Kernel-based learning for classification and dimensionality reduction

Aalaila Yahya

September 2021

Contents

List of Figures

Acknowledgement

I wish to express specifically, my appreciation and fundamental respect to my supervisor Professor *Abdallah Mkhadri* for his invaluable assistance and help that he provided during my study.

I wish to express my deepest gratitude to my supervisor, Professor *Diego Peluffo*, your guidance, advice and encouragement had a great and positive impact into completing this work, especially as I got introduced to the Machine Learning landscape. To the Smart Data Analysis Systems Group (*SDAS Group* for short), my gratitude for financing an advanced machine to conduct the computational part of this work is beyond comprehension.

I would like to pay my special regards to the whole academic crew of the Mathematics department at FSSM, your efforts during these last academic years are dully noted.

I also wish to recognize Professors *Abdelaziz Nasroallah* and *lalla Aicha Allamy* for accepting to evaluate and judge my humble work.

At last but not least, I will always be in depth to my dear family and friends, whom support and help to do the right thing even when the road got tough. Thank you for keeping me going.

Introduction

Machine Learning (ML), to use *Arthur Samuel*'s definition, is the field of study that gives computers the ability to learn without being explicitly programmed. Granted, it is rather a broad definition to machine learning landscape, but any statistical, data-mining and computing technique that falls under that definition is considered a feature of Machine Learning. The construction of machines capable of learning from experience has for a long time been the object of both philosophical and technical debate. The technical aspect of the debate has received a huge rush of interest in the last three decades combined with advances made in electronic computers. Research thus far have demonstrated that machines can display a significant level of learning ability, though the boundaries of this ability are far from being clearly defined.

Developing the theoretical tools for machine learning reside on mimicking human ability to recognize patterns of learning. For example, it is beyond trivial for a human mind to distinguish handwritten numbers of all sorts of variations This ability is speculated to be developed for children as soon as 12 months of age. A harder task would be recognizing the meaning of new words by context, which children develop starting from seven to eight years old[4]. One way of thinking about it is, children take the time to make mistakes and learn from them, usually with the *supervision* of an "expert" (often the parent), which guide them through the answers. Children during time of learning collect, subconsciously, examples with their right answers provided by the expert (hand written digits in our example). This leads to an interesting and broad question, can we "train" a computer based on mathematical model to recognize handwritten digits? This paradigm is called *Supervised Learning* in ML context.

In this work, we will focus on a sub-field of machine learning theory called *Kernel based learning* for supervised type and dimensionality reduction techniques, starting from the well established field of Support vector Machines (SVMs for short) and the derived formulations for Least-squares SVMs (LSSVM for

short). The usage of kernels to extend the reach of theoretical formulations has proven its significance in the discipline, especially since *Vapnik* and his associates proposed SVM formulations for classification and function approximation in early 90's[24]. it has been exploited and further simplified by *Suykens* and his collaborations with many other academics to derive the LSSVM formulations[23], which proven its worth in the interpretable and exploratory data analysis framework. The contribution of SVMs for *Classification*, *Regression*, *function estimation* and *Clustering* type problems has been fully exploited in the last 15 years[22].

This work is arranged as follows. In part I, we first discuss briefly in chapter 1 both classification and dimensionality reduction paradigms for machine learning, providing a somewhat introductory level of mathematical background of learning. Second, in chapter 2 we introduce the concept of support vector machine formulations for classification, especially *Hard-margin SVM*, *L1 SVM* and *L2 SVM*. Third, in chapter 3 we dive into the need for kernel type learning for non-linearly learnable data types and reformulate SVMs with kernels. At last, in chapter 4 we introduce the LSSVM derivation of SVM formulations for classification and dimensionality reduction. Part II, chapters 5 and 6 is devoted to computational experiment of the aforementioned techniques for classification using both artificially controlled and real-life data.

Part I

Theoretical formulation

Chapter 1

Classification and DR

Machine learning approaches branch out into multiple learning paradigms (see Figure 1.1) most common are, Supervised, Unsupervised, semi-supervised and Reinforcement learning[1].

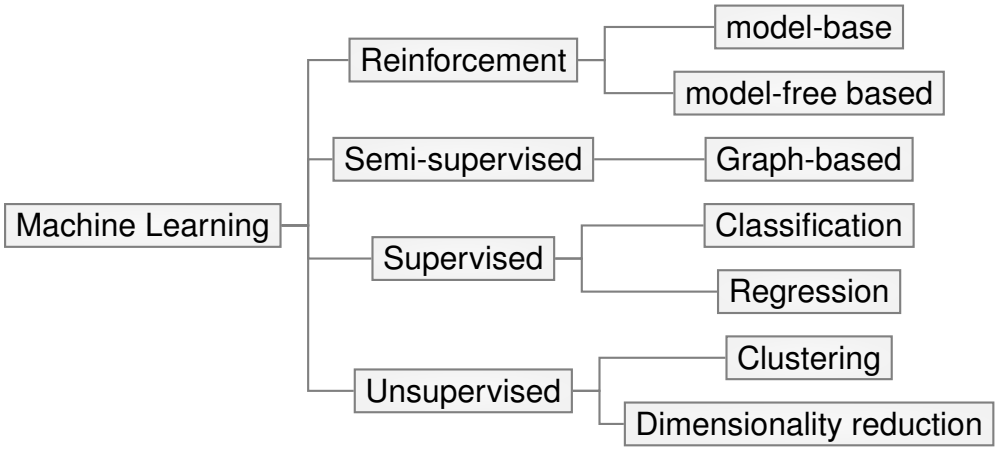

Figure 1.1: General taxonomy to machine learning

In this chapter, we will discuss three well-known ML paradigms, namely, *Classification* and *Dimensionality Reduction* (DR for short). In each section we begin with a general view of the method answering typical questions, such as, what is required from the learner? what is the input and output of approaches in each paradigm. However, mathematically speaking, only classification and DR have a solid theoretical basis in contrast with *clustering*, its nature, intuition and goals are most often that not subjective which leads to different mathematical

background and axioms that might contradict each other.

1.1 Classification

Working on real life problems, one encounters various Machine Learning problems, to which suitable techniques have been introduced and indeed developed. In particular, classification problems rise in multiple domains such as, medical diagnosis, handwriting recognition and image processing. Classification paradigm involves a set of data along with their labels, the aim generally is to segregate the data into categories. As a respond to such reoccurring framework, numerous techniques and approaches have been developed, namely, Logistic regression, Naive Bayes classifier, Perceptron, Quadratic classifiers, Decision trees, Support Vector machines and K-nearest neighbors.

Generally speaking, a classification learner has access to a data set $\mathcal{S}$, often called training data, and produce an approximation function h that assign new data instances to their labels. A classification model goes like this,

Input

- Domain set $\mathcal{X}$ which contains elements to label, it is called set of *attributes* or *classifiers*, which usually is, or a subset of $\mathbb{R}^d$.
- Labels domain $\mathcal{Y}$ which contains the attributes' *labels*, in binary framework it is usually equal to $\{-1, 1\}$, in contrast with multi-class problems where it is a finite set of $\mathbb{N}$.
- A dataset $\mathcal{S}$ containing finite pairs in $\mathcal{X} \times \mathcal{Y}$ of the following form

$$\mathcal{S} = \{(\mathbf{x}_1, y_1), \dots (\mathbf{x}_n, y_n)\},$$

  $\mathcal{S}$ is called set of *training data* or set of *training examples*.

output

- Function $h : \mathcal{X} \longrightarrow \mathcal{Y}$ called *hypothesis* or *classifier*.
- Set of function $\mathcal{H}$, containing function from which the learner chooses h. $\mathcal{H}$ is usually called a *hypothesis class*.

The learner, given $\mathcal{S}$, tracks any sort of regularities or patterns that predict the labels and mimic them in a stable way through h. The mathematical underlying assumption however, is that these pair of instances are drawn from an unknown distribution $\mathcal{D}$ over the set $\mathcal{X}$ and the labels are provided by a "God like" function f, where any instance $\mathbf{x}$ is perfectly labeled by $f(\mathbf{x})$, the learner however, has no access to both of these entities. The learner is trying to select a hypothesis h that belongs to a prior hypothesis class $\mathcal{H}$ following a "measure of success" through which the success is assessed. Formally, h is assessed based on its performance on labeling the available instances correctly, that is given an instance $\mathbf{x}_i$, how far from the truth $h(\mathbf{x}_i)$ is? For that purpose a new notion of *true risk*, denoted $\boldsymbol{L}_{\mathcal{D}}(h)$, is introduced as follows

$$\boldsymbol{L}_{\boldsymbol{\mathcal{D}}}(h) := \mathbb{P}_{\mathbf{x}\sim\mathcal{D}}\left[h(\mathbf{x}) \neq f(\mathbf{x})\right] = \mathcal{D}\left[\{\mathbf{x} \in \mathcal{X},\ h(\mathbf{x}) \neq f(x)\}\right]. \tag{1.1}$$

Since distribution $\mathcal{D}$ and the labeling function f are unknown, this makes the expression in (1.1) impossible to evaluate (unless one makes assumptions about the form of $\mathcal{D}$ and follow through with inferential techniques). A descent approximation of the *true risk* is the *empirical risk* $\boldsymbol{L}_{\boldsymbol{\mathcal{S}}}(h)$ defined as follows,

$$\boldsymbol{L}_{\boldsymbol{\mathcal{S}}}(h) = \frac{1}{n}\sum_{k=1}^{n} \mathbf{1}_{\{h(\mathbf{x}_k)\neq y_k\}}. \tag{1.2}$$

This quantity approximates the error made by the hypothesis h, which if $\mathcal{S}$ is well represented[1] is a good estimator. Since the input training data $\mathcal{S}$ is the only available representation of information for the learner, a good approach is to minimize the empirical risk on $\mathcal{S}$, that is search for a solution belonging to the class hypothesis $\mathcal{H}$ that works well on the available data. This paradigm is refereed to as *Empirical Risk Minimization* (ERM or $\text{ERM}_{\mathcal{H}}$ for short). The objective of this section is to ensure that such a paradigm is a successful way of learning, presenting a sort of mathematical insurance for ERM learnability over a set of examples $\mathcal{S}$.

The hypothesis class $\mathcal{H}$ needs to be defined prior to the learning process, choosing such a class can be seen as a bias aspect of learning. Also, it can be seen as implementing *prior knowledge*[2] into the learner. This gives rise to a valid

[1] $\mathcal{S}$ is thought of as a snapshot of reality, which means it can be misleading or under representing the truth.

[2] Usually derived from analyzing the data's intricacies and complications through data visualisation, feature extraction or dimensionality reduction techniques.

question, why the need to define a prior hypothesis class from which the learner looks for a classifier h that minimizes the empirical risk on $\mathcal{S}$. Every learning task depend on the underlining distribution $\mathcal{D}$, it has been shown that no learning algorithm can learn any learning task independently from $\mathcal{D}$, this is elaborated in a well known theorem called *no free lunch theorem*[19]. It is necessary for any learner to have access to a form of prior knowledge. One approach is to assume that $\mathcal{D}$ is from a family of parametric distribution, in which one uses the tools of inferential statistics to incorporate this knowledge to the learner[2]. Another approach is referred to as PAC learning (see definition 1.1.1), which assumes existence of h in some predefined hypothesis class $\mathcal{H}$, such that $\boldsymbol{L}_{\mathcal{D}}(h) = 0$. A softer type of prior knowledge on $\mathcal{D}$ is assuming that $\min_{h \in \mathcal{H}} \boldsymbol{L}_{\mathcal{D}}(h)$ is small. In a sense, this weaker assumption on $\mathcal{D}$ is a prerequisite for using the *agnostic PAC* model (see definition 1.1.6), in which we require that the risk of the output hypothesis will not be acceptably larger than $\min_{h \in \mathcal{H}} \boldsymbol{L}_{\mathcal{D}}(h)$.

1.1.1 Agnostic PAC learnability

Now we define a unifying learning background for classification through PAC learning.

Definition 1.1.1 (PAC learnablity) *A hypothesis class $\mathcal{H}$ is Probably approximately correct (PAC) learnable if there exist a function $n_{\mathcal{H}} :]0,1[^2 \rightarrow \mathbb{N}$ and a learning algorithm with the following property:*
For every $\epsilon, \delta \in]0,1[$, distribution $\mathcal{D}$ over $\mathcal{X}$, and a labeling function f, when running the learning algorithm on $n \geq n_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$, the algorithm returns a hypothesis h such that,

$$\mathbb{P}\left[\boldsymbol{L}_{\mathcal{D}}(h) \leq \epsilon\right] = 1 - \delta. \tag{1.3}$$

In other words, given an arbitrary tolerance error ϵ and an approximation δ, the hypothesis generated by the learner should have an error less then ϵ with probability $1 - \delta$.
The function $n_{\mathcal{H}}$ stands for the measurement for the so-called *Sample Complexity* of learning $\mathcal{H}$, that is the number of samples required to reach a PAC solution.

Having defined PAC learnablity paradigm, we now show that a specific classifier by the name of *Bayes classifier*, is the optimal classification algorithm that there is. Consider the following definition of Bayes classifier,

Definition 1.1.2 *Let Y a random variable takes values in $\{1, \ldots, m\}$, that is a multi-classification framework, with m classes. The Bayes classification rule, denoted h_b, is defined as follows:*

$$h_b(\mathbf{x}) = arg \max_{1 \leq k \leq m} \{\mathbb{P}(Y = k | X = \mathbf{x})\}. \tag{1.4}$$

We remind Bayes famous formula

$$p(\mathbf{x}) = \mathbb{P}(Y = k | X = \mathbf{x}) = \frac{f_k(\mathbf{x})\pi_k}{\sum_{i=1}^{m} f_i(\mathbf{x})\pi_i}, \tag{1.5}$$

where, $f_k(\mathbf{x}) = \mathbb{P}(X = \mathbf{x} | Y = k)$ is called *class conditional distribution* and $\pi_k = \mathbb{P}(Y = k)$ is called the *prior probability* of the k-th class. The Bayes classifier is defined as follows:

Definition 1.1.3 *The Bayes classification rule, h_b is defined as follows,*

$$h_b(\mathbf{x}) = arg \max_{1 \leq k \leq m} \{f_k(\mathbf{x})\pi_k\}, \tag{1.6}$$

Example 1.1.1 *In the case of Binary classification where $m = 2$ and $\pi_1 = \pi_0$, the Bayes classification rule becomes*

$$h_b(\mathbf{x}) = \begin{cases} 1 & \text{if} \quad p(\mathbf{x}) > \frac{1}{2} \\ -1 & \text{otherwise} \end{cases} \tag{1.7}$$

Theorem 1.1.1 *The Bayes classifier is optimal. Meaning,*

$$\boldsymbol{L}_{\mathcal{D}}(h_b) \leq \boldsymbol{L}_{\mathcal{D}}(h)$$

For any other classification rule h.

Proof 1.1.1 *For mathematical convenience, we will consider the binary case, which can be easily extended to a multi-class classification problem. Let h be*

any binary classifier and X sampled from $\mathcal{D}$, the True Risk then is defined as follows:

$$\begin{aligned}\boldsymbol{L}_{\mathcal{D}}(h) &= \mathbb{P}(Y \neq h(X)) \\ &= \mathbb{E}_{X=x}[Y \neq h(X)]\end{aligned}$$

$$\begin{aligned}\mathbb{P}(Y \neq h(X) \,|\, X = \mathbf{x}) &= 1 - \mathbb{P}(Y = h(X) \,|\, X = \mathbf{x}) \\ &= 1 - \mathbb{P}(Y = 1 \,|\, X = \mathbf{x})\, \mathbf{1}_{\{h(\mathbf{x})=1\}} - \\ &\quad \mathbb{P}(Y = -1 \,|\, X = \mathbf{x})\, \mathbf{1}_{\{h(\mathbf{x})=-1\}} \\ &= [1 - 2\mathbb{P}(Y = 1 \,|\, X = \mathbf{x})]\, \mathbf{1}_{\{h(\mathbf{x})=1\}} + \mathbb{P}(Y = 1 \,|\, X = \mathbf{x})\end{aligned}$$

Then

$$\mathbb{P}(Y \neq h_b(X) \,|\, X = \mathbf{x}) - \mathbb{P}(Y \neq h(X) \,|\, X = \mathbf{x}) = \alpha \left[\mathbf{1}_{h_b(\mathbf{x})=1} - \mathbf{1}_{\{h(\mathbf{x})=1\}}\right],$$

with $\alpha = [1 - 2\mathbb{P}(Y = 1 \,|\, X = \mathbf{x})]$. After studying the sign of the last product throughout all the possible cases, we find

$$\mathbb{P}(Y \neq h_b(X) \,|\, X = \mathbf{x}) \leq \mathbb{P}(Y \neq h(X) \,|\, X = \mathbf{x}),$$

for any binary classifier h. ■

Unfortunately, since we do not know the distribution $\mathcal{D}$, we cannot utilize this optimal classifier h_b, what the learner does have access to is a training sample. Now, we can present the formal definition of agnostic PAC learnability, which is a natural extension of the definition of PAC learnability to the more realistic, non-realizable, learning setup we have just discussed.

Clearly, we cannot hope that the learning algorithm will find a hypothesis whose error is smaller than the minimal possible error, that of the Bayes classifier. Furthermore, as we showed, once we make no prior assumptions about the data-generating distribution, no algorithm can be guaranteed to find a classifier that is as good as the Bayes optimal one. Instead, we require that the learning algorithm will find a classifier whose error is not much larger than the best possible error of a classifier in some given benchmark hypothesis class (see definition 1.1.6). Of course, the strength of such a requirement depends on the choice of that hypothesis class.

While this definition is intuitively correct, it is not general. The notion "how much $h(\mathbf{x})$ differs from the true label y" is up for interpretation, this notion is known in the literature as "loss measure" which is represented by *Loss function* ℓ. Formally the definition goes as follows,

Definition 1.1.4 (loss function) *Let $\mathcal{H}$ be a hypothesis class and Z a nonempty set. ℓ is a loss function if it ranges form $\mathcal{H} \times Z$ to the set of nonnegative numbers $\mathbb{R}^+$,*

$$\ell : \mathcal{H} \times Z \longrightarrow \mathbb{R}^+ \tag{1.8}$$

$$(h, z) \longrightarrow \ell(h, z) \tag{1.9}$$

There exist various loss functions in machine learning framework, two widely used loss functions *0-1 loss function* and *Square loss function*:

- 0-1 loss function

$$\ell_{0-1}(h, (\mathbf{x}, y)) := \begin{cases} 0 & \text{if } h(\mathbf{x}) = y \\ 1 & \text{if } h(\mathbf{x}) \neq y \end{cases} \tag{1.10}$$

- Square loss function

$$\ell_s\left(h, (\mathbf{x}, y)\right) := \left(h(\mathbf{x}) - y\right)^2 \qquad y \in \mathbb{R}. \tag{1.11}$$

The loss measure we used in (1.1) is the 0-1 loss function, now we extend the *True risk* definition using the general definition of *loss functions*.

Definition 1.1.5 (True risk) *Let h be a hypothesis selected from a hypothesis class $\mathcal{H}$ and $\mathcal{D}$ a distribution over Z, the generalized true error risk is defined as follows*

$$\boldsymbol{L}_{\boldsymbol{\mathcal{D}}}(h) := \mathbb{E}_{z \sim \mathcal{D}}\left[\ell(h, z)\right] \tag{1.12}$$

In this definition, contrary to the formula given in (1.1), it is assumed that the distribution $\mathcal{D}$ is representing the pairs $(\mathbf{x}_i, y_i)$ (instead of representing only $\mathbf{x}_i$ and the labels y_i are provided by a flawless function f). This gives rise to a natural extension of the *Empirical Risk*, denoted $\boldsymbol{L}_{\mathcal{S}}(h)$, is defined as follows,

$$\boldsymbol{L}_{\boldsymbol{\mathcal{S}}}(h) := \frac{1}{n} \sum_{i=1}^{n} \ell(h, (\mathbf{x}_i, y_i)) \tag{1.13}$$

Definition 1.1.6 (Agnostic PAC) *A hypothesis class $\mathcal{H}$ is Agnostic Probably approximately correct (Agnostic PAC) learnable if there exist a function $n_{\mathcal{H}} :]0,1[^2 \to \mathbb{N}$ and a learning algorithm with the following property: For every $\epsilon, \delta \in]0,1[$ and for every distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, when running the learner on $n \geq n_{\mathcal{H}}(\epsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$, the algorithm returns a hypothesis h such that,*

$$\mathbb{P}\left[\boldsymbol{L}_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}\left(h'\right) + \epsilon\right] = 1 - \delta \tag{1.14}$$

In this section, we will provide a sufficient condition on the hypothesis class $\mathcal{H}$, to guarantee that ERM framework is solid, for that we define a notion called *uniformly convergent hypothesis class $\mathcal{H}$.*

Definition 1.1.7 (ϵ-representative sample) *A training set S is called ϵ-representative with respect to a domain Z, hypothesis class $\mathcal{H}$, loss function ℓ, and distribution $\mathcal{D}$ on Z if*

$$\forall h \in \mathcal{H}, \quad |\boldsymbol{L}_{\mathcal{S}}(h) - \boldsymbol{L}_{\mathcal{D}}(h)| \leq \epsilon$$

Definition 1.1.8 (Uniform Convergence) *We say that a hypothesis class $\mathcal{H}$ has the uniform convergence property if there exists a function $n_{\mathcal{H}}^{\mathrm{UC}} :]0,1[^2 \to \mathbb{N}$ such that for every $\epsilon, \delta \in]0,1[$, distribution $\mathcal{D}$, if $\mathcal{S}$ is a sample of $n \geq n_{\mathcal{H}}^{\mathrm{UC}}(\epsilon, \delta)$ examples drawn i.i.d. according to $\mathcal{D}$, then, with probability of at least $1 - \delta, S$ is ϵ-representative.*

Similar to the definition of sample complexity for PAC learning, the function $n_{\mathcal{H}}^{\mathrm{UC}}$ measures the (minimal) sample complexity of obtaining the uniform convergence property, namely, how many examples we need to ensure that with probability of at least $1 - \delta$ the sample would be ϵ-representative. The term uniform here refers to having a fixed sample size that works for all members of $\mathcal{H}$ and over all possible probability distributions over the domain.

Proposition 1.1.1 *If a class $\mathcal{H}$ has the uniform convergence property with a function $n_{\mathcal{H}}^{UC}$ then the class is agnostically PAC learnable with the sample complexity $n_{\mathcal{H}}(\epsilon, \delta) \leq n_{\mathcal{H}}^{UC}(\epsilon/2, \delta)$. Furthermore, in that case, the $\mathrm{ERM}_{\mathcal{H}}$ paradigm is a successful agnostic PAC learner for $\mathcal{H}$.*

Proof 1.1.2 *Let $\epsilon,\ \delta > 0$, if $\mathcal{H}$ has the uniform convergence property, then there exists $n(\epsilon, \delta) \in \mathbb{N}$, such that for every $h \in \mathcal{H}$ we have the following*

$$|\boldsymbol{L}_{\mathcal{S}}(h) - \boldsymbol{L}_{\mathcal{D}}(h)| \leq \epsilon. \tag{1.15}$$

Let h' be a hypothesis provided by an ERM rule, then for every $h \in \mathcal{H}$ we have the following with $1 - \delta$ probability,

$$\begin{aligned} \boldsymbol{L}_{\mathcal{D}}(h') &\leq \boldsymbol{L}_{\mathcal{S}}(h') + \frac{\epsilon}{2} \\ &\leq \boldsymbol{L}_{\mathcal{S}}(h) + \frac{\epsilon}{2} + \frac{\epsilon}{2} \\ &\leq \boldsymbol{L}_{\mathcal{D}}(h) + \epsilon \end{aligned}$$

An important type of hypothesis classes, especially in the context of machine learning, is the case where $\mathcal{H}$ is finite.

Corollary 1.1.1 *Let $\mathcal{H}$ be a finite hypothesis class, then $\mathcal{H}$ is agnostic PAC learnable.*

Proof 1.1.3 *Let ϵ, δ be nonnegative real numbers. Let $n \geq 1$ and $Z_{(n)} = (Z_1, Z_2 \ldots, Z_n)$ be a sequence of i.i.d random variables. We have,*

$$\mathbb{P}_{Z_{(n)}}\left[\exists h \in \mathcal{H},\ |\boldsymbol{L}_{\mathcal{S}}(h) - \boldsymbol{L}_{\mathcal{D}}(h)| \geq \epsilon\right] \leq \sum_{h \in \mathcal{H}} \mathbb{P}_{Z_{(n)}}\left[|\boldsymbol{L}_{\mathcal{S}}(h) - \boldsymbol{L}_{\mathcal{D}}(h)| \geq \epsilon\right]$$

using the theorem of large numbers and Hoeffding's Inequality we have

$$\mathbb{P}_{Z_{(n)}}\left[|\boldsymbol{L}_{\mathcal{S}}(h) - \boldsymbol{L}_{\mathcal{D}}(h)| \geq \epsilon\right] \leq e^{-2n\epsilon^2},$$

which yields

$$\mathbb{P}_{Z_{(n)}}\left[\exists h \in \mathcal{H},\ |\boldsymbol{L}_{\mathcal{S}}(h) - \boldsymbol{L}_{\mathcal{D}}(h)| \geq \epsilon\right] \leq 2Ce^{-2n\epsilon^2} \quad \textit{with, } C = |\mathcal{H}|.$$

Choosing

$$n \geq \frac{1}{\epsilon^2} \log\left(\frac{2C}{\delta}\right),$$

yields the desired result. ■

Corollary 1.1.2 *Let $\mathcal{H}$ be a finite hypothesis class, Z a nonempty domain and ℓ a loss function. Then $\mathcal{H}$ is agnostic PAC learnable. That is any ERM rule is PAC learnable over $\mathcal{H}$ with sample complexity*

$$n_{\mathcal{H}} \geq \frac{1}{\epsilon^2} \log\left(\frac{2C}{\delta}\right) \quad \textit{with, } C = |\mathcal{H}|.$$

The finite hypothesis class is important to discuss because of the nature of Machine Learning applications. In practice, using a computer, we will probably maintain real numbers using floating point representation, say, of 64 bits. It follows that in practice, our hypothesis class is parameterized by the set of scalars that can be represented using a 64 bits floating point number. There are at most 2^{64} such numbers; hence the actual size of our hypothesis class is at most 2^{64}. More generally, if our hypothesis class is parameterized by d numbers, in practice we learn a hypothesis class of size at most 2^{64d}. Applying Corollary 1.1.2 we obtain that the sample complexity of such classes is bounded by $\frac{128d+2\log(2/\delta)}{\epsilon^2}$. This aspect has a draw back of being dependent on the specific representation of real numbers used by our machine, which differs accordingly. For that reason, we will introduce a final characterization of Agnostic PAC learnablity using Vapnik–Chervonenkis dimension (VC-dim for short).

1.1.2 Vapnik–Chervonenkis dimension

Definition 1.1.9 (Shattering) *Let $\mathcal{H}$ be a set of functions from a nonempy set $\mathcal{X}$ to $\{0,1\}$ and C a subset of $\mathcal{X}$. $\mathcal{H}$ is said to shatter C if*

$$|\mathcal{H}_C| = 2^{|C|} \tag{1.16}$$

with

$$\mathcal{H}_C = \{(h(c_1), \ldots, h(c_n)),\ h \in \mathcal{H}\}.$$

Example 1.1.2 *Consider $\mathcal{H} = \left\{h_a = \mathbf{1}_{]-\infty,a]},\ a \in \mathbb{R}\right\}$ and $C = \{c\}$, with $c \in \mathbb{R}$. The class of functions $\mathcal{H}$ indeed shatters C, in fact we have*

$$|\mathcal{H}_C| = 2,$$

because the possible elements of this set are $(1,0)$ and $(0,1)$. Whereas, if $C = \{c_1, c_2\}$, with $c_1, c_2 \in \mathbb{R}$ such that $c_1 < c_2$, then $\mathcal{H}$ does not shatter C, in fact

$$|\mathcal{H}_C| = 3,$$

because the possible elements of this set are $(1,0)$, $(0,0)$ and $(1,1)$, no classifier $h \in \mathcal{H}$ can account for the labeling $(0,1)$, because any threshold that assigns the label 0 to c_1 must assign the label 0 to c_2 as well. Therefore not all functions from C to $\{0,1\}$ are included in $\mathcal{H}_C$.

Definition 1.1.10 *The VC-dimension of a hypothesis class $\mathcal{H}$, denoted $\mathrm{VCdim}(\mathcal{H})$, is the maximal size of a set $C \subset \mathcal{X}$ that can be shattered by $\mathcal{H}$. If $\mathcal{H}$ can shatter sets of arbitrarily large size we say that $\mathcal{H}$ has infinite VC-dimension.*

The following results are of great impact, as far as characterizing agnostic PAC learnability by the VC-dimension.

Theorem 1.1.2 (The Fundamental Theorem of Statistical Learning) *Let $\mathcal{H}$ be a hypothesis class of functions from a domain $\mathcal{X}$ to $\{0,1\}$ and let the loss function be the $0-1$ loss. Then, the following are equivalent:*

- *$\mathcal{H}$ has a finite VC-dimension.*
- *$\mathcal{H}$ has the uniform convergence property.*
- *Any ERM rule is a successful agnostic PAC learner for $\mathcal{H}$.*

Proof 1.1.4 *Proof is provided in [19] Page 49.*

Corollary 1.1.3 *Let $\mathcal{H}$ be a hypothesis class of functions from a domain $\mathcal{X}$ to $\{0,1\}$ and let the loss function be the 0-1 loss. Assume that $\mathrm{VCdim}(\mathcal{H}) = d < \infty$. Then, there are absolute constants C_1, C_2 such that $\mathcal{H}$ is agnostic PAC learnable with sample complexity*

$$C_1 \frac{d + \log(1/\delta)}{\epsilon^2} \leq n_{\mathcal{H}}(\epsilon, \delta) \leq C_2 \frac{d + \log(1/\delta)}{\epsilon^2}$$

1.1.3 Regularized Loss Minimization

Another paradigm to learning is through what is called *Regularized Loss Minimization* or RLM for short. In RLM we minimize the empirical risk and a regularization function. Intuitively, the regularization function stabilize the learning algorithm. An algorithm is considered stable if a slight change of its input does not change its output much. Formally, a regularization function is a mapping $R : \mathbb{R}^d \to \mathbb{R}$, and the regularized loss minimization rule outputs a hypothesis in

$$\mathrm{argmin}_{\mathbf{w} \in \mathbb{R}^d} \left(\boldsymbol{L}_{\mathcal{S}}(\mathbf{w}) + R(\mathbf{w}) \right),$$

where, the hypothesis class considered is halfspace class, which is of great importance for the rest of the chapters. The empirical risk $\boldsymbol{L}_{\mathcal{S}}$ in this context depends on $\mathbf{w}$ instead of h.

There are many possible regularization functions one can use, reflecting some prior belief about the problem. Throughout this section we will focus on one of the most simple regularization functions: $R(\mathbf{w}) = \lambda\|\mathbf{w}\|^2$, where $\lambda > 0$ is a scalar and the norm is the ℓ_2 norm, $\|\mathbf{w}\| = \sqrt{\sum_{i=1}^{d} \mathbf{w}_i^2}$. This yields the learning rule:

$$\text{argmin}_{\mathbf{w}} \left(\boldsymbol{L}_{\mathcal{S}}(\mathbf{w}) + \lambda\|\mathbf{w}\|^2 \right)$$

This type of regularization function is often called Tikhonov regularization. It is one of the famous learning paradigms in machine learning, used in Soft margin Support vector machines with norm L2 in section 2.3 and Least Squares Support vector machines in section 4.2. Further details on this paradigm is provided in[13]. A useful paradigm in machine learning optimization problems is to solve these problems with iterative methods instead of one-step closed form approach. In *Big Data* context, where, as the name suggests, the number of data examples n and the dimension of features d are of higher orders. For that reason, solving these optimization problems in closed form is often expensive or simply unfeasible. Various iterative techniques have been developed and well established to deal with such problems. *Gradient Descent* (GD for short) and *Stochastic Gradient Descent* (SGD for short) are widely used among ML community.

SGD is overall similar to GD except for one aspect, we quickly remind the GD algorithm as follows

Algorithm 1 Gradient Descent

1: **Input**: i_m maximal number of iterations.
μ step size.
2: **Initialize** : $\mathbf{w}^{(0)} = 0$ and $i = 0$.
3: **While** $i < i_m$:
$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mu \nabla f\left(\mathbf{w}^{(i)}\right)$
$i = i + 1$.
4: **Output:** $\mathbf{w}^{(i_m)}$,

where ∇ refers to the gradient operator of differentiable function. In the case where, f is convex but non-differentiable, the update rule is done after taken $\mathbf{v}^{(i)} \in \partial f(\mathbf{w}^{(i)})$, with ∂f is the subgradient of f, then

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mu \mathbf{v}^{(i)} \tag{1.17}$$

The issue in large scale datasets is the update rule in step 3 of algorithm 1, assume for purposes of illustration, the goal is to minimize *Empirical Risk* on an $n \times d$ data set $\mathcal{S}$, where n is large ($> 10^4$ for example). To compute the update rule, one needs to compute the following gradient,

$$\nabla \boldsymbol{L}_{\boldsymbol{\mathcal{S}}}(h) = \sum_{i=0}^{n} \nabla \ell\left(h, (\mathbf{x}_i, y_i)\right).$$

This is a sum of n elements, which is devastating to calculate i_m times over. For this reason, we will introduce *Stochastic Gradient Descent* algorithm, where the update rule is replaced by a randomized procedure.

Stochastic Gradient Descent Let f be a λ-strongly convex function, that is, for all $\mathbf{w}$, $\mathbf{u}$ and a scalar $\alpha \in]0,1[$

$$f(\alpha\mathbf{w} + (1-\alpha)\mathbf{u}) \leq \alpha f(\mathbf{w}) + (1-\alpha) f(\mathbf{u}) - \frac{\lambda}{2}\alpha(1-\alpha)\|\mathbf{w}-\mathbf{u}\|^2 \quad (1.18)$$

The update rule in the stochastic sense is to choose $\mathbf{v}^{(i)}$ randomly such that $\mathbb{E}\left[\mathbf{v}^{(i)}\right] \in \partial f(\mathbf{w}^{(i)})$ and update similarly to (1.17). One last component to discuss is the step μ, which is an important aspect of the algorithm, generally speaking, it is beneficial to update the step in each iteration as to decrease it as the iterations increase, for that reason a widely used update for the step is $\mu_i = \frac{1}{\lambda i}$.

Back to the RLM context, we set

$$f(\mathbf{w}) = \boldsymbol{L}_{\mathcal{S}}(\mathbf{w}) + \lambda\|\mathbf{w}\|^2$$

Then update rule for RLM framework is as follows,

* Choose $\mathbf{v}^{(i)}$ randomly such that $\mathbb{E}\left[\mathbf{v}^{(i)}\right] \in \partial \boldsymbol{L}_{\mathcal{S}}(\mathbf{w}^{(i)})$
* Make the following update rule

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \mu_i\left(\lambda\mathbf{w}^{(i)} + \mathbf{v}^{(i)}\right) \quad (1.19)$$

$$= \frac{1}{\lambda i}\sum_{k=0}^{i} \mathbf{v}^{(k)} \quad (1.20)$$

Algorithm 2 Stochastic Gradient Descent

1: **Input**: i_m maximal number of iterations.
2: **Initialize** : $\mathbf{w}^{(0)} = 0$, μ as a constant, and $i = 1$.
3: **While** $i < i_m$:
Choose $\mathbf{v}^{(i)}$ randomly such that $\mathbb{E}\left[\mathbf{v}^{(i)}\right] \in \partial f(\mathbf{w}^{(i)})$.
Update $\mu_i = \frac{1}{\lambda i}$.
$\mathbf{w}^{(i+1)} = \dfrac{1}{\lambda i} \sum_{k=0}^{i} \mathbf{v}^{(i)}$.
$i = i + 1$.
4: **Output:** $\mathbf{w}^* = \sum_{k=1}^{i_m} \mathbf{w}^{(k)}$.

1.1.4 Baseline classifiers

Throughout the years, various classification approaches have been developed and implemented. In this section we mention a few famous techniques to classification problems.

Linear Discriminant Analysis LDA for short, Also known as Gaussian Bayes classifier, in which one assumes that $f_k(\mathbf{x})$ in (1.6) is a d-dimensional Gaussian distribution with mean $\boldsymbol{\mu}_k$ and a common covariance matrix $\boldsymbol{\Sigma}$ across classes. The Gaussian Bayes classifier assigns an observation $\mathbf{x}$ to the class for which $\mathbb{P}(Y = k \,|\, X = \mathbf{x})$ is the largest, in other words,

$$\max_{1\le k\le m} \mathbb{P}(Y = k \,|\, X = \mathbf{x}) = \max_{1\le k\le m} \left[\mathbf{x}^t \boldsymbol{\Sigma}^{-1} \mu_k - \frac{1}{2}\mu_k^t \boldsymbol{\Sigma}^{-1} \mu_k + \log(\pi_k)\right]. \tag{1.21}$$

The resultant decision rule is linear in $\mathbf{x}$ since the quadratic terms cancel out, due to the common covariance matrix. Note that we still need to replace μ_k,π_k and Σ

by their estimators $\hat{\boldsymbol{\mu}}_k$, $\hat{\pi}_k$ and $\hat{\boldsymbol{\Sigma}}$ respectively.

$$\begin{aligned}\hat{\boldsymbol{\Sigma}} &= \frac{1}{n-m}\sum_{k=1}^{m}\sum_{i=1}^{n}\left(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k\right)\left(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k\right)^{\mathsf{T}} \\ \hat{\pi}_k &= \frac{n_k}{n} \\ \hat{\boldsymbol{\mu}}_k &= \frac{1}{n_k}\sum_{i,Y_i=k}\mathbf{x}_i,\end{aligned}$$

where $n = \sum_{i=1}^{m} n_i$, with m number of classes considered and n_k the size of class k.

Quadratic Discriminant Analysis QDA for short, similarly to Gaussian Bayes where we assume that the class conditional is assumed to be Gaussian distributed, except that the covariance matrix is not common across classes, meaning for every class k, the covariance matrix is denoted $\boldsymbol{\Sigma}_k$. QDA classifier assigns an observation $\mathbf{x}$ to the class for which $\mathbb{P}(Y = k \,|\, X = \mathbf{x})$ is the largest, in other words,

$$\max_{1\le k\le m}\mathbb{P}(Y = k \,|\, X = \mathbf{x}) = \max_{1\le k\le m}\left[-\frac{1}{2}\mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}_k^{-1}\mathbf{x} + \mathbf{x}^{\mathsf{T}}\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k^{T}\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k + \log\pi_k\right]. \tag{1.22}$$

This, unlike LDA, leads to a non-linear decision rule, in fact it is quadratic in $\mathbf{x}$, hence the name QDA.

There are several classical classification techniques that perform well on a variety of data types, like *K-nearest-neighbors*, *Random Forests*, *Decision trees*, *Logistic Regression* and *Neural Networks* [19, 13, 28, 8].

1.2 Dimensionality Reduction

Dimensionality reduction (DR for short) is the process of taking data in a high dimensional space and mapping it into a new space whose dimensionality is much smaller. There are several reasons to reduce the dimensionality of the data. First, high dimensional data impose computational challenges. Moreover, in some situations high dimensionality might lead to poor generalization abilities of the learning algorithm (for example, in Nearest Neighbor classifiers the sample complexity increases exponentially with the dimension). Finally, dimensionality reduction

can be used for interpretability of the data, for finding meaningful structure of the data, and for illustration purposes. In this section we will introduce two popular DR approaches with decades of well founded research behind them, namely *Principal Component Analysis* (PCA for short) and *Fisher Discriminant Analysis* (FDA for short).

1.2.1 Principle Component Analysis (PCA)

There exists various formulation of PCA[11], as well as various interpretation as for what this method can be used. For instance, PCA can be thought of as a clustering method in unsupervised settings or as a dimensionality reduction problem. It is extremely hard to visualize and interpret the data when the number of explanatory variable are high. For that, the premise of PCA is to reduce the dimension of the data for a better interpretation and visualization of the data. Indeed, lowering the data dimension will result by loosing a portion of the information, but the aim is to sacrifice as small of the information as possible while reducing the dimension. Consider a given set of zero mean data $\{\mathbf{x}_i \in \mathbb{R}^d,\ 1 \leq i \leq n\}$ as a cloud of points for which one tries to find projected variables $\mathbf{w}^\mathsf{T}\mathbf{x}$ with maximal variance, which translates to the following quadratic optimization problem.

$$\max_{\mathbf{w}\in\mathbb{R}^d} Var(\mathbf{w}^\mathsf{T}\mathbf{x}) = \max_{\mathbf{w}\in\mathbb{R}^d} \mathbf{w}^\mathsf{T}\mathbf{C}\mathbf{w} \tag{1.23}$$

Where $\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^\mathsf{T}$ is the empirical estimation of the covariance matrix. The goal is to maximize the variance such that $||\mathbf{w}||^2 = 1$. This problem translates directly to solving for eigenvalues problem as follows

$$\mathbf{C}\mathbf{w} = \alpha\mathbf{w},$$

where α is the Lagrangian multiplier
This means trying to maximize the projected data's variance is the same as picking the eigenvector corresponding to the highest eigenvalues of $\mathbf{C}$ because

$$\mathbf{w}^\mathsf{T}\mathbf{C}\mathbf{w} = \alpha\mathbf{w}^\mathsf{T}\mathbf{w} = \alpha.$$

Let $\mathbf{v}$ be the eigenvector associated with the highest eigenvalue of $\mathbf{C}$, $\mathbf{v}$ is called the first *principal component* on which we project the original data. Inductively, if one chooses to reduce the dimension from d to $m << d$, we choose

the first m eigenvalues (sorted decreasingly), forming a projection matrix $\mathbf{P}_m = [\mathbf{w}_1, \dots \mathbf{w}_m]^\mathsf{T}$. That means for all i the projected data set $\{\mathbf{z}_i \in \mathbb{R}^m, \ \ 1 \leq i \leq m\}$ is defined

$$\mathbf{z}_i = \mathbf{P}_k^\mathsf{T} \mathbf{x}_i \qquad 1 \leq i \leq n.$$

However there is an algorithmic complexity to it as well, the computational complexity of PCA is of order d^3 solving for $\mathbf{C}$'s eigenvalue, plus nd^2 for construction the matrix $\mathbf{C}$, which becomes a problem when d is very large. A practical solution to this hurdle is to consider the matrix $\mathbf{B}$ defined as follows

$$\mathbf{B} = \mathbf{X}\mathbf{X}^\mathsf{T} \ \in \mathbb{R}^{n \times n},$$

where $\mathbf{B} = (b_{i,j})$, with $b_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Let $\mathbf{v}$ be an eigenvector of $\mathbf{B}$, that is for some $\lambda \in \mathbb{R}$ we have $\mathbf{B}\mathbf{v} = \lambda \mathbf{v}$. This leads to the following

$$\mathbf{X}^\mathsf{T}\mathbf{B} = \mathbf{X}^\mathsf{T}\mathbf{X}\mathbf{X}^\mathsf{T}\mathbf{v} = \lambda \mathbf{X}^\mathsf{T}\mathbf{v} \Longrightarrow \mathbf{A}\left(\mathbf{X}^\mathsf{T}\mathbf{v}\right) = \lambda\left(\mathbf{X}^\mathsf{T}\mathbf{v}\right).$$

This directly implies, the PCA problem is equivalent to picking the eigenvector $\mathbf{v}$ corresponding to the highest eigenvalue of $\mathbf{B}$, then defining $\mathbf{u}$ to be

$$\mathbf{u} = \frac{\mathbf{X}^\mathsf{T}\mathbf{v}}{||\mathbf{X}^\mathsf{T}\mathbf{v}||}$$

as the first principle component. In this case, the computational complexity becomes of order n^3 solving for $\mathbf{B}$'s eigenvalues, plus dn^2 for constructing the matrix $\mathbf{B}$. In addition, to solve PCA translates to computing the components of the matrix $\mathbf{B}$, which are nothing but the inner products between the vectors $\mathbf{x}_i$. This aspect will lay the ground work for Kernel PCA in latter sections. Consider the following pseudo-algorithm for PCA technique, implementing two cases of order between n and d.

Algorithm 3 Principle Component Analysis

1: **Input**: Matrix of example $\mathbf{X} \in \mathbb{R}^{n \times d}$.
Number of desired components m.

2: **If**: $n < d$
$\mathbf{A} = \mathbf{X}^\mathsf{T}\mathbf{X}$
let $\mathbf{u}_1$, $\mathbf{u}_2$ ... $\mathbf{u}_m$ be the eigenvectors corresponding to the m largest eigenvalues of $\mathbf{A}$.

3: **Else**:
$\mathbf{B} = \mathbf{X}\mathbf{X}^\mathsf{T}$
for $i \in \{1, \ldots, m\}$: Let $\mathbf{u}_i = \dfrac{\mathbf{X}^\mathsf{T}\mathbf{v}_i}{||\mathbf{X}^\mathsf{T}\mathbf{v}_i||}$
where $\mathbf{v}_i$ are the eigenvectors corresponding to the m largest eigenvalues of $\mathbf{B}$.

4: **Output:** $\mathbf{U} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m\}$

Consider a generic data-set called `Iris` retrieved from the UCI benchmark repository[7]. This dataset contains 4 explanatory variables and a labeling variable, with 150 examples. Because the dimension of $\mathbf{x}_i$, visualising the data is impossible, however, using PCA can help reduce the dimension to achieve that. The two first principle components will be chosen (two eigenvectors corresponding to the largest eigenvalues of equation 4.12), then the data will be plotted according to the two principle components carrying along the class information to show case that PCA can perform clustering tasks as well. The first five example of `Iris` dataset is summarized in Table 1.1 and first five values of the first two *principle components* is summarized in Table 1.2.

$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
-0.900681	1.032057	-1.341272	-1.312977
-1.143017	-0.124958	-1.341272	-1.312977
-1.385353	0.337848	-1.398138	-1.312977
-1.506521	0.106445	-1.284407	-1.312977
-1.021849	1.263460	-1.341272	-1.312977

Table 1.1: First five examples of `Iris` dataset.

Although reducing the data dimension, the main information of the attributes, which segregate them, is kept. In Figure 1.2 we can see clearly that PCA while reducing the dimensionality, performed a clustering outcome. This is apparent as the resulting plot is somewhat segregating the classes.

$\mathbf{x}_1$	$\mathbf{x}_2$
-2.264542	0.505704
-2.086426	-0.318477
-2.367950	0.337848
-2.304197	-0.575368
-2.388777	0.674767

Table 1.2: First five examples of projected `Iris` dataset on the first two principle components.

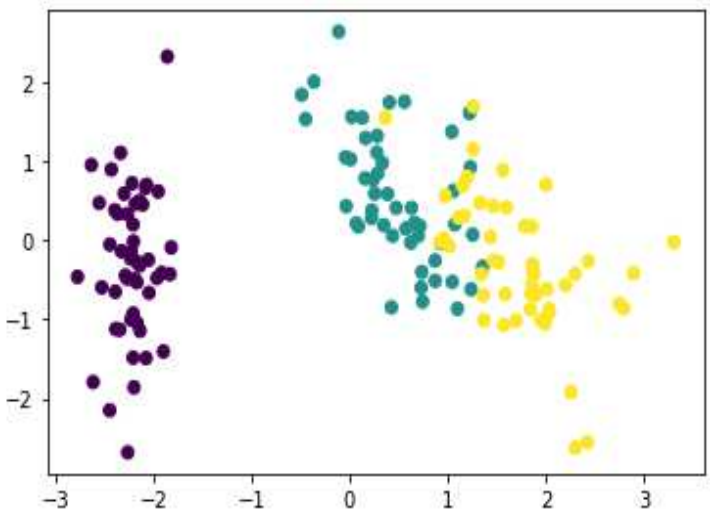

Figure 1.2: scatter plot of `Iris` dataset using only the first two principle components.

1.2.2 Fisher Discriminant analysis (FDA)

Unlike PCA, Fisher Discriminant analysis aims to project the original data into only one-dimensional space. In fact, by using FDA one achieves both DR and classification. Because in binary framework, the premise of this method is maximizing the between-class variance and minimize the within-class variance for the two classes. To elaborate on this mathematically, consider a binary classification set-up, where we have a training data with two classes 1 and -1. Let $\boldsymbol{\mu}^+$, $\boldsymbol{\Sigma}^+$ and $\boldsymbol{\mu}^-$, $\boldsymbol{\Sigma}^-$ the respective means and covariance of the two classes at hand, the idea

of FDA approach is to

$$\begin{cases} \text{Within class :} & \min\limits_{\mathbf{w}} \left\{ \mathbf{w}^{\mathsf{T}} \left(\boldsymbol{\Sigma}^{+} + \boldsymbol{\Sigma}^{-} \right) \mathbf{w} \right\} \\ \text{between class :} & \max\limits_{\mathbf{w}} \left\{ \left|\left| \mathbf{w}^{\mathsf{T}} (\boldsymbol{\mu}^{+} - \boldsymbol{\mu}^{-}) \right|\right|^{2} \right\}. \end{cases}$$

Setting $\boldsymbol{\Sigma}^{+} + \boldsymbol{\Sigma}^{-} = \boldsymbol{\Sigma}_w$ and $\boldsymbol{\Sigma}_b = (\boldsymbol{\mu}^{+} - \boldsymbol{\mu}^{-}) (\boldsymbol{\mu}^{+} - \boldsymbol{\mu}^{-})^{\mathsf{T}}$, this leads to optimize the following simultaneously

$$\min_{\mathbf{w}} \left\{ \mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_w \mathbf{w} \right\} \qquad \text{and} \qquad \max_{\mathbf{w}} \left\{ \mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_b \mathbf{w} \right\} \tag{1.24}$$

One way to incorporate the two expression in one optimization problem is to maximize the so-called *Rayleigh quotient*,

$$\max_{\mathbf{w}} \frac{\mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_b \mathbf{w}}{\mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_w \mathbf{w}}. \tag{1.25}$$

We can put constraints on the denominator according to which we maximize the nominator, that is we impose the constraint $\mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_w \mathbf{w} = 1$ and maximize $\mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_b \mathbf{w}$ accordingly. This leads to

$$\begin{cases} \max\limits_{\mathbf{w}} \mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_b \mathbf{w} \\ \text{s. t.} \quad \mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_w \mathbf{w} = 1, \end{cases} \tag{1.26}$$

to which the corresponding Lagrangian expressed as

$$L(\mathbf{w}, \alpha) = \mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_b \mathbf{w} + \left(1 - \mathbf{w}^{\mathsf{T}} \boldsymbol{\Sigma}_w \mathbf{w}\right).$$

The complimentary conditions are obtained by differentiating with respect to $\mathbf{w}$ as follows

$$* \quad \frac{\partial L}{\partial \mathbf{w}} = \boldsymbol{\Sigma}_b \mathbf{w} - \alpha \boldsymbol{\Sigma}_w \mathbf{w} = 0 \quad \Longrightarrow \quad \boldsymbol{\Sigma}_w^{-1} \boldsymbol{\Sigma}_b \mathbf{w} = \alpha \mathbf{w}.$$

this means, maximizing (1.25) is the same as picking the eigenvector corresponding to the highest eigenvalue of the matrix $\boldsymbol{\Sigma}_w^{-1} \boldsymbol{\Sigma}_b$. We denote such eigenvector $\mathbf{v}$, the projection function f is

$$f(\mathbf{x}) = \mathbf{v}^{\mathsf{T}} \mathbf{x}. \tag{1.27}$$

For example the projected means are

$$f(\hat{\boldsymbol{\mu}}^+) = \mathbf{v}^\mathsf{T}\hat{\boldsymbol{\mu}}^+ \qquad \text{and} \qquad f(\hat{\boldsymbol{\mu}}^-) = \mathbf{v}^\mathsf{T}\hat{\boldsymbol{\mu}}^-,$$

where $\hat{\boldsymbol{\mu}}^+$ and $\hat{\boldsymbol{\mu}}^-$ the empirical means of each class, e.i

$$\hat{\boldsymbol{\mu}}^+ = \frac{1}{n^+}\sum_{i=1}^{n^+}\mathbf{x}_i^+ \qquad \text{and} \qquad \hat{\boldsymbol{\mu}}^- = \frac{1}{n^-}\sum_{i=1}^{n^-}\mathbf{x}_i^-$$

Although this method is primarily for dimensionality reduction, but it is a nice layout for implementing a classification method after preforming it. The nature of FDA, which is maximizing between class variance and minimizing within class variance is a way to collapse all data points that belongs to the same class in a condensed fog, achieving, at least visually, a classification task. other than visualisation one can use a classification rule, one often computes then the midpoint of the line and checks at which side a projected data point is located in order to make a decision.

As an example we consider an artificially controlled data generated following a Gaussian distribution and rotated according to a positive number and assigned random binary labels. Performing FDA as a dimensionality reduction technique accounting for the labels differ drastically from PCA, which is used in unsupervised settings. In Figure 1.3 we can see that this approach searches for the best projection on the real numbers' line in such a manner to segregate the two classes.

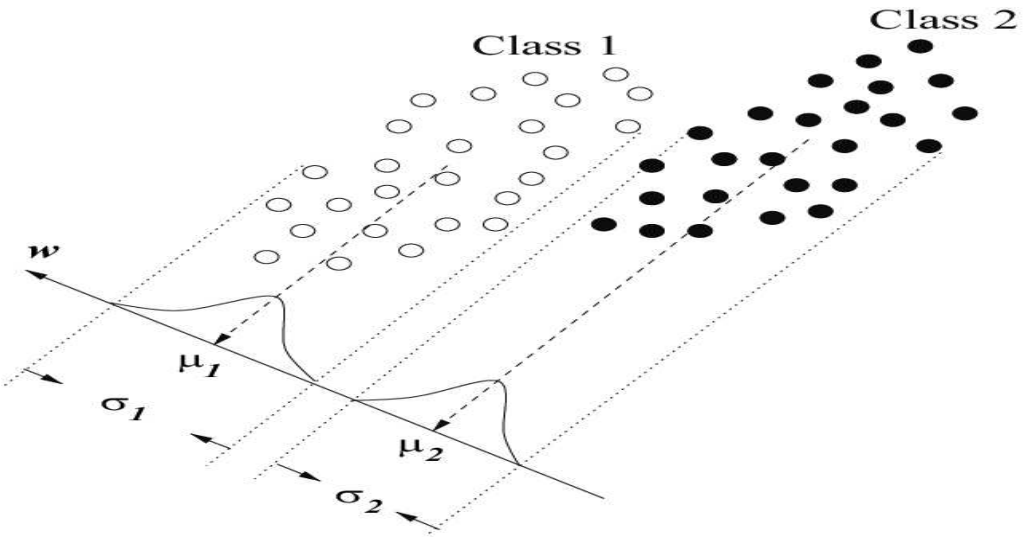

Figure 1.3: Illustration to Fisher Discriminant Analysis technique.

Chapter 2

Support vector Machines

2.1 Introduction

Support vector machines (SVM for short) are a well known family of classifiers, in fact it is widely used in image recognition, pattern recognition[26] and medical decision tools[20]. The simplicity and interpretative nature makes SVM various formulations famously explored since *Vapnick et al.* [24] proposed the notion in 1995. The SVMs simplicity originates from the hypothesis class on which the algorithm operates, that means, the family of linear predictors, one of the most useful families of hypothesis classes. Many learning algorithms that are being widely used in practice rely on linear predictors, first and foremost because of the ability to learn them efficiently in many cases. In addition, linear predictors are intuitive, are easy to interpret, and fit the data reasonably well in many natural learning problems. Formally, an affine function is defined as follows

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \sum_{i=1}^{n} w_i x_i + b, \tag{2.1}$$

where, $\mathbf{w}$ and $\mathbf{x}$ are two vectors of the same size and b a real number. The vector $\mathbf{w}$ and real number b are often called *weight* and *bias* parameters. They are the parameters that controls the function. The learning methodology implies that these parameters must be learned from the data at hand. Formally, one talks about Hypothesis Classes as means of making a ML problem PAC learnable with ERM. First we define the class of affine functions as follows

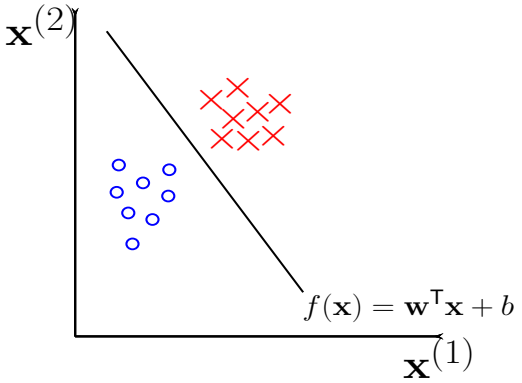

Figure 2.1: Binary SVM classifier. The decision function is formed through the optimal hyperplane $f(\mathbf{x}) = \mathbf{w}^{\top}\mathbf{x} + b$.

$$L_p = \{f_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \mid \mathbf{w} \in \mathbb{R}^n,\ b \in \mathbb{R}\}\,. \tag{2.2}$$

Many Hypothesis Classes Are composition of functions of the form $\phi : \mathbb{R} \longrightarrow \mathcal{Y}$ with function in L_p, in particular $h = I_d$ for Linear regression [27] and $\phi = \text{sign}$ for classification approaches. This latter hypothesis class will be of interest since SVM formulations make the assumption to learn for the class of *halfspaces* $\mathcal{H}_{\mathbf{w},b}$, which in a binary classification framework, is defined as follows:

$$\mathcal{H}_{\mathbf{w},b} = \text{Sign} \circ L_p = \{\text{Sign} \circ f_{\mathbf{w},b}, \quad f_{\mathbf{w},b} \in L_p\} \tag{2.3}$$

In other words, each halfspace hypothesis in $\mathcal{H}_{\mathbf{w},b}$ is parameterized by $\mathbf{w}$ and b and upon receiving a vector $\mathbf{x}$ the hypothesis returns the label $\text{Sign}(f_{\mathbf{w},b}(\mathbf{x}))$.

Let $\mathcal{S} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ be the set of training data, where y_i are the label of each d-dimensional instance $\mathbf{x}_i$, this means, $y_i = 1$ when $\mathbf{x}_i$ belongs to the positive class, and equals -1 when $\mathbf{x}_i$ belongs to the negative class. In the next two sections we will discuss to paradigms of SVM formulations exploring two assumptions, namely, the *Hard-Margin SVM* formulation for linearly separable datasets (see 2.2), in contrast with the contrary case, that is, when the data samples are not linearly separable using *Soft-Margin SVM* formulation for linearly separable datasets (see 2.3).

2.2 Hard margin SVM

In this section, we assume the realizable case. Formally speaking, we say that a data set is linearly separable if there exists a half-space (see 2.1) ($\mathbf{w}$,b), such that $y_i = \text{Sign}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ for each instance $\mathbf{x}_i$. Equivalently, this definition can be rewritten as follows,

$$\forall i \in \{1, \ldots, n\} \qquad y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) > 0 \tag{2.4}$$

By assumption, there will be always a hyperplane to separate the two classes in which no data instance lies within. For that reason, we introduce the notion of "separability with margin δ" which is characterized by a constraint similar to (2.4) as follows

$$\forall i \in \{1, \ldots, n\} \qquad y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) > \delta. \tag{2.5}$$

This leads to the inequality constraint that characterized SVM formulation,

$$\forall i \in \{1, \ldots, n\} \qquad y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) > 1, \tag{2.6}$$

which follows from (2.5), one can divide both sides by the margin δ, and set the new parameters to $\frac{\mathbf{w}}{\delta}$ and $\frac{b}{\delta}$.

Under these assumption, any hyperplane H that satisfy this condition is an ERM learner with zero 0-1 loss function. This leads to various option as to where to draw the line arise. The question then is which one is the best fit to separate the data into classes. One approach is choosing the hyperplane that maximize the distance between the nearest data point and the hyperplane itself, hence, the name *hard margin SVM*. The following lemma characterizes that distance.

Lemma 2.2.1 *Let* $\mathbf{x}$ *be a vector in* $\mathbb{R}^d$ *space and* H *be a hyperplane defined as*

$$\left\{ \mathbf{v} \mid \langle \mathbf{w}, \mathbf{v} \rangle + b = 0, \quad \mathbf{w} \in \mathbb{R}^d, \quad b \in \mathbb{R} \right\}.$$

Then, the distance between H *and* $\mathbf{x}$ *is given by*

$$d(H, \mathbf{x}) = \min\{||\mathbf{x} - v|| \mid \langle \mathbf{w}, \mathbf{v} \rangle + b = 0\} = \frac{|\langle \mathbf{w}, \mathbf{v} \rangle + b|}{||\mathbf{w}||}.$$

The essence of Hard margin-SVM relies on the following optimization problem

$$\begin{cases} \min\limits_{\mathbf{w}} & \frac{1}{2}||\mathbf{w}||^2 \\ \text{s. t.} & y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) \geq 1 \;\; \forall i \in \{1, \ldots, n\} \end{cases} \tag{2.7}$$

This can be interpreted in multiple ways, for instance, maximizing the margin is the same as maximizing $\frac{1}{||\mathbf{w}||}$, which is equivalent to minimizing $||\mathbf{w}||^2$, under the constraint (2.6). Another way to think about it is under the *Regularization Loss Minimization* (RLM) framework (see 1.1.3), where $R(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2$ and 0-1 loss function. The corresponding Lagrangian for the *Hard-Margin SVM* optimization problem introducing Lagrangian multipliers α_i, is expressed as follows

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i \left[y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) - 1\right], \tag{2.8}$$

to which the corresponding dual is found by differentiating with respect to $\mathbf{w}$ and b,

$$* \quad \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$* \quad \frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \Longrightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0.$$

Plugging the proceeding results back into (2.8) yield the following dual optimization problem

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} & \left\{ \mathcal{L}(\boldsymbol{\alpha}) = \sum\limits_{i=1}^{n} \alpha_i - \frac{1}{2} \sum\limits_{i=1}^{n} \sum\limits_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \right\} \\ \text{s. t.} & \sum\limits_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0. \end{cases} \tag{2.9}$$

Let $\boldsymbol{\alpha}^* = (\alpha_i^*)_i$ be the solution of (2.9) and $\mathbf{w}$ is the solution of (2.7), the norm $||\mathbf{w}||$ is given by

$$\langle \mathbf{w}^*, \mathbf{w}^* \rangle = \langle \sum_{i=1}^{n} \alpha_i^* y_i \mathbf{x}_i, \sum_{j=1}^{n} \alpha_j^* y_j \mathbf{x}_j \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i^* \alpha_j^* y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{2.10}$$

then the Hard-Margin SVM classifier follows

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b^* \tag{2.11}$$

Remark 2.2.1 *Note that*

$$\alpha_i \left(y_i f(\mathbf{x}_i) - 1 \right) = 0 \qquad \forall i \in \{1, \ldots, n\}$$

Which means for each i either $\alpha_i = 0$ or $f(\mathbf{x}_i) = \mp 1$. Here lies one component that distinguishes SVM from other classification methods, most lagrangian multipliers will cancel out except for those associated with the sample points lying in the two hyperplanes $f(\mathbf{x}) = 1$ and $f(\mathbf{x}) = -1$. Such sample points are called Support vectors*, it is an important concept because they are the only data points contributing to the decision hyperplane, by changing other sample points the hyperplane does not change. This shows in (2.10) (2.11), the only instances contributing in the norm of $\mathbf{w}^*$, therefore the margin, and the classifier are those for which α_i are nonzero. Another important aspect of this formulation, and for all SVM formulation for that matter, is the emergence of the inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$, this will give rise to kernel implementation of kernels into higher dimensional spaces where linear separability for the data is highly achievable, see chapters 3 and 4.*

Algorithm

Algorithm 4 Hard margin SVM

1: **Input**: Data examples $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$.

2: **Solve**
(2.7) : $(\mathbf{w}^*, b^*) = \text{argmin}_{\mathbf{w}} \frac{1}{2} ||\mathbf{w}||^2$ s. t. $\quad y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) \geq 1 \;\; i \in \{1, \ldots, n\}$

3: **Output:** $\mathbf{w}^* = \dfrac{\mathbf{w}^*}{||\mathbf{w}^*||}$ and $b^* = \dfrac{b^*}{||\mathbf{w}^*||}$

2.3 Soft Margin SVM

The hard margin classifier is an important concept, as a starting point for the analysis and construction of more sophisticated SVM formulations, but it fails to

prevail in many real-world problems. In cases where the data is noisy, there will in general be no linear separation in the original space. The way hard margins SVM was set up is to always produces a perfectly consistent hypothesis, that is a hypothesis with no training error. This is of course a result of its motivation in terms of a bound that depends on the margin, a quantity that is negative unless the data are perfectly separated. The dependence on a quantity like the margin opens the system up to the danger of falling hostage to the idiosyncrasies of a few points. In real life problems, where data is almost certainly noisy, this paradigm can result in a brittle estimator. Furthermore, in the cases where the data are not linearly separable in the original space, the optimisation problem cannot be solved as the primal has an empty feasible region and the dual an unbounded objective function.

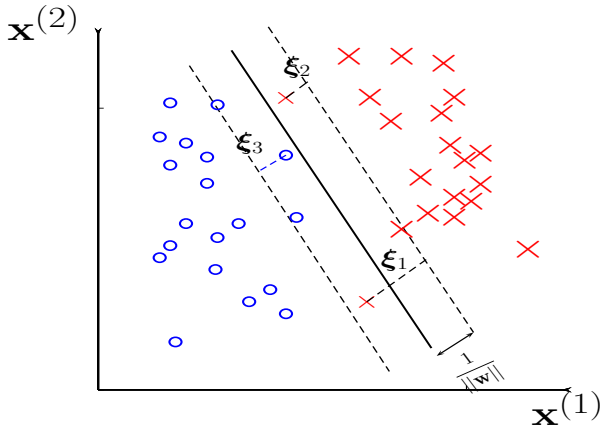

Figure 2.2: Soft Margin Support vector machine, using slack variable ξ_1, ξ_2 and ξ_3 allowing three mistakes while maximizing the margin.

One approach to deal with this problem is relaxing this assumption of realizability. That is, allow the margin constraints to be violated and keep the margin as wide as possible so that other points can still be classified correctly (see 2.2). To do so we introduce the vector of *Slack variables* $\boldsymbol{\xi} = (\xi_i)_{i\in\mathcal{I}}$ into the objective function, where $\mathcal{I}$ stands for the number of mistakes allowed (2.2). Consider the following optimization problem

$$\begin{cases} \min\limits_{\mathbf{w}} & \frac{1}{2}||\mathbf{w}||^2 + C\sum\limits_{i=1}^{n} \xi_i^q \\ \text{s. t.} & y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0 \qquad \forall i \in \{1, \ldots, n\}, \end{cases} \tag{2.12}$$

where $q \in \{1, 2\}$ and C is a trade-off between the amount of mistakes allowed and the maximization of the margin. In practice the parameter C is varied through a wide range of values and the optimal performance assessed using a separate validation set or a technique known as cross-validation for tuning the parameter C according to the data type using only the training set.
Since q can only take one of the two values $\{1, 2\}$, we will devote the next to paragraphs to the two variants of Soft margine SVM called *L1-SVM* (see section 2.3.1) and *L2-SVM* (see section 2.3.2).

2.3.1 1-Norm Soft Margin SVM

Consider the optimization problem in (2.12) where $q = 1$. The L1-SVM stands for the following optimization problem

$$\begin{cases} \min\limits_{\mathbf{w}} & \frac{1}{2}||\mathbf{w}||^2 + C\sum\limits_{i=1}^{n} \xi_i \\ \text{s. t.} & y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) \geq 1 - \xi_i \\ & \xi_i \geq 0 \qquad \forall i \in \{1, \ldots, n\}. \end{cases} \tag{2.13}$$

The corresponding lagrangian is expressed as follows

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}; \boldsymbol{\alpha}, \boldsymbol{\gamma}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i \left[y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) + \xi_i - 1\right] - \sum_{i=1}^{n} \gamma_i \xi_i, \tag{2.14}$$

with $\alpha_i > 0$ and $\gamma_i > 0$.
The corresponding dual is found by differentiating with respect to $\mathbf{w}$, ξ_i and b, setting the partial derivatives to 0

$$* \quad \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$* \quad \frac{\partial L}{\partial b} = -\sum_{1}^{n} \alpha_i y_i = 0 \quad \Longrightarrow \quad \sum_{1}^{n} \alpha_i y_i = 0$$

$$* \quad \frac{\partial L}{\partial \xi_i} = C - \alpha_i - \gamma_i = 0 \quad \Longrightarrow \quad C = \alpha_i + \gamma_i \quad \forall i \in \{1, \ldots, n\}.$$

Plugging the proceeding results back into (2.14) yield the following

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{2.15}$$

Along with the rest of KKT complementary conditions: For all $i \in \{1, \ldots, n\}$

$$\alpha_i \left[y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) + \xi_i - 1 \right] = 0 \tag{2.16}$$

$$\gamma_i \xi_i = 0 \implies (C - \alpha_i) \, \xi_i = 0 \tag{2.17}$$

The proceeding constraints gives an interpretative value to the Soft margin SVM formulation. There are mainly three possible cases for $\boldsymbol{\alpha}$ detailed in the following, for a fixed instance i, we have

1. $\alpha_i = 0$, then $\xi_i = 0$, which means the instance $\mathbf{x}_i$ is correctly classified.

2. $0 < \alpha_i < C$, then following 2.16 we have $y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) + \xi_i - 1 = 0$ and $\xi_i = 0$, therefore $y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) = 1$. In this case $\mathbf{x}_i$ is a support vector that is called *unbounded support vector*, inspired by the fact $0 < \alpha_i < C$.

3. $\alpha_i = C$, then $y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) = 1 - \xi_i$ and $\xi_i \geq 0$, in which case $\mathbf{x}_i$ is a support vector that is called *bounded support vector*. Furthermore, if $0 \leq \xi_i < 1$, $\mathbf{x}_i$ is correctly classified because $y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) > 0$, otherwise if $\xi_i \geq 1$, then $\mathbf{x}_i$ is misclassified.

Let $\boldsymbol{\alpha}^* = (\alpha_i^*)_i$ be the solution of (2.15) and $\mathbf{w}^*$ is the solution of (2.13), the norm $||\mathbf{w}^*||$ is given by

$$\langle \mathbf{w}^*, \mathbf{w}^* \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i^* \alpha_j^* y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

then the *Soft Margin-SVM* classifier follows

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b^* \tag{2.18}$$

Remark 2.3.1 *This problem is equivalent to the maximal margin solutions (2.10) (2.11), with the additional constraint that all Lagrange multipliers are upper bounded by the the trade-off constant C. This gives rise to the name* box constraint *that is frequently used to refer to this formulation, since the vector $\boldsymbol{\alpha}$ is constrained to lie inside the box with side length C in the positive orthant. The trade-off parameter between accuracy and regularisation directly controls the size of the $\boldsymbol{\alpha}$. This makes sense intuitively as the box constraints limit the influence of outliers, which would otherwise have large Lagrange multipliers.*

2.3.2 2-Norm Soft Margin SVM

The last case to discuss is where $q = 2$ in the (2.12) optimization problem,from which the name L2-SVM arose. Consider the following

$$\begin{cases} \min\limits_{\mathbf{w}} & \frac{1}{2}||\mathbf{w}||^2 + C\sum\limits_{i=1}^{n} \xi_i^2 \\ \\ \text{s. t.} & y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) \geq 1 - \xi_i \qquad \forall i \in \{1, \ldots, n\}. \end{cases} \tag{2.19}$$

The corresponding lagrangian for the 2-norm soft margin optimisation problem is

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi} : \boldsymbol{\alpha}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) + \xi_i - 1 \right] \tag{2.20}$$

with $\alpha_i > 0$.

The corresponding dual is found by differentiating with respect to $\mathbf{w}$, ξ_i and b, imposing stationarity,

$$* \quad \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$* \quad \frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \Longrightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$* \quad \frac{\partial L}{\partial \xi_i} = 2C\xi_i - \alpha_i = 0 \quad \Longrightarrow \quad \xi_i = \frac{\alpha_i}{2C} \quad \forall i \in \{1, \ldots, n\}.$$

Plugging the proceeding results back into (2.20) yield the following

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \frac{1}{2C} \sum_{i=1}^{n} \alpha_i^2, \tag{2.21}$$

along with the rest of KKT complementary conditions:

$$\alpha_i \left[y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) + \xi_i - 1 \right] = 0 \quad \forall i \in \{1, \ldots, n\}.$$

Let $\boldsymbol{\alpha}^* = (\alpha_i^*)_i$ be the solution of (2.21) and $\mathbf{w}$ is the solution of (2.19), the norm $||\mathbf{w}||$ is given by

$$\langle \mathbf{w}^*, \mathbf{w}^* \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i^* \alpha_j^* y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

The *Soft Margin-SVM* classifier follows

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b^* \tag{2.22}$$

Remark 2.3.2 *Consider a further formulation of* (2.21) *in the following form*

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\langle \mathbf{x}_i, \mathbf{x}_j \rangle + \frac{\delta_{ij}}{C} \right) \tag{2.23}$$

This formulation will be susceptible for kernel trick implementation as introduced in (3).

Here we introduce a simple pseudo-algorithm for L2-SVM

Algorithm 5 Lq Soft-margin SVM

1: **Input**: Training data $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$.

2: **Solve** 2.12

3: **Output:** $\mathbf{w}^* = \dfrac{\mathbf{w}^*}{||\mathbf{w}^*||}$ and $b^* = \dfrac{b^*}{||\mathbf{w}^*||}$

2.3.3 Illustration : SVM

Consider an artificially controlled data generated by Python's function `make_blobs` scatter plotted in Figure 2.3. Using Python's class function `SVC` we are able to plot the decision boundaries in Figure 2.4 and extracting the support vectors that contribute into the decision making.

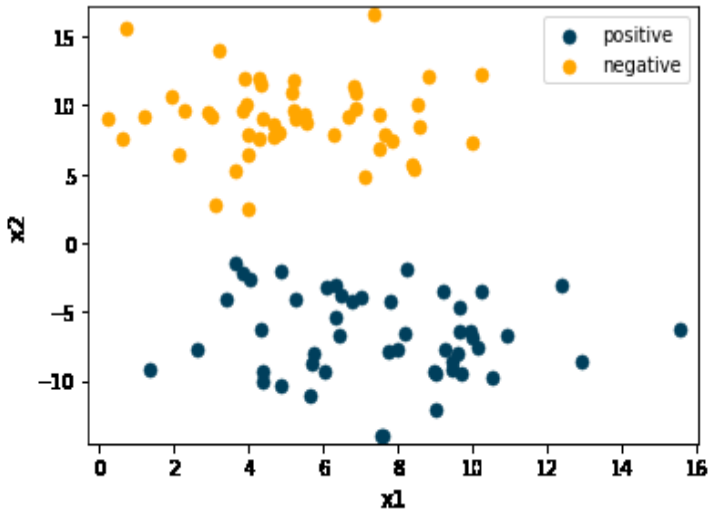

Figure 2.3: Scatter plot of the make blobs data.

Although the hard-margin SVM formulation could well work on this data type, since the realizability assumption holds, but it is always preferable to use soft-margin formulation as it robust to outliers. The number support vectors retrieved from `SVC` class function are 23, in contrast to a 100 original data point. This gives an early indication that a low percentage of the data (in this case 23%) is contributing to the decision making, which means removing or replacing the remaining examples will not effect the boundaries.

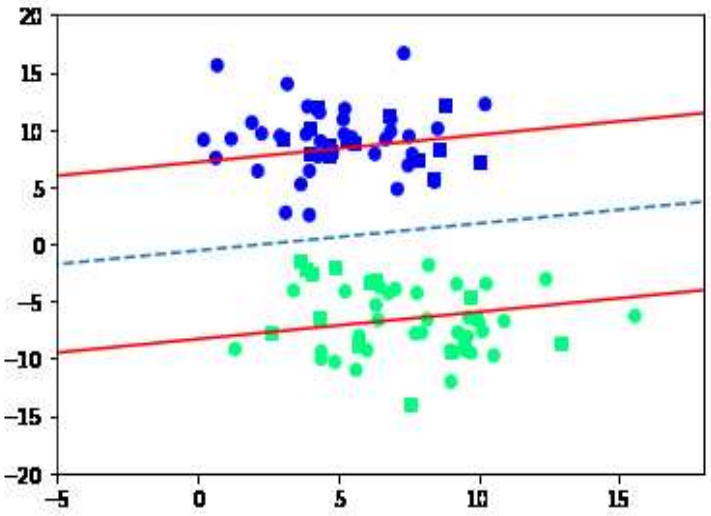

Figure 2.4: Decision boundaries obtained by soft margin SVM approach with *cost* equal to $C = 0.001$, with testing data represented by squares and training data by circles.

Chapter 3

Kernel Support Vector Machines

Although *Soft Margin SVM* formulations remedied to some extent the problem of data inseparability, but most often than not the nature of the data makes it extremely difficult to find a suitable solution. Generally speaking, complex real-world applications require more expressive hypothesis classes than linear functions. In chapter 2, we discussed the SVM decision boundaries, in the context of hard margin (3.12), soft margin with both formulations (2.18) and (2.22). This turns out to be a combination of inner products of the attributes instead of isolated terms (see 2.21, 2.9 and 2.15). This gives rise to kernel representations. We start the present chapter by describing the idea of embedding the data into a high dimensional feature space (3.1). We then introduce the idea of kernels (3.2), which is a type of a similarity measure between instances. The special property of kernel similarities is that they can be viewed as inner products in some Hilbert space (or Euclidean space of some high dimension) to which the instance space is virtually embedded. We introduce the "kernel trick" that enables computationally efficient implementation of learning, without explicitly handling the high dimensional representation of the domain instances. Kernel based learning algorithms, and in particular kernel-SVM as explored in (3.3), are very useful and popular machine learning tools, whether for classification, DR or clustering tasks. Their success may be attributed both to being flexible for accommodating domain specific prior knowledge and to having a well developed set of efficient implementation algorithms.

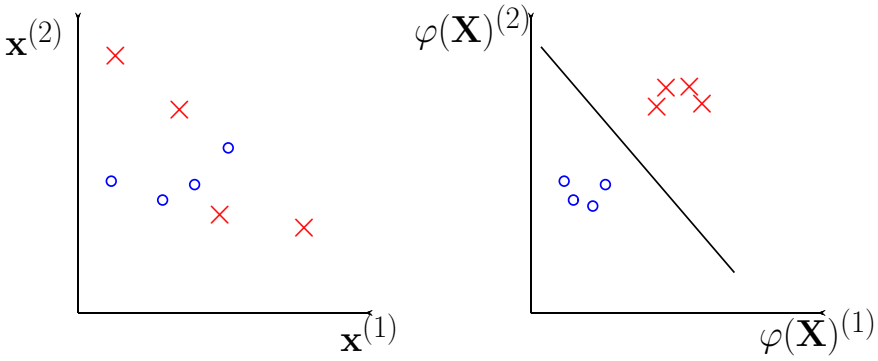

Figure 3.1: A feature map can simplify the classification task

3.1 Learning in feature space

Let $\mathcal{X}$ be the attributes space and $\mathcal{S}$ a data-set containing n attributes $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, the idea of embedding the data into a higher dimensional space, for separability purposes is, choosing a mapping function φ from the attributes space $\mathbb{X}$ to F called feature space. The complexity of the mapping function to be learned depends on the way it is represented, and the difficulty of the learning task can vary accordingly. Ideally a representation that matches the specific learning problem should be chosen. Generally speaking, the preprocessing procedure is as simple as the following

$$\mathbf{x} = (x_1, \ldots, x_d) \longmapsto \varphi(\mathbf{x}) = (\varphi_1(\mathbf{x}), \ldots, \varphi_{d_h}(\mathbf{x}))$$

This step is equivalent to mapping the input space $\mathcal{X}$ into a new space, $\mathcal{F} = \{\varphi(\mathbf{x}) \mid \mathbf{x} \in \mathcal{X}\}$.

Generally speaking the high-level idea is expressed in steps as follows

1. Given some domain set $\mathcal{X}$ and a learning task, choose a mapping $\varphi : \mathcal{X} \to \mathcal{F}$ for some feature space $\mathcal{F}$, that will usually be $\mathbb{R}^{d_h}$ for some d_h (however, the range of such a mapping can be any Hilbert space, including such spaces of infinite dimension).

2. Given a sequence of labeled examples, $\mathcal{S} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$, create the image sequence $\hat{\mathcal{S}} = (\varphi(\mathbf{x}_1), y_1), \ldots, (\varphi(\mathbf{x}_n), y_n)$

3. Train a linear predictor h over $\hat{S}$.

4. Predict the label of a test point, $\mathbf{x}$, to be $h(\varphi(\mathbf{x}))$.

Consider the following example, where the target function

$$f\left(m_1, m_2, r\right) = C\frac{m_1 m_2}{r^2},$$

giving Newton's law of gravitation, expressing the gravitational force between two bodies with masses m_1, m_2 and separation r. This law is expressed in terms of the observable quantities, mass and distance. A support vector classifier as discussed in the previous chapter could not represent it as written, but a simple change of coordinates as follows

$$(m_1, m_2, r) \longmapsto (x, y, z) = (\ln m_1, \ln m_2, \ln r)$$

gives the representation

$$g(x, y, z) = \ln f\left(m_1, m_2, r\right) = \ln C + \ln m_1 + \ln m_2 - 2\ln r = c + x + y - 2z$$

which could be learned by a linear machine.

Knowing the benefits of simple interpretation of a linear classifier, such a simple trick can help simplify to a certain extent the learning procedure, for a linear learning algorithm to do an efficient job. The vectors in the original space $\mathbf{x}_i$ are usually called *attributes* and the transformed vectors $\varphi(\mathbf{x}_i)$ are called *features*. The input space $\mathbf{X}$ is referred to as the *input space* and the new space $\mathbf{F}$ is referred to as *feature space*. The procedure of choosing the most suitable representation φ is known as *feature selection*. Although such a trick is extremely beneficial, but feature selection is subjective to the learning task at hand, which does not guarantee realizablity assumption in the feature space. In this chapter we will explore how such mappings can be made into very high dimensional space combined with so-called *kernel trick* so that linear separation becomes much easier.

3.2 General aspects of Kernel theory

We have seen that embedding the input space into some high dimensional feature space makes halfspace learning more expressive. However, the computational complexity of such learning may still pose a serious hurdle, computing linear separators over very high dimensional data may be computationally dreadful. The common solution to this concern is kernel based learning. The term "kernels"

is used in this context to describe inner products in the feature space. Given an embedding φ of some domain space $\mathcal{X}$ into some high dimensional space $\mathcal{F}$, we define the kernel function $\mathcal{K}(\mathbf{x}, \mathbf{x}') = \langle \varphi(\mathbf{x}), \varphi(\mathbf{x}') \rangle$. One can think of $\mathcal{K}$ as specifying similarity between instances and of the embedding φ as mapping the domain set $\mathcal{X}$ into a space where these similarities are realized as inner products. It turns out that many learning algorithms for halfspaces can be carried out just on the basis of the values of the kernel function over pairs of domain points. The main advantage of such algorithms is that they implement linear separators in high dimensional feature spaces without having to specify points in that space or expressing the embedding φ explicitly. There is two ways of defining kernel functions depending on the mathematical nature of the domain set $\mathcal{X}$. In this section, we will provide both definitions and *Mercer theorem* that leads to realizing the so-called *Kernel Trick*.

Definition 3.2.1 *Let $\mathcal{K} : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ be a kernel function, where $\mathcal{X}$ be a nonempty set. $\mathcal{K}$ is a positive definite kernel if the following inequality hold for any $n \geq 1$, $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$ and scalars $c_1, \ldots, c_n$,*

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{3.1}$$

However, if $\mathcal{X}$ is equipped with a measure ν and $\mathcal{K}$ belongs to $L^2(\mathcal{X} \times \mathcal{X}, \nu \times \nu)$, we say $\mathcal{K}$ is $L^2 -$ positive definite when the following operator

$$\mathcal{K}f(\mathbf{x}) := \int_{\mathcal{X}} \mathcal{K}(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\nu(y)$$

is positive, that is, when the following condition hold

$$\langle \mathcal{K}f, f \rangle_2 = \int_{\mathcal{X}} \left(\int_{\mathcal{X}} \mathcal{K}(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) \right) f(\mathbf{x}) d\nu(\mathbf{x}) \geq 0, \qquad \forall f \in L^2(\mathcal{X}, \nu), \tag{3.2}$$

where, $\langle ., . \rangle_2$ is the inner product of $L^2(\mathcal{X}, \nu)$, which is defined as,

$$\langle f, g \rangle_2 = \int_{\mathcal{X}} f(\mathbf{x}) g(\mathbf{x}) d\nu(\mathbf{x})$$

The following theorem is of a pillar role as to why this kernel implementation to learning theory works, it is called `Mercer` theorem.

Theorem 3.2.1 *Let $\mathcal{X}$ be a topological Hausdorff space, equipped with a finite Borel measure ν, such that, supp$(\nu) = \mathcal{X}$. Then for every continuous positive kernel $\mathcal{K}$, there exists a scalar sequence $(\lambda_n)_n \in l_1$ such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq 0$ and an orthonormal system $\{\varphi_n\}_n$ in $L^2(\mathcal{X}, \nu)$ consisting of continuous functions, such that*

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \sum_{n=1}^{\infty} \lambda_n \varphi_n(\mathbf{x}) \varphi_n(\mathbf{y}) \quad \mathbf{x}, \mathbf{y} \in \mathcal{X} \tag{3.3}$$

Converges uniformly.

The following examples of positive definite kernels are of wide use in kernel-based approaches in machine learning framework.

Example 3.2.1 (Linear Kernel function) *Let $d \in \mathbb{N}$ and $\mathcal{X} = \mathbb{R}^d$ equipped by the Euclidean norm $||\mathbf{x}||_2 = \sqrt{\sum_{i=1}^{d} x_i^2}$. The linear kernel is defined as $\mathcal{K}(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle$ and the Gram matrix is $\mathbf{K} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ for $i, j \in \{1, \ldots, n\}$*
$vspace 0.2cm$.
Let $\mathbf{v} \in \mathbb{R}^d$,

$$\mathbf{v}^t \mathbf{K} \mathbf{v} = \sum_{i=1}^{d} \sum_{j=1}^{d} v_i \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) v_j = \sum_{i=1}^{d} \sum_{j=1}^{d} v_i \langle \mathbf{x}_i, \mathbf{x}_j \rangle v_j = ||\sum_{i=1}^{d} v_i \mathbf{x}_i||_2^2 \geq 0.$$

Since $\mathbf{v}^t \mathbf{K} \mathbf{v}$ is non-negative independently from n, then $\mathcal{K}$ is a positive definite kernel function.

Example 3.2.2 (Gaussian Kernel function) *Let $\mathcal{X}$ be a compact[1] subset of $\mathbb{R}^d$, we define the Gaussian kernel as,*

$$\mathcal{K}(\mathbf{x}, \mathbf{y}) = \exp\left[-\frac{1}{2\sigma^2} ||\mathbf{x} - \mathbf{y}||_2^2\right] \tag{3.4}$$

This theorem opened up a theoretical argument as why kernels are widely used, especially in the dual forms of linear learners. Consider a positive definite kernel

[1] Closed and bounded

operating in a compact subset of $\mathbb{R}^n$, the Mercer theorem grants the following representation,

$$\mathcal{K}(\mathbf{x},\mathbf{y}) = \sum_{n=1}^{\infty} \lambda_n \varphi_n(\mathbf{x})\varphi_n(\mathbf{y}) \quad \mathbf{x},\mathbf{y} \in \mathcal{X} \tag{3.5}$$

$$= \sum_{n=1}^{\infty} \sqrt{\lambda_n}\varphi_n(\mathbf{x})\sqrt{\lambda_n}\varphi_n(\mathbf{y}) \tag{3.6}$$

$$= \varphi(\mathbf{x})^{\mathsf{T}}\varphi(\mathbf{y}) \tag{3.7}$$

where, $\varphi(\mathbf{x}) = \sqrt{\lambda_n}\,[\varphi_1(\mathbf{x}), \ldots \varphi_n(\mathbf{x}) \ldots]$ we can take advantage of this fact to implement high level linear learnablity.

3.3 Kernel methods to Nonlinear data

Let $\{(\mathbf{x}_i, y_i),\ \mathbf{x}_i \in \mathbb{R}^d\}$ and $\{(\varphi(\mathbf{x}_i), y_i),\ \varphi(\mathbf{x}_i) \in \mathbb{R}^{d_h}\}$ the attributes data and feature data, respectively, where $\varphi : \mathbb{R}^d \longrightarrow \mathbb{R}^{d_h}$ the mapping feature which transform the data from the original space to a high dimensional, maybe infinite, feature space. In the next sections we redefine SVM formulation, introduced in chapter 2, with kernel implementation. The intuition behind the formulations stays the same.

3.3.1 Hard-margin SVM

The Hard-Margin SVM classifier applied on the feature data is expressed similarly to (2.7) in the following fashion

$$\begin{cases} \min\limits_{\mathbf{w}} & \dfrac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} \\ \text{s. t.} & y_i\left(\langle \mathbf{w}, \varphi(\mathbf{x}_i)\rangle\right) + b) \geq 1 \ \ \forall i \in \{1, \ldots, n\} \end{cases} \tag{3.8}$$

The corresponding dual is found by differentiating the Lagrangian with respect to $\mathbf{w}$ and b, which yields the following dual optimization problem

$$\begin{cases} \max\limits_{\alpha \in \mathbb{R}^n} & \left\{ \mathcal{Q}(\boldsymbol{\alpha}) = \sum\limits_{i=1}^{n} \alpha_i - \dfrac{1}{2}\sum\limits_{i=1}^{n}\sum\limits_{i=1}^{n} \alpha_i\alpha_i y_i y_j \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j)\rangle \right\} \\ \text{s. t.} & \sum\limits_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0. \end{cases} \tag{3.9}$$

Considering that φ maps onto a high dimensional, possibly infinite, feature space. Computing $\langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$ for every pair of data is nearly impossible. We can define a kernel $\mathcal{K} : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$, such that $\mathcal{K}(\mathbf{x}, \mathbf{y}) = \langle \varphi(\mathbf{x}), \varphi(\mathbf{y}) \rangle$. Then (3.9) naturally becomes,

$$\max_{\alpha \in \mathbb{R}^n} \left\{ \mathcal{Q}(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_i y_i y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) \right\} \tag{3.10}$$

$$\text{s. t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad \alpha_i \geq 0. \tag{3.11}$$

The Kernel SVM classifier follows similarly to 2.11

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i^* y_i \mathcal{K}(\mathbf{x}_i, \mathbf{x}) + b^*. \tag{3.12}$$

3.3.2 Soft Margin SVM

in the previous chapter, we have discussed the two approaches of Soft margin Support vector machines, namely L1-SVM and L2-SVM. The only difference between the two techniques is the norm used to measure the slack variables. The implementation of the Kernel Trick in these algorithms is similar to what we did in the previous section. In this section we will revisit Soft margin SVM using kernels from another perspective, namely through RLM framework. Formally, a regularization function is a mapping $R : \mathbb{R}^n \longrightarrow \mathbb{R}$, and the regularized loss minimization rule aims to solve the following

$$\text{argmin}_{\mathbf{w}} \left\{ L(\mathbf{w}) + R(\mathbf{w}) \right\}. \tag{3.13}$$

There are many possible regularization functions one can use reflecting some prior belief about the problem. We will focus on one of the most simple regularization functions: $R(\mathbf{w}) = \frac{\lambda}{2} ||\mathbf{w}||^2$ where $\lambda > 0$ is a scalar, which reflects the characteristics of SVM formulations, that is maximizing the margin and hence minimizing the norm of $\mathbf{w}$. Which loss function to use is dependant on the learner at hand. In our case, we will work with the *hinge* loss function defined as follows, given a training set $\mathcal{S} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ and a half space $(\mathbf{w}, b)$

$$\ell_{hinge} \left((\mathbf{w}, b), (\mathbf{x}, y) \right) = \max\{0,\ 1 - y \left(\langle \mathbf{w}, \mathbf{x} \rangle + b \right)\}, \tag{3.14}$$

this means the Empirical risk to minimize is

$$\boldsymbol{L}_{\mathcal{S}}^{hinge}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \max\{0,\ 1 - y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right)\} \tag{3.15}$$

The optimization problem for Soft Margin SVM through Kernels is expressed as follows

$$\text{argmin}_{\mathbf{w}} \left\{ \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{n}\sum_{i=1}^{n} \max\{0,\ 1 - y_i\left(\langle \mathbf{w}, \varphi(\mathbf{x})\rangle + b\right)\} \right\} \tag{3.16}$$

This formulation with $L_{hinge}^{\mathcal{S}}$ being convex is a convex optimization problem, but since the hinge loss function is not differentiable, which leads us to solve Soft margin SVM formulation with an iterative approach. Namely, by implementing the SGD algorithm (see algorithm 2) to reach the optimal solution. Stochastic Gradient Descent is of the Gradient descent family [17] in which the iterative step is random, yet sufficient and converges rapidly.

Algorithm 6 Stochastic Gradient Descent for Soft margin SVM

1: **Input**: i_m maximal number of iterations.
2: **Initialize** $\theta^{(1)} = 0$ and $i = 0$.
3: **While** $i < i_m$
$i = i + 1$
Let $\mathbf{w}^{(i)} = \frac{1}{\lambda i}\boldsymbol{\theta}^{(i)}$
Choose i uniformly at random from $\{1, \ldots, n\}$
If $\left(y_i \left\langle \mathbf{w}^{(i)}, \mathbf{x}_i \right\rangle < 1\right)$
Set: $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + y_i \mathbf{x}_i$
Else: $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)}$
4: **Output:** $\mathbf{w}^* = \frac{1}{i_m}\sum_{i=1}^{i_m} \mathbf{w}^{(i)}$

Discussion

Support vector machines formulations in general work extremely well on variety of data types, it belongs to type of learners over a linear hypothesis class which eventually provide a simple and effective interpretation of the results. Major aspect of its success is due to the sparseness of lagrange multipliers, this gives rise

to the notion of support vectors, which is a set of important features contributing to the decision boundaries. Furthermore, the dual formulations of these problems, provides the opportunity for kernel trick implementation which was revolutionary for SVMs in terms of accuracy and robustness.

Chapter 4

Kernel Least Squares SVM

4.1 LS-SVM for supervised learning

4.1.1 LS-SVM for classification

Support vector machines formulation reached a state of fame among ML community for a reason, its capability to deal with linear and nonlinear classification problems by means of Quadratic programming (QP) with a state of the art accuracy[5] is undeniable. But one cannot help but ask the question, *is there a way to further simplify these SVM formulation without loosing its main attributes*, in this chapter we will try to dive into it. Suykens et *al.*[23] proposed the Least-squares Support Vector Machines (LS-SVM) formulation, where they turned the threshold inequality constraint in the L2-SVM formulation (2.19) into an equality constraint that leads to solving a set of linear equations, which is for many practitioners in different areas, easier to use than QP solvers, the following SVM modification was originally proposed by Suykens[23] :

$$\begin{cases} \min\limits_{\mathbf{w}} & \frac{1}{2}\mathbf{w}^{\intercal}\mathbf{w} + \frac{\gamma}{2}\sum\limits_{i=1}^{n} e_i^2 \\ \text{s. t.} & y_i\left(\langle \mathbf{w}, \varphi(\mathbf{x}_i)\rangle + b\right) = 1 - e_i \qquad \forall i \in \{1, \ldots, n\}, \end{cases} \tag{4.1}$$

where $\varphi : \mathbb{R}^d \longrightarrow \mathbb{R}^{d_h}$ is a function which maps the input space into a so-called higher, possibly infinite, dimensional feature space, the weight vector[1] $\mathbf{w} \in \mathbb{R}^{d_h}$,

[1] To not confuse with $\mathbf{w}$ in previous sections, where $\mathbf{w}$ was d-dimensional.

error variable $e_i \in \mathbb{R}$ and a bias term $b \in \mathbb{R}$. One important aspect to discuss about (2.19) (which will be a key motivation to pursue later sections) is the nature of the problem at hand. The cost function implemented is the *squared loss function*, defined in (1.11) which is used when the errors e_i are assumed to be issued from a Gaussian distribution.

Considering that $\mathbf{w}$ is in a higher and possibly infinite dimensional feature space, solving (2.19) in the primal space would be dreadfully and eventually impossible. Furthermore, solving the dual formulation instead is appealing due to its interpretability and simplicity. For the aforementioned reasons corresponding dual with α_i as lagrangian multipliers is expressed as follows,

$$\mathcal{L}(\mathbf{w}, b, e_i, \alpha_i) = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\langle \mathbf{w}, \varphi(x_i) \rangle + b \right) + e_i - 1 \right]. \tag{4.2}$$

The corresponding dual is found by differentiating with respect to $\mathbf{w}$, α_i, e_i and b, imposing stationarity,

$$* \quad \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$* \quad \frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \Longrightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$* \quad \frac{\partial L}{\partial e_i} = e_i - \gamma \alpha_i = 0 \quad \Longrightarrow \quad \alpha_i = \gamma e_i \quad \forall i \in \{1, \ldots, n\}$$

$$* \quad \frac{\partial L}{\partial \alpha} = y_i \left(\langle \mathbf{w}, \varphi(x_i) \rangle + b \right) + e_i - 1 = 0 \quad \Longrightarrow \quad y_i \left(\langle \mathbf{w}, \varphi(x_i) \rangle + b \right) + e_i = 1.$$

Replacing $\mathbf{e}$ and $\mathbf{w}$, we get the following

$$\frac{1}{\gamma}\boldsymbol{\alpha} + \boldsymbol{\alpha}^\mathsf{T}\varphi^\mathsf{T}\varphi + \mathbf{y}b = \mathbf{1}_n,$$

this leads to the following linear Karush-Kuhn-Tucker (KKT) system

$$\left[\begin{array}{c|c} 0 & \mathbf{y}^\mathsf{T} \\ \hline \mathbf{y} & \frac{1}{\gamma}\mathbf{I} + \boldsymbol{\Omega} \end{array}\right] \left[\begin{array}{c} b \\ \boldsymbol{\alpha} \end{array}\right] = \left[\begin{array}{c} 0 \\ \mathbf{1}_n \end{array}\right] \tag{4.3}$$

or

$$\begin{cases} \mathbf{y}^\mathsf{T}\boldsymbol{\alpha} = 0 \\ \left(\frac{1}{\gamma}\mathbf{I} + \boldsymbol{\Omega}\right)\boldsymbol{\alpha} + b\mathbf{y} = \mathbf{1}_n, \end{cases} \tag{4.4}$$

where $\boldsymbol{\Omega} = (\boldsymbol{\Omega}_{i,j})$, with, $\boldsymbol{\Omega}_{i,j} = y_i y_j \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j)\rangle$.
Using the kernel trick[2] within $\boldsymbol{\Omega}_{i,j}$ yields the following,

$$\boldsymbol{\Omega}_{i,j} = y_i y_j \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) \qquad \forall i, j \in \{1, \ldots, n\}.$$

First we prove that $\mathbf{A} = \frac{1}{\gamma}\mathbf{I} + \boldsymbol{\Omega}$ is positive definite, let $\mathbf{c} \in \mathbb{R}^n \setminus \{0\}$

$$\begin{aligned} \mathbf{c}^\mathsf{T}\mathbf{A}\mathbf{c} &= \sum_{i=1}^{n}\sum_{j=1}^{n} c_i^\mathsf{T}\left[\langle\varphi(x_i), \varphi(x_j)\rangle + \frac{\delta_{i,j}}{\gamma}\right] c_j \\ &= \langle \sum_{i=1}^{n} c_i^\mathsf{T}\varphi(x_i), \sum_{j=1}^{n} c_j^\mathsf{T}\varphi(x_j)\rangle + \frac{1}{\gamma}\sum_{i=1}^{n} c_i^2 \\ &= \left|\left|\sum_{i=1}^{n} c_i^\mathsf{T}\varphi(x_i)\right|\right|^2 + \frac{1}{\gamma}\sum_{i=1}^{n} c_i^2 > 0 \qquad \text{(because } \gamma > 0\text{)}. \end{aligned}$$

In this case, $\mathbf{A}$ is invertible, and $\boldsymbol{\alpha}$ is expressed as,

$$\boldsymbol{\alpha} = \mathbf{A}^{-1}\left(\mathbf{1}_p - b\mathbf{y}\right). \tag{4.5}$$

Substituting $\boldsymbol{\alpha}$ in (4.4) yields the expression of the bias term b,

$$\boxed{b = \left(\mathbf{y}^\mathsf{T}\mathbf{A}^{-1}\mathbf{y}\right)^{-1}\left(\mathbf{y}^\mathsf{T}\mathbf{A}^{-1}\mathbf{1}_p\right)} \tag{4.6}$$

then plugging it back in (4.5) yields $\boldsymbol{\alpha}$.

The solution obtained by solving the KKT linear system in (4.3) will be denoted $(\boldsymbol{\alpha}^*, b^*)$, the Least-Squares SVM classifier is expressed as follows,

$$h(\mathbf{x}) = \text{Sign}\left(\sum_{i=1}^{n} \alpha_i^* y_i \mathcal{K}(\mathbf{x}, \mathbf{x_i}) + b^*\right). \tag{4.7}$$

[2]See theorem 3.2.1

This method cannot be used unless the left hand matrix in 4.3 is positive definite, which is not the case. For that, after some algebraic manipulations, the linear system (4.3) is equivalent to

$$\left[\begin{array}{c|c} \mathbf{y}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{y} & 0 \\ \hline 0 & \mathbf{A} \end{array}\right] \left[\begin{array}{c} b \\ \boldsymbol{\alpha} + \mathbf{A}^{-1}\mathbf{y}b \end{array}\right] = \left[\begin{array}{c} \mathbf{y}^{\mathsf{T}}\mathbf{A}^{-1}\mathbf{1}_n \\ \mathbf{1}_n \end{array}\right] \tag{4.8}$$

4.1.2 Algorithm

Algorithm 7 LS-SVM for classification

1: **Input**: Data examples $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$.
Kernel function $\mathcal{K}$.
Regularization parameter γ[3].
2: **Form** $\mathbf{A} = \frac{1}{\gamma}\mathbf{I} + \boldsymbol{\Omega}$, after forming $\boldsymbol{\Omega}_{ij} = \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j)$.
3: **Solve** using CG
η from $\mathbf{A}\eta = \mathbf{y}$.
δ from $\mathbf{A}\delta = \mathbf{1}_n$.
4: **compute** $s = \mathbf{y}^{\mathsf{T}}\eta$.
5: **compute** $b = \frac{1}{s}\left(\eta^{\mathsf{T}}\mathbf{1}_n\right)$ and $\boldsymbol{\alpha} = \delta - \eta b$.
6: **Output:** $\boldsymbol{\alpha}$ and b

4.1.3 Discussing the LS-SVM formulation

It is apparent that solving an optimization problem by means of quadratic programming is harder than solving a set of linear systems. The Least-squares SVM has brought this simplification to the table. Solving LS-SVM means solving the Karush-Kun-Thucker system in (4.4), having said that, this formulation bring many challenges as well. First, the KKT condition $\alpha_i = \gamma e_i \ \forall i$ means that sparsity of $\boldsymbol{\alpha}$ is not guaranteed as it is proportionate to the error vector. Furthermore, every data point is a support vector which means, unlike *Vapnik* SVM formulations, all the data points contributes in the decision making. Granted, the importance of these Support vectors differs with how much $|\alpha_i|$ is close to zero. The points with large $|\alpha_i|$ are located close from the decision boundary. Suykens et *al*..[21] proposed a solution to this draw back by imposing sparseness by pruning. Since not all the data is that relevant to the decision boundary, Sparseness is imposed then by

gradually omitting the least important data from the training set and re-estimating the LS-SVM. The pruning is an approach to reconcile one of the shortcomings of LS-SVM formulation, namely, Sparseness. One can proceed as follows, from the linear system compute the vector of support values, apply the simple heuristic that data corresponding to small α_i values are less relevant for the construction of the model, in analogy with standard SVM formulations where zero α_i values do not contribute to the model. One works then in several steps where in each iteration typically 5 percent of the data with the smallest support values are removed. Then, gradually prunes the support value spectrum. In each of these steps we re-estimates the linear system. Simple LS-SVM pruning pseudo-algorithm is expressed in the following form:

Algorithm 8 Pruning for LS-SVM

1: **Input**: Data examples $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$.
2: **Solve** LSSVM based on 7.
3: **calculate**
refined data-set by removing a small amount of points (e.g. 5 percent of the set) with smallest values in the sorted $|\alpha_i|$ spectrum.
4: **Solve** LSSVM with the new data-set using 7.
5: **Check** for the cross validation performance on the data, if the performance worsen, then break the loop, otherwise go to step 2.
6: **Output:** pruned LSSVM model.

Another aspect to discuss is the algorithm proposed by Suykens et *al.* [23] detailed in algorithm 7, step 3 aims to solve two linear systems involving the matrix $\mathbf{A}$. The size of the matrix $\mathbf{A}$ grows with the number of instances, if n is sufficiently large, then $\mathbf{A}$, depending on the memory of the computer becomes costly to store. In large scale problems, using direct closed form methods can be unfeasible, hence the model training takes longer time to converge. There is diverse techniques to overcome such a drawback, namely, solving the LSSVM implementing iterative methods, such as *Successive Over Relaxation* (SOR for short),*Conjugate Gradient* (GC) and *Generalized Minimal Residual* (GMRES for short). Suykens et *al.* [23] worked with CG iterative method, which showed promising results. Another approach was proposed in [6], where solving two linear systems in step 3 was developed efficiently to a one single step, the preliminary results improved the efficiency of the aforementioned technique.

4.1.4 LS-SVM approach for Kernel FDA

Although the kernel trick is desirable with all the benefits discussed in chapter 3, but implementing it is not always achievable. In SVM formulation, whether in hard margin or soft margin approaches, the classifier is expressed as a function of the inner products of the instances, in that manner kernel trick can be implemented. As discussed, it is due to the difficulty of choosing the feature function φ and the computational cost of calculation every feature $\varphi(\mathbf{x}_i)$ in the feature space F, which can be eventually high-dimensional if not infinite. In that regard, formulation an FDA approach implementing the usage of kernels is impossible, that is due the the target variables z_1 and z_2 in (1.27) being of the following form

$$f(\varphi(\mathbf{x})) = \mathbf{w}^\mathsf{T}\varphi(\mathbf{x}) + b.$$

Having said that, there is a natural implementation of LS-SVM formulation to Kernel FDA. Namely, the equality constraint in LS-SVM (4.1)

$$y_i \left(\langle \mathbf{w}, \varphi(\mathbf{x}_i)\rangle + b\right) = 1 - e_i,$$

can be interpreted as having target values $+1$ and -1 upon which one wants to minimize the errors e_k. This is done by taking a squared error in the loss function which corresponds to minimizing the within-class scatter for both classes. Hence, in the LS-SVM classifier formulation one fixes in fact the between-class scatter by taking target values $+1$ and -1 for the $f(\varphi)$ target variable on the line $f(\varphi(\mathbf{x})) = \mathbf{w}^\mathsf{T}\varphi(\mathbf{x}) + b$. This would lead to a LS-SVM classifier formulation in the primal weight space as follows

$$\begin{cases} \min\limits_{\mathbf{w},b,\mathbf{e}} & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \gamma\frac{1}{2}\left(\sum\limits_{k=1}^{n_1}\left(e_k^{(1)}\right)^2 + \sum\limits_{l=1}^{n_2}\left(e_l^{(2)}\right)^2\right) \\ \text{s. t.} & y_k\left[\mathbf{w}^\mathsf{T}\varphi\left(\mathbf{x}_k\right) + b\right] = 1 - e_k^{(1)}, k = 1, \ldots, n_1 \\ & y_l\left[\mathbf{w}^\mathsf{T}\varphi\left(\mathbf{x}_l\right) + b\right] = 1 - e_l^{(2)}, l = 1, \ldots, n_2 \end{cases} \tag{4.9}$$

where n_1, n_2 are the number of data points belonging to class 1 and 2 respectively. The indices k, l run over the elements of class 1 and 2 , respectively.

4.2 LS-SVM for unsupervised Learning

So far, we have explored LS-SVM formulation for supervised framework, mainly classification tasks. However, LS-SVM technique can be implemented in an unsu-

pervised settings, that is when the labels of the instances are missing. In this section of the chapter, we show how support vector machine alike formulations can also be given to the well-known method of principal component analysis (PCA), which is a frequently used technique in unsupervised learning. The new formulation can be extended in a straightforward way to the nonlinear case by applying the kernel trick, and leads to what is presently called kernel PCA. The scope now is to first formulate linear PCA analysis within the LS-SVM classifier context in section 4.2.1 and, second, to extend this formulation to a high dimensional feature space with application of the kernel trick in section 4.2.2.

4.2.1 LS-SVM approach for linear PCA

We have discussed in chapter 1 the well-known and well-founded dimensionality reduction method *Principal Component Analysis* (PCA), from a pure data-mining prospective, using simple algebraic methods. The premise was to find a direction(s) in a lower dimension on which one preserves the maximum variance possible, which translated into finding the eigenvector associated with the maximum eigenvalue (eventually going down the eigenvectors associated with decreasing sequence of eigenvalues) of the empirical covariance matrix. The LS-SVM formulation approach considers projecting the data into a lower space while maximizing the error $e_i = \left[0 - \mathbf{w}^\mathsf{T} x_i\right]$. That is to consider "$0$" as target value around which the scattered data maintains the maximum variance possible. Consider the following formulation

$$\begin{cases} \max\limits_{\mathbf{w},\mathbf{e}} & \dfrac{\gamma}{2}\displaystyle\sum_{i=1}^{n} e_i^2 - \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} \\ \text{s. t.} & e_i = \mathbf{w}^\mathsf{T}\mathbf{x}_i \qquad \forall i \in \{1,\ldots,n\}. \end{cases} \tag{4.10}$$

This formulation states that one considers the difference between $\mathbf{w}^\mathsf{T} x_i$ (the projected data points to the target lower dimension space) and the value 0 as error variables. The projected variables correspond to what one calls the score variables. These error variables are maximized for the given n data points while keeping the norm of $\mathbf{w}$ small by the regularization term. The value γ is a positive real constant.
The corresponding Lagrangian is expressed as follows

$$\mathcal{L}(\mathbf{w}, e_i; \boldsymbol{\alpha}) = \frac{\gamma}{2}\sum_{i=1}^{n} e_i^2 - \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} - \sum_{i=1}^{n}\alpha_i\left[e_i - \mathbf{w}^\mathsf{T}x_i\right]. \tag{4.11}$$

The corresponding dual is found by differentiating with respect to $\mathbf{w}$, e_i and b

$$\boldsymbol{\Omega}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}, \tag{4.12}$$

where $\boldsymbol{\Omega} = (\boldsymbol{\Omega})_{i,j}$, with $\boldsymbol{\Omega}_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j\rangle$ and $\lambda = \frac{1}{\gamma}$.

This means solving the dual problem stands for solving the Linear system (4.12) which translates to finding the eigenvectors corresponding to the eigenvalues of tht matrix $\boldsymbol{\Omega}$. The score variable that we are interested in $z(\mathbf{x})$ is

$$z(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x} = \sum_{i=1}^{n}\alpha_i\langle \mathbf{x}_i, \mathbf{x}\rangle,$$

where $\boldsymbol{\alpha}$ is the eigenvector corresponding to the largest eigenvalue. Note that all eigenvalues are positive and real because the matrix is symmetric and positive defmite. One has in fact n local minima as solution to the problem for which one selects the solution of interest. The optimal solution is the eigenvector corresponding to the largest eigenvalue because in that case

$$\sum_{i=1}^{n}\left(\mathbf{w}^\mathsf{T}\mathbf{x}_i\right)^2 = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n}\frac{1}{\gamma^2}\alpha_i^2 = \max_{\lambda\text{eigv}}\lambda. \tag{4.13}$$

It is worth mentioning that to use PCA one need to do some reprocessing on the data, namely, centering and standardizing it. That is why we assumed that the data matrix is with mean 0. However, LS-SVM formulation of Principle Component analysis gives the opportunity to work on the data without centering it, mainly through including a bias term b into the score variable $z(\mathbf{x})$.

Including a bias term b One aims to achieve a maximum variance around the target 0, but including a bias term in the process, meaning

$$\max_{\mathbf{w},b}\sum_{i=1}^{n}\left[0 - (\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)\right]^2 \tag{4.14}$$

In the same way as in the previous chapter, the LS-SVM approach to PCA is to consider the distance between the projected data and 0 as an error, which has to be maximized. This translates to the following optimizing problem:

$$\begin{cases} \max\limits_{\mathbf{w},\mathbf{e}} & \frac{\gamma}{2}\sum\limits_{i=1}^{n} e_i^2 - \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} \\ \\ \text{s. t.} & e_i = \mathbf{w}^\mathsf{T} x_i + b \qquad \forall i \in \{1,\ldots,n\}. \end{cases} \tag{4.15}$$

The corresponding lagrangian is expressed as follows

$$\mathcal{L}(\mathbf{w},\mathbf{e},\boldsymbol{\alpha}) = \frac{\gamma}{2}\sum_{i=1}^{n} e_i^2 - \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} - \sum_{i=1}^{n}\alpha_i\left[e_i - \mathbf{w}^\mathsf{T}x_i - b\right]. \tag{4.16}$$

The dual formulation is found by differentiating with respect to $\mathbf{w}$, e_i and b in the following fashion

$$* \quad \frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n}\alpha_i\mathbf{x}_i = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{i=1}^{n}\alpha_i\mathbf{x}_i$$

$$* \quad \frac{\partial L}{\partial e_i} = e_i - \gamma\alpha_i = 0 \quad \Longrightarrow \quad \alpha_i = \gamma e_i \quad \forall i \in \{1,\ldots,n\}$$

$$* \quad \frac{\partial L}{\partial b} = \sum_{i=1}^{n}\alpha_i = 0 \quad \Longrightarrow \quad \sum_{i=1}^{n}\alpha_i = 0$$

$$* \quad \frac{\partial L}{\partial \alpha} = e_i - \mathbf{w}^\mathsf{T}x_i - b = 0 \quad \Longrightarrow \quad e_i = w^\mathsf{T}x_i + b.$$

Using $\sum\limits_{i=1}^{n}\alpha_i = 0$ and $\mathbf{w} = \sum\limits_{i=1}^{n}\alpha_i\mathbf{x}_i$ in the last equation yields an expression for the bias term b,

$$\sum_{i=1}^{n} e_i = \sum_{i=1}^{n}\frac{1}{\gamma}\alpha_i = \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\langle\mathbf{x}_i,\mathbf{x}_j\rangle - nb = 0$$

$$b = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\langle \mathbf{x}_i, \mathbf{x}_j\rangle \tag{4.17}$$

Setting $\lambda = \dfrac{1}{\gamma}$ and eliminating the weights $\mathbf{w}$ and the error term $\mathbf{e}$ yield the following, for all i

$$\begin{aligned}\lambda\alpha_i &= \left(\sum_{j=1}^{n}\alpha_j\mathbf{x}_j\right)^{\mathsf{T}}\mathbf{x}_i - \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\langle \mathbf{x}_i, \mathbf{x}_j\rangle \\ &= \sum_{j=1}^{n}\alpha_j\mathbf{x}_j^{\mathsf{T}}\left(\mathbf{x}_i - \bar{\mathbf{x}}\right) \\ &= \sum_{j=1}^{n}\alpha_j\left(\mathbf{x}_j - \bar{\mathbf{x}}\right)^{\mathsf{T}}\left(\mathbf{x}_i - \bar{\mathbf{x}}\right).\end{aligned}$$

Rewriting the latter expression in a matrix form translates to the following eigenvalues problem

$$\boldsymbol{\Omega}_c\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}, \tag{4.18}$$

where $\boldsymbol{\Omega}_c = (\boldsymbol{\Omega}^c_{i,j})$ with $\boldsymbol{\Omega}^c_{i,j} = \left(\mathbf{x}_j - \bar{\mathbf{x}}\right)^{\mathsf{T}}\left(\mathbf{x}_i - \bar{\mathbf{x}}\right)$, which can be decomposed as $\boldsymbol{\Omega}_c = \mathbf{M}_c\boldsymbol{\Omega}\mathbf{M}_c$ with $\boldsymbol{\Omega}_{i,j} = \langle \mathbf{x}_i, \mathbf{x}_j\rangle$. The matrix $\mathbf{M}_c$ is called the centering term, as it "centers" the matrix $\boldsymbol{\Omega}$, its expression is inducted from (4.25) as follows

$$\mathbf{M}_c = \mathbf{I} - \frac{1}{n}\mathbf{1}_n\mathbf{1}_n^{\mathsf{T}}. \tag{4.19}$$

Including a bias term b in the LS-SVM approach leads to the centering assumption in the conventional PCA approach. By including this bias, reprocessing the data to center it is not required.

The score variable then is expressed as follows

$$z(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b = \sum_{i=1}^{n}\alpha_i\langle \mathbf{x}_i, \mathbf{x}\rangle - \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\langle \mathbf{x}_i, \mathbf{x}_j\rangle$$

where $\boldsymbol{\alpha}$ is the eigenvector corresponding to the largest eigenvalue. Note that all eigenvalues are positive and real because the matrix $\mathbf{M}_c$ is symmetric and positive definite. The optimal solution is the eigenvector corresponding to the largest eigenvalue,

$$\sum_{i=1}^{n} \left(\mathbf{w}^{\mathsf{T}}\mathbf{x}_i + b\right)^2 = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \frac{1}{\gamma^2}\alpha_i^2 = \max_{\lambda \text{eigv}} \lambda. \tag{4.20}$$

4.2.2 LS-SVM approach for Kernel PCA

We have discussed so far the LS-SVM formulations and interpretation of linear PCA, but as we stressed enough in the nonlinear cases, mapping the raw data into a higher dimension feature space using a feature mapping function φ yields impressive results. So in this section we will try to extend the LS-SVM interpretation to the nonlinear PCA. The kernel PCA method was originally introduced by Schölkopf et *al*. in [18]. The main difference is that we now formulate an optimization problem with primal-dual interpretations where the dual problem can be related to kernel PCA. This derivation is in the style of LS-SVM primal-dual formulations and interpretations. For that reason, we consider the same objective as in the first section, namely to maximize

$$\max_{\mathbf{w}} \sum_{i=1}^{n} \left[0 - \mathbf{w}^{\mathsf{T}}(\varphi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}_{\varphi})\right]^2, \tag{4.21}$$

where $\hat{\boldsymbol{\mu}}_{\varphi} = \frac{1}{n}\sum_{i=1}^{n} \varphi(\mathbf{x}_i)$ and $\varphi : \mathbb{R}^d \longrightarrow \mathbb{R}^{d_h}$ the mapping to a high dimensional feature space which might be infinite dimensional. We take here the centering approach instead of using a bias term in the formulation. The following optimization problem is formulated now in the primal weight space,

$$\begin{cases} \max\limits_{\mathbf{w},\mathbf{e}} & \dfrac{\gamma}{2}\displaystyle\sum_{i=1}^{n} e_i^2 - \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} \\ \text{s. t.} & e_i = \mathbf{w}^{\mathsf{T}}(\varphi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}_{\varphi}) \qquad \forall i \in \{1, \ldots, n\}. \end{cases} \tag{4.22}$$

with the corresponding lagrangian

$$\mathcal{L}(\mathbf{w}, \mathbf{e}, \boldsymbol{\alpha}) = \frac{\gamma}{2}\sum_{i=1}^{n} e_i^2 - \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} - \sum_{i=1}^{n} \alpha_i \left[e_i - \mathbf{w}^{\mathsf{T}}(\varphi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}_{\varphi})\right]. \tag{4.23}$$

The corresponding dual is found by differentiating with respect to $\mathbf{w}$, e_i and b, this yields the an equation similar to the linear case

$$\lambda\alpha_i = \sum_{i=1}^{n} \alpha_j \left(\varphi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}_\varphi\right)^\mathsf{T} \left(\varphi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}_\varphi\right) \tag{4.24}$$

Rewriting the latter expression in a matrix form translates to the following eigenvalues problem

$$\boldsymbol{\Omega}_c\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha}, \tag{4.25}$$

where $\boldsymbol{\Omega}_c = (\boldsymbol{\Omega}^c_{i,j})$, with $\boldsymbol{\Omega}^c_{i,j} = \left(\varphi(\mathbf{x}_j) - \hat{\boldsymbol{\mu}}_\varphi\right)^\mathsf{T} \left(\varphi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}_\varphi\right)$. Similarly to Linear PCA, the matrix $\boldsymbol{\Omega}_c$ could be expressed including a centering matrix as $\mathbf{M}_c\boldsymbol{\Omega}_c\mathbf{M}_c$, with $\boldsymbol{\Omega}_{i,j} = \langle\varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j)\rangle$. Expanding the components of $\boldsymbol{\Omega}_c$, we find for any i and j

$$\begin{aligned}
\boldsymbol{\Omega}^c_{i,j} &= \left(\varphi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}_\varphi\right)^\mathsf{T} \left(\varphi(\mathbf{x}_j) - \hat{\boldsymbol{\mu}}_\varphi\right) \\
&= \varphi(\mathbf{x}_i)^\mathsf{T}\varphi(\mathbf{x}_j) - \hat{\boldsymbol{\mu}}_\varphi \left(\varphi(\mathbf{x}_i) + \varphi(\mathbf{x}_j)\right) + \hat{\boldsymbol{\mu}}_\varphi{}^\mathsf{T}\hat{\boldsymbol{\mu}}_\varphi \\
&= \varphi(\mathbf{x}_i)^\mathsf{T}\varphi(\mathbf{x}_j) - \frac{1}{n^2}\sum_{k=1}^{n} \varphi(\mathbf{x}_i)^\mathsf{T}\varphi(\mathbf{x}_k) - \frac{1}{n^2}\sum_{k=1}^{n} \varphi(\mathbf{x}_j)^\mathsf{T}\varphi(\mathbf{x}_k) \\
&+ \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n} \varphi(\mathbf{x}_k)^\mathsf{T}\varphi(\mathbf{x}_l) \\
&= \mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{n^2}\sum_{k=1}^{n} \mathcal{K}(\mathbf{x}_i, \mathbf{x}_k) - \frac{1}{n^2}\sum_{k=1}^{n} \mathcal{K}(\mathbf{x}_j, \mathbf{x}_k) + \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n} \mathcal{K}(\mathbf{x}_k, \mathbf{x}_l).
\end{aligned}$$

The target variable then is expressed after finding $\boldsymbol{\alpha}$ as the eigenvector associated with the largest eigenvalue of the the Gram's matrix as

$$\begin{aligned}
z(\mathbf{x}) &= \mathbf{w}^\mathsf{T} \left(\varphi(\mathbf{x}) - \hat{\boldsymbol{\mu}}_\varphi\right) \\
&= \sum_{i=1}^{n} \alpha_i \left(\varphi(\mathbf{x}_i) - \hat{\boldsymbol{\mu}}_\varphi\right)^\mathsf{T} \left(\varphi(\mathbf{x}) - \hat{\boldsymbol{\mu}}_\varphi\right) \\
&= \sum_{i=1}^{n} \alpha_i \left[\mathcal{K}(\mathbf{x}_i, \mathbf{x}) - \frac{1}{n^2}\sum_{k=1}^{n} \mathcal{K}(\mathbf{x}_i, \mathbf{x}_k) - \frac{1}{n^2}\sum_{k=1}^{n} \mathcal{K}(\mathbf{x}, \mathbf{x}_k)\right. \\
&\left. + \frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n} \mathcal{K}(\mathbf{x}_k, \mathbf{x}_l)\right]
\end{aligned}$$

4.3 Further LSSVM applications

LSSVM formulation was introduced first in the context of classification, extending SVMs for further simplicity, which was provided by turning the conditions in 2.12 into an equality constraint. It resulted into solving one linear equation of the type $\mathbf{Ax} = \mathbf{b}$, although this was essentially simpler for interpretation, but it introduced a couple of drawbacks. In large data context the matrix A becomes a nightmare to store computationally, especially in LSSVM with kernels context, because implementing kernels leads to parameter tuning. Various techniques to face this issue was introduced, such as sparsing by pruning[21], solving the linear system with iterative methods and solving the primal formulation[23]. Generally speaking, LSSVM and SVM formulation with RBF kernel performs exceptionally well in comparison with baseline algorithms (see table 6.1). However, LSSVM formulation was extended to numerous paradigm, in supervised setting for example, LSSVM can be a useful tool for regression type problems[9]. Furthermore, an active area of research within classification task, is *Multiple Expert Learning* (MEL for short)[25], in which the labels are not provided by one expert, but rather multiple expert. That is, the label y_i of one example $\mathbf{x}_i$ could be in cases 1 and in other cases -1, then the question would be how could a learner spot various patterns within an indecisive data set. Researches interested in kernel based theoretical approaches are trying to model this problem using SVMs and then extending to LSSVM formulations. Having said that, great impact of LSSVM reside as well in unsupervised framework, namely *Spectral Clustering*[15, 3], in which various implementation of LSSVM like formulations have a state-of-the-art performance level, whether in static state or dynamic state[14].

Part II

Experiments and results

Chapter 5

Experimental setup

5.1 Database

5.1.1 Toy database

The purpose of toy or controlled datasets, is to assess the weak and strong points of certain algorithms. For that we have selected two types of controlled datasets: `Bull's eye` and `Moons` where each one have numerous variations for analysis purpose. For instance, `Bull's eye` data sets is linearly inseparable, it is controlled by the `variance` within each class, and the distance between each circle (see Figure 5.1). For sensitivity analysis of algorithms to hardly separable data, we tested various classifiers across an array of noise values $(0.1, \ldots, 0.8)$ with 300 fixed sample size.

Name	K	d	n	`noise`
`Bull's eye`	2	2	$[100, \ldots, 500, 1000]$	$[0.01, 0.02, \ldots, 0.08]$
`Moons`	2	2	$[100, \ldots, 500, 1000]$	$[0.5, 1, 1.5, 2, 2.5]$

Table 5.1: CONTROLLED-TOY DATA

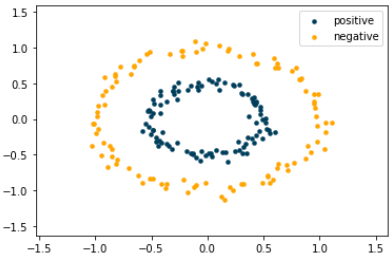

Figure 5.1: Bull's eye data

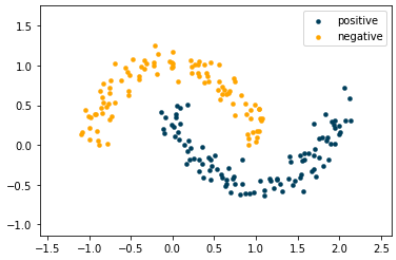

Figure 5.2: Moons data

5.1.2 Real-life database

We will consider 8 datasets collected for various reasons and purposes, ranges from economic, medical diagnosis, game analysis to image recognition. refer to something for taxonomy of data. The UCI datasets can be obtained from the UCI *University of California Irvine (UCI)* benchmark repository. These datasets have been referred numerous times in the literature, which makes them very suitable for benchmarking purposes. As a preprocessing step, all records containing missing values are removed and all data sets were normalized for the benchmarking process. The following binary and multiclass classification datasets were retrieved from [7] and been distinguished into two types: binary and multi-class. The database lists as follows:Breast Cancer (`bc`), Credit card Approval Statlog (`cca`), Glass type (`gls`), hepatitis (`hep`), Johns Hopkins university ionosphere (`ion`), iris (`iri`), sonar (`snr`), Wine quality (`wq`). The main characteristics of these datasets are summarized in Table 5.2.

Name	K	n	d
`bc`	2	569	32
`cca`	2	690	14
`gls`	5	214	7
`hep`	2	270	19
`ion`	2	351	37
`iri`	3	150	4
`snr`	2	208	60
`wq`	3	178	12

Table 5.2: REAL-LIFE DATA

5.2 Performance measures

The most basic metric to evaluate a classification algorithm in a supervised settings is the accuracy score (see 5.2), but there are various disadvantages of such a metric, namely, if the dataset is considerably imbalanced, then the accuracy score can be misleading. In this section we describe and list the set of considered metrics to measure any given classification algorithm to be used in further sections.

* **Confusion matrix** The *confusion matrix* is a sort of detailed accuracy score of the following form

$$\begin{bmatrix} \alpha_{tp} & \alpha_{fp} \\ \alpha_{fn} & \alpha_{tn} \end{bmatrix} \tag{5.1}$$

where, α_{tp}, α_{tn}, α_{fp} and α_{fn} stands for true positive, true negative, false positive and false negative respectively.

* **Accuracy, Precision, Recall and Specificity**

$$p = \frac{\alpha_{tp}}{\alpha_{tp} + \alpha_{fp}} \tag{5.2}$$

$$r = \frac{\alpha_{tn}}{\alpha_{tn} + \alpha_{fn}} \tag{5.3}$$

$$\text{Acc} = \frac{\alpha_{tn} + \alpha_{tp}}{\alpha_{tn} + \alpha_{tp} + \alpha_{fp} + \alpha_{fn}} \tag{5.4}$$

* f_β score

$$f_\beta = \left(1 + \beta^2\right) \frac{p.r}{\beta^2 p + r} \tag{5.5}$$

Precision, Recall and Specificity are measures used in imbalanced data context. Which measure to use depends highly on the type of problem. For instance, in a medical context where, the task is trying to develop a classification model for breast cancer prediction, most of the data sets collected are imbalanced simply because the majority of examples will not have a cancer presence. For that reason, The aforementioned measures are of great use instead of pure accuracy which can be misleading. In this medical context, *false negatives* are crucial in comparison

with *false positive*. In the latter case, the patient will undergo further tests that will determine with great certainty the tumor situation, whereas, the first error is fatal for the patient. For that reason, the best classification model ought to be chosen as to minimize *false negatives* at the expense of *false positives*. A classical example in machine learning is *Spam filtering*, in which minimizing *false positives* is the criterion for a good classification model. f_β score, captures a trade-off between precision and recall controlled by β, the most common values used are 0.5, 1 and 2.

There exists multiple performance measures that vary with the nature and the rigour of the application. Measures like the *Receiver operating characteristic* (ROC for short) and *Area Under the Curve* (AUC for short) are of wide use in computational comparison of classification tasks. We will limit the performance measures to the *Accuracy* (ACC) for balanced data and f_1 *score* for imbalanced data.

5.3 Experiment description

The considered baseline classification approaches are, K-nearest neighbors (KNN), Linear discriminant analysis (LDA), Quadratic discriminant analysis (QDA), Logistic regression (Logit), and Decision Trees (DT). KNN was used with the number of neighbors $K = 3$, the rest of the classifiers do not need any tuning. These classification approaches are embedded in various modules of `Python`'s famous library `Skit-learn`[1] in the following fashion

* KNeighborsClassifier() from `neighbors`module .
* GaussianNB from `naive_bayes` module.
* QuadraticDiscriminantAnalysis from `discriminant_analysis` module.
* LogisticRegression() from `linearmodel` module.
* tree() from `Tree` module.

However, we wish to asses the performance of Support vector machines (SVM) and Least-squares Support vector machines (LSSVM) classifiers using linear, polynomial and radial basis function (RBF) kernel each. These latter algorithms demand tuning some parameters, such as *the cost* C, *regularization parameter* γ,

[1] scikit-learn.org

degree of polynomial kernel d and *the width of RBF kernel* σ. Generally speaking, for tuning parameters we used a cross validation approach in the training data, implementing 10-fold cross validation at times, and 4-fold cross validation when the data is not sufficiently large. To tune these parameters for classic support vector machines we used a built in function in `sklearn`'s module `model_selection` named `GridSearchCV` with 10-fold cross validation (indicated by the optional variable `cv`$= 10$ in `GridSearchCV` class). Also, we used two arrays of various *Cost* and *gamma* ranging between 10^{-2} and 10^{3}, 10^{-2} and 10^{2} with a step of 10^{-1} respectively. Regarding the degree of the polynomial kernel, we fixed the value at $d = 3$ as the classifier tend to overfit when the degree exceeds 3. On the other hand, for Least-Squares Support Vector Machines, we used three costumed functions in python titled `Grid_rbf`, `Grid_poly` and `Grid_rbf` for RBF, polynomial and linear kernel functions, respectively. The algorithm for RBF-LSSVM grid search details as follows

Algorithm 9 RBF Grid search

1: **Input**: Γ vector containing the span of γ values.
Σ vector containing the span of σ values.
$\mathbf{X}_{train}$, $\mathbf{y}_{train}$training matrix of attributes and response variable.
i_{max} maximal number of iterations.

2: **Initialize**: $i = 0$, $\Sigma_0 = \sqrt{n}\,\Sigma$ and $\Gamma_0 = \Gamma$

3: **While**: $i < i_{max}$:

Apply 10-fold cross-validation on the training/validation data for each (σ, γ) combination from the $\Sigma_i \times \Gamma_i$.

Choose optimal (σ, γ) from the tuning sets $\Sigma_i \times \Gamma_i$ by looking at the best cross-validation performance.

$i = i + 1$ and construct a locally refined grid $\Sigma_i \times \Gamma_i$ around the optimal hyperparameters (σ, γ).

4: Construct the LS-SVM classifier using the total training/validation set for the optimal choice of the tuned hyperparameters (σ, γ).

5: **Output:** (σ^*, γ^*)

The cross-validation procedure in this algorithm was done using `StratifiedKFold` function embedded in `sklearn`'s module `model_selection` instead of the `cross_val_score` function, for reasons of control. Using `Strat-`

`ifiedKFold` grants the freedom to control the the number of splits in a way to well represent multiple classes in each split. For the entire benchmark study, we used the maximal iterations to be $i_{max} = 3$ whereas, we used Γ and Σ to be :

$$\Gamma = \{0.5, 5, 10, 15, 25, 50, 100, 250, 500\}$$

$$\Sigma = \{0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 500, 1000\}.$$

For polynomial kernel's parameters (γ, d), the algorithm follows the same philosophy, instead of a vector containing a span of σ values, we used a vector of degrees $\mathbf{d}$ of the following values :

$$\mathbf{d} = \{2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

For better interpretation, for each algorithm we average `accuracy` over ten times randomized training and testing datasets. The average accuracy for each algorithm across all datasets is reported in Table 6.1 for the considered real life datasets and Figures 6.1, 6.2 and 6.3 for artificial datasets, for a better grasp on the performance of the algorithms.

5.3.1 Experiment description on artificial data

The usage of artificially controlled data is simply to try and assess the strong and weak aspects of each classification approach. The objective of the first experiment conducted in this regard, is to measure how sensitive these algorithms are with respect to hardly separable datasets. In that regard we consider two aforementioned datasets : Bull's eye and Moons. First we tweak the noise[2] variable gradually from 0.1 to 0.9 fixing the number of data samples at 200 and from 1 to 2.5 with step 0.5 fixing the data samples at 100. Since it is a balanced data set of 50% each, the accuracy score of each algorithm is recorded. The results are shown in graph format 6.1 using Excel. Furthermore, time of training-testing of each technique was reported to better understand time complexity of each algorithm, for that purpose we used `Time` library in Python, the results are reported in 6.7, 6.6, 6.5 and 6.4.

[2] noise variable built in the class `make_circles` in python sklearn

5.3.2 Experiment description on real-life data

In this section we will draw a computational experiment on multiple data sets, binary and multi-class. Eight real life data sets reported in 5.2 are considered. This classification comparison is meant to be a preliminary experiment that would hopefully indicate the more "sophisticated" methods of comparison to draw next. The experiment is basically comparing the average accuracy, training time, testing time over 10 randomized trials of the algorithms mentioned in 5.3, in which we will restrict ourselves to fairly balanced binary and multi-class datasets reported in Table 5.2. Furthermore, most of the considered datasets contained missing values (usually indicated as `NaN` or *N/A* in python), approaches to remedy this problem such as imputation of missing values are discussed in [10, 12, 16]. Most common approach to such an issue, is considering the column in which the missing value occurs as a continuous variable explained by the other explanatory variables through a sub-model (usually linear regression model). In our case, for simplicity purposes, any data sets containing missing values were trimmed down by deleting the rows containing any missing value.

Chapter 6

Results and discussion

6.1 Controlled data sets

6.1.1 Bull's eye

The results reported in Figure 6.1 suggest a good performance of RBF kernel for both SVM and LSSVM, as the noise increased, the performance kept an acceptable rate head-to-head with KNN state-of-the art algorithm. Whereas other baseline algorithms like LDA, QDA and logistic regression performed poorly. As for DT algorithm correlated negatively with the complexity of the data's separability. LSSVM and SVM with RBF kernel ranged roughly between 0.8 and 1 accuracy score throughout the increase of data's complexity. This could be explained by the utility of the kernel matrix, in unsupervised settings the kernel matrix's entries can express the similarities between data points in a precise manner, capturing the similarities can lead to a state of the art accuracy in supervised settings with the extra information that the labels provide. Figure 6.2 provides another aspect of RBF-LSSVM and RBF-SVM advantages in terms of the sample size. For KNN the accuracy improves as the sample increases which make sense, more points of the same class will have more neighbors, not until sample size reaches 1000 that its accuracy exceeds 90%, whereas for LSSVM with RBF and polynomial kernels record over 90% of accuracy across sample sizes.

6.1.2 Moons

The results of the experiments on moons data are reported in the Figure 6.6 and 6.7 which explain the major drawbacks of LSSVM formulation for classification,

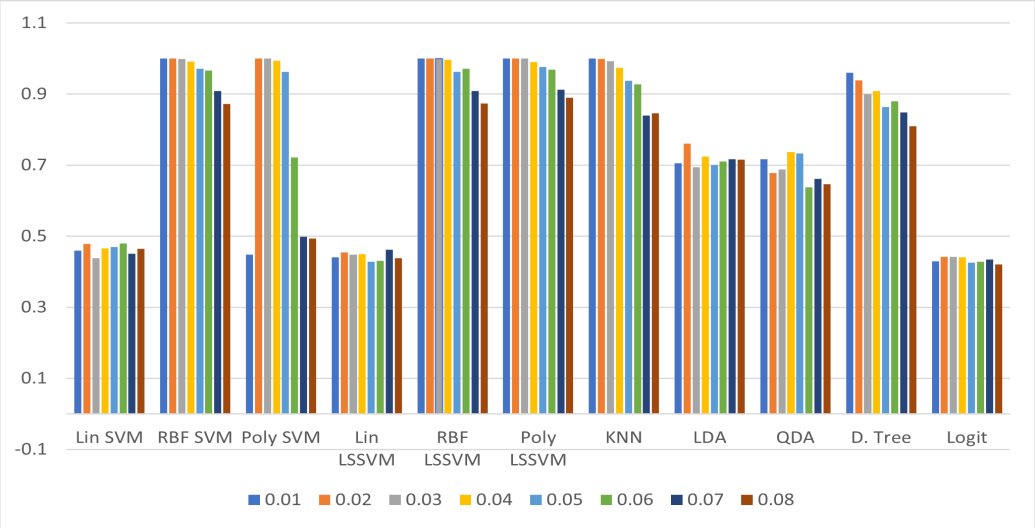

Figure 6.1: Performance comparison of multiple classification algorithm to the different bull's eye variations ranging from 0.01 to 0.08 incrementally

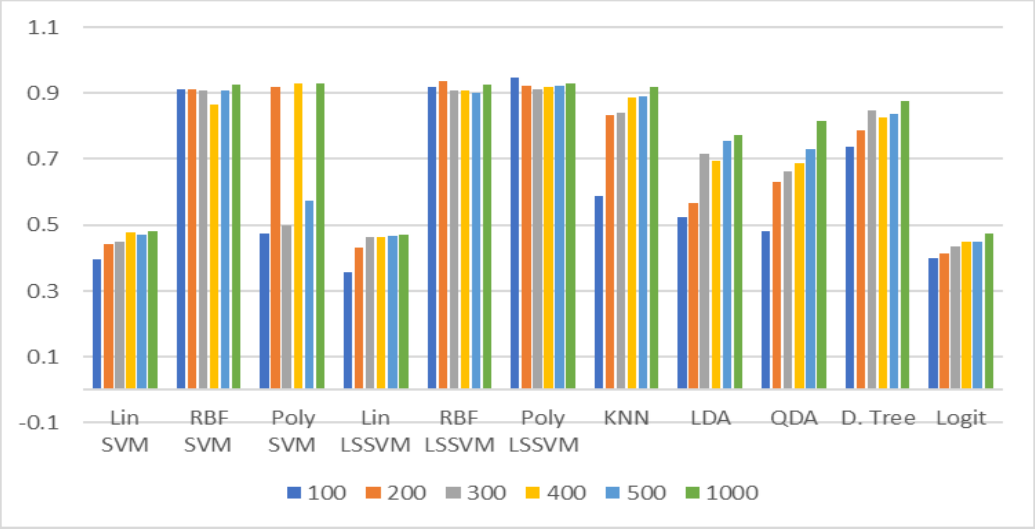

Figure 6.2: Performance comparison of multiple classification algorithm to bull's eye data with samples sizes 100, 200, 300, 400, 500 and 1000

as discussed in the theoretical part, an equality constraint leads to losing sparsity of the lagrangian multipliers, so every data point contributes to the decision mak-

ing. This is translated in Figures 6.6 and 6.7 as the training time of LSSVM with various kernels (even linear) is significantly higher than the other classifiers. Various approaches were introduced to deal with such a drawback, namely sparsing by pruning[21] and other iterative approaches. Having said that, with respect to testing time, LSSVM with RBF kernel takes significantly less time than KKT algorithm (6.5) with at least the same, if not higher performance accuracy, this is due to KNN calculating the euclidean distance to of each testing point to make a decision.

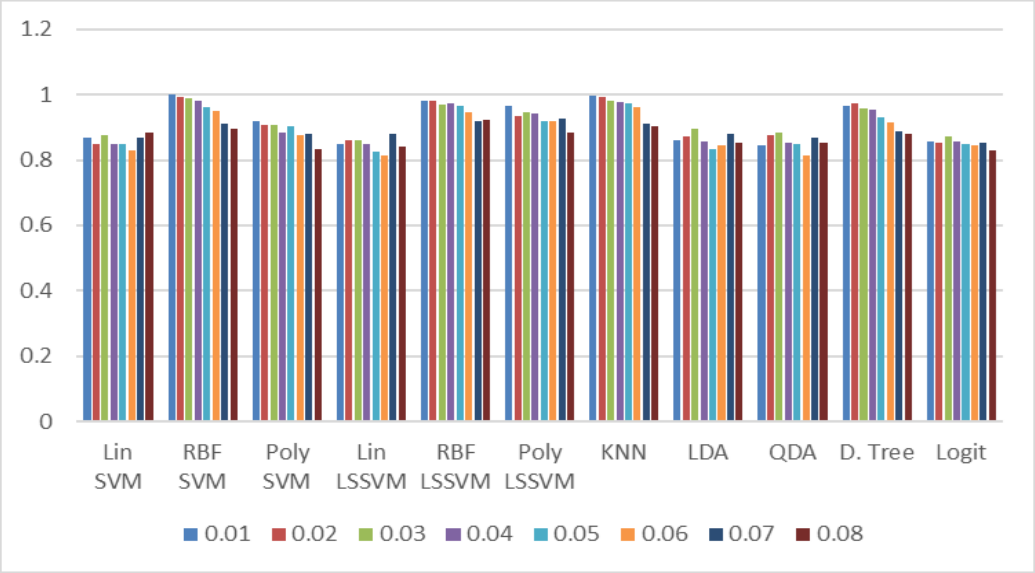

Figure 6.3: Performance comparison of multiple classification algorithm to the different Moons variations

Figure 6.4: Training time of each classification algorithms on moons data with noise = 0.1 and sample size $n = 1000$

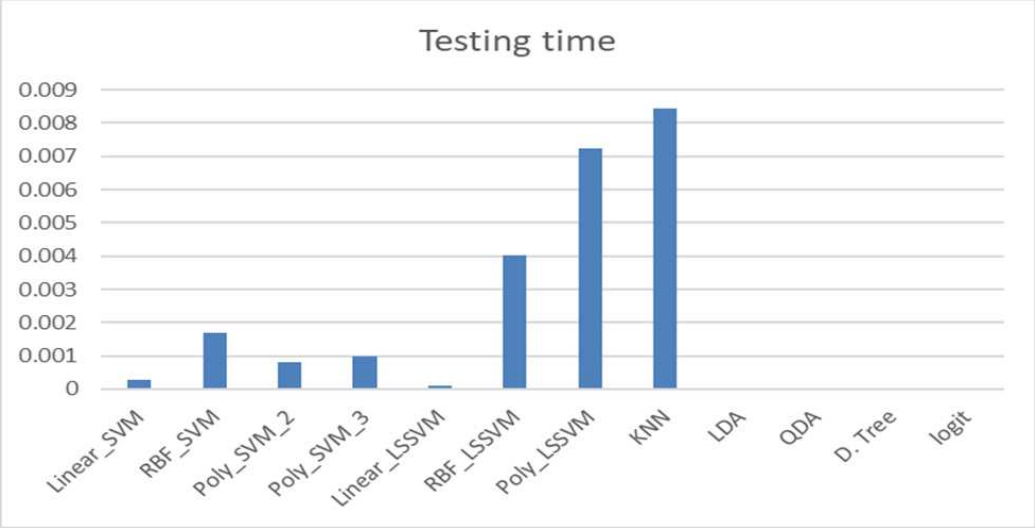

Figure 6.5: Testing time of moons dataset with noise = 0.1 and sample size $n = 1000$

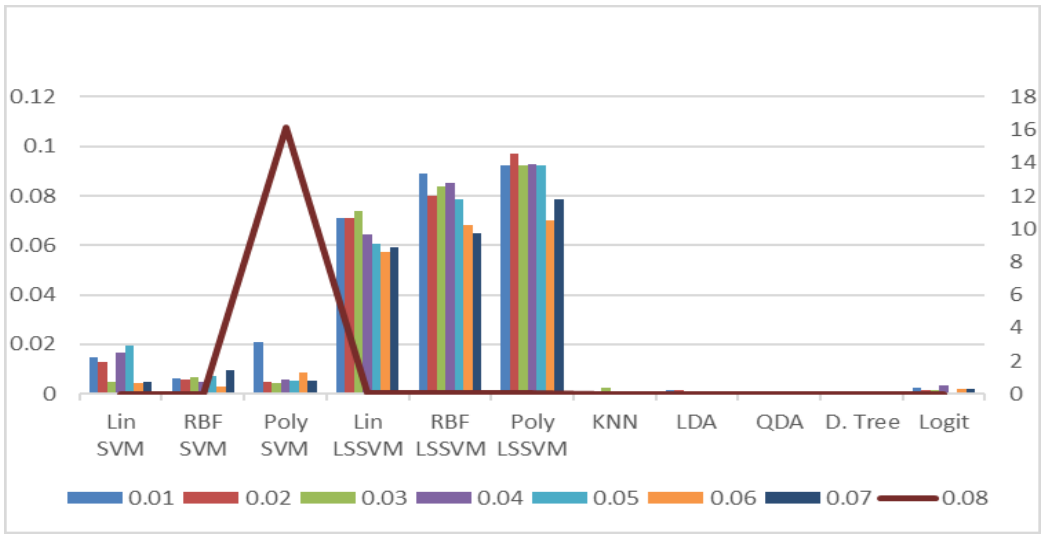

Figure 6.6: Training time across 8 variations of moons dataset with sample size $n = 1000$.

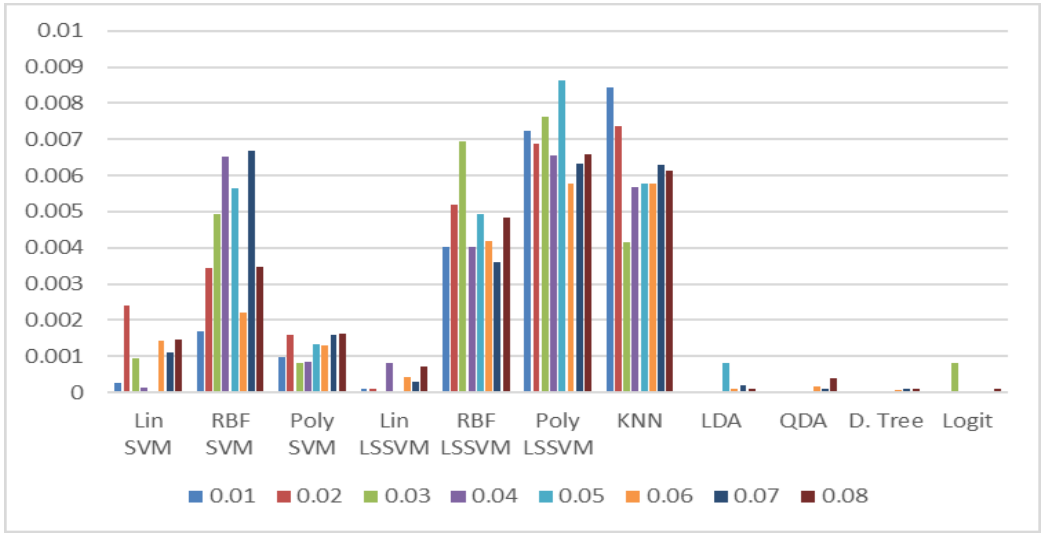

Figure 6.7: Testing time across 8 variations of moons dataset with sample size $n = 1000$.

6.2 Real-life datasets

n	bc	crx	gls	hea	ion	iri	snr	wq		
n_{train}	512	621	192	243	315	135	187	160		
n_{test}	57	69	22	27	36	15	21	18	ACC	AR
SVM_lin	0.858	0.855	0.628	0.826	0.871	0.953	0.752	0.972	0.84	4.5
SVM_rbf	0.861	0.863	0.637	0.869	0.944	0.969	0.878	0.97	**0.87**	**1.5**
SVM_poly	0.845	0.851	0.628	0.802	0.835	0.956	0.856	0.956	0.84	4.5
LSSVM_lin	0.861	0.855	0.571	0.858	0.872	0.858	0.765	0.978	0.83	6.5
LSSVM_rbf	0.876	0.862	0.668	0.844	0.949	0.976	0.827	0.98	**0.87**	**1.5**
LSSVM_poly	0.685	0.64	0.577	0.725	0.791	0.962	0.822	0.931	0.77	10.5
KNN	0.821	0.83	0.629	0.822	0.828	0.971	0.811	0.956	0.83	6.5
LDA	0.812	0.796	0.44	0.848	0.878	0.942	0.71	0.981	0.8	9
QDA	0.809	0.675	0.435	0.802	0.909	0.958	0.625	0.978	0.77	10.5
DT	0.817	0.845	0.663	0.73	0.888	0.94	0.681	0.909	0.81	8
Logit	0.872	0.858	0.638	0.847	0.874	0.953	0.789	0.976	**0.85**	**3**

Table 6.1: Comparison of the ten times randomized test set performance of LS-SVM and SVM, both with linear, polynomial (poly) and Radial Basis Function (RBF) kernels, in contrast with five base line classifiers KNN, LDA, QDA, DT and Logit on 8 binary and multiclass data sets. The average accuracy (Av.acc) and average Average Rank (Avv.R) are reported in the last two columns of the table. Both SVM and LSSVM with RBF kernel performed well on average.

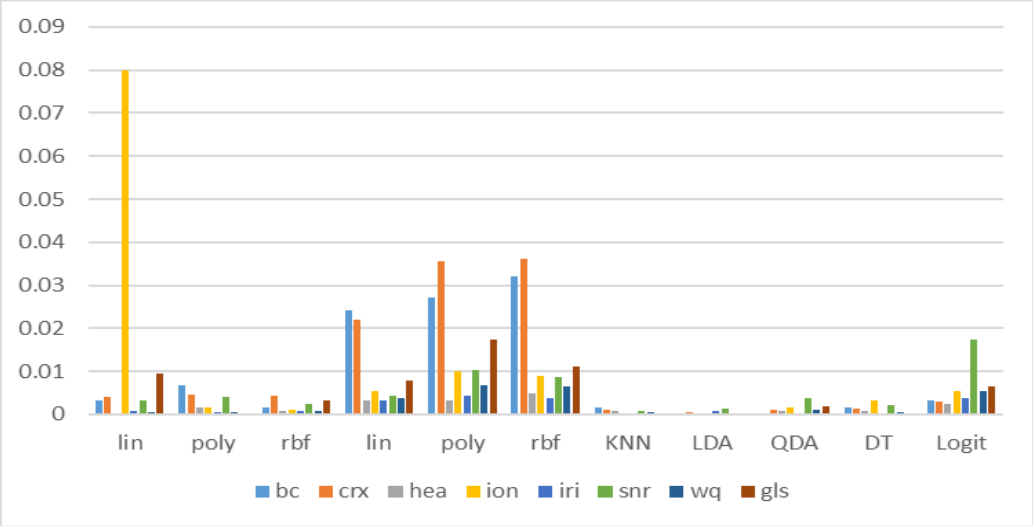

Figure 6.8: Comparison of training time of the considered classifiers across 8 real-life datasets

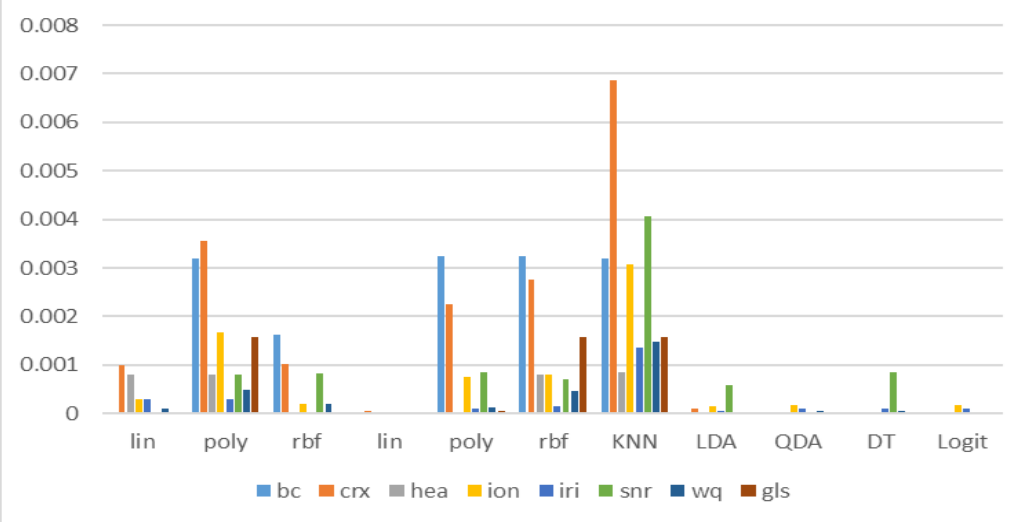

Figure 6.9: Comparison of testing time of the considered classifiers across 8 real-life datasets

As discussed in artificial datasets results, `LSSVM` approach with linear, polynomial and RBF kernels require more time in the training process (6.8). Whereas,

KNN classifier takes more time in the testing process, this is due to labeling unseen data requires calculating the distance to each new example to all points (6.9).

Bibliography

[1] BALTRUŠAITIS, T., AHUJA, C., AND MORENCY, L.-P. Multimodal machine learning: A survey and taxonomy. *IEEE transactions on pattern analysis and machine intelligence 41*, 2 (2018), 423–443. 1

[2] BARBER, D. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012. 1.1

[3] BLANCO VALENCIA, X. P., BECERRA, M., CASTRO OSPINA, A., ORTEGA ADARME, M., VIVEROS MELO, D., PELUFFO ORDÓÑEZ, D., ET AL. Kernel-based framework for spectral dimensionality reduction and clustering formulation: A theoretical study. 4.3

[4] CAIN, K., OAKHILL, J. V., AND ELBRO, C. The ability to learn new word meanings from context by school-age children with and without language comprehension difficulties. *Journal of child language 30*, 3 (2003), 681–694. (document)

[5] CHOROWSKI, J., WANG, J., AND ZURADA, J. M. Review and performance comparison of svm-and elm-based classifiers. *Neurocomputing 128* (2014), 507–516. 4.1.1

[6] CHU, W., ONG, C. J., AND KEERTHI, S. S. An improved conjugate gradient scheme to the solution of least squares svm. *IEEE Transactions on neural networks 16*, 2 (2005), 498–501. 4.1.3

[7] DUA, D., AND GRAFF, C. UCI machine learning repository, 2017. 1.2.1, 5.1.2

[8] HEALY, L. M. Logistic regression: An overview. *Eastern Michighan College of Technology* (2006). 1.1.4

[9] HOU, L., YANG, Q., AND AN, J. An improved lssvm regression algorithm. In *2009 International Conference on Computational Intelligence and Natural Computing* (2009), vol. 2, IEEE, pp. 138–140. 4.3

[10] HOUARI, R., BOUNCEUR, A., KECHADI, T., ABDELKAMEL, T., AND EULER, R. A new method for estimation of missing data based on sampling methods for data mining. In *Advances in computational science, engineering and information technology*. Springer, 2013, pp. 89–100. 5.3.2

[11] JOLLIFFE, I. T. Principal components in regression analysis. In *Principal component analysis*. Springer, 1986, pp. 129–155. 1.2.1

[12] LIN, W.-C., AND TSAI, C.-F. Missing value imputation: a review and analysis of the literature (2006–2017). *Artificial Intelligence Review 53*, 2 (2020), 1487–1509. 5.3.2

[13] MKHADRI, A., CELEUX, G., AND NASROALLAH, A. Regularization in discriminant analysis: an overview. *Computational Statistics & Data Analysis 23*, 3 (1997), 403–423. 1.1.3, 1.1.4

[14] PELUFFO-ORDÓNEZ, D. H., GARCÍA-VEGA, S., ALVAREZ-MEZA, A. M., AND CASTELLANOS-DOMÍNGUEZ, C. G. Kernel spectral clustering for dynamic data. In *Iberoamerican Congress on Pattern Recognition* (2013), Springer, pp. 238–245. 4.3

[15] PELUFFO-ORDÓNEZ, D. H., LEE, J. A., AND VERLEYSEN, M. Generalized kernel framework for unsupervised spectral methods of dimensionality reduction. In *2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)* (2014), IEEE, pp. 171–177. 4.3

[16] RUBIN, D. B. An overview of multiple imputation. In *Proceedings of the survey research methods section of the American statistical association* (1988), Citeseer, pp. 79–84. 5.3.2

[17] RUDER, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747* (2016). 3.3.2

[18] SCHÖLKOPF, B., SMOLA, A., AND MÜLLER, K.-R. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation 10*, 5 (1998), 1299–1319. 4.2.2

[19] SHALEV-SHWARTZ, S., AND BEN-DAVID, S. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014. 1.1, 1.1.4, 1.1.4

[20] STOEAN, R., AND STOEAN, C. Modeling medical decision making by support vector machines, explaining by rules of evolutionary algorithms with feature selection. *Expert Systems with Applications 40*, 7 (2013), 2677–2686. 2.1

[21] SUYKENS, J. A., LUKAS, L., AND VANDEWALLE, J. Sparse least squares support vector machine classifiers. In *ESANN* (2000), Citeseer, pp. 37–42. 4.1.3, 4.3, 6.1.2

[22] SUYKENS, J. A., VAN GESTEL, T., DE BRABANTER, J., DE MOOR, B., AND VANDEWALLE, J. P. *Least squares support vector machines*. World scientific, 2002. (document)

[23] SUYKENS, J. A., AND VANDEWALLE, J. Least squares support vector machine classifiers. *Neural processing letters 9*, 3 (1999), 293–300. (document), 4.1.1, 4.1.3, 4.3

[24] VAPNIK, V., GUYON, I., AND HASTIE, T. Support vector machines. (document), 2.1

[25] WALLACE, B. C., SMALL, K., BRODLEY, C. E., AND TRIKALINOS, T. A. Who should label what? instance allocation in multiple expert active learning. In *Proceedings of the 2011 SIAM international conference on data mining* (2011), SIAM, pp. 176–187. 4.3

[26] WU, Y.-C., LEE, Y.-S., AND YANG, J.-C. Robust and efficient multiclass svm models for phrase pattern recognition. *Pattern recognition 41*, 9 (2008), 2874–2889. 2.1

[27] YU, C., AND YAO, W. Robust linear regression: A review and comparison. *Communications in Statistics-Simulation and Computation 46*, 8 (2017), 6261–6282. 2.1

[28] ZOU, J., HAN, Y., AND SO, S.-S. Overview of artificial neural networks. *Artificial Neural Networks* (2008), 14–22. 1.1.4