University of Bergen
Department of Informatics

An Optimization Perspective: Understanding the Supervised Learning Landscape

Author: Marius H. Naasen
Supervisor: Jan-J. Rückmann
Co-supervisor: Troels A. Bojesen

UNIVERSITY OF BERGEN
Faculty of Mathematics and Natural Sciences

May, 2024

Abstract

The learning process of Machine Learning (ML) involves applying principles from mathematical optimization. This master's thesis explored Supervised Learning (SL), which is a subfield of ML where a computer program learns from a dataset under the guidance of a knowledgeable supervisor. Our focus centered on two key ML algorithms: linear regression and Support Vector Machines (SVM). For linear regression, we discussed well-established optimization models, namely, Maximum Likelihood Estimation (MLE) and Maximum a Posteriori (MAP). After presenting what can be considered the primal problem, specifically, the soft-margin SVM, we examined the optimization used to derive the corresponding dual problem. We also emphasized Hyperparameter Optimization (HPO), acknowledging the influence of algorithm settings on ML program performance through hyperparameter tuning, presented here as a bilevel optimization model. In conclusion, numerical experiments for each discussed SL algorithm were provided, highlighting their characteristics when altering the corresponding hyperparameter values.

Acknowledgements

You have been nothing but good to me as a supervisor; yet, I see you more as a mentor—Jan. Taking all your optimization courses has allowed me to evolve as a student of mathematics, and I have thoroughly enjoyed working alongside you as a TA this final autumn semester. Experiencing the fulfillment of holding my first pair of lectures was indeed a joy! Thank you for all the guidance and professional growth you have given me during my master's studies.

Furthermore, I would like to extend my sincere gratitude to you, Troels, as my second supervisor. With your commitment and persistence, you have challenged and assisted me with shaping the structure of this master's thesis, making it more coherent.

In the final days leading up to the completion of my master's studies, my father has helped motivate me, and his dedication to assisting me in correcting the final text has been greatly appreciated.

Til slutt vil jeg uttrykke min takknemlighet til min kjære gruppe bestående av Hanna, Maria og Tora. Jeg føler glede over å ha fått bli kjent med dere og alt vi har lært av hverandre. Å ha fått dele denne tiden med dere har gjort meg til et bedre menneske!

`Marius H. Naasen`
Wednesday 8$^{\text{th}}$ May, 2024

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Famously quoted by Tom M. Mitchell and perhaps the most well-known statement in the field of Machine Learning (ML), it is often introduced to anyone who wishes to endeavor into the field:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

This encapsulates the learning process of ML, which concerns itself with computer programs that can learn to perform tasks by themselves, without being explicitly told how, often achieving a capability comparable to that of human level or even surpassing it.

Adding further context, the focus of ML is to construct computer programs that go beyond memorization, to a degree of understanding that grants generalization of a given task. Using the terms by Mitchell, in working with task T, an ML program aims to achieve an understanding of experience E to such an extent that it is capable of going beyond and generalize to perform well on experiences not yet encountered.

The concept of learning may sound abstract and is typically associated with capabilities of conscious living beings. It might seem unnatural for a deterministic machine, operating and expressing itself in terms of 0 and 1, to possess learning capabilities. Nevertheless, the deterministic nature of a machine and its expression in terms of 0 and 1 becomes a powerful advantage when coupled with the principles of optimization. At the

very heart of the learning process of a ML program lies the use of optimization. Being a field of mathematics, optimization deals with finding the best possible solution to a given problem. By (optimization) problem, we mean a problem we want to optimize. An optimization problem can be framed into a mathematical model, commonly called an optimization model. With this optimization model, we can apply mathematical theory and optimization algorithms to obtain a solution.

As we will explore in this master's thesis, the learning process can be formulated as an optimization problem, henceforth, an optimization model may be constructed. Some of these models have a closed-form solution, which allows for obtaining the optimal solution immediately. On the other hand, some models have to be solved iteratively, with the use of iterative methods, e.g. gradient descent. In adapting an iterative approach, we will gradually approach a better and better solution. This gradual approach of improving solutions in the optimization model is reflected in the computer program, as it gradually learns and improves. This gradual improvement of a computer program, within the ML context, is what we mean by a computer program learning.

Assuming familiarity with the fundamentals of optimization, we will explore the learning process in ML. More specifically, we will delve into the learning process within the subfield called Supervised Learning (SL). Here, we are letting the ML program learn from experience, in form of a dataset, with the goal of predicting the correct output given an input. This can include tasks such as image classification, speech recognition and medical diagnosis. Particular to the field of SL, as the ML program makes predictions, it will conceptually be aided by an all-knowing supervisor or oracle. This allows the program to realize if it is correct or wrong in its predictions, which allows for correction and improvement.

Furthermore, in SL, two distinct types of problems are considered: regression and classification. In regression problems, the goal is to predict a continuous output, often represented by a real number. For instance, one might apply regression to predict house prices based on various features describing the house, such as square footage, number of bedrooms, and location. On the other hand, in classification problems, the objective is to assign inputs to discrete categories or classes. For instance, classifying emails as spam or non-spam is a common classification task.

With an understanding of the learning process, we also want to touch on the topic of Hyperparameter Optimization (HPO). Much like an instrument, an ML algorithm has different settings that can be tuned, depending on the needs and wants of the ML practitioner. The specific setting of an algorithm is determined by the adjustable parameters

of the algorithm. We call these parameters hyperparameters. Depending on the given task, different settings to the hyperparameters of a ML algorithm may yield finalized ML programs that may differ quite in performance.

We begin in Chapter 2 by recalling concepts and results of optimization used throughout the later chapters. Thereafter, we will provide an introduction to ML in Chapter 3, which grants the context and understanding needed for the subsequent chapters. As SL involves regression and classification problems, we will explore the respective algorithms: linear regression in Chapter 4 and Support Vector Machines (SVM) in Chapter 5. With a conceptual understanding of the algorithms for each problem, in Chapter 6, we will delve into how we should tune such algorithms by inspecting their hyperparameters. We will see that a significant challenge in the hyperparameter tuning problem is time, as the training component is often time-consuming, especially when dealing with a large-scale training dataset. In Chapter 7 we conduct numerical experiments regarding the discussed SL algorithms. Our aim is to provide practical examples and investigations to complement and solidify the theoretical framework discussed in the preceding chapters.

Chapter 2

Fundamental Optimization Principles

This chapter aims to describe and formalize the optimization concepts discussed in the subsequent chapters. Our focus is on providing a review, rather than an in-depth investigation, assuming familiarity with the content.

We will begin by presenting properties and statements related to non-linear optimization. Furthermore, we consider theory from convex optimization.

2.0.1 Non-Linear Optimization

Suppose we have the optimization model given by

$$\begin{aligned} &\min f(\mathbf{x}) \\ &\text{s.t.} \\ &h_1(\mathbf{x}) = 0, \ldots, h_m(\mathbf{x}) = 0, \\ &g_1(\mathbf{x}) \leq 0, \ldots, g_r(\mathbf{x}) \leq 0, \end{aligned} \tag{2.1}$$

where f, h_i, g_j are continuously differentiable functions from $\mathbb{R}^n$ to $\mathbb{R}$. With vector notation, we write model (2.1) as

$$\begin{aligned} &\min f(\mathbf{x}) \\ &\text{s.t.} \\ &h(\mathbf{x}) = \mathbf{0}, \\ &g(\mathbf{x}) \leq \mathbf{0}, \end{aligned} \tag{2.2}$$

where $h : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^n \to \mathbb{R}^r$ are the functions

$$h = (h_1, \ldots, h_m), \quad g = (g_1, \ldots, g_r).$$

Before considering the KKT conditions, we recall the definitions in [48], which defines the Lagrangian function, regularity with respect to a feasible point $\mathbf{x}$ and the active set for a point $\mathbf{x}$.

Definition 2.1. *([48]).*

The Lagrangian function $\mathcal{L} : \mathbb{R}^{(n+m+r)} \to \mathbb{R}$ with respect to optimization model (2.1) is defined as

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\omega}, \boldsymbol{\alpha}) = f(\mathbf{x}) + \sum_{i=1}^{m} \omega_i h_i(\mathbf{x}) + \sum_{j=1}^{r} \alpha_j g_j(\mathbf{x}), \tag{2.3}$$

where $\boldsymbol{\omega} \in \mathbb{R}^m$ and $\boldsymbol{\alpha} \in \mathbb{R}^r$.

Definition 2.2. *([48]).*

A feasible vector $\mathbf{x}$ with respect to optimization model (2.1) is said to be regular if the gradients of the equality constraints and the active inequality constraints are linearly independent at $\mathbf{x}$. We also say that $\mathbf{x}$ is regular in the case where there are no equality constraints and all the inequality constraints are inactive at $\mathbf{x}$.

Definition 2.3. *([48]).* For any feasible point $\mathbf{x}$, the index set of active inequality constraints is denoted by

$$A(\mathbf{x}) = \{j \mid g_j(\mathbf{x}) = 0\}. \tag{2.4}$$

If $j \in A(\mathbf{x})$, we say that the j-th constraint is active at $\mathbf{x}$. For the contrary, we say the j-th constraint is inactive at $\mathbf{x}$.

Proposition 2.1. *([48], Part of Prop. 4.3.1 (Karush-Kuhn-Tucker Necessary Conditions)).*

Let $\mathbf{x}^*$ *be a local minimizer and assume that* $\mathbf{x}^*$ *is regular. Then there exist unique Lagrange multiplier vectors* $\boldsymbol{\omega}^* = (\omega_1^*, \ldots, \omega_m^*)$, $\boldsymbol{\alpha}^* = (\alpha_1^*, \ldots, \alpha_r^*)$, *such that*

$$\begin{aligned} D_{\mathbf{x}}\mathcal{L}(\mathbf{x}^*, \boldsymbol{\omega}^*, \boldsymbol{\alpha}^*) &= \mathbf{0}^T, \\ \alpha_j^* \geq 0, \quad j &= 1, \ldots, r, \\ \alpha_j^* = 0, \quad \forall\ j &\notin A(\mathbf{x}^*), \end{aligned}$$

where $A(\mathbf{x}^*)$ *is the active index set at the point* $\mathbf{x}^*$.

2.0.2 Convex Optimization

Before looking into properties of convex optimization, let us first review definitions and useful properties of convex functions given in [48].

Definition 2.4. *([48], Appendix B).*

Let S be a convex subset of $\mathbb{R}^n$. A function $f : S \to \mathbb{R}$ is called **convex** if

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y}), \quad \forall\ \mathbf{x}, \mathbf{y} \in S,\ \forall\ t \in [0,1]. \tag{2.5}$$

We call f **strictly convex** if the inequality in (2.5) is strict for all $\mathbf{x}, \boldsymbol{y} \in S$ with $\boldsymbol{x} \neq \boldsymbol{y}$, and all $t \in (0,1)$.

Proposition 2.2. *([48], Part of Prop. B.4).*

Let S be a convex subset of $\mathbb{R}^n$ and let $f : \mathbb{R}^n \to \mathbb{R}$ be twice continuously differentiable over $\mathbb{R}^n$. Consider the Hessian of $f(\mathbf{x})$ denotes as $\mathcal{H}_f(\mathbf{x})$.

(i) If $\mathcal{H}_f(\mathbf{x})$ is positive semidefinite for all $\mathbf{x} \in S$, then f is convex over S.

(ii) If $\mathcal{H}_f(\mathbf{x})$ is positive definite for all $\mathbf{x} \in S$, then f is strictly convex over S.

(iii) If $f(\mathbf{x}) = \mathbf{x}^T Q\mathbf{x}$, where Q is a symmetric (n,n)-matrix, then f is convex if and only if Q is positive semidefinite. Furthermore, f is strictly convex if and only if Q is positive definite.

Convex functions enjoy certain useful properties for describing a global minimizer, as well as conditions for optimality. These properties are highlighted in Proposition 2.3 and 2.4.

Proposition 2.3. *([48], Part of Prop. 1.1.2).*

If X is a convex subset of $\mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ is convex over X, then a local minimizer of f over X is also a global minimizer of f over X. If in addition f is strictly convex over X, then f has at most one global minimizer over X.

Proposition 2.4. *([48], Part of Prop. 1.1.3).*

Let X be a convex set and let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function over X. If X is open and f is continuously differentiable over X, then $Df(\mathbf{x}^*) = \mathbf{0}^T$ is a necessary and sufficient condition for a vector $\mathbf{x}^* \in X$ to be a global minimizer of f over X.

If we have the special case where an optimization model has the characteristics of a convex objective function along with linear inequality constraints, then, as stated in [48], we can construct another optimization model. This constructed model will have the same optimal value and has as optimal solutions the Lagrange multipliers of the former model. The following proposition taken from [48] encapsulates this idea.

Proposition 2.5. *([48], Part of Prop. 4.4.2).*

Consider the following. Let the primal model be defined as

$$\begin{aligned} &\min f(\mathbf{x}) \\ &s.t. \\ &\mathbf{a}_j^T \mathbf{x} \leq \mathbf{b}_j, \quad j = 1, ..., r, \\ &\mathbf{x} \in X, \end{aligned} \tag{2.6}$$

where $\mathbf{a}_j \in \mathbb{R}^n$ and $b_j \in \mathbb{R}$, $f : \mathbb{R}^n \to \mathbb{R}$ is a convex continuously differentiable function, and X is a polyhedral.

With the corresponding dual function (refer to Definition 2.1) given by

$$\mathcal{L}(\boldsymbol{\alpha}) = f(\mathbf{x}) + \sum_{j=1}^{r} \alpha_j (\mathbf{a}_j^T \mathbf{x} - b_j), \tag{2.7}$$

the dual function defined by

$$q(\boldsymbol{\alpha}) = \inf_{\mathbf{x} \in X} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}), \tag{2.8}$$

and the dual model defined as

$$\max q(\boldsymbol{\alpha}) \tag{2.9}$$

$$s.t.$$

$$\boldsymbol{\alpha} \geq 0. \tag{2.10}$$

(a) If the primal model (2.6) has an optimal solution, then the dual model (2.9) also has an optimal solution and both corresponding optimal values are equal.

(b) Let $A(\mathbf{x})$ denote the active index set, that is, $A(\mathbf{x}) = \{j | \mathbf{a}_j^T \mathbf{x} - \mathbf{b}_j = \mathbf{0}\}$. In order for $\mathbf{x}^*$ to be an optimal primal solution and $\boldsymbol{\alpha}^*$ to be an optimal dual solution, it is necessary and sufficient that $\mathbf{x}^*$ is primal feasible, $\boldsymbol{\alpha}^* \geq \mathbf{0}$, $\alpha_j^* = 0$ for all $j \notin A(\mathbf{x}^*)$, and

$$\mathbf{x}^* \in \arg\min_{\mathbf{x} \in X} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*). \tag{2.11}$$

Proposition 2.5 allows us to solve the dual optimization model (2.9) instead of the primal optimization model (2.6). We then find that the optimal solution of the dual model (2.9) corresponds to the Lagrange multipliers of the primal model (2.6). Furthermore, it guarantees that the optimal values are equal.

Chapter 3

Foundations of Machine Learning (ML)

This chapter aims to give a general introductory overview of the field of ML, along with establishing a common vocabulary and some notation. Based on this, in the following chapters we will delve into the various algorithms of Supervised Learning (SL).

We start by discussing what can be considered as the primary subfields of ML. Then we move on to the general framework the ML practitioner operates in, namely, the so-called ML pipeline. We conclude this chapter with the no-free-lunch theorems, which gives confidence in the argument of the need in exploring and inventing a large variety of ML algorithms.

3.1 The Primary Subfields of ML

As introduced in [71], ML is one of the major subfields of the much broader field called AI (see Figure 3.1). In [57], it is further explained that AI is the ability for a digital computer or computer-controlled robot to perform tasks, commonly associated with intelligent beings. AI is frequently applied to the project of developing systems endowed with the intellectual processes characteristic of humans, such as the ability to reason, discover meaning, generalize or learn from past experience.

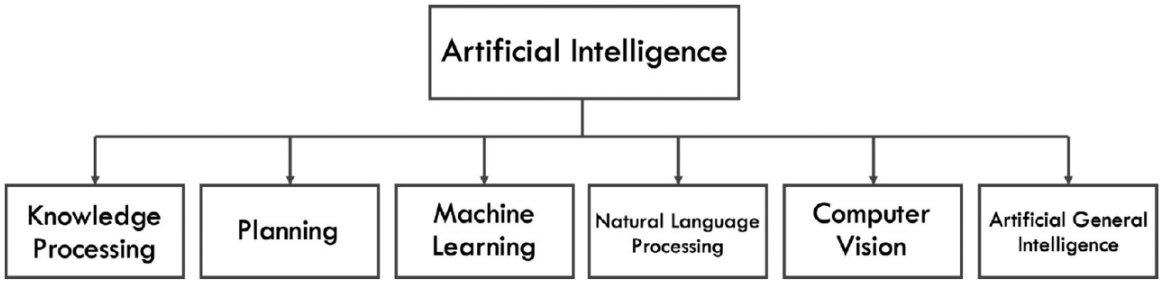

Figure 3.1: The subfields of AI.

Figure (taken from [71]).

As mentioned in [21], the goal of ML is to create computer programs that are capable of imitating the human learning experience, gradually improving over time. Furthermore, [50] shares the concept that the computer learns without being explicitly programmed. Imagine the task of teaching a computer to recognize pictures of different people. A trivial task for the human mind, but challenging to describe the process to a computer. Other tasks such as audio recognition, autonomous vehicles and chat bots share the same complications as it would be difficult to describe the task or write sufficient computer code. Instead of dealing with the cumbersome task of explicitly instructing the computer which can be time-consuming or practically impossible, ML algorithms opt to learn by themselves from data, experience or both.

The ML field is traditionally dissected into three main fields: SL, unsupervised learning and Reinforcement Learning (RL) (see Figure 3.2).

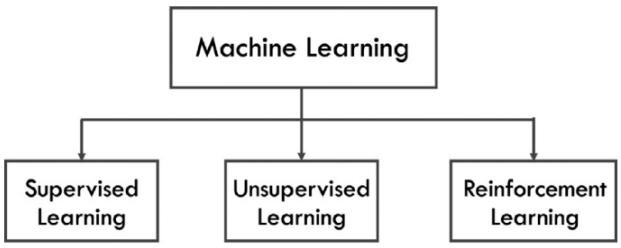

Figure 3.2: The subfields of ML.

Figure (modified and taken from [71]).

In the following, we describe these three fields in more detail.

3.1.1 Supervised Learning

Adapting the naming in [65], we have that **Supervised Learning (SL)** applies supervised algorithms to learn a function mapping

$$f : \mathbb{R}^n \to \mathbb{R} \tag{3.1}$$

where input $\mathbf{x} = (x_1, \ldots, x_n)$ is called the **independent variable** and output y is called the **dependent variable**. The domain and co-domain of this mapping may vary. As mentioned in [70], it is common for the independent and dependent variable to be from a vector space, but this is not strictly necessary as it is also common to have them as matrices or scalar values. The variability in the domain and co-domain depends on the specific SL task under consideration which will be further discussed in this section. If the dependent variable is a vector, we represent it using bold symbol $\mathbf{y}$. However, if the dependent variable is a scalar or a categorical value (see definition below) we use the non-bold symbol y.

In learning the mapping f, a SL algorithm will utilize a dataset of input-output pairs

$$\mathcal{D}_{\text{SL}} = \{(\mathbf{x}^{(i)}, y_i\}_{i=1}^{N} \tag{3.2}$$

where N is the number of $(\mathbf{x}^{(i)}, y_i)$ pairs in $\mathcal{D}_{\text{SL}}$. Since there is no standard formulation, we will use our own terminology by defining $\mathcal{D}_{\text{SL}}$ as a **dataset**. With respect to dataset $\mathcal{D}_{\text{SL}}$, we call $\mathbf{x}^{(i)}$ a **data point**, y_i a **target** and the pair $(\mathbf{x}^{(i)}, y_i)$ a **labeled data point**. As stated in [70], depending on the dataset, the data $\mathbf{x}^{(i)}$ and y_i may represent complex structure such as sentences, images, videos, sound recordings, etc. If this is the case, we have to perform data transformations to turn the data into a more manageable format for the SL algorithms e.g. vectors, matrices and scalars (see the data processing step in the SL pipeline in Section 3.2). Further influenced by the dataset, if $\mathbf{x}^{(i)}$ is a vector, its coefficients can be the representations of concrete ideas, such as height, weight, income etc. In [70], these representations making up the coefficients of the independent variable are called **features**. Just like the independent variable, the dependent variable can in principle be any type of object.

For y, we distinguish between two important cases. In adapting the naming and description in [70], if y is a **categorical variable**, then

$$y \in \mathcal{C} = \{\nu_1, \nu_2, \ldots, \nu_k\} \tag{3.3}$$

where $\mathcal{C}$ represents a finite set of elements ν_j called **labels** or **classes**. From a statistics view, every ν_j are from a finite parameter space or outcome space Ω. Examples of labels for the categorical case are the months of a year with $\mathcal{C} = \{\texttt{January}, \ldots, \texttt{December}\}$ or labels representing means of transport with $\mathcal{C} = \{\texttt{Car}, \texttt{Bike}, \texttt{Bus}\}$. When y is a categorical variable, the task is referred to as a **classification task**. If the dependent variable y take on continuous values, we call y a **continuous variable** and the task is referred to as a **regression task**. We will refer to mapping f (see Equation (3.1)) in the classification and regression task as a **decision boundary** and a **regression line** or **regression curve** respectively.

As stated in [53] and [52], the goal of a SL algorithm is to have the algorithm output, that is, the mapping, in ML terms called the **model**, have generalizing capabilities with respect to the dataset. By generalizing capabilities, we mean that the model is able to sufficiently predict on a different unseen dataset. It will be more clear what we mean by sufficient in Chapter 4 and 5. Generalization is desired for the ML models due to the prediction concept described in [44]. The following is a reworded formulation. Consider the case where after we applied a SL algorithm on a dataset, that we observe a new data point $\hat{\mathbf{x}}$ not found in the dataset, but assumed to be generated by the same unknown process that generated dataset $\mathcal{D}_{\text{SL}}$. Using the model given by the SL algorithm, we would then like the prediction $\hat{y}(\hat{\mathbf{x}})$ corresponding to $\hat{\mathbf{x}}$ to be accurate by some measure, thus making the model fit for predicting on data not yet seen. As [44] highlights, in SL, our interest is in describing $\hat{y}$ conditioned on knowing $\hat{\mathbf{x}}$. From a probabilistic modelling perspective, we are therefore concerned primarily with the conditional distribution $P(\hat{y}|\hat{\mathbf{x}}, \mathcal{D}_{\text{SL}})$, which models the probability of $\hat{y}$ being the output, given the test data point $\hat{\mathbf{x}}$ and the dataset $\mathcal{D}_{\text{SL}}$.

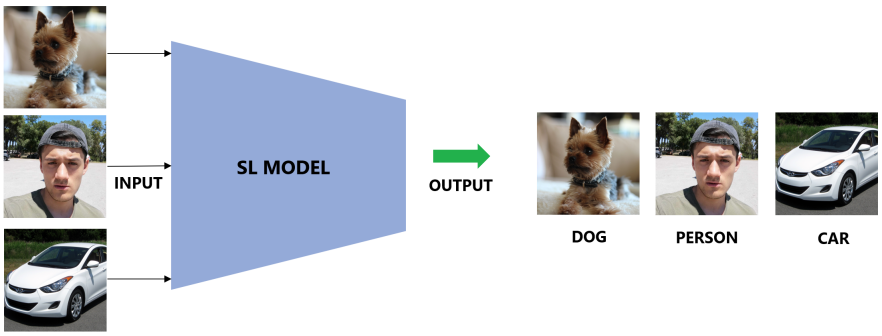

Figure 3.3: Illustrated is the conceptual ideas of SL regarding the task of image classification. Given an input image $\mathbf{x}^{(i)}$ (depicted on the left side), the SL model uses mapping f to predict the corresponding label y_i (depicted on the right side).

Figure (modified and taken from [66] and [81]).

From a visual standpoint, in the regression task, our objective is to identify a mapping that describes the relationship between the independent and dependent variables. On the left-hand side of Figure 3.4, the blue points represent labeled data points $(\mathbf{x}^{(i)}, y_i)$ from a dataset $\mathcal{D}_{\text{SL}}$. The black line depicts the mapping. In the context of the classification task, our goal is to establish a mapping that effectively separates the data points $\mathbf{x}^{(i)}$ based on their corresponding labels y_i. Shown on the right-hand side of Figure 3.4, we observe blue and red points, each denoting data points $\mathbf{x}^{(i)}$ associated with two distinct labels. The mapping, represented by the black line, effectively separates the points according to their color, indicating the differentiation of data points $\mathbf{x}^{(i)}$ according to their corresponding labels y_i.

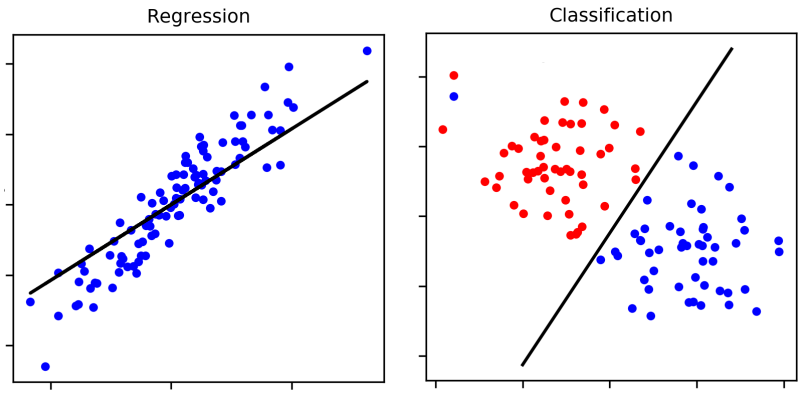

Figure 3.4: In the regression task (left), we are concerned with finding a mapping that describes the relation between the independent and dependent variable. The classification task (right) can be seen as finding a mapping that separates the labels.

Figure (taken form [42]).

We will now consider the following example which illustrates that there are multiple ways to represent the independent and dependent variable given a dataset $\mathcal{D}_{\text{SL}}$.

Example 3.1. ***The MNIST Dataset***

Figure 3.5: Images from the MNIST dataset.

Figure (taken from [3]).

As mentioned in [3], widely considered as the 'hello world' dataset of ML, the MNIST dataset consists of a total of 70′000 grayscale images of numbers from zero to nine. Each image is 28×28 pixels in size. Each image in the dataset is labeled with its corresponding number. The usual SL task is then to apply algorithms to the MNIST dataset with the goal of obtaining a model that can adequately classify an image to its corresponding number it is depicting. We will explore how the independent and dependent variables can be represented. The following representations discussed are widely known.

The input images, being of size 28×28, result in a total of 784 features describing an image. As grayscale images have a single color channel, each pixel has a value ranging from 0 to 255, where 0 represents a completely black pixel, 255 a completely white pixel, and values in-between variations of gray.
One way to represent the images is to concatenate each grayscale value corresponding to a pixel of an image into a vector with 784 entries, making the i-th image $\mathbf{x}^{(i)}$ a 784-dimensional vector. Alternatively, another way to represent the independent variable is not to concatenate grayscale-pixel values into a vector but to store them in a 28×28 matrix, leaving the i-th image $\mathbf{x}^{(i)}$ as a 28×28 matrix.
Considering the dependent variable y, the most natural way to represent the depicting number corresponding to an image is as a value from $\mathcal{C} = \{0, ..., 9\}$. With this, we see that y is a categorical variable. Another common representation is using one-hot encodings, described in [51]. For the i-th image, we would then have a vector y_i with the number of entries corresponding to the number of possible

number depictions, in our case, a 10-dimensional vector. A vector y_i will then consist of all zeros, except for a value of 1 at the row corresponding to the number depicted by $\mathbf{x}^{(i)}$.

3.1.2 Unsupervised Learning

Unsupervised learning is the part of ML that utilizes algorithms for various tasks such as analysis and clustering of unlabeled datasets, as stated in [20]. Being an unlabeled dataset implies that we consider the SL dataset structure given by (3.2), only that there is no corresponding output y_i for each input $\mathbf{x}^{(i)}$. As defined in [70], we consider the unlabeled dataset given by

$$\mathcal{D}_{\text{UL}} = \{\mathbf{x}^{(i)}\}_{i=1}^{N} \tag{3.4}$$

where N is the number of elements in $\mathcal{D}_{\text{UL}}$. For a description of $\mathbf{x}^{(i)}$, see Section 3.1.1.

According to [22], unsupervised learning allows models to act on the given dataset without any supervision. There is no training dataset nor supervisor that are telling the models what is correct and what is wrong. In the words of [70]:

> "unlike SL, we are not told what the desired output is for each input".

Further stated from [70]:

> "The goal [of unsupervised learning] is to find 'interesting patterns' in the data".

The goal of unsupervised learning is not to learn a mapping to predict an appropriate output given an input as seen in the SL case, instead the focus is on discovering 'interesting patterns' in the data. This search for uncovering structures in the data is referred to as **knowledge discovery**, as named in [70].

In [20], clustering and dimensionality reduction are considered as two important applications of unsupervised learning. We will now consider these two methods.

Clustering is the task of sort data into groups. We then want similar data points to be in similar groups (clusters). The similarity is determined by the rule of the clustering algorithm, which there are multiple of, as seen in Figure 3.6.

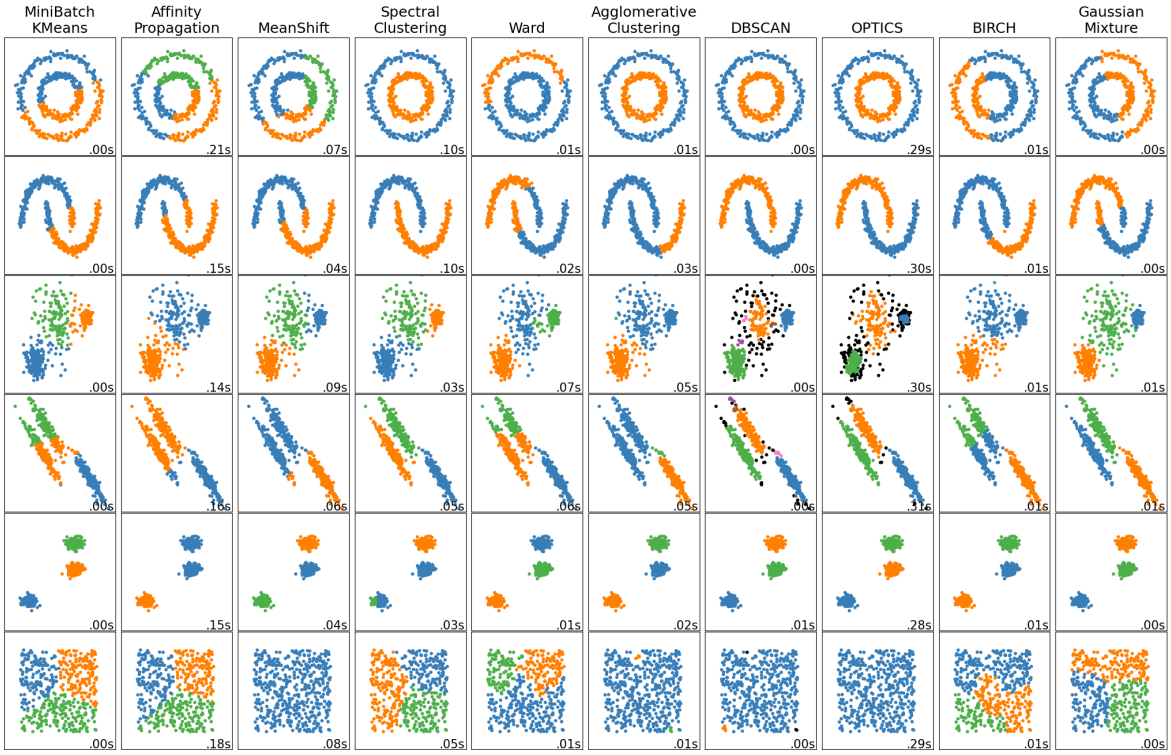

Figure 3.6: The rows represent different datasets and the columns are different clustering algorithms. This illustrates that the clustering algorithms have different use cases depending on the structure of the data. The last dataset consists of points randomly distributed with no meaningful structure.

Figure (taken from [23]).

Described in [20], **dimensionality reduction** is a useful tool to reduce the dimension of high-dimensional data while preserving the integrity of the dataset as much as possible. To explain the dataset's integrity, we consider **Principal Component Analysis (PCA)**, which is an often sought-out dimensionality reduction algorithm. In [20], we have the following explanation of the Principal Component Analysis (PCA) procedure which we will now describe. Assuming our data points $\mathbf{x}^{(i)}$ reside in a coordinate system, we start by performing a change of basis to these data points. In the new coordinate system, the first principal component is the direction that maximizes the variance of the data points, the second principal component is the direction that secondly maximizes the variance, and so on. Through the right change of basis, these principal components will be orthogonal to each other (see Figure 3.7). By starting to discard the principal components with the least variance describing the data points, we essentially reduce the

dimensions and compress the data. This is done while preserving the integrity of the data by removing the least important dimensions with respect to the variance of the data.

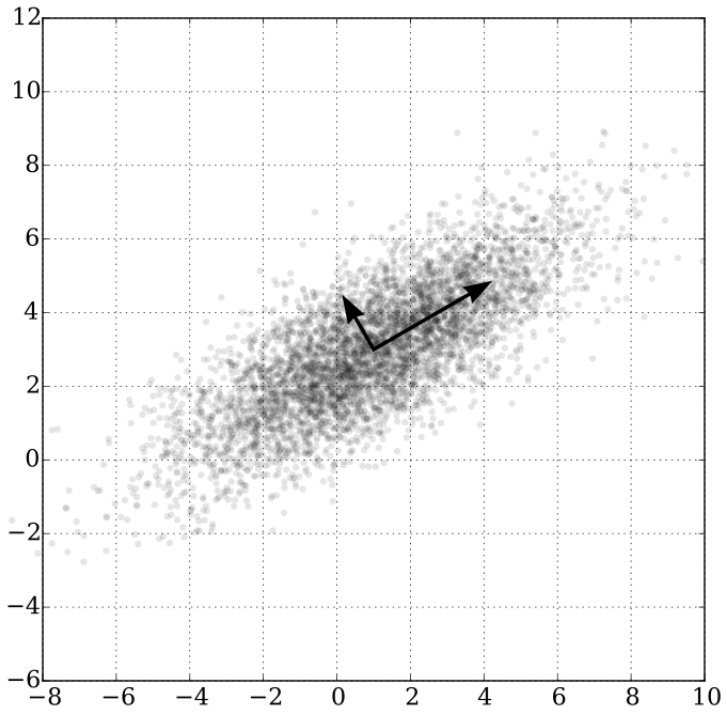

Figure 3.7: A change of basis has been performed and the two principal components are indicated by the black arrows. The greater arrow indicates the first principal component, as it captures the most variance of the data.

Figure (taken from [7]).

Listed in [6] are reasons for considering dimensionality reduction. Notable is the removal of redundant features. Decreasing the number of features can enhance computational time by working with a more compressed dataset. Furthermore, a reduction in features opens up the possibility for visualization of higher dimensional data in dimensions such as two or three dimensions.

3.1.3 Reinforcement Learning (RL)

The third subfield of ML is **Reinforcement Learning (RL)**. In [34], they characterize RL as the science of decision making, [33] adds:

> "RL is a feedback-based ML technique in which an agent learns to behave in an environment by performing actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback."

The term **agent** refers to the decision-making entity of interest in RL. An RL algorithm serves as the mechanism that enables the agent to learn from its experiences. The agent is the output of the RL algorithm, representing the accumulated results of the learning process. The agent behaves in a way as to maximize its positive feedback, known as **reward** in RL terminology.

Given an input **state**, by state we mean an object describing the current state the environment is in, the trained agent will output the action it deems optimal based on its training experiences and past rewards. The **environment**, encompassing everything beyond the agent's control, responds by providing a new state and a reward based on the agent's action.

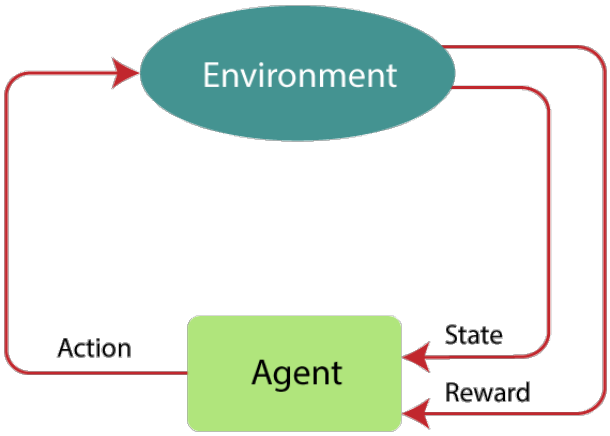

Figure 3.8: The iterative process between the agent and the environment. At iteration step t, the agent receives a state S_t and a reward R_t. The agent then decides an action A_t and the environment responds by giving a new state S_{t+1} and a reward R_{t+1}.

Figure (taken from [33]).

RL, therefore, considers itself with a back-and-forth play between the agent and the environment. In [49] the contrast between RL and SL is highlighted. Unlike a SL algorithm, a RL algorithm has no available labeled dataset with optimal outputs to guide its learning process. The RL algorithm must discover the optimal outputs through trial and error by actively engaging with the environment.

To make the agent-environment relationship more clear, we will consider the RL model AlphaZero. As described in [35], AlphaZero is a computer program designed using RL to play the board games chess, Go and Shogi. Taking chess as an example, the agent takes as input a board configuration and outputs the move to make. In this context, the opponent acts as the environment. Analogous conclusions as stated by [49] when applying RL to the board game backgammon, are also applicable when applying RL in the game of chess. A single chess game involves a dozens of moves, and it is only at the end of the game that the reward, in the form of a win or a loss, is obtained. The reward must then be appropriately propagated to all of the moves that led to the game's conclusion, even though some moves will have been good moves, while others less so.

In this master's thesis, the description of SL has been emphasized compared to unsupervised learning and RL. This emphasis is due to our focus will be on optimization within the context of SL algorithms. In the next section, we will look into concepts of the ML pipeline. The ML pipeline represents the standard framework that a ML practitioner operates within. We will consider steps of the ML pipeline, starting with data processing and extending all the way to the final step involving the finalization of a ML model.

3.2 The ML Pipeline

Important for the process of creating ML models is the **ML pipeline**. Rephrasing the description in [25], the ML pipeline is described as a way to automate the workflow of the ML practitioner. Note that the steps of the workflow may be defined differently depending on the author. Thus, we have divided the workflow into four steps, data processing, model training, model evaluation, and model deployment. For the model training and model evaluation steps we will adapt a more intuitive explanation of the processes in order to understand the relation to the ML pipeline as a whole. For a more in-depth analysis of these two steps, refer to Chapter 6. Further mentioned in [25] is that the ML pipeline consists of having these steps automated by for example code. The ML pipeline and its corresponding workflow steps can be seen in Figure 3.9.

An imaginative analogy in [24] describes the ML pipeline like an assembly line in a manufacturing plant. The product goes through various stages along the assembly line, and at the end, we have the final product.

There may be differences in the workflow steps of the ML pipeline depending on whether we are considering the SL, RL or the unsupervised learning setting. In our

explanation, we will therefore consider the workflow steps of the ML pipeline in the case of SL, we will call this the **SL pipeline**.

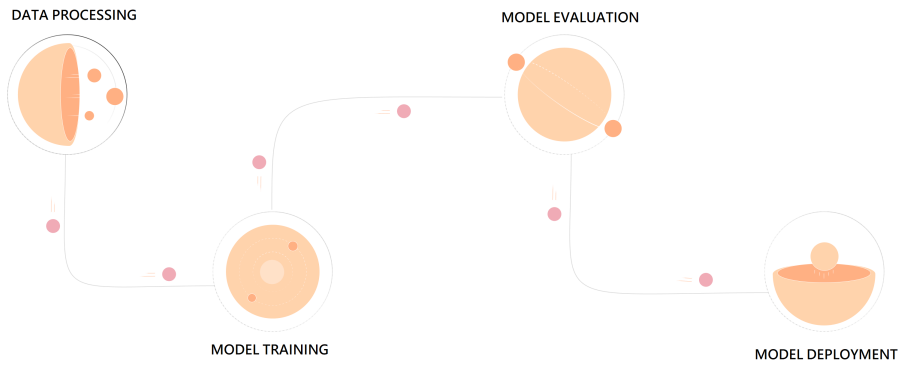

Figure 3.9: An overview of the ML pipeline.

Figure (modified and taken from [25]).

Data processing consists of preparing the data for the SL algorithms. Assume we have some gathered data (which can be seen as a data extraction step in a more comprehensive pipeline description) from some data organization or database which we want to use for producing SL models. Before using this data, we need to perform data processing, that is, data cleaning and transformations, which we will now explain.

In [64] they consider various data cleaning processes. The removal of corrupted data may be necessary if we encounter data corruption, which can be due to reasons such as hardware failure, software failure, file format incompatibility, data transmission errors, human errors, etc. Handling missing values is necessary when there are data points that have only been partially recorded or when parts of a data point have been corrupted. This is due to many SL algorithms cannot handle missing values. There is a wide variety of techniques; the most straightforward ones simply remove the data points with missing values or calculate and insert the mean or the median of the missing value with respect to the complete data points. For further data cleaning processes, we refer to [8].

The process of data transformation as seen in [2] is about transforming data into a format that is readable for SL models. The transformation method called **vectorization** is the process of transforming the contents from the dataset $\mathcal{D}_{\text{SL}}$ into vectors or matrices which are more suitable for the SL algorithms, compared to data in video or image formats. Another transformation method considers the encoding of non-numerical data such as going from categorical data or text data into numerical data. There are also more optional steps; [20] mentions performing dimensionality reduction (refer to the method of

PCA in Subsection 3.1.2) and [13] considers standardization of the data. Standardization is the process of transforming either $\mathbf{x}^{(i)}$, y_i, or both, within the dataset $\mathcal{D}_{\text{SL}}$. Assuming the data points $\mathbf{x}^{(i)}$ are vectors, standardization of the data points consists of first calculating the sample mean and sample variance with respect to all data points in $\mathcal{D}_{\text{SL}}$. Thereafter, for each data point, we subtract the calculated sample mean and divide by the sample variance. As [13] states, our data points are now transformed such that there is a mean of 0 and standard deviation of 1. Both processes of dimensionality reduction and standardization have the goal of improving the model performance.

We now consider the **model training** step. During model training, it is common practice to not let the model train on the whole dataset $\mathcal{D}_{\text{SL}}$, but a subset called $\mathcal{D}_{\text{SL-train}}$. Well-known, as seen in [12] is the use of **training-validation-test split** which splits the whole dataset $\mathcal{D}_{\text{SL}}$ into a training dataset $\mathcal{D}_{\text{SL-train}}$, a validation dataset $\mathcal{D}_{\text{SL-val}}$, and a test dataset $\mathcal{D}_{\text{SL-test}}$, with the properties that the three sets are mutually disjoint and the union of the three datasets is equal to $\mathcal{D}_{\text{SL}}$. Such a splitting is done to perform both model training and evaluation, where the two latter sets, $\mathcal{D}_{\text{SL-val}}$ and $\mathcal{D}_{\text{SL-test}}$ are used in the model evaluation step of the SL pipeline.

As described in [26], the objective of **model training** is to learn the mapping f (See Equation (3.1)) given dataset $\mathcal{D}_{\text{SL-train}}$. This process can be framed as an optimization model where we optimize the **training parameters** $\mathbf{w}$ corresponding to mapping f. Depending on the chosen SL algorithm, the mapping f, or equivalently, the model output $\hat{y}(\mathbf{w}, \mathbf{x})$ may differ e.g. for linear regression we have $\hat{y}(\mathbf{w}, \mathbf{x}) = \mathbf{w}^T\mathbf{x}$ (see Chapter 4).

We begin by considering the objective function, commonly called the **loss function** in ML. As highlighted in [72], the loss function is usually constructed in a way such that it in some aspect captures how well the model fits the data, by comparing the predicted model output $\hat{y}_i(\mathbf{w}, \mathbf{x}^{(i)})$ with the true value y_i. A loss function for model training is usually constructed such that it considers each labeled data point $(\mathbf{x}^{(i)}, y_i)$ from $\mathcal{D}_{\text{SL-train}}$. To mathematically convey the description in [72], let h_{train} be a given comparison function for model training that takes the predicted value $\hat{y}_i(\mathbf{w}, \mathbf{x}^{(i)})$ and the true value y_i, e.g. for the regression task it is common to see

$$h_{\text{train}}(\hat{y}_i(\mathbf{w}, \mathbf{x}^{(i)}), y_i) = (\hat{y}_i(\mathbf{w}, \mathbf{x}^{(i)}) - y_i)^2. \tag{3.5}$$

The general form of L_{train} is then given by

$$L_{\text{train}}(\mathbf{w}) = \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{SL-train}}} h_{\text{train}}(\hat{y}(\mathbf{w}, \mathbf{x}), y). \tag{3.6}$$

Model training refers to the process of solving the following optimization model:

$$\min_{\mathbf{w}\in\mathbb{R}^n} L_{\text{train}}(\mathbf{w}). \tag{3.7}$$

Models do not only possess training parameters, but also **hyperparameters**. Hyperparameters are different from training parameters in that they are fixed parameters chosen before initiating model training. They can be seen as configurations of the settings of a SL algorithm, thus giving rise to models possessing different **hyperparameter configurations**. Hyperparameters can be real-valued, discrete or categorical values depending on the SL algorithm, [11] presents hyperparameter examples such as kernel functions and regularization-parameters which will become more familiar in Chapter 4 and 5. Models with different hyperparameter configurations, will in general perform differently, which is why the next SL pipeline stage is concerned with selecting the best model among trained models with different hyperparameter configurations.

Model evaluation is performed after model training and can also be framed as an optimization model. During model training, it is common to train multiple models, using different algorithms, and different hyperparameter configurations. Say $M = \{1 \dots, k\}$ is the set containing the indices of k trained models with m_i being model i and $\mathbf{w}^{(i)}$ being the corresponding training parameters after model training. In the model evaluation step we want to find the models in M that are performing best with respect to a validation loss function $L_{\text{val}}(\mathbf{w})$ and the validation dataset $\mathcal{D}_{\text{SL-val}}$. The validation loss function $L_{\text{val}}(\mathbf{w})$ and the corresponding validation comparison function h_{val} is conceptually the same as $L_{\text{train}}(\mathbf{w})$ and h_{train} respectively, though it is not uncommon to have h_{val} be defined differently than h_{train}. We have

$$L_{\text{val}}(\mathbf{w}) = \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{SL-val}}} h_{\text{val}}(\hat{y}(\mathbf{w},\mathbf{x}), y). \tag{3.8}$$

The optimization model for model evaluation is defined as

$$\min_{i\in M} L_{\text{val}}(\mathbf{w}^{(i)}). \tag{3.9}$$

Assume index i^* is the optimal solution to optimization model (3.9) with model m_{i^*} being the corresponding model. As described in [12], we can get an unbiased estimate on how well the model is doing on unseen data by considering $\mathcal{D}_{\text{SL-test}}$. The process of obtaining the estimate for m_{i^*} is done by considering L_{test} and h_{test}. These can be

constructed with a similar approach to that of L_{val} and h_{val}. If no anomalies present themselves when reviewing the estimate for model m_{i^*}, we select this as the final model.

Example 3.2. ***A Desert Analogy on Hyperparameters***

Imagine having the task of driving through a sandy desert and available we have different vehicle types to choose from along with different vehicle configurations corresponding to the vehicle types. In this example, driving through the desert represents the SL dataset or problem class. The choice of vehicle corresponds to selecting a SL algorithm that suits the given problem. Just as different vehicle types have a varying capability in handling a sandy terrain, different SL algorithms have their distinct strengths and weaknesses when applied to different datasets and problem classes.

In this example, the hyperparameters represent the different configurations available for the driver to perform on a vehicle. Adjusting vehicle traits such as engine power, gross vehicle weight, suspensions, tire type and tire pressure will result in a vehicle which will likely perform differently compared to a vehicle with a different configuration.

Just as the driver may experiment with different vehicle configurations to find the best working vehicle setup, a ML practitioner adjusts the hyperparameters to find the optimal configuration for their chosen algorithm. The practitioner will in general try a variety of different algorithms along with different hyperparameter configurations in order to identify the optimal model.

Model deployment is the final stage of the SL pipeline. Assume i^* is the obtained solution from (3.9) and that the corresponding model m_{i^*} shows no anomalies on the test. Within the model deployment step, we wish to put model m_{i^*} into production. This model will then be available for users, developers or systems such that they can utilize the model to solve or as part of solving their respective tasks, as described in [63].

The process of model deployment can be a challenging task, but it concerns itself more with business and practical difficulties rather than dealing with theoretical and mathematical aspects and will therefore not be discussed further.

3.3 The No-Free-Lunch Theorems

An important concept to keep in mind when we start to consider optimization ML algorithms in the following chapters are the **no-free-lunch theorems** of Wolpert and Machready from their paper, *No Free Lunch Theorems for Optimization* [89].

The following consequential description of the no-free-lunch theorems can be found in [67].

> "The NFL [no-free-lunch theorems] stated that within certain constraints, over the space of all possible problems, every optimization technique will perform as well as every other one on average (including Random Search [see chapter (6)]. That is, if there exists a set of problems P for which technique A beats technique B by a certain amount, there also exists an equal-sized set of problems P^* for which the opposite is true."

As we have seen in the previous section when describing the ML pipeline, the model training step consists in solving the underlying optimization model (3.7).

Following [54], we conclude that the no-free-lunch theorems will apply not only to SL algorithms but also to ML algorithms, since optimization is the core of the learning process.

In taking a pessimistic perspective on the statement from [67], it suggests that all algorithms are equally bad, with no universal algorithm suitable for all types of problems. However, this is not the conclusion we should draw from the no-free-lunch theorems. As [67] continues:

> "this is of considerable theoretical interest but, I think, of limited practical value, because the space of all possible problems likely includes many extremely unusual and pathological problems which are rarely if ever seen in practice."

Problems encountered in practical situations often exhibit structures that can be exploited. As stated in [59], if we have information about the type of problem we are dealing with and can incorporate it into our algorithm as assumptions, it now has an advantage over random guessing.

The implications of the no-free-lunch theorems for the field of ML are concluded in the following quote from [70]:

> "as a consequence of the no-free-lunch theorem, we need to develop many different types of models [ML algorithms], to cover the wide variety of data that occurs in the real world."

With this, we recognize the need for a variety of algorithms, each with its own strengths and weaknesses. It's crucial to pay attention to the assumptions made when considering different ML algorithms. These assumptions are often chosen to better solve specific types of problems and form the backbone of what makes the algorithms come to realization and perform as they do.

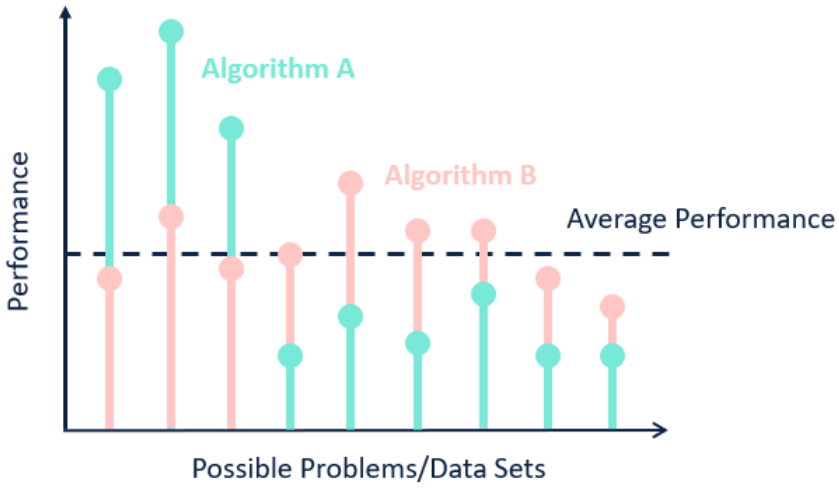

Figure 3.10: Algorithm A outperforms algorithm B on the first three problems, however, when considering the entire problem space, their performance averages out and we cannot conclude that one is better than the other given the entire problem space.

Figure (taken from [83]).

Chapter 4

Optimization In Linear Regression

In the field of mathematical statistics, [91] presents **linear regression** as one of the oldest subjects, tracing its roots back approximately two centuries. Further description by [91] includes the broader context of **regression**, describing it as a statistical method for investigating the relationships between dependent and independent variables. Specifically, in the case of linear regression, we make the assumption that the connection between the dependent variable and independent variable can be described by an underlying unknown function. While linear regression is versatile enough to handle multiple dependent variables, our discussion here focuses on the simplicity of a single dependent variable. Thus, when referring to the established notation in Subsection 3.1.1, we have dependent variable $y \in \mathbb{R}$ and independent variable $\mathbf{x} \in \mathbb{R}^n$. The underlying unknown function establishes the relationship given by

$$y = f(\mathbf{x}). \tag{4.1}$$

Furthermore, with linear regression we assume that function $f(\mathbf{x})$ is linear, yielding the expression

$$\mathbf{y} = \mathbf{w}^T\mathbf{x} \tag{4.2}$$

with vector $\mathbf{w} \in \mathbb{R}^n$ being the training parameters (for definition, refer to Section 3.2) with coefficients yet to be determined.

Recalling the general SL dataset (3.2), we consider the dataset given by

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y_i\}_{i=1}^N, \tag{4.3}$$

with data point $\mathbf{x}^{(i)} \in \mathbb{R}^n$ and target $y_i \in \mathbb{R}$.

As seen in [36], linear regression is applied to the regression task of ML. The objective is to estimate the coefficients of the linear function (4.2), which can equivalently be understood as to determine a straight line or surface that minimizes the distance between predictions and actual output values. The estimation of the coefficients is dependent on the estimation method applied.

In this chapter we will begin by introducing the general framework of linear regression. With the given framework, we will delve into two established estimation methods, Maximum Likelihood Estimation (MLE) and Maximum a Posteriori (MAP) For a description of kernels and the technique of feature mapping, see Section 5.4.

From [70], we draw inspiration from the following quote:

> "Linear regression is the "work horse" of statistics and (supervised) machine learning. When augmented with kernels or other forms of basis function expansion [feature mapping], it can model also non-linear relationships."

4.1 General Framework

Assume that the data points in $\mathcal{D}$ are Independent and Identically Distributed (i.i.d). That is, we assume that each data point $\mathbf{x}^{(i)}$ has been drawn from the same probability distribution along with assuming that $\mathbf{x}^{(i)}$ are mutually independent for $i = 1, \ldots, N$. We will also make use of matrix notation

$$\mathcal{D} = (\mathbf{X}, \mathbf{Y}) \tag{4.4}$$

with vector $\mathbf{Y} = (y_1, \ldots, y_N)$ and matrix $\mathbf{X} \in \mathbb{R}^{N \times n}$ where row i corresponds to $(\mathbf{x}^{(i)})^T$.

As stated in the introduction, when forming the connection between the independent and dependent variables, we assume the existence of an unknown underlying linear function f, which establishes Equation (4.2). Building on this idea, [85] describes the additional noise assumption (for a description of noisy data, see Section 5.3.1). With this, we assume that each labeled data point $(\mathbf{x}^{(i)}, y_i)$ in $\mathcal{D}$ has been perturbed by $\epsilon_i \in \mathbb{R}$

to some degree. We assume that the perturbation has a Gaussian distribution, that is, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. As [85] highlights, this implies that for every target y_i we have

$$y_i \sim \mathcal{N}(\mathbf{w}^T \mathbf{x}^{(i)}, \sigma^2). \tag{4.5}$$

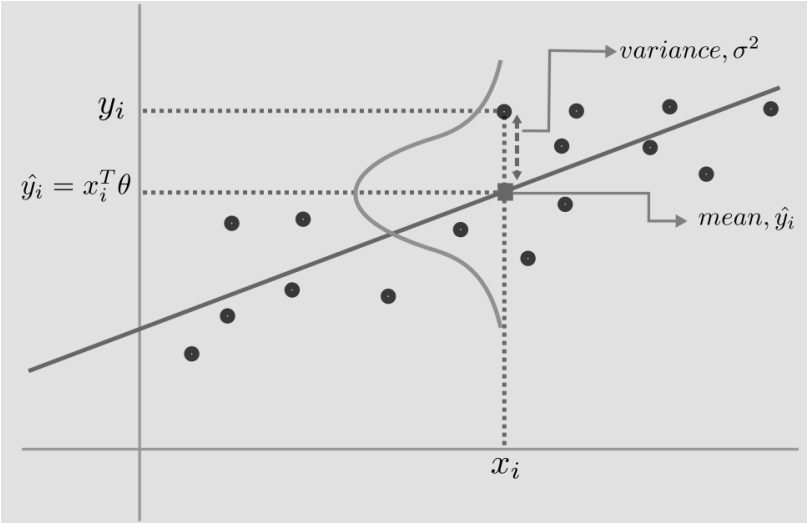

Figure 4.1: Shown is the labeled data point $(\mathbf{x}^{(i)}, y_i)$, along with the corresponding prediction $\hat{y}_i$. Within the established framework, $\hat{y}_i$ models the true underlying function f, but due to noise expressed as perturbation, the true value y_i is offset by some ϵ, and therefore, $\hat{y}_i$ does not perfectly align with the true value y_i.

Figure (taken from [74]).

There is also value in having a visual understanding of linear regression, as described in [85]. Within our framework, linear regression can be intuitively understood as finding the line, or surface, that best fits the dataset $\mathcal{D}$, according to some chosen estimation method. This idea is better explored by considering an example. In Figure 4.2, the red lines indicate inferior models while the black line indicates a better approximation for describing the underlying unknown function f, using some given estimation method. Our focus is on determining the coefficients that define the best model based on the chosen estimation method.

Assuming the relationship between the dependent and independent variables is described by the linear function f, as seen in Equation (4.2), we will have a line or surface constrained to pass through the origin due to $f(\mathbf{0}) = \mathbf{w}^T \mathbf{0}$. However, as stated in [70], we can liberate the line or surface by augmenting both $\mathbf{w}$ and each data point $\mathbf{x}^{(i)}$ by considering the transformation denoted by $\phi : \mathbb{R}^n \to \mathbb{R}^{n+1}$. For a given data point

$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)})^T$, we have the transformation

$$\phi(\mathbf{x}^{(i)}) = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, 1)^T.$$

By additionally adding a row to w, we have

$$\mathbf{w} = (w_1, w_2, \ldots, w_n, b)^T,$$

where b is simply yet another coefficient to be determined. Performing such a transformation liberates the model from having to pass through the origin, providing greater flexibility in capturing the underlying patterns in the data. This is an example of the technique of feature mapping which is further discussed in Section 5.4.

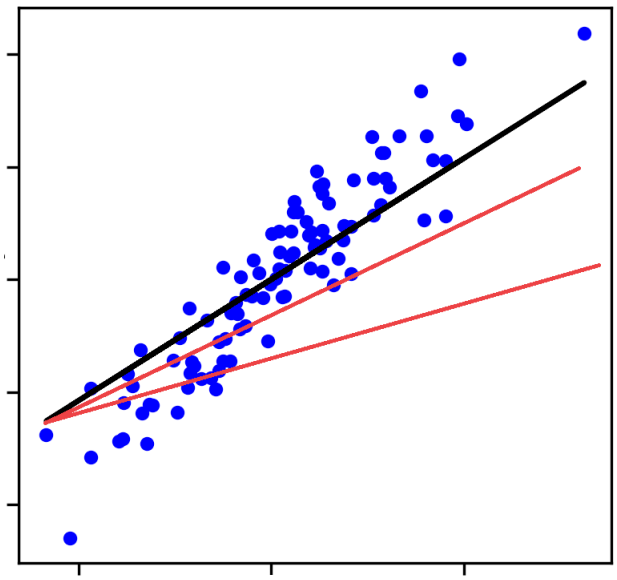

Figure 4.2: Shown are blue dots that represent data points. We have the independent variable along the horizontal axis, and the dependent variable along the vertical axis. Given some estimation method, the black line illustrates the better model while the red lines depicts inferior models.

Figure (modified and taken from [42]).

Given our framework, we are now ready to start introducing estimation methods, namely, Maximum Likelihood Estimation (MLE) and Maximum a Posteriori (MAP). These will be used to try to estimate the coefficients $\mathbf{w}$ belonging to the assumed true underlying unknown function f. With a sufficient estimation of $\mathbf{w}$, we will have succeeded

in describing the connection between the dependent and independent variables. As we will see, at the core of both these estimation methods is the construction and solving of an optimization model.

4.2 Maximum Likelihood Estimation (MLE) for Linear Regression

4.2.1 Maximum Likelihood Estimation (MLE)

From [16], we have the descriptive quote:

> "In statistics, maximum likelihood estimation (MLE) is a method of estimating the parameters of an assumed probability distribution, given some observed data. This is achieved by maximizing a likelihood function so that, ... the observed data is most probable. The point in the parameter space that maximizes the likelihood function is called the maximum likelihood estimate."

Definition 4.1 formalizes the ideas outlined in the previous quote. It consists of the likelihood function definition in [73] and the likelihood and MLE definitions in [16].

Definition 4.1. *([73], [16]).*

Let $X_1, X_2, \ldots, X_n$ be a random sample from a distribution with parameter $\boldsymbol{\theta}$ from parameter space Ω. Suppose that we have observed $X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n$. If X_i for $i = 1, \ldots, n$ are continuous, then the **likelihood function** is defined as

$$\Theta(x_1, x_2, \ldots, x_n|\boldsymbol{\theta}) = F(x_1, x_2 \ldots, x_n|\boldsymbol{\theta}), \tag{4.6}$$

where F is the joint Probability Density Function (pdf) of $X_1, X_2, \ldots, X_n$. Given $\hat{\boldsymbol{\theta}} \in \Omega$, we say that $\Theta(x_1, x_2, \ldots, x_n|\hat{\boldsymbol{\theta}})$ is the **likelihood** of $\mathbf{x}$ given $\hat{\boldsymbol{\theta}}$.

In Definition 4.1, we want to make it clear that the likelihood function (4.6) varies depending on parameter $\boldsymbol{\theta}$, and not on the sampled data $x_1, \ldots, x_n$. Note that each random variable X_i has taken the observed value of x_i for $i = 1, \ldots, n$. As the observed values do not change, they remain fixed. Moreover, $\boldsymbol{\theta}$ varies because, as stated in Definition 4.1, it

represents the parameter corresponding to the distribution random variables $X_1, \ldots, X_n$ are sampled from. As the quote at the start of this subsection describes, we are interested in estimating the parameter $\boldsymbol{\theta}$ that maximizes the likelihood of observing the data $x_1, \ldots, x_n$. Assuming $\boldsymbol{\theta}^*$ is the parameter that maximizes the likelihood function implies that, given the observed data $x_1, \ldots, x_n$, the distribution from which these are sampled is most likely to have the parameter $\boldsymbol{\theta}^*$. The method of MLE can be expressed as the following maximization model,

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta} \in \Omega} \Theta(x_1, x_2, \ldots, x_n | \boldsymbol{\theta}). \tag{4.7}$$

4.2.2 Optimization Model

We will now apply MLE to the linear regression framework discussed in Section 4.1. From the general approach to MLE, seen in (4.7), we have the unconstrained optimization problem given by

$$\mathbf{w}^* = \arg\max_{\mathbf{w} \in \mathbb{R}^n} P(\mathcal{D}|\mathbf{w}). \tag{4.8}$$

Recall from (4.5) that $y_i \sim \mathcal{N}(\mathbf{w}^T \mathbf{x}^{(i)}, \sigma^2)$. With respect to Definition 4.1, we have that $\mathbf{w}$ acts as parameters and each target y_i from $\mathcal{D}$ acts as the observed random sample from distribution $\mathcal{N}(\mathbf{w}^T \mathbf{x}, \sigma^2)$. Optimal parameters $\mathbf{w}^*$ represents the parameter values that maximizes the likelihood of observing dataset $\mathcal{D}$.

We will now follow the derivation outlined in [85]. Expanding the objective function in (4.8) we get

$$\arg\max_{\mathbf{w} \in \mathbb{R}^n} P(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}, y_1, \ldots, y_N | \mathbf{w}). \tag{4.9}$$

By the chain rule of probability along with the i.i.d assumption, we can extract the probability of y_1, conditioned on $\mathbf{x}_1$ and $\mathbf{w}$. From (4.9) we have

$$\begin{aligned} &\arg\max_{\mathbf{w} \in \mathbb{R}^n} P(y_1 | \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}, y_2, \ldots, y_N, \mathbf{w}) P(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}, y_2, \ldots, y_N | \mathbf{w}) \\ &= \arg\max_{\mathbf{w} \in \mathbb{R}^n} P(y_1 | \mathbf{x}^{(1)}, \mathbf{w}) P(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}, y_2, \ldots, y_N | \mathbf{w}). \end{aligned} \tag{4.10}$$

Moreover, by performing an identical unrolling procedure to (4.10) for $y_2, \ldots, y_n$, we get

$$\begin{aligned}
&\arg\max_{\mathbf{w}\in\mathbb{R}^n} \prod_{i=1}^{N} P(y_i|\mathbf{x}^{(i)}, \mathbf{w}) P(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}|\mathbf{w}) \\
&= \arg\min_{\mathbf{w}\in\mathbb{R}^n} \Big(-\sum_{i=1}^{N} \log P(y_i|\mathbf{x}^{(i)}, \mathbf{w}) \Big),
\end{aligned} \tag{4.11}$$

where the latter follows from the rules of the logarithm, minimizing instead of maximizing and discarding $P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}|\mathbf{w})$, since $\mathbf{x}^{(i)}$ is considered observed, and therefore independent of the training parameters $\mathbf{w}$.

Continuing, we need an expression for substituting $P(y_i|\mathbf{x}^{(i)}, \mathbf{w})$. With y_i for $i = 1, \ldots, N$ being drawn from a Gaussian distribution, by (4.5), the pdf for y_i for $i = 1, \ldots, N$ is thus given by

$$P(\eta|\mathbf{x}, \mathbf{w}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\eta - \mathbf{w}^T\mathbf{x}}{\sigma}\right)^2}. \tag{4.12}$$

Further simplification of (4.11) yields

$$\begin{aligned}
&\arg\min_{\mathbf{w}\in\mathbb{R}^n} \Big(-\sum_{i=1}^{N} \log(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{y_i - \mathbf{w}^T\mathbf{x}^{(i)}}{\sigma})^2}) \Big) && \text{from substituting in (4.12),} \\
&= \arg\min_{\mathbf{w}\in\mathbb{R}^n} \Big(-\sum_{i=1}^{N} \log(\frac{1}{\sigma\sqrt{2\pi}}) + \sum_{i=1}^{N} \frac{1}{2}(\frac{y_i - \mathbf{w}^T\mathbf{x}^{(i)}}{\sigma})^2 \Big) && \text{by distributing the logarithm,} \\
&= \arg\min_{\mathbf{w}\in\mathbb{R}^n} \sum_{i=1}^{N} (y_i - \mathbf{w}^T\mathbf{x})^2 && \text{by discarding the constants,} \\
&= \arg\min_{\mathbf{w}\in\mathbb{R}^n} \frac{1}{N} \sum_{i=1}^{N} (y_i - \mathbf{w}^T\mathbf{x})^2 && \text{by dividing by } N.
\end{aligned} \tag{4.13}$$

The objective function in (4.13) represents the mean squared difference between the true value and the predicted value. This objective function formulation is commonly known as the **Mean Squared Error (MSE)**.

With the simplified optimization model given by (4.13), we are now in a position where we can more easily describe its properties, which will give rise to the closed-form solution.

4.2.3 Closed-Form Solution

Consider the equivalent formulation of (4.13) using matrix notation as defined in (4.4). Using $||\cdot||_2$ to denote the Euclidean distance we have

$$\arg\min_{\mathbf{w}\in\mathbb{R}^n} \frac{1}{N}||\mathbf{Y}-\mathbf{X}\mathbf{w}||_2^2 \tag{4.14}$$

The optimization model in (4.14) represents an unconstrained convex optimization model. Moreover, under certain conditions, there exists a closed-form solution with a uniquely determined solution $\mathbf{w}^*$. Proposition 4.1 is useful for proving this statement. Given that the proposition is originally an exercise in [45], we have included a proof. Since the validity of statements (i) and (ii) in Proposition 4.1 is apparent, our focus lies on proving statement (iii).

Proposition 4.1. *([45]).*

Let matrix $\mathbf{B}\in\mathbb{R}^{n\times k}$ be given and let matrix $\mathbf{A}=\mathbf{B}\mathbf{B}^T$.

Then the following holds:

- *(i): $\mathbf{A}$ is symmetric.*
- *(ii): $\mathbf{A}$ is positive semidefinite.*
- *(iii): $\mathbf{A}$ is positive definite if and only if $\mathbf{B}$ has full row rank.*

Proof. (iii) : Assume $\mathbf{A}$ is positive definite, that is,

$$\mathbf{x}^T\mathbf{A}\mathbf{x} > 0 \ \forall \mathbf{x}\in\mathbb{R}^n\setminus\{\mathbf{0}\}. \tag{4.15}$$

Replacing $A=\mathbf{B}\mathbf{B}^T$ in (4.15) we have

$$0 < \mathbf{x}^T\mathbf{B}\mathbf{B}^T\mathbf{x} = (\mathbf{B}^T\mathbf{x})^T(\mathbf{B}^T\mathbf{x}) \ \forall \mathbf{x}\in\mathbb{R}^n\setminus\{\mathbf{0}\}. \tag{4.16}$$

From (4.16) it must be that $\mathbf{B}^T\mathbf{x}\neq\mathbf{0}$. Assuming this is not the case, that is, $\mathbf{B}^T\hat{\mathbf{x}}=\mathbf{0}$ for some $\hat{\mathbf{x}}\in\mathbb{R}^n\setminus\{\mathbf{0}\}$, it would contradict (4.16). With this fact along with (4.16) we have

$$\mathbf{B}^T\mathbf{x}\neq 0 \ \forall \mathbf{x}\in\mathbb{R}^n\setminus\{\mathbf{0}\}. \tag{4.17}$$

Denoting $\mathbf{B}^T = (\boldsymbol{\beta}_1 \dots \boldsymbol{\beta}_n)$ where vector $\boldsymbol{\beta}_i$ is the i-th row of $\mathbf{B}$, we can write (4.17) as

$$x_1\boldsymbol{\beta}_1 + \ldots + x_n\boldsymbol{\beta}_n \neq 0 \ \forall \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}. \tag{4.18}$$

No non-trivial solution exists, that is, $\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n\}$ form a linearly independent set and full row rank of $\mathbf{B}$ follows.

Proving the other direction, assume $\mathbf{B}$ has full row rank, that is, $\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n\}$ form a linearly independent set. With this assumption we have

$$\mathbf{B}^T\mathbf{x} \neq 0 \ \forall \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}. \tag{4.19}$$

From (4.19), it must be for all $\mathbf{x}$ in $\mathbb{R}^n \setminus \{\mathbf{0}\}$ that

$$0 < |\mathbf{B}^T\mathbf{x}|_2^2 = (\mathbf{B}^T\mathbf{x})^T(\mathbf{B}^T\mathbf{x}) = \mathbf{x}^T\mathbf{B}\mathbf{B}^T\mathbf{x} = \mathbf{x}^T\mathbf{A}\mathbf{x}, \tag{4.20}$$

and the result follows. □

We now present the closed-form solution. The following proposition is based on the conclusive ideas in [85]. These are, however, presented with little reasoning. Since it considers itself with optimization, we have decided to include a proof of the proposition here.

Proposition 4.2. *(Ordinary Least Squares Solution [85]).*

Let

$$f(\mathbf{w}) = \frac{1}{N}||\mathbf{Y} - \mathbf{X}\mathbf{w}||_2^2 \tag{4.21}$$

where $\mathbf{X}, \mathbf{Y}$ *are defined as in (4.4) and* $\mathbf{w} \in \mathbb{R}^n$*. It then follows that* f *is convex over* $\mathbb{R}^n$*. If in addition,* $\mathbf{X}$ *has full column rank, then* f *attains a unique global minimum at*

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}. \tag{4.22}$$

Proof. To prove the convexity of f over $\mathbb{R}^n$ we will use Proposition 2.2 part (i) which ensures that f is convex over $\mathbb{R}^n$ if the Hessian $\mathcal{H}_f(\mathbf{w})$ is positive semidefinite for all $\mathbf{w} \in \mathbb{R}^n$.

Expanding and applying the rules of transposes to f we get

$$\begin{aligned}\frac{1}{N}||\mathbf{Y}-\mathbf{X}\mathbf{w}||_2^2 &= \frac{1}{N}(\mathbf{Y}-\mathbf{X}\mathbf{w})^T(\mathbf{Y}-\mathbf{X}\mathbf{w}) \\ &= \frac{1}{N}(\mathbf{Y}^T-\mathbf{w}^T\mathbf{X}^T)(\mathbf{Y}-\mathbf{X}\mathbf{w}) \\ &= \frac{1}{N}(\mathbf{Y}^T\mathbf{Y}-\mathbf{Y}^T\mathbf{X}\mathbf{w}-\mathbf{w}^T\mathbf{X}^T\mathbf{Y}+\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}). \qquad (4.23)\end{aligned}$$

By making use of the commutative property of the dot product in Euclidean space, we can combine the second and third term in (4.23). We then have

$$\frac{1}{N}(\mathbf{Y}^T\mathbf{Y}-2\mathbf{Y}^T\mathbf{X}\mathbf{w}+\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}). \qquad (4.24)$$

Taking the derivative of (4.24) we get

$$D_{\mathbf{w}}f(\mathbf{w}) = \frac{2}{N}(\mathbf{w}^T\mathbf{X}^T\mathbf{X}-\mathbf{Y}^T\mathbf{X}). \qquad (4.25)$$

From (4.25), we have that the Hessian of f is given by

$$\mathcal{H}_f(\mathbf{w}) = \frac{2}{N}\mathbf{X}^T\mathbf{X}. \qquad (4.26)$$

Applying Proposition 4.1 to $\mathbf{X}^T\mathbf{X}$ (with $B = \mathbf{X}^T$ and $B^T = \mathbf{X}$), we have that $\mathbf{X}^T\mathbf{X}$ is positive semidefinite. Thus, f is a convex function over $\mathbb{R}^n$.

Assume now that $\mathbf{X}$ has full column rank. With f being convex over $\mathbb{R}^n$, Proposition 2.4 ensures that $D_{\mathbf{w}}f(\mathbf{w}) = \mathbf{0}^T$ is a sufficient condition for optimality.

Equating the derivative in (4.25) to zero and making the appropriate rearrangement, we have

$$\mathbf{w}^T\mathbf{X}^T\mathbf{X} = \mathbf{Y}^T\mathbf{X}. \qquad (4.27)$$

By Proposition 4.1 and $\mathbf{X}$ having full column rank, it must be that $\mathbf{X}^T\mathbf{X}$ is positive definite. Positive definiteness of $\mathbf{X}^T\mathbf{X}$, implies the existence of its inverse. Transposing (4.27), and then left-multiplying by the inverse of $\mathbf{X}^T\mathbf{X}$, we have

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}. \tag{4.28}$$

Equation (4.28) describes a local minimizer of our optimization model (4.14). Since f is a convex function, by Proposition 2.3, we have that any local minimizer must also be a global minimizer, hence, (4.28) describes a global minimizer. Furthermore, since the Hessian of f (4.26) is positive definite the global minimizer obtained in (4.28) is uniquely determined. □

The method of applying MLE, which gives rise to the optimization model (4.14), is also known as **ordinary least squares**. As described in [70], its name comes from the fact that we minimize the sum of the squares of the differences between the true values and the predicted values.

We want to close this section by mentioning that even though we have the closed-form solution (4.28), it can be beneficial to reject the closed-form solution and find numerical solutions to optimization model (4.13) instead. The main reason, as described in [85], is due to computational cost of the closed-form solution (4.28). Potential issues are the matrix multiplications and matrix inversion. As previously stated, $\mathbf{Y} \in \mathbb{R}^N$ and $\mathbf{X} \in \mathbb{R}^{N\times n}$, where N describes the number of data points, and n the dimensionality of each $\mathbf{x}^{(i)}$. These matrix multiplications and inversions can therefore become costly as the number or dimensionality of the data increases. These operations start to become time consuming to calculate due to their complexity.

Furthermore, as discussed in [85], heuristics applied to the optimization model (4.13), particularly the gradient descent algorithm, have been extensively tested. In the general case, this algorithm converges to a local minimizer. Leveraging the convexity of the optimization model (4.13), Proposition 2.3 ensures that any local minimizer is also a global minimizer. Consequently, the gradient descent algorithm in many cases converges to a global minimizer.

4.3 Maximum a Posteriori (MAP) for Linear Regression

4.3.1 Maximum a Posteriori (MAP)

Before introducing the general method of MAP, let us first consider Bayes' theorem, as described in [65].

Proposition 4.3. *(Bayes' Theorem [65]).*

Let $A_1, \ldots, A_k$ be a collection of mutually exclusive and exhaustive events with $P(A_i) > 0$ for $i = 1, \ldots, k$. Then for any other event B for which $P(B) > 0$,

$$P(A_j|B) = \frac{P(A_j, B)}{P(B)} = \frac{P(B|A_j)P(A_j)}{\sum_{i=1}^{k} P(B|A_i)P(A_i)}, \ j = 1, \ldots, k. \tag{4.29}$$

We will now dissect each of the components of (4.29). We will refer to A_j and B as our hypothesis and evidence respectively.

We call $P(A_j|B)$ the **posterior probability**, or simply the **posterior**. This is the probability of hypothesis A_j, when taking into consideration the observed evidence B.

The likelihood, which was already covered in Section 4.2 is given by $P(B|A_j)$. In this context, it can be interpreted as the probability of observing evidence B, given that hypothesis A_j is true.

The **prior probability**, or **prior**, is given by $P(A_j)$ and can be understood as the initial belief or probability for hypothesis A_j, before taking evidence B into account.

The **marginal likelihood** is given by $P(B)$. It can be interpreted as the probability of observing evidence B, independent of any hypothesis. Alternatively, we can represent the probability of the evidence by $\sum_{i=1}^{k} P(B|A_i)P(A_i)$. With this, we can interpret the probability of B in terms of considering all possible hypotheses. For each of the hypotheses, we calculate the probability of seeing evidence B given the hypothesis. Summing over all of the possible hypotheses, we have our probability of B.

In the context of MAP, we wish to find that hypothesis that maximizes the posterior, given the evidence B. Denoting our hypothesis space by $\mathcal{H}_{\mathcal{S}}$, we have

$$\mathbf{w}^* = \arg\max_{A \in \mathcal{H}_{\mathcal{S}}} P(A|B). \tag{4.30}$$

4.3.2 Optimization Model

We will now be applying MAP to the framework discussed in Section 4.1. From the general approach to MAP seen in (4.30), we have the unconstrained optimization problem given by

$$\mathbf{w}^* = \arg\max_{\mathbf{w}\in\mathbb{R}^n} P(\mathbf{w}|\mathcal{D}). \tag{4.31}$$

Explaining in terms of hypothesis A and evidence B, as seen in Section 4.3.1, $\mathcal{D}$ acts as the evidence, and parameter $\mathbf{w}$ acts as the hypothesis. The hypothesis space corresponds to $\mathbb{R}^n$. We then have the optimal parameters maximizing the posterior given by $\mathbf{w}^*$.

From [85], we can make the following derivation. We start by applying Bayes' theorem (4.3) to expand the objective function of (4.31). Expressing our dataset using matrix notation as defined in (4.4), we have, we have

$$\arg\max_{\mathbf{w}\in\mathbb{R}^n} \frac{P(\mathbf{X},\mathbf{Y}|\mathbf{w})P(\mathbf{w})}{P(\mathbf{X},\mathbf{Y})}. \tag{4.32}$$

As previously discussed in Section 4.3.1, the objective function in (4.32) is now expressed in terms of the likelihood, prior, and marginal likelihood. The crucial part is the prior $P(\mathbf{w})$, for which we lack a specified probability distribution. As explained in [85], this presents an opportunity to incorporate our own subjective prior beliefs regarding the probability distribution for $\mathbf{w}$, before considering the observed dataset $\mathcal{D}$. As further stated in [85], this incorporation of belief allows individuals conducting the regression analysis to infuse their prior knowledge, comprised of experience, expert advice, and other factors, into the formulation of the prior distribution.

In the continued derivation in [85], it becomes necessary to decide on a prior for $P(\mathbf{w})$. One natural approach, as seen in [70], is to acknowledge a lack of prior knowledge regarding the potential values of $\mathbf{w}$. Instead, we should let the likelihood $P(\mathbf{X},\mathbf{Y}|\mathbf{w})$, i.e., the observed data guide our conclusions. By adopting this perspective, we state that no particular $\mathbf{w}$ should have a higher likelihood of occurring than any other. Mathematically, we have that for all $\mathbf{w}\in\mathbb{R}^n$, $P(\mathbf{w}) = \alpha$ for some real $\alpha > 0$. This corresponds to applying

a uniform distribution for $\mathbf{w}$, that is,

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \text{ with } w_i \sim \ Uniform(-\infty, +\infty). \tag{4.33}$$

Applying the uniform distribution along with (4.32) gives us

$$\arg\max_{\mathbf{w}\in\mathbb{R}^n} \frac{P(\mathbf{X}, \mathbf{Y}|\mathbf{w}) \cdot \alpha}{P(\mathbf{X}, \mathbf{Y})}. \tag{4.34}$$

Since both $P(\mathbf{X}, \mathbf{Y})$ and α are positive constants independent of $\mathbf{w}$, they can be discarded and we have

$$\arg\max_{\mathbf{w}\in\mathbb{R}^n} P(\mathbf{X}, \mathbf{Y}|\mathbf{w}). \tag{4.35}$$

The optimization model represented by (4.35) is exactly the same as the MLE optimization model discussed in Section 4.2. By assuming $\mathbf{w}$ being drawn from a uniform distribution, we have reached the same optimization model. This is consistent with our expectations, since we are not incorporating any prior belief about what values for $\mathbf{w}$ are more probable than others. Essentially, our inference relies solely on the observed data, as when using the method of MLE.

With this, the question arises, what happens if we use a different prior distribution for $P(\mathbf{w})$, that is, we do not assume the probability for $\mathbf{w}$ to be uniform, but allow some values to be more likely than others? By using a non-uniform distribution we are saying that we have beliefs that certain $\mathbf{w}$ values are more likely to occur than others. Assuming different probability distributions for $\mathbf{w}$ does, in fact, yield different objective functions.

A shortened table summary from [70] when applying different priors can be found in Table 4.1.

Likelihood Distribution	Prior Distribution	Name
Gaussian	Uniform	Least Squares
Gaussian	Gaussian	Ridge
Gaussian	Laplace	Lasso

Table 4.1: Three well-established methods of regression analysis when incorporating different prior distributions for $\mathbf{w}$ using the method of MAP.

Among the three priors found in Table 4.1, as done in [85], we will henceforth consider $\mathbf{w}$ being drawn from a multivariate Gaussian distribution. The pdf for a t-dimensional multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and positive definite covariance matrix $\boldsymbol{\Sigma}$, as seen in [14], is given by

$$P(\boldsymbol{\eta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{t}{2}} det(\boldsymbol{\Sigma})^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{\eta} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\eta} - \boldsymbol{\mu}) \right). \tag{4.36}$$

For the Gaussian distribution corresponding to $\mathbf{w}$, we assume a mean of $\mathbf{0}$, and a covariance matrix given by $\tau^2\mathbf{I}$ with $\tau^2 > 0$ and $\mathbf{I}$ being the (n, n)-identity matrix. for $\mathbf{w}$ we have

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \tau^2\mathbf{I}). \tag{4.37}$$

The pdf corresponding to (4.37) is constructed using (4.36). We have

$$P(\boldsymbol{\eta}|\mathbf{0}, \tau^2\mathbf{I}) = \frac{1}{(2\pi\tau^2)^{\frac{n}{2}}} \exp\left(-\frac{1}{2\tau^2}\boldsymbol{\eta}^T\boldsymbol{\eta} \right). \tag{4.38}$$

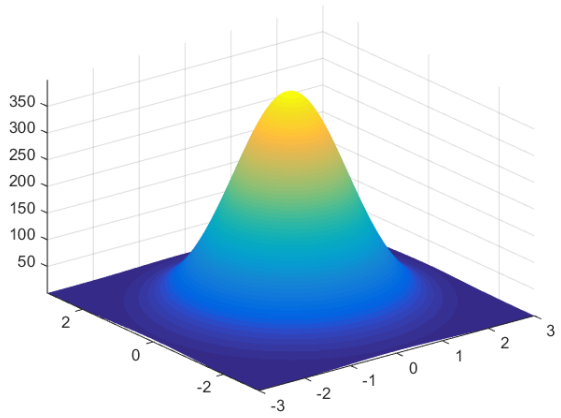

Figure 4.3: Depicted is a 2-dimensional Gaussian distribution with zero mean and a covariance matrix equal to the identity matrix positively scaled by some factor. In the n-dimensional case, we can imagine an n-dimensional hypersphere centred at the origin. $\mathbf{w}$ is then more likely to take on values closer to center of the sphere.

Figure (taken from [18]).

We will now continue the derivation as seen in [85], although we will make it more concise, as the derivational steps are similar to that of MLE.

In applying the chain rule of probability to optimization model (4.32) we have

$$\arg\max_{\mathbf{w}\in\mathbb{R}^n} \frac{P(\mathbf{Y}|\mathbf{X},\mathbf{w})P(\mathbf{X}|\mathbf{w})P(\mathbf{w})}{P(\mathbf{Y}|\mathbf{X})P(\mathbf{X})}. \tag{4.39}$$

We have that $\mathbf{X}$ is independent of $\mathbf{w}$ because $\mathbf{X}$ is part of the observed data collected before considering $\mathbf{w}$. Therefore, $P(\mathbf{X}|\mathbf{w}) = P(\mathbf{X})$. Further simplifying (4.39), we have

$$\arg\max_{\mathbf{w}\in\mathbb{R}^n} \frac{P(\mathbf{Y}|\mathbf{X},\mathbf{w})P(\mathbf{w})}{P(\mathbf{Y}|\mathbf{X})}. \tag{4.40}$$

Since $P(\mathbf{Y}|\mathbf{X})$ is a constant independent of $\mathbf{w}$, and the logarithm is monotonically increasing, we simplify (4.41) to

$$\arg\max_{\mathbf{w}\in\mathbb{R}^n} \log P(\mathbf{Y}|\mathbf{X},\mathbf{w}) + \log P(\mathbf{w}). \tag{4.41}$$

Since each $y_i \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}^{(i)}, \sigma^2)$ as seen in (4.5), we can use the matrix notation in (4.4) to describe $\mathbf{Y}$ from a multivariate Gaussian distribution, that is,

$$\mathbf{Y} \sim \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I}). \tag{4.42}$$

The pdf corresponding to (4.42) is constructed using (4.36). Note that the when considering $\sigma^2\mathbf{I}$, we have $(\sigma^2\mathbf{I})^{-1} = \frac{1}{\sigma^2}\mathbf{I}$ and $det(\sigma^2\mathbf{I}) = \sigma^{2n}$. We have that the pdf is given by

$$P(\boldsymbol{\eta}|\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I}) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\bigg(-\frac{1}{2\sigma^2}||\boldsymbol{\eta} - \mathbf{X}\mathbf{w}||_2^2\bigg). \tag{4.43}$$

Using the pdfs (4.38) and (4.42) for $\mathbf{w}$ and $\mathbf{Y}$ respectively, we can substitute them into (4.41). We have

$$\arg\max_{\mathbf{w}\in\mathbb{R}^n} \bigg[\log\bigg(\frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}}\bigg) - \frac{1}{2\sigma^2}||\mathbf{Y} - \mathbf{X}\mathbf{w}||_2^2 + \log\bigg(\frac{1}{(2\pi\tau^2)^{\frac{n}{2}}}\bigg) - \frac{1}{2\tau^2}\mathbf{w}^T\mathbf{w}\bigg]. \tag{4.44}$$

By discarding unnecessary constants in (4.44) we get

$$\arg\max_{\mathbf{w}\in\mathbb{R}^n} \bigg(-\frac{1}{2\sigma^2}||\mathbf{Y} - \mathbf{X}\mathbf{w}||_2^2 - \frac{1}{2\tau^2}\mathbf{w}^T\mathbf{w}\bigg). \tag{4.45}$$

Minimizing and defining $\psi = \frac{\sigma^2}{N\tau^2}$ for (4.45) results in

$$\arg\min_{\mathbf{w}\in\mathbb{R}^n} \frac{1}{N}||\mathbf{Y} - \mathbf{X}\mathbf{w}||_2^2 + \psi\mathbf{w}^T\mathbf{w}. \tag{4.46}$$

The optimization model derived in (4.46) is called **ridge regression**. The objective function closely resembles the objective function found when using MLE in Section 4.2, except for the additional term given by $\psi||\mathbf{w}||_2^2$. We follow the description of $\psi||\mathbf{w}||_2^2$ in [79]. By introducing the prior belief for $P(\mathbf{w})$ as a zero-mean Gaussian, we have built in the assumption that $\mathbf{w}$ are more likely to take on values centred at the origin. This assumption of $\mathbf{w}$, can be understood as a penalization in the objective function of (4.46), with the degree of penalization determined by $\psi > 0$. The further away $\mathbf{w}$ strays from the origin, the more the objective function is penalized. We would then prefer to take a smaller $\mathbf{w}$ to achieve a smaller penalty term.

From a statistics perspective, in [79], it is shown that with the suggested prior belief for $\mathbf{w}$, there is a decrease in the variance of the MAP estimator, that is, the optimal solution of the optimization model (4.46). The decrease in variance comes from the prior belief that the values of $\mathbf{w}$ are centered around the origin.

With the formulation given by (4.46), we will in the next section consider the closed-form solution.

4.3.3 Closed-Form Solution

Applying the matrix notation seen in (4.4), optimization model (4.46) can be equivalently formulated as

$$\arg\min_{\mathbf{w}\in\mathbb{R}^n} ||\mathbf{Y} - \mathbf{X}\mathbf{w}||_2^2 + \Psi\mathbf{w}^T\mathbf{w}. \tag{4.47}$$

We have purposefully left out the constant $\frac{1}{N}$. Doing so will not affect the optimization model. Furthermore, we will substitute $\psi = \frac{\sigma^2}{N\tau^2}$ with the parameter Ψ where $0 < \Psi \in \mathbb{R}$. In the context of SL, Ψ is a hyperparameter. We make these changes in order to reach the same conclusions presented in [85].

In describing the corresponding closed-form solution for the optimization model (4.47), like we did with in the presenting of the closed-form solution for MLE, we rely on the

conclusive statements from [85]. Additionally, these statements are presented without proof. Since the discussion is relevant to optimization, we have decided to prove the statement in the following proposition.

Proposition 4.4. *(Ridge Regression Solution [85]).*

Let

$$f(\mathbf{w}) = ||\mathbf{Y} - \mathbf{X}\mathbf{w}||_2^2 + \Psi \mathbf{w}^T \mathbf{w}, \tag{4.48}$$

where $\mathbf{X}, \mathbf{Y}$ *are defined as in (4.4),* $\mathbf{w} \in \mathbb{R}^n$ *and* $0 < \Psi \in \mathbb{R}$*. It then follows that* f *is strictly convex over* $\mathbb{R}^n$ *and* f *attains a unique global minimum at*

$$\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \Psi\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}, \tag{4.49}$$

where $\mathbf{I}$ *is the* (n, n)*-identity matrix.*

Proof. To prove f is strictly convex over $\mathbb{R}^n$ we will use Proposition 2.2 part (ii) which ensures that f is strictly convex over $\mathbb{R}^n$ if the Hessian $\mathcal{H}_f(\mathbf{w})$ is positive definite for all $\mathbf{w} \in \mathbb{R}^n$.

Taking the derivative of f we get

$$D_{\mathbf{w}} f(\mathbf{w}) = 2(\mathbf{w}^T\mathbf{X}^T\mathbf{X} - \mathbf{Y}^T\mathbf{X}) + 2\Psi\mathbf{w}^T. \tag{4.50}$$

From (4.50), we have that the Hessian of f is given by

$$\mathcal{H}_f(\mathbf{w}) = 2\mathbf{X}^T\mathbf{X} + 2\Psi\mathbf{I}. \tag{4.51}$$

From Proposition 4.1, we have that $\mathbf{X}^T\mathbf{X}$ is positive semidefinite, that is,

$$\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} \geq 0, \ \forall \ \mathbf{w} \in \mathbb{R}^n. \tag{4.52}$$

As $\Psi > 0$, we have

$$\mathbf{w}^T 2\Psi\mathbf{I}\mathbf{w} > 0, \ \forall \ \mathbf{w} \in \mathbb{R}^n \setminus \{\mathbf{0}\}, \tag{4.53}$$

This implies that $2\Psi\mathbf{I}$ is positive definite by definition. The combination of (4.52) and (4.53) implies that the Hessian in (4.51) is positive definite, which establishes that f is strictly convex over $\mathbb{R}^n$.

With f being convex over $\mathbb{R}^n$, Proposition 2.4 ensures that $D_{\mathbf{w}}f(\mathbf{w}) = \mathbf{0}^T$ is a sufficient condition for optimality. Equating the expression in (4.50) to zero and making the appropriate rearrangement, we have

$$(\mathbf{X}^T\mathbf{X} + \Psi\mathbf{I})\mathbf{w} = \mathbf{X}^T\mathbf{Y}. \tag{4.54}$$

As stated by (4.52) and (4.53), $\mathbf{X}^T\mathbf{X} + \Psi\mathbf{I}$ is positive definite, hence, its inverse exists. From (4.54), we get

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \Psi\mathbf{I})^{-1}\mathbf{X}^T\mathbf{Y}. \tag{4.55}$$

Equation (4.55) describes the global minimizer of f. From Proposition 2.3, this minimizer is uniquely determined due to the strict convexity of f over $\mathbb{R}^n$. □

Chapter 5

Optimization In Support Vector Machines (SVM)

Support Vector Machines (SVM) are a family of SL algorithms that were initially introduced for the classification task of SL, as stated in [5]. In its most extensive form, the SVM algorithm is capable of finding non-linear decision boundaries separating datasets with multiple classes. Described in [56], the development of the SVM started in 1962 and was first published in 1964 by Vladimir Vapnik and Alexey Chervonenkis. Later development of the SVM as stated in [5] gave rise to Support vector regression machines which made the SL algorithm applicable for the regression task.

In this chapter, we will explore SVM with a focus on solving the classification task, which was its initial purpose. Moreover we will focus on the binary classification task because SVM for multiclass classification consists of breaking the multiclass classification problem into smaller binary classification problems. Inspecting the binary classification task will be sufficient in understanding SVM. From the generalized SL dataset (3.2) in Chapter 3, we have that the generalized binary classification task considers the dataset given by

$$\mathcal{D} = \{(\mathbf{x}^{(i)} \in \mathbb{R}^n, y_i \in \mathcal{C})\}_{i=1}^{N}, \tag{5.1}$$

where the labels are given by $\mathcal{C} = \{+1 - 1\}$. This is the dataset we will consider throughout this chapter.

We will start this chapter by investigating the McCulloch-Pitts neuron and the perceptron algorithm, which, described in [87] can both be seen as foundational work in

which the SVM is built upon. Further, we will investigate the hard-margin SVM which assumes that the data are linearly separable. Following the definition in [1], we call $\mathcal{D}$ a linearly separable dataset if and only if there exists $\mathbf{z} \in \mathbb{R}^n$ and $q \in \mathbb{R}$ such that for all $i = 1, \ldots, N$ we have

$$y_i(\mathbf{z}^T \mathbf{x}^{(i)} + q) > 0. \tag{5.2}$$

After the discussion of the hard-margin SVM we introduce the soft-margin SVM which relaxes the linearly separable assumption of dataset $\mathcal{D}$. Thereafter, we will consider the dual formulation of the soft-margin SVM which is often referred to as the dual SVM. Important to the dual SVM is its valuable characteristics which we will explore and leverage by considering the concepts of feature mapping and kernels in the subsequent sections. Doing so will lead us to the development of the kernel SVM.

5.1 Rosenblatt's Perceptron

5.1.1 The McCulloch-Pitts Neuron

As mentioned in [32], the foundation for what would evolve into the perceptron was established by Warren McCulloch and Walter Pitts in 1943. In their paper, *A logical calculus of the ideas immanent in nervous activity* [68], they proposed a simplified mathematical model of how neurons in the brain could be represented. Prominent in their paper is showing that their proposed neuron is capable of modelling multiple logic gates such as AND, OR, NOR as seen in [55]. Such a neuron is commonly known as a **McCulloch-Pitts neuron**.

Adapting the idea from the biological neuron, the McCulloch-Pitts neuron either fires or does not fire, the output is either 0 or 1. Adapting notation in [27], the output of the neuron is determined by the binary input $\mathbf{I} \in \mathbb{R}^n$, which is an N-dimensional vector with components taking on values of either 0 or 1, the weights $\mathbf{W} \in \mathbb{R}^N$, the summation function $g(\mathbf{I}) = \mathbf{I}^T\mathbf{W}$, and the threshold function f_θ in [32] defined as

$$f_\theta(x) = \begin{cases} 1 & \text{if } \mathbf{I}^T\mathbf{W} \geq \theta, \\ 0 & \text{if } \mathbf{I}^T\mathbf{W} < \theta, \end{cases} \tag{5.3}$$

where θ is the fixed threshold value. Note that in the McCulloch-Pitts paper, the neuron had no weights, or rather, the weights were all fixed to 1. The idea of weights were later introduced by Donald Hebb in 1949 as stated in [32]. We have that the output of the McCulloch-Pitts neuron can be represented by the composite function $h(\mathbf{I}) = f_\theta(g(\mathbf{I})) = f_\theta(\mathbf{I}^T\mathbf{W})$.

Due to having a higher similarity with the perceptron, an illustration of the McCulloch-Pitts neuron with weights can be seen in Figure 5.1.

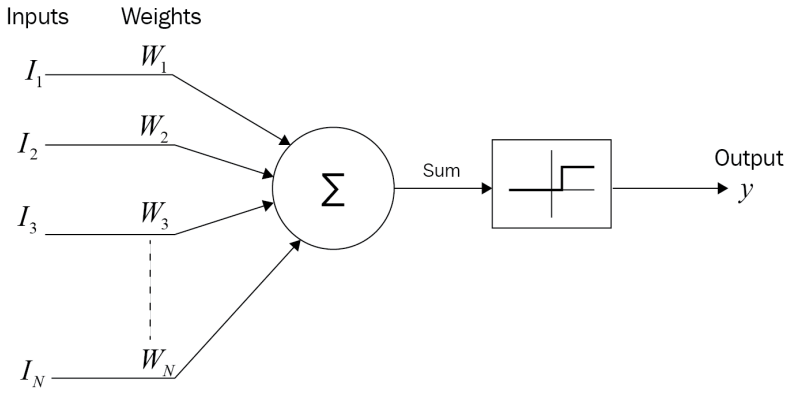

Figure 5.1: The modified McCulloch-Pitts neuron with weights as introduced by Donald Hebb. Given some binary stimuli $\mathbf{I}$ and by using fixed weights $\mathbf{W}$, the value $\mathbf{I}^T\mathbf{W}$ is calculated. This value is then used as input for the threshold function f_θ. The neuron then either fires a signal, indicated by 1, or remains dormant, indicated by 0.

Figure (taken from [27]).

With the McCulloch-Pitts neuron established, we may now continue to the framework of the perceptron, which provides ideas and concepts important for the SVM.

5.1.2 Invention of the Perceptron

In the 1957 paper *The Perceptron — A Perceiving and Recognizing Automaton* [76], the author Frank Rosenblatt proposed an SL algorithm for finding a separating decision boundary, that is, a separating hyperplane with respect to the binary classification dataset (5.1), assuming that the data are linearly separable. The main component of the perceptron is the McCulloch-Pitts neuron in order to represent the separating hyperplane.

An important component of the perceptron is that it has a learning algorithm in order to update the weights by training on the data points in the dataset. A detailed

description of the learning algorithm is given in [87]. Summarized, the learning algorithm goes through all data points $\mathbf{x}^{(i)}$ until they all are classified correctly to the corresponding target y_i. If a data point is misclassified, the algorithm will adjust the hyperplane in order to correctly classify the misclassified data point. The algorithm keeps going until all data points are correctly classified and a separating hyperplane has been found. In [9], it is guaranteed that if a binary classification dataset is linearly separable, then the learning algorithm will eventually converge to a separating hyperplane.

Important to note is that although the converged hyperplane separates the data points, in regards to the converged hyperplane, the perceptron algorithm has no criteria of goodness defined by a constructed measure. All it guarantees is that it separates a dataset into two classes. We can understand the SVM as an extension of the perceptron in the sense that the it actively seeks a hyperplane with respect to a defined measure. In the next section we will delve into the consideration of such a measure when examining SVM.

	McCulloch-Pitts Neuron	**Perceptron**
Inputs	The original neuron implementation processes only binary input data (from [32]).	The perceptron allows for real-valued inputs. In the sense of belonging to a larger network of neurons, this allows the perceptron to take into account the strength of the input.
Learning Rule	The neuron does not involve weight updates. The weights are fixed in advance (from [32]).	The learning algorithm for the perceptron proposed by Rosenblatt updates the weights to minimize the misclassification over a set of training examples (from [87]).
Learning Capability	The neuron is a simple mathematical model with no learning capabilities. It is a static model used to illustrate basic logical and computational operations (from [87]).	The perceptron has the ability to learn decision boundaries in order to classify data into two classes. Its limitation is that it assumes the data is linearly separable and binary (from [87]).
Applications	The neuron is mainly used as a theoretical model for understanding basic concepts of neural computation and logical operations.	The perceptron has historically been used for binary classification tasks where the classes can be separated by a linear boundary. It is a stepping stone in the development of more sophisticated algorithms (from [87]).

Table 5.1: A table highlighting the differences between the McCulloch-Pitts neuron and the perceptron algorithm.

5.2 Hard-Margin SVM

The hard-margin SVM is a variant within the SVM family that makes assumptions about what an effective separating hyperplane should be. Formulating such assumptions about

the quality of a hyperplane enables the creation of a hierarchy of hyperplanes, providing insights into determining the optimal separating hyperplane. Naturally, this can be formulated as an optimization model.

Consider the binary classification dataset $\mathcal{D}$. For this section, we make the assumption that the dataset is linearly separable, that is, there exists $\hat{\mathbf{z}} \in \mathbb{R}$ and $\hat{q} \in \mathbb{R}$ such that for all $i = 1, \ldots, N$, the labeled data points $(\mathbf{x}^{(i)}, y_i)$ in $\mathcal{D}$ satisfies Inequality (5.2).

From [86], we have that there is an infinite amount of such separating hyperplanes. The question arises, how should we choose between the hyperplanes, that is, how do we determine that one is better than another? The key observation is that we would like to maximize the distance from the data points $\mathbf{x}^{(i)}$ in $\mathcal{D}$ to the separating hyperplane, this maximum distance being called the **margin**. Motivations for maximizing the margin is given in [44], which builds on the underlying assumption that the data in $\mathcal{D}$ is generated from some unknown distribution. Data from $\mathcal{D}$ along with unobserved data points not in $\mathcal{D}$ but generated from the same distribution will then be distributed similarly, hopefully with some sort of pattern. As explained in [86], a larger distance between the hyperplane and the closest data point from $\mathcal{D}$ allows for a larger margin of error, where the model is less likely to misclassify on unseen data points.

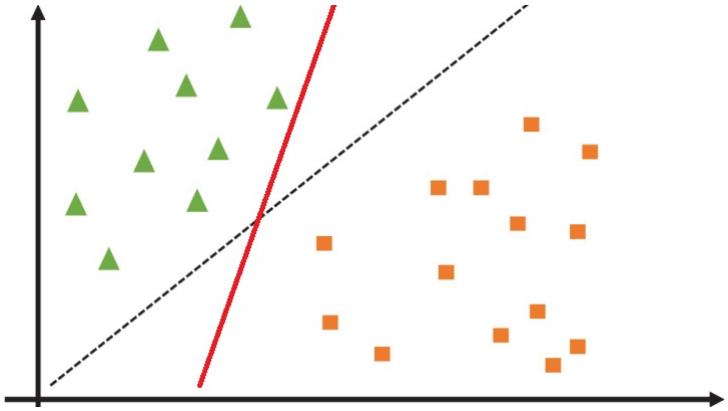

Figure 5.2: Consider the maximal margin hyperplane indicated by the dashed lines and the hyperplane indicated by red. Both hyperplanes are separating the training data. Note however that the hyperplane indicated by red has a smaller margin and is at a higher risk of misclassifying on unseen data points generated from the same distribution as the training data.

Figure (taken from [82]).

The following derivation to retrieve the hard-margin SVM optimization model is based on [86]. We have decided to adapt the explanation of the derivation into our own words

while also formalizing certain parts from [86]. We do this because we believe that investigating the derivation in detail will provide an understanding of how the SVM algorithm works.

In order to effectively describe the margin of a hyperplane mathematically, we will need equations to describe the distance from a hyperplane to an arbitrary data point. Let $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$ be given and consider the hyperplane defined by

$$H = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{w}^T\mathbf{x} + b = 0\}.$$

Also, consider an arbitrary point $\mathbf{x} \in \mathbb{R}^n$ with $\mathbf{x}_p$ being its projection onto H, thus satisfying $\mathbf{w}^T\mathbf{x}_p + b = 0$. Now define the difference $\mathbf{d} = \mathbf{x} - \mathbf{x}_p$. From [86], it follows that $\mathbf{d}$ is parallel with $\mathbf{w}$, that is, $\mathbf{d} = t\mathbf{w}$ for some $t \in \mathbb{R}$. Making the rearrangement $\mathbf{x}_p = \mathbf{x} - \mathbf{d} = \mathbf{x} - t\mathbf{w}$ we can derive that

$$\begin{aligned}
&\mathbf{w}^T(\mathbf{x} - t\mathbf{w}) + b = 0 && \text{by substituting for } \mathbf{x}_p, \\
&\mathbf{w}^T\mathbf{x} + b - t\mathbf{w}^T\mathbf{w} = 0 && \text{by expanding}, \\
&t = \frac{\mathbf{w}^T\mathbf{x} + b}{\mathbf{w}^T\mathbf{w}} && \text{by solving for } t, \\
&\mathbf{d} = (\frac{\mathbf{w}^T\mathbf{x} + b}{\mathbf{w}^T\mathbf{w}})\mathbf{w} && \text{by right-multiplying with } \mathbf{w}.
\end{aligned}$$

Since we are interested in the distance and not the distance vector itself, we can take the Euclidean norm of $\mathbf{d}$. We then have

$$\begin{aligned}
||\mathbf{d}||_2 &= \sqrt{(\frac{\mathbf{w}^T\mathbf{x} + b}{\mathbf{w}^T\mathbf{w}})\mathbf{w}^T(\frac{\mathbf{w}^T\mathbf{x} + b}{\mathbf{w}^T\mathbf{w}})\mathbf{w}} && \text{by Euclidean norm definition}, \\
&= \left|\frac{\mathbf{w}^T\mathbf{x} + b}{\sqrt{\mathbf{w}^T\mathbf{w}}}\right| && \text{by simplifying}.
\end{aligned}$$

An equation describing a point and its distance to a hyperplane has been obtained and we may now proceed to explicitly define the margin. We define the margin $\mathcal{G}(\mathbf{w}, b)$ mathematically as the distance from the hyperplane given by $\mathbf{w}$ and b to the closest data point across both labels, that is,

$$\mathcal{G}(\mathbf{w}, b) = \min_{i \in \{1,\ldots,N\}} \left|\frac{\mathbf{w}^T\mathbf{x}^{(i)} + b}{||\mathbf{w}||_2}\right|. \tag{5.4}$$

Maximizing (5.4) with respect to $\mathbf{w}$ and b gives us the hyperplane with the largest margin. This results in the optimization model given by

$$\max_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} \mathcal{G}(\mathbf{w}, b). \tag{5.5}$$

In leaving the optimization model (5.5) unconstrained, observe that we can then move the hyperplane arbitrarily far away to maximize the margin. By adding constraints, specifically the inequalities describing the dataset $\mathcal{D}$ being linearly separable (5.2), we can enforce the hyperplane to lie between the two classes. Alternatively, we can express this as enforcing the data points to be on the correct side of the hyperplane. We have

$$\begin{aligned} &\max_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} \left(\min_{i\in\{1,\dots,N\}} \left| \frac{\mathbf{w}^T\mathbf{x}^{(i)} + b}{||\mathbf{w}||_2} \right| \right) \\ &\text{s.t.} \\ &y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) > 0 \quad i = 1, \dots, N. \end{aligned} \tag{5.6}$$

Since $\mathbf{w}$ is a constant with respect to the inner minimization, we can simplify the objective function in (5.6) and we then have

$$\begin{aligned} &\max_{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}} \frac{1}{\sqrt{\mathbf{w}^T\mathbf{w}}} \left(\min_{i\in\{1,\dots,N\}} |\mathbf{w}^T\mathbf{x}^{(i)} + b| \right). \\ &\text{s.t.} \\ &y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) > 0 \quad i = 1, \dots, N. \end{aligned} \tag{5.7}$$

Further simplification is obtained by considering the hyperplane representation which is not uniquely determined. We consider the following example to educate this idea.

Example 5.1. *([86]).* ***Hyperplane Representation***

Let $\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$ and $k \in \mathbb{R} \setminus \{0\}$ be given. Consider

$$H = \{\mathbf{x} \in \mathbb{R}^n | \mathbf{w}^T\mathbf{x} + b = 0\} \tag{5.8}$$

and

$$\hat{H} = \{\mathbf{x} \in \mathbb{R}^n | k(\mathbf{w}^T\mathbf{x} + b) = k \cdot 0\}. \tag{5.9}$$

Since $H \subseteq \hat{H}$ and $\hat{H} \subseteq H$, we have $H = \hat{H}$ and the two hyperplane representations

represent the same hyperplane. Hyperplanes are therefore scale invariant and its representation is not unique.

Assume that $\hat{\mathbf{w}}$ and $\hat{b}$ satisfies the linearly separable constraints from (5.6) and let $\mathbf{x}^*$ be the data point that is closest to the hyperplane. We have

$$|\hat{\mathbf{w}}^T\mathbf{x}^* + \hat{b}| = t > 0. \tag{5.10}$$

Since the hyperplane representation is not unique, we can change it with $\frac{\hat{\mathbf{w}}}{t}$ and $\frac{\hat{b}}{t}$, which corresponds to $k = \frac{1}{t}$ in (5.9). With this new hyperplane representation we have for $\mathbf{x}^*$

$$|\frac{\hat{\mathbf{w}}}{t}^T\mathbf{x}^* + \frac{\hat{b}}{t}| = 1. \tag{5.11}$$

We can therefore, given the closest lying data point $\mathbf{x}^*$ perform a change in hyperplane representation such that $\mathbf{x}^*$ lies one unit away from the hyperplane, where the unit is with respect to $\frac{\hat{\mathbf{w}}^T}{t}$.

Consider optimization model (5.7) and a feasible solution $(\hat{\mathbf{w}}, \hat{b})$. Suppose that for some $t > 0$ we have

$$\min_{i\in\{1\ldots N\}} |\hat{\mathbf{w}}^T\mathbf{x}^{(i)} + \hat{b}| = t. \tag{5.12}$$

As outlined in Example 5.1, for Equation (5.12), we can make a change of hyperparameter representation such that

$$\min_{i\in\{1\ldots N\}} |\frac{\hat{\mathbf{w}}}{t}^T\mathbf{x}^{(i)} + \frac{\hat{b}}{t}| = 1. \tag{5.13}$$

As this approach is applicable for any feasible solution of optimization model (5.7), we can enforce the additional constraint given by

$$\min_{i\in\{1\ldots N\}} |\mathbf{w}^T\mathbf{x}^{(i)} + b| = 1. \tag{5.14}$$

With the constraint given by Equation (5.14), we can replace the inner minimization of the objective function of (5.6) by the constant 1. The optimization model (5.7) can

now be represented by

$$\min \mathbf{w}^T\mathbf{w} \tag{5.15}$$

s.t.

$$y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) > 0 \quad i = 1, \ldots, N, \tag{5.16}$$

$$\min_{i\in\{1,\ldots,N\}} |\mathbf{w}^T\mathbf{x}^{(i)} + b| = 1, \tag{5.17}$$

$$\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R},$$

where the minimization comes from observing that the square root function is monotonically increasing. It turns out that we can replace the constraints in (5.15) with a new set of constraints, thus giving rise to an optimization model which is often considered the optimization model when addressing the hard-margin SVM. It is given by

$$\min \mathbf{w}^T\mathbf{w} \tag{5.18}$$

s.t.

$$\begin{aligned} &y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \quad i = 1, \ldots, N, \\ &\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}. \end{aligned} \tag{5.19}$$

We can replace optimization model (5.15) with model (5.18) because any optimal solution of the former is also an optimal solution of the latter and vice versa. A sketch of the proof is outlined in [86]. We have therefore decided to include a formal portrayal of the same proof here. For clarity, we first consider an auxiliary statement in Proposition 5.1 before proving the statement in Proposition 5.2.

Proposition 5.1. *([86]).*

An optimal solution $(\mathbf{w}^*, b^*)$ *for optimization model (5.15) is feasible for optimization model (5.18), and conversely, an optimal solution for (5.18) is feasible for (5.15).*

Proof. Assume $(\mathbf{w}^*, b^*)$ is an optimal solution of optimization model (5.15). Considering Equation (5.17) we have

$$\min_{i\in\{1,\ldots,N\}} |(\mathbf{w}^*)^T\mathbf{x}^{(i)} + b^*| = 1. \tag{5.20}$$

Let data point $\mathbf{x}^*$ correspond to the global minimizer i^* of (5.17). Furthermore, let y^* denote the target corresponding to $\mathbf{x}^*$, that is, we have the labeled data point $(\mathbf{x}^*, y^*)$.

Multiplying (5.20) by $|y^*|$ gives

$$|y^*||(\mathbf{w}^*)^T\mathbf{x}^* + b^*| = |y^*| \cdot 1. \tag{5.21}$$

Moreover, since $y^* \in \{+1, -1\}$ and by applying the rules of absolute values it follows from (5.21) that

$$|y^*((\mathbf{w}^*)^T\mathbf{x}^* + b^*)| = 1. \tag{5.22}$$

By (5.16) we have that $y^*((\mathbf{w}^*)^T\mathbf{x}^* + b^*) > 0$. The absolute value sign in (5.22) can therefore be discarded and it must be that

$$y^*((\mathbf{w}^*)^T\mathbf{x}^* + b^*) = 1. \tag{5.23}$$

Since $\mathbf{x}^*$ is the global minimizer of (5.17), it must be that (5.19) holds and the point $(\mathbf{w}^*, b^*)$ is feasible for optimization model (5.18).

In showing the converse, assume $(\mathbf{w}^*, b^*)$ is an optimal solution of optimization model (5.18). Optimality implies feasibility and from (5.19) we have

$$y_i((\mathbf{w}^*)^T\mathbf{x}^{(i)} + b^*) \geq 1 \quad i = 1, \ldots, N. \tag{5.24}$$

Inequality (5.24) implies Inequality (5.16). What remains to be shown is that Equation (5.17) holds for $(\mathbf{w}^*, b^*)$. Assume the contrary,

$$\min_{i \in \{1,\ldots,N\}} |(\mathbf{w}^*)^T\mathbf{x} + b^*| = \alpha \neq 1. \tag{5.25}$$

We have two cases. Assume that $\alpha < 1$. Consider the global minimizer i^* of (5.25) with the corresponding data point $\mathbf{x}^*$. Let y^* denote the target corresponding to $\mathbf{x}^*$. Multiplying of Equation (5.25) with $|y^*|$, we have

$$|y^*||(\mathbf{w}^*)^T\mathbf{x}^* + b^*| = |y^*|\alpha. \tag{5.26}$$

Applying the rules of absolute values to (5.26) it must be that

$$|y^*((\mathbf{w}^*)^T\mathbf{x}^* + b^*)| = \alpha < 1. \tag{5.27}$$

This however is contradicting Inequality (5.19) because α is strictly less than 1.

Now assume $\alpha > 1$. Since the hyperplane representation is not unique, we can change $\mathbf{w}^*$ and b^* by $\frac{\mathbf{w}^*}{\alpha}$ and $\frac{b^*}{\alpha}$ respectively. Considering (5.25) with the new hyperplane representation we will have

$$\min_{\{1,\ldots,N\}} \left| \left(\frac{\mathbf{w}^*}{\alpha} \right)^T \mathbf{x}^{(i)} + \frac{b^*}{\alpha} \right| = 1. \tag{5.28}$$

The constraints given by (5.19) is satisfied for $(\frac{\mathbf{w}^*}{\alpha}, \frac{b^*}{\alpha})$ since $\mathbf{x}^*$ is the global minimizer of (5.28). This means that $(\frac{\mathbf{w}^*}{\alpha}, \frac{b^*}{\alpha})$ is feasible, but this contradicts the optimality of $\mathbf{w}^*$, b^* since

$$(\mathbf{w}^*)^T\mathbf{w}^* > (\frac{\mathbf{w}^*}{\alpha})^T \frac{\mathbf{w}^*}{\alpha},$$

and the result follows. □

In the following proposition we only prove one direction of the equivalence as the other direction is proved analogously.

Proposition 5.2. *([86]).*

The point $(\mathbf{w}^*, b^*)$ is optimal for optimization model (5.15) if and only if $(\mathbf{w}^*, b^*)$ is optimal for optimization model (5.18).

Proof. Let $(\mathbf{w}^*, b^*)$ be an optimal solution of optimization model (5.15). Invoking Proposition (5.1), it must be that $(\mathbf{w}^*, b^*)$ is feasible for model (5.18). In proceeding, assume $(\mathbf{w}^*, b^*)$ is not optimal for model (5.18). This implies that there exists $(\hat{\mathbf{w}}, \hat{b})$ that is optimal for model (5.18) and it must be that

$$((\mathbf{w}^*)^T)\mathbf{w}^* > \hat{\mathbf{w}}^T\mathbf{w}. \tag{5.29}$$

Applying Proposition (5.1) to $(\hat{\mathbf{w}}, \hat{b})$, we have that $(\hat{\mathbf{w}}, \hat{b})$ is feasible for model (5.15). This however contradicts the optimality of $(\mathbf{w}^*, b^*)$ with respect to model (5.15) due to Inequality (5.29).

□

It is interesting that the optimization model (5.18) is a quadratic optimization problem with linear constraints. We can then solve the hard-margin SVM optimization model with commercially available quadratic programming solvers.

As described in [49], let $(\mathbf{w}^*, b^*)$ be the optimal solution corresponding to the hard-margin SVM model (5.18). Given a test data point $\hat{\mathbf{x}}$, we can make predictions by looking at the sign of

$$\hat{y} = (\mathbf{w}^*)^T \hat{\mathbf{x}} + b^*. \tag{5.30}$$

The sign of $\hat{y}$ represents the side of the decision boundary (hyperplane) data point $\hat{\mathbf{x}}$ occupies. By considering the sign function given by

$$sign(x) = \begin{cases} 1 & \text{if } x \geq 0, \\ 0 & \text{if } x = 0, \\ -1 & \text{if } x < 0, \end{cases} \tag{5.31}$$

and applying it to $\hat{y}$, we can get the side of the decision boundary, and thus the predicted class for test data point $\hat{\mathbf{x}}$.

Throughout this derivation of the SVM we have assumed that the data points in dataset $\mathcal{D}$ are linearly separable. We will in the following section discard this assumption and see how the SVM can be generalized to where the optimal hyperplane is not necessarily separating all the data points, thus purposefully misclassifying certain data points.

5.3 Soft-Margin SVM

5.3.1 Primal Formulation

Extending the idea of the hard-margin SVM is the work of Corinna Cortes and Vladimir Vapnik in their 1995 paper *Support-Vector Networks* [58]. This work alleviates the SVM from the linearly separable assumption on the dataset. Their paper introduces a way for the SVM to intentionally misclassify training data points. This sacrification of certain data points allows the SVM to potentially find better decision boundaries which means

possibly better generalization to unseen data points. Henceforth, we assume that the dataset $\mathcal{D}$ (5.1) is not necessarily linearly separable.

Introduced in [58] is the use of slack variables $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_N)$ for the hard-margin SVM optimization model (5.18). We have their proposed optimization model,

$$\min \mathbf{w}^T\mathbf{w} + \Gamma \sum_{i=1}^{N} \xi_i \tag{5.32}$$

s.t.

$$y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i \quad i = 1, \ldots, N, \tag{5.33}$$

$$\boldsymbol{\xi} \geq \mathbf{0},$$

$$\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}, \boldsymbol{\xi} \in \mathbb{R}^N.$$

The idea of the slack variables, as described in [69] is to penalize the constraints that are not satisfied by introducing slack variable ξ_i for each Inequality (5.19). If a constraint (5.19) is already satisfied before introducing the slack variable, then the slack variable will have a value 0. If a constraint (5.19) is not satisfied, then we allow slacking of the constraint through the corresponding slack variable. If a constraint (5.19) is violated, then we can always increase the slack variable until the constraint becomes satisfied. The additional summation term in the objective function (5.32) is the accumulation of the slack variables, which can be seen as the total penalty in order to satisfy all the constraints.

In [69], Γ is described as a hyperparameter determining how significant the slacking of the constraints should be. A lower value of Γ implies that the penalty for slacking the constraints is lower, thus misclassifcation is given less importance while maximizing the margin is prioritized. Conversely, for a higher value of Γ it is prioritized to get as many data points correctly classified as possible.

An additional interpretation of how Γ influences the optimization model of (5.32) is provided by [86]. The tuning of Γ can be seen as a reflection of the level of noise presented in the data points in $\mathcal{D}$. Noisy data, as defined by [40], refers to data containing a significant amount of additional, meaningless information known as noise. Such data is a mixture of meaningful information and various forms of interference and errors.

In the context of the SVM, noisy data could manifest as the corruption of a labeled data point $(\mathbf{x}^{(i)}, y_i)$, such as the flipping of target y_i or the perturbation of the coefficients

of the data point $\mathbf{x}^{(i)}$. With the guidelines in [86], in cases where the data has little or no noise, each data point is treated seriously. However, in the presence of noise, as it is often the case in real-world problems, it is preferable to adjust to a lower value of Γ. This adjustment signifies a reduced trust in the labeled data points.

Similar to the hard-margin SVM setting, we use Equation (5.30) and the sign function (5.31) to make predictions on a test data point $\hat{\mathbf{x}}$.

5.3.2 Dual Formulation

In the 1995 paper by Cortes and Vapnik [58], we also come across the dual problem of the soft-margin SVM. Their approach to derive the dual is succinct, assuming the reader is familiar with duality theory. It is worth noting that online searches reveal a common practice of employing simplified or partially complete derivations to streamline the process. Since there is interesting optimization theory in which the dual problem is dependent on, we have decided that we will in this section give a more detailed and complete description of the dual problem derivation. The derivation is a compilation of the following resources: [62], [78] and [49].

To derive the dual problem we will start by revisiting the soft-margin SVM model (5.32) which we considered in the previous section. We make the insignificant change of dividing the objective function in model (5.32) by 2. We have

$$\min \zeta = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \Gamma \sum_{i=1}^{N} \xi_i \tag{5.34}$$

s.t.

$$y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \xi_i \quad i = 1, \ldots, N, \tag{5.35}$$

$$\boldsymbol{\xi} \geq \mathbf{0}, \tag{5.36}$$

$$\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}, \boldsymbol{\xi} \in \mathbb{R}^N.$$

We will refer to model (5.34) as the primal model. Important for deriving the dual is that the primal model (5.34) is a convex optimization problem. First, note that all the constraints in the primal are linear. By [4], this implies that the constraints form a convex set, and the feasible set must therefore be convex. Second, inspecting the derivatives of the objective function, we have

$$D_{\mathbf{w}}\zeta = \mathbf{w}^T, \quad D_b\zeta = 0, \quad D_{\boldsymbol{\xi}}\zeta = (\Gamma \ \ldots \ \Gamma). \tag{5.37}$$

We see that the Hessian of ζ with respect to all the variables is given by

$$\mathcal{H}_\zeta(\mathbf{w}, b, \boldsymbol{\xi}) = \begin{bmatrix} 1 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 \end{bmatrix}. \tag{5.38}$$

Decomposing (5.38), we see that all the elements are zero except for the elements of the n by n upper left submatrix which is the identity matrix. We have that for all $(\mathbf{w}, b, \boldsymbol{\xi}) \in \mathbb{R}^{n+1+N}$,

$$(\mathbf{w}, b, \boldsymbol{\xi})^T \mathcal{H}_\zeta(\mathbf{w}, b, \boldsymbol{\xi})(\mathbf{w}, b, \boldsymbol{\xi}) = \mathbf{w}^T\mathbf{w} \geq 0 \tag{5.39}$$

With the Hessian $\mathcal{H}_\zeta(\mathbf{w}, b, \boldsymbol{\xi})$ being positive semidefinite, the objective function is convex for $\mathbf{w} \in \mathbb{R}^n$, $b \in \mathbb{R}$ and $\boldsymbol{\xi} \in \mathbb{R}^N$. Therefore, it is also convex over the feasible set formed by the constraints, as the feasible set is a subset of $\mathbb{R}^{n+1+N}$.

With optimization model (5.34) being a convex optimization problem, we can apply Proposition 2.5. This allows us to construct and solve the dual optimization model corresponding to model (5.34) instead of the primal model (5.34). As the proposition states, the optimal solution of the dual model corresponds to the Lagrange multipliers of the primal model. Furthermore, it guarantees that the optimal values are equal.

Note that our goal when solving the primal model (5.34) is finding the optimal primal solution that describes the optimal hyperplane. Proposition 2.5 does not guarantee any simple way of obtaining the optimal solution of the primal model (2.6) when we solve the dual model (2.9). As we will see by following the description in [62], when deriving the corresponding dual model from the given primal model (5.34), the retrieval of the primal variables $\mathbf{w}, b$ will be of little concern.

With $\boldsymbol{\alpha} \in \mathbb{R}^N$ and $\boldsymbol{\beta} \in \mathbb{R}^N$ denoting the Lagrange multipliers for the constraints (5.35) and (5.36) respectively, we have the Lagrangian function given by

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \Gamma \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i (y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) + \xi_i - 1) - \sum_{i=1}^{N} \beta_i \xi_i, \quad (5.40)$$

and the dual function given by

$$q(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{\substack{\mathbf{w}\in\mathbb{R}^n, b\in\mathbb{R}, \\ \xi\in\mathbb{R}^N}} \Big\{ \mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \Big\}. \quad (5.41)$$

Observe that for any $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, the infimum on the right-hand side of (5.41) is a convex optimization model. By applying the same line of reasoning as we did to show that the objective function of the primal model (5.34) is a convex function, note that the first derivatives of (5.41) is given by

$$D_{\mathbf{w}}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{w}^T - \sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}^{(i)})^T, \quad (5.42)$$

$$D_b\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = -\sum_{i=1}^{N} \alpha_i y_i, \quad (5.43)$$

$$D_{\boldsymbol{\xi}}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = (\Gamma \dots \Gamma) - \boldsymbol{\alpha}^T - \boldsymbol{\beta}^T. \quad (5.44)$$

Taking the second derivative for (5.42), we will have the identity matrix, while for (5.43) and (5.44) we will have zeros. The Hessian of (5.41) is therefore identical to (5.38) and (5.41) is a convex optimization problem.

With the infimum on the right-hand side of (5.41) being a convex optimization problem given any $\boldsymbol{\alpha}, \boldsymbol{\beta}$, we have from Proposition 2.4 that the necessary optimality condition is also sufficient. Following the approach in [62], we proceed by applying Proposition 2.4 to the infimum on the right-hand side of (5.41) to obtain the sufficient conditions. Taking the derivative of the Lagrangian function (5.40) and equating it to zero we have the

following set of sufficient conditions,

$$D_{\mathbf{w}}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \mathbf{w}^T - \sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}^{(i)})^T = \mathbf{0}^T, \tag{5.45}$$

$$D_b\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{i=1}^{N} \alpha_i y_i = 0, \tag{5.46}$$

$$D_{\xi_i}\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \Gamma - \alpha_i - \beta_i = 0 \quad i = 1, \ldots, N. \tag{5.47}$$

Note that condition (5.45) gives us a way of obtaining $\mathbf{w}$ as a function of $\boldsymbol{\alpha}$. Calculations in [62] show that combining (5.47) and the non-negativity constraints for the Lagrange multipliers $\boldsymbol{\alpha}$ results in the constraints given by

$$0 \leq \alpha_i \leq \Gamma \quad \forall i = 1, \ldots, N. \tag{5.48}$$

Substituting for $\mathbf{w}$ with sufficient condition (5.45) and adding the sufficient conditions given by (5.46) and (5.48) as constraints, algebraic manipulation and simplification as described in [62] yield to the following reduced dual function which only depends on $\boldsymbol{\alpha}$,

$$q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)}. \tag{5.49}$$

We have now arrived at the dual optimization model formulation of primal model (5.34). We have

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)} \tag{5.50}$$

s.t.

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

$$0 \leq \alpha_i \leq \Gamma \quad i = 1, \ldots, N.$$

From [49], we have a description of how to obtain the value of b given that we solve the dual model (5.50). Consider the complementary slackness condition of the Lagrange

multipliers $\boldsymbol{\alpha}, \boldsymbol{\beta}$. If $0 < \alpha_i < \Gamma$ for some $i \in \{1, \ldots, N\}$, then it must be that the i-th constraint (5.35) of the primal model (5.34) is active. We will then have

$$y_i(\mathbf{w}^T\mathbf{x}^{(i)} + b) = 1. \tag{5.51}$$

Solving equation (5.51) for b and substituting for $\mathbf{w}$ using (5.45) we have

$$b = y_i - \sum_{j=1}^{N} y_j \alpha_j (\mathbf{x}^{(j)})^T \mathbf{x}^{(i)}. \tag{5.52}$$

With $\mathbf{w}, b$ both being obtainable using the dual SVM formulation, from [49], we can make predictions on a data point $\hat{\mathbf{x}}$ using

$$\hat{y} = sign\left(\sum_{i=1}^{N} y_i \alpha_i (\mathbf{x}^{(i)})^T \hat{\mathbf{x}} + b \right) \tag{5.53}$$

which is obtained by combining equation (5.45) with the prediction equation (5.30).

By [49], the data points $\mathbf{x}^{(i)}$ with a corresponding positive Lagrange multiplier α_i have been given the unique name of **support vectors**. As described in [60], these data points are called support vectors due to supporting the optimal hyperplane, in the sense that, a change in the support vectors creates a change in the hyperplane. This is reflected by observing that only the Lagrange multipliers with $\alpha_i > 0$ contribute to the value of $\mathbf{w}, b$ in equations (5.45) and (5.52) respectively. To this observation, [60] adds,

> " If you remove all the training data points other than the support vectors, the solution remains unchanged."

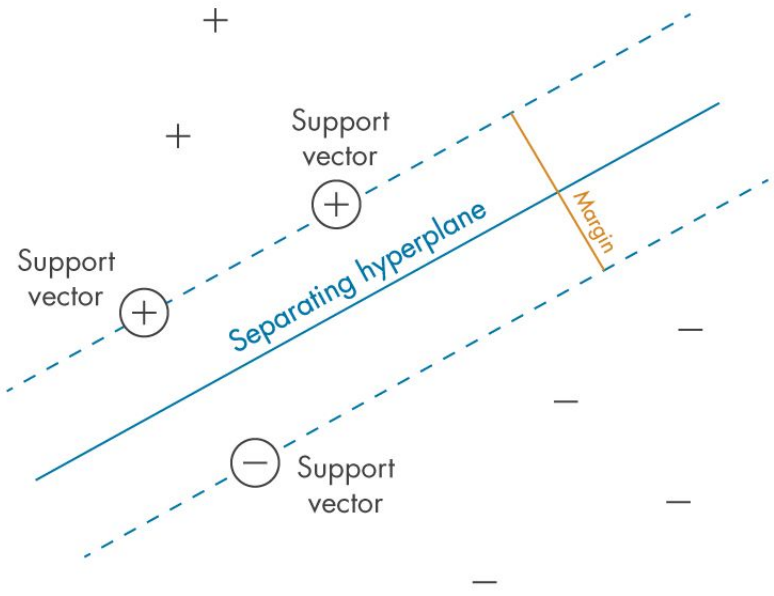

Figure 5.3: The support vectors account for the data points that lie at the margin (as seen in the figure), or within the margin which corresponds to a positive ξ_i.

Figure (taken from [41]).

Just like the primal model (5.34), [49] ensures that the dual model (5.50) is a quadratic programming problem which in general has a computational complexity of $O(N^3)$ where N is the number of decision variables. Moreover, [49] continues that a key characteristic of the dual model (5.50) is that for the primal model (5.34) we have $n+1$ decision variables where n corresponds to the dimension of the data points $\mathbf{x}^{(i)}$ while the dual model (5.50) consists of N decision variables where N corresponds to the number of labeled data points $(\mathbf{x}^{(i)}, y_i)$ in $\mathcal{D}$.

It may thus look disadvantageous to consider the dual problem when the amount of data points is much larger than the dimension of the data points. The dual model does however have another important characteristic. As seen in the objective function of optimization model (5.50) and the retrieval of b in (5.52), the data points $\mathbf{x}^{(i)}$ in the dataset $\mathcal{D}$ arise as the scalar product among themselves. After presenting the concepts of feature mapping and kernels in the subsequent sections, we will see that this characteristic can be explored to formalize the kernel SVM in Section 5.5. With the kernel SVM, we obtain the possibility to map the data into a higher dimensional space, where the dimensionality can far exceed the number of data points without the computational costs.

5.4 Feature Mapping and Kernels

5.4.1 Feature Mapping

In [86] it is mentioned that in ML, it is common to encounter datasets where the data points are distributed in such a way that no adequate linear decision boundary can be found. An ML practitioner using the algorithms discussed in Chapter 4 and 5 might find it challenging to create models that are able to make use of the non-linear structures and patterns of the data in order to make classification or regression conclusions.

Mentioned in [75], an often applied technique in this scenario is the technique called **feature mapping**. This consists of using something called a **basis function** $\phi(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^D$ which usually is constructed with $D > n$ to map the data points $x^{(i)}$ into a higher dimensional space. As explained in [77], having data points residing in a higher dimensional space can potentially lead to uncovering and capturing patterns of non-linearities among the original features of the data.

The following example taken from [38] illustrates benefits of using feature mapping.

Example 5.2.

Consider the data points from $\mathbb{R}^2$ which can be seen on the left side in Figure 5.4. All the $\mathbf{x}^{(i)}$ form a dataset that is neither linearly separable nor will any linear decision boundary suffice in separating the two classes. However, by constructing the basis function

$$\phi(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}^3,$$

given by

$$\phi(\mathbf{x}) = (x_1, x_2, x_1^2 + x_2^2),$$

we are able to encode the Euclidean norm squared as the third component for each data point $\mathbf{x}^{(i)}$. Using basis function ϕ results in the transformed dataset seen on the right side of Figure 5.4. This transformed dataset happens to be linearly separable, which is not always the case, depending on the applied basis function. A sufficiently good decision boundary can now be found by applying an appropriate SL algorithm.

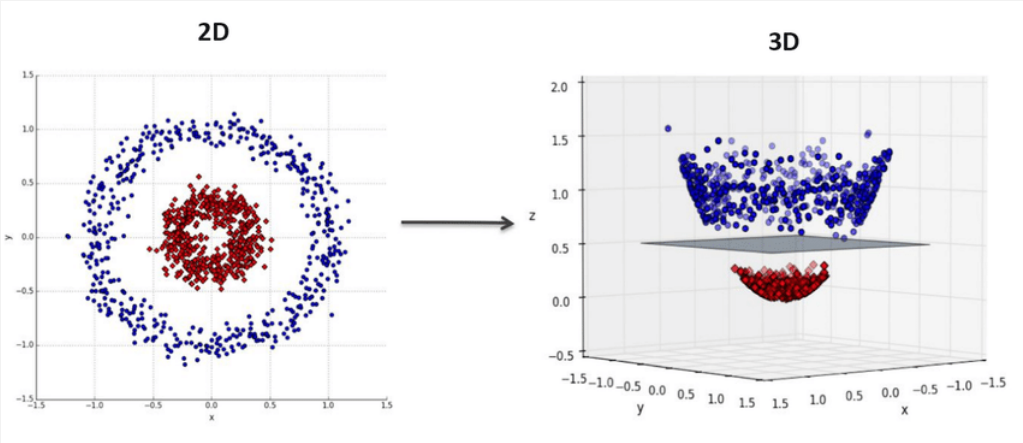

Figure 5.4: Depicted on the left-hand side are the labeled data points $(\mathbf{x}^{(i)}, y_i)$ from a binary classification dataset where $\mathbf{x}^{(i)} \in \mathbb{R}^2$. By using feature mapping, specifically, applying the basis function ϕ to every data point $\mathbf{x}^{(i)}$, we have the transformed labeled data points illustrated on the right-hand side for which a sufficient separating hyperplane can be obtained.

Figure (taken from [93]).

As demonstrated in Example 5.2, using the technique of feature mapping can be beneficial, however, the construction of the basis functions may not be immediately apparent. An initial approach involves manually crafting the basis functions based on intuition gained from inspecting the provided data. While this method is feasible for data in two or three dimensions, [88] notes that it becomes cumbersome or even infeasible when dealing with higher-dimensional data. The challenge lies in intuitively finding meaning in the data when they exist in a higher dimensional space.

To this initial approach of manually crafting the basis functions, [88] provides an answer by considering a specific example which serves to both motivate the use of feature mapping and the introduction of kernels. Due to its effectiveness, we consider this specific example.

Example 5.3. *([88]).*

Consider the basis function $\phi : \mathbb{R}^n \to \mathbb{R}^{(2^n)}$ defined by

$$\phi(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \\ x_1x_2 \\ \vdots \\ x_1x_n \\ \vdots \\ x_2x_3 \\ \vdots \\ x_{n-1}x_n \\ \vdots \\ x_1x_2\dots x_n \end{bmatrix}. \tag{5.54}$$

The basis function ϕ performs a bit-flipping operation on the n original variables $x_1, \dots, x_n$. Each x_i can independently be in an on or off state, and since there are n such variables, the dimensionality of $\mathbf{x}$ after the mapping via ϕ is 2^n.

Discussed in [88] are advantages of using the basis function ϕ in the context of the classification task. Most notably, given the labeled data points $\{\mathbf{x}^{(i)}, y_i\}_{i=1}^K$, after transformation of each data point $\mathbf{x}^{(i)}$, there may exist a hyperplane separating the data points $\phi(\mathbf{x}^{(i)})$ into their respective classes depending on target y_i.

Also observed are downsides to using basis function ϕ, especially, the dimensionality of 2^n. With the dimensionality of the new Euclidean space being exponential in size with respect to n, as n grows, the dimensionality quickly becomes infeasible for computers to handle with respect to computational power and computational memory.

Shifting focus to another aspect highlighted in [88], costly is the process of computing the scalar product between points in the new Euclidean space as it consists of performing 2^n multiplication operations and 2^n addition operations. Nonetheless,

consider the function $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ given by

$$k(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{n} (1 + x_i y_i). \tag{5.55}$$

Expanding the right-hand side of (5.55) we have

$$= (1 + x_1 y_1) \cdot (1 + x_2 y_2) \cdot \ldots \cdot (1 + x_n y_n). \tag{5.56}$$

From the expanded form in (5.56), using the same idea of the flipping operator to define basis function ϕ, we can expand (5.56) further into

$$\begin{aligned} = 1 + x_1 y_1 + \ldots + x_n y_n + x_1 y_1 x_2 y_2 + \ldots + x_1 y_1 x_n y_n + \\ + x_2 y_2 x_3 y_3 + \ldots x_2 y_2 x_n y_n + \ldots x_1 z_1 \ldots x_n y_n. \end{aligned} \tag{5.57}$$

Observing that this is the same as the scalar product between vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ that has been transformed by basis function ϕ we have

$$\prod_{i=1}^{n} (1 + x_i y_i) = \phi(\mathbf{x})^T \phi(\mathbf{y}). \tag{5.58}$$

With (5.55), we can calculate the scalar product between $\phi(\mathbf{x})$ and $\phi(\mathbf{x})$ using a total of $2n$ multiplication operations. This is a reduction in operations compared to calculating the scalar product without (5.55), then consisting of a total of $2 \cdot 2^n$ multiplication and addition operations.

In Example 5.3, the function in (5.55) allows us to calculate the scalar product between vectors $\mathbf{x}, \mathbf{y}$ that has been mapped by (5.54) implicitly, that is, we do not have to first perform the mapping of $\mathbf{x}, \mathbf{y}$, and then calculate the scalar product $\phi(\mathbf{x})^T \phi(\mathbf{x})$.

The function in (5.55) is called a kernel function, and its definition will be provided in the following subsection (see Definition 5.1). Additionally, kernel functions will be further explored in the next section.

Concluding this section, we have observed the existence of so called kernel functions which in the case of Example 5.3 allows for a more efficient way to calculate the scalar product between vectors after the transformation (5.54). Do these formulas always exist, or are there perhaps conditions which need to be met? In the next section we will answer these questions and look into how these mathematical structures work.

5.4.2 Kernel Functions and Kernels

For our discussion on kernels we will follow the definitions in [70] and [10]. We will also further generalize the result in [70] by including the result in [88].

Definition 5.1. *([70]).*

A **kernel function** or **kernel** is a function

$$k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}.$$

If $k(\mathbf{x}, \mathbf{y}) = k(\mathbf{y}, \mathbf{x})$ for all $\mathbf{x}, \mathbf{y}$ in $\mathbb{R}^n$, then we say that k is symmetric.

Definition 5.2. *([10]).*

Let $V = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\}$ be a set of vectors from $\mathbb{R}^n$. We then define the **Gram matrix** G as the n by n matrix with elements given by

$$g_{ij} = (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)},$$

that is,

$$G = \begin{bmatrix} (\mathbf{x}^{(1)})^T \mathbf{x}^{(1)} & \cdots & (\mathbf{x}^{(1)})^T \mathbf{x}^{(n)} \\ \vdots & \ddots & \vdots \\ (\mathbf{x}^{(n)})^T \mathbf{x}^{(1)} & \cdots & (\mathbf{x}^{(n)})^T \mathbf{x}^{(n)} \end{bmatrix}.$$

We now define a positive semidefinite kernel which will be the main machinery when relating kernels to the technique of feature mapping. Definition 5.3 is derived from [70]. It's important to note that we have adapted the definition to apply specifically to positive semidefinite kernels, as opposed to positive definite kernels.

Definition 5.3. *([70]).*

Let $V = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\}$ be a set of vectors from $\mathbb{R}^n$ and $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a symmetric kernel function. Define matrix K analogously to the construction of the Gram matrix, but applying k instead of the scalar product. We have

$$K = \begin{bmatrix} k(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & \cdots & k(\mathbf{x}^{(1)}, \mathbf{x}^{(n)}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}^{(n)}, \mathbf{x}^{(1)}) & \cdots & k(\mathbf{x}^{(n)}, \mathbf{x}^{(n)}) \end{bmatrix}. \tag{5.59}$$

If K is positive semidefinite, then we call it a **positive semidefinite kernel**. Furthermore, if K is positive definite, then we call it a **positive definite kernel**.

The interesting property of a positive semidefinite kernel K as stated in [88], is that the elements in K corresponds to inner products between the vectors from V after some transformation $\mathbf{x} \to \phi(\mathbf{x})$. This idea is proved in [70] for positive definite kernels. By using the same line of reasoning in [70] along with the comments in [88] stating that a positive semidefinite matrix is orthogonally diagonalizable, we are assured that the statement indeed holds for positive semidefinite kernels as well. This is encapsulated in Proposition 5.3.

Proposition 5.3. *([88], [70]).*

Let $V = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}\}$ be a set of vectors from $\mathbb{R}^n$, $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ be a symmetric kernel function and consider matrix K as given in (5.59). If K is positive semidefinite, then there exists a function $\phi : \mathbb{R}^n \to \mathbb{R}^D$ for some $D \in \mathbb{N}$ such that for $\mathbf{x}^{(i)}, \mathbf{x}^{(j)} \in V$ we have for element K_{ij} of the matrix K that

$$K_{ij} = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)}). \tag{5.60}$$

Proof. If K is positive semidefinite, it must be a symmetric matrix with non-negative eigenvalues. As stated in [88], being symmetric implies K is orthogonally diagonalizable. For K we have

$$K = PEP^T \tag{5.61}$$

where P is an orthogonal matrix and E a diagonal matrix of eigenvalues belonging to K. If we now consider an element of K we have

$$K_{ij} = (E^{\frac{1}{2}} P_{i,:})^T (E^{\frac{1}{2}} P_{j,:}) \tag{5.62}$$

with the subscript $P_{i,:}$ to denote the vector containing all elements from row i of P. By defining $\phi(\mathbf{x}^{(i)}) = E^{\frac{1}{2}} P_{i,:}$ for Equation (5.62) we get that

$$K_{ij} = \phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)}), \tag{5.63}$$

□

Combining the line of reasoning in [70] with the results in [88] we have more generally that if K is a positive semidefinite kernel, then there exists a function $\phi : \mathbb{R}^n \to \mathbb{R}^D$ for some $D \in \mathbb{N}$ such that for $\mathbf{x}, \mathbf{x}^{'}$ we have

$$k(\mathbf{x}, \mathbf{x}^{'}) = \phi(\mathbf{x})^T \phi(\mathbf{x}^{'}). \tag{5.64}$$

After having introduced the concepts of feature mapping and kernels, we are now prepared to delve into the discussion of the kernel SVM.

5.5 Kernel SVM

Recall that in the context of the dual SVM optimization model (5.50), we noted that the data points $\mathbf{x}^{(i)}$ appear as scalar products among themselves. This observation applies to the objective function in (5.50), the equation in (5.52) used for calculating b, and the prediction equation (5.53). We will now explore and make use of this property.

Suppose K is a positive semidefinite kernel constructed from the data points $\mathbf{x}^{(i)}$ in $\mathcal{D}$ and some symmetric kernel function $k(\cdot, \cdot)$. Proposition 5.3 ensures that each element K_{ij} of K corresponds to the scalar product between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ after some transformation $\phi : \mathbb{R}^n \to \mathbb{R}^D$. Also consider the dual SVM optimization model (5.50) with the adjustment that we express the scalar product $\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$ between the data points $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ in terms of elements K_{ij} of K. With this modification we have that the kernel SVM optimization model is given as

$$\begin{aligned}
&\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \qquad (5.65)\\
&\text{s.t.}\\
&\sum_{i=1}^{N} \alpha_i y_i = 0,\\
&0 \leq \alpha_i \leq \Gamma \quad i = 1, \ldots, N.
\end{aligned}$$

By replacing $(\mathbf{x}^{(i)})^T \mathbf{x}^{(j)}$ with $K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, we are now implicitly calculating the scalar product $\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$. Doing the same changes to Equation (5.52) used to calculate b

we get

$$b = y_i - \sum_{j=1}^{N} y_j \alpha_j K(\mathbf{x}^{(j)}, \mathbf{x}^{(i)}). \tag{5.66}$$

When predicting with the kernel SVM on a data point $\hat{\mathbf{x}}$ we apply that kernel function used to construct the positive semidefinite kernel K. Also relying on (5.64), we have

$$\hat{y} = sign\left(\sum_{i=1}^{N} y_i \alpha_i k(\mathbf{x}^{(i)}, \hat{\mathbf{x}}) + b \right). \tag{5.67}$$

As stated in [88], the positive semidefinite kernel K is constructed from the data points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ in $\mathcal{D}$ and the kernel function given by $k(\cdot, \cdot)$. With this, K is an (N, N)-matrix with fixed size and fixed elements because the data points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ in the dataset $\mathcal{D}$ are fixed. From an implementation perspective, this allows for the precomputation of matrix K. This results in a significant improvement in computational efficiency because we do not have to recalculate the elements in K, but can store them in memory or storage devices. If we are interested in the value of $\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)})$, then we can query the storage medium and directly retrieve it.

In order to get a better understanding we will consider popular and well-established kernel functions in the following example.

Example 5.4.

For the following kernel functions, we consider $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$.

1. *Linear Kernel Function*:

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'. \tag{5.68}$$

   The linear kernel function is equivalent to calculating the scalar product between $\mathbf{x}$ and $\mathbf{x}'$.

2. *Product Kernel Function*:

$$k(\mathbf{x}, \mathbf{x}') = \prod_{i=1}^{n} (1 + x_i x_i'). \tag{5.69}$$

For a description of the product kernel function, see Example 5.3.

3. *Polynomial Kernel Function*:

$$k_d(\mathbf{x}, \mathbf{x}^{'}) = (\mathbf{x}^T\mathbf{x}^{'} + t)^d. \tag{5.70}$$

with $0 \leq t \in \mathbb{R}$ being a parameter and d being the degree of the polynomial.

4. *Radial Basis Kernel Function*:

$$k(\mathbf{x}, \mathbf{x}^{'}) = e^{-\frac{||\mathbf{x}-\mathbf{x}^{'}||^2}{2\kappa^2}} \tag{5.71}$$

with κ being a parameter. As described in [62], the radial basis kernel function can be thought of as centering a Gaussian distribution with variance κ^2 around each data point. Depending on the label of the data point, these Gaussian distributions can be thought of as either "bumps" or "crevices", either "pushing up" or "pulling down" the surface which separates the two classes.

An interesting property of using the radial basis kernel function in regards to feature mapping is that it implicitly maps the data points into an infinite dimensional space, as described in [47]. By (5.71), using the Taylor expansion of e, we have

$$= e^{-\frac{\mathbf{x}^T\mathbf{x}}{\kappa^2}} \cdot e^{-\frac{\mathbf{x}'^T\mathbf{x}'}{\kappa^2}} \cdot \sum_{t=0}^{\infty} \frac{2^t}{\kappa^{2t} \cdot t!} \cdot (\mathbf{x}^T\mathbf{x}')^t. \tag{5.72}$$

The exponential factors outside of the summation in (5.72) can be treated as a constant. Each term t in the summation in (5.72) can be understood as the polynomial kernel of degree t.

In conducting our numerical experiments in Chapter 7, we will use the equivalent description of the radial basis function given by

$$k(\mathbf{x}, \mathbf{x}^{'}) = \exp\Big(-\gamma||\mathbf{x} - \mathbf{x}^{'}||^2 \Big) \tag{5.73}$$

where $\gamma = \frac{1}{\kappa}^2$. We adapt this formulation because it corresponds with the Scikit-Learn library implementation used in Chapter 7. As stated in [37]:

> "Intuitively, the gamma parameter [γ] defines how far the influence of a single training example [labeled data point $(\mathbf{x}^{(i)}, y_i)$] reaches, with low values meaning 'far' and high values meaning 'close'."

Chapter 6

Hyperparameter Optimization (HPO)

We briefly touched on Hyperparameter Optimization (HPO) in Section 3.2 when discussing the steps of the SL pipeline. Recalling from Section 3.2, the concepts of HPO appears in the model training and model evaluation steps. From this discussion, we understand hyperparameters as parameters belonging to a given ML algorithm. Varying the hyperparameter configuration corresponding to a ML algorithm gives rise to different configurations of the ML algorithm where hyperparameters are regarded as traits that configure the structure of the algorithm. Once a hyperparameter configuration is set, the values remain fixed and do not change, unlike, for example, the training parameters, which are estimated during model training.

The importance of hyperparameters comes from their role in creating diverse configurations for an ML algorithm. Depending on the specific dataset, different configurations can lead to ML models with a varying degree of performance with respect to a given performance metric. Given a dataset, our motivation and understanding of HPO, is to determine which hyperparameter configurations result in the ML model with the highest performance.

In this chapter, we will begin by framing the HPO problem, consisting of model training and evaluation, as a bilevel optimization model. We conclude with Section 6.2 to present well-established HPO algorithms, in the field of ML.

6.1 Optimization Model

The contents discussed throughout this section heavily rely on the work of [72] as the publication presents well-defined optimization models for both model training and model evaluation. We will be relying on both their notation and optimization models. As [72] discusses a particular kind of hyperparameters, which are non-negative regularization hyperparameters, we will make the corresponding changes to promote an optimization model that generalizes to a larger variety of hyperparameters, including integer and categorical values. In doing so, we will adapt the material from [92], which cultivates the idea of describing the HPO model in terms of a **blackbox**. The following quote from [43] efficiently describes a blackbox:

> "In optimization, a blackbox is any process that when provided an input, returns an output, but the inner workings of the process are not analytically available. The most common form of blackbox is computer simulation, but other forms exist, such as laboratory experiments for example."

In the context of HPO, we will use a blackbox to describe the process of inputting a hyperparameter configuration, and having the corresponding output as training parameters obtained after training with the input hyperparameter configuration. We define the blackbox as

$$f : \Lambda \to \mathbb{R}^{\rho} \tag{6.1}$$

where ρ is the number of training parameters and Λ is the parameter space encapsulating all hyperparameter types under consideration which can include real, integer and categorical values. Note that, in the standard case, ρ is equal to the dimensionality of the data points $\mathbf{x}^{(i)}$. However, ρ may differ, depending on techniques such as feature mapping and kernelization (see Section 5.4).

Noteworthy is that there is no unique way to define the blackbox. We can equally describe the entire process of model training and evaluation as a single blackbox, with the input being a hyperparameter configuration and the output being the performance of the corresponding model on unseen data. In choosing the former blackbox definition, we emphasize that the unknown processes take place during training when applying different hyperparameter configurations.

The work in [72], with hyperparameters being non-negative reals, allows for an explicit expression of the corresponding objective functions, granting the use of continuity and derivative properties. According to [72], their optimization model formulation enables the use of continuous optimization methods, such as sequential quadratic programming methods.

In declaring the assumption of a train-validation-test split (see Section 3.2) for a given dataset $\mathcal{D}$, we are ready to describe the optimization model which describes model training and evaluation. We define the loss function $g_{val} : \mathbb{R}^{\rho} \to \mathbb{R}$. There are multiple loss functions to choose from, as seen in [72]. To make it concrete, for the regression task we choose the mean squared error given by

$$g_{val}(\mathbf{w}_{\boldsymbol{\lambda}}) = \sum_{i=1}^{N} (y_i - \hat{y}_i(\mathbf{w}_{\boldsymbol{\lambda}}, \mathbf{x}^{(i)}))^2 \tag{6.2}$$

where $\hat{y}_i(\mathbf{w}_{\boldsymbol{\lambda}}, \mathbf{x}^{(i)})$ is the prediction of the ML model. To clarify, in the case of having a linear regression model, we have

$$\hat{y}_i(\mathbf{w}_{\boldsymbol{\lambda}}, \mathbf{x}^{(i)}) = \mathbf{w}_{\boldsymbol{\lambda}}^T \mathbf{x}^{(i)}. \tag{6.3}$$

Since we covered the binary classification task in Chapter 5, we consider g_{val} defined as

$$g_{\text{val}}(\mathbf{w}_{\boldsymbol{\lambda}}) = -\frac{1}{N} \sum_{i=1}^{N} \delta(y_i, \text{sign}(\hat{y}_i(\mathbf{w}_{\boldsymbol{\lambda}}, \mathbf{x}^{(i)}))) \tag{6.4}$$

where $\delta(a, b)$ is the Kronecker delta function defined as

$$\delta(a, b) = \begin{cases} 1 & \text{if } a = b, \\ 0 & \text{if } a \neq b, \end{cases} \tag{6.5}$$

and $\text{sign}(x)$ defined as

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{if } x < 0. \end{cases} \tag{6.6}$$

Given $\mathbf{w}_{\boldsymbol{\lambda}}$, loss function (6.4) represents the percentage of correct predictions multiplied by negative one. We call the percentage of correct predictions the **accuracy**, as defined in [15]. We have the factor of negative one because we minimize in optimization model (6.7).

We have the optimization model given by

$$\begin{aligned} \min \quad & g_{val}(\mathbf{w}_{\boldsymbol{\lambda}}) \\ \text{s.t.} \quad & \mathbf{w}_{\boldsymbol{\lambda}} \in \arg\min f(\boldsymbol{\lambda}), \\ & \boldsymbol{\lambda} \in \Lambda, \end{aligned} \tag{6.7}$$

where $\mathbf{w}_{\boldsymbol{\lambda}}$ is the optimal training parameter given the hyperparameter configuration $\boldsymbol{\lambda}$, the parameter space for hyperparameters is given by Λ, and f as defined in (6.1). This constitutes the bilevel optimization model from [72], but with a more generalized parameter space represented by Λ, and the use of blackbox f.

The optimization model (6.7), and the problem of HPO in general, have specific important properties to keep under consideration, as highlighted in [92]. They discuss three crucial aspects for the general case of HPO, which we will now present.

The first aspect are the properties of the loss function used during training. It should be noted that for the constructed optimization model (6.7), the training loss function does not explicitly appear, as it is a part of the blackbox f. For a description of where the loss function explicitly appears, please refer to Section 3.2.

As defined in Section 3.2, the training loss function usually takes the given form

$$L_{\text{train}}(\mathbf{w}) = \sum_{(\mathbf{x},y)\in\mathcal{D}_{\text{SL-train}}} h_{\text{train}}(\hat{y}(\mathbf{w},\mathbf{x}),y). \tag{6.8}$$

where $h_{\text{train}}(\hat{y}_i(\mathbf{w},\mathbf{x}^{(i)}),y)$ is a given comparison function considering the predicted value $\hat{y}(\mathbf{w},\mathbf{x}^{(i)})$ and the true value y_i. The properties of L_{train} are highly dependent on the comparison function h_{train} and the expression for $\hat{y}$. As seen in Chapter 4, applying MLE leads to $h(\hat{y}(\mathbf{w},\mathbf{x}),y) = (\hat{y}(\mathbf{w},\mathbf{x}) - y)^2$ and $\hat{y} = \mathbf{w}^T\mathbf{x}$. For this specific case, we have a nicely behaved loss function that is convex and differentiable. This however, is not always the case as described in [92], here is the possibility of the loss function being neither convex nor differentiable. Without convexity, traditional optimization methods are unsuitable, since they may converge to a local minimizer, rather than a global minimizer.

In the absence of differentiability, we will have to rely on **Derivative-Free Optimization (DFO)** and **Blackbox Optimization (BBO)**. As defined in [43], DFO considers itself with algorithms that does not use derivatives, and BBO rely on algorithms that assume either or both the objective function and constraints are given by blackboxes. The strongly suggested comment from [43] states that:

> "if (generalised) gradient information is available, reliable, and obtainable at reasonable cost, then DFO and BBO will almost never outperform modern gradient-based methods."

The lack of differentiability for the loss function L means we will have to settle for possibly inferior results.

The second aspect outlined in [92] is the large variety of hyperparameters, which includes continuous, discrete and categorical hyperparameters. From this, many numerical optimization methods that only aim to tackle numerical or continuous variables are unsuitable for HPO problems. This suggest a need to modify these algorithm, or to discard them all together for other more capable algorithms.

The final aspect of HPO from [92] is that it can be computationally expensive to train ML models. Factors such as the size of the training data or the complexity of a model, for example, the number of training parameters, can impact the computational cost. Additionally, statements from [84] highlight expenses in terms of financial resources and time.

Inspecting the optimization model (6.7), it becomes evident that performing a single evaluation of the objective function requires first training a model with a specified hyperparameter configuration. In retrieving the estimated training parameters, predictions must be made on the validation dataset to obtain the objective function value. Given that the training of certain models can take days or weeks, depending on available hardware as suggested by [84], it becomes vital to employ optimization algorithms suitable for the HPO problem.

With an initial overview of the HPO setting, we are now ready to discuss applicable algorithms for solving the HPO problem. Assuming a blackbox f as described in (6.1) for the objective function in our HPO optimization model (6.7), we have forfeited the use of optimization algorithms that rely on properties like continuity and differentiability. This is reflected in the algorithms in the subsequent sections, as they operate without such assumptions.

6.2 HPO Algorithms

6.2.1 Grid Search

Grid search is a natural and intuitive approach for solving the HPO model (6.7). We adapt the description from [19]. Assuming we have k distinct hyperparameters to tune, by inputting a finite set S_i for each hyperparameter i, grid search creates an n-dimensional grid. Each point on the grid corresponds to a hyperparameter configuration and the total number of points is equal to multiplying the cardinality of each of the k sets for the hyperparameters. The total number of points is equal to

$$\prod_{i=1}^{k} |S_i|. \tag{6.9}$$

With a defined grid, the algorithm systematically picks a hyperparameter configuration, trains the corresponding model, then evaluates the performance of the model. This exhaustive search explores the entire grid, until the best hyperparameter configuration is found.

The strengths of the algorithm, as highlighted in [80], lie mostly in its simplicity to understand and its power of parallelization. The algorithm is non-sequential, meaning that the points on the grid do not depend on each other. The evaluation of the points can, therefore, be run in parallel, e.g., using multiple computers.

Further drawing from the insights of [80], grid search also comes with challenges and drawbacks. Most notably, we have the fact that the algorithm scales poorly. Inspecting (6.9), we observe a rapid increase in the number of evaluations as we raise the number of hyperparameters to tune and the points to evaluate for each hyperparameter. The algorithm is also not data-efficient and runs naively. The evaluation of new hyperparameter configurations does not take into account the results of previous evaluations. Additionally, there is the argument that computing power might be wasted on configurations that do not significantly impact the results. Lastly, there is the challenge of how to define the grid. There is no guarantee that the values selected for the grid are the best. The candidates provided as input for hyperparameters might yield inferior results compared to some other candidates.

In the following example, we make the procedure of grid search more concrete.

Example 6.1. ***Grid Search Procedure***

Suppose we are considering the classification task and we would like to apply the kernel SVM algorithm using the RBF kernel (see Section 5.5). The corresponding hyperparameters would be Γ which is a positive real value that determines the costs of slacking a constraint, and γ which is a positive real value influencing the width of the Gaussians centred around each data point $\mathbf{x}^{(i)}$ in the training dataset.

With hyperparameters Γ and γ, we choose their respective finite sets to be

$$S_1 = \{0.1, 1, 10, 100\} \text{ and } S_2 = (0.1, 0.5, 1, 1.5, 2\}.$$

This establishes a 2-dimensional grid with a total of $|S_1| \cdot |S_2| = 20$ points. By searching for the best hyperparameter configuration, we perform an exhaustive search of the grid and after everything has been evaluated, choose the best configuration.

6.2.2 Random Search

Following the description in [80], closely related to grid search, we have the **random search** algorithm. For discrete hyperparameters, we still supply the algorithm with a finite set containing hyperparameter values, but for continuous hyperparameters we supply a bound for the values. In both cases, we additionally supply probability distributions for the values each hyperparameter can take, the conventional method would be to use uniform distributions, this however, need not to be the correct distributions

To cultivate this understanding, consider the case where we have $n + m$ hyperparameters to be tuned. Let n of the hyperparameters be continuous where the bounds corresponding to hyperparameter i is given by l_i and u_i respectively. We then treat hyperparameter i as a random variable X_i, that is,

$$X_i \sim Uniform(l_i, u_i), \text{ for } i = 1, \ldots, n.$$

Furthermore, let m of the hyperparameters be discrete where possible values for hyperparameter j is represented by the set S_j. Like for the continuous hyperparameters,

we treat each hyperparameter j as a random variable Y_j, we have

$$Y_j \sim DiscreteUniform(S_j), \text{ for } j = 1, \ldots, m.$$

When sampling hyperparameter configurations, we sample from the joint probability distribution given by

$$\boldsymbol{\theta} \sim P(X_1, \ldots, X_n, Y_1, \ldots, Y_m).$$

A common approach when sampling, especially for hyperparameter values that can range in multiple orders of magnitude, is the trick of using log-uniform sampling. Described in [80], this trick aims to better explore the parameter space. Assuming a continuous hyperparameter spans a large range of values, it can be more effective to explore this range when sampling from the log-uniform distribution instead of the uniform distribution. By implementing a log-uniform sampling, there is a more equal probability of exploring the different orders of magnitude (see Example 6.2).

Example 6.2. ***Using Log-Uniform Sampling***

Suppose we wish to apply ridge regression and optimize for the best hyperparameter Ψ, which is a non-negative real value (see Section 4.3). In assuming no prior knowledge for a good value for Ψ, we input the interval $I = [0.1, 100]$ to random search.
If we were to sample uniformly from I, the probability of sampling from the range $[0.1, 1]$ is a lot lower than the range $[1, 100]$ because $[1, 100]$ is much larger in scale. In applying the log-uniform distribution, in this case, transformation function $f(x) = \log_{10}(x)$, we take the logarithm of the lower and upper bounds of I. This results in the new log space interval $\hat{I} = [-1, 2]$. We observe the following:

(i) The log space interval $[-1, 0]$ corresponds to the linear space interval $[0.1, 1]$.

(ii) The log space interval $[0, 1]$ corresponds to the linear space interval $[1, 10]$.

(iii) The log space interval $[1, 2]$ corresponds to the linear space interval $[10, 100]$.

With observations $(i) - (iii)$, by sampling from $\hat{I}$ and transforming the sample into linear space using $r(x) = 10^x$, we equally explore the intervals $[0.1, 1]$, $[1, 10]$, and $[10, 100]$. This trick enables equal exploration among intervals of different orders of magnitude.

Random search shares many of the same advantages and drawbacks as grid search which are listed in [80]. The most notable benefit of random search over grid search is that there is no grid of points to restrict ourselves to. Instead, random search can freely sample from the parameter space, determined by the input bounds.

As highlighted in Figure 6.1 from [46], the comparison between grid search and random search is noticeable in the cases where one of the hyperparameters is more important in terms of model performance than the other. As the value for the less important hyperparameter has little impact, it would be beneficial to spend time and computational power on exploring different values of the important hyperparameter. Both algorithms do a total of 9 evaluations, however, random search allows for testing a broader range of important hyperparameter values as it is not restricted to points on a grid. This exact property makes random search more capable of spotting hyperparameter values that are sensitive. Being sensitive, they might not get selected using the predefined grid of grid search, but by randomly sampling, they have a higher probability of being selected.

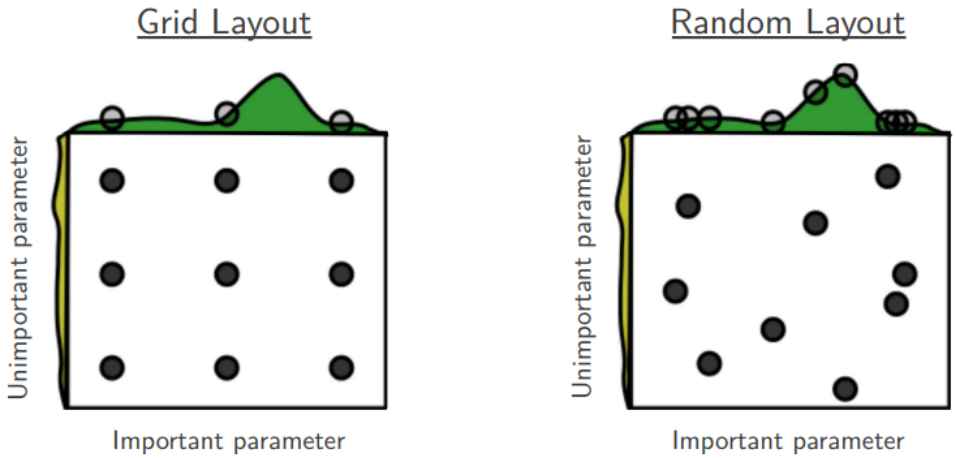

Figure 6.1: Illustrated are the hyperparameter configurations evaluated by grid search and random search within the same parameter space. The green distribution overlaid on the figures indicates model performance, with higher values being better. The figure captures the concept that random search explores more of the relevant parameter space, especially when hyperparameters have a varying degree of importance with respect to model performance.

Figure (taken from [46]).

Also, being of a non-sequential nature, random search has the potential for parallelization. Nevertheless, the algorithm remains unsophisticated and naive, as it is not data-efficient, computationally expensive to perform, and may potentially evaluate hyperparameter configurations that have little importance.

Example 6.3 aims to make the use of random search more clear.

Example 6.3. ***Random Search Procedure***

Consider the setting of Example 6.1, but using random search instead. Thus, we have hyperparameters Γ and γ to optimize. Suppose we have confidence that the optimal value for Γ lies in the interval $I_\Gamma = [1, 10]$, but we are less sure about the optimal value for γ, leaving us with the broader interval of $I_\gamma = [0.001, 1]$.

Since I_γ scales multiple orders of magnitude, we use log-uniform sampling. With $f(x) = \log_{10}(x)$, we have the log space interval $\hat{I}_\gamma = [-3, 0]$.

To match with the grid search example in Example 6.1, we allow random search to make a total of 9 evaluations.

Chapter 7

Numerical Experiments

In this chapter, we adopt an empirical perspective to assess the performance of the discussed SL algorithms. This chapter aims to provide concrete examples that solidify and support the theoretical framework.

To start, we will provide a description and reasoning for the selection of datasets for both the classification and regression tasks. Furthermore, we will detail the programming environment, including the programming language, libraries, and other relevant information that could impact the experiments.

Our focus will be on the SL algorithms: linear regression and SVM. We will look into the individual characteristics of each algorithm. Evaluating these algorithms across different datasets will reveal their strengths and weaknesses. We will not optimize for the hyperparameters, but rather see how models behave with different hyperparameter configurations.

7.1 Datasets and Programming Setup

In conducting our experiments, we will apply a train-validation-test split to the datasets using the `train_test_split` function from Scikit-Learn (for function details, refer to [31]). With the chosen function parameters summarized in Table 7.2, we split $\mathcal{D}_{\text{SL}}$ into $\mathcal{D}_{\text{SL-train}}$, $\mathcal{D}_{\text{SL-val}}$ and $\mathcal{D}_{\text{SL-test}}$ with a data ratio of 70%, 15% and 15% respectively.

Additionally, we will perform standardization of the datasets. For the regression datasets we perform standardization for $\mathbf{x}^{(i)}$ and y_i. For the classification datasets we

only standardize $\mathbf{x}^{(i)}$. In standardizing the data, we calculate the sample mean and sample variance only from dataset $\mathcal{D}_{\text{SL-train}}$. This is because the data in $\mathcal{D}_{\text{SL-val}}$ and $\mathcal{D}_{\text{SL-test}}$ are to remain unseen for the SL models, in the sense that they are not part of model training. As the SL models make predictions on an unseen data point $\hat{\mathbf{x}}$ such as the data points in $\mathcal{D}_{\text{SL-val}}$ and $\mathcal{D}_{\text{SL-test}}$, we standardize $\hat{\mathbf{x}}$ using the sample mean and variance calculated from the data points in $\mathcal{D}_{\text{SL-train}}$.

In the following subsections we discuss the datasets chosen for our numerical experiments. Thereafter, we give a description of the programming setup, including tables of functions and their corresponding parameter settings. This summary aims to simplify the reproduction of the results in this chapter.

7.1.1 Regression Datasets

Note that for the regression datasets we will consider, the independent variable $\mathbf{x}$ is a scalar. We will therefore denote the independent variable and a data point by x and x_i respectively.

The first regression dataset, called $\mathcal{R}_1$ is generated using the function from Scikit-Learn called `make_regression`. For a description of the function, refer to [28]. For the function parameter settings used in our experiments, please consult Table 7.3.

Using `make_regression`, we generate a linear dataset where the true underlying function is given by

$$y(x) = ax + b \tag{7.1}$$

with $a = 41.74110031$ and $b = 30$. Dataset $\mathcal{R}_1$ consists of 100 labeled data points (x_i, y_i) with $x_i, y_i \in \mathbb{R}$. Each y_i has been perturbed by some Gaussian noise $\epsilon_i \sim \mathcal{N}(0, 50^2)$. Dataset $\mathcal{R}_1$ is illustrated in Figure 7.1.

The second regression dataset called $\mathcal{R}_2$ has a non-linear structure. The dataset is generated by sampling from the true underlying function given by

$$y(x) = x^3 \cdot \sin(x). \tag{7.2}$$

Dataset $\mathcal{R}_2$ consists of 100 labeled data points (x_i, y_i) with $x_i, y_i \in \mathbb{R}$. To introduce noise for each labeled data point (x_i, y_i), during the sampling process each y_i has been perturbed by $\epsilon_i \sim \mathcal{N}(0, 90^2)$. In sampling x_i, we uniformly sample from the interval $[0, 8]$. Dataset $\mathcal{R}_2$ is illustrated in Figure 7.2.

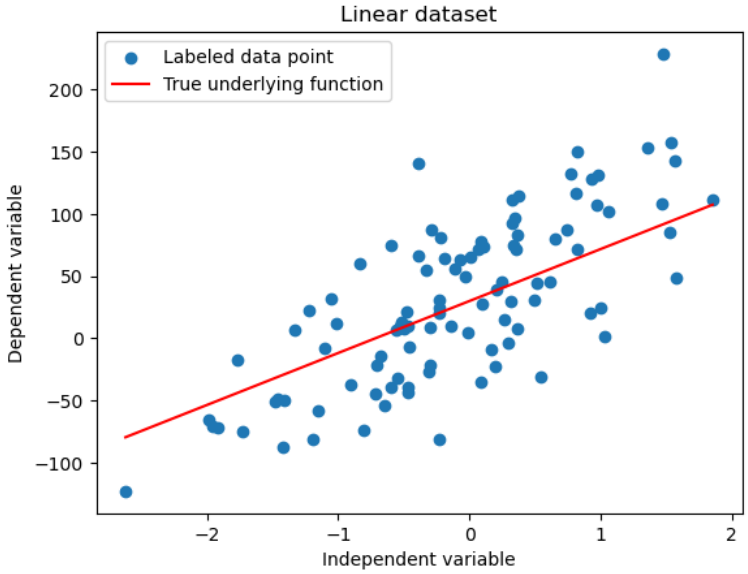

Figure 7.1: Depicted are the labeled data points of $\mathcal{R}_1$ before performing standardization and a train-validation-test split of the dataset. The red line indicates the true underlying function.

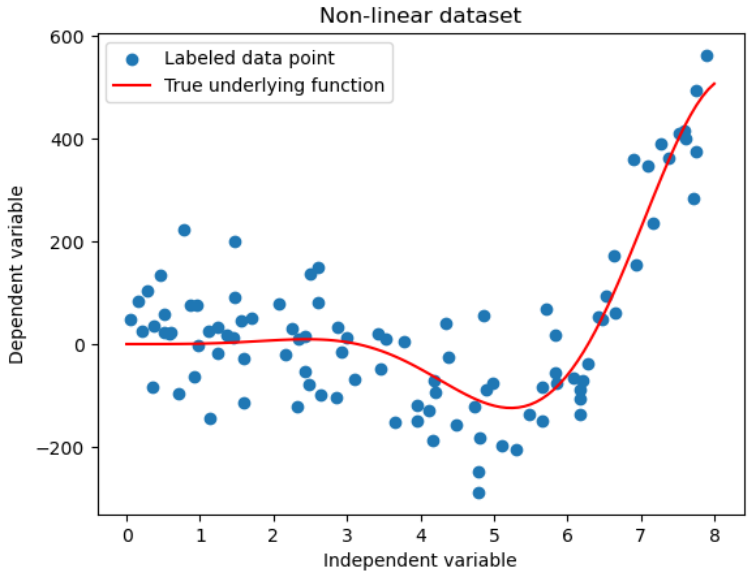

Figure 7.2: Depicted are the labeled data points of $\mathcal{R}_2$ before performing standardization and a train-validation-test split of the dataset. The red curve indicates the true underlying function.

7.1.2 Classification Datasets

Famously known, as described in [17], **Fisher's Iris dataset**, or simply, the **Iris dataset**, is a classification dataset consisting of 50 samples for each of the three Iris species: Iris setosa, Iris virginica, and Iris versicolor. For each sample, there are four features measured: the length and width of the sepals and petals, in centimeters.

For our setup, we turn the dataset into a binary dataset by only considering the species Iris setosa and Iris versicolor. Furthermore, we apply dimensionality reduction utilizing the method of PCA. This reduces the features (dimensions) of $\mathbf{x}^{(i)}$ from four to two. This makes the dataset appropriate for 2-dimensional visualization.

The Iris dataset before performing preprocessing is depicted in Figure 7.3. For simplicity, we will refer to the preprocessed Iris dataset simply as the Iris dataset. Fisher's Iris dataset is publicly available in [61].

The **two moons dataset** is a synthetic dataset generated using the Scikit-Learn function called `make_moons`. Documentations for the `make_moons` function can be found in [39]. For the function parameter settings used in our experiments, please consult Table 7.4.

As seen in [39], the `make_moons` function is utilized to generate datasets for classification tasks, enabling the evaluation of SL algorithms. The datasets generated by the `two_moons` function consist of data that follows a non-linear pattern. This is because the two moons dataset consists of two interleaving half-moon shapes, each representing a distinct label. Each data point $\mathbf{x}^{(i)}$ is characterized by two features. Thus, the data points reside in a 2-dimensional plane and can be visually illustrated. To sufficiently classify the data points from this dataset, the decision boundary need to have non-linear curvatures.

We will consider the generated two moons dataset depicted in Figure 7.4. The generated dataset has a total of 1000 labeled data points.

Figure 7.3: Depicted are the labeled data points of Fisher's Iris dataset after performing dimensionality reduction and standardization.

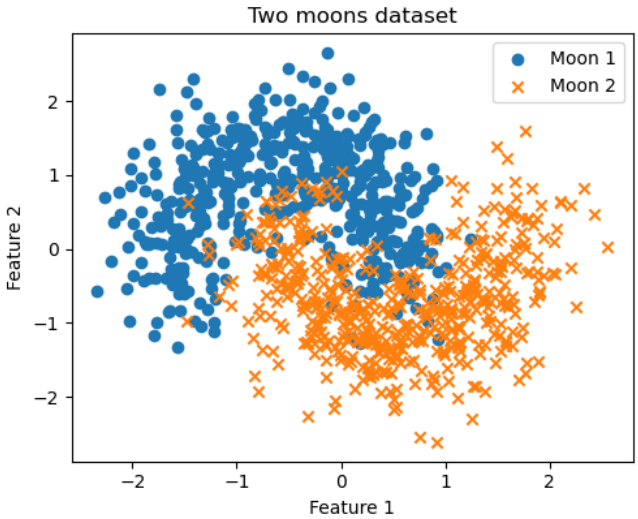

Figure 7.4: Depicted are the labeled data points of the two moons dataset after performing standardization.

7.1.3 Programming Setup

To conduct our investigations we rely on Python version 3.10.11 along with the use of the programming libraries listed in Table 7.1.

Programming Library	Version
Numpy	1.24.2
Matplotlib	3.8.0
Pandas	2.1.1
Scikit-Learn	1.3.0
Scikit-Optimize	0.9
JupyterLab	4.0.8

Table 7.1: Programming libraries with their corresponding version.

In the following we have tables of functions we use for setting up our numerical experiments. Each table has the applied parameter values corresponding to the function. If a parameter value has not been specified, the default value has been used (see the corresponding citation in each table for the default parameter value).

We would also like to clarify that the PCA algorithm and the SL algorithms applied in our experiments are algorithm implementations from the Scikit-Learn library. For information on the implementation and usage of the SL algorithms, please refer to the Scikit-Learn documentation in [29]. The parameter settings for the SL algorithms can be found in Section 7.2.

train_test_split	
Parameter name	**Parameter value**
shuffle	True
random_state	42
test_size	0.3 (for train / val-test split) 0.5 (for val / test split)

Table 7.2: Parameter values used for the `train_test_split` function in [31].

make_regression	
Parameter name	**Parameter value**
n_samples	100
n_features	1
noise	50
bias	30
random_state	42

Table 7.3: Parameter values used for the `make_regression` function in [30].

make_moons	
Parameter name	**Parameter value**
n_samples	1000
shuffle	True
noise	0.3
random_state	42

Table 7.4: Parameter values used for the `make_moons` function in [39].

7.2 SL Algorithms

7.2.1 Linear Regression

With linear regression, we will consider the methods of MLE and MAP. For both methods, we will experiment with one model using feature mapping and one without. For the hyperparameter Ψ corresponding to the method of MAP (see Section 4.3), we will experiment with different values to gain a better understanding of how the hyperparameter influences the models.

To better illustrate the differences between the use of MLE and MAP, we construct the basis functions $\phi_{\mathcal{R}_1}$ and $\phi_{\mathcal{R}_2}$ for $\mathcal{R}_1$ and $\mathcal{R}_2$ respectively. For $\mathcal{R}_1$ we consider the basis function given by

$$\phi_{\mathcal{R}_1}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^{10} \end{bmatrix} \tag{7.3}$$

and for $\mathcal{R}_2$, we use the basis function given by

$$\phi_{\mathcal{R}_2}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^{20} \end{bmatrix}. \tag{7.4}$$

In performing the train-validation-test split, we split $\mathcal{R}_1$ into $\mathcal{R}_{\text{1-train}}$, $\mathcal{R}_{\text{1-val}}$ and $\mathcal{R}_{\text{1-test}}$. Dataset $\mathcal{R}_2$ is split into $\mathcal{R}_{\text{2-train}}$, $\mathcal{R}_{\text{2-val}}$ and $\mathcal{R}_{\text{2-test}}$. Furthermore, the loss function for linear regression using the method of MLE and MAP can be found in optimization models (4.14) and (4.47) respectively. As stated in [90], a common evaluation loss function for regression tasks is the loss function of MSE. We will therefore apply the loss function of MSE for model evaluation when using both methods of MLE and MAP. Note that all upcoming MSE values are rounded to two decimal digits after the comma.

We are now set to conduct our numerical experiments for linear regression.

Consider dataset $\mathcal{R}_1$. Recall that the dataset is constructed with the true underlying function being linear. This suggests that applying MLE and MAP without performing feature mapping should be sufficient for modelling the data in the dataset. The result for MLE without feature mapping for dataset $\mathcal{R}_1$ is illustrated in Figure 7.5. Furthermore, illustrated in Figure 7.6 and 7.7 is the use of MAP with the respective hyperparameter values of $\Psi = 10^{-2}$ and $\Psi = 10^{2}$. We observe that Figure 7.7 with $\Psi = 10^{2}$ depicts a greater penalization of $\mathbf{w}$ compared to Figure 7.6 with $\Psi = 10^{-2}$, as evidenced by the steeper slope of the regression line in the former.

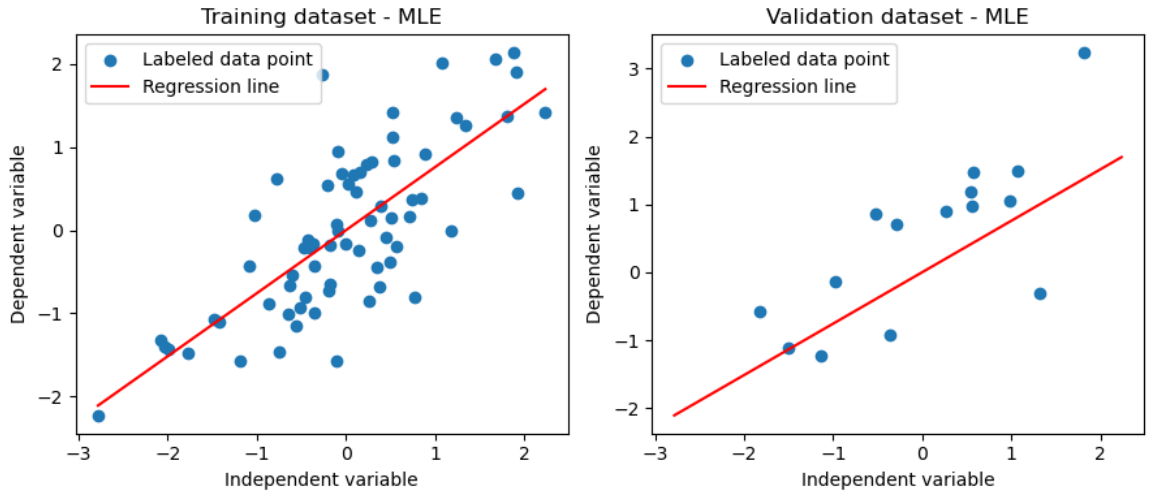

Figure 7.5: The use of MLE without feature mapping is demonstrated on datasets $\mathcal{R}_{\text{1-train}}$ and $\mathcal{R}_{\text{1-val}}$, resulting in MSE values of 0.43 and 0.80 respectively.

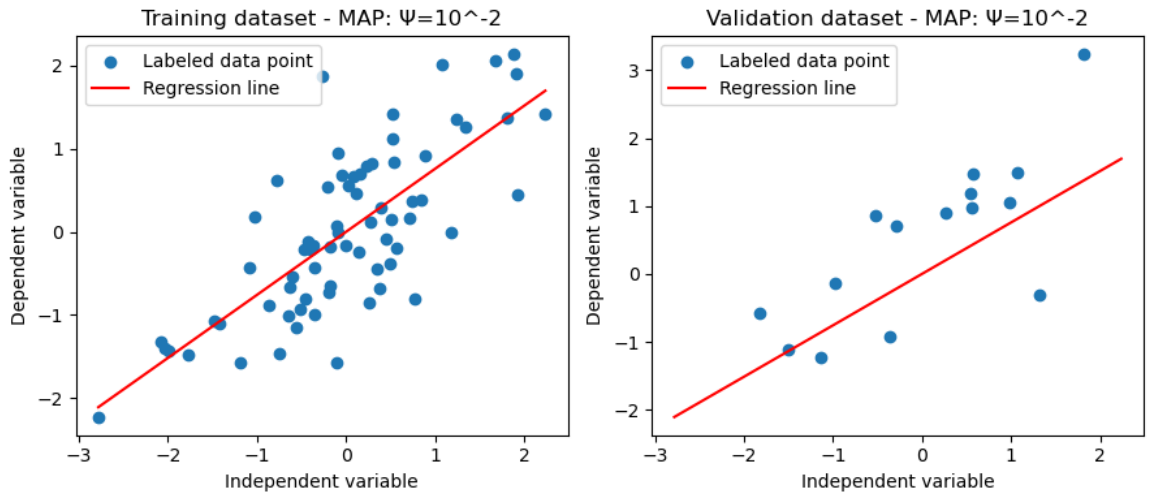

Figure 7.6: The use of MAP without feature mapping is demonstrated on datasets $\mathcal{R}_{\text{1-train}}$ and $\mathcal{R}_{\text{1-val}}$ with a hyperparameter value of $\Psi = 10^{-2}$. This results in MSE values of 0.43 and 0.80 for the training and validation datasets respectively.

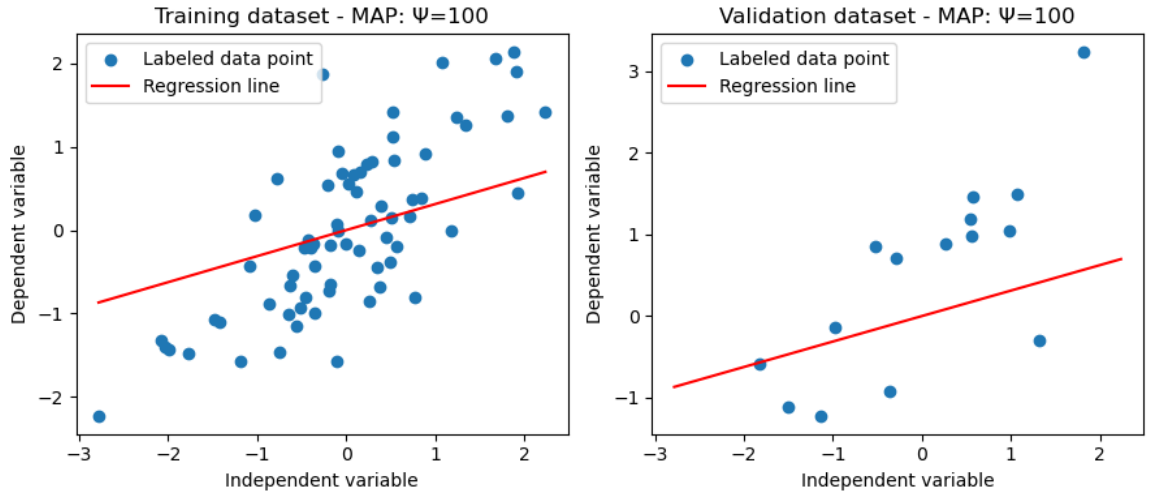

Figure 7.7: The use of MAP without feature mapping is demonstrated on datasets $\mathcal{R}_{\text{1-train}}$ and $\mathcal{R}_{\text{1-val}}$ with a hyperparameter value of $\Psi = 10^{2}$. This results in MSE values of 0.62 and 1.13 for the training and validation datasets respectively.

Consider now applying basis function $\phi_{\mathcal{R}_1}$ to the data points x_i in $\mathcal{R}_1$. The result for MLE with $\phi_{\mathcal{R}_1}$ for dataset $\mathcal{R}_1$ is illustrated in Figure 7.8. Recall from Section 4.2 that when applying MLE to linear regression, we optimize for the training parameters $\mathbf{w}$ that maximizes the likelihood of the observed data as seen in optimization model (4.8). This means that the $\hat{\mathbf{w}}$ corresponding to the regression curve in Figure 7.8 represent the training parameters that maximizes the likelihood of observing the data in $\mathcal{R}_1$, that is, $P(\mathcal{R}_1|\hat{\mathbf{w}})$.

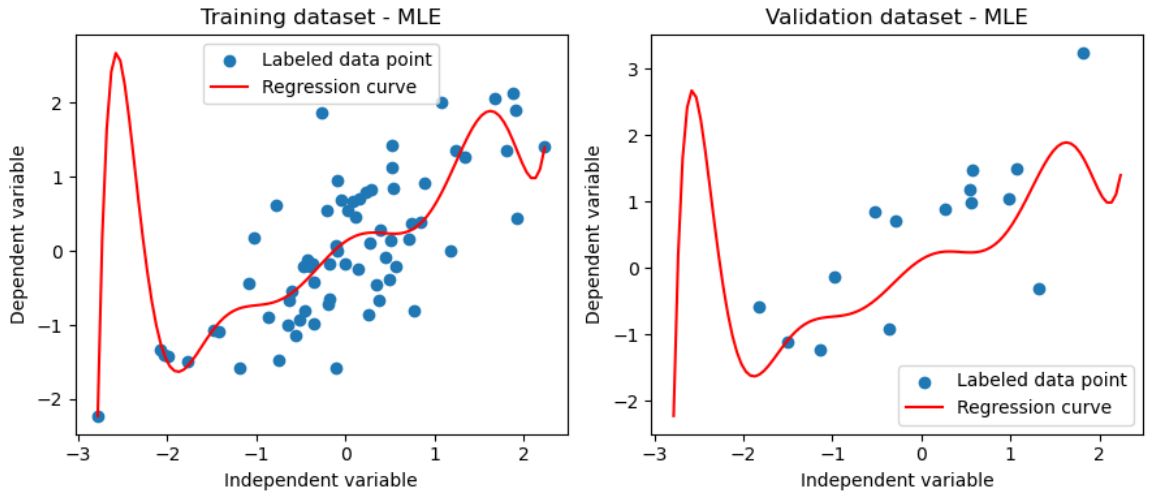

Figure 7.8: The use of MLE with feature mapping is demonstrated on datasets $\mathcal{R}_{\text{1-train}}$ and $\mathcal{R}_{\text{1-val}}$, resulting in MSE values of 0.40 and 0.94 respectively.

Illustrated in Figure 7.9 and 7.10 is the use of MAP with basis function $\phi_{\mathcal{R}_1}$ and the respective hyperparameter values of $\Psi = 10^{-1}$ and $\Psi = 10$. In using MAP, we have the prior assumption that the training parameters $\mathbf{w}$ follow a multivariate Gaussian distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$ with $\boldsymbol{\Sigma}$ being a diagonal matrix with positive coefficients.

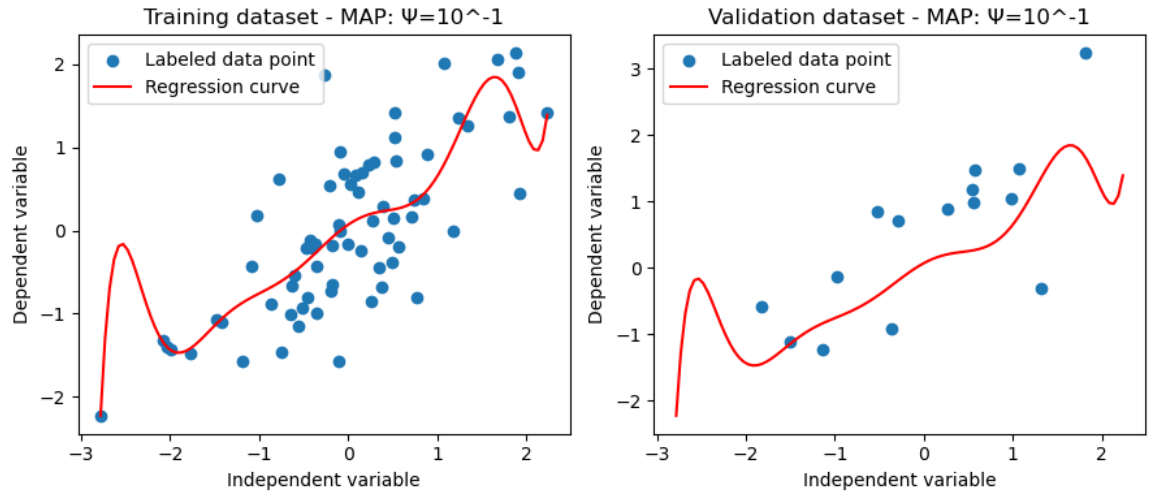

Figure 7.9: The use of MAP with feature mapping is demonstrated on datasets $\mathcal{R}_{\text{1-train}}$ and $\mathcal{R}_{\text{1-val}}$ with a hyperparameter value of $\Psi = 10^{-1}$. This results in MSE values of 0.40 and 0.88 for the training and validation datasets respectively.

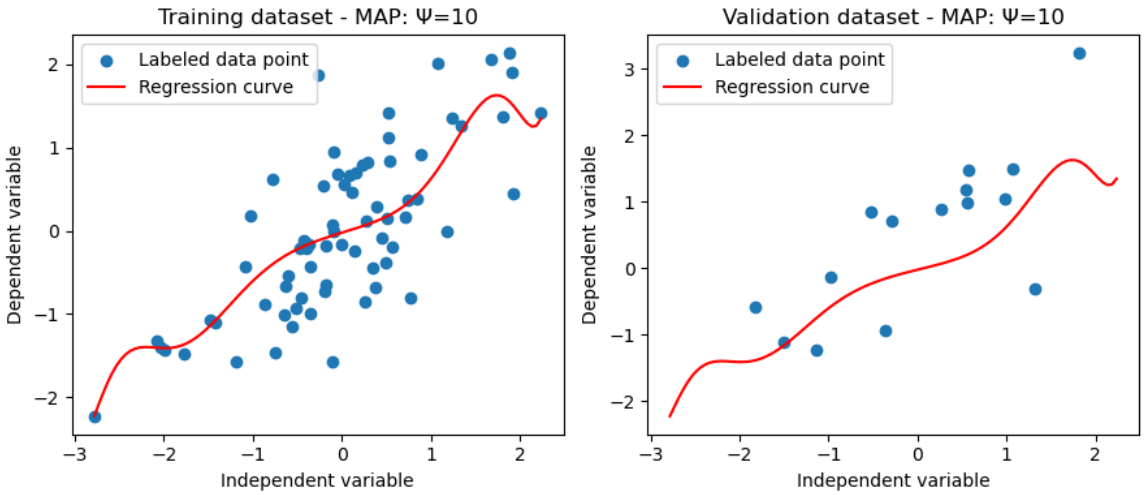

Figure 7.10: The use of MAP with feature mapping is demonstrated on datasets $\mathcal{R}_{1\text{-train}}$ and $\mathcal{R}_{1\text{-val}}$ with a hyperparameter value of $\Psi = 10$. This results in MSE values of 0.42 and 0.86 for the training and validation datasets respectively.

Now, consider dataset $\mathcal{R}_2$. Recall that the dataset is constructed with the true underlying function being non-linear. Contrary to dataset $\mathcal{R}_1$, this suggests that applying MLE and MAP without performing feature mapping may not be sufficient for modelling the data in the dataset. The results for MLE and MAP without feature mapping on dataset $\mathcal{R}_1$ are illustrated in Figure 7.11 and Figure 7.12, respectively. As seen, the depicted regression lines for both figures are not sufficiently capable of modelling the non-linearity of the data, as the true underlying function seen in Figure 7.2 is non-linear.

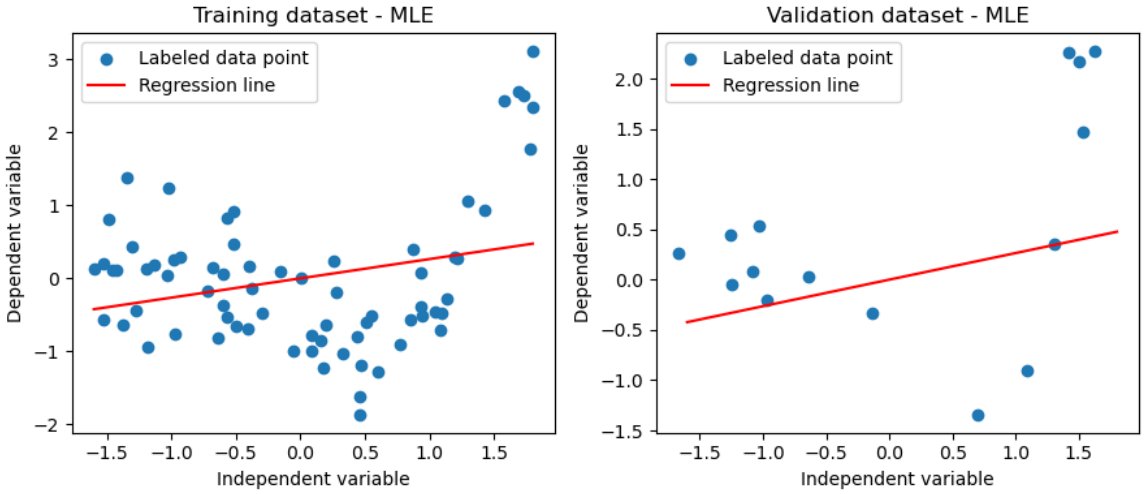

Figure 7.11: The use of MLE without feature mapping is demonstrated on datasets $\mathcal{R}_{2\text{-train}}$ and $\mathcal{R}_{2\text{-val}}$, resulting in MSE values of 0.93 and 1.14 respectively.

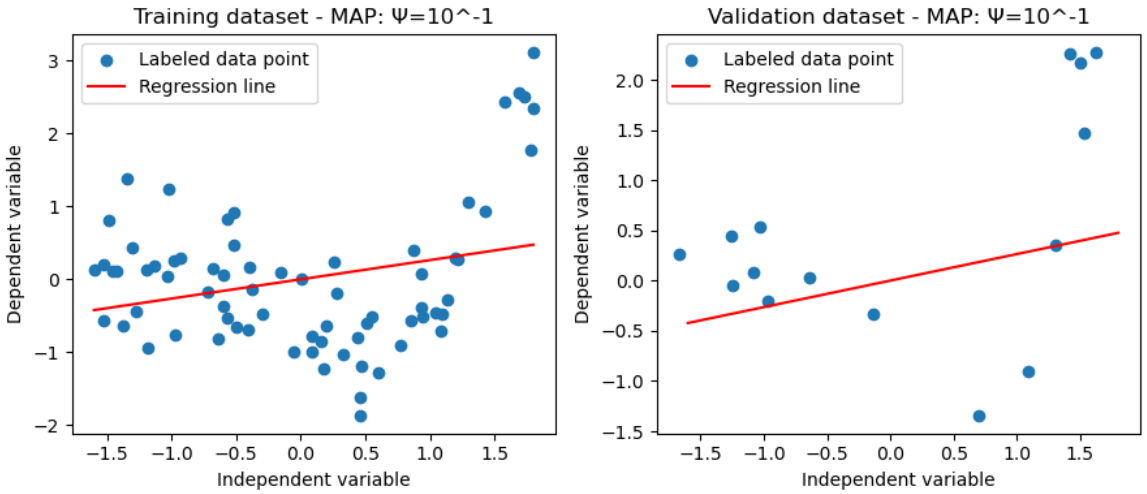

Figure 7.12: The use of MAP without feature mapping is demonstrated on datasets $\mathcal{R}_{2\text{-train}}$ and $\mathcal{R}_{2\text{-val}}$ with a hyperparameter value of $\Psi = 10^{-1}$. This results in MSE values of 0.93 and 1.14 for the training and validation datasets respectively.

Analogous to our experiments with dataset $\mathcal{R}_2$, we apply MLE and MAP utilizing the basis function $\phi_{\mathcal{R}_2}$. The result for MLE with $\phi_{\mathcal{R}_2}$ for dataset $\mathcal{R}_2$ is illustrated in Figure 7.13. The results of applying MAP with feature mapping and the respective hyperparameter values of $\Psi = 10^{-2}$ and $\Psi = 10$ is shown in Figure 7.14 and 7.15. As we increase the value of Ψ, we observe a decrease in fluctuation for the regression curve along with a decrease in the value of MSE for the validation dataset.

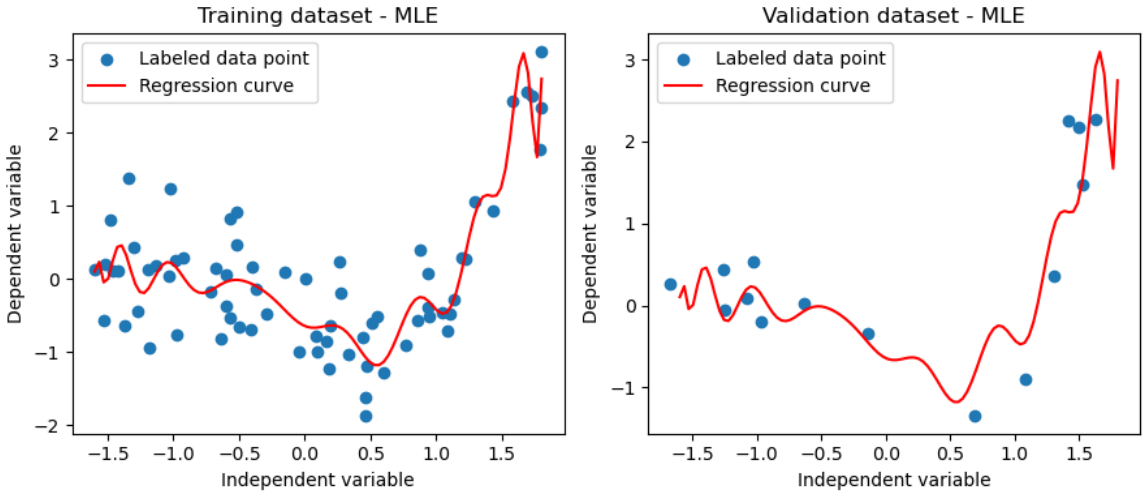

Figure 7.13: The use of MLE with feature mapping is demonstrated on datasets $\mathcal{R}_{2\text{-train}}$ and $\mathcal{R}_{2\text{-val}}$, resulting in MSE values of 0.23 and 41.81 respectively.

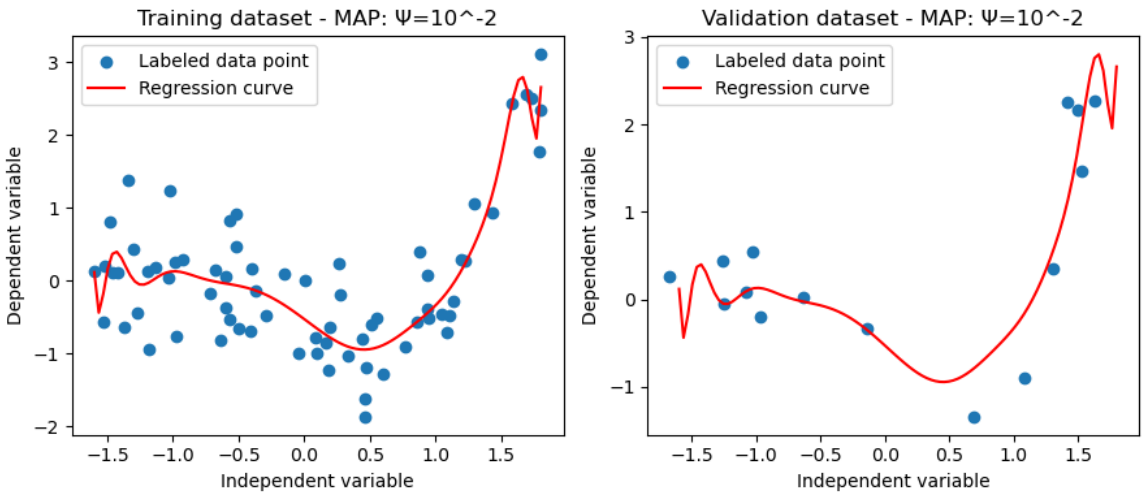

Figure 7.14: The use of MAP with feature mapping is demonstrated on datasets $\mathcal{R}_{\text{2-train}}$ and $\mathcal{R}_{\text{2-val}}$ with a hyperparameter value of $\Psi = 10^{-2}$. This results in MSE values of 0.25 and 11.68 for the training and validation datasets respectively.

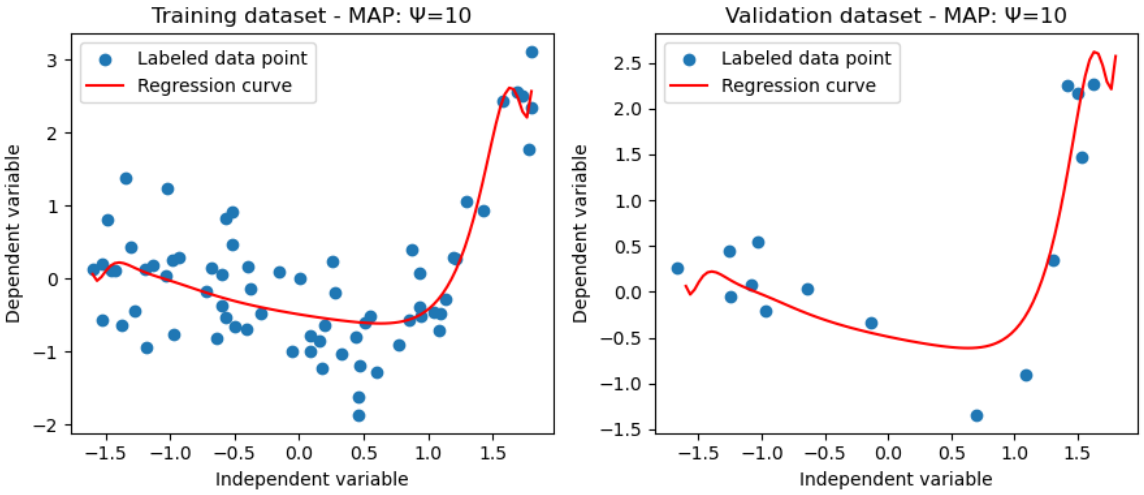

Figure 7.15: The use of MAP with feature mapping is demonstrated on datasets $\mathcal{R}_{\text{2-train}}$ and $\mathcal{R}_{\text{2-val}}$ with a hyperparameter value of $\Psi = 10$. This results in MSE values of 0.29 and 0.40 for the training and validation datasets respectively.

7.2.2 Soft-Margin SVM

In this subsection we will apply the soft-margin SVM to the Iris dataset and the two moons dataset. For each dataset we will apply the soft-margin SVM without using feature mapping to get an understanding of the algorithm in its original state. Furthermore, we will be applying the technique of feature mapping to gain better insight into the technique and to observe the influence of various values of the hyperparameter Γ. For a description

of Γ and soft-margin SVM, please refer to Section 5.3. In our experiments we will use the accuracy metric for both training and evaluation. This corresponds to using loss function (6.4) for optimization model (6.7). Recall that this loss function (6.4) represents the percentage of correct predictions multiplied by negative one. Note that all upcoming accuracy values are rounded to two decimal digits after the comma.

As presented in Section 5.3, applying soft-margin SVM corresponds to solving the optimization model (5.32). Consequently, the algorithm searches for a hyperplane that maximizes the margin, but allow for certain degrees of misclassifiactions of labeled data points $(\mathbf{x}^{(i)}, y_i)$ depending on the hyperparameter Γ. In the case of applying feature mapping to the data points $\mathbf{x}^{(i)}$, the algorithm searches for a hyperplane that maximizes the margin in the new Euclidean space in which the data points are mapped to.

In applying feature mapping we will consider the basis functions given by

$$\phi_{\mathcal{A}_1}(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix} \tag{7.5}$$

and

$$\phi_{\mathcal{A}_2}(\mathbf{x}) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ x_2^2 \\ \vdots \\ x_2^3 \end{bmatrix}. \tag{7.6}$$

We are now ready to examine the results for the soft-margin SVM.

Consider the Iris dataset. In Figure 7.16 we have the decision boundary corresponding to $\Gamma = 1$ without using feature mapping. What we want to illustrate is that with the

Iris dataset being linearly separable, the soft-margin SVM finds a hyperplane perfectly separating the two classes.

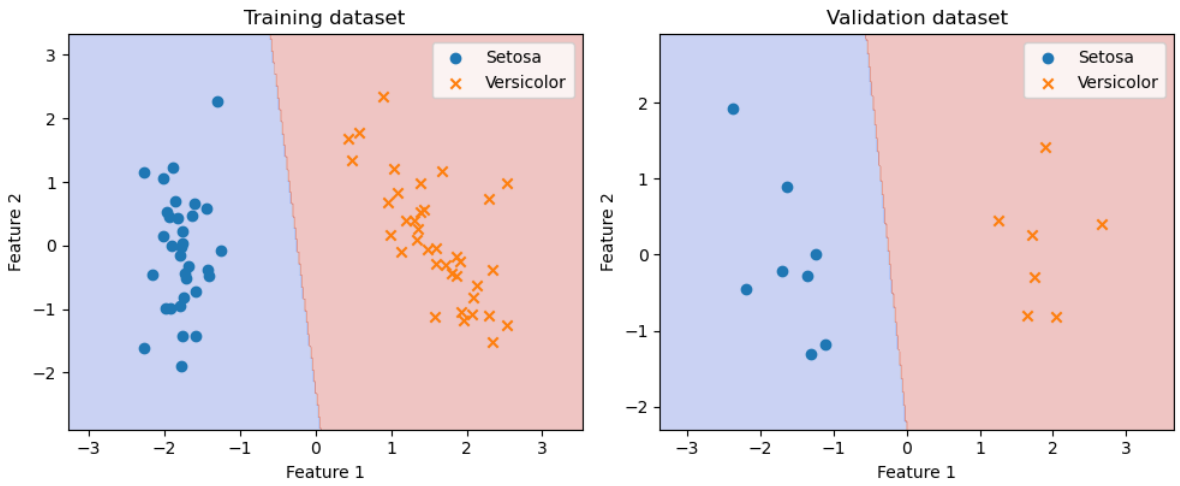

Figure 7.16: The use of soft-margin SVM without feature mapping demonstrated on the Iris training (left-hand side) and validation (right-hand side) dataset with $\Gamma = 1$. This results in accuracy scores of 1 for both the training and validation datasets.

Now consider the two moons dataset. In Figure 7.17, we once more apply the algorithm without feature mapping and the hyperparameter $\Gamma = 1$. We observe that due to the non-linear structure of the dataset, the found decision boundary is not sufficiently separating the two classes. This is as expected because without using feature mapping, no hyperplane that sufficiently separates the classes can be obtained.

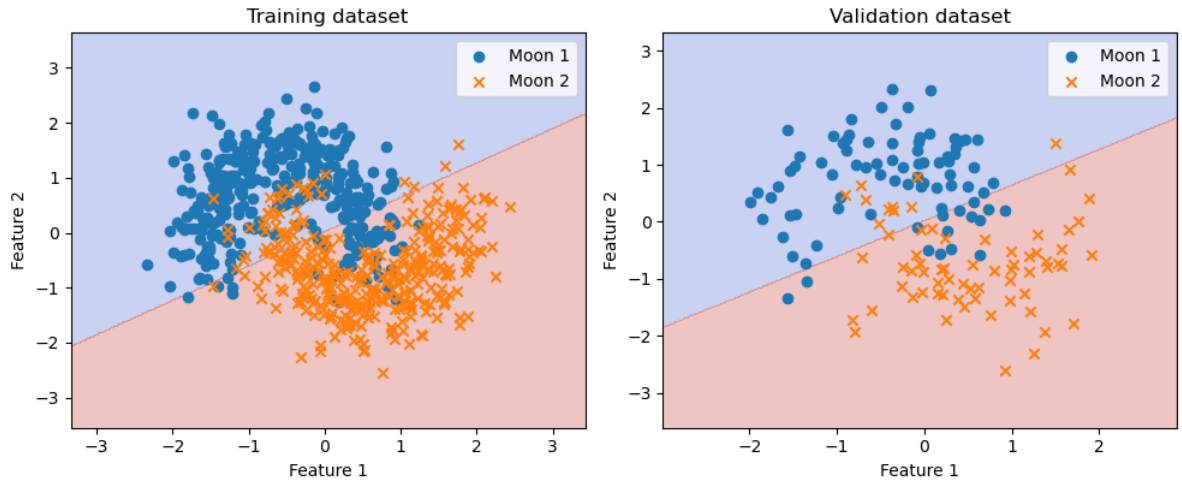

Figure 7.17: The use of soft-margin SVM without feature mapping demonstrated on the two moons training (left-hand side) and validation (right-hand side) dataset with $\Gamma = 1$. This results in accuracy scores of 0.85 and 0.83 for the training and validation datasets respectively.

To illustrate how feature mapping can influence the decision boundary for soft-margin SVM we apply basis functions $\phi_{\mathcal{A}_1}$ and $\phi_{\mathcal{A}_2}$ to the data points $\mathbf{x}^{(i)}$ in the two moons dataset. Applying $\phi_{\mathcal{A}_1}$ with $\Gamma = 1$, we observe that the corresponding decision boundary in Figure 7.18 has a slight curvature but it is not sufficiently able to follow the curves of the half moons. However, examining Figure 7.20 corresponding to using basis function $\phi_{\mathcal{A}_2}$ with $\gamma = 1$, we see that the corresponding decision boundary is able to replicate the curvature of the half moons. From this, we can conclude that by applying basis function $\phi_{\mathcal{A}_1}$, the data points are mapped to an Euclidean space where a sufficient hyperplane cannot be obtained. Nonetheless, in applying basis function $\phi_{\mathcal{A}_2}$, we map the data points to an Euclidean space where a sufficient separating hyperplane exists, as seen by the results in Figure 7.18 and 7.20.

Before concluding this subsection, we want to highlight the influence of different values of the hyperparameter Γ. Recalling the contents of Section 5.3, the hyperparameter Γ determines how costly it is to misclassify a labeled datapoint $(\mathbf{x}^{(i)}, y_i)$. A lower value of Γ implies that the penalty for misclassifying a labeled datapoint is lower. Conversely, with a greater value of Γ, the algorithm prioritizes to get as many data points correctly classified as possible.

Using basis function $\phi_{\mathcal{A}_2}$, consider Figure 7.19 and 7.20 with $\Gamma = 10^{-2}$ and $\Gamma = 1$ respectively. With the value of Γ lower in the former, a comparison of the two figures reveals that Figure 7.19 allows for a larger number of misclassifications compared to Figure 7.20. This is to be expected, as seen in the discussion on the soft-margin SVM optimization model (5.32).

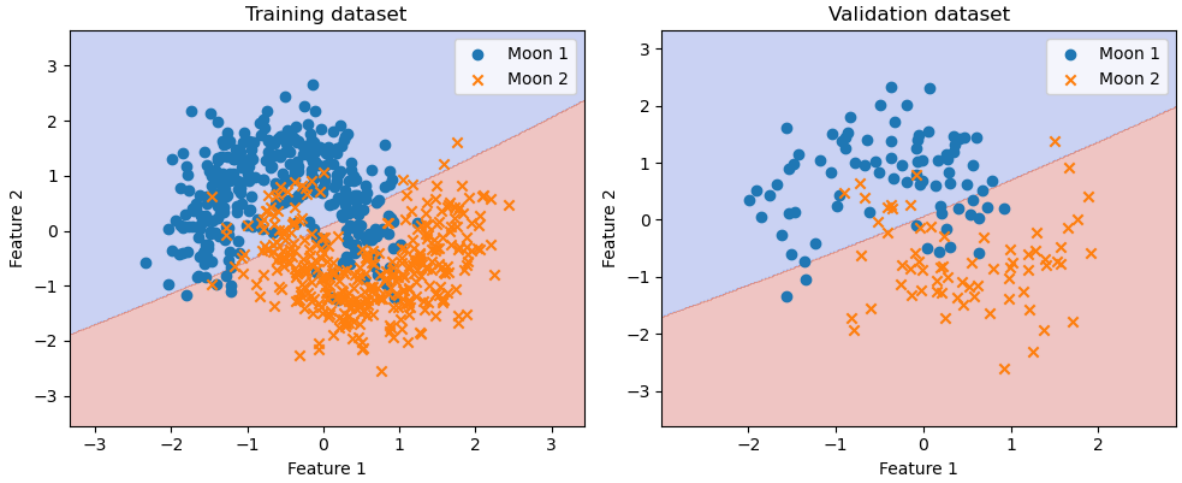

Figure 7.18: The use of soft-margin SVM with basis function $\phi_{\mathcal{A}_1}$ demonstrated on the two moons training (left-hand side) and validation (right-hand side) dataset with $\Gamma = 1$. This results in accuracy scores of 0.85 and 0.83 for the training and validation datasets respectively.

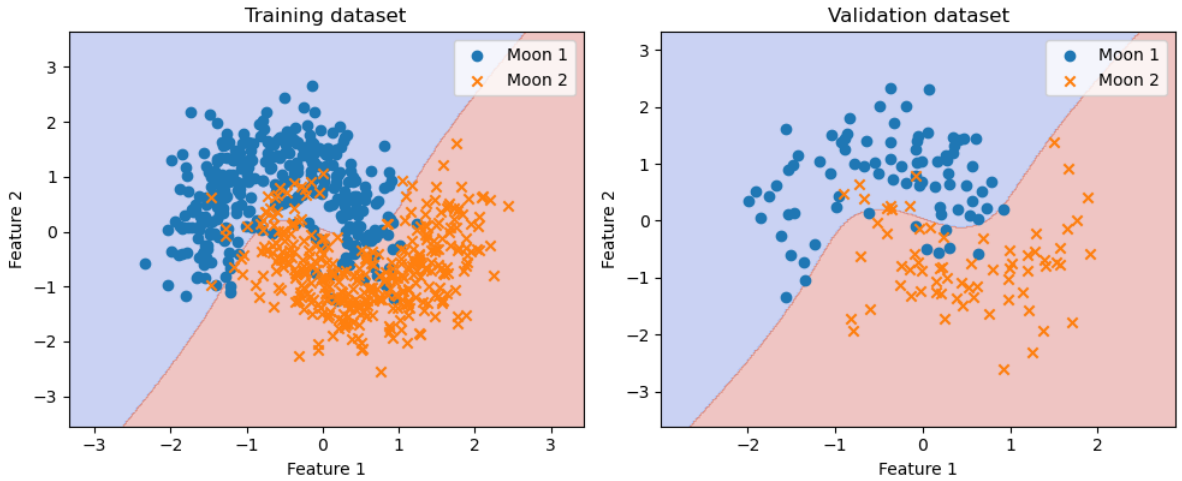

Figure 7.19: The use of soft-margin SVM with basis function $\phi_{\mathcal{A}_2}$ demonstrated on the two moons training (left-hand side) and validation (right-hand side) dataset with $\Gamma = 10^{-2}$. This results in accuracy scores of 0.90 and 0.90 for the training and validation datasets respectively.

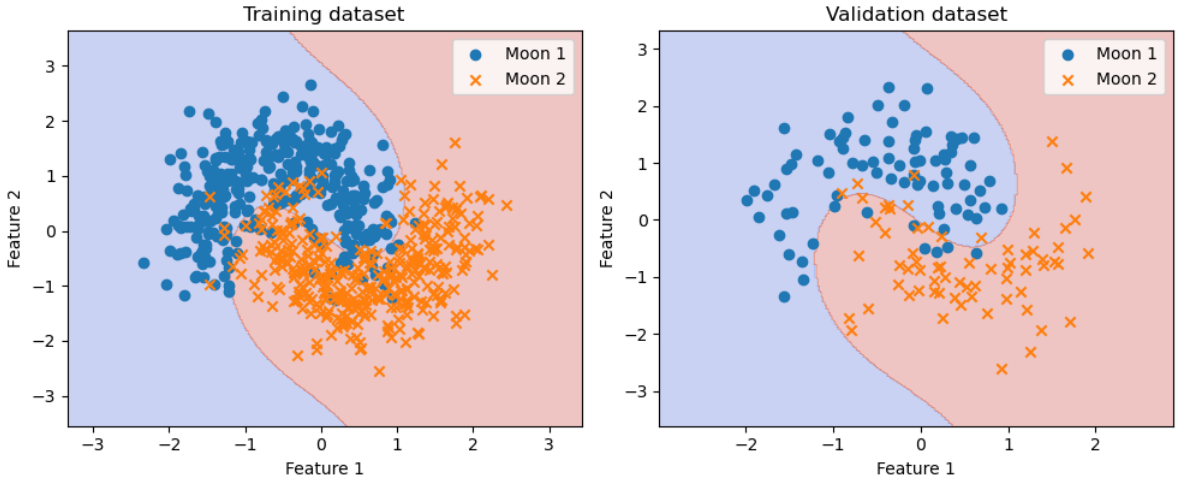

Figure 7.20: The use of soft-margin SVM with basis function $\phi_{\mathcal{A}_2}$ demonstrated on the two moons training (left-hand side) and validation (right-hand side) dataset with $\Gamma = 1$. This results in accuracy scores of 0.92 and 0.91 for the training and validation datasets respectively.

7.2.3 Kernel SVM

In this subsection we will be applying the RBF kernel to kernel SVM. We will refer to it as the RBF SVM. We will set the hyperparameter Γ to 1. Our focus will be on observing the influence of different values of the hyperparameter γ. For a description of

the hyperparameter Γ, please refer to Section 5.3. For a description of γ and kernel SVM in general, refer to Section 5.5.

Similar to our experiments with the soft-margin SVM (see Subsection 7.2.2), we will use the accuracy metric for both training and evaluation. Note that all upcoming accuracy values are rounded to two decimal digits after the comma. Additionally, for the RBF SVM algorithm we will exclusively focus on the two moons dataset. This choice is made to better illustrate the impact of the hyperparameter γ.

As presented in Section 5.5, RBF SVM implicitly maps the data points $\mathbf{x}^{(i)}$ into a higher dimensional space with the use of kernels, this without the computational cost of mapping the data points. This allows for non-linear decision boundaries. With the capability of constructing non-linear decision boundaries, we can expect the algorithm to produce sufficient results for the two moons dataset.

We are now ready to examine the results of RBF SVM on the two moons dataset (see Subsection 7.1.2).

The results of RBF SVM with hyperparameter values of $\gamma = 10^{-1}, \gamma = 10$ and $\gamma = 10^2$ are illustrated in Figure 7.21, 7.22 and 7.23 respectively. Intuitively, as stated in [37], we observe that a greater value of γ corresponds with a smaller area of influence around each labeled data point. This intuitive explanation can be better understood by recalling the contents in Example 5.4. The RBF kernel we have applied is given by

$$k(\mathbf{z}, \mathbf{z}^{'}) = \exp\bigg(-\gamma||\mathbf{z} - \mathbf{z}^{'}||^2\bigg). \tag{7.7}$$

where $\mathbf{z}, \hat{\mathbf{z}} \in \mathbb{R}^2$. In (7.7), we see that with a greater value of γ, the kernel function output decreases faster as $||\mathbf{z} - \mathbf{z}^{'}||^2$ increases compared to a lower value of γ.

The heat map of the decision boundaries corresponding to the hyperparameter values of $\gamma = 10^{-1}, \gamma = 10$ and $\gamma = 10^2$ are illustrated in Figure 7.24, 7.25 and 7.26 respectively. Especially in Figure 7.26, we observe the idea described in Example 5.4. When applying the RBF SVM, the decision boundary will consist of a combination of elevated and lowered hills. Depending on the target y_i of a data point $\mathbf{x}^{(i)}$, the algorithm will either elevate or lower the decision boundary around data point $\mathbf{x}^{(i)}$.

Furthermore, inspecting the accuracy scores in Figures 7.21, 7.22 and 7.23, we observe the trend that as γ increases, the accuracy scores for the training datasets improve. The

same is true for the accuracy scores for the validation datasets, however, with $\gamma = 10^2$ in Figure 7.23, the accuracy score for the validation dataset start to decrease compared to that of Figure 7.22. Given a performance metric, performing well on the training data does not necessarily correspond to performing well on the validation dataset.

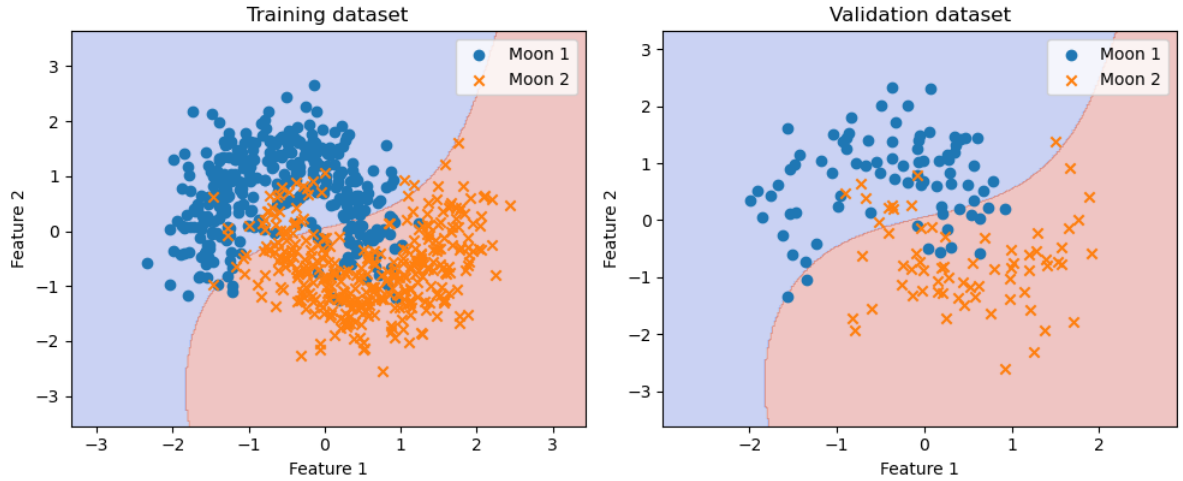

Figure 7.21: The use of RBF SVM demonstrated on the two moons training (left-hand side) and validation (right-hand side) dataset with $\Gamma = 1$ and $\gamma = 10^{-1}$. This results in accuracy scores of 0.87 and 0.84 for the training and validation datasets respectively.

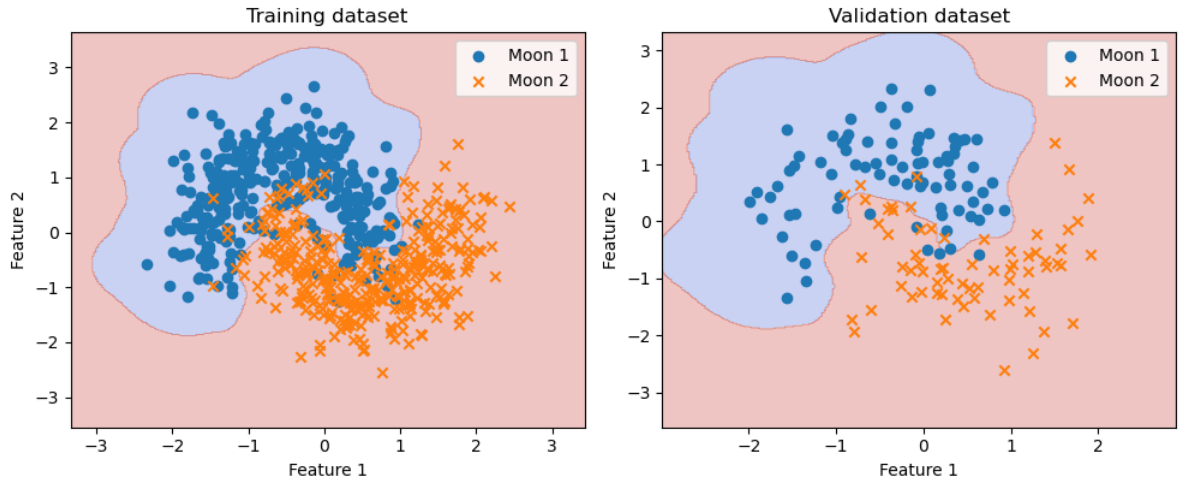

Figure 7.22: The use of RBF SVM demonstrated on the two moons training (left-hand side) and validation (right-hand side) dataset with $\Gamma = 1$ and $\gamma = 10$. This results in accuracy scores of 0.93 and 0.93 for the training and validation datasets respectively.

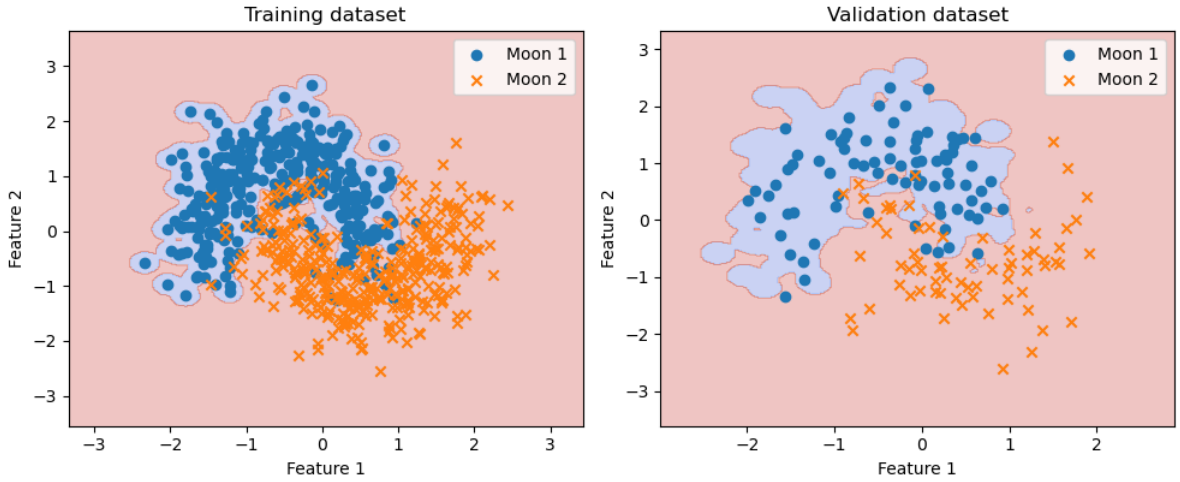

Figure 7.23: The use of RBF SVM demonstrated on the two moons training (left-hand side) and validation (right-hand side) dataset with $\Gamma = 1$ and $\gamma = 10^2$. This results in accuracy scores of 0.96 and 0.90 for the training and validation datasets respectively.

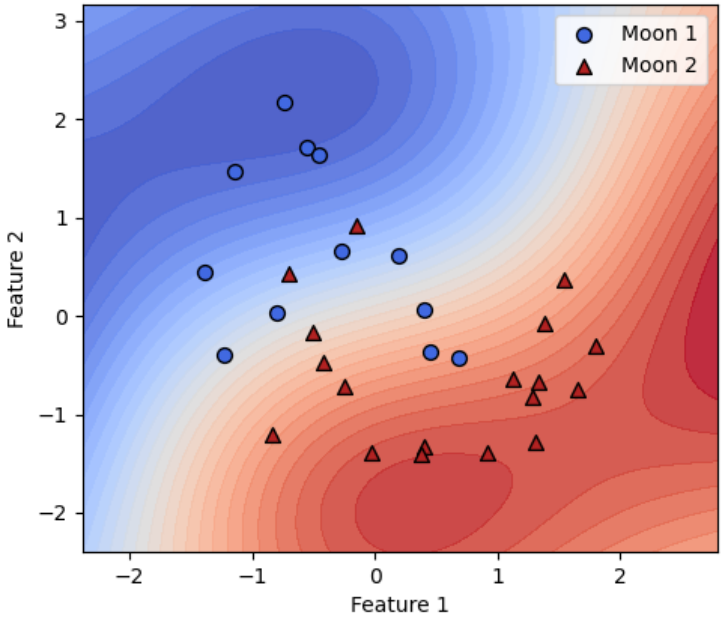

Figure 7.24: A heat map of the optimal RBF SVM decision boundary with $\Gamma = 1$, $\gamma = 10^{-1}$ for the two moons dataset. Illustrated are some of the labeled data points from the training dataset.

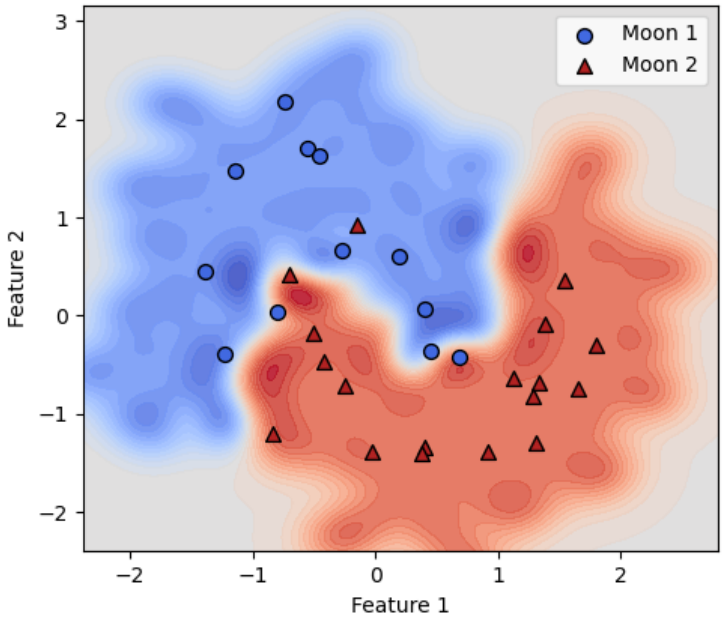

Figure 7.25: A heat map of the optimal RBF SVM decision boundary with $\Gamma = 1$, $\gamma = 10$ for the two moons dataset. Illustrated are some of the labeled data points from the training dataset.

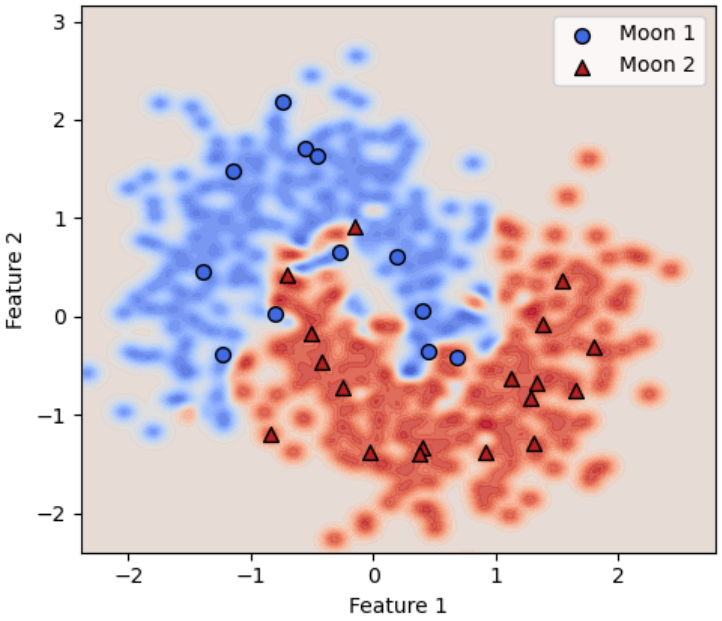

Figure 7.26: A heat map of the optimal RBF SVM decision boundary with $\Gamma = 1$, $\gamma = 10^2$ for the two moons dataset. Illustrated are some of the labeled data points from the training dataset.

Chapter 8

Conclusion

In this master's thesis, our primary objective has shed light on the fundamental use and importance of optimization in Machine Learning (ML). With an assumed background in optimization, we explored where optimization principles are applied and which theories and algorithms are involved. In this exploration, we found optimization to be of utmost importance in the learning process within the field of Supervised Learning (SL). Examining the learning processes for linear regression and Support Vector Machines (SVM), we observed that their optimization models were convex, allowing for closed-form solutions or the application of quadratic solvers to obtain optimal solutions. Noteworthy is that these algorithms represent only a subset of the diverse landscape of ML algorithms. In this broader context, it is not uncommon to encounter algorithms with learning processes expressed in terms of non-convex optimization models.

Additionally to the study of the two SL algorithms, we also discussed the process of Hyperparameter Optimization (HPO). Within this domain, it became clear that conventional properties like continuity and differentiability may not always be assumed. Furthermore, with the wide range of forms hyperparameters can take, like real, discrete or categorical values, we had to compensate by considering algorithms capable of handling such variability. This led to the consideration of derivative-free and black-box algorithms, which make little to no assumptions about the true underlying function we are maximizing or minimizing depending on the context.

While the ML algorithms and optimization discussed in this master's thesis only scratch the surface of the interplay between optimization and ML, it is key to appreciate the valuable insights gained from such exploration. While the algorithms and concepts may appear straightforward, they make up the foundation of more complex and sophisticated ML algorithms.

Abbreviations and Notation

AI Artificial Intelligence.

BBO Blackbox Optimization.

DFO Derivative-Free Optimization.

HPO Hyperparameter Optimization.

i.i.d Independent and Identically Distributed.

MAP Maximum a Posteriori.

ML Machine Learning.

MLE Maximum Likelihood Estimation.

MSE Mean Squared Error.

PCA Principal Component Analysis.

pdf Probability Density Function.

RBF Radial Basis Function.

RL Reinforcement Learning.

SL Supervised Learning.

SVM Support Vector Machines.

Bibliography

[1] *Linear Separability*. Wikipedia, 2022, (visited 04.09.2023).
URL: https://en.wikipedia.org/wiki/Linear_separability.

[2] *Introduction to Transforming Data*. Google, 2022, (visited 14.06.2023).
URL: https://developers.google.com/machine-learning/data-prep/transform/introduction.

[3] *MNIST Database*. Wikipedia, 2023, (visited 02.06.2023).
URL: https://en.wikipedia.org/wiki/MNIST_database.

[4] *Linear Inequality*. Wikipedia, 2023, (visited 02.09.2023).
URL: https://en.wikipedia.org/wiki/Linear_inequality.

[5] *Support Vector Machine*. Wikipedia, 2023, (visited 07.08.2023).
URL: https://en.wikipedia.org/wiki/Support_vector_machine.

[6] *What is Dimensionality Reduction? Overview, and Popular Techniques*. Simplilearn, 2023, (visited 09.06.2023).
URL: https://www.simplilearn.com/what-is-dimensionality-reduction-article.

[7] *Principal Component Analysis*. Wikipedia, 2023, (visited 09.06.2023).
URL: https://en.wikipedia.org/wiki/Principal_component_analysis.

[8] *Data Cleansing*. Wikipedia, 2023, (visited 09.12.2023).
URL: https://en.wikipedia.org/wiki/Data_cleansing.

[9] *Perceptron*. Wikipedia, 2023, (visited 10.08.2023).
URL: https://en.wikipedia.org/wiki/Perceptron.

[10] *Gram Matrix*. Wikipedia, 2023, (visited 10.09.2023).
URL: https://en.wikipedia.org/wiki/Gram_matrix.

[11] *Hyperparameter (Machine Learning)*. Wikipedia, 2023, (visited 12.02.2023).
URL: https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning).

[12] *Training, Validation and Test Datasets*. Wikipedia, 2023, (visited 14.06.2023).
URL: https://en.wikipedia.org/wiki/Training,_validation,_and_test_data_sets.

[13] *Data Standardization: How It's Done Why It's Important*. Simplilearn, 2023, (visited 14.12.2023).
URL: https://www.simplilearn.com/what-is-data-standardization-article.

[14] *Multivariate Normal Distribution*. Wikipedia, 2023, (visited 18.12.2023).
URL: https://en.wikipedia.org/wiki/Multivariate_normal_distribution.

[15] *Accuracy and Precision*. Wikipedia, 2023, (visited 19.12.2023).
URL: https://en.wikipedia.org/wiki/Accuracy_and_precision.

[16] *Likelihood Function*. Wikipedia, 2023, (visited 20.11.2023).
URL: https://en.wikipedia.org/wiki/Likelihood_function.

[17] *Iris Flower Data Set*. Wikipedia, 2023, (visited 23.10.2023).
URL: https://en.wikipedia.org/wiki/Iris_flower_data_set.

[18] *Multivariate Normal Distribution*. Brilliant, (visited 01.07.2023).
URL: https://brilliant.org/wiki/multivariate-normal-distribution/.

[19] *Tuning the hyper-parameters of an estimator*. Scikit-Learn, (visited 03.12.2023).
URL: https://scikit-learn.org/stable/modules/grid_search.html.

[20] *What is Unsupervised Learning?* IBM, (visited 06.02.2023).
URL: https://www.ibm.com/topics/unsupervised-learning.

[21] *What is Machine Learning?* IBM, (visited 09.02.2023).
URL: https://www.ibm.com/topics/machine-learning.

[22] *Unsupervised Machine Learning*. JavaTPoint, (visited 09.06.2023).
URL: https://www.javatpoint.com/unsupervised-machine-learning.

[23] *2.3. Clustering*. Scikit-Learn, (visited 09.06.2023).
URL: https://scikit-learn.org/stable/modules/clustering.html.

[24] *What Is a Machine Learning Pipeline?* Datatron, (visited 10.02.2023).
URL: https://datatron.com/what-is-a-machine-learning-pipeline/.

[25] *What Is a Machine Learning Pipeline?* Valohai, (visited 10.02.2023).
URL: https://valohai.com/machine-learning-pipeline/.

[26] *Supervised Learning*. Wikipedia, (visited 12.12.2023).
URL: https://en.wikipedia.org/wiki/Supervised_learning.

[27] *The McCulloch-Pitts Neuron*. O'Reilly, (visited 13.08.2023).
URL: https://www.oreilly.com/library/view/artificial-intelligence-by/9781788990547/97eeab76-9e0e-4f41-87dc-03a65c3efec3.xhtml.

[28] *sklearn.datasets.make_regression*. Scikit-Learn, (visited 13.12.2023).
URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html.

[29] *API Reference*. Scikit-Learn, (visited 14.12.2023).
URL: https://scikit-learn.org/stable/modules/classes.html.

[30] *sklearn.datasets.make_regression*. Scikit-Learn, (visited 14.12.2023).
URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html.

[31] *sklearn.model_selection.train_test_split*. Scikit-Learn, (visited 14.12.2023).
URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

[32] *History of the Perceptron*. California State University, (visited 16.08.2023).
URL: https://home.csulb.edu/~cwallis/artificialn/History.htm.

[33] *Reinforcement Learning Tutorial*. JavaTpoint, (visited 16.09.2023).
URL: https://www.javatpoint.com/reinforcement-learning.

[34] *What is Reinforcement Learning?* Synopsys, (visited 16.09.2023).
URL: https://www.synopsys.com/ai/what-is-reinforcement-learning.html.

[35] *AlphaZero: Shedding new light on chess, shogi, and Go*. Deepmind, (visited 18.09.2023).
URL: https://www.deepmind.com/blog/alphazero-shedding-new-light-on-chess-shogi-and-go.

[36] *What is Linear Regression?* IBM, (visited 18.11.2023).
URL: https://www.ibm.com/topics/linear-regression.

[37] *RBF SVM parameters*. Scikit-Learn, (visited 19.12.2023).
URL: https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html.

[38] *Kernel Method*. Wikipedia, (visited 22.08.2023).
URL: https://en.wikipedia.org/wiki/Kernel_method.

[39] *sklearn.datasets.make_moons*. Scikit-Learn, (visited 23.10.2023).
URL: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html.

[40] *What is Noise in Data Mining?* JavaTpoint, (visited 30.10.2023).
URL: https://www.javatpoint.com/what-is-noise-in-data-mining.

[41] *What is a Support Vector Machine?* MathWorks, (visited 31.10.2023).
URL: https://nl.mathworks.com/discovery/support-vector-machine.html.

[42] N. Agnihotri. *Classification of machine learning algorithms*. EngineersGarage, (visited 07.06.2023).
URL: https://www.engineersgarage.com/machine-learning-algorithms-classification/.

[43] C. Audet and W. Hare. *Derivative-Free and Blackbox Optimization*. Springer, 2017. ISBN 978-3-319-68912-8.

[44] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[45] A. Beck. *Introduction To Nonliner Optimisation*. Orient Blackswan, 2017. ISBN 978-9386235350.

[46] J. Bergstra and Y. Bengio. *Random Search for Hyper-Parameter Optimization*. In *Journal of Machine Learning Research*, Vol. 13, pages 281–305. JMLR, Inc. and Microtome Publishing, 2012.

[47] M. Bernstein. *The Radial Basis Function Kernel*. University of Wisconsin-Madison, (visited 05.11.2023).
URL: https://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf.

[48] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995. ISBN 978-1-886529-05-2.

[49] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN 978-0387-31073-2.

[50] S. Brown. *Machine Learning Explained*. MIT Sloan School, 2021, (visited 02.06.2023).
URL: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained.

[51] J. Brownlee. *Why One-Hot Encode Data in Machine Learning.* Machine Learning Mastery, 2017, (visited 06.06.2023).
URL: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/.

[52] J. Brownlee. *Why Do Machine Learning Algorithms Work on New Data?* Machine Learning Mastery, 2018, (visited 14.09.2023).
URL: https://machinelearningmastery.com/what-is-generalization-in-machine-learning/.

[53] J. Brownlee. *Difference Between Algorithm and Model in Machine Learning.* Machine Learning Mastery, 2020, (visited 10.06.2023).
URL: https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/.

[54] J. Brownlee. *No Free Lunch Theorem for Machine Learning.* Machine Learning Mastery, 2021, (visited 21.09.2023).
URL: https://machinelearningmastery.com/no-free-lunch-theorem-for-machine-learning/.

[55] A. Chandra. *McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron.* Medium, (visited 13.08.2023).
URL: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1.

[56] A. Chervonenkis. *Early History of Support Vector Machines.* In *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 13–20. Springer, 2013. ISBN 978-3-642-41135-9.

[57] B. Copeland. *Artificial Intelligence.* Encyclopædia Britannica, Inc., 2022, (visited 09.02.2023).
URL: https://www.britannica.com/technology/artificial-intelligence.

[58] C. Cortes and V. Vapnik. *Support-Vector Networks.* ATT Bell Labs, 1995.

[59] P. Domingos. *The Master Algorithm.* Basic Books, 2015. ISBN 978-0-465-06192-1.

[60] P. Felzenszwalb. *Large Margin Separators.* Brown University, 2017, (visited 25.09.2023).
URL: https://cs.brown.edu/people/pfelzens/engn2520/CS1420_Lecture_10.pdf.

[61] R. Fisher. *Iris.* UCI Machine Learning Repository, 1988, (visited 19.12.2023).
URL: https://archive.ics.uci.edu/dataset/53/iris.

[62] N. Hobbs. *Kernel SVM for Image Classification*. NathanielHobbs, 2018, (visited 02.09.2023).
URL: http://www.nathanielhobbs.com/documents/cvx_opt/cvx_opt_final_report.pdf.

[63] T. Hovsepyan. *What is Machine Learning Model Deployment*. Plat, 2022, (visited 14.06.2023).
URL: https://plat.ai/blog/machine-learning-model-deployment/.

[64] IBM. *Data Pipelines Explained*. Youtube, 2022, (visited 10.02.2023).
URL: https://www.youtube.com/watch?v=6kEGUCrBEU0&ab_channel=IBMTechnology.

[65] K. Berk J. Devore and M. Carlton. *Modern Mathematical Statistics with Applications*. Springer, 2021. ISBN 978-3-030-55155-1.

[66] C. Kang. *Image Classification with Cat and Dog*. Github, 2020, (visited 06.06.2023).
URL: https://goodboychan.github.io/python/deep_learning/tensorflow-keras/vision/2020/10/16/01-Image-Classification-with-Cat-and-Dog.html.

[67] S. Luke. *Essentials of Metaheuristics*. Lulu, 2013. ISBN 978-1-300-54962-8.

[68] W. McCulloch and W. Pitts. *A logical calculus of the ideas immanent in nervous activity*. In *The bulletin of mathematical biophysics*, Vol. 5, pages 115–133. Springer, 1943.

[69] R. Misra. *Support Vector Machines — Soft Margin Formulation and Kernel Trick*. Medium, 2019, (visited 05.09.2023).
URL: https://towardsdatascience.com/support-vector-machines-soft-margin-formulation-and-kernel-trick-4c9729dc8efe.

[70] K. Murphy. *Machine Learning A Probabilistic Perspective*. Cambridge, MA, 2012. MIT Press. ISBN 978-0-262-01802-9.

[71] R. Nian. *A review On reinforcement learning: Introduction and applications in industrial process control*. In *Computers Chemical Engineering*, Vol. 139. ScienceDirect, 2020.

[72] T. Okuno and A. Takeda. *Bilevel Optimization of Regularization Hyperparameters in Machine Learning*. In *Springer Optimization and Its Applications*, Vol. 161, pages 169–194. Springer, 2020. ISBN 978-3-030-52118-9.

[73] H. Pishro-Nik. *Introduction to Probability, statistics, and Random Processes*. Kappa Research, LLC, 2014. ISBN 978-0990637202.

[74] S. Ramamoorthy. *The Principle of Maximum Likelihood.* (visited 07.06.2023).
URL: https://blog.suriya.app/2017-01-22-mle-linear-regression/.

[75] C. Rasmussen and C. Williams. *Gaussian Process for Machine Learning.* Cambridge, MA, 2006. MIT Press. ISBN 026218253X.

[76] F. Rosenblatt. *The Perceptron — A Perceiving and Recognizing Automaton.* In *Report: Cornell Aeronautical Laboratory*, Vol. 85, pages 460–461. Cornell Aeronautical Laboratory, 1957.

[77] sagarika3kundu (username). *ML — Feature Mapping.* GeeksforGeeks, 2023, (visited 22.08.2023).
URL: https://www.geeksforgeeks.org/feature-mapping/.

[78] D. Sontag. *Support Vector Machines Kernels Lecture 6.* MIT CSAIL, (visited 02.09.2023).
URL: http://people.csail.mit.edu/dsontag/courses/ml12/slides/lecture6.pdf.

[79] M. Taboga. *"Ridge Regression", Lectures on Probability Theory and Mathematical Statistics.* Kindle Direct Publishing, 2021, (visited 27.11.2023).
URL: https://www.statlect.com/fundamentals-of-statistics/ridge-regression.

[80] AIxplained (username). *Automated Machine Learning: Grid Search and Random Search.* Youtube, 2022, (visited 03.12.2023).
URL: https://www.youtube.com/watch?v=zzGNjh37Li8&ab_channel=AIxplained.

[81] IFCAR (username). *File:2011 Hyundai Elantra GLS – 06-02-2011 2.jpg.* Wikipedia, 2011, (visited 13.10.2023).
URL: https://commons.wikimedia.org/w/index.php?curid=15477040.

[82] Sidartha (username). Hyperplane svm example class separation classification problem. Adobe Systems, (visited 05.07.2023).
URL: https://stock.adobe.com/no/contributor/209665167/sidartha?load_type=author&prev_url=detail&asset_id=382138217.

[83] SydneyF (username). *There is No Free Lunch in Data Science.* Alteryx, 2019, (visited 09.02.2023).
URL: https://community.alteryx.com/t5/Data-Science/There-is-No-Free-Lunch-in-Data-Science/ba-p/347402.

[84] M. Walsh. *ChatGPT Statistics (2023) — The Key Facts and Figures.* Style Factory, 2023, (visited 02.12.2023).
URL: https://www.stylefactoryproductions.com/blog/chatgpt-statistics.

[85] K. Weinberger. *Linear Regression.* Cornell University, (visited 08.02.2023).
URL: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote08.html.

[86] K. Weinberger. *Lecture 9: SVM.* Cornell University, (visited 08.02.2023).
URL: http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote09.html.

[87] K. Weinberger. *Lecture 3: The Perceptron.* Cornell University, (visited 14.08.2023).
URL: http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html.

[88] K. Weinberger. *Lecture 13: Kernels.* Cornell University, (visited 22.08.2023).
URL: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote13.html.

[89] D. Wolpert and W. Macready. *No Free Lunch Theorems for Optimization.* In *Transactions on Evolutionary Computation*, Vol. 1, pages 67–82. IEEE, 1997.

[90] Q. Wu and P. Vos. *Inference and Prediction.* In *Handbook of Statistics*, Vol. 38, pages 111–172. Elsevier, 2018.

[91] X. Yan. *Linear Regression Analysis: Theory and Computing.* World Scientific, 2009. ISBN 978-981-283-410-2.

[92] L. Yang and A. Shami. *On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice.* In *Neurocomputing*, Vol. 415, pages 295–316. Elsevier, 2020. ISBN 0925-2312.

[93] J. Álvarez. *Support Vector Machine and Kernel Classification Algorithms.* In *Digital Signal Processing with Kernel Methods*, pages 433–502. Wiley-IEEE Press, 2018. ISBN 9781118705810.