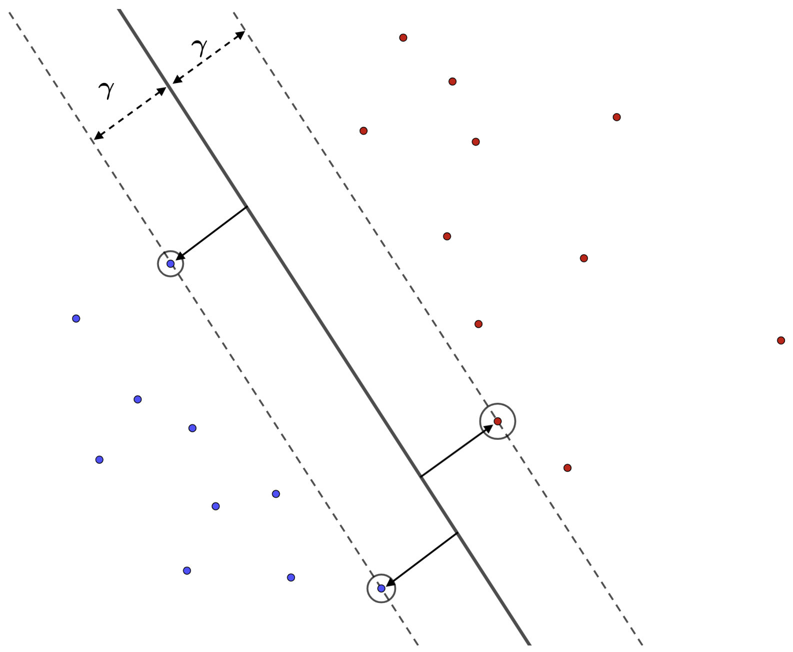

Figure 1: Hard SVM with margin γ and support vectors cicled.

$$\frac{\partial L(\mathbf{w}, b, \gamma, \alpha, \lambda)}{\partial \mathbf{w}} = -\sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x_i}) + 2\lambda \mathbf{w} = \mathbf{0},$$
$$\frac{\partial L(\mathbf{w}, b, \gamma, \alpha, \lambda)}{\partial \gamma} = -1 + \sum_{i=1}^{N} \alpha_i = 0, \quad \text{and}$$
$$\frac{\partial L(\mathbf{w}, b, \gamma, \alpha, \lambda)}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0.$$

Substituting in the original expression, we can write the function as a function of the kernel matrix

$$L(\mathbf{w}, b, \gamma, \alpha, \lambda) = -\frac{1}{4\lambda} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x_i}, \mathbf{x_j}) - \lambda.$$

To eliminate λ we optimise the function with respect to it, obtaining

$$\lambda = \frac{1}{2} \left(\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x_i}, \mathbf{x_j}) \right)^{1/2},$$

finally

$$L(\alpha) = -\left(\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \right)^{1/2},$$

this expression is called the *dual Lagragian.* We have the following algorithm:

Algorithm 3.2 (Hard Margin SVM [18])**.** The hard margin SVM is given by the following algorithm.

Input	training set $S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\mathbf{N}, y_N)$
Process	find α^* as solution to the optimisation problem:
maximise	$W(\alpha) = -\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$
subject to	$\sum_{i=1}^{N} y_i \alpha_i = 0,\ \sum_{i=1}^{N} \alpha_i = 1$ and $0 \le \alpha_i, i = 1, \cdots, N$.
4	$\gamma^* = \sqrt{-W(\alpha^*)}$
5	choose i such that $0 < \alpha_i^*$
6	$b = y_i(\gamma^*)^2 - \sum_{j=1}^{N} y_j \kappa(\mathbf{x}_j, \mathbf{x}_i)$
7	$f(\cdot) = \operatorname{sgn}\left(\sum_{j=1}^{N} \alpha_j^* y_j \kappa(x_j, \cdot) + b\right)$;
8	$\mathbf{w} = \sum_{j=1}^{N} y_j \alpha_j^* \phi(\mathbf{x}_j)$
Output	weight vector $\mathbf{w}$, dual solution α^*, margin γ^* and function f implementing the decision rule represented by the hyperplane

Table 1: Hard margin SVM algorithm [18]

The following theorem characterizes the output and stability of the algorithm.

Theorem 3.3 (SVM characterization [18])**.** *Fix $\delta > 0$. Suppose that a training sample*

$$S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\},$$

is drawn according to a distribution $\mathcal{D}$ is linearly separable in the feature space implicitly defined by the kernel κ and suppose that the Hard margin SVM algorithm outputs $\mathbf{w}, \alpha^*.\gamma^*$ and the function f. Then the function f realises the hard margin support vector machine in the feature space defined by κ with geometric margin γ^*. Furthermore, with probability $1-\delta$, the generalisation error of the resulting classifier is bounded by

$$\frac{4}{N\gamma^*}\sqrt{tr(\mathbf{K})} + 3\sqrt{\frac{\ln(2/\delta)}{2N}}$$

where $\mathbf{K}$ is the corresponding kernel matrix.

Proof. See page 216 [18]. □

It is important to notice that from The Karush-Kuhn-Tucker (KKT) complementary conditions [4][20] in convex programming the optimal solutions α^*, $(\mathbf{w}^*, b)$ must satisfy

$$\alpha_i^*[y_i(\langle \mathbf{w}^*, \phi(\mathbf{x}_i)\rangle + b^*) - \gamma^*] = 0, \quad i = 1, \dots, N.$$

Which implies that only the $\mathbf{x}_i$ that lie closest to the hyperplane have their corresponding α_i^* non-zero. For this reason the inputs with non-zero α_i^* are called *support vectors.*

Another important remark is concerning the convexity of the optimization problem. The convexity of the function we want to optimize ensures that we do not fall into local minima. We already know that the kernel matrix is convex from the semi-definite positive property. Now, we want to check whether the matrix $\mathbf{G} = (y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}^N$ is also positive semi-definite

$$\begin{aligned}\beta' \mathbf{G} \beta &= \sum_{i,j=1}^N \beta_i \beta_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \\ &= \left\langle \sum_{i=1}^N \beta_i y_i \phi(\mathbf{x}_i), \sum_{j=1}^N \beta_j y_j \phi(\mathbf{x}_j) \right\rangle = \left\| \sum_{i=1}^N \beta_i y_i \phi(\mathbf{x}_i) \right\|^2 \geq 0.\end{aligned}$$

This equation shows an essential property of kernel methods. For the kernel to define a feature space, it must be positive definite; this property ensures that we do not fall into local minima and that a unique solution to the optimization problem exists.

3.1.2 Soft margin classifier

Although the hard margin SVM is an important concept, its practical uses are limited as, in most cases, the given data will not be naturally linearly separable in the embedded space. This makes hard margin SVM a non-robust estimator. We want to develop a more robust version that can adjust to noisy data and outliers in the training data set. For that reason, we will allow some points to not be separated by the hyperplane. Therefore not all components of

$$\xi_i = (\gamma - y_i g(\mathbf{x}_i))_+$$

will be equal to zero. This give the following criterion.

Computation 3.4 (1-norm soft margin SVM [18])**.** The 1-norm soft margin SVM is obtained by solving the following optimisation problem

$$\begin{aligned} \min_{\mathbf{w}, b, \gamma, \xi} \quad & -\gamma + C \sum_{i=1}^N \xi_i \\ \text{subject to} \quad & y_i(\langle \mathbf{w}, \phi(\mathbf{x_i}) + b \rangle) \geq \gamma - \xi_i, \xi_i \geq 0, \\ & i = 1, \ldots, N, \text{ and } \|\mathbf{w}\|^2 = 1. \end{aligned}$$

The parameter C controls the trade-off between the margin and the size of the slack variables.

In the same way, as we did for the hard margin SVM, we will study the Lagrangian corresponding to the 1-norm soft margin optimization problem.

$$L(\mathbf{w}, b, \gamma, \mathbf{xi}, \alpha, \beta, \gamma) = -\gamma + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i [y_i(\langle \phi(\mathbf{x_i})), \mathbf{w}) - \gamma + \xi_i] \\ - \sum_{i=1}^{N} \beta_i \xi_i + \lambda \left(\|\mathbf{w}\|^2 - 1 \right)$$

with $\alpha_i \geq 0$ and $\beta_i \geq 0$. That can be adapted to the dual objective function

$$L(\alpha) = -\left(\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x_i}, \mathbf{x_j}) \right)^{1/2}.$$

Therefore, we have the algorithm for the soft margin SVM and the theorem that characterize it.

Theorem 3.5 (1-norm soft margin support vector machine [18]). *Fix $\delta > 0$ and $C \in [1/N, \infty)$. Suppose that a training sample*

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$$

is drawn according to a distribution $\mathcal{D}$. The following algorithm outputs the weight vector $\mathbf{w}$, the dual solution α^*, margin γ^* and function f implementing the decision rule represented by the hyperplane.

Input	training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x_N}, y_N)$
Process	find α^* as solution to the optimisation problem:
maximise	$W(\alpha) = -\sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$
subject to	$\sum_{i=1}^{N} y_i \alpha_i = 0, \ \sum_{i=1}^{N} \alpha_i = 1$ and $0 \leq \alpha_i \leq C, i = 1, \cdots, N$.
4	$\lambda^* = \frac{1}{2} \left(\sum_{i,j=1}^{N} y_i y_j \alpha_i^* \alpha_j^* \kappa(\mathbf{x_i}, \mathbf{x_j}) \right)^{1/2}$
5	choose i, j such that $-C < \alpha_i^* y_i < 0 < \alpha_j^* y_j < C$
6	$b^* = -\lambda^* \left(\sum_{k=1}^{N} \alpha_k^* y_k \kappa(\mathbf{x}_k, \mathbf{x}_i) + \sum_{k=1}^{N} \alpha_{k=1}^{N} \alpha_k^* y_k \kappa(\mathbf{x}_k \mathbf{x}_j) \right)$
7	$2\lambda^* \sum_{k=1}^{N} \alpha_k^* y_k \kappa(\mathbf{x}_k, \mathbf{x}_j) + b^*$
8	$f(\cdot) = \text{sgn} \left(\sum_{j=1}^{N} \alpha_j^* y_j \kappa(\mathbf{x}_j, \cdot) + b^* \right)$;
9	$\mathbf{w} = \sum_{j=1}^{N} y_j \alpha_j^* \phi(\mathbf{x}_j)$
Output	weight vector $\mathbf{w}$, dual solution α^*, margin γ^* and function f implementing the decision rule represented by the hyperplane

Table 2: 1-norm soft margin SVM [18].

Then the function f realises the 1-norm soft margin support vector machine in the feature space defined by κ. Furthermore, with probability $1 - \delta$, the generalisation error is bounded by

$$\frac{1}{CN} - \frac{\sqrt{-W(\alpha^*)}}{CN\gamma^*} + \frac{4}{N\gamma^*}\sqrt{tr(\mathbf{K})} + 3\sqrt{\frac{\ln(2/\delta)d}{2N}},$$

where $\mathbf{K}$ is the corresponding kernel matrix.

Proof. See page 233 [18]. □

If we compare this algorithm to the Hard Margin SVM, we realize that we have to solve the same optimization with the additional constraint that C bounds all the α_i. For this reason, this constrain is called the *box constrain*, and it limits the influence of outliers. Moreover, an important problem is finding the parameter C that best fits the data. This can be done with *crossvalidation* techniques [24].

3.1.3 ν-support vector machine

As the value of C must satisfy that $C \geq 1/N$, this suggest using

$$C = \frac{1}{\nu N}, \quad \nu \in (0,1]$$

as a way to have control over the number of outliers. This give rise to the ν-support vector machine.

Theorem 3.6 (ν-support vector machine [18]). *Fix $\delta > 0$ and $\nu \in (0,1]$. Suppose that a training sample*

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$$

is drawn according to a distribution $\mathcal{D}$. he following algorithm outputs the weight vector $\mathbf{w}$, the dual solution α^*, margin γ^* and function f implementing the decision rule represented by the hyperplane.

Input	training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x_N}, y_N)$
Process	find α^* as solution to the optimisation problem:
maximise	$W(\alpha) = -\sum_{i,j=1}^N \alpha_i\alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j)$
subject to	$\sum_{i=1}^N y_i\alpha_i = 0, \ \sum_{i=1}^N \alpha_i = 1$ and $0 \leq \alpha_i \leq 1/(\nu N), i = 1, \cdots, N.$
4	$\lambda^* = \frac{1}{2}\left(\sum_{i,j=1}^N y_i y_j \alpha_i^* \alpha_j^* \kappa(\mathbf{x_i}, \mathbf{x_j})\right)^{1/2}$
5	choose i, j such that $-1/(\nu N) < \alpha_i^* y_i < 0 < \alpha_j^* y_j < 1/(\nu N)$
6	$b^* = -\lambda^* \left(\sum_{k=1}^N \alpha_k^* y_k \kappa(\mathbf{x}_k, \mathbf{x}_i) + \sum_{k=1}^N \alpha_{k=1}^N \alpha_k^* y_k \kappa(\mathbf{x}_k \mathbf{x}_j)\right)$
7	$2\lambda^* \sum_{k=1}^N \alpha_k^* y_k \kappa(\mathbf{x}_k, \mathbf{x}_j) + b^*$
8	$f(\cdot) = \operatorname{sgn}\left(\sum_{j=1}^N \alpha_j^* y_j \kappa(\mathbf{x}_j, \cdot) + b^*\right)$;
9	$\mathbf{w} = \sum_{j=1}^N y_j \alpha_j^* \phi(\mathbf{x}_j)$
Output	weight vector $\mathbf{w}$, dual solution α^*, margin γ^* and function f implementing the decision rule represented by the hyperplane

Table 3: ν-norm soft margin SVM.

Then the function f realises the 1-norm soft margin support vector machine in the feature space defined by κ. Furthermore, with probability $1 - \delta$, the generalisation error is bounded by

$$\nu - \frac{\nu\sqrt{-W(\alpha^*)}}{\gamma^*} + \frac{4}{N\gamma^*}\sqrt{tr(\mathbf{K})} + 3\sqrt{\frac{\ln(2/\delta}{2N}},$$

where $\mathbf{K}$ *is the corresponding kernel matrix. Furthermore, there are at most* νN *training points that fail to achieve a margin* γ^**, while at least* νl *of the training points have margin at most* γ^**.*

Proof. See page 226 [18]. □

The parameter ν corresponds to the noise level inherent in the data. It imposes a lower bound on the generalization error achievable by any learning algorithm. Again, the parameter ν is an *hyperparameter* and must be specified by the user of the algorithm or by using a technique such as *crossvalidation* [24].

3.1.4 2-norm soft margin SVM

Previously we have used the 1-norm regularization in the derivation of the SVM. This norm is not the only choice for the norm of the regularization term. Depending on the regularization term used, different results will be obtained. The *1-norm* regularization term, also known as *L1 regularization* or *Lasso regularization.* Allows the less important feature's coefficients to be zero. In this way, when we have many features, it can be used as a *feature selection* technique. On the other hand, the *2-norm* regularization term, also known as the *L2 regularization* or *Rigdge regularization*, only acts as a smoother of the features. The bigger the associated coefficients, the smoother the result will be. We introduce the *2-norm* soft margin SVM.

Computation 3.7 (2-norm soft margin SVM [18])**.** The 2-norm soft margin SVM is obtained by solving the following optimisation problem

$$\begin{array}{ll} \min\limits_{\mathbf{w},b,\gamma,\xi} & -\gamma + C\sum_{i=1}^{N}\xi_i^2 \\ \text{subject to} & y_i(\langle \mathbf{w}, \phi(\mathbf{x_i}) + b\rangle) \geq \gamma - \xi_i, \\ & i = 1,\dots,N, \text{ and } \|\mathbf{w}\|^2 = 1. \end{array}$$

Obtaining again the dual formulation of the optimization problem:

$$L(\alpha,\lambda) = -\frac{1}{4C}\sum_{i=1}^{N}\alpha_i^2 - \left(\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j k(\mathbf{x}_i,\mathbf{x}_j)\right)^{1/2}. \tag{3.4}$$

The 2-norm regularization of the primal problem corresponds to regularising the dual with the 2-norm of the Lagrange multipliers. For a given C maximising the above objective over α is equivalent to maximising:

$$\begin{aligned} W(\alpha) &= -\mu\sum_{i=1}^{N}\alpha_i^2 - \sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j k(\mathbf{x}_i,\mathbf{x}_j) \\ &= -\sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j (k(\mathbf{x}_i,\mathbf{x}_j) + \mu\delta_{ij}), \end{aligned}$$

For a $\mu = \mu(C)$. This is just the hard-margin algorithm but with the objective function depending on $(k(\mathbf{x}_i, \mathbf{x}_j) + \mu\delta_{ij})$ instead of the kernel $k(\mathbf{x}_i, \mathbf{x}_j)$.

Theorem 3.8 (2-norm soft margin SVM characterization [18]). *The 2-norm soft margin SVM is given by the following algorithm:*

Input	training set $S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\mathbf{N}, y_N)$
Process	find α^* as solution to the optimisation problem:
maximise	$W(\alpha) = -\sum_{i,j=1}^N \alpha_i\alpha_j y_i y_j (k(\mathbf{x}_i, \mathbf{x}_j) + \mu\delta_{ij})$
subject to	$\sum_{i=1}^N y_i\alpha_i = 0,\ \sum_{i=1}^N \alpha_i = 1$ and $0 \leq \alpha_i, i = 1, \cdots, N$.
4	$\gamma^* = \sqrt{-W(\alpha^*)}$
5	choose i such that $0 < \alpha_i^*$
6	$b = y_i(\gamma^*)^2 - \sum_{j=1}^N y_j\kappa(\mathbf{x}_j, \mathbf{x}_i)$
7	$f(\cdot) = \operatorname{sgn}\left(\sum_{j=1}^N \alpha_j^* y_j (k(\mathbf{x}_i, \mathbf{x}_j) + \mu\delta_{ij}) + b\right)$;
8	$\mathbf{w} = \sum_{j=1}^N y_j\alpha_j^*\phi(\mathbf{x}_j)$
Output	weight vector $\mathbf{w}$, dual solution α^*, margin γ^* and function f implementing the decision rule represented by the hyperplane

Table 4: 2-norm soft-margin SVM algorithm [18]

Moreover, given $\delta > 0$. Suppose that a training sample $S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\mathbf{N}, y_N)$ drawn according to a distribution $\mathcal{D}$ in the feature space implicitly defined by the kernel k and suppose that the 2-norm soft-margin SVM algorithm outputs $\mathbf{w}, \alpha^*, \gamma^*$ and the function f. Then the function f realises the hard margin support vector machine in the feature space defined by $(k(\mathbf{x}_i, \mathbf{x}_j) + \mu\delta_{ij})$ with geometric margin γ^*. This is equivalent to minimising the expression $-\gamma + C\sum_{i=1}^N \xi_i^2$ involving the 2-norm of the slack variable for some value of C. hence realising the 2-norm support vector machine. Furthermore, with probability $1-\delta$, the generalisation error of the resulting classifier is bounded by:

$$\min\left(\frac{\mu\left\|\alpha^*\right\|^2}{N\gamma^{*4}} + \frac{8\sqrt{tr(\mathbf{K})}}{N\gamma^*} + 3\frac{\ln 4/\delta}{2N}, \frac{4\sqrt{tr(\mathbf{K}) + N\mu}}{N\gamma^*} + 3\sqrt{\frac{\ln 4/\delta}{2N}}\right) \tag{3.5}$$

Proof. See page 230 [18]. □

3.2 Support vector machines for regression

SVM as defined by Vapnik [12] can be implemented for regression problems. This can be done by choosing an appropriate loss function. Regression consists of finding a function that best fits the data. This will give rise to what we will call Support Vector Regression (SVR). In order to derive the SVR algorithm, first, we will study the ε-insensitive regression.

3.2.1 ε-insensitive regression

We will develop a regression algorithm that ignores errors smaller than a certain threshold $\varepsilon > 0$. This restriction will generate a band around the output, referred to as a *tube*. This type of loss function is referred to as an ε-insensitive loss function.

Definition 3.9 (Linear and quadratic ε-insensitive loss function [18]). The linear ε-insensitive loss function $\mathcal{L}^{\varepsilon}(\mathbf{x}, y, g)$ is defined by

$$\mathcal{L}^{\varepsilon}(\mathbf{x}, y, g) = |y - g(\mathbf{x})|_{\varepsilon} = \max(0, |y - g(\mathbf{x})| - \varepsilon),$$

where g is a real-valued function on a domain X, $\mathbf{x} \in X$ and $y \in \mathbb{R}$. Similarly the quadratic ε-insensitive loss is given by

$$\mathcal{L}_2^{\varepsilon}(\mathbf{x}, y, g) = |y - g(\mathbf{x})|_{\varepsilon}^2.$$

The objective will be to optimize the sum of the quadratic ε-insensitive losses subject to the constraint that the norm is bounded. We have, therefore, the following algorithm.

Computation 3.10 (Quadratic ε-insensitive SVR [18]). The weight vector $\mathbf{w}$ and the threshold b for the quadratic ε-insensitive support vector regression are chosen to optimise the following problem

$$\begin{aligned}
\min_{\mathbf{w},b,\xi,\hat{\xi}} \quad & \|\mathbf{w}\|^2 + C\sum_{i=1}^{N}(\xi_i^2 + \hat{\xi}_i^2), \\
\text{subject to} \quad & (\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b) - y_i \leq \varepsilon + \xi_i, i = 1, \dots, N, \\
& y_i - (\langle \mathbf{w}, \phi(\mathbf{x}_i\rangle + b) \leq \varepsilon + \hat{\xi}_i, i = 1, \dots, N.
\end{aligned}$$

Using again the Lagragian function method to derive the dual problem obtain the following problem.

$$\begin{aligned}
\max_{\alpha,\hat{\alpha}} \quad & \sum_{i=1}^{N} y_i(\hat{\alpha}_i - \alpha_i) - \varepsilon\sum_{i=1}^{N}(\hat{\alpha}_i + \alpha_i) - \frac{1}{2}\sum_{i,j=1}^{N}(\hat{\alpha}_i - \alpha_i)(\hat{\alpha}_i - \alpha_j)(\kappa(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{C}\delta_{ij}), \\
\text{subject to} \quad & \sum_{i=1}^{N}(\hat{\alpha}_i - \alpha_i) = 0 \\
& \hat{\alpha}_i \geq 0, \alpha_i \geq 0, i = 1, \dots, N.
\end{aligned}$$

And the corresponding KKT complementary conditions are

$$\begin{aligned}
\alpha_i(\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b - y_i - \varepsilon - \xi_i) = 0, &\quad i = 1, \dots, N, \\
\hat{\alpha}_i\left(y_i - \langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle - b - \varepsilon - \hat{\xi}_i\right) = 0, &\quad i = 1, \dots, N, \\
\xi_i\hat{\xi}_i = 0,\ \alpha_i\hat{\alpha}_i = 0, &\quad i = 1, \dots, N.
\end{aligned}$$

From which the following algorithm is obtained.

Input	training set $S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\mathbf{N}, y_N),\ C > 0$
Process	find α^* as solution to the optimisation problem:
maximise	$W(\alpha) = \sum_{i=1}^N y_i\alpha_i - \varepsilon \sum_{i=1}^N \lvert\alpha_i\rvert - \frac{1}{2}\sum_{i,j=1}^N \alpha_i\alpha_j(\kappa(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{C}\delta_{ij})$
subject to	$\sum_{i=1}^N \alpha_i = 0,$
4	$\mathbf{w} = \sum_{j=1}^N \alpha_j^*\phi(\mathbf{x}_j)$
5	$b^* = -\varepsilon - (\alpha_i^*/C) + y_i - \sum_{j=1}^N \alpha_j^*\kappa(\mathbf{x}_j, \mathbf{x}_i)$ for i with $\alpha_i^* > 0.$
6	$f(\mathbf{x}) = \sum_{j=1}^N \alpha_j^*\kappa(\mathbf{x}_j, \mathbf{x}) + b^*,$
Output	weight vector $\mathbf{w}$, dual solution α^*, margin γ^* and function f implementing 2-norm SVR.

Table 5: 2-norm support vector regression [18].

Algorithm 3.11 (2-norm support vector regression [18])**.** The following algorithm implement the 2-norm support vector regression algorithm.

In the same manner we can implement the regression algorithm but with a linear loss.

Computation 3.12 (Linear ε-insensitive SVR [18])**.** The weight vector $\mathbf{w}$ and the threshold b for the linear ε-insensitive support vector regression are chosen to optimise the following problem

$$\begin{aligned}
\min_{\mathbf{w},b,\xi,\hat{\xi}} \quad & \|\mathbf{w}\|^2 + C\sum_{i=1}^N(\xi_i + \hat{\xi}_i),\\
\text{subject to} \quad & (\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b) - y_i \leq \varepsilon + \xi_i, i = 1, \dots, N,\\
& y_i - (\langle \mathbf{w}, \phi(\mathbf{x}_i\rangle + b) \leq \varepsilon + \hat{\xi}_i, i = 1, \dots, N,\\
& \xi_i, \hat{\xi}_i \geq 0, i = 1, \dots, N.
\end{aligned}$$

And the corresponding dual problem can be derived as always.

$$\begin{aligned}
\max_{\alpha,\hat{\alpha}} \quad & \sum_{i=1}^N y_i(\hat{\alpha}_i - \alpha_i) - \varepsilon\sum_{i=1}^N(\hat{\alpha}_i + \alpha_i) - \frac{1}{2}\sum_{i,j=1}^N \hat{\alpha}_i - \alpha_i)(\hat{\alpha}_i - \alpha_j)(\kappa(\mathbf{x}_i, \mathbf{x}_j),\\
\text{subject to} \quad & \sum_{i=1}^N(\hat{\alpha}_i - \alpha_i) = 0\\
& 0 \geq \hat{\alpha}_i, \alpha_i \leq C, i = 1, \dots, N.
\end{aligned}$$

And the following algorithm is obtained:

Algorithm 3.13 (1-norm support vector regression [18])**.** The following algorithm implement the 1-norm support vector regression algorithm.

The algorithm is called *Support Vector Regression* because if we take a band of $\pm\varepsilon$ around the function output by the learning algorithm, the points that are not strictly inside the tube are called *support vectors.*

Input	training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x_N}, y_N),\ C > 0$
Process	find α^* as solution to the optimisation problem:
maximise	$W(\alpha) = \sum_{i=1}^N y_i\alpha_i - \varepsilon \sum_{i=1}^N \lvert\alpha_i\rvert - \frac{1}{2}\sum_{i,j=1}^N \alpha_i\alpha_j(k(\mathbf{x}_i, \mathbf{x}_j)$
subject to	$\sum_{i=1}^N \alpha_i = 0,\ -C \le \alpha_i \le C, i = 1, \ldots, l.$
4	$\mathbf{w} = \sum_{j=1}^N \alpha_j^*\phi(\mathbf{x}_j)$
5	$b^* = -\varepsilon + y_i - \sum_{j=1}^N \alpha_j^*\kappa(\mathbf{x}_j, \mathbf{x}_i)$ for i with $0 < \alpha_i^* < C$.
6	$f(\mathbf{x}) = \sum_{j=1}^N \alpha_j^*\kappa(\mathbf{x}_j, \mathbf{x}) + b^*$,
Output	weight vector $\mathbf{w}$, dual solution α^*, margin γ^* and function f implementing 2-norm SVR.

Table 6: 1-norm support vector regression [18].

At last we can give a result about the generalization power of this algorithm.

Theorem 3.14 (Characterization of the ε-insensitive regression [18])**.** *Fix $B > 0$ and $\delta \in (0, 1)$. Let $\mathcal{F}_B$ be the class of linear functions with norm at most B, mapping from a feature space defined by the kernel k over a space $\mathcal{X}$. Let*

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_l)\} \tag{3.10}$$

be drawn iid according to a probability distribution $\mathcal{D}$ on $\mathcal{X} \times \mathbb{R}$. Then with probability at least $1 - \delta$ over the random draw of S, we have for all $g \in \mathcal{F}_B$

$$\mathbb{P}_{\mathcal{D}}(|y - g(\mathbf{x})| > \gamma) \le \frac{\left\|\xi + \hat{\xi}\right\|_1}{N(\gamma - \varepsilon)} + \frac{4B\sqrt{tr(\mathbf{K})}}{N(\gamma - \varepsilon)} + 3\sqrt{\frac{\ln 2/\gamma}{2N}} \tag{3.11}$$

where $\mathbf{K}$ is the kernel matrix of the training set S.

Proof. [18]. □

3.2.2 ν-support vector regression

In our derivation of the SVM, we obtained that in the ν-SVM the hyperparameter ν represented the fraction of of support vectors. We can adopt the same approach in what is known as the ν-support vector regression.

Computation 3.15 (ν- support vector regression [18])**.** The weight vector $\mathbf{w}$ and threshold b for the ν-supprot vector regression are chosen to optimise the following problem:

$$\left.\begin{array}{ll} \min_{\mathbf{w},b,\varepsilon,\boldsymbol{\xi},\hat{\boldsymbol{\xi}}} & \frac{1}{2}\|\mathbf{w}\|^2 + C\left(\nu\varepsilon + \frac{1}{N}\sum_{i=1}^N\left(\xi_i + \hat{\xi}_i\right)\right), \\ \text{subject to} & (\langle \mathbf{w}, \phi(\mathbf{x}_i)\rangle + b) - y_i \le \varepsilon + \xi_i \\ & y_i - (\langle \mathbf{w}, \boldsymbol{\phi}(\mathbf{x}_i)\rangle + b) \le \varepsilon + \hat{\xi}_i, \\ & \xi_i, \hat{\xi}_i \ge 0, i = 1, \ldots, N, \end{array}\right\} \tag{3.12}$$

After the usual analysis the following algorithm is obtained.

Algorithm 3.16 (ν-support vector regression [18])**.** The ν-support vector regression algorithm is implemented as:

Input	training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\mathbf{N}, y_N),\ C > 0$
Process	find α^* as solution to the optimisation problem:
maximise	$W(\alpha) = \sum_{i=1}^N y_i\alpha_i - \varepsilon \sum_{i=1}^N \lvert\alpha_i\rvert - \frac{1}{2}\sum_{i,j=1}^N \alpha_i\alpha_j(k(\mathbf{x}_i, \mathbf{x}_j)$
subject to	$\sum_{i=1}^N \alpha_i = 0,\ \sum_{i=1}^N \lvert\alpha_i\rvert \leq C\nu,\ -C/N \leq \alpha_i \leq C/N, i = 1, \ldots, l.$
4	$\mathbf{w} = \sum_{j=1}^N \alpha_j^* \phi(\mathbf{x}_j)$
5	$b^* = -\varepsilon + y_i - \sum_{j=1}^N \alpha_j^* \kappa(\mathbf{x}_j, \mathbf{x}_i)$ for i with $0 < \alpha_i^* < C/N$.
6	$f(\mathbf{x}) = \sum_{j=1}^N \alpha_j^* \kappa(\mathbf{x}_j, \mathbf{x}) + b^*$,
Output	weight vector $\mathbf{w}$, dual solution α^*, margin γ^* and function f implementing 2-norm SVR.

Table 7: ν-norm support vector regression [18].

In this algorithm, the parameter ν controls the fraction of training points that fall outside the tube, i.e., there are at most νN training points outside the tube. Moreover, at least νN of the training points are support vectors.

3.3 The perceptron

In this section we present *the perceptron* an *online* classification algorithms. Contrary to the previous algorithms trained in *batches*, the online algorithm processes the data at the same time it is received. At each time, the algorithm predicts the correct output. The true output is then made available, and the degree of mismatch or loss is recorded. The learner then adjusts according to the feedback received. The learner's objective is to adjust to the underlying patterns of the data at quickly as possible. The percepton was one of the first algorithms of this kind [5]. It was designed to simulate a neuron of the brain. This algorithm is the start of the field of *Artificial Neural Networks* (ANN) and *Deep Learnin* (DL) [23]. These are two of the most successful techniques in contemporary machine learning and the object of much research. However, although NN presents impressive empirical performance, the theoretical basis of its success is still unknown. For historical reasons and to introduce this field, we present the *perceptron* in this work.

The perceptron learns a threshold linear function:

$$h(\mathbf{x}) = \operatorname{sgn}(\langle \mathbf{w}, \phi(\mathbf{x})\rangle). \tag{3.13}$$

The algorithm make an update whenever a misclassified exampled is processed. The updating rule is given by:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y_i\phi(\mathbf{x}_i). \tag{3.14}$$

The corresponding dual update rule is given by:

$$\alpha_i = \alpha_i + 1, \tag{3.15}$$

the weight vector is expressed as:

$$\mathbf{w}_t = \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i). \tag{3.16}$$

Then the corresponding algorithm is:

Algorithm 3.17 (Kernel perceptron [18])**.** The dual perceptron algorithm is implemented as:

```
Input     | training sequence $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
Process   | $\boldsymbol{\alpha} = 0, i = 0,$ loss $= 0$
2         |  repeat
3         |    $i = i + 1$
4         |      if sgn$\left(\sum_{j=1}^{N} \alpha_j y_j \kappa(\mathbf{x}_j, \mathbf{x}_i)\right) \neq y_i$      (3.17)
5         |          $\alpha_i = \alpha_i + 1$
6         |          loss $=$ loss $+1$
7         |  until finished
8         | $f(\mathbf{x}) = \sum_{j=1}^{\ell} \alpha_j y_j \kappa(\mathbf{x}_j, \mathbf{x})$
Output    |  dual variables $\boldsymbol{\alpha}$, loss and function $f$
```

In order to present a result of the algorithm's performance, we must suppose that the algorithm is trained in batches. In this case the following theorem about the performance of the algorithm follows.

Theorem 3.18. *Perceptron characterization [18] Fix $\delta > 0$. Suppose the hard margin support vector machine has margin γ on the training set:*

$$S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\} \tag{3.18}$$

drawn iid according to a distribution $\mathcal{D}$ and contained in a ball of radius R about the origin. Then with probability at least $1 - \delta$ over the draw of the set S, the generalisation error of the function $f(\mathbf{x})$ obtained by running the perceptron algorithm on S in batch mode in bounded by:

$$\mathbb{P}_{\mathcal{D}}(f(\mathbf{x}) \neq y) \leq \frac{2}{N}\left(\frac{R^2}{\gamma^2}\ln N + \ln\frac{N}{2\delta}\right), \tag{3.19}$$

provided

$$\frac{R^2}{\gamma^2} \leq \frac{N}{2}. \tag{3.20}$$

This theorem follows from a classical result of Novikoff [6]. It states that the number of updates that the perceptron algorithm needs to generalize is bounded by:

$$\frac{R^2}{\gamma^2} \tag{3.21}$$

With this result, we end the discussion about perceptrons as they were first introduced by Rosenblatt [5].

4 Kernels in Unsupervised Learning

Unsupervised learning consists in extracting valuable patterns from unlabeled data. We will study algorithms to cluster data into categories and algorithms for visualization purposes. Principal components analysis (PCA) algorithms try to find a set of k directions in the embedding space containing the maximum amount of variance in the data. We will also study how to find correlations between two different representations of the same data (canonical correlation analysis (CCA)). All these problems can be reduced by performing an eigenvalue decomposition. Moreover, they can be solved or approximated efficiently using several well-known techniques from computational linear algebra. We will show how they can be solved in kernel-defined feature space by writing the dual representation of the algorithms.

4.1 Kernel Principal Component Analysis

4.1.1 Principal Component Analysis

The goal of this section is to find a direction that maximises the variance in the feature space. The first step in order to perform PCA is to center the data in feature space. We show how to do this for illustration purposes. This operation can be obtained implicitly by transforming the kernel matrix. The new feature map after centering the data set is:

$$\hat{\phi}(\mathbf{x}) = \phi(\mathbf{x}) - \phi_S = \phi(\mathbf{x}) - \frac{1}{N}\sum_{i=1}^{N}\phi(\mathbf{x}_i). \tag{4.1}$$

The new kernel for this space is:

$$\begin{aligned}\hat{k}(\mathbf{x},\mathbf{z}) = \left\langle \hat{\phi}(\mathbf{x}), \hat{\phi}(\mathbf{z})\right\rangle &= \left\langle \phi(\mathbf{x}) - \frac{1}{N}\sum_{i=1}^{N}\phi(\mathbf{x}_i), \phi(\mathbf{z}) - \frac{1}{N}\sum_{i=1}^{N}\phi(\mathbf{x}_i)\right\rangle \\ &= k(\mathbf{x},\mathbf{z}) - \frac{1}{N}\sum_{i=1}^{N}k(\mathbf{x},\mathbf{x}_i) - \frac{1}{N}\sum_{i=1}^{N}k(\mathbf{z},\mathbf{x}_i) + \frac{1}{N^2}\sum_{i,j=1}^{N}k(\mathbf{x_i},\mathbf{x}_j).\end{aligned}$$

In terms of operations on the kernel matrix this can be written as:

$$\hat{\mathbf{K}} = \mathbf{K} - \frac{1}{N}\mathbf{jj}'\mathbf{K} - \frac{1}{N}\mathbf{Kjj}' + \frac{1}{N^2}(\mathbf{j}'\mathbf{Kj})\mathbf{jj}', \tag{4.2}$$

where $\mathbf{j}$ represents the vector whose components are all 1.

If the data have been centred in the feature space then the variance of the projection into a normalised direction $\mathbf{w}$ can be obtained as follows

$$\begin{aligned}\frac{1}{N}\sum_{i=1}^{N}(P_{\mathbf{w}}(\phi(\mathbf{x}_i)))^2 &= \hat{\mathbb{E}}[\mathbf{w}'\phi(\mathbf{x})\phi(\mathbf{x})'\mathbf{w}] = \mathbf{w}'\hat{\mathbb{E}}[\phi(\mathbf{x})\phi(\mathbf{x})']\mathbf{w} \\ &= \frac{1}{N}\mathbf{w}'\mathbf{X}'\mathbf{X}\mathbf{w} = \mathbf{w}'\mathbf{C}\mathbf{w},\end{aligned}$$

where is $\mathbf{C} = \frac{1}{l}\mathbf{X}'\mathbf{X}$ is the covariance matrix of the data sample. The directions of maximal variance can be found as follows.

Computation 4.1 (Maximising variance [18])**.** The direction that maximises the variance can be found by solving the following problem

$$\begin{aligned}\max_{\mathbf{w}} \quad & \mathbf{w}'\mathbf{C}\mathbf{w}, \\ \text{subject to} \quad & \|\mathbf{w}\|_2 = 1.\end{aligned}$$

This computation, combined with the Rayleigh theorem for eigenvectors, shows that the mutually orthogonal directions of maximum variance in order of decreasing size are given by the eigenvectors of $\mathbf{C}$. Moreover, the corresponding eigenvalue's size equals the variance in the chosen direction. Now we will introduce the concept of PCA.

Definition 4.2 (Principal components analysis)**.** [18] Principal components analysis (PCA) takes an initial subset of the principal axes of the training data and projects the data (both training and test) into the space spanned by this set of eigenvectors. We effectively prepossess a data set by projecting it into the subspace spanned by the first k eigenvectors of the covariance matrix of the training set for some $k < l$. The new coordinates are known as the *principal coordinates* with the eigenvectors referred to as the *principal axes.*

With this definition, the principal component analysis algorithm performs the following computation:

Algorithm 4.3 (Primal principal components analysis [18])**.** The primal principal components analysis performs the following computation:

Input	Data $S = \{\mathbf{x}_1, \dots, \mathbf{x}_N\} \subset \mathbb{R}^n$, dimension k
process	$\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$ $C = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)'$ $[U, \Lambda] = \text{eig}(N\mathbf{C})$ $\tilde{\mathbf{x}}_i = \mathbf{U}_k'\mathbf{x}_i, \ i = 1, \dots, N.$
Output	Transformed data $\hat{S} = \{\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_N\}$.

Table 8: Primal PCA algorithm.

4.1.2 Kernel principal component analysis

We now apply PCA into a kernel-defined feature space using the dual representation of the algorithm to obtain the Kernel PCA. We already know how to compute projections onto the feature space, using the dual representation computed from the eigenvectors and eigenvalues of the kernel matrix. We also want to perform a stability analysis to assess when the resulting projection captures a stable data pattern. We denote by U_k the subspace spanned by the first k eigenvectors in the feature space. The k-dimensional vector projection of new data into this subspace is:

$$P_{U_k}(\phi(\mathbf{x})) = (\mathbf{u}_j'\phi(\mathbf{x}))_{j=1}^k = \left(\sum_{i=1} Nl\alpha_i^j \kappa(\mathbf{x}_i, \mathbf{x})\right)_{j=1}^k,$$

where

$$\alpha^j = \lambda_j^{-1/2}\mathbf{v}_j$$

is given in terms of the corresponding eigenvector and eigenvalue of the kernel matrix. This is the basis of the Kernel PCA algorithm.

Algorithm 4.4 (Kernel PCA [18])**.** The kernel PCA algorithm performs the following computation:

Input	Data $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^n$, dimension k
process	$\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j), i, j = 1, \ldots, N$ $\mathbf{K} - \frac{1}{N}\mathbf{jj}'\mathbf{K} - \frac{1}{N}\mathbf{Kjj}' + \frac{1}{N^2}(\mathbf{j}'\mathbf{Kj})\mathbf{jj}'$, $[V, \Lambda] = \mathrm{eig}(\mathbf{K})$ $\tilde{\mathbf{x}}_i = \left(\sum_{i=1}^N \alpha_i^j \kappa(\mathbf{x}_i, \mathbf{x})\right)_{j=1}^k$.
Output	Transformed data $\hat{S} = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_N\}$.

Table 9: Kernel PCA algorithm.

To assess the stability of the Kernel PCA Method we ask ourselves whether the projections captures new data drawn according to the same distribution as the training data. Therefore, we want that the average residual of the test data is not much larger that the average residual of the training data. For this reason we use the following function as our pattern function

$$f(\mathbf{x}) = \left\|P_{U_k}^{\perp}(\phi(\mathbf{x}))\right\|^2 = \|\phi(\mathbf{x}) - P_{U_k}(\phi(\mathbf{x}))\|^2 = \|\phi(\mathbf{x})\|^2 - \|P_{U_k}(\phi(\mathbf{x}))\|^2,$$

this is the squared norm of the orthogonal projections for the subspace U_k spanned by the first k eigenvectors. We want the expected value of the pattern function to be small

$$\mathbb{E}_{\mathcal{X}}[f(\mathbf{x})] = \mathbb{E}_{\mathcal{X}}\left[\left\|P_{U_k}^{\perp}(\phi(\mathbf{x}))\right\|^2\right] \approx 0.$$

The following theorem gives a bound of this value

Theorem 4.5 (PCA Stability [18]). *If we perform PCA in the feature space defined by a kernel κ then with probability greater than $1-\delta$, for any $1 \leq k \leq l$, if we project new data onto the space U_k spanned by the first k eigenvectors in the feature space, the expected squared residual is bounded by*

$$\mathbb{E}\left[\left\|P_{U_k}^{\perp}(\phi(\mathbf{x})\right\|^2\right] \leq \min_{1\leq t\leq k}\left(\frac{1}{l}\lambda^{>t}(S)+\frac{8}{N}\sqrt{(t+1)\sum_{i=1}^{N}\kappa(\mathbf{x}_i,\mathbf{x}_i)^2}\right)+3R^2\sqrt{\frac{\ln(2N/\delta)}{2N}}, \tag{4.4}$$

where the support of the distribution is in a ball of radius R in the feature space.

This theorem tells us that the expected squared residual of a test point will be small provided the residual eigenvalues are small for some value $t \leq k$, which is modest compared to l. The lesson is that we should use kernel PCA when the eigenvalues become small early in the spectrum. Thus capturing a high proportion of the variance of the data in many dimensions significantly smaller than the same subspace will, with high probability, capture most of the variance of the test data.

4.2 Kernel Cluster Analysis

Another popular algorithm in unsupervised learning is cluster analysis. Cluster analysis tries to discover the internal organization by finding structure within the data in the form of clusters. We refer to clusters as subgroups of data that are"close" together. We use cluster analysis because it helps us better understand the data by breaking it into subsets that are significantly more uniform than the overall dataset. It can also be used as a first step before another learning algorithm. For example, before using a KSVM to classify a dataset, we first divide the data set into clusters and then apply the KSVM within a single cluster. Each application suggests its criterion to assess the quality of the clustering obtained. It typically involves some measure of fit between a data item and the cluster to which it is assigned. Hence, a stable clustering algorithm will give assurances about the expected value of this fit for a new randomly drawn example. This assurance implies that the pattern of clusters identified in the training set is not a random occurrence. However, instead, it characterizes some underlying property of the distribution generating the data. The most common choice for the measure assumes that each cluster has a center and assesses the fit of a point by its squared distance from the cluster's center to which it is assigned. This division naturally creates a *Voronoi diagram* of regions, each containing one of the cluster centers. We will adopt the squared distance criterion for assessing the quality of clustering. We will use a kernel that will ensure that the algorithms can be developed in full generality without specifying the particular similarity measure being used.

To apply cluster analysis, we start with a set of unlabelled data

$$S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\},$$

we want to find an assignment of each point to one of a finite, but not necessarily prespecified, number of N of classes, i.e., we seek a map

$$f : S \to \{1, 2, \dots, N\}.$$

This partition of the data should be chosen among all possible assignments in such a way as to solve the measure of clustering quality given in the following computation.

Computation 4.6 (Cluster Analysis [18])**.** The clustering function should be chosen to optimise

$$f = \arg\min_{f} \sum_{i,f:f_i=f(\mathbf{x}_i)=f(\mathbf{x}_j)=f_j} = \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2,$$

where we have as usual assumed a projection function ϕ into a feature space F, in which the kernel κ computes the inner product

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle.$$

From now on we will denote $f_i = f(\mathbf{x}_i)$. Although this is an interesting first approach from which numerous interesting properties can be deduced, the criterion fails to capture the between cluster separation and only takes into account the within-cluster similarity. A more balanced criterion would be:

$$\min_{f} \left(\sum_{i,j:f_i,f_j} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 - \lambda \sum_{i,f:f_i \neq f_j} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 \right).$$

But this expression can be written as:

$$\begin{aligned}\sum_{i,j:f_i \neq f_j} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x_j})\|^2 &= \sum_{i,j=1}^{l} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x_j})\|^2 - \sum_{i,j:f_i=f_j} \|\phi(\mathbf{x_i}) - \phi(\mathbf{x_j})\|^2 \\ &= A - \sum_{i,j:f_i=f_j} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x_j})\|^2,\end{aligned}$$

where A is a constant that depends on the dataset. Therefore the minimisation problem can be expressed as

$$\min_{f} \left((1+\lambda) \sum_{i,j:f_i=f_j} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 - \lambda A \right),$$

from the two optimisation problems are the same. If we minimize within-cluster distances for a fixed number of clusters we are automatically maximizing the between cluster distances.

Another attractive property of the optimization problem is that we can write the expression as follows.

$$\begin{aligned}
\mathbf{opt} &= \sum_{i,j:f_i=f_j} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 \\
&= \sum_{k=1}^{N} \sum_{i:f_i=k} \sum_{j:f_j=k} \langle \phi(\mathbf{x}_i) - \phi(\mathbf{x}_j), \phi(\mathbf{x}_i - \phi(\mathbf{x}_j)\rangle \\
&= \sum_{k=1}^{N} 2\left(|f^{-1}(k)| \sum_{i:f_i=k} \kappa(\mathbf{x}_i, \mathbf{x_i})\right) - \sum_{i:f_i=k} \sum_{j:f_j=k} \kappa(\mathbf{x}_i, \mathbf{x}_j) \\
&= \sum_{k=1}^{N} 2|f^{-1}(k)| \sum_{i:f_i=k} \|\phi(\mathbf{x}_i) - \mu_k\|^2,
\end{aligned}$$

where in the last line we have expressed

$$\mu_k = \frac{1}{|f^{-1}(k)|} \sum_{i \in f^{-1}(k)} \phi(\mathbf{x}_i)$$

as the centre of mass of the cluster k, a point that it is usually referred to as the *centroid* of the cluster. Therefore the optimisation criterion is equivalent to

$$f = \arg\min_{f} \sum_{i=1}^{l} \left\|\phi(\mathbf{x}_i) - \mu_{f(\mathbf{x}_i)}\right\|^2,$$

that seeks a clustering of points minimising the sum-squared distances to the centres of mass of the clusters. The following theorem formalise this idea.

Theorem 4.7 (Clustering optimisation criterion [18]). *The solution of the clustering optimisation criterion*

$$f = \arg\min_{f} \sum_{i,j:f_i=f_j} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2$$

can be found in the form

$$f(\mathbf{x}_i) = \arg\min_{1 \le k \le N} \|\phi(\mathbf{x}_i) - \mu_k\|,$$

where μ_j is the centroid of the points assigned to cluster j.

Proof. See page 269 [18]. □

Therefore it is clear now how to assign data points to a cluster. Any clustering algorithm must attempt to minimize the cost function. We can efficiently achieve this by assigning points to a cluster and fixing several centers N, Then, we try to find the center that minimizes the cost function. Thus, we have the following optimization strategy.

Input	Data $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$, integer N
process	$\mu = \arg\min_\mu \sum_{i=1}^{N} \min_{1\leq k\leq N} \left\|\phi(\mathbf{x}_i) - \mu_k\right\|^2$
Output	$f(\cdot) = \arg\min_{1\leq k\leq N} \left\|\phi(\cdot) - \mu_k\right\|$

Table 10: Cluster optimisation strategy algorithm.

Computation 4.8 (Clustering optimisation strategy [18]). The clustering optimisation strategy is given by

An appropriate pattern function to assess the stability of this strategy would be

$$\mathbb{E}_{\mathcal{D}} \min_{1\leq k\leq N} \left\|\phi(\mathbf{x}) - \mu_k\right\|^2 ;$$

the smaller the bound obtained the better the quality of the clustering achieved.

Although we have found an appropriate criterion to cluster the data, the related optimization problem is not convex. Moreover, finding a solution with a value better than some threshold turns out to be NP-complete. Therefore we have to develop some heuristics to seek a local optimum of the cost function. Another way to tackle this problem would be by relaxing the cost function to give an approximation that can be globally optimized. The first method will lead to the k-means algorithm, while the relaxation method gives the spectral clustering algorithm. The approaches can be applied in kernel-defined feature space in both cases.

4.2.1 Greedy solutions: k-means

We have already deduced that the way to solve the clustering problem is by identifying the centers of mass of the members of each cluster. The *k-means algorithm* tries to do this. This algorithm keeps a set of the centroids of the clusters $C_1, \ldots, C_N$ that are initialized randomly and then seeks to minimize the expression

$$\sum_{i=1}^{l} \left\|\phi(\mathbf{x}_i) - C_{f(\mathbf{x}_i)}\right\|^2 ,$$

by adapting both f and the centers. This method will converge to a solution in which C_k is the center of mass of the points assigned to cluster k. The algorithm alternates between updating f to adjust the assignment of points to clusters and updating the C_k giving the position of the centers in a two-stages iterative procedure. In the first stage of the algorithm, the points are moved to the cluster whose cluster center is closest. Then in the second stage, the center of each cluster is positioned at the center of mass of the points assigned to that cluster. In each of these states, the criteria we seek to minimize reduces its value. Since the number of possible clustering is finite, it follows that after a finite number of iterations, the algorithm will converge to a stable clustering assignment provided ties are broken deterministically. In order to implement the algorithm in the dual form, we must represent the clusters by an indicator matrix $\mathbf{A}$ of dimension $l \times N$ containing a 1 to indicate the containment of an example in a cluster.

$$\mathbf{A}_{ik} = \begin{cases} 1, & \text{if } \mathbf{x}_i \text{ is in cluster k;} \\ 0, & \text{otherwise.} \end{cases}$$

And we say that the clustering is given by the matrix $\mathbf{A}$. It is worth noticing the each row of $\mathbf{A}$ contains exactly one 1, while the column sums give the number of points assigned to the different clusters. This kind of matrices are known as *cluster matrices*. The coordinates of the centroid C_k are obtained as the N columns of the matrix

$$\mathbf{X}'\mathbf{A}\mathbf{D}$$

where $\mathbf{X}$ contains the training example feature vectors as rows and $\mathbf{D}$ is a diagonal $N \times N$ matrix with diagonal entries the inverse of the column sums of $\mathbf{A}$, indicating the number of points ascribed to that cluster. The distances of a new test vector $\phi(\mathbf{x})$ from the centroids is now given by

$$\begin{aligned} \|\phi(\mathbf{x}) - C_k\|^2 &= \|\phi(\mathbf{x})\|^2 - 2\langle\phi(\mathbf{x}, C_k)\rangle + \|C_k\|^2 \\ &= \kappa(\mathbf{x}, \mathbf{x}) - 2(\mathbf{k}'\mathbf{A}\mathbf{D})_k + (\mathbf{D}\mathbf{A}'\mathbf{X}\mathbf{X}'\mathbf{A}\mathbf{D})_{kk}, \end{aligned}$$

where $\mathbf{k}$ is the vector of inner products between $\phi(\mathbf{x})$ and the training examples. Hence, the cluster to which $\phi(\mathbf{x})$ should be assigned is given by

$$\operatorname*{arg\,min}_{1\leq k\leq N} \|\phi(\mathbf{x}) - C_k\|^2 = \operatorname*{arg\,min}_{1\leq k\leq N} (\mathbf{D}\mathbf{A}'\mathbf{K}\mathbf{A}\mathbf{D})_{kk} - 2(\mathbf{k}'\mathbf{A}\mathbf{D})_k,$$

where $\mathbf{K}$ is the kernel matrix of the training set. This provides the rule for classifying new data. The updated rule consists in reassigning the entries in the matrix $\mathbf{A}$ according the the same rule in order to redefine the clusters. It is important to notice that this algorithm is prone to local minima since the optimisation is not convex. In order to solve this we will present a relaxed algorithms.

4.2.2 Relaxed solution: spectral methods

The goal of this subsection is to make a convex relaxation of the problem in order to obtain a closed-form approximation.

Clustering into two classes. We consider only the classification in two clusters. In this approximation the cluster assignment will be given by a vector $\mathbf{y} \in \{-1, +1\}^l$, that associates to each point a $\{-1, +1\}$ label. In this approximation the clustering quality criterion will be minimised by maximising

$$\sum_{y_i \neq y_j} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 .$$

If we assume that the data is normalised and the sizes of the clusters are equal this correspond to minimising the so-called *cut cost*

$$2 \sum_{y_i \neq y_j} \kappa \mathbf{x}_i, \mathbf{x}_j = \sum_{i,j=1}^{N} \kappa \mathbf{x}_i, \mathbf{x}_j - \sum_{i,j=1}^{N} y_i y_j \kappa \mathbf{x}_i, \mathbf{x}_j,$$

since this criteria measures the kernel "weight" between vertices in different clusters. Hence, we must solve

$$\begin{array}{ll} \max & \mathbf{y}'\mathbf{K}\mathbf{y}, \\ \text{subject to} & \mathbf{y} \in \{-1, +1\}^N. \end{array}$$

And we can relax this optimisation by removing the restriction that $\mathbf{y}$ be a binary vector while controlling its norm. This is achieved by maximising the Raleigh quotient

$$\max \frac{\mathbf{y}'\mathbf{K}\mathbf{y}}{\mathbf{y}'\mathbf{y}}$$

This is solved by the eigenvector of the matrix $\mathbf{K}$ corresponding to the largest eigenvalue with the value of the quotient equal to the eigenvalue $\lambda_{\mathbf{1}}$. Hence, we obtain a lower bound on the cut cost of

$$0.5 \left(\sum_{i,j=1}^{N} \kappa(\mathbf{x}_i, \mathbf{x}_j) - \lambda_1 \right).$$

giving a corresponding lower bound on the value of the sum-squared criterion.

4.3 Data visualisation

Visualization refers to techniques that can present a dataset

$$S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$$

in such a way, it can reveal some underlying structure of the data that can be easily understood or appreciated by a user. In this section, we aim to provide a two- or three-dimensional 'mapping' of the data to be later displayed as a graph. This set of methods is significant in kernel methods as we typically embed the data into a high-dimensional vector space. Although, at first glance, this may seem superficial, visually displaying the data in the chosen feature space helps us get a 'feel' for the structure of the data, hence suggesting why certain points are outliers or what type of relations can be found. PCA is one method we can already apply to visualize data better; this algorithm will form the core of the classical multidimensional scaling algorithm. During this section, we will assume that a proper kernel has already been adapted to best capture the view of the data we are concerned about. Therefore we will like to develop algorithms that can find low-dimensional representations of high-dimensional data.

4.3.1 Multidimensional scaling

Multidimensional scaling (MDS) comprises a series of techniques directly aimed at finding optimal low-dimensional data embedding primarily for visualization purposes. To do MDS, we need to start with a matrix of distances or similarities rather than a Gram matrix of inner products or even a Euclidean embedding. For this reason, the first step of MDS is to convert the matrix of similarities into a matrix of inner products. For metric MDS, it is assumed that the distances correspond to embeddings in a Euclidean space, while for non-metric MDS, these similarities can be measured in any way. The next step for the classical algorithm is to obtain the first two or three eigenvectors of the eigendecomposition of the resulting Gram matrix to define two or three-dimensional projections of the points for visualization. It is crucial to notice that if we use a kernel-defined feature space, the first two stages are no longer required, and MDS reduces to computing the first two or three kernel PCA projections.

Algorithm 4.9 (MDS for kernel-embedded data [18])**.** The MDS algorithm for data in a kernel-defined feature space is as follows:

Input	Data $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, dimension $k = 2, 3$.
process	$\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j), i, j = 1, \ldots, l$ $\mathbf{K} - \frac{1}{N}\mathbf{jj}'\mathbf{K} - \frac{1}{N}\mathbf{Kjj}' + \frac{1}{N^2}(\mathbf{j}'\mathbf{Kj})\mathbf{jj}'$, $[V, \Lambda] = \mathrm{eig}(\mathbf{K})$ $\alpha^j = \frac{1}{\sqrt{\lambda_j}}\mathbf{v}_j,\ j = 1, \ldots, k.$ $\tilde{\mathbf{x}}_i = \left(\sum_{i=1}^{N} \alpha_i^j \kappa(\mathbf{x}_i, \mathbf{x})\right)_{j=1}^{k}.$
Output	Transformed data $\hat{S} = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_N\}$.

Table 11: MDS for kernel-embedded data algorithm [18].

Visualisation quality. We can chose another criteria to assess the quality of the representation of the data. This will give rise to another method that it is strongly related to MDS. The problem of visualisation can be stated as follow. Given a set of points

$$S = \{\phi(\mathbf{x}_i), \ldots, \phi(\mathbf{x}_N)\}$$

in a kernel-defined feature space F with

$$\phi : X \to F,$$

find a projection τ from X into $\mathbb{R}^k$, for small k such that

$$\|\tau(\mathbf{x}_i) - \tau(\mathbf{x}_j)\| \approx \|\phi(\mathbf{x}_i) - \phi(\mathbf{x_j})\|\,,\ \text{for } i, j = 1, \ldots, N.$$

From now on, we will denote as τ_s the projection onto the sth component of τ. As we have already deduced, the embedding determined by kernel PCA minimizes the sum-squared residuals

$$\sum_{i=1}^{N} \|\tau(\mathbf{x}_i) - \phi(\mathbf{x}_i)\|^2 ,$$

where we make τ an embedding into a k-dimensional subspace of the feature space F. The next method aims to directly control the relationship between the original and projection distances by solving the following computation.

Computation 4.10 (Visualisation quality [18])**.** The quality of a visualisation can be optimised as follows

$$\begin{aligned} \min_{\tau} \qquad & E(\tau) = \sum_{i,j=1}^{N} \kappa(\mathbf{x}_i, \mathbf{x}_j) \|\tau(\mathbf{x}_i), \tau(\mathbf{x}_j)\|^2 \\ \text{subject to} \qquad & \|\tau_s\| = 1, \quad \tau_s \perp \mathbf{j}, \quad s = 1, \dots, k, \\ & \tau_s \perp \tau_t, \quad s, t = 1, \dots, k. \end{aligned}$$

It can be shown that solving this computation corresponds to minimizing

$$E(\tau) = 2lk - \sum_{i,j=1}^{l} 0.5 \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 \|\tau(\mathbf{x}_i) - \tau(\mathbf{x}_j)\|^2 .$$

Therefore, it corresponds to optimizing the correlation between the original and projected squared distances. The minimization problem aims to put large distances between points with small inner products and small distances between points having large inner products. The constraints ensure equal scaling in all dimensions centered around the origin. The different dimensions must be mutually orthogonal to ensure no structure is unnecessarily reproduced. Now we introduce the concept of the Laplacian matrix.

Definition 4.11 (Laplacian matrix [18])**.** The *Laplacian matrix* $\mathbf{L}(\mathbf{K})$ of a kernel matrix $\mathbf{K}$ is defined by

$$\mathbf{L}(\mathbf{K}) = \mathbf{D} = \mathbf{K}$$

where $\mathbf{D}$ is the diagonal matrix with entries

$$\mathbf{D}_{ii} = \sum_{j=1}^{N} \mathbf{K}_{ij}.$$

It is important to notice that if the kernel matrix has positive entries then the Laplacina matrix $\mathbf{L}(\mathbf{K})$ is positive semi-definite. The following theorem characterizes the above optimization solution using the Laplacian matrix's eigenvectors. This matrix can also be used in clustering as it frequently possesses more balanced properties than the kernel matrix.

Theorem 4.12. *Let*

$$S = \{\mathbf{x}_1, \dots, \mathbf{x}_l\}$$

be a set of points with kernel matrix $\mathbf{K}$*. The visualisation problem is solved by computing the eigenvectors* $\mathbf{v}^1, \mathbf{v}^2, \cdots, \mathbf{v}^l$ *with corresponding eigenvalues* $0 = \lambda_1, \lambda_2 \leq \cdots \leq \lambda_l$ *of the Laplacian matrix* $\mathbf{L}(\mathbf{K})$*. An optimal embedding* τ *is given by* $\tau_i = \mathbf{v}^{i+1}, i = 1, \ldots, k$ *and the minimal value of* $E(\tau)$ *is*

$$2 \sum_{l=2}^{k+1} \lambda_l. \tag{4.7}$$

If $\lambda_{k+1} < \lambda_{k+2}$ *then the optimal embedding is unique up to orthonormal transformations in* $\mathbb{R}^k$*.*

This theorem provides a total characterization of the minimization problem.

5 Conclusions

In this thesis, we have characterized the notion of a kernel as it is used in contemporary machine learning. Moreover, we have cemented some of the essential concepts in statistical learning theory. In Statistical Learning Theory, learning is finding a function that fits some data that comes from an unknown probability distribution. The agnostic setting of SLT is the distinction factor with respect to more classical learning theory. The discrepancy between the predicted data from the learned function and the actual data is known as the *expected risk.* This discrepancy is measured by a *loss function.* The choice of loss function results in different algorithms. As the probability distribution is unknown, we have to work with the *empirical risk.* This minimization criterion is known as the *empirical risk minimization* principle. However, the empirical risk is not the only measure of how well a function has been learned. We can find a function that perfectly fits the known data but does not fit the underlying distribution of the data. In this case, we say that the function *overfit* the data. To solve this, the space in which the solution is searched is restricted. In kernel theory, we choose this space as a *Reproducing Kernel Hilbert Space.* Then, a measure of the capability of a function to fit the data is derived. This *capacity* of the space will be added in the bound of the expected risk and is known as the *structural risk.* The measure of the capacity of a functional space can be done in different ways. In this work, we have introduced the VC dimension and the Rademacher complexity. Thus, an analysis of the generalization capacities of any machine learning algorithm will depend on the empirical risk and the structural risk. When searching for a solution to the learning problem, we will look in nested spaces of functions with increasing capacities. This method is computationally expensive; for this reason, a regularization term is added to the empirical risk minimization problem. Thus, in the empirical risk minimization setting, two things must be chosen to define the problem: the loss function and the hypotheses space.

The hypotheses space in the theory of kernels in an RKHS. An RKHS is a Hilbert space whose evaluation function is continuous. Given an RKHS, there exists a unique kernel that characterizes the kernel. Moreover, kernels are characterized by Mercer's theorem. In this second interpretation, kernels are positive definite functions that accept a base of eigenfunctions. The Representer Theorem states that the solution to the empirical minimization problem with a differentiable loss function can be expressed as a linear combination of the kernel evaluated in each data point. In this way, kernels represent a natural way to represent the hypotheses space in the empirical risk minimization problem. Furthermore, we showed how the most natural way to work with kernels is as being an inner product in a feature space. Thus, the kernel method is interpreted as first sending the data points to a higher dimensional space and then, through the kernel, calculating the inner product in that space. This way, if a linear algorithm is applied to the points in the feature space, it corresponds to a nonlinear algorithm in the original space.

Once we characterized the learning problem, we proceeded to develop the kernelized version of different algorithms. In Supervised learning, it is sufficient to choose the proper loss function to implement different algorithms. The algorithms are normally first deduced in primal form, and the form of the algorithm that only depends on the kernel is called the dual algorithm. For different choices of loss functions, we obtained different algorithms. All the algorithms were proven to generalize by providing a bound on the expected risk. In Unsupervised learning, we applied the vision of the kernels being inner products in a feature space. In this way, we implemented algorithms that only depended on the inner product of the data points.

After this discussion, we have shown how kernel methods allow for more powerful machine learning algorithms. For this reason, in the following years after its implementation, a large amount of literature was produced to produce more specialized kernels. However, soon the big drawbacks of the theory of kernels were discovered. In order to implement any kernel algorithm, at least $O(N^2)$ operations are necessary for obtaining the Gram matrix. This is incredibly expensive for large applications. At the same time, neural networks started to show great generalization capabilities correlated with the number of parameters and hidden layers in the NN. Great results were produced in imaging processing and natural language processing. In this way *deeps* neural networks substituted *shallow* kernel methods. Although the machine learning community is now more centered on developing DNN, this does not mean kernel methods are still not used. For simpler, less demanding applications, kernel methods are still an excellent choice. Moreover, the theory begins that these methods may still be proven useful. In the last decade, the field of quantum computing has developed to the point that quantum computers are closer to being a reality. For this reason, there was a surge in the development of quantum machine learning algorithms. At first, due to their empirical success, an effort was made to develop a quantum version of neural networks, but it was found that there was no method to do this naturally. Instead, kernel methods proposed an extremely natural framework to translate classical machine learning algorithms to a quantum setting [26]. These results may be a signal that the underlying theory of kernels may still be relevant in the future.

References

[1] Mercer, J. (1909) *Functions of positive and negative type and their connection with the theory of integral equations.* Philosophical Transactions of the Royal Society of London, Series A 209, pp. 415-446.

[2] Moore, E.H. (1916) *On properly positive Hermitian matrices.* Bull. Amer. Math. Soc. 23(59).

[3] Aronszajn, N. (1950) *Theory of Reproducing Kernels.* Transactions of the American Mathematical Society.

[4] Kuhn, H. W., Tucker, A. W. (1951) *Nonlinear programming.* Proceedings of 2nd Berkeley Symposium. Berkeley: University of California Press. pp. 481-491

[5] Rosenblatt, F. (1960). *Perceptron simulation experiments.* Proceedings of the IRE, 48(3), 301-309.

[6] Novikoff, A. B. (1963) *On convergence proofs for perceptrons.* Stanford Research Inst. Menlo Park, CA.

[7] Steward, J. (1976) *Positive definite functions and generalizations, an historical survey.* Rocky Mountain J. Math., 6:409-434.

[8] Ivanov, V.V. (1976) *The Theory of Approximate Methods and Their Application to the Numerical Solution of Singular Integral Equations.* Nordhoff International, Leyden, 1976.

[9] Tikhonov, A.N., Arsenin, V. Y. (1977) *Solutions of Ill-posed Problems.* W. H. Winston, Washington, D.C.

[10] Jolliffe, I. T. (1986) *Principal Component Analysis.* Springer Series in Statistics. pp. 487.

[11] Rudin, W. (1991) *Functional Analysis.* International Series in Pure and Applied Mathematics. Vol. 8 (Second ed.). New York, NY.

[12] Cortes,C., Vapnik, V. (1995) *Support-vector networks.* Mach Learn 20, 273-297

[13] Smola, A., Schölkopf. (1998) *On a kernel-based method for pattern recognition, regression, approximation and operator inversion.* Algorithmica, 22:211 - 231.

[14] Vapnik, V. (2000) *The Nature of Statistical Learning Theory.* Springer New York, NY.

[15] Shawe-Taylor, J., Cristianini N. (2000) *An introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge University Press.

[16] Evgeniou, T. (2000) *Learning with kernel machine architectures.* PhD thesis, Massachusetts Institute of Technology.

[17] Schölkopf, B., Smola, A.J. (2002) *Learning with Kernels.* MIT Press, Cambridge, Ma.

[18] Shawe-Taylor, J., Cristianini N. (2004) *Kernel Methods for Pattern Analysis.* Cambridge University Press.

[19] Hamers, B. (2004) *Kernel Models for Large Scale Applications.* PhD thesis.

[20] Nocedal, J., Wright, S. J. (2006) *Numerical Optimization.* New York. Springer.

[21] Luxburg, U., Schölkopf, B. (2008) *Statistical Learning Theory: Models, Concepts, and Results.* https://arxiv.org/abs/0810.4752

[22] Hofmann, T., Schölkopf, B., and Smola, AJ. (2008) *Kernel Methods in Machine Learning.* Annals of Statistics, 36:1171-1220.

[23] Schmidhuber, J. (2015) *Deep learning in neural networks: An overview.* Neural Networks. 61: 85-117.

[24] Hastie, T., Tibshirani, R., Friedman, J. (2019) *The Elements of Statistical Learning.* Second Edition. Springer New York, NY.

[25] Schuld, M. (2021) *Supervised quantum machine learning models are kernel methods.* https://arxiv.org/abs/2101.11020

[26] Schuld, M., Petruccione, F. (2021) *Machine Learning with Quantum Computers.* Springer Cham.