

Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi

**Challenges in biomedical data science:
data-driven solutions to clinical questions**

by

Samuele Fiorini

Theses Series

DIBRIS-TH-2017-XX

DIBRIS, Università di Genova

Via Opera Pia, 13 16145 Genova, Italy

<http://www.dibris.unige.it/>

Università degli Studi di Genova

**Dipartimento di Informatica, Bioingegneria,
Robotica ed Ingegneria dei Sistemi**

**Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum**

**Challenges in biomedical data science:
data-driven solutions to clinical questions**

by

Samuele Fiorini

December, 2017

Dottorato di Ricerca in Informatica ed Ingegneria dei Sistemi
Indirizzo Informatica
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università degli Studi di Genova

DIBRIS, Univ. di Genova
Via Opera Pia, 13
I-16145 Genova, Italy
<http://www.dibris.unige.it/>

Ph.D. Thesis in Computer Science and Systems Engineering
Computer Science Curriculum
(S.S.D. INF/01)

Submitted by Samuele Fiorini
DIBRIS, Univ. di Genova

. . . .

Date of submission: December 4, 2017

Title: Machine Learning 4 healthcare.

Advisor: Annalisa Barla
Dipartimento di Informatica, Bioingegneria, Robotica ed Ingegneria dei Sistemi
Università di Genova

. . .

Ext. Reviewers:
Lo Scopriremo
Lo Scopriremo
Lo Scopriremo

Abstract

Abstract

Contents

1	Introduction	1
2	Basic notation and definitions	2
Part I		4
3	Background	4
3.1	What is data science and why should we care?	4
3.2	Challenges in biomedical data science	4
3.3	From clinical questions to data analysis	5
3.3.1	How to predict phenotypes from observed data?	6
3.3.2	Which variables are the most significant?	6
3.3.3	How to stratify the data?	6
3.3.4	How to represent the samples?	6
3.3.5	Are there recurring patterns in the data?	7
3.3.6	How to deal with missing values?	7
4	Machine learning state of the art	8
4.1	Supervised learning	10
4.1.1	Regularization methods	13
4.1.1.1	Ordinary least squares	14
4.1.1.2	Ridge regression	15
4.1.1.3	Lasso	17
4.1.1.4	Elastic-Net	20
4.1.1.5	Logistic Regression	23
4.1.1.6	Support Vector Machine	25
4.1.2	The kernel trick	26
4.1.3	Decision trees	28
4.1.4	Ensemble methods	31

4.1.4.1	Random Forests	32
4.1.4.2	Gradient Boosting	33
4.1.5	Deep learning	35
4.1.5.1	Multi-Layer Perceptron	36
4.1.5.2	Long-Short Term Memory Network	37
4.2	Feature selection	38
4.3	Unsupervised learning	38
4.3.1	Cluster analysis	38
4.3.1.1	The k -means algorithm	38
4.3.1.2	Spectral clustering	40
4.3.1.3	Hierarchical clustering	40
4.3.2	Dimensionality reduction and feature learning	41
4.3.2.1	Principal component analysis	42
4.3.2.2	Multi-dimensional scaling	42
4.3.2.3	Isomap	42
4.3.2.4	t-Distributed Stochastic Neighbor Embedding	42
4.4	Model selection and evaluation	42
4.4.1	Model selection strategies	42
4.4.2	Feature selection stability	42
4.4.3	Performance metrics	42
4.5	Computational requirements and implementations	46

Part II 49

5 ADENINE: a HPC-oriented tool for biological data exploration 49

5.1	Exploratory data analysis	49
5.2	ADENINE overview	50
5.3	Software description	51
5.4	Usage example	54
5.5	Results	55

6 Model for metabolic age prediction 58

6.1	Aging and metabolism	58
-----	--------------------------------	----

7	Temporal model for multiple sclerosis evolution	59
7.1	PCOs data set description	60
7.2	Problem description	62
7.2.1	Data preprocessing	62
7.2.2	Experimental design	62
7.2.3	Learning $f(x)$	63
7.2.4	Learning $g(x)$	64
7.3	Results	64
8	Temporal model for glucose predictions	67
8.1	Diabetes care	67
8.2	Methods	68
8.2.1	Experimental Setting	68
8.2.2	Autoregressive Integrated Moving Average	70
8.2.3	Kalman Filter	70
8.3	Data	71
8.4	Results	71
8.5	Conclusions and Future Works	71
9	Conclusion	74
Appendix A	Appendix	75
A.1	Useful theorems and definitions	75
A.2	Empirical risk minimization	76
A.3	Maximum likelihood estimation	77
A.4	ERM vs MLE/MAP	78
A.4.1	Linear regression revisited	79
A.4.2	Logistic regression revisited	80

List of Figures

List of Tables

1 Introduction

Understanding the underlying mechanisms of biological systems can be a challenging task. Different domains can be involved and their interactions can be unknown.

Nowadays, most of the life science research aims at investigating on the extraction of meaningful information from heterogeneous sources of biological data. Thanks to the remarkable technological progresses of the last decades, the dimensions of modern data collections is continuously increasing.

This PhD thesis is divided in two parts. Part I presents a thorough description of the multi-disciplinary prerequisites that are relevant for the comprehension of Part II, which, in turn, presents the original contributions of my work.

Part I is organized as follows: Chapter 3 introduces the concept of *data science* (Section 3.1) and its declination toward life science and biomedical studies. In this chapter, the major challenges of the field are presented (Section 3.2) along with several examples of the most common clinical/biological questions and their translation to data analysis tasks (Section 3.3). Chapter 4 summarizes basic notation and definitions adopted throughout the thesis (Section 2) and presents an overview of the statistical and technological tools that are mostly relevant for this work. In particular, this chapter defines the concept of *machine learning* from a general perspective and provides rigorous description of a selection of supervised and unsupervised learning strategies (Section ??). This chapter also defines the concept of variable/feature selection (Section 4.2) and introduces the most relevant model selection and evaluation strategies (Section 4.4). At the end of this chapter, hints on the computational requirements and implementation strategies are finally presented (Section 4.5).

Part II describes the main contributions of my PhD work which consisted in the process of translating into data analysis tasks a number of biological questions coming from real-world clinical environments. For each task, this second part shows how the previously introduced tools can be exploited in order to develop statistically sound models that are capable of providing insightful answers to different clinical questions. This part is organized as follows: Chapter 5 introduces ADENINE, an open-source Python framework for large-scale data exploration that I developed during my PhD. Chapter 6 describes a work I conducted in collaboration with *Istituto Giannina Gaslini Children's Hospital* on metabolic age estimation from peripheral blood mononuclear cells samples. Chapter 7 describes a work I conducted in collaboration with the *Italian Multiple Sclerosis Foundation* in which I developed a temporal model that aims at predicting the evolution of multiple sclerosis patients exploiting the use of patient-friendly and inexpensive measures such as patient centered outcomes. Chapter 8 describes a work held in collaboration with *Ospedale Policlinico San Martino* in which I developed a machine learning time-series forecasting approach for glucose sensor data collected by type I and type II diabetic patients.

My conclusions are finally drawn in Chapter 9.

2 Basic notation and definitions

In this thesis, datasets \mathcal{D} are described as input-output pairs, $X \in \mathbb{R}^{n \times d}$ and $Y \in \mathbb{R}^{n \times k}$, respectively. The i -th row of X is a d -dimensional data point \mathbf{x}_i belonging to the input space $\mathcal{X} \subseteq \mathbb{R}^d$. The corresponding outputs \mathbf{y}_i belong to the output space \mathcal{Y} .

The nature of the output space defines the problem as *binary classification* if $\mathcal{Y} = \{a, b\}$ (with $a \neq b$), *multiclass classification* if $\mathcal{Y} = \{\alpha, \beta, \dots, \omega\}$ (with $\alpha \neq \beta \neq \dots \neq \omega$), *regression* if $\mathcal{Y} \subseteq \mathbb{R}$ and *vector-valued regression* if $\mathcal{Y} \subseteq \mathbb{R}^k$. For binary classification problems common choices for the label encoding are $a = 1, b = -1$ or $a = 0, b = 1$. For multiclass classification problems classes are usually encoded as natural numbers, *i.e.* $\alpha, \beta, \dots, \omega \in \mathbb{N}$.

Predictive models are functions $f : \mathcal{X} \rightarrow \mathcal{Y}$. The number of relevant variables is d^* . In feature selection tasks, the number of selected features is \tilde{d} .

A kernel function acting on the elements of the input space is defined as $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, where $\phi(\mathbf{x})$ is a *feature map* from $\mathbb{R}^d \rightarrow \mathbb{R}^{d'}$. Feature learning algorithms project the data into a p -dimensional space.

Whenever possible, real-valued variables will be indicated with lowercase letters (*e.g.* a), unidimensional vectors with lowercase bold letters (*e.g.* \mathbf{a}) and matrices, or tensors, with capital letters (*e.g.* A). When the value of some variable/parameter is the result of some data-drive estimation, such variable will be highlighted with a hat (*e.g.* \hat{a}). When used in the context of a data matrix, a subscript index will be used to identify a sample whereas a superscript index will refer to a given feature. So, for instance, given a data matrix $X \in \mathbb{R}^{n \times d}$ the j -th feature of the i -th sample is \mathbf{x}_i^j , with $0 \leq i \leq n$ and $0 \leq j \leq d$.

Part I

3 Background

This chapter defines stuff....

3.1 What is data science and why should we care?

{

- **Data engineering**
- **Data exploration**
- **Machine learning and data understanding**
- **Data visualization**

}

- cross-disciplinary field
- Drew Conway's Data Science Venn Diagram, first published on his blog in September 2010
- data-intensive applications (maybe)

3.2 Challenges in biomedical data science

The process of modeling complex systems often implies collecting large amount of data in the field of life science, where large, multivariate and noisy measurements are typically acquired with the aim of describing multifactorial diseases.

In the era of personalized medicine, biospecimen collection and biological data management is still a challenging and expensive task [Toga and Dinov, 2015]. Only few large-scale research enterprises, such as ENCODE [Consortium et al., 2004], ADNI [Jack et al., 2008], MOPED [Kolker et al., 2012] or TCGA ², have sufficient financial and human resources to manage, share and distribute access of heterogeneous types of biological data. To date, many biomedical studies still rely on a small number of collected samples [McNeish and Stapleton, 2016; Button et al., 2013; Yu et al., 2013]. This effect is even worse in case of rare diseases [Garg et al., 2016] or in high-throughput molecular data (*e.g.* genomics and proteomics) where the dimensionality

¹<http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram>

²<https://cancergenome.nih.gov>

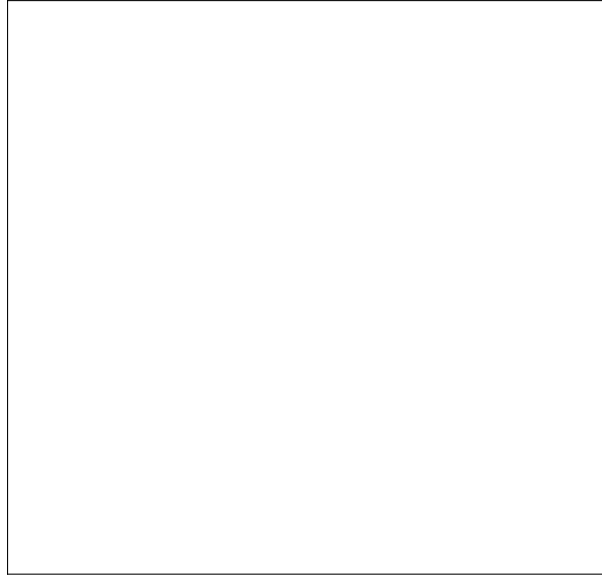


Figure 3.1: Drew Conway's Data Science Venn Diagram¹.

of the problem can be in the order of hundreds of thousands. The setting where the number of measured variables heavily outnumbers the amount of collected samples is usually referred to as *large p small n* scenario, or simply $n \ll p$. In this case, the main goal of the learning step is often to identify a meaningful subset of relevant variables that are the most representative of the observed phenomenon. In machine learning, this is known as variable/feature selection and several techniques addressing this task were presented so far [Guyon et al., 2002]. Variable selection not only increases the prediction power of the learning machine, but it also promotes model interpretability, that is crucial in biology [Altmann et al., 2010]. Regardless [Okser et al., 2014] of the learning machine, regularization can be introduced in several ways and it is of fundamental use in order to achieve the following desired properties:

- identify models with good generalization properties, even with a limited amount of collected samples;
- achieve solutions that are robust to noise;
- learn the data structure when unknown;
- exploit prior knowledge on the data structure;
- promote interpretability performing variable selection;
- reduce the feasible set in order to help solving inverse problems.

In this paper we illustrate how regularization impacts in finding robust and meaningful models and we clarify how to choose the most suitable regularization scheme according to the biological context. The remainder of the paper is organized as follows:

3.3 From clinical questions to data analysis

In applied life science, the biological question at hand usually drives the data collection and therefore the statistical challenge to be solved. In order to achieve meaningful results, thorough data analysis protocol must be followed (see Section 4.4). In this section, my goal is to illustrate some of the most recurrent biological questions and how they can be translated into machine learning tasks. **{assist next chapter}**

3.3.1 How to predict phenotypes from observed data?

Starting from a collection of input measures that are likely to be related with some known target phenotype, the final goal here is to learn a model that represents the relationship between input and output. Several researches fall in this class, for instance in molecular (*e.g.* lab tests, gene expression, proteomics, sequencing) [Angermueller et al., 2016; Okser et al., 2014; Abraham et al., 2013] or radiomics/imaging studies (*e.g.* MRI, PET/SPECT, microscopy) [Min et al., 2016; Helmstaedter et al., 2013]. Biological questions of this class are usually tackled by *supervised learning* models. In particular, when the observed clinical outcome is expressed as a one-dimensional continuous value, as in survival analysis, a *single-output regression* problem is posed. Moreover, if the outcome is vector-valued, as in the case of multiple genetic trait prediction [He et al., 2016], the problem can be cast in a *multiple-output regression* framework [Argyriou et al., 2008; Baldassarre et al., 2012]. Biological studies involving categorical outcomes translate into *classification* problems. In particular, if the clinical outcome assumes only two values, as in the *case-control* scenario, the classification problem is said to be *binary*, whilst, if multiple classes are observed, the classification task becomes *multiclass*.

3.3.2 Which variables are the most significant?

In the above case, a complementary question revolves around the interpretability of the predictive model. In particular, if dealing with high-throughput data, the main goal is to identify a relevant subset of meaningful variables for the observed phenomenon. This problem can be cast into a variable/feature selection problem [Guyon et al., 2002].

A machine learning model is said to be *sparse* when it only contains a small number of non-zero parameters, with respect to the number of features that can be measured on the objects this model represents [Hastie et al., 2015; Meier et al., 2008]. This is closely related to feature selection: if these parameters are weights on the features of the model, then only the features with non-zero weights actually enter the model and can be considered *selected*.

3.3.3 How to stratify the data?

Collecting measures from several samples, the final goal here is to divide them in homogeneous groups, according to some *similarity* criterion. In machine learning, this is usually referred to as *clustering* [Hastie et al., 2009].

3.3.4 How to represent the samples?

In order to formulate a model of some natural phenomenon, it is necessary to design and follow a suitable data collection protocol. A natural question that may arise here is whether the raw collected measures are intrinsically representative of the target phenomenon or if some transformation must be applied in order to achieve a data representation that can be successfully exploited by a learning machine. For instance, it may be plausible to assume that the data lie in a low-dimensional embedding or that they can be better represented by a richer polynomial or Gaussian expansion. A common solution, in this case, is to take advantage of *feature engineering* techniques to obtain hand crafted features. However, this process can be very time-consuming and it may require the help of domain experts. The process of automatically identify suitable representations from the data itself is usually referred to as *(un)supervised feature learning* [Angermueller et al., 2016; Mamoshina et al., 2016].

3.3.5 Are there recurring patterns in the data?

Analyzing data coming from complex domains, one may be interested in understanding whether complex observations can be represented by some combination of simpler events. In machine learning this typically translates into *adaptive sparse coding* or *dictionary learning* problems [Masecchia et al., 2015; Alexandrov et al., 2013].

3.3.6 How to deal with missing values?

Applied life science studies must often deal with the issue of missing data. For instance, peaks can be missed in mass-spectrometry [Jung et al., 2014] or gene expression levels can be impossible to measure due to insufficient array resolution or image corruption [Stekhoven and Bühlmann, 2011; Troyanskaya et al., 2001]. Common strategies, such as discarding the samples with missing entries, or replacing the holes with the mean, median or most represented value, fall short when the missing value rate is high or the number of collected samples is relatively small. In machine learning this task usually translates into a *matrix completion* problem [Candès and Recht, 2009].

4 Machine learning state of the art

This chapter starts defining the concept of machine learning in its two major declinations: supervised and unsupervised learning. It continues providing a comprehensive overview of algorithms, models and techniques relevant for the biomedical data science applications described in Part II. At the end of this chapter, an overview on the computational requirements and the most recent machine learning technologies is given.

The term *Machine Learning* (ML) first appeared in the late 50's in the field of computer science and it is now becoming a buzzword used in several contexts spanning from particle physics and astronomy to medicine and social sciences [Service, 2017]. With a simple search on Google Trends¹ it is possible to roughly quantify the pervasiveness of this term on the Internet in the last few years. From Figure 4.1 we can see that the interest toward both the terms *machine learning* and *data science* is growing, with the first consistently superior to the second.

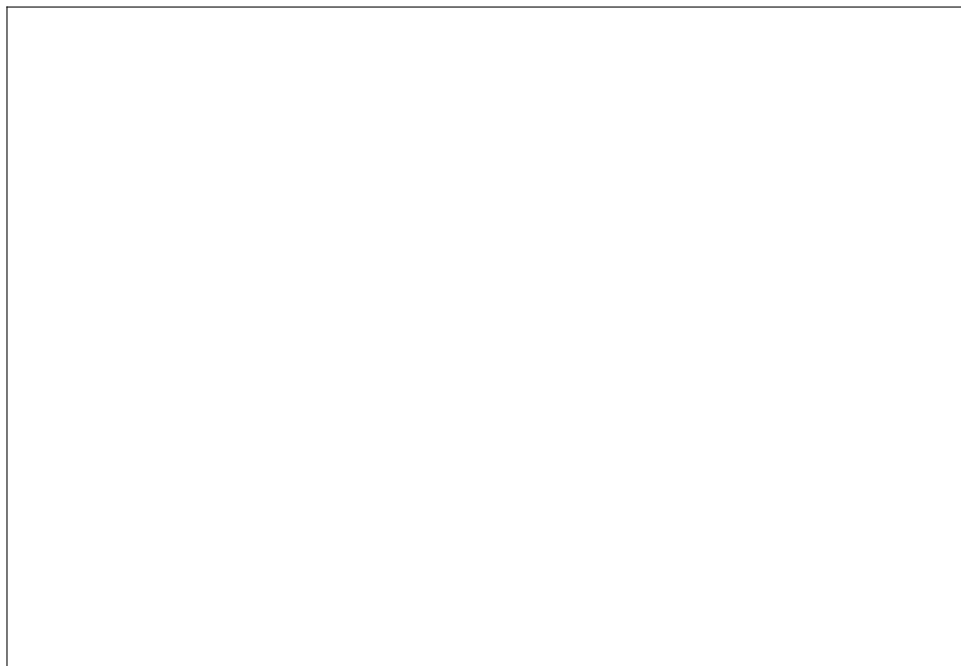


Figure 4.1: The Internet popularity over the past five years of two terms: *data science* and *machine learning*. The vertical axis represents the number of Google searches of an input term normalized with respect to its maximum (source: Google Trends).

A partial explanation to this phenomenon can be found in a recent article published on Science [Appenzeller, 2017], where the authors observed how the explosion of modern data collection abilities is leading the human kind toward another *scientific revolution*. Biomedical applications are prototypical in this sense. For instance, the volume of the raw data acquired from a genome sequencer for a single DNA has a volume of approximately 140 GB [Marx,

¹<https://trends.google.com>

2013]. Another example can be the 3D reconstruction of cardiac MRI acquisition which needs around 20 GB for a single human heart, or the 3D CT scan which has a volume in the order of GB for each patient, and so on. It has been estimated that an average hospital currently stores more than 665 TB of data that needs to be analyzed and understood. Such massive amounts of data have long overwhelmed human analysis and insights potential. This makes ML a key element for clinicians and scientists that try to make sense of large-scale observations.

But, what is *machine learning*? And how does it differ from classical statistics?

A unique answer to this question is not easy to provide. In fact, ML can be defined in different ways and from several standpoints. Let's see three remarkable examples.

1. Kevin P. Murphy in its *Machine Learning - A Probabilistic Perspective* [Murphy, 2012] defines machine learning as follows.

"[. . .] a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty [. . .]"

2. Trevor Hastie, a well-known applied statistician, in a famous seminar², held in October 2015 at the Stanford University, gave the following three definitions.

Machine Learning constructs algorithms that can learn from data.

Statistical Learning is a branch of applied statistics that emerged in response to machine learning, emphasizing statistical models and assessment of uncertainty.

Data Science is the extraction of knowledge from data, using ideas from mathematics, statistics, machine learning, computer science, engineering...

3. Carl E. Rasmussen in the preface of its renowned *Gaussian Processes for Machine Learning* [Rasmussen and Williams, 2006] introduces the difference between classical statistics and ML as follows.

"[. . .] in statistics a prime focus is often in understanding the data and relationships in terms of models giving approximate summaries such as linear relations or independencies. In contrast, the goals in machine learning are primarily to make predictions as accurately as possible and to understand the behaviour of learning algorithms [. . .]"

It looks like each author, according to his background, expertise and experience, provides a slightly different definition of ML. Trying to summarize these three standpoints, we can say that *ML is an interdisciplinary field that borrows the concept of data-driven model from statistics in order to devise algorithms that can exploit hidden patterns in current data and make accurate predictions on future data.*

As of today ML is the workhorse of data science.

²part of Data Science @ Stanford Seminar series (source: <https://goo.gl/UFgqxU>)

4.1 Supervised learning

Humans are remarkably good at *learning by examples*. When a kid is taught what a pencil looks like, he will be capable of understanding the concept of pencil from a limited number of guided observations. Similarly, when future radiologists are trained to distinguish between healthy tissues from tumors in MRI scans, they will be provided with several annotated biomedical images from which they will be able to generalize. The applied learning paradigm is characterized by the presence of two key objects: *data* and *labels*. In the last example, the MRI scans are the data, and their annotations (*e.g.* tumor vs healthy tissue) are the labels.

Supervised learning is the branch of ML in which predictive models are trained on labeled data. In the ML jargon, and in this thesis, one usually refers to *data* as collections of *samples* described by an arbitrarily large number of *predictors (features)* that are used as *input* in a training process having labels as *output*.

Input samples throughout this thesis are represented as d -dimensional vectors \mathbf{x} belonging to an input space \mathcal{X} , where typically $\mathcal{X} \subseteq \mathbb{R}^d$ and labels are represented with the variable y belonging to an output space \mathcal{Y} . The nature of \mathcal{Y} defines the learning task as *binary classification* if $\mathcal{Y} = \{-1, +1\}$, *multiclass classification* if $\mathcal{Y} = \{1, 2, \dots, k\}$, *regression* if $\mathcal{Y} \subseteq \mathbb{R}$ and *vector-valued regression* if $\mathcal{Y} \subseteq \mathbb{R}^k$. The remainder of this section summarizes the methods that are most relevant with the data-driven strategies adopted to tackle the biomedical data science challenges described in the second part of in this thesis.

Given a set of input-output pairs $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n = (X, \mathbf{y})$, supervised learning methods aim at finding a function of the inputs $f(\mathbf{x})$ that approximates the output y . This translates into the minimization problem defined in Equation (4.1).

$$\operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i), y_i) \quad (4.1)$$

The loss function $L(f(\mathbf{x}), y)$ can be seen as a measure of *adherence* to the available training data. Several loss functions for regression and classification problems were proposed; Table 4.1 defines the most commonly adopted in biomedical studies while their visual representation is presented in Figure 4.2. Choosing the appropriate loss function for the problem at hand is crucial and there is no trivial solution for this problem. Different choices for $L(f(\mathbf{x}), y)$ identifies different learning machines, that are known under different names. The most popular methods will be presented in the next few sections.

Identifying a reliable data-driven model can be a very tricky task. Many unwanted and concurrent factors may be misleading and the solution may have poor predictive power for several reasons. Including:

1. the acquisition devices may introduce random fluctuations in the measures;
2. the amount of collected samples n may be small with respect to the number of observed variables d ;
3. a non-negligible number of the measured variables may not be representative of the target phenomenon.

Table 4.1: Definition of the loss functions for regression (top) and classification (bottom) problems represented in Figure 4.2.

Loss function	$L(f(\mathbf{x}), y)$	Learning problem
Square	$(y - f(\mathbf{x}))^2$	regression
Absolute	$ y - f(\mathbf{x}) $	regression
ε -insensitive	$ y - f(\mathbf{x}) _\varepsilon$	regression
Zero-one	$\mathbb{1}\{y \neq f(\mathbf{x})\}$	classification
Square	$(1 - yf(\mathbf{x}))^2$	classification
Logistic	$\log(1 + e^{-yf(\mathbf{x})})$	classification
Hinge	$ 1 - yf(\mathbf{x}) _+$	classification
Exponential	$e^{-yf(\mathbf{x})}$	classification

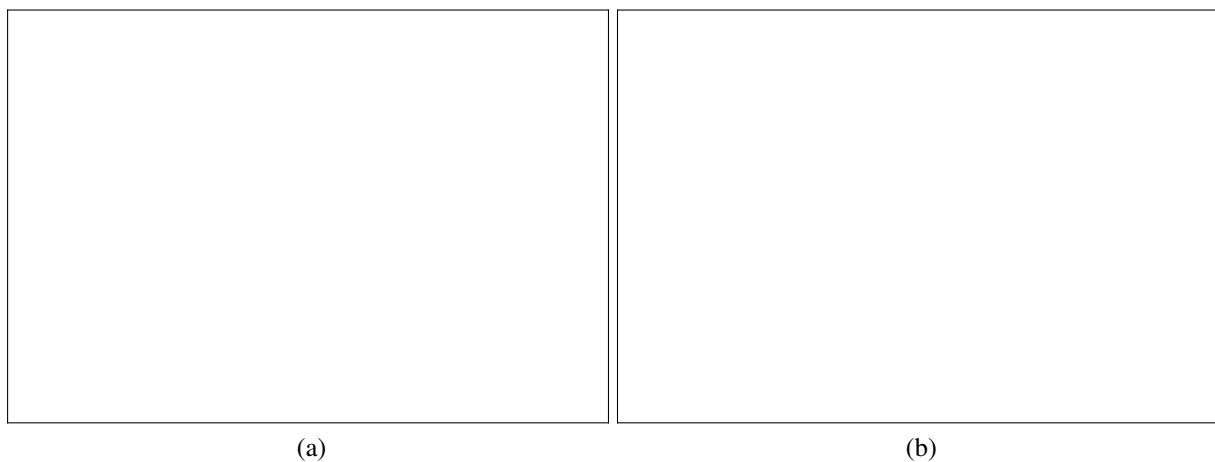


Figure 4.2: An overview on the most common loss functions for regression (a) and classification (b) problems plotted against the corresponding prediction error.

From a modeling standpoint, every combination of the factors above can be seen as *noise* affecting the data. Precautions in the model formulation process must be taken in order to achieve solutions that are insensitive to small changes in the input data and that are, in general, *robust* to the noise effect.

Considering a ML model (\hat{f}) fitted on a collection of data (\mathcal{D}), the most desirable property of \hat{f} is that it should be able to achieve good prediction performance not only on \mathcal{D} , but also on all the future, therefore unseen, data points \mathcal{D}' . In other words, assuming that the samples in \mathcal{D} are affected by some kind of random³ component, \hat{f} should be a predictive function that does not *follow the noise*, but rather models the true input-output relationship. In ML, a model that fits well \mathcal{D} but performs poorly on \mathcal{D}' is said to be *overfitting*.

In ML literature, *regularization* is the most important countermeasure to overfitting and it is widely adopted, under several forms, to build predictive models out of noisy data.

The original contribution of this PhD thesis mainly relies on the application of data science and ML concepts to noisy domains. Therefore, regularization strategies are of primary interest in this discussion. For each learning algorithm described, particular emphasis will be put on the

³here with *random* means "uncorrelated with the input-output relationship"

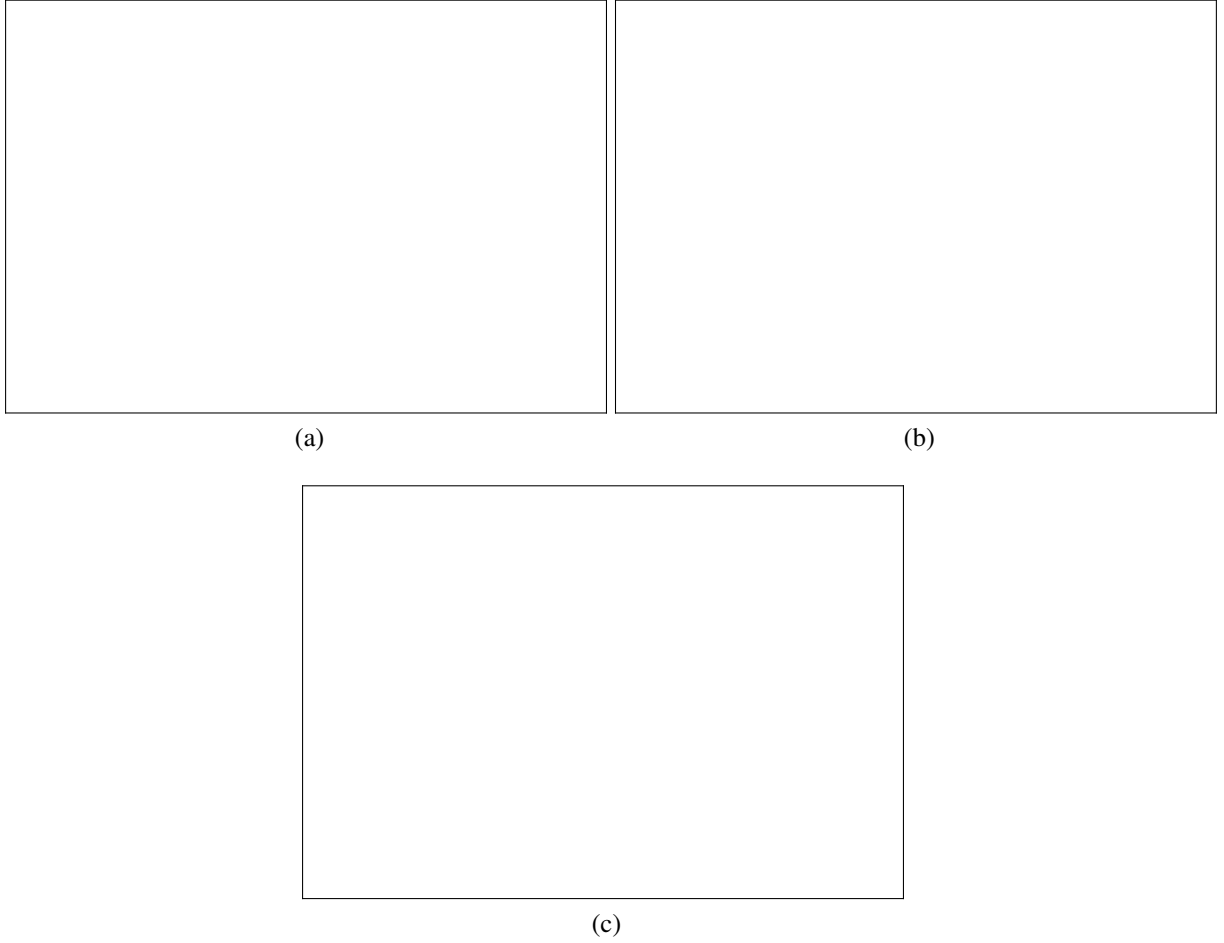


Figure 4.3: An example of underfit (a), overfit (b) and optimal fit (c) for a nonlinear regression problem. The data are a downsampled version ($f_s = 546 \text{ Hz}$) of the first observation of gravitational waves from a binary black hole merger detected on September 14th, 2015, 09:50:45 UTC at LIGO Hanford (WA).

relevant regularization strategies.

In its broader definition *regularization* can be seen as the process of introducing additional information in order to solve a possibly ill-posed problem. As shown in Equation (4.2), this is typically translated in the use of a regularization penalty $\mathcal{R}(f)$, controlled by a regularization parameter λ [Tikhonov, 1963; Evgeniou et al., 2000].

$$\operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i), y_i) + \lambda \mathcal{R}(f) \quad (4.2)$$

Choosing different $\mathcal{R}(f)$ implies inducing different effects on the solution and it also leads to the definition of different learning machines (see Section 4.1.1). With the regularization parameter λ it is possible to control the trade-off between adherence to the training data and strength of the effect induced by $\mathcal{R}(f)$. As an example, we can think of using a penalty that induces smoothness, such as the ℓ_2 -norm, or sparsity, such as the ℓ_1 -norm, in the solution. A pictorial representation of a learning machine working in overfitting, underfitting and optimal

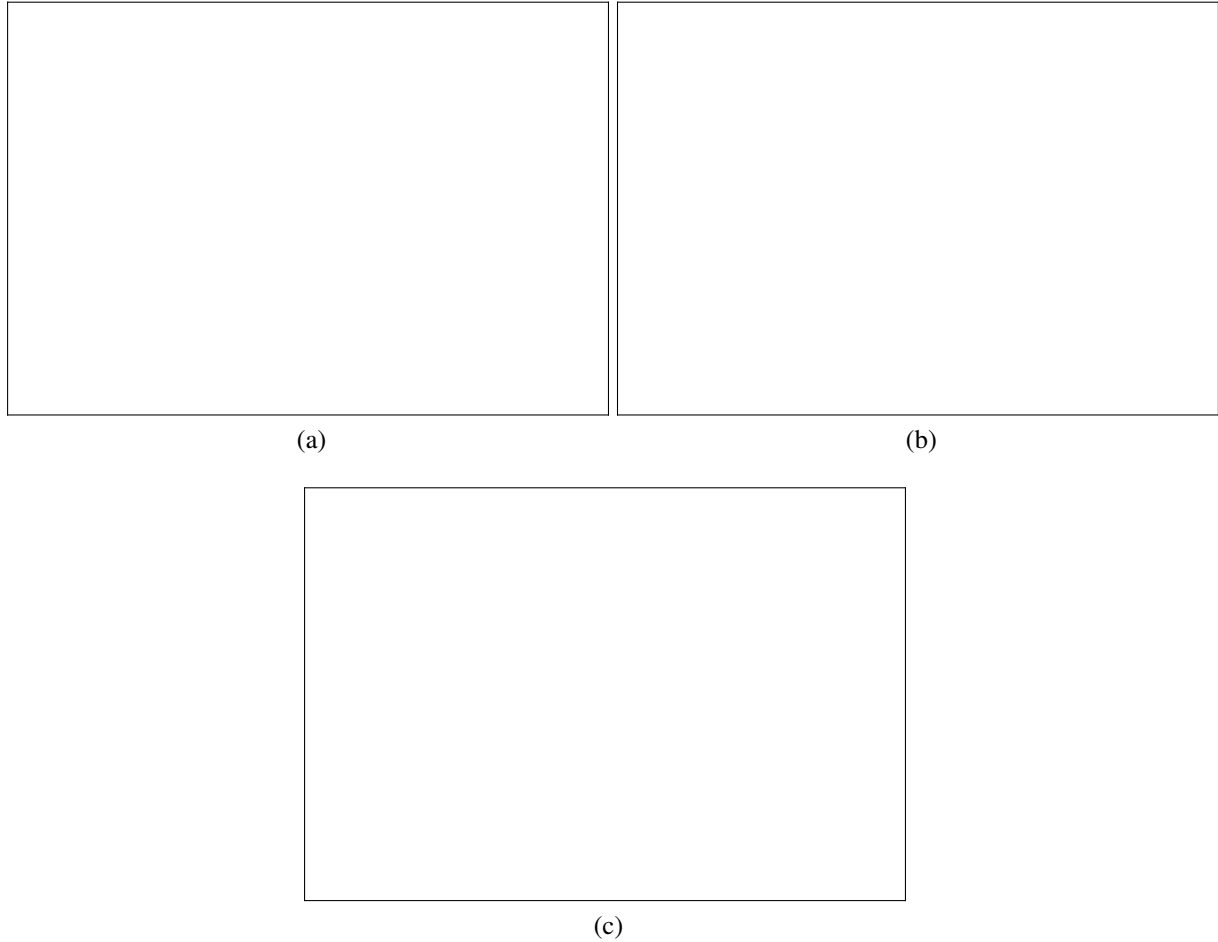


Figure 4.4: An example of underfit (a), overfit (b) and optimal fit (c) for a nonlinear binary classification problem. Each data point is a pulsar candidate randomly sampled from the High Time Resolution Universe Survey (South) dataset. The data are standardized and their dimensionality is reduced by the t-SNE algorithm [Van der Maaten and Hinton, 2008].

fitting regime in a regression and a classification case can be seen in Figure 4.3⁴ and Figure 4.4⁵, respectively.

Supervised learning machines may rely on very different mathematical backgrounds such as generalized linear models, nonlinear deep neural networks, kernels, trees, ensemble of trees, *etc.* Nevertheless, disregarding their nature, they all share the common structure defined in Equation (4.2). The solution of this problem can be achieved either by Empirical (or Structured) Risk Minimization (ERM) either by Maximum Likelihood/A Posteriori (MLE/MAP) Estimation. See Appendix A for more details on this two strategies, and their connection.

4.1.1 Regularization methods

Regularization methods is a broad class of models that include linear and nonlinear techniques for both regression and classification. The main characteristic of the methods falling in this class,

⁴source <https://losc.ligo.org/events/GW150914/>

⁵source <https://archive.ics.uci.edu/ml/datasets/HTRU2>

is that they are particularly straightforward to express as in Equation (4.2). In fact, as described in [Evgeniou et al., 2000], they are easily identifiable by the use of one loss function $L(f(\mathbf{x}), y)$ and one, or more, regularization penalty $\mathcal{R}(f)$.

In the following sections an overview of the most popular regularization methods is presented.

4.1.1.1 Ordinary least squares

We start this discussion focusing on linear models $\hat{y} = f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}$ and taking into account the most popular loss function for regression problems: the square loss $L(\hat{y}, y) = (\mathbf{x}^T \mathbf{w} - y)^2$. The data fitting problem expressed in Equation (4.3) is known as *Ordinary Least Squares* (OLS), or simply as *linear regression*, and it does not include any regularization term.

$$\hat{\mathbf{w}}_{\text{OLS}} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w} - y_i)^2 = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \|\mathbf{X} \mathbf{w} - \mathbf{y}\|_2^2 \quad (4.3)$$

The minimization problem in Equation (4.3) is convex and differentiable, hence its solution can be achieved in closed-form as

$$\hat{\mathbf{w}}_{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

or by iterative minimization routines such as (stochastic) gradient descent-like algorithms [Boyd and Vandenberghe, 2004; Sra et al., 2012]. A pictorial representation of the solution of OLS can be seen in Figure 4.12a.

In case of multiple regression tasks, the OLS approach can be extended to vector-valued regression as well. In this case the least squares problem can be written as

$$\begin{aligned} \hat{W}_{\text{OLS}} &= \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \sum_{t=1}^k (\mathbf{x}_i^T \mathbf{w}^t - y_i^t)^2 \\ &= \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \frac{1}{n} \|\mathbf{X} W - \mathbf{Y}\|_F^2 \end{aligned} \quad (4.4)$$

where $\|A\|_F = \sqrt{\sum_{i=1}^n \sum_{t=1}^k |a_{it}|^2}$ is the *Frobenius* norm (also known as *Hilbert-Schmidt* norm) and it can be considered as an extension of the ℓ_2 -norm to the matrix case. Lacking of appropriate regularization penalties, solving the problem in Equation (4.4) corresponds to solving k isolated regression problems, one for each task. The vector-valued OLS approach, even if theoretically legit, is rather uncommon in practical applications and a regularized version of Equation (4.4) is typically preferred (see following sections).

Even though mainly used for regression, the square loss can also be used to solve binary classification problems (see Table 4.1). In this case, it can be rewritten as

$$(\mathbf{x}^T \mathbf{w} - y)^2 = (1 - y \cdot \mathbf{x}^T \mathbf{w})^2 \quad (4.5)$$

exploiting the fact that the two classes are encoded with binary labels: $\mathbf{y} \in \{+1, -1\}^n$. For multiclass classification problems, strategies such as *One-vs-One* (OVO) or *One-vs-All* (OVA) can be adopted to reduce the problem to multiple binary classifications [Hastie et al., 2009].

OLS is probably the most naïve prediction strategy, nevertheless it is widely adopted in several studies. Let's see what happens when we use the OLS model on a real regression problem.

For this example, and the following ones, we take into account the dataset $\mathcal{D}_{\text{aging}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n=111}$ where each input sample $\mathbf{x}_i \in \mathbb{R}^{12}$ presents a set of measures describing the metabolic state of a healthy subject and $y_i \in \mathbb{N}_+$ is its age expressed in years. For the sake of this discussion a thorough description of $\mathcal{D}_{\text{aging}}$ at this point is irrelevant⁶, we can simply think as the $d = 12$ variables as predictors of the outcome y and we look for some linear input-output relationship. In order to do that, we randomly split $\mathcal{D}_{\text{aging}}$ in two chunks obtaining a training and a test set of $n_{\text{tr}} = 74$ and $n_{\text{ts}} = 37$ samples respectively. Then, we fit the OLS model on the training set obtaining the weights vector $\hat{\mathbf{w}}_{\text{OLS}}$ represented in Figure 4.5. As we can see, in order



Figure 4.5: A pictorial representation of the vector $\hat{\mathbf{w}}_{\text{OLS}}$ obtained fitting an OLS model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (*i.e.* directly proportional to the output) and a negative (*i.e.* inversely proportional) weight are represented in blue and red, respectively.

to achieve a predictive model, OLS can only spread the weights across all the input variables. Evaluating $\hat{\mathbf{w}}_{\text{OLS}}$ on the test set, this model has a Mean Absolute Error (MAE) of 10.598 years and explains the 74.29% of the variance.

This result looks promising, but we will see in the next sections whether they can be improved with the use of some regularization penalty.

4.1.1.2 Ridge regression

In its original proposition, *ridge regression* [Hoerl and Kennard, 1970] is defined as a least squares problem penalized by the squared ℓ_2 -norm of the regression coefficients, see Equation (4.6).

⁶this regression problem is widely described and analyzed in Chapter 6.

$$\mathcal{R}_{\ell_2}(\mathbf{w}) = \sum_{j=1}^d (w_j)^2 = \|\mathbf{w}\|_2^2 \quad (4.6)$$

Therefore, the ridge regression minimization problem can be written as in Equation (4.7).

$$\hat{\mathbf{w}}_{\ell_2} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w} - y_i)^2 + \lambda \sum_{j=1}^d (w_j)^2 = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \|X\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \quad (4.7)$$

This penalty leads to smooth solutions as it shrinks the coefficients toward zero, but it does not achieve a parsimonious representation, as it always keep all the variables in the model. The ridge regression problem of Equation (4.7) is convex and differentiable and a pictorial representation of its solution in a 2D case is depicted in Figure 4.12b. The ridge coefficients $\hat{\mathbf{w}}_{\ell_2}$ can be estimated in closed-form as

$$\hat{\mathbf{w}}_{\ell_2} = (X^T X + \lambda n I)^{-1} X^T \mathbf{y}$$

where I is the $d \times d$ identity matrix. An estimate for the ridge coefficients can also be obtained with gradient descent-like optimization routines [Boyd and Vandenberghe, 2004; Sra et al., 2012].

The regularization parameter λ plays the fundamental role of balancing the trade-off between data adherence and smoothness of the solution. Penalizing the ℓ_2 -norm of the regression coefficients, their value is shrunk toward zero. This results in an increased robustness of the solution to the noise affecting the training data.

In case of multiple outputs, the vector-valued ridge regression problem can be written as in Equation (4.8).

$$\begin{aligned} \hat{W}_{\ell_2} &= \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \sum_{t=1}^k (\mathbf{x}_i^T \mathbf{w}^t - y_i^t)^2 + \lambda \sum_{j=1}^d \sum_{t=1}^k |w_j^t|^2 \\ &= \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \frac{1}{n} \|XW - Y\|_F^2 + \lambda \|W\|_F^2 \end{aligned} \quad (4.8)$$

As already seen for vector-valued OLS, the Frobenius norm penalty does not induce any task coupling, hence solving the problem in Equation (4.8) still corresponds to individually solve k regression tasks.

This method can be applied to binary classification problems by using the margin loss function of Equation (4.5). Nevertheless, for ℓ_2 -norm penalized classification problems the use of the *logistic loss* is usually preferred (see Section 4.1.1.5).

In deep learning literature, penalizing the regression coefficients with the ℓ_2 -norm is known as *weight decay* [Krogh and Hertz, 1992]. Ridge regression can also be considered a form of *Tikhonov regularization* [Tikhonov, 1963] and of *regularization network* [Evgeniou et al., 2000].

Let's see what happens when this method is applied to a real regression problem. For ease of comparison, we take into account the dataset $\mathcal{D}_{\text{aging}}$, introduced in Section 4.1.1.1. Compared to OLS, ridge regression has the parameter λ that must be fixed before fitting the model. In this

example, we estimated the best value $\hat{\lambda}_{cv}$ according to a standard grid-search cross-validation strategy [Hastie et al., 2009]. This consists in fixing a range of (30) possible values for λ (in a logarithmic scale from 10^{-3} to 10^2) and pick the best value as the one achieving the lowest validation error, estimated via (5-fold) cross-validation. Therefore, once the best value for the regularization parameter is fixed ($\hat{\lambda}_{cv} = 20.43$), the experimental setup used for OLS is preserved. The ridge coefficients $\hat{\mathbf{w}}_{\ell_2}$ are represented in Figure 4.6. Comparing Figure 4.6 and

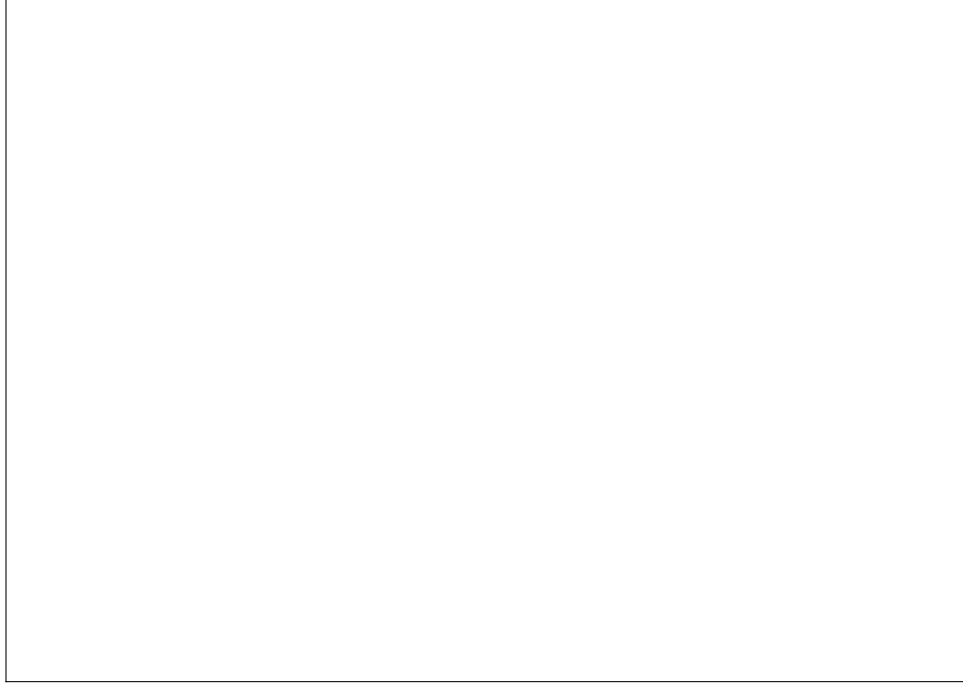


Figure 4.6: A pictorial representation of the vector $\hat{\mathbf{w}}_{\ell_2}$ obtained fitting a ridge regression model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (*i.e.* directly proportional to the output) and a negative (*i.e.* inversely proportional) weight are represented in blue and red, respectively.

Figure 4.5 we can see that the amplitude of the ridge regression coefficients is, in absolute value, decreased by the use of the ℓ_2 penalty. Indeed, several entries of $\hat{\mathbf{w}}_{\ell_2}$ are very small, but none of them is exactly zero. This is the expected behavior of the ℓ_2 -norm penalty. Evaluating $\hat{\mathbf{w}}_{\ell_2}$ on the test set, this model has $\text{MAE} = 8.615$ years and explains the 81.19% of the variance, outperforming OLS.

4.1.1.3 Lasso

The Lasso [Tibshirani, 1996] can be defined as a least square problem penalized by the ℓ_1 -norm of the regression coefficients, see Equation (4.9).

$$\mathcal{R}_{\ell_1}(\mathbf{w}) = \sum_{j=1}^d |w_j| = |\mathbf{w}|_1 \quad (4.9)$$

Therefore, the Lasso minimization problem can be written as in Equation (4.10).

$$\hat{\mathbf{w}}_{\ell_1} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w} - y_i)^2 + \lambda \sum_{j=1}^d |w_j| = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_1 \quad (4.10)$$

The Lasso can be used to perform linear model fitting and, thanks to its desirable properties, it is a popular choice for embedded variable selection [Guyon and Elisseeff, 2003], see Section 4.2. At first, the ℓ_1 -norm enforces sparsity in the solution, hence producing compact and easily interpretable results. Secondly, the Lasso optimization problem is convex and, although non-differentiable, it is computationally feasible even in very high dimensional scenarios. Popular minimization algorithms for the Lasso problem are, for instance, the *Fast Iterative Shrinkage-Thresholding Algorithm* (FISTA) [Beck and Teboulle, 2009] and the *coordinate descent algorithm* [Wu and Lange, 2008]. A pictorial representation of the Lasso solution can be seen in Figure 4.12c.

A popular application of the Lasso is to perform shrinkage and variable selection in survival analysis for Cox proportional hazard regression [Tang et al., 2017; Gui and Li, 2005; Tibshirani et al., 1997] and additive risk models [Ma and Huang, 2007]. Such ℓ_1 -penalized methods are extensively applied in literature to predict survival time from molecular data collected from patients affected by different kinds of tumor.

The Lasso can also be extended to vector-valued regression problems by using the mixed $L_{2,1}$ -norm, defined in Equation (4.11), as regularization penalty [Gramfort et al., 2012].

$$\mathcal{R}_{\ell_1}(W) = \sum_{j=1}^d \sqrt{\left(\sum_{t=1}^k |w_j^t|^2 \right)} = \|W\|_{2,1} \quad (4.11)$$

Therefore, the vector-valued regularization problem can be written as in Equation (4.12)

$$\begin{aligned} \hat{W}_{\ell_1} &= \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \sum_{t=1}^k (\mathbf{x}_i^T \mathbf{w}^t - y_i^t)^2 + \lambda \sum_{j=1}^d \sqrt{\left(\sum_{t=1}^k |w_j^t|^2 \right)} \\ &= \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \frac{1}{n} \|\mathbf{X}W - \mathbf{Y}\|_F^2 + \lambda \|W\|_{2,1} \end{aligned} \quad (4.12)$$

and it is known as Multi-task Lasso [Lee et al., 2010]. Such norm enforces a row-structured sparsity in the regression weights, hence preserving the interpretability of the solution.

Originally proposed to solve regression problems, the Lasso can also be adopted in binary classification tasks; although, in this case, *sparse logistic regression* (see Section (4.1.1.5)) is often preferred [Wu et al., 2009].

Let's see what happens when the Lasso model is applied to a real regression problem. Once again, for ease of comparison with OLS and ridge, we take into account the dataset $\mathcal{D}_{\text{aging}}$, introduced in Section 4.1.1.1. The experimental setup in this case is identical to the one previously applied for ridge regression. The best value for the regularization parameter, chosen via grid-search cross-validation, is $\hat{\lambda}_{\text{cv}} = 1.27$. The Lasso coefficients $\hat{\mathbf{w}}_{\ell_1}$ are represented in Figure 4.7. Comparing $\hat{\mathbf{w}}_{\ell_1}$ with $\hat{\mathbf{w}}_{\ell_2}$ and $\hat{\mathbf{w}}_{\text{OLS}}$ (Figure 4.7, Figure 4.6 and Figure 4.5, respectively) we can observe that, for the first time, to only 6 variables, out of 12, a non-negative value is assigned. This is an example of the *sparsity*-enforcing effect of the ℓ_1 -norm regularization penalty. The 6

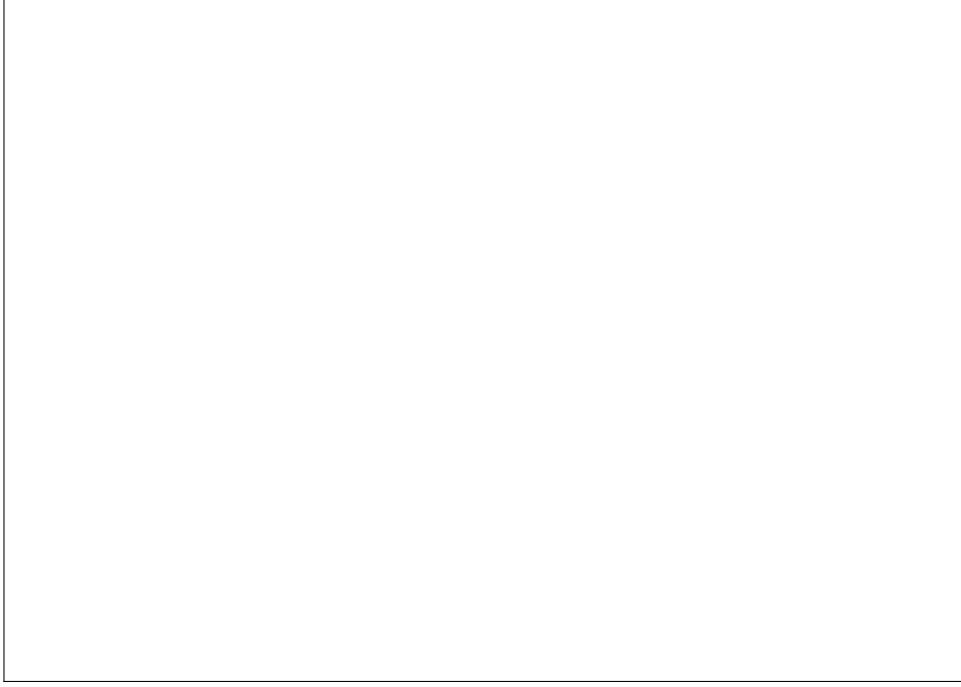


Figure 4.7: A pictorial representation of the vector $\hat{\mathbf{w}}_{\ell_1}$ obtained fitting a Ridge model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (*i.e.* directly proportional to the output) and a negative (*i.e.* inversely proportional) weight are represented in blue and red, respectively.

variables with nonzero weight can be considered as *selected* for the prediction problem at hand. Evaluating $\hat{\mathbf{w}}_{\ell_1}$ on the test set, the Lasso has $\text{MAE} = 8.387$ years and explains the 81.51% of the variance, slightly outperforming ridge.

Using sparsity-enforcing penalties, such as the ℓ_1 -norm, an insightful experiment is the analysis of the regularization path. This can be done by iteratively fitting the model with decreasing values of the regularization parameter λ , usually expressed in logarithmic scale. An example of the Lasso path for the aging problem is reported in Figure 4.8. Weights corresponding to features that are more likely to be relevant for the prediction problem should early move away from the horizontal axis. Conversely, as the regularization parameter increases, the model should enforce less sparsity, hence tolerating more and more irrelevant features having nonzero weight. The vertical dashed line in Figure 4.8 corresponds to $\hat{\lambda}_{\text{cv}}$ and it hits the profiles of the weights consistently with what shown in Figure 4.7.

When used for variable selection, the Lasso has two major drawbacks. First, in presence of groups of correlated variables, this method tends to select only one variable per group, ignoring the others. Secondly, the method cannot select more variables than the sample size [Waldmann et al., 2013; De Mol et al., 2009b]. The effect of such drawbacks is dramatic when using the Lasso in $n \ll d$ scenarios. In order to ameliorate this issues, several Lasso-inspired models were proposed [Meinshausen and Bühlmann, 2010; Hoggart et al., 2008; Zou, 2006]. In the next section we will describe one of the most popular and straightforward Lasso extensions: the Elastic-Net [Zou and Hastie, 2005].

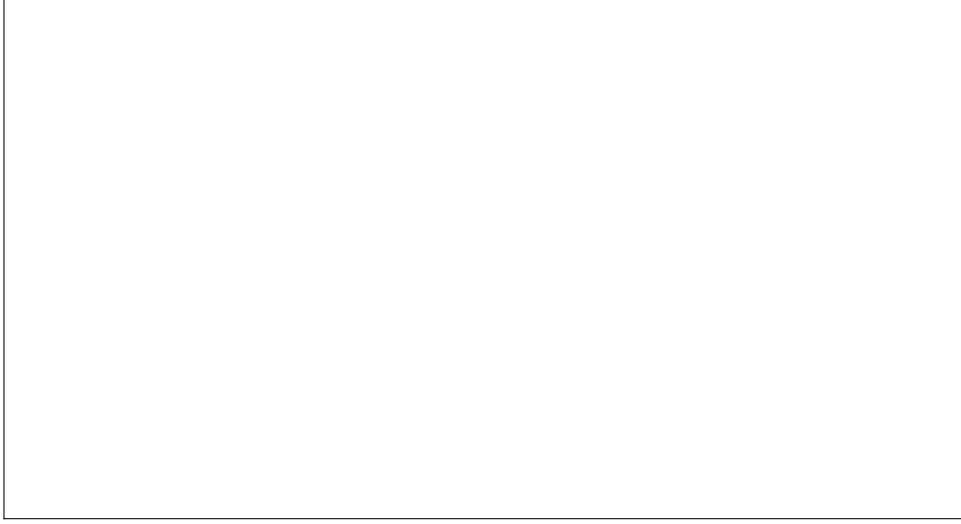


Figure 4.8: Profiles of the Lasso coefficients for the aging problem as λ decreases. The vertical dashed line represents the optimal value $\hat{\lambda}_{cv}$ estimated by grid-search (5-fold) cross-validation.

4.1.1.4 Elastic-Net

The Elastic-Net [Zou and Hastie, 2005; De Mol et al., 2009a] method can be formulated as a least squares problem penalized by a convex combination of Lasso (ℓ_1 -norm) and ridge regression (ℓ_2 -norm) penalties, as in Equation (4.13).

$$R_{\ell_1\ell_2}(\mathbf{w}) = \sum_{j=1}^d (\alpha |w_j| + (1 - \alpha) w_j^2) = \alpha \|\mathbf{w}\|_1 + (1 - \alpha) \|\mathbf{w}\|_2^2 \quad (4.13)$$

Therefore, the Elastic-Net minimization problem can be written as in Equation (4.14)

$$\begin{aligned} \hat{\mathbf{w}}_{\ell_1\ell_2} &= \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w} - y_i)^2 + \lambda \left[\sum_{j=1}^d (\alpha |w_j| + (1 - \alpha) w_j^2) \right] \\ &= \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda [\alpha \|\mathbf{w}\|_1 + (1 - \alpha) \|\mathbf{w}\|_2^2] \end{aligned} \quad (4.14)$$

with $0 \leq \alpha \leq 1$, or equivalently as

$$\hat{\mathbf{w}}_{\ell_1\ell_2} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \tau \|\mathbf{w}\|_1 + \mu \|\mathbf{w}\|_2^2 \quad (4.15)$$

where $\tau = \lambda\alpha$ and $\mu = \lambda(1 - \alpha)$. The first formulation of the problem, Equation (4.14), is more convenient when we want to control the overall amount of regularization with λ , given a certain amount of sparsity α . In fact, it is easy to see that fitting the Elastic-Net model for $\alpha = 0$ or $\alpha = 1$ is equivalent to solve ridge or Lasso regression, respectively. On the other hand, writing the Elastic-Net problem as in Equation (4.15) is more convenient when we want to separately control the ℓ_1 - and ℓ_2 -norm penalties, as in [De Mol et al., 2009b].

The Elastic-Net⁷ model is widely adopted to perform linear model fitting and variable selection.

⁷actually, in the original paper the authors refer to Equations (4.14) and (4.15) as *naïve* Elastic-Net [Zou and Hastie, 2005] because empirical evidence show that the weights $\hat{\mathbf{w}}_{\ell_1\ell_2}$ may suffer from over-shrinking when appropriate rescaling strategies, also described in [De Mol et al., 2009b], are not applied.

Indeed, the combined presence of the two norms promotes sparse solutions where groups of correlated variables can be simultaneously selected, hence overcoming the variable selection drawbacks of the Lasso and making the Elastic-Net suitable for variable selection in $n \ll d$ scenarios. As already seen for the Lasso, the minimization problem in Equation (4.14) is convex and non-differentiable, due to the ℓ_1 -norm, and it can be efficiently solved either by proximal forward-backward splitting strategies (*e.g.* FISTA [Beck and Teboulle, 2009]), or by coordinate descent [Wu and Lange, 2008]. A pictorial representation of the Elastic-Net solution can be seen in Figure 4.12d.

The Elastic-Net method is successfully applied in several biomedical fields, including gene expression [Hughey and Butte, 2015; De Mol et al., 2009b], genome-wide association studies [Waldmann et al., 2013] and other molecular data [Aben et al., 2016; Hughey and Butte, 2015].

The Elastic-Net can also be extended to vector-valued regression problems by using a convex combination of the $L_{2,1}$ - and the Frobenius norms. This multiple output regression problem is known as Multi-task Elastic-Net [Chen et al., 2012] can be written as in Equation (4.16).

$$\begin{aligned}\hat{W}_{\ell_1\ell_2} &= \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \sum_{t=1}^k (\mathbf{x}_i^T \mathbf{w}^t - y_i^t)^2 + \lambda \left[\alpha \sum_{j=1}^d \sqrt{\left(\sum_{t=1}^k |w_j^t|^2 \right)} + (1 - \alpha) \sum_{j=1}^d \sum_{t=1}^k |w_j^t|^2 \right] \\ &= \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \frac{1}{n} \|XW - Y\|_F^2 + \lambda [\alpha \|W\|_{2,1} + (1 - \alpha) \|W\|_F^2]\end{aligned}\tag{4.16}$$

Even though originally proposed to solve regression problems, as already seen for Lasso and ridge regression, the Elastic-Net can be adopted in binary classification tasks. Nevertheless, for classification problems, the use of the logistic loss is usually preferred.

Let's see what happens when the Elastic-Net model is applied to an actual regression problem. As usually, we tackle the aging task (introduced in Section 4.1.1.1). The experimental setup in this case is the same already adopted for the Lasso in Section 4.1.1.3. The only difference is that the grid-search cross-validation routine looks for the optimal values for the two regularization parameters $(\hat{\lambda}_{cv}, \hat{\alpha}_{cv}) = (0.57, 0.48)$ in a 2D grid consisting of 30×30 values (α candidates range from 0 to 1 in a linear scale, whilst λ candidates range from 10^{-3} to 10^2 in a logarithmic scale). The Elastic-Net coefficients $\hat{\mathbf{w}}_{\ell_1\ell_2}$ are represented in Figure 4.9.

Comparing $\hat{\mathbf{w}}_{\ell_1\ell_2}$ in Figure 4.9 with $\hat{\mathbf{w}}_{\ell_1}$ in Figure 4.7 we can notice that in the Elastic-Net solution 10 variables have nonzero weight, that is more with respect to the 6 of the Lasso. This is the expected behavior of the added ℓ_2 -norm, which helps the model to select groups of collinear variables. Evaluating the Elastic-Net solution on the test set, $\hat{\mathbf{w}}_{\ell_1\ell_2}$ achieves MAE = 8.321 years explaining the 82.13%, slightly outperforming Lasso, hence ranking first in this little challenge of linear square loss-based regression methods.

As already seen for the Lasso, we can inspect the Elastic-Net weights path, obtained fixing $\alpha = \hat{\alpha}_{cv}$ for decreasing values of λ , see Figure 4.10. The vertical dashed line in Figure 4.10 corresponds to $\hat{\lambda}_{cv}$ and it hits the profiles of the weights consistently with the Elastic-Net solution represented in Figure 4.7. The Elastic-Net, compared to the Lasso, produces smoother regularization paths in which the allegedly correlated variables enter the solution earlier.

Another comparison between the behavior of OLS, ridge, Lasso and Elastic-Net regression is presented in Figure 4.11. The four solutions achieved for the same aging regression problem are

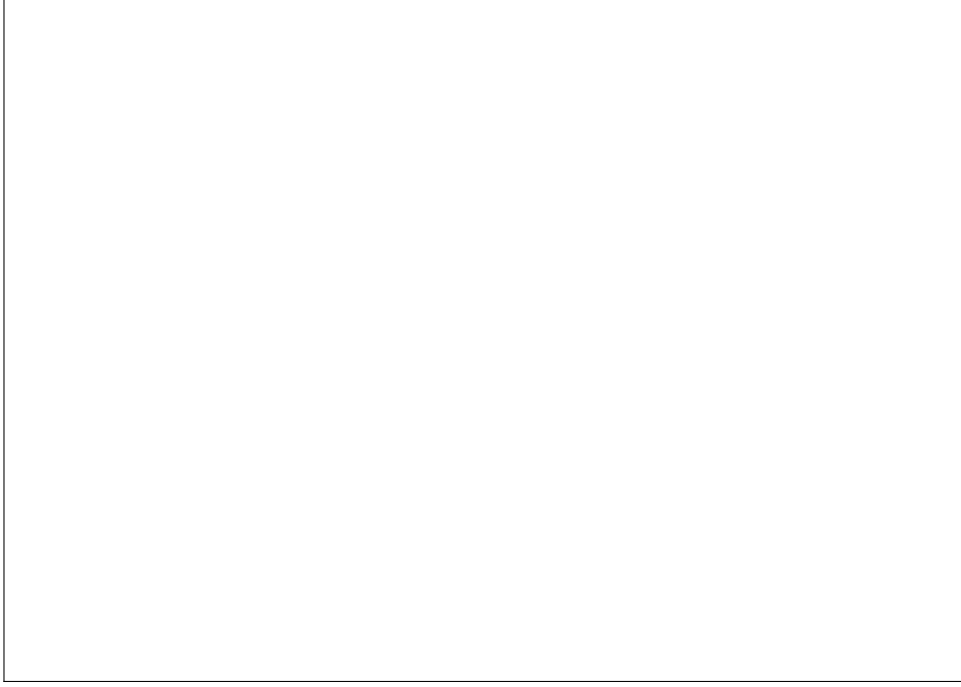


Figure 4.9: A pictorial representation of the vector $\hat{\mathbf{w}}_{\ell_1 \ell_2}$ obtained fitting a Elastic-Net model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (*i.e.* directly proportional to the output) and a negative (*i.e.* inversely proportional) weight are represented in blue and red, respectively.



Figure 4.10: Profiles of the Elastic-Net coefficients for the aging problem as λ decreases. The vertical dashed line represents the optimal value $\hat{\lambda}_{\text{cv}}$ estimated by grid-search (5-fold) cross-validation.

represented in a scatter plot where horizontal and vertical axis represents their ℓ_2 - and ℓ_1 -norm, respectively. As expected,

1. the unpenalized solution $\hat{\mathbf{w}}_{\text{OLS}}$ shows the highest values for the two norms and it is placed in the top-right side of the plot,
2. the ridge solution $\hat{\mathbf{w}}_{\ell_2}$ has lowest ℓ_2 -norm,

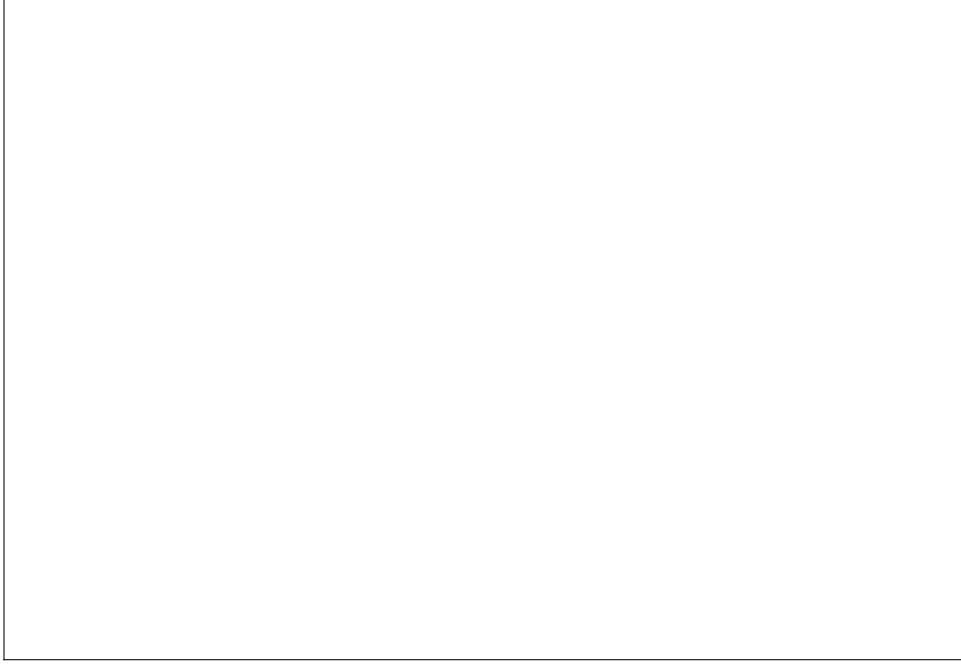


Figure 4.11: A comparison of the value of the ℓ_1 and ℓ_2 norms of the weights obtained by OLS, ridge, Lasso and Elastic-Net.

3. the Lasso solution $\hat{\mathbf{w}}_{\ell_1}$ has lowest ℓ_1 -norm and
4. the Elastic-Net solution shows the lowest values for the two norms and it is placed in the bottom-left side of the plot.

This is consistent with the type of regularization imposed to each method. Interestingly, in this case, the method that performs better on the test set (Elastic-Net) has lowest norms.

The Elastic-Net penalty is not the only method in which sparsity is enforced on a group level. For example, (overlapping) group Lasso and graph Lasso penalties can be applied when the variables are naturally partitioned in (overlapping) groups, or when their interaction can be modeled by a graph. A detailed description of these methods is beyond the scope of this thesis and we refer to [Jacob et al., 2009; Witten and Tibshirani, 2009] and references therein for their comprehensive description.

4.1.1.5 Logistic Regression

Logistic regression is one of the most popular linear methods for classification problems⁸. In its unpenalized form, it can be simply seen as the problem of minimizing the logistic loss (see Table 4.1) on a given training dataset. Therefore, logistic regression can be written as in Equation (4.17),

$$\hat{\mathbf{w}}_{\text{LR}} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{x}_i^T \mathbf{w}}) \quad (4.17)$$

where the labels $y_i \in \{+1, -1\}$, $\forall i = 1, \dots, n$. The minimization problem above is convex and differentiable, although as the gradient is nonlinear in \mathbf{w} , it does not have a closed-form

⁸as counter-intuitive as it sounds

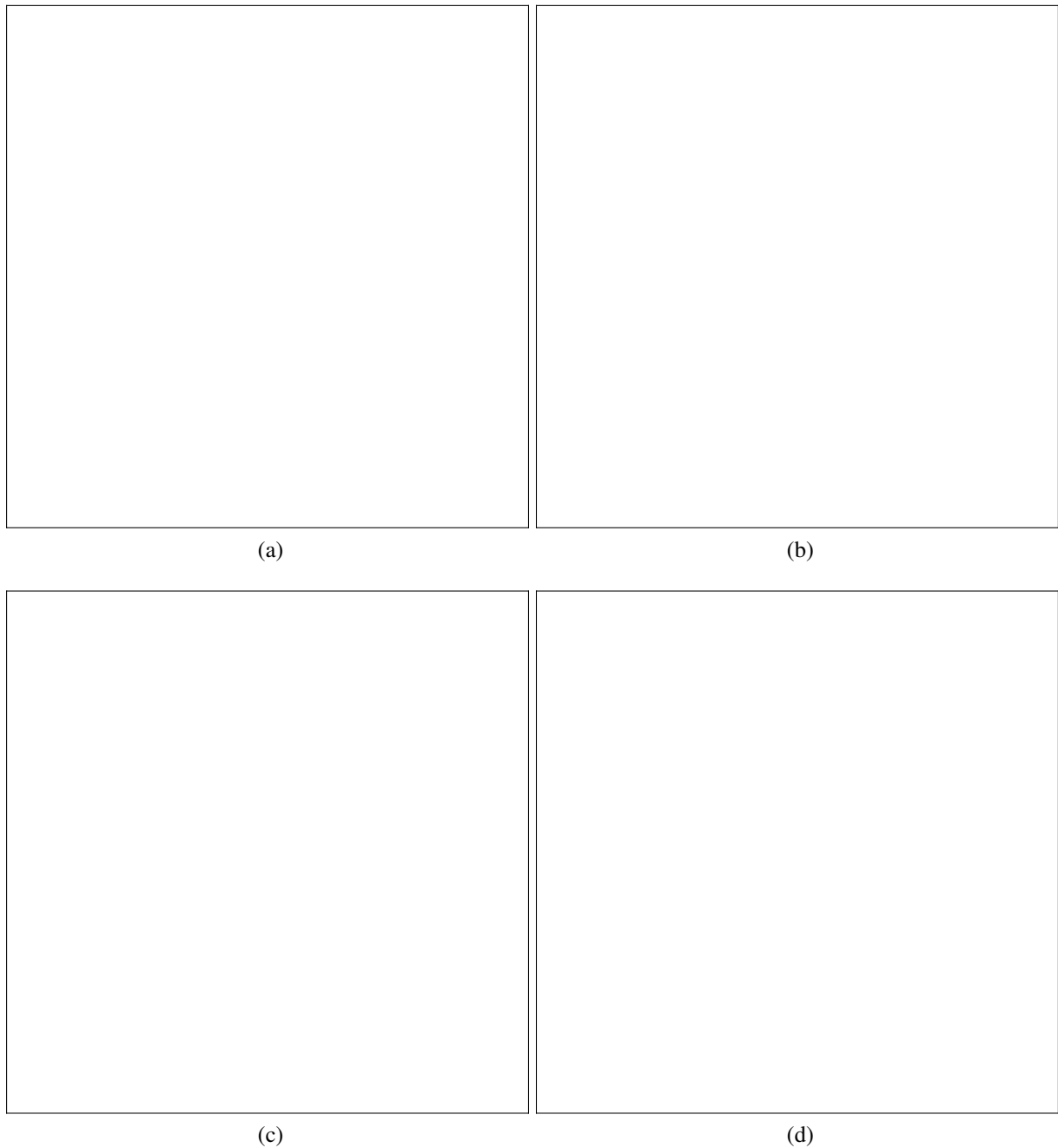


Figure 4.12: Pictorial representation of the contour lines of the square loss in a 2D regression problem with various penalties: (a) ordinary least squares (no penalty), (b) ridge regression (ℓ_2 -norm penalty), (c) the Lasso (ℓ_1 -norm penalty) and finally (d) the Elastic-Net (ℓ_1 - and ℓ_2 -norm penalties).

solution. So, unpenalized logistic regression problems are typically solved by (stochastic) gradient descent-like techniques [Boyd and Vandenberghe, 2004; Sra et al., 2012].

Unpenalized logistic regression, although theoretically sound, is somewhat uncommon in recent real-world studies. As already seen for the square loss, regularization penalties are typically used. For instance, we can have ℓ_2 -regularized logistic regression,

$$\hat{\mathbf{w}}_{\ell_2} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{x}_i^T \mathbf{w}}) + \lambda \|\mathbf{w}\|_2^2 \quad (4.18)$$

or various forms of sparse logistic regression such as

$$\hat{\mathbf{w}}_{\ell_1} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{x}_i^T \mathbf{w}}) + \lambda \|\mathbf{w}\|_1 \quad (4.19)$$

which uses the Lasso penalty, or

$$\hat{\mathbf{w}}_{\ell_1 \ell_2} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{x}_i^T \mathbf{w}}) + \lambda [\alpha \|\mathbf{w}\|_1 + (1 - \alpha) \|\mathbf{w}\|_2^2] \quad (4.20)$$

which uses the Elastic-Net penalty. The sparse logistic regression minimization problem can be efficiently solved either by proximal forward-backward splitting strategies (e.g. FISTA [Beck and Teboulle, 2009]), or by coordinate descent [Wu and Lange, 2008].

Moreover, multi-class classification with logistic regression can be achieved by OVO, AVA approaches, as well as with the multiclass generalization of logistic regression: *softmax* regression [Hastie et al., 2009] (also known as multinomial logistic regression), which can be expressed as

$$\hat{W}_{\text{LR}} = \underset{W \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} -\frac{1}{n} \left[\sum_{i=1}^n \sum_{t=1}^k \mathbb{1}\{y_i = t\} \log \left(\frac{e^{\mathbf{x}_i^T \mathbf{w}^t}}{\sum_{l=1}^k e^{\mathbf{x}_i^T \mathbf{w}^l}} \right) \right] \quad (4.21)$$

where the indicator function $\mathbb{1}\{\cdot\}$ includes in the functional only the correctly classified samples.

4.1.1.6 Support Vector Machine

Support Vector Machine (SVM) is a class of powerful algorithms that can be written as a penalized model and that can be used for both regression (SVR) and classification (SVC) problems [Evgeniou et al., 2000]. In the first case, the adopted loss function is Vapnik's ε -insensitive loss:

$$|y - \mathbf{x}^T \mathbf{w}|_\varepsilon = \begin{cases} 0 & \text{if } |y - \mathbf{x}^T \mathbf{w}| < \varepsilon \\ |y - \mathbf{x}^T \mathbf{w}| - \varepsilon & \text{otherwise} \end{cases} \quad (4.22)$$

and in the second case the Hinge loss:

$$|1 - y \mathbf{x}^T \mathbf{w}|_+ = \max[0, 1 - y \mathbf{x}^T \mathbf{w}] \quad (4.23)$$

as reported in Table 4.1 and shown in Figures 4.4c 4.2b. The standard formulation of SVM is penalized with the ℓ_2 -norm [Vapnik, 2013]. Therefore, sticking to linear models, the SVR minimization problem can be written as

$$\hat{\mathbf{w}}_{\text{SVR}} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n |y_i - f(\mathbf{x}_i)|_\varepsilon + \lambda \|\mathbf{w}\|_2^2 \quad (4.24)$$

while the SVC minimization problem is

$$\hat{\mathbf{w}}_{\text{SVC}} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n |1 - y_i f(\mathbf{x}_i)|_+ + \lambda \|\mathbf{w}\|_2^2 \quad (4.25)$$

where, as usually, λ controls the trade-off between data adherence and smoothing of the solution. Equations (4.24) and (4.25) are known as the *primal* SVM problem. However, in order to generalize the SVM to nonlinear cases (see Section 4.1.2), it is often convenient to transform this minimization problem in its *dual* form. Several algorithms to find the solution of the SVM minimization problem in both primal and dual forms were developed, such as Newton's method or coordinate descent algorithm, see [Smola and Schölkopf, 2004; Shawe-Taylor and Sun, 2011] for an exhaustive review.

The standard formulation of SVM does not cope well with high-dimensional data, as no sparsity-enforcing penalty is adopted. Recently, ℓ_1 -norm penalized SVM were proposed as well in [Zhu et al., 2004; Peng et al., 2016].

As for the square loss-based methods, let's apply (ℓ_2 -penalized) SVR to the aging problem (introduced in Section 4.1.1.1). The experimental setup in this case is the same already adopted for Lasso and ridge regression (see Sections 4.1.1.3 4.1.1.2). The weights $\hat{\mathbf{w}}_{\text{SVR}}$ are represented in Figure 4.13. As expected, none of the is exactly zero and they look similar to ridge coefficients in Figure 4.6, as the two model share the same regularization penalty. Evaluating $\hat{\mathbf{w}}_{\text{SVR}}$ on the



Figure 4.13: A pictorial representation of the vector $\hat{\mathbf{w}}_{\text{SVR}}$ obtained fitting a SVR model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (*i.e.* directly proportional to the output) and a negative (*i.e.* inversely proportional) weight are represented in blue and red, respectively.

test set, the SVR model has $\text{MAE} = 9.280$ and explains the 77.74% of the variance.

4.1.2 The kernel trick

For ease of reading, all the regularization methods presented so far were focused on learning linear input-output relationships $y = \mathbf{x}^T \mathbf{w}$. Actually, they can all be extended to the nonlinear case exploiting the so-called *kernel trick*.

The basic idea of the kernel trick is to use a map that projects the features on a higher (possibly infinite) dimensional space in which the prediction problem is, to some extent, easier to solve. An example of kernel trick for nonlinear classification problem is presented in Figure 4.14.

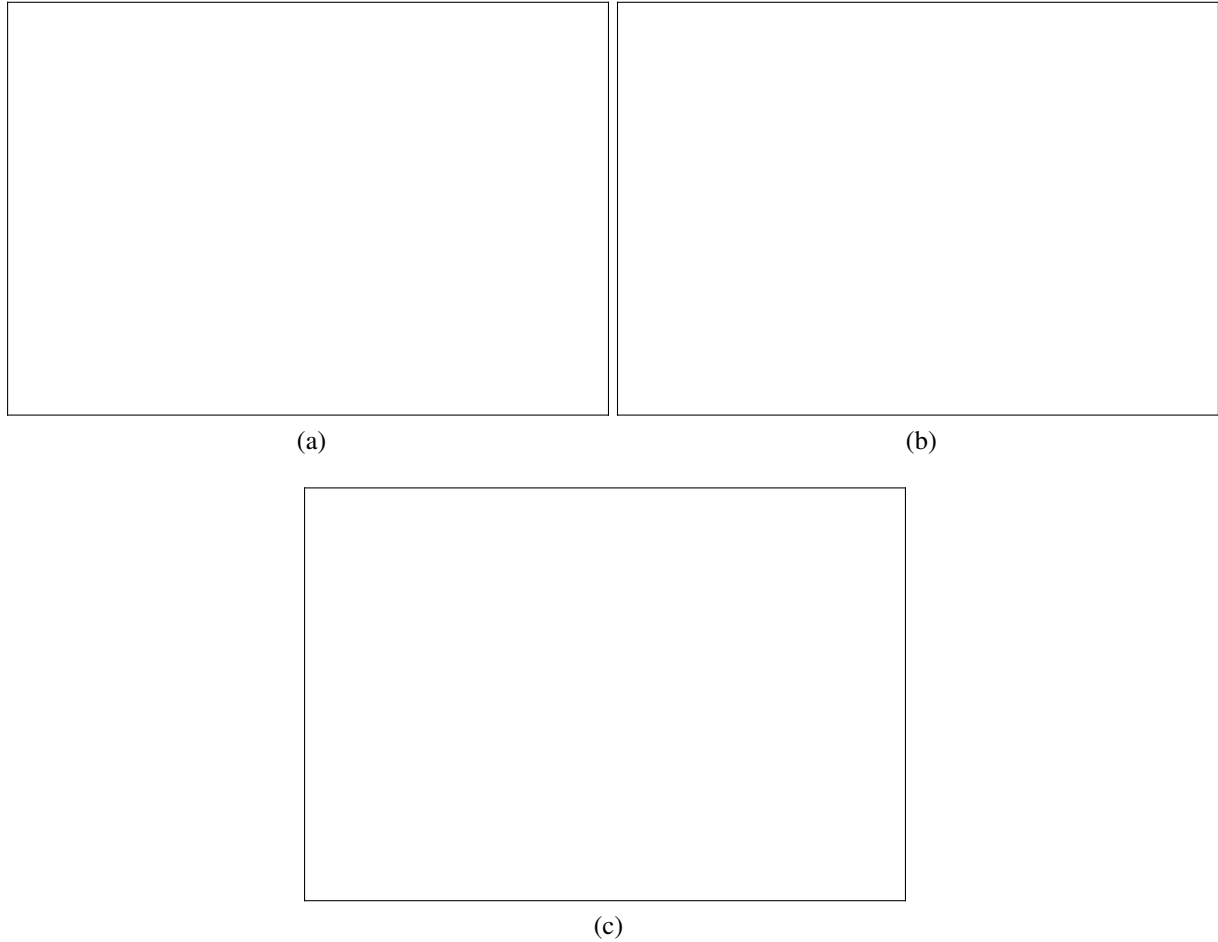


Figure 4.14: Pictorial representation of the kernel trick. Panel (a) shows a 2D classification problem where the input-output relationship is nonlinear. Panel (b) shows the *kernel explosion*, *i.e.* the projection of the 2D problem in a higher (3D) dimensional space in which the two classes are linearly separable, as shown in Panel (c).

We define the nonlinear feature mapping function as $\phi : \mathcal{X} \rightarrow \mathcal{F}$; for instance $\mathcal{X} \in \mathbb{R}^d$ and $\mathcal{F} \in \mathbb{R}^p$ with $p \geq d$, or even $p \gg d$. Given ϕ , the representer theorem [Smola and Schölkopf, 1998], states that the learning problem of Equation (4.2) admits a solution of the form

$$\hat{f}(\mathbf{x}) = \sum_{i=1}^n k(\mathbf{x}_i, \mathbf{x}_j) \alpha_i$$

where $\alpha \in \mathbb{R}^n$ and the function k behaves like an inner product, *i.e.* it is: symmetric, positive

definite, and it can be defined as in Equation (4.26).

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \cdot \phi(\mathbf{x}_j) \quad (4.26)$$

In the ML literature, k is typically called the *kernel* function and it comes in handy to solve the dual version of the learning problems.

Let's see a practical example: the kernel ridge regression. We recall that standard ridge regression problem can be rewritten as

$$\hat{\mathbf{w}}_{\ell_2} = \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} J(\mathbf{w})$$

where the objective function $J(\mathbf{w})$ is

$$J(\mathbf{w}) = \frac{1}{n} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 = \frac{1}{n} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$$

Therefore, the solution of the ridge regression problem can be evaluated by imposing $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = 0$ which leads to the following linear system of equations.

$$\mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{X}^T \mathbf{y} + \lambda n \mathbf{w} = 0 \Rightarrow (\mathbf{X}^T \mathbf{X} + \lambda n \mathbf{I}) \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

This can either be solved with respect to \mathbf{w} (see Section 4.1.1.2) obtaining the solution of the *primal* form

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda n \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \quad (4.27)$$

or it can be rewritten as

$$\mathbf{w} = \frac{1}{\lambda n} \mathbf{X}^T (\mathbf{y} - \mathbf{X} \mathbf{w}) = \mathbf{X}^T \boldsymbol{\alpha}$$

where $\boldsymbol{\alpha} = \frac{1}{\lambda n} (\mathbf{y} - \mathbf{X} \mathbf{w})$ is the *dual* variable. Therefore, the dual formulation of the problem becomes

$$\begin{aligned} \boldsymbol{\alpha} &= \frac{1}{\lambda n} (\mathbf{y} - \mathbf{X} \mathbf{w}) \\ &\Rightarrow \lambda n \boldsymbol{\alpha} = (\mathbf{y} - \mathbf{X} \mathbf{w}) \\ &\Rightarrow \lambda n \boldsymbol{\alpha} = (\mathbf{y} - \mathbf{X} \mathbf{X}^T \boldsymbol{\alpha}) \\ &\Rightarrow (\mathbf{X} \mathbf{X}^T + \lambda n \mathbf{I}) \boldsymbol{\alpha} = \mathbf{y} \\ &\Rightarrow \boldsymbol{\alpha} = (\mathbf{K} + \lambda n \mathbf{I})^{-1} \mathbf{y} \end{aligned} \quad (4.28)$$

where \mathbf{K} is the $n \times n$, symmetric and positive semi-definite kernel matrix, with entries $K_i^j = k(\mathbf{x}_i, \mathbf{x}_j)$. In this case $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$, and this corresponds to the linear kernel. Nevertheless, several other options are available (see Table 4.2 or [Bishop, 2006] for more details). Choosing a different kernel for ridge regression simply boils down to choose a different formulation for the kernel function k , hence plugging the obtained kernel matrix \mathbf{K} in Equation (4.28) to achieve the desired solution.

The advantage of the kernel trick is not only statistical, but also computational. Indeed, we may notice that solving the primal problem (*i.e.* in \mathbf{w}) requires $O(n^3)$ operations, while solving the dual (*i.e.* in $\boldsymbol{\alpha}$) requires $O(d^3)$ operations which is better for large scale and relatively low-dimensional problem.

A similar derivation can also be obtained applying the kernel trick to logistic regression and SVM [Bishop, 2006; Shawe-Taylor and Cristianini, 2004].

Table 4.2: Popular kernel functions. RBF stands for Radial Basis Function.

name	formulation
linear	$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$
polynomial	$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^d$
RBF	$k(\mathbf{x}, \mathbf{x}') = e^{-\gamma \ \mathbf{x} - \mathbf{x}'\ ^2}$
sigmoid	$k(\mathbf{x}, \mathbf{x}') = \tanh(\gamma \mathbf{x}^T \mathbf{x}' + c)$

Table 4.3: Overview of the matrix norms used for multiple-output regression.

Matrix norm	Notation	Definition
Frobenius	$\ A\ _F$	$\sqrt{\text{trace}(A^T A)}$
Nuclear	$\ A\ _*$	$\text{trace} \sqrt{(A^T A)}$
Mixed $L_{2,1}$	$\ A\ _{2,1}$	$\sum_j \ \mathbf{a}_j\ _2$

4.1.3 Decision trees

Decision trees are simple, yet powerful, methods for regression and classification. They are based on recursive binary splits of the feature space, in which they fit a very simple model, such as a constant. Fitting a decision tree on a given dataset results in partitioning the feature space in *cuboid* regions with edges aligned to the axes. In each region, there is a separate model to predict the target variable. In regression problems this can be the average of the samples falling in the region, whereas for classification problems, it can be the assignment to the most represented class. In particular, when decision trees are applied to classification problems, the feature space partitioning aims at keeping samples with the same labels grouped together. In this thesis, we refer to one of the most popular method for decision tree-based classification and regression, known as CART [Breiman et al., 1984].

An example of application of this technique for classification and regression is shown in Figure 4.15.

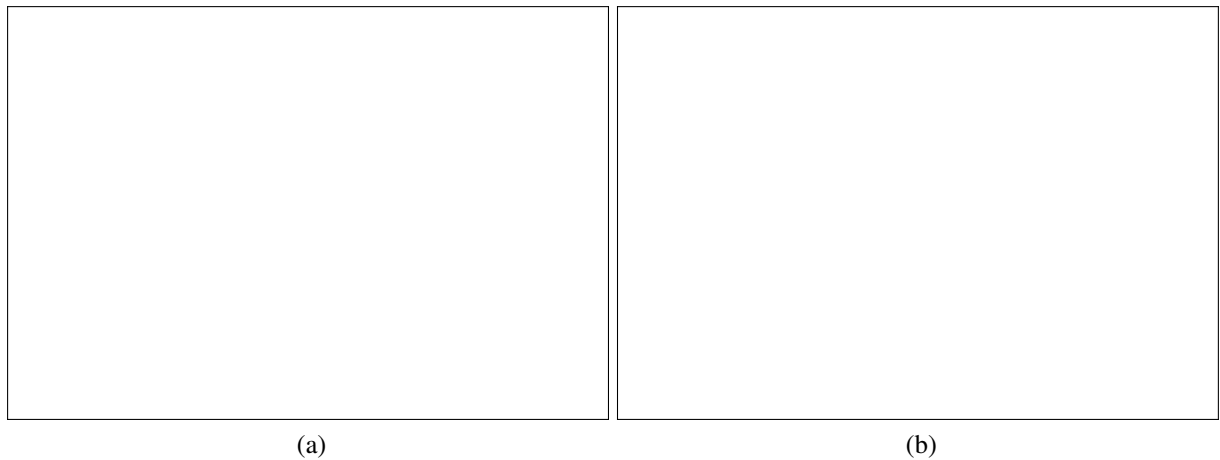


Figure 4.15: Two examples of decision tree applications for: (a) 2D multiclass classification problem, (b) 1D regression problem.

Fitting a decision tree implies learning from the data the structure of the tree itself, including the selection of the input variable for each splitting recursion, as well as the optimal splitting threshold. Moreover, it must also be defined the value of the prediction for each region. Let's take into account the usual setting: $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n = (X, \mathbf{y})$. Even when the number of nodes is fixed, defining at the same time: which variables to split, their splitting threshold and the predictive value is computationally unfeasible as it has a combinatorially large number of possible solutions. Therefore, starting from the root node, *i.e.* the whole input space, and fixing a maximum tree depth, we resort to a greedy approximation.

Let the data at the node m be denoted by Q_m , then for each candidate split $\theta = (j, s_m)$, consisting of the j^{th} feature and the threshold s_m , we define two half-planes as follows.

$$Q_{m_1}(\theta) = (\mathbf{x}_i, y_i) | \mathbf{x}_i^j \leq s_m \text{ and } Q_{m_2}(\theta) = (\mathbf{x}_i, y_i) | \mathbf{x}_i^j > s_m$$

Then, at each node an impurity measure is evaluated using a suitable impurity function $H(Q_m)$, which depends on the given task. Therefore, we can formulate an objective function like

$$J(Q_m, \theta) = \frac{n_1}{n_m} H(Q_{m_1}(\theta)) + \frac{n_2}{n_m} H(Q_{m_2}(\theta))$$

where n_1 and n_2 is the number of samples in region Q_{m_1} and Q_{m_2} , respectively, while $n_m = n_1 + n_2$. For each node we then select the best parameter solving the optimization problem in Equation (4.29).

$$\hat{\theta}_m = \underset{\theta}{\operatorname{argmin}} J(Q_m, \theta) \quad (4.29)$$

This procedure is recursively applied to $Q_{m_1}(\hat{\theta}_m)$ and $Q_{m_2}(\hat{\theta}_m)$ until the maximum tree depth, that we have fixed at the beginning, is reached.

Dealing with regression problems, a typical impurity measure can be, for instance, the Mean Squared Error (MSE)

$$H_{\text{MSE}}(Q_m) = \frac{1}{n_m} \sum_{\mathbf{x}_i \in R_m} (y_i - \bar{y}_m)^2 \quad (4.30)$$

where $\bar{y}_m = \frac{1}{n_m} \sum_{\mathbf{x}_i \in R_m} y_i$, is the average output of Q_m , *i.e.* the training samples in region R_m ; or the Mean Absolute Error (MAE)⁹, which is simply defined as below.

$$H_{\text{MAE}}(Q_m) = \frac{1}{n_m} \sum_{\mathbf{x}_i \in R_m} |y_i - \bar{y}_m| \quad (4.31)$$

On the other hand, dealing with (multiclass) classification problems, where the classes are identified by the index $k = 1, \dots, K$, typical impurity measures are, for instance, the Gini criterion

$$H_{\text{Gini}}(Q_m) = \sum_{k=1}^K p_{mk}(1 - p_{mk}) \quad (4.32)$$

where $p_{mk} = \frac{1}{n_m} \sum_{\mathbf{x}_i \in R_m} \mathbb{1}(y_i = k)$ is the proportion of class k observation on the region R_m ; or the Cross-Entropy (CE)

$$H_{\text{CE}}(Q_m) = - \sum_{k=1}^K p_{mk} \log(p_{mk}) \quad (4.33)$$

⁹we previously came across this measure when evaluating the performance of learning methods on the aging problem, see Sections 4.1.1.1, 4.1.1.2, 4.1.1.3, *etc.*

which is an impurity measure heavily also applied as loss function for other learning machines (see Appendix A).

The main free parameter of this model is the maximum depth of the tree, *i.e.* its size. Deeper trees are capable of modeling complex input-output relationships with respect to shallow ones. Obviously, too deep trees can overfit the training data. Therefore, the optimal depth of the tree should be estimated from the data via, for instance, grid-search cross-validation.

Decision trees are important in biomedical data science applications as they are easily interpretable and capable of modeling nonlinear input-output relationships. Indeed, the prediction is given following a number of binary decisions which, in some cases, may mimic the way doctors perform diagnosis on their patients. Relatively small decision trees can actually be represented graphically. For instance, applying a this method to the aging problem presented in Section 4.1.1.1, we obtain the tree shown in Figure 4.16. As usually, the main free parameter of the model, *i.e.* the maximum depth of the tree, is chosen via grid-search cross-validation (and it is 3, in this case).



Figure 4.16: The graph structure learned from the aging problem (see Section 4.1.1.1). The maximum depth of the tree is 3 and it is chosen via grid-search cross-validation.

A very well-known weakness of decision trees is their instability. In fact, decision trees are not robust to noise affecting the input data. Even minor perturbations of a single feature may generate a different split which propagates down to all the splits below. In practice, this is alleviated by the use of decision trees as base learners of ensemble strategies (see Section 4.1.4). More details on this learning paradigm and its applications can be found reading [Hastie et al., 2009], Chapter 9 or [Bishop, 2006], Chapter 14.

4.1.4 Ensemble methods

The key idea of ensemble methods is to build a prediction model by aggregating a collection of multiple *base learners* that are trained to solve the same problem [Zhou, 2012].

Bagging is a common ensemble strategy that consists in fitting multiple models $f_b(\mathbf{x})$ for $b = 1, \dots, B$ each one on a *bootstrap* sampling¹⁰ \mathcal{D}_b of the training dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$. For each sample \mathbf{x}_i , the bagging estimate $\hat{f}(\mathbf{x}_i)$ is obtained by combining the predictions of the base learners $\hat{f}_b(\mathbf{x}_i)$. For instance, in case of classification tasks, the bagged model may select the most predicted class casting a vote among the B base learners. On the other hand,

¹⁰random sampling with replacement

for regression problems, the bagging estimate can be a (weighted) mean, or the median, of the predictions of the B base learners. Decision trees (see Section 4.1.3) are typical base learner for bagged estimators.

Boosting is another popular ensemble strategy that, unlike bagging, performs predictions by sequentially fitting a collection of base learner that cast a weighted vote [Hastie et al., 2009]. At each boosting step, the weight corresponding to samples that were misclassified at the previous iterations increases. Therefore, each successive classifier is somewhat forced to learn the relationships between input and output that were previously missed. From a theoretical standpoint, it would be possible to boost any learning machine, nevertheless boosting methods are mainly used with decision trees as base learners [Hastie et al., 2009].

4.1.4.1 Random Forests

Random Forests (RF) are ensembles of decision trees, proposed by Leo Breiman in the early 2000s [Breiman, 2001] to ameliorate the instability of decision trees. In RF each tree is grown on a bootstrap sample from the training data, typically to its maximum depth. To increase robustness to noise and diversity among the trees, each node is split using the best split among a subset of features randomly chosen at that node. The number of features on which the trees are grown is one of the free parameters of the model. The final prediction is made by aggregating the prediction of M trees, either by a majority vote in the case of classification problems, or by averaging predictions in the case of regression problems. Another important free parameter of the model is the number of trees in the ensemble. RF are a bagging approach, which works on the assumption that the variance of individual decision trees can be reduced by averaging trees built on many uncorrelated subsamples. Moreover, increasing the number of trees in the ensemble, RF does not overfit the data. Therefore, this parameter is typically chosen to be *as large as possible*, consistently with the available hardware and computational time.

RF can provide several measures of *feature importance*, computed by looking at the increase in prediction error when data for a feature is permuted while all other features remain unchanged. Feature selection based on random forests is most often performed using one of these measures. However, several techniques for applying regularization to random forests have been proposed. These techniques broadly fall under two categories:

1. cost-complexity pruning, which consists in limiting tree depth, resulting in less complex models [Ishwaran et al., 2008; Kulkarni and Sinha, 2012]; and
2. Gini index penalization [Deng and Runger, 2013; Liu et al., 2014].

In addition, [Joly et al., 2012] proposed using an ℓ_1 -norm to reduce the space-complexity of random forests.

RF naturally handle both numerical and categorical variables, multiple scales, and non-linearities. They also require little parameter tuning. This makes them popular for the analysis of diverse types of biological data, such as gene expression, Genome-wide Association Studies (GWAS) data or mass spectrometry [Qi, 2012]. In practice, feature selection schemes that rely on RF may to be unstable [Kursa, 2014], therefore feature selection stability measures must be adopted to avoid drawing inconsistent conclusions.

Let's see what happens when RF are applied to the aging problem (see Section 4.1.1.1). The experimental setup is identical to the one previously applied for ridge and Lasso regression (see

Section 4.1.1.2 and 4.1.1.3), where the only parameter optimized via grid-search cross-validation is the maximum number of features each tree can grow on (and in this case it is chosen as 4). The number of trees in the ensemble is fixed to 500. The RF model on the test set achieves $MAE = 6.321$ and Explained Variance $EV = 88.17\%$. Figure 4.17 shows the feature importance measure estimated by this model. As we can see, the features that are most relevant for RF

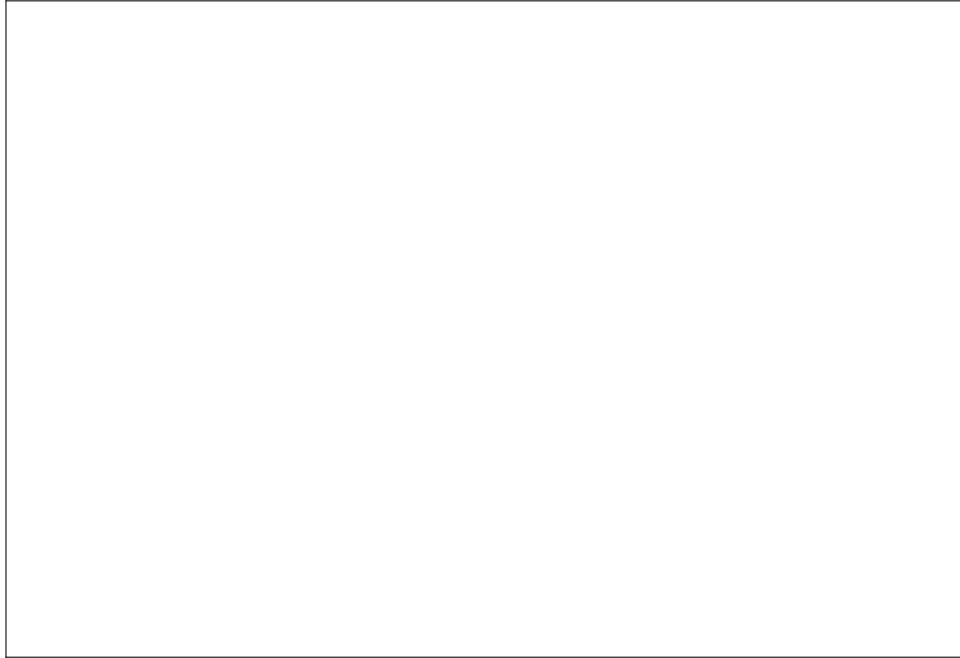


Figure 4.17: A pictorial representation of the feature importance achieved by a RF model with 500 trees growing on 4, out of 12, features.

are comparable to the ones selected by Elastic-Net (see Figure 4.12d). For RF models it is interesting to investigate on the effect of an increasing value for the number of trees in the forest. As shown in Figure 4.18, increasing the number of trees in the forest does not lead to overfit. In fact, test error (Figure 4.18a) and explained variance (Figure 4.18b), after approximately 400 trees, reach a plateau region instead of growing.

4.1.4.2 Gradient Boosting

Gradient boosting [Friedman, 2001] (GB) is one of the most widely applied boosting methods in biological problems. This technique iteratively combines the predictions obtained by several base learners, such as decision trees, into a single model. The key idea behind GB is that, under some general hypothesis on the cost function, boosting can be seen as an iterative gradient method for numerical optimization. In particular, in GB at each boosting iteration a new base learner is fitted on the residuals obtained at the previous boosting iteration. GB has several desirable properties [Mayr et al., 2014], such as its capability to learn nonlinear input-output relationship, its ability to embed a feature importance measure (as RF) and its stability in case of high-dimensional data [Buehlmann, 2006]. When used for feature selection, GB shows interesting performance when casted in the stability selection framework [Meinshausen and Bühlmann, 2010], leading to an effective control of the false discovery rate.

As for most of the learning machine, GB may suffer of overfitting. The main regularization parameter to control is the number of boosting iterations M , *i.e.* the number of base learners,

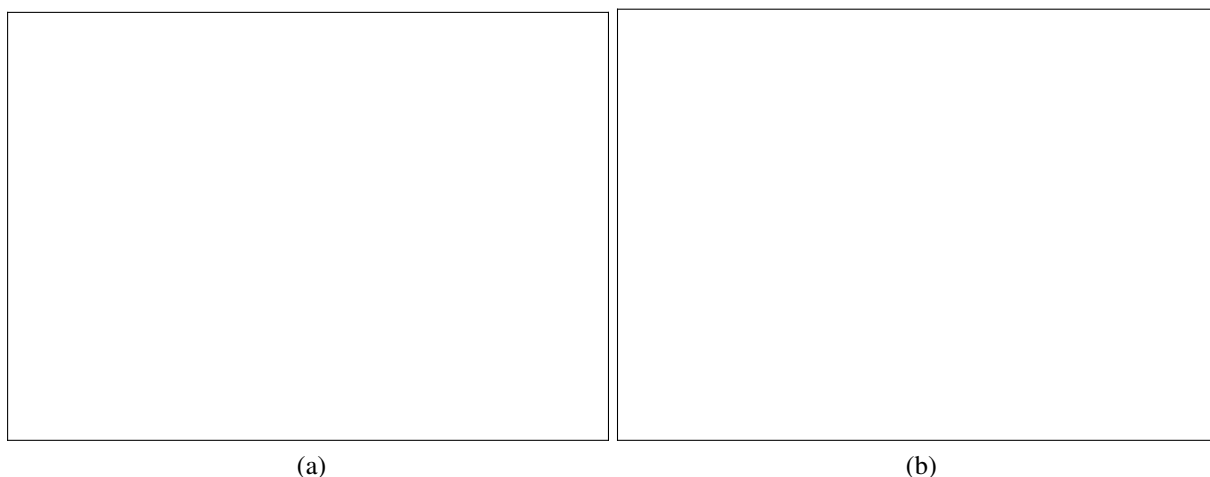


Figure 4.18: The effect of the number of trees in a RF model in terms of MAE, panel (a), and EV, panel (b). As expected, larger forests do not lead to overfit, *i.e.* the training error does not reach zero.

fitted on the training data. This is typically optimized by cross-validated grid-search, or by information criteria-based heuristics [Tutz and Binder, 2006, 2007]. Regularization in GB can also be controlled by shrinking the contribution of each base learner by a constant $0 < \nu < 1$ that controls the learning rate of the boosting procedure. In order to achieve comparable predictive power, smaller values of ν imply larger number of M , so there is a trade-off between them. As usually the base learners are decision trees and another important parameter to tune is their maximum depth [Hastie et al., 2009].

In a recent paper [Lusa et al., 2015], the authors show that in high-dimensional balanced binary classification problems, if the base learner is likely to overfit the training data, the use of *Stochastic GB* [Friedman, 2002] is preferable. The latter is a modified version of the original method, where each base learner is fitted on a random extraction without resubmission of a fraction η of the training data, where η is another regularization parameter to choose.

Let's see what happens when GB is used to tackle the aging problem (see Section 4.1.1.1). The experimental setup is identical to the one previously applied for Elastic-Net regression (see Section 4.1.1.4), where the two parameters optimized via grid-search cross-validation are the maximum depth (in a range from 3 to 20) of the trees and the number of boosting iterations (with a maximum value of 500). The selected tree depth is 8 and the number of boosting iterations is 31. The GB model on the test set achieves $\text{MAE} = 7.955$ and Explained Variance $\text{EV} = 84.88\%$. Figure 4.19 shows the feature importance measure estimated by this model. As we can see, the features that are most relevant for GB are comparable to the ones relevant for RF (see Figure 4.17). As we have already seen for RF, we shall investigate on the effect of increasing boosting iterations for GB. As shown in Figure 4.20, after approximately 50 boosting iterations the model reaches perfect overfit of the training set, *i.e.* zero training error (Figure 4.20a) and 100% trained explained variance (Figure 4.20b).



Figure 4.19: A pictorial representation of the feature importance achieved by a GB model with learning rate 0.1 after 31 boosting iterations. Each tree has maximum depth of 11 and it grows on 4, out of 12, features.

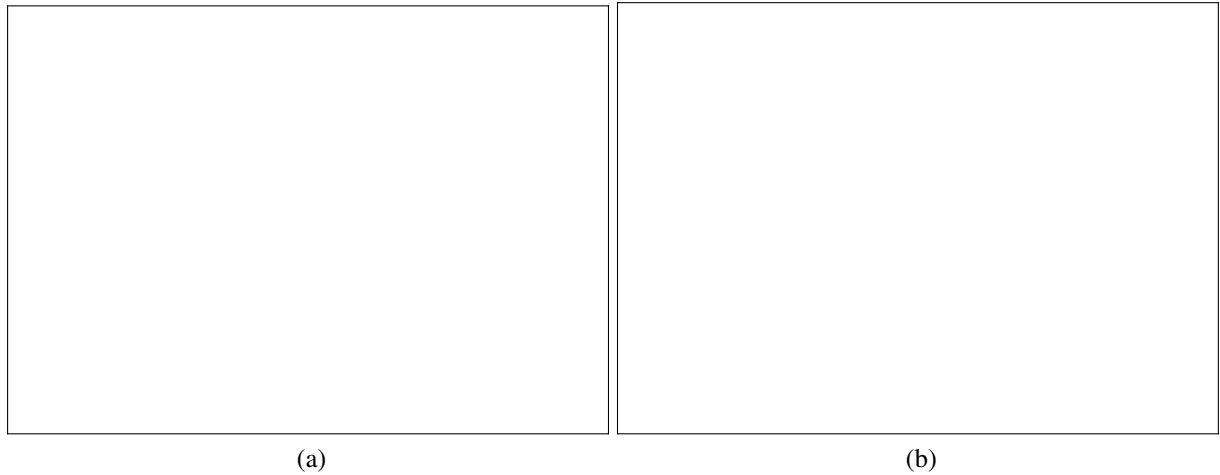


Figure 4.20: The effect of the number of boosting iterations in a GB model (learning rate 0.1) in terms of MAE, panel (a), and EV, panel (b). As expected, increasingly large boosting iterations lead to perfect overfit, *i.e.* zero training error.

4.1.5 Deep learning

Deep Learning (DL) is a branch of ML that, in the recent past, is becoming extremely appealing thanks to the high predictive power that it has empirically shown on real-world problems. This section is definitely not an attempt to summarize the heterogeneous plethora of the DL methods presented in literature so far. Indeed, even a dedicated PhD thesis may not be enough to accomplish this task. In this section we will only get a grasp on the mechanisms behind the

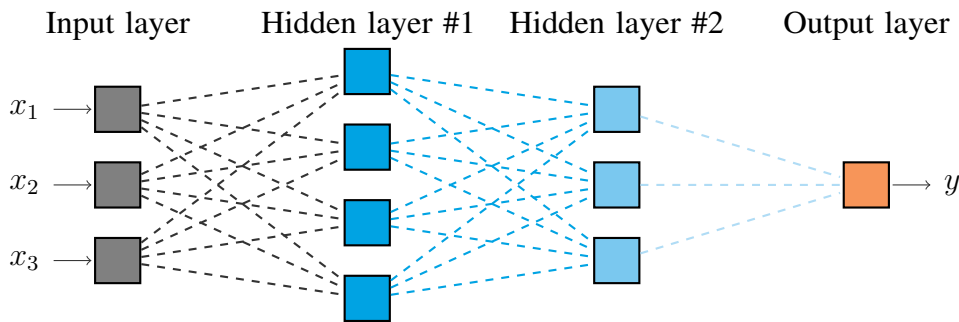
supervised DL methods applied in Part II¹¹.

One of the main characteristics of DL methods is that, starting from raw data, they aim at learning a suitable feature representation (see Section 4.3) and a prediction function, at the same time. DL methods stand on the shoulders of the classical, and shallow, Neural Networks of the 80s, and they can actually be seen as their extension. In DL the final prediction is achieved by composing several layers of non-linear transformations. The intuition behind DL method is that, starting from raw data, their multi-layer architecture can achieve representations at a more abstract level, leading to top performance in prediction tasks. DL architectures can be devised to tackle binary or multiclass classification [Angermueller et al., 2016; Leung et al., 2014] as well as single or multiple output regression [Chen et al., 2016; Ma et al., 2015] tasks.

4.1.5.1 Multi-Layer Perceptron

In order to understand the principles guiding DL methods, we sketch here the ideas behind the most basic one: the Multi-Layer Perceptron (MLP), also known as deep Feedforward Neural Network. Typically, MLPs are structured as fully connected graphs organized in *layers* that can be of three different types: *input*, *hidden* and *output* (see Figure 4.21). Each node of the graph is called *unit*. The number of units in the input layer matches the dimensionality of the raw data (d), while number and type of output units are related to the learning task. For multiclass classification problems, with K classes, the output layer has K units, each one representing the probability of the input sample to be assigned to one specific class. On the same line, for multiple-output regression problems, with K tasks, each one of the K unit corresponds to the prediction for a given task. Size and number of the hidden layers can be arbitrarily chosen according to prediction task and available computational resources.

Figure 4.21: A pictorial representation of the structure of a MLP with two hidden layers having four and three hidden units, respectively. According to the nature of the output layer, this network topology can be adopted either for regression or binary classification problems starting from raw samples in a three-dimensional space.



In MLPs the information flows through the graph from the input to the output. Each layer l transforms its input data \mathbf{x}^{l-1} by composing an affine transformation and an activation function $\sigma(\mathbf{x})$ that acts element-wise on its input vector. In other words, defining as p_{l-i} the number of units in the layer $l - i$, the layer l applies the transformation

$$\mathbf{x}^l = \sigma(\mathbf{x}^{(l-1)}W_l + b_l)$$

¹¹unsupervised DL methods are available as well, but as they are not used in Part II, their discussion is not presented here. We recall to [LeCun et al., 2015; Goodfellow et al., 2016; Chollet, 2018] for a more comprehensive overview.

where $W_l \in \mathbb{R}^{p_{l-1} \times p_l}$ and $\mathbf{b} \in \mathbb{R}^{p_l}$ are the weights of the model that are learned from the data.

The function $\sigma(\mathbf{x})$ is known as *activation function* and it can be defined in different ways. In classical neural networks, activation functions are modeled as sigmoids (e.g. $\sigma(x) = \tanh(x)$, $\sigma(x) = (1 + e^{-x})^{-1}$) whilst in modern DL architectures the most used activation function is the Rectified Linear Unit: $f(x) = \max(0, x)$.

Particular attention must be paid when fitting deep models as they can be prone to overfit the training set [Angermueller et al., 2016]. The network topology itself defines the *degrees of freedom* of the model: deeper and wider networks can approximate well very complicated input-output relationship, but also the noise affecting the data. Although, tuning the number of hidden layers and their size is not the recommended strategy to prevent from overfitting, as it may lead to suboptimal solutions.

Regularization in MLPs can be controlled by penalizing the weights of the network. A common regularization strategy consists in adding an ℓ_2 -norm penalty in the objective function, as in Equation (4.2). In the DL community this procedure is known as weight decay [Krogh and Hertz, 1992]. Although less common, the ℓ_1 -norm can also be adopted as regularization penalty, as in [Leung et al., 2014].

Training MLPs, and deep networks in general, consists in solving a minimization problem via suitable optimization algorithms [Ruder, 2016]. All these methods iteratively update the weights of the network in order to decrease the training error. Another fundamental regularization strategy is the *Early stopping* [Prechelt, 1998]. This consists in interrupting the fitting process as soon as the error on an external validation set increases [Angermueller et al., 2016].

Another common regularization strategy in DL is *Dropout* [Srivastava et al., 2014]. This technique consists in temporarily deactivating a defined number of randomly chosen units of the network at training phase. This reduces the degrees of freedom of the model and it implicitly allows to achieve an ensemble of several smaller networks whose predictions are combined. The use of dropout alone can improve the generalization properties, as in [Chen et al., 2016], but it is usually adopted in combination with weight decay or other forms of regularization, as in [Leung et al., 2014].

Training MLP may imply dealing with non-convex optimization problems which are usually tackled via stochastic gradient descend, or similar methods [Ruder, 2016; Sra et al., 2012].

MLP architectures are also used as final layers of another class of DL methods called: Convolutional Neural Network (CNN). CNN are a different class of DL methods that exploit convolutional masks with weights estimated from the data to learn a suitable feature representation. This class of methods is largely applied to computer vision and image recognition problems. Describing CNN architectures is beyond the scope of this thesis, as they are not mentioned in the Part II. A thorough description of CNN can be found here [Goodfellow et al., 2016].

Let's see what happens when a relatively simple MLP is applied to the aging problem (see Section 4.1.1.1). The experimental setup is the same used for RF in Section 4.1.4.1, where the only parameter optimized via grid-search cross-validation is the weight decay constant. The resulting MLP has 1 hidden layer with 1024 units, ReLu activation and weight decay constant 0.001. Such MLP is trained with early stopping and, after XX iterations reaches $\text{MAE} = 9.14$ explaining the 78.84% of the variance of the test set. The relatively worse performance of this method, compared, for instance, to RF, may be due to the fact that this dataset has a reduced number of samples. It is known that DL methods, in order to reach optimal solutions, need more

training samples than conventional learning machines.

4.1.5.2 Long-Short Term Memory Network

Long-Short Term Memory Network (LSTM) belongs to the class of Recurrent Neural Network (RNN) [Goodfellow et al., 2016]. LSTM use special units, called *memory cells*, which have the ability to learn long-term dependencies [LeCun et al., 2015]. Instead of just applying an element-wise nonlinear function, such as sigmoid or hyperbolic tangent, to the affine transformation of inputs and recurrent units. LSTM cells have an internal self-loop together with a system of *input*, *output* and *forget gates* [Goodfellow et al., 2016]. Compared to standard RNNs, LSTMs can be trained via gradient descent as they do not suffer from the vanishing/exploding gradient problem [Bengio et al., 1994]. In the recent past, this deep learning architectures demonstrated to achieve promising results in several sequences learning tasks, such as speech recognition [Graves and Schmidhuber, 2005], image captioning [Vinyals et al., 2015], or time-series forecast [Schmidhuber et al., 2005].

4.2 Feature selection

4.3 Unsupervised learning

In Section 4.1 we started the discussion of supervised learning methods with a metaphorical comparison with the human ability of learning from examples. Actually, in order to understand some concept, human beings need way less supervision than modern learning algorithms. In some circumstances it may be even the case that little/no supervision at all is needed to accomplish a task. For instance, let's say that a kid growing with a Brittany dog at home visit some friend having a Bernese Mountain Dog. The kid, so far, has only being taught what his dog looks like and these two dog breeds definitely look different one another. Nevertheless, the kid will certainly be able to understand that both of them are animals and, in particular, dogs. Who taught the kid the appearance of every possible dog breed?

The learning paradigm in which we try to infer some interesting properties, or recurrent patterns from unlabeled samples is known as unsupervised learning. Quoting Kevin P. Murphy: "*Unsupervised learning is arguably more typical of human and animal learning.*" [Murphy, 2012].

In this section, given a set of input data $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$, with samples represented as d -dimensional vectors $\mathbf{x}_i \in \mathbb{R}^d$ ($\forall i = 1, \dots, n$), our interest will be directed toward two different goals: (i) grouping similar objects together (Section 4.3.1) and (ii) achieving a meaningful low-dimensional representation of our data (Section 4.3.2). In both cases we will try to infer the properties of the probability distribution $p(\mathbf{x})$ from which the collected data are assumed to be drawn.

4.3.1 Cluster analysis

Cluster analysis, also known as *data segmentation* or *clustering*, is the class of unsupervised learning algorithm which aim at grouping together similar objects, by some definition of

similarity.

This task is fairly common in biomedical studies, where cluster analysis can be used, for instance, to perform patients stratification or to find groups of correlated biological measures, such as gene expression, blood exams, and so on. This section presents an overview on the most popular clustering algorithms.¹²

4.3.1.1 The k -means algorithm

k -means is the most intuitive strategy for cluster analysis.

Given n samples $\mathbf{x}_i \in \mathbb{R}^d$, the goal of k -means partitions the data space into K non-overlapping groups containing similar samples. Let $\boldsymbol{\mu}_k \in \mathbb{R}^d$ be the *prototypical* vector associated with the k^{th} cluster, we can think of this as the center of mass of the cluster, *i.e.* its centroid. Fixed the number of cluster K , the output of the k -means algorithm is a set of cluster centroids as well as the assignment of each sample to a single cluster. For ease of notation, we introduce a binary indicator function r_k defined as

$$r_k(\mathbf{x}_i) = \begin{cases} 1 & \text{if } \mathbf{x}_i \text{ is assigned to cluster } k \\ 0 & \text{otherwise} \end{cases}$$

(this is known as 1-of- K encoding). Therefore, we can define the objective function

$$J(\boldsymbol{\mu}) = \sum_{i=1}^n \sum_{k=1}^K r_k(\mathbf{x}_i) d(\mathbf{x}_i, \boldsymbol{\mu}_k)$$

where $d(a, b)$ is some distance measure. In case $d(a, b)$ is the squared Euclidean distance, which is the typical choice for k -means, the objective function can be rewritten as in Equation (4.34).

$$J(\boldsymbol{\mu}) = \sum_{i=1}^n \sum_{k=1}^K r_k(\mathbf{x}_i) \|\mathbf{x}_i - \boldsymbol{\mu}_k\|_2^2 \quad (4.34)$$

Our goal is to find the centroids $\boldsymbol{\mu}_k$ for $k = 1, \dots, K$ that minimize $J(\boldsymbol{\mu})$. We may observe that, given the best centroids, the samples will be associated to the cluster with the nearest centroid. So, let's focus on the problem of finding $\boldsymbol{\mu}_k$. Fixing $r_k(\mathbf{x})$ for all k , the objective function in Equation (4.34) is quadratic in $\boldsymbol{\mu}$, thus the minimization problem can be easily solved setting its derivative to zero, see Equation (4.35).

$$2 \sum_{i=1}^n r_k(\mathbf{x}_i) (\mathbf{x}_i - \boldsymbol{\mu}_k) = 0 \Rightarrow \boldsymbol{\mu}_k = \frac{\sum_{i=1}^n r_k(\mathbf{x}_i) \mathbf{x}_i}{\sum_{i=1}^n r_k(\mathbf{x}_i)} \quad (4.35)$$

Intuitively, $\boldsymbol{\mu}_k$ corresponds to the mean of the points assigned to the cluster k . The k -means algorithm starts with random centroids. Then, the points are assigned to the cluster having the nearest centroid. Successively, the centroids position is updated. At each step the value of $J(\boldsymbol{\mu})$ is reduced, hence the convergence of k -means is assured. Nevertheless, the algorithm may converge to a local minimum, achieving a suboptimal solution.

¹²a good reference for the clustering algorithms proposed in literature is section 2.3. *Clustering* of the SCIKIT-LEARN documentation <http://scikit-learn.org/stable/modules/clustering.html>.

For a given dataset, defining an optimal value for K is still an open problem. The k -means algorithm works with a number of cluster fixed a priori¹³. This is be given from some prior knowledge on the problem, or it can be estimated following some heuristics. Otherwise, an estimate of K can be defined by optimizing data-driven coefficients like the silhouette score [Rousseeuw, 1987], or some information criteria, such as AIC or BIC [Bishop, 2006]. These estimates can be done with or without cross-validation [Fiorini et al., 2017].

As shown in [Arthur and Vassilvitskii, 2007], smart choices of the initial position of the clusters can substantially improve the result. For instance the initial μ_k , for $k = 1, \dots, K$, can be chosen to be *far away* from each other. This solution is called k -means++ and it follows the steps summarized below.

-
- 1: choose the first centroid μ_1 at random from the samples in the dataset
 - 2: compute the distance between μ_1 and all the other points in the dataset: $D = d(\mathbf{x}_i, \mu_1) \forall i = 1, \dots, n$
 - 3: choose a new data point as centroid μ_2 following a weighted probability distribution $\propto D^2$
 - 4: update the distance vector as $D = \min[d(\mathbf{x}_i, \mu_1), d(\mathbf{x}_i, \mu_2)] \forall i = 1, \dots, n$
 - 5: repeat steps 3 and 4 until k centroids are defined
 - 6: use this centroids initialization to run standard k -means
-

Even though this initial seeding comes with an increased computational time at the beginning, then the k -means algorithm converges super fast actually leading to a decreased overall computational time. Moreover, the achieved solution has considerably lower final cluster reconstruction, compared to standard k -means, hence k -means++ is usually preferable.

4.3.1.2 Spectral clustering

Spectral clustering is a popular clustering algorithm initially proposed in [Shi and Malik, 2000] and then further analyzed and explained in [Ng et al., 2002; Von Luxburg, 2007]. Assuming the usual setup, the problem of clustering can be translated in the problem of inferring a *similarity graph* $\mathcal{G} = (V, E)$, where each data point \mathbf{x}_i corresponds to a vertex and the weight s_{ij} of the edge connecting \mathbf{x}_i and \mathbf{x}_j , is proportional to the similarity between the two points. For instance, in the so-called ε -neighborhood graph, the vertexes corresponding to \mathbf{x}_i and \mathbf{x}_j are connected if $s(\mathbf{x}_i, \mathbf{x}_j) > \varepsilon$, for a given ε . Our hope is then to achieve a graph structure where the edges between points of the same group is high and it is low between points belonging to different groups.

Therefore, let's consider a set of points $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^n$, the steps of the unnormalized spectral algorithm are summarized as follows.

Normalized version this algorithm is identical but the formulation of L which is substituted by the normalized graph Laplacian [Von Luxburg, 2007].

As for the k -means algorithm, finding K is a difficult problem. A common solution for spectral clustering is the *eigengap heuristics*, described in [Von Luxburg, 2007].

¹³some relevant work on automatic cluster discovery is presented in [Ball and Hall, 1967; Pelleg et al., 2000; Muhr and Granitzer, 2009]

-
- 1: fix a symmetric and non-negative similarity function $s(\mathbf{x}_i, \mathbf{x}_j)$ and compute the similarity matrix $S = \begin{bmatrix} 1 & \dots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{1n} & \dots & 1 \end{bmatrix}$
 - 2: construct a similarity graph \mathcal{G} from S , let W be its adjacency matrix and D be its degree matrix
 - 3: choose a new data point as centroid μ_2 following a weighted probability distribution $\propto D^2$
 - 4: compute the unnormalized graph Laplacian $L = D - W$;
 - 5: compute the first k eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_k$ of L and organize them in the matrix $V \in \mathbb{R}^{n \times k}$ column-wise
 - 6: for $i = 1, \dots, n$ let $\psi_i \in \mathbb{R}^k$ be the vector corresponding to the i^{th} row of V
 - 7: cluster the points $\{\psi_i\}_{i=1}^n$ using k -means into K clusters
-

4.3.1.3 Hierarchical clustering

As suggested by the name, hierarchical clustering techniques produces hierarchical (tree) representation of a given dataset, where each level of the hierarchy consist in groups of samples (clusters). Individual data points can be found at the leaves of the learned tree, while all the data points are grouped in a single cluster at the root level.

There are two main families of hierarchical clustering: *agglomerative* (bottom-up) and *divisive* (top-down). Both of them are iterative techniques. The first, at each iteration, groups together the most similar points, hence creating more and more large clusters, while the divisive strategy works the other way around: by iteratively dividing groups of points. From now on we will refer to agglomerative clustering only.

Hierarchical clustering techniques have two main degrees of freedom: the choice for the similarity of points, also known as *affinity*, and groups of points, also known as *linkage*. The first is used to compute the initial value of the similarity matrix

$$S_0 = \begin{bmatrix} 0 & \dots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{1n} & \dots & 0 \end{bmatrix}$$

while the second is used to update $S_0 \in \mathbb{R}_+^{n \times n} \rightarrow S_1 \in \mathbb{R}_+^{n-1 \times n-1}$ once the two most similar data points are collapsed into a single cluster. While any symmetric and non-negative function is a valid affinity, the choices of the linkage function are more interesting and worth mentioning. Let G and H be two clusters, there are four main linkage options:

Single $l(G, H) = \max_{\substack{\mathbf{x}_i \in G \\ \mathbf{x}_j \in H}} s(\mathbf{x}_i, \mathbf{x}_j)$

Complete $l(G, H) = \min_{\substack{\mathbf{x}_i \in G \\ \mathbf{x}_j \in H}} s(\mathbf{x}_i, \mathbf{x}_j)$

Average $l(G, H) = \frac{1}{n_G n_H} \sum_{\mathbf{x}_i \in G} \sum_{\mathbf{x}_j \in H} s(\mathbf{x}_i, \mathbf{x}_j)$

Ward $l(G, H) = \frac{n_G n_H}{n_G + n_H} \|\mu_G - \mu_H\|^2$

where n_A and μ_A are number of points and centroid of the cluster A , respectively.

Interestingly, finding the number of clusters K in this context translates into the problem selecting at which level the tree should stop growing.

4.3.2 Dimensionality reduction and feature learning

As extensively presented in this thesis, most of the ML methods aim at learning some hidden relationship in a set of data. Typically, the dataset we want to analyze consists in a collection of d measures from n samples $x_i \in \mathcal{X} \in \mathbb{R}^d$ and we want our ML algorithms to crunch such data in order to perform some learning task, *e.g.* classification, regression, clustering and so on.

It is often the case that the feature space \mathcal{X} is not the most suitable to our purposes. Sometimes projecting the data in a different space $\mathcal{X}' \in \mathbb{R}^p$, maybe with $p < d$ or even $p \ll d$, can greatly ease the successive learning step. We have already explored this concept talking about the kernel trick in Section 4.1.2.

Heavily decreasing the dimensionality of the data is also helpful as it reduces computational times and eases data visualization. These two aspects are really important when dealing with large scale biomedical data.

This section presents a compact overview of the dimensionality reduction and unsupervised feature learning algorithms that are most relevant with the biomedical data science projects presented in the second part of this thesis.

4.3.2.1 Principal component analysis

4.3.2.2 Multi-dimensional scaling

4.3.2.3 Isomap

4.3.2.4 t-Distributed Stochastic Neighbor Embedding

{maybe}

4.4 Model selection and evaluation

4.4.1 Model selection strategies

4.4.2 Feature selection stability

4.4.3 Performance metrics

Dealing with real-world datasets, having a quantitative and robust way to assess the performance of the applied algorithm is very important. Of course, according to the learning task, different performance metrics should be used. We have already met some of them in the previous section. Here, comprehensive overview of the most representative performance metrics is provided. To

ease readability, the performance metrics are organized according to the correspondent learning task.

Regression The ideal metric for a regression task is a quantitative measure of how distant the predictions $\hat{\mathbf{y}}$ are with respect to the actual output values \mathbf{y} . The following list shows different ways to measure such distance.

- (i) *Mean Absolute Error* (MAE): this is probably the most intuitive way to assess the regression performance. MAE is very well suited for single task regression and time-series forecasting problems. MAE is scale-dependent, hence not suitable for comparisons across different domains.

$$\text{MAE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n |\hat{y}_i - y_i| \quad (4.36)$$

- (ii) *Mean Squared Error* (MSE): widely applied to assess the regression performance, MSE is a second order measure, hence it incorporate bias and variance of the model. MSE of an unbiased estimator corresponds to its variance. MSE is scale-dependent (as MAE).

$$\text{MSE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (4.37)$$

- (iii) *Residual Sum of Squares* (RSS): this data discrepancy measure is mainly used to measure the performance of least squares methods, as it corresponds to the square loss function.

$$\text{RSS}(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (4.38)$$

- (iv) *Coefficient of determination* (R^2): this metric measures the proportion of the variation of the output that is correctly predicted. This metric is not scale dependent, the best possible score is $R^2 = 1.0$, a constant model predicting the average of the output achieves $R^2 = 0.0$. R^2 is defined as below, where \bar{y} is the average output, $\bar{y} = \frac{1}{n} \sum_{j=1}^n y_j$.

$$R^2(\hat{\mathbf{y}}, \mathbf{y}) = 1 - \frac{\sum_{i=1}^n (\hat{y}_i - y_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (4.39)$$

- (v) *Explained Variance* (EV): similar to R^2 , this metric is not scale-dependent and it can be used to compare the outcome of different regression problems. EV is often expressed as percentage, best possible value is 100%, lower is worse. EV is defined as below, where $\text{Var}(\mathbf{y}) = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y})^2$.

$$\text{EV}(\hat{\mathbf{y}}, \mathbf{y}) = 1 - \frac{\text{Var}(\mathbf{y} - \hat{\mathbf{y}})}{\text{Var}(\mathbf{y})} \quad (4.40)$$

Classification Compared to regression, assessing the performance of a classifier seems like a much trivial task. It turns out that this is actually not true; assessing the classification performance by different metrics may let you draw very different conclusions about the quality of $\hat{\mathbf{y}}$. What follows is a list of most important classification metrics, where the meaning of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) is illustrated in the prototypical confusion matrix (for binary classification) represented in Table 4.4.

Table 4.4: A prototypical confusion matrix for a binary classification problem. The two classes are encoded as ± 1 , the number of true positive examples is n_+ , whereas the number of true negative examples is n_- ; hats denote estimated values. The extension for multiclass problems is straightforward, once a positive class is defined.

		Predicted value	
		\hat{n}_{+1}	\hat{n}_{-1}
True value	n_{+1}	True Positive	False Negative
	n_{-1}	False Positive	True Negative

- (i) *Accuracy*: this is the most intuitive classification metric, it simply consists in the fraction of correct predictions. The accuracy measure can be used for binary as well as multiclass classification tasks. The best possible score is 1.0, while the expected score of a random classifier is the fraction of the most represented class.

$$\text{Acc}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}(\hat{y}_i = y_i) \quad (4.41)$$

- (ii) *Balanced accuracy*: for heavily unbalanced binary classification problems this metric is preferable with respect to the standard classification accuracy, as the balanced accuracy of the random classifier is 0.5 by construction.

$$\text{Bacc}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2} \cdot \left[\frac{\text{TP}}{\text{TP} + \text{TN}} + \frac{\text{TN}}{\text{TN} + \text{FP}} \right] \quad (4.42)$$

- (iii) *Precision* (also known as positive predictive value): is the fraction of TP over the total number of samples classified as positive. Precision is defined for binary classification, but it can be extended to multiclass problems selecting the positive class.

$$\text{Prec}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (4.43)$$

- (iv) *Recall* (also known as sensitivity, True Positive Rate (TPR)): is the fraction of TP over the total number of actually positive samples. Recall is defined for binary classification, but it can be extended to multiclass problems selecting the positive class.

$$\text{Rec}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (4.44)$$

- (v) *F₁-score* (also known as F-measure): is the harmonic mean of precision and recall and it can be used to control both of them at the same time.

$$\text{F}_1(\hat{\mathbf{y}}, \mathbf{y}) = \frac{2\text{TP}}{2\text{TP} + \text{FN} + \text{FP}} \quad (4.45)$$

- (vi) *Matthews correlation coefficient* (MCC, also known as φ -coefficient): MCC values ranges from -1 to $+1$, where $MCC = 1$ is the optimal classification, $MCC = 0$ is the score achieved by the random classifier even in cases of unbalanced problems and $MCC = -1$ is the inverse prediction. MCC is defined for binary classification, but it can be extended to multiclass problems selecting the positive class.

$$MCC(\hat{y}, y) = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \quad (4.46)$$

- (vii) *Area Under Receiver Operating Curve* (AUC): the Receiver Operating characteristic Curve (ROC) is a plot used to assess the performance of a binary classifier as its classification threshold is varied. In ROC analysis, the horizontal axis is the False Positive Rate (FPR, also known as fall-out), defined as,

$$FPR(\hat{y}, y) = \frac{FP}{FP + TN} \quad (4.47)$$

while the vertical axis is the recall, defined in Equation (4.44). An example of ROC analysis is reported in Figure 4.22. The ROC of a random classifier is simply the bottom-left to top-right diagonal, plots in the upper triangular area indicate *good* classifiers, and vice-versa. A fundamental metric for classification problems is the AUC. AUC ranges from 0 to 1, where optimal classification leads to $AUC = 1$ and the random classifier achieves $AUC = 0.5$.



Figure 4.22: An example of ROC analysis on the MNIST dataset [LeCun et al., 2010]. In this example a linear SVM is trained on only 1% of the dataset, where 200 additional noisy features were added to make the classification problem harder.

Clustering Quantitatively measuring the output of a clustering algorithm is not a trivial task, in particular when no ground truth on the clusters is provided. We report here a list of the most common clustering performance metrics.

- (i) *Silhouette*: the main characteristic of this metric is that it can be used when no ground truth on the clustering is provided [Rousseeuw, 1987]. The silhouette coefficient ranges from -1 to 1 and it is higher for compact and well-separated clusters. Silhouette score around 0 are associated to partially overlapping clusters. Let $a(\mathbf{x}_i)$ be the average dissimilarity of \mathbf{x}_i with all the other points in the same cluster; let also be $b(\mathbf{x}_i)$ be the average distance of \mathbf{x}_i to all the points in the nearest cluster. We can interpret $a(\mathbf{x}_i)$ as a measure of point-to-cluster assignment, where the higher $a(\mathbf{x}_i)$ is, the more the point \mathbf{x}_i is correctly assigned. On the other hand, $b(\mathbf{x}_i)$ can be seen as a clusters separability index. The silhouette coefficient is defined as follows.

$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max[a(\mathbf{x}_i), b(\mathbf{x}_i)]} \quad (4.48)$$

The quality of the achieved clustering can be graphically assessed visualizing the silhouette coefficient for each sample, as in Figure 4.23¹⁴, or it can be globally measured by its average score $\bar{s} = \frac{1}{n} \sum_{i=1}^n s(\mathbf{x}_i)$.

- (ii) *Homogeneity*: a clustering satisfies the homogeneity property if each cluster contains only members of a single class. In order to evaluate the homogeneity, the clustering ground truth must be provided. In this sense, this can be seen as a supervised measure.
- (iii) *Completeness*: a clustering satisfies the completeness property if all members of a given class are assigned to the same cluster. As for the homogeneity, this is a supervised measure.
- (iv) *V-measure*: this is simply defined as the harmonic mean of completeness and homogeneity.
- (v) *Adjusted Rand Index (ARI)*: this index is a chance normalized measure of the similarity between two cluster assignments which is robust with respect to permutations of the cluster identifier. This index ranges from -1 and 1 , where -1 stands for independent cluster assignments and 1 is perfect match. As it requires the clusters ground truth, this is a supervised metric. Let \mathbf{y} and $\hat{\mathbf{y}}$ be a ground truth and an estimated cluster assignment, respectively, ARI is defined as follows.

$$\text{ARI}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{\text{RI} - E[\text{RI}]}{\max(\text{RI}) - E[\text{RI}]} \quad (4.49)$$

where RI (the unadjusted Rand Index) is given by

$$\text{RI} = \frac{g + h}{\psi}$$

where g is the number of pairs of elements that are in the same set both in \mathbf{y} and $\hat{\mathbf{y}}$; h is the number of pairs of elements that are in different sets in both \mathbf{y} and $\hat{\mathbf{y}}$ and ψ is the total number of pairs (without ordering).

- (vi) *Adjusted Mutual Information (AMI)*: similar to ARI, this is an information theory-based metric to measure the similarity between two cluster assignments. Let $Y = \{\mathbf{y}_1, \dots, \mathbf{y}_R\}$ and $\hat{Y} = \{\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_S\}$ be a ground truth and an estimated clustering partitions, respectively, AMI is given by

$$\text{AMI}(\hat{Y}, Y) = \frac{\text{MI} - E[\text{MI}]}{\max[H(\hat{Y}), H(Y)] - E[\text{MI}]} \quad (4.50)$$

¹⁴image created with ADENINE, see Chapter 5.

where H is the entropy and MI is the Mutual Information, defined as

$$\text{MI}(\hat{Y}, Y) = \sum_{i=1}^R \sum_{j=1}^S P(i, j) \log \left[\frac{P(i, j)}{P(i)P'(j)} \right]$$

where $P(i)$ is the probability that a randomly picked sample falls in \mathbf{y}_i , likewise for $\hat{\mathbf{y}}$ and $P'(j)$. While, $P(i, j)$ is the probability that a random sample falls in both clusters $\mathbf{y}_i, \hat{\mathbf{y}}_j$.

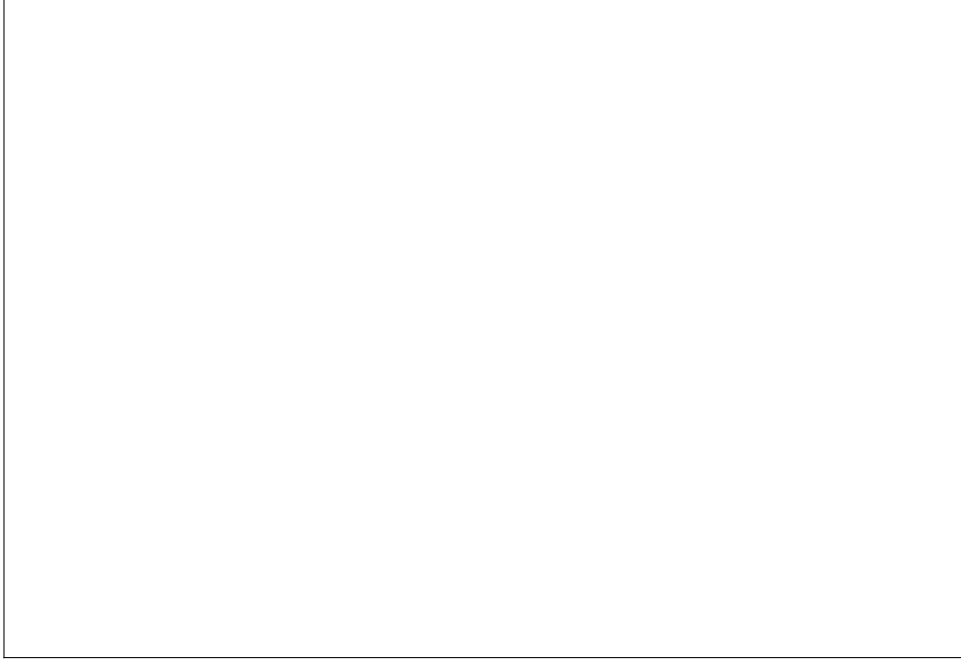


Figure 4.23: An example of Silhouette analysis where a 10 clusters k -means runs on a randomly sampled 80% of the MNIST dataset [LeCun et al., 2010].

4.5 Computational requirements and implementations

- MPI
- GPU and accelerators

Part II

5 ADENINE: a HPC-oriented tool for biological data exploration

This chapter presents the first original contribution of the thesis, which is the development of a machine learning framework designed for biological data exploration and visualization called ADENINE. The goal of this framework is to help biomedical data scientists achieving a first and quick overview of the main structures underlying their data. This software tool encompasses state of the art techniques for missing values imputing, data preprocessing, dimensionality reduction and clustering. ADENINE has a scalable architecture which seamlessly work on a single workstation as well as on a high-performance computing facility. ADENINE is capable of generating publication-ready plots along with quantitative descriptions of the results. At the end of the chapter, few examples of exploratory analysis with ADENINE on biological datasets are presented.

5.1 Exploratory data analysis

In biology, as well as in any other scientific domain, exploring and visualizing the collected measures is an insightful starting point for every data analysis process. For instance, the aim of a biomedical study can be detecting groups of patients that respond differently to a given treatment, or inferring possible molecular relationships among all, or a subset, of the measured variables. In both cases, data scientists will be asked to extract meaningful information from collections of complex and possibly high-dimensional measures, such as gene sequencing data, biomedical imaging, patients assessment questionnaires, and so on.

In these cases, a preliminary Exploratory Data Analysis (EDA) is not only a good practice, but also a fundamental step to run before further and deeper investigations can take place. Representative examples are, for instance, Kaggle competitions (source: www.kaggle.com). Kaggle is the most popular community-based data science challenge platform online. Browsing through the hosted competitions, we can see that people runs EDAs to get a grasp on the datasets before testing their own prediction strategy.

Running EDA, despite being a valuable and widely applied data science practice, is definitely a nontrivial task. Indeed, for a given a dataset, it is usually unclear which algorithm will provide the most insightful results. A common practice is to empirically test some of them and compare their results from a qualitative/quantitative standpoint. Moreover, applying EDA methods on large-scale data can be burdensome or even computationally unfeasible.

In the last few years, a fair number of data exploration software and libraries were released. Such tools may be grouped in two families: GUI-based and command-line applications.

Among the first group we recall Divvy [Lewis et al., 2013], a software tool that performs dimensionality reduction and clustering on input data sets. *Divvy* is a light framework; however, its collection of C/C++ algorithm implementations does not cover common strategies such as

kernel principal component analysis [Schölkopf et al., 1997] or hierarchical clustering [Hastie et al., 2009], it does not offer strategies to perform automatic discovery of the number of clusters and it is only available for macOS. One of the most popular GUI-based data analysis software is KNIME (source: www.knime.com). KNIME has a rather user-friendly interface where creating EDA pipelines is as simple as drawing small trees, where each node is a box representing some algorithm. KNIME is also suitable for large datasets as it can distribute the computation on Oracle Grid Engine clusters. The main drawback of such GUI-based softwares is that they do not allow to automatically generate EDA pipelines. Moreover, plugging in hand-crafted algorithm is not easy. One possibility is to use KNIME Meta Nodes, but programming relatively complex algorithms just using them can be cumbersome.

Nextflow (source: www.nextflow.io) is a command-line workflow framework that allows to easily distribute the computations of complex pipelines. One of the strength of *Nextflow* pipelines is that they are made by chaining together differently many processes that can even be written in differently many languages. *Nextflow* is not just a simple computational framework, it is also a proper Domain Specific Language (DSL) which extends Groovy. Although very powerful, *Nextflow* is more tailored toward the bioinformatics and computational biology area than a more general data science audience. Using *Nextflow*, everything must be hard-coded, therefore plugging in custom algorithms is as hard as using standard libraries. Similar considerations can be made for Snakemake (source <https://snakemake.bitbucket.io>), which is a pythonic workflow management system which is mainly tailored for the parallelization of the analysis of DNA data.

The most notable project that spans between the two families is *Orange*¹ [Demšar et al., 2013], a data mining software suite that offers both visual programming front-end and Python APIs. In the context of data exploration, *Orange* can be successfully employed. However, in order to test different data analysis pipelines, each one must be manually created as it does not support their automatic generation. Moreover, large data sets are difficult to analyze as it can run only on a single workstation, lacking of distributed computing support.

5.2 ADENINE overview

This chapter introduces ADENINE (source : <http://slipguru.github.io/adenine>), a command-line Python tool for data exploration and visualization that, combining different EDA methods, creates textual and graphical analytical reports of large scale, data collections.

Missing data imputing, preprocessing, dimensionality reduction and clustering strategies are considered as building blocks for data analysis pipelines. The user is simply required to specify the input data and to select the desired blocks. ADENINE, then, takes care of generating and running the pipelines obtained by all possible combinations of the selected blocks. Table 5.1 shows the list of building blocks currently available. Every algorithm implementation in ADENINE is inherited, or extended, from SCIKIT-LEARN [Pedregosa et al., 2011] which, as of today, is definitely the most complete ML open source Python library available. Moreover, virtually any custom algorithm can be easily plugged in an ADENINE pipeline, as long as it is implemented following the SCIKIT-LEARN standard.

ADENINE is developed with the aim of speeding-up EDA on large biomedical data collections.

¹if I have to pick one, this is definitely my favorite – Ed.

Step	Algorithm	Reference
Imputing	mean	[Troyanskaya et al., 2001]
	median	
	most frequent	
	k -nearest neighbors	
Preprocessing	recentering	
	standardize	
	normalize	
	min-max	
Dimensionality reduction	principal component analysis (PCA)	[Jolliffe, 2002]
	incremental PCA	[Ross et al., 2008]
	randomized PCA	[Halko et al., 2011]
	kernel PCA	[Schölkopf et al., 1997]
	isomap	[Tenenbaum et al., 2000]
	locally linear embedding	[Roweis and Saul, 2000]
	spectral embedding	[Ng et al., 2002]
	multi-dimensional scaling	[Borg and Groenen, 2005]
	t-distributed stochastic neighbor embedding	[Van der Maaten and Hinton, 2008]
Clustering	k -means	[Bishop, 2006]
	affinity propagation	[Frey and Dueck, 2007]
	mean shift	[Comaniciu and Meer, 2002]
	spectral	[Shi and Malik, 2000]
	hierarchical	[Hastie et al., 2009]
	DBSCAN	[Ester et al., 1996]

Table 5.1: Pipeline building blocks currently available in ADENINE.

It has a scalable architecture, that allows its pipelines to seamlessly run in parallel as separate Python processes on a single workstation or as separate tasks in a High-Performance Computing (HPC) cluster facility. This remarkable feature allows to explore and visualize massive amounts of data in a reasonable computational time. Moreover, as ADENINE makes large use of NUMPY and SCIPY, it automatically benefits from their bindings with optimized linear algebra libraries (such as OpenBLAS or Intel® MKL). Thanks to this hybrid parallel architecture, with ADENINE it is possible to get a grasp on the main structures underlying your data in a reasonable computational time.

In ADENINE, any tabular data format (.csv, .tsv, .json, and so on) can be given as input. Moreover, as genomic is one of the major sources of data in the biological world, ADENINE also natively supports data integration with the NCBI Gene Expression Omnibus (GEO) archive [Barrett et al., 2013]; which data sets can be retrieved specifying their GEO accession number.

5.3 Software description

ADENINE is developed around the data analysis concept of *pipeline*. A pipeline is a sequence of the following fundamental steps:

1. missing values imputing;

2. data preprocessing;
3. dimensionality reduction and
4. unsupervised clustering.

For each task, is it possible to use either the off-the-shelf algorithms in Table 5.1, either any other custom algorithm as long as it is implemented in Python as a SCIKIT-LEARN `Transformer` object. A `Transformer` is simply a Python class having a `fit_transform` method, see for instance snippet below.

```
class DummyTransformer(object):
    """A dummy transformer class."""

    def fit(self, X, y):
        """Fit DummyTransformer."""
        return self

    def transform(self, X):
        """Transform X."""
        return X

    def fit_transform(self, X, y=None):
        """Fit to data, then transform it."""
        return self.fit(X, y).transform(X)
```

SCIKIT-LEARN also provides an utility object called `FunctionTransformer` that returns a `Transformer` given any arbitrary input callable (*i.e.* function), which can be very resourceful for our purposes.

Data collected in biomedical research studies often present missing values. Devising imputing strategies is a common practice to deal with such issue [De Souto et al., 2015]. ADENINE offers an improved version of the `Imputer` class provided by SCIKIT-LEARN. In addition to the pre-existent feature-wise `"mean"`, `"median"` and `"most_frequent"` strategies, this extension adds `"nearest_neighbors"`, *i.e.* an implementation of the k -nearest neighbors imputing method proposed for microarray data in [Troyanskaya et al., 2001].

Collecting data from heterogeneous sources may imply dealing with variables lying in very different numerical ranges and this could have a negative influence on the behavior of data analysis techniques. To tackle this issue ADENINE offers different strategies to preprocess data, such as re-centering, standardizing or rescaling. We recall here that any other preprocessing step, such as feature exclusion, log-transformation and so on, can be implemented as `Transformer` to be easily chained in an ADENINE pipeline.

ADENINE includes a set of linear and nonlinear dimensionality reduction and manifold learning algorithms that are particularly suited for projection and visualization of high-dimensional data. Such techniques rely on the fact that it is often possible to *decrease* the dimensionality of the problem estimating a low-dimensional embedding in which the data lie, as described in Section 4.3.2.

Besides offering a wide range of clustering techniques, ADENINE implements strategies and heuristics to automatically estimate parameters that yield the most suitable cluster separation. The selection of the optimal number of clusters in centroid-based algorithms follows the B -fold cross-validation strategy presented in Algorithm 1, where $\mathcal{S}(X, \hat{y})$ is the mean silhouette coefficient of the input samples, see Equation (4.48).

Algorithm 1 Automatic discovery of the optimal clustering parameter.

```

1: for clustering parameter  $k$  in  $k_1 \dots k_K$  do
2:   for cross-validation split  $b$  in  $1 \dots B$  do
3:      $X_b^{tr}, X_b^{vld} \leftarrow b$ -th training, validation set
4:      $\hat{m} \leftarrow$  fit model on  $X_b^{tr}$ 
5:      $\hat{y} \leftarrow$  predict labels of  $X_b^{vld}$  according to  $\hat{m}$ 
6:      $s_b \leftarrow$  evaluate silhouette score  $\mathcal{S}(X_b^{vld}, \hat{y})$ 
7:   end for
8:    $\bar{S}_k = \frac{1}{B} \sum_{i=1}^B s_i$ 
9: end for
10:  $k_{opt} = \operatorname{argmax}_k k(\bar{S}_k)$ 

```

For affinity propagation [Frey and Dueck, 2007] and k -means [Bishop, 2006] clustering parameters can be automatically defined ("**preference**" and "**n_clusters**", respectively). Mean shift [Comaniciu and Meer, 2002] and DBSCAN [Ester et al., 1996] offer an implicit cluster discovery. For hierarchical [Hastie et al., 2009] and spectral clustering [Shi and Malik, 2000], no automatic discovery of clustering parameters is offered. However, graphical aids are generated to evaluate clustering performance such as dendrogram tree and a plot of the eigenvalues of the Laplacian of the affinity matrix², respectively.

In order to speed-up EDA on large-scale datasets, ADENINE exploits two main parallel computing paradigms: multiprocessing and multithreading. The latter is provided by the use of optimized linear algebra libraries of NUMPY and SCIPY and it comes *for free*. So, let's focus on multiprocessing. This parallel program paradigm consists in dividing the workload among different tasks which can possibly run on different nodes of an HPC infrastructure. The input of each ADENINE pipeline is the raw dataset to which successive transformations and algorithms are applied. Therefore, each pipeline is independent and this makes ADENINE suitable for the application of distributed computing schemes. ADENINE offers different job distribution strategies.

1. When ADENINE is running on a single workstation (or on a single node of an HPC architecture), each pipeline is parallelized by using the standard multiprocessing Python library. Each pipeline run is mapped to a job. All the jobs are organized in a queue and the multiprocessing library automatically assigns the cores available to the next job. In this way, ADENINE is capable of exploiting all the cores of the machine in which it is running.
2. When an HPC architecture is available, ADENINE is capable of distributing its pipelines across multiple nodes. In particular, the following two options are available.

MPI ADENINE implements an MPI-based (source: <http://mpi-forum.org/>) master/slave job distribution strategy. A single master task T_0 creates a queue of jobs $Q = \{J_1, \dots, J_M\}$, where M is the number of pipelines. The jobs in Q are assigned

²see the *eigengap heuristics* [Von Luxburg, 2007].

to a pool of slaves $S = \{T_1, \dots, T_S\}$. Once the slave T_j (with $j > 0$) has completed its task, it communicates the results to T_0 and becomes idle. At this point, T_0 assigns to T_j the next task in Q . This procedure is repeated several times, until Q is empty. The number of slaves S can be controlled by the user with the parameter `-np` of the `mpirun` command. S must be carefully chosen according to the size of the dataset, the number of pipelines generated and the available hardware. Thanks to MPI, the assignment of each task to its node is completely transparent to the user. When all the computation is done, the results are simply stored in a single result folder.

Dask ADENINE can also exploit the Dask parallel computing library (source: <https://dask.pydata.org>). While MPI is a recognized standard in the HPC world, Dask is relatively recent project developed for dynamic task scheduling focused on big data applications. In particular, ADENINE exploits the Dask-based backend of Joblib (source: <https://pythonhosted.org/joblib/>), which is the parallel computing library mainly used in SCIKIT-LEARN to parallelize independent tasks. Dask can be used by simply running a `dask-scheduler` executable process on one node and the `dask-worker` executable on several processes, which can run on other nodes of the HPC architecture. After an initial hand-shake between the scheduler and its workers, a list of jobs to run can be sent to the scheduler which automatically distributes the workload among its workers and collects the results. Using Dask in ML applications, where communications between tasks is usually negligible, can be easier with respect to MPI.

5.4 Usage example

In this section we show how ADENINE can be used to perform two EDAs on a gene expression microarray data set obtained from the GEO repository (accession number GSE87211). This data set was collected in a recent medical study that aimed at understanding the underlying mechanism of colorectal cancer (CRC) as well as identifying molecular biomarkers, fundamental for the disease prognostication. It is composed of 203 colorectal cancer samples and 160 matched mucosa controls. The adopted platform was the Agilent-026652 Whole Human Genome Microarray, which was used to measure the expression of 34127 probe sets.

ADENINE offers a handy tool to automatically download the data set from the GEO repository given only its accession name. It also let the user select phenotypes and/or probe sets of interest. Given these preferences, ADENINE automatically converts the data set from the *SOFT* format to a comma-separated values text file. To download the remote GEO data set specifying the tissue type as phenotype of interest we used the following command.

```
$ ade_GEO2csv.py GSE87211 --label_field characteristics_ch1.3.tissue
```

This automatically creates `GSE87211_data.csv` and `GSE87211_labels.csv` which contain gene expression levels and tissue type of each sample, respectively.

The first EDA aims at stratifying the samples according to their tissue type (mucosa or rectal tumor) this can be performed by executing the following command.

```
$ ade_run.py config.py
```

Where `config.py` is a configuration file which should look like the snippet below.

```

from adenine.utils import data_source
data_file = 'GSE87211_data.csv'
labels_file = 'GSE87211_labels.csv'
X, y, feat_names, index = data_source.load(
    'custom', data_file, labels_file, samples_on='rows', sep=',')
step1 = {'Recenter': [True]}
step2 = {'KernelPCA': [True, {'kernel': ['linear', 'rbf', 'poly']}],
        'Isomap': [True, {'n_neighbors': 5}]}
step3 = {'KMeans': [True, {'n_clusters': ['auto', 2]}]}

```

Each step variable refers to a dictionary having the name of the building block as key and a list as value. Each list has a *on/off* trigger in first position followed by a dictionary of keyword arguments for the class implementing the corresponding method. When more than one method is specified in a single step, or more than a single parameter is passed as list, ADENINE generates the pipelines composed of all possible combinations.

The configuration snippet above generates eight pipelines with similar structure. The first and the second halves have recentered and ℓ_2 -normalized samples, respectively. Each sample is then projected on a 2D space by isomap or by linear, Gaussian or polynomial kernel PCA. k -means clustering with automatic cluster discovery is eventually performed on each dimensionality-reduced data set, as in Algorithm 1. Results of such pipelines are all stored in a single output folder. Once this process is completed, plots and reports can be automatically generated running the following command.

```
$ ade_analysis.py results/ade_output_folder_YYYY-MM-DD_hh:mm:ss
```

The aim of the second EDA is to uncover the relationships among a set of genes known from the literature to be strongly associated with CRC. Specifically this signature is composed of the following genes: APC, KRAS, CTNNB1, TP53, MSH2, MLH1, PMS2, PTEN, SMAD4, STK11, GSK3B and AXIN2 [Schulz, 2005]. We also considered probe sets measuring expression level of the same gene, and we labelled them with a progressive number. Three partially overlapping sublists compose this signature.

- S1) Genes fundamental for the progression of CRC (*i.e.* APC, KRAS, CTNNB1, TP53).
- S2) Genes relevant in the *Wnt signaling pathway*, which is strongly activated in the first phases of CRC (*i.e.* APC, CTNNB1, GSK3B, AXIN2).
- S3) Genes involved in hereditary syndromes which predispose to CRC (*i.e.* APC, MSH2, MLH1, PMS2, PTEN, SMAD4, STK11) [Schulz, 2005].

A reduced version of the GEO data set that comprises only such genes can be easily created calling `ade_GEO2csv.py` with the option `--signature GENE_1, GENE_2, ..., GENE_N`. On the same line, the option `--phenotypes P_1, P_2, ..., P_M` can be used to keep only mucosa or rectal tumor samples. To run such experiment, one simply needs to select and activate the hierarchical clustering building block and to follow the same steps presented above.

5.5 Results

In the first EDA, we compared the clustering performance achieved by the eight ADENINE pipelines and we reported in Figure 5.1 an intuitive visualization of the results achieved by the top three, evaluated in terms of silhouette score [Rousseeuw, 1987]. As expected, the top performing pipelines show a clear separation between the two sample groups, as the k -means algorithm devises a domain partitioning that is consistent with the actual tissue types.

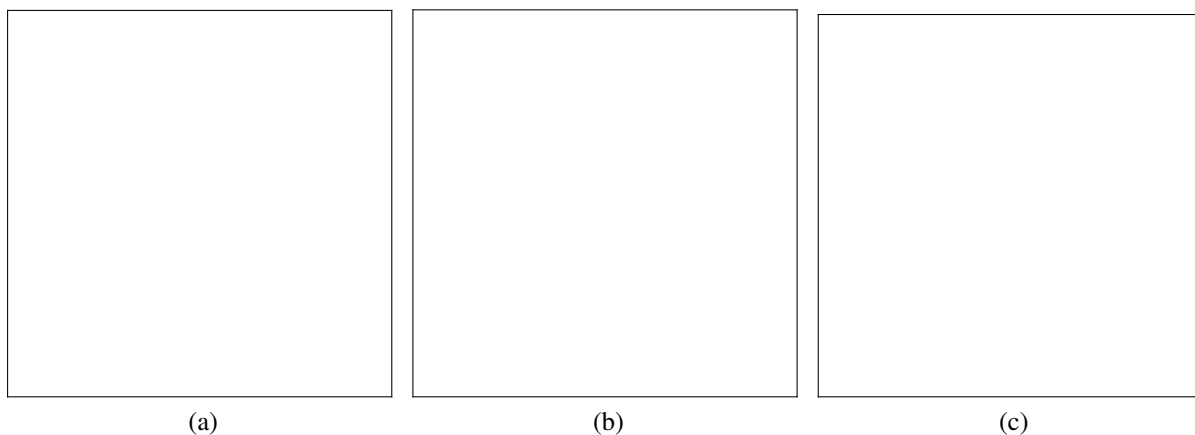


Figure 5.1: Three different 2D projections of the samples of the GEO gene expression data set used in this work. Projections on the left (a), middle (b) and right (c) panes are obtained via linear PCA, Gaussian PCA and isomap, respectively. The color of each point corresponds to the actual tissue type, while the background color is automatically learned by the k -means clustering algorithm. White hexagons correspond to cluster centroids.

For the second EDA, the relationships among the probe sets corresponding to the genes of the signature are separately explored learning a different hierarchical clustering [Hastie et al., 2009] tree for mucosa (Figures 5.2a) and CRC samples (Figure 5.2b), separately. The two trees are learned from different tissues, nevertheless they show some remarkable similarities. For instance, the pairs TP53-TP53.1 and MSH2-PMS2.1 always share a common parent. Interestingly, the first is a relationship between probe sets of the same gene, and the second is confirmed in literature, as MSH2 and PMS2 are both involved in hereditary non-polyposis CRC, a syndrome that predisposes for CRC. Moreover, two probe sets of the the genes of *S1*, namely APC and CTNNB1, are consistently close to the root of the two trees. This suggest that the expression level of these two genes highly differs from the others. Two interesting differences between the two trees can also be noticed. First, most of the elements of the sublist *S3*, which contains genes that enhance the risk of developing CRC, tend to be grouped together in Figure 5.2b, while the same observation cannot be done for Figure 5.2a. Secondly, probe sets of the genes belonging to sublists *S2* and *S3* tend more to more closely connected in Figure 5.2b than in Figure 5.2a.

{add more usage examples?!}



Figure 5.2: An example of hierarchical trees visualization learned by two ADENINE pipelines on mucosa (a) and CRC (b) samples. Each probe set is color coded according to the corresponding sublist. This visualization provides insights on the underlying structure of the measured gene expression level.

6 Model for metabolic age prediction

abstract here.

6.1 Aging and metabolism

7 Temporal model for multiple sclerosis evolution

abstract here.

Multiple Sclerosis (MS) is a neurodegenerative and chronic disease of the central nervous system characterized by damages to the myelin sheaths, resulting in a wide range of symptoms, such as fatigue, numbness, visual disturbances, bladder problems, mobility issues and cognitive deficits.

People with MS (PwMS) are mainly classified according to their disease course: relapsing-remitting (*RR*), secondary-progressive (*SP*), primary-progressive (*PP*) and progressive-relapsing (*PR*) [Giovannoni et al., 2016]. Neurological disability in *RR* patients is mainly due to the development of multifocal inflammatory lesions and it results in relapses, that are attacks of neurological worsening, followed by partial or complete recovery. Disability accrues predominantly in progressive courses (*SP*, *PP*, *PR*) that are more characterized from diffuse immune mechanisms and neurodegeneration. An estimated 15% of PwMS have a *PP* or *PR* course at the onset, the remaining 85% is diagnosed with a *RR* course. About 80% of *RR* patients develop *SP* course within 15–20 years if untreated, or if the adopted pharmacological and rehabilitative protocols are not continuously adjusted according to the evolution of the disease [Scalfari et al., 2014].

For this reason, the prediction of the transition from *RR* to *SP* is one of the most important methodological gaps that MS researchers are currently addressing. The availability of a statistical model able to predict disease worsening is one of the major unmet needs that could significantly improve timeliness, personalization and, consequently, the efficacy of the treatments. Nowadays, there are no clear clinical, imaging, immunologic or pathologic criteria to foresee the transition from *RR* to *SP* [Lublin et al., 2014]. Several clinical factors relating to possible *SP* course predictors have been identified [Bergamaschi et al., 2015; Dickens et al., 2014]. However, as showed by [Vukusic and Confavreux, 2003], studies investigating on prognostic factors for MS course evolution generally suffer from two shortcomings: they report a high proportion of *RR* patients not monitored enough to reach progressive course and they lead, to some extent, to contradictory results. Currently, MS research mainly focuses on developing and assessing drugs and rehabilitative protocols for *RR* patients disregarding progressive courses.

In the recent past, researchers explored the potential role of Patient-Centered Outcomes (*PCO*) to follow the progression of neurodegenerative diseases and to take timely healthcare decisions [Black, 2013]. *PCOs* comprise self- and physician-administered tests, questionnaires and clinical scales consisting of either ordinal or categorical scaled answers. As opposed to stressful, not frequently repeatable and expensive clinical exams, like magnetic resonance imaging or blood tests, *PCOs* are patient-friendly and low-cost measures that could allow to investigate the individual changes and disease impact on several aspects such as physical, cognitive, psychological, social and well-being domains [Fiorini et al., 2015]. To date, *PCOs* are extensively used to assess general health status, to support diagnosis and monitor progress of disease and to quantify the patients' perception of the effectiveness of a given therapy or procedure [Nelson et al., 2015]. Nevertheless, it is still unclear which are the most informative *PCOs* and, contextually, whether they can be used as *predictors* for disease evolution.

In our study, we propose a machine learning approach that, leveraging on *PCO* data, aims at predicting the temporal evolution of MS disease course providing insights on the most appropriate use of PCOs. We resort to a vast category of predictive models, ranging from sparse regularization to ensemble and deep learning methods. These models are widely adopted in the biomedical context as they benefit from good generalization properties as well as they allow to address regression and classification problems within the same statistical and computational framework [LeCun et al., 2015; Qi, 2012; Nowak et al., 2011; Teramoto et al., 2009; Zou and Hastie, 2005].

7.1 PCOs data set description

The predictive model presented in this work is based on a *PCO* data set acquired from a cohort of PwMS progressively enrolled within an ongoing funded project. Ethical review committee approval *023REG2014* was obtained for this work.

Each patient is evaluated every four months through the items of the PCOs reported in Table 7.1 which cover physical, cognitive and psychosocial domains. *PCO* data are intrinsically noisy due to the subjectivity of self-reported measures provided by the patients that can be influenced by personal feelings and opinions. In order to ameliorate this issue, 4 questionnaires out of 10 are administered by medical staff which is trained to keep a homogeneous level of evaluation.

In our analysis we considered all the PCOs reported in Table 7.1 except EDSS. Such scale is based on a neurological examination and, although usually adopted as an index of the disability level, it focuses mainly on deambulation disability without taking into account other aspects that could impact patient disability, such as upper limb or cognitive functions [Meyer-Moock et al., 2014; Uitdehaag, 2014].

Acronym	Full name	Reference
<i>MFIS</i>	Modified fatigue impact scale	[Flachenecker et al., 2002]
<i>HADS</i>	Hospital anxiety and depression scale	[Honarmand and Feinstein, 2009]
<i>LIFE</i>	Life satisfaction index	[Franchignoni et al., 1999]
<i>OAB</i>	Overactive bladder questionnaire	[Cardozo et al., 2014]
<i>EDINB</i>	Edinburgh handedness inventory	[Oldfield, 1971]
<i>ABILH</i>	Hand ability index	[Arnould et al., 2012]
<i>FIM</i>	Functional independence measure	[Granger et al., 1990]
<i>MOCA</i>	Montreal cognitive assessment	[Dagenais et al., 2013]
<i>PASAT</i>	Paced auditory serial addition task	[Aupperle et al., 2002]
<i>SDMT</i>	Symbol digit modality test	[Parmenter et al., 2007]
<i>EDSS</i>	Expanded disability status scale	[Kurtzke, 1983]

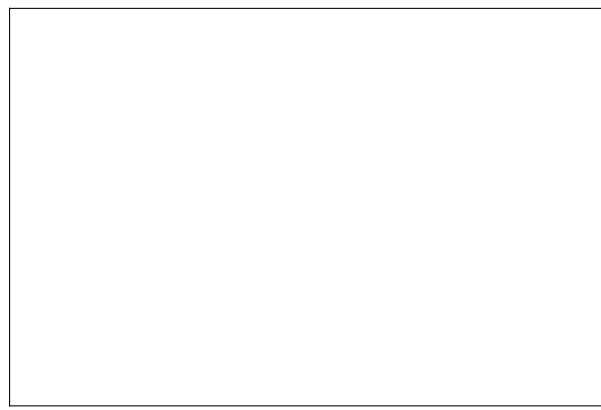
Table 7.1: The set of available PCOs. The first 6 are self-reported, while the last 5 are administered by trained medical staff. In our analysis all PCOs were used, with the exception of *EDSS*.

The collected *PCO* data set comprises additional information such as: i) number of relapses in the last four months (NR), ii) educational level expressed in terms of total years of education (EDU), iii) height (H) and iv) weight (W). Each sample of the data set is represented by a vector of $d = 165$ predictors. Moreover, a neurologist assigns to each patient the corresponding disease course. The global distribution of MS types across time points is depicted in Figure 7.1b.

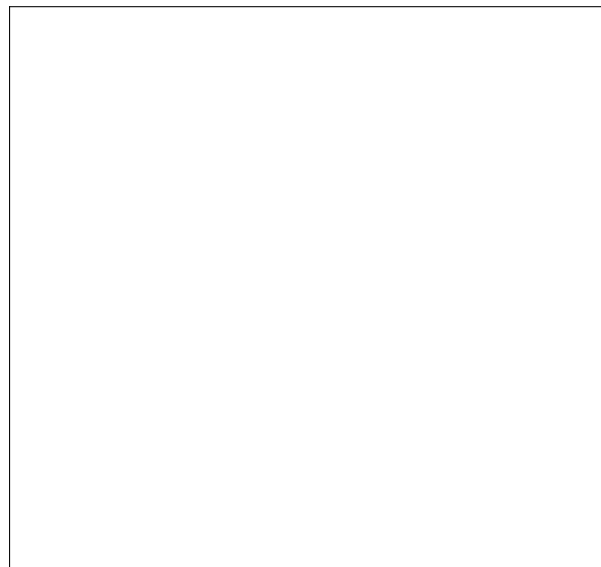
In this work we focus on predicting MS course evolution of *RR* and *SP* patients, hence the subjects with *PR* and *PP* forms will not be taken into account. We considered all the patients with a minimum of 1 time point (the most recently enrolled) up to $T = 8$ time points for a total of 2699 samples, of which 1220 *RR* and 1579 *SP* (see Figure 7.1a). As this is an ongoing project, the number of PwMS decreases with time. We expect to fill the gap of samples between *Exam* 1 and *Exam* 8 by the end of the funded study.

In this work we analyze PCOs data acquired every four months from a cohort of MS patients enrolled in a funded study. Currently, we have collected data for eight examinations and, as patients enrollment is still ongoing, the number of individuals for each time point is successively decreasing

The number of considered samples across examinations is hence 2699, of which 1220 *RR* and 1579 *SP*.



(a)



(b)

Figure 7.1: An overview of the *PCO* data set used in this study. The left panel (a) shows a bar chart of the number of MS patients in each disease form at different examinations. The right panel (b) presents a representation of the distribution of the total amount of acquisitions (3137), divided according to the disease form. **{break figure in two}**

7.2 Problem description

Predicting the MS course evolution can be split in three different related tasks: Current Course Assignment (CCA), PCOs Evolution Prediction (PEP) and Future Course Assignment (FCA). In particular, given the 165-dimensional representation of a patient at a fixed time point \mathbf{x}_i^t , CCA consists in assigning the corresponding disease course y_i^t . Given the historical representation of a patient \mathbf{x}_i^t for $t = 1, \dots, \tau$, PEP consists in predicting the patient representation $\mathbf{x}_i^{\tau+1}$. Finally, FCA consists in foreseeing the MS disease course $y_i^{\tau+1}$ from \mathbf{x}_i^t for $t = 1, \dots, \tau$.

Here, we developed a predictive model that solves these tasks assuming the temporal structure outlined in Figure 7.2. The CCA problem is translated into a binary classification task and we address it by learning a discriminative function $f(\mathbf{x}_i^t) = y_i^t$. The PEP problem is modeled as $g(\mathbf{x}_i^t) = \mathbf{x}_i^{t+1}$, where $g(\mathbf{x})$ is a multiple-output regression function. Once $\hat{f}(\mathbf{x})$ and $\hat{g}(\mathbf{x})$ are learned by training on historical *PCO* data, the FCA problem is finally solved by the temporal model $\hat{f} \circ \hat{g}(\mathbf{x}_i^t) = y_i^{t+1}$. In time-series data analysis, this is known as *one-step-ahead forecast*. Notably, the FCA model allows to foresee if the patient at the next time point is going to experience a transition from *RR* to *SP*, or not.

7.2.1 Data preprocessing

Analyzing *PCO* data is challenging from several respects. First, items belonging to different questionnaires are encoded with numerical values in different ranges. To tackle this issue, we opted for a $[0 - 1]$ scaling of the ordinal answers and a binary one-hot-encoding of the categorical ones. Secondly, as the missing data amount to 1.52% of the entire data set, we resort to the K-nearest neighbor data imputing strategy proposed in [Troyanskaya et al., 2001]. To ensure unbiasedness of the results, this preprocessing phase is not performed on the entire data collection, but it is separately evaluated prior to each model fitting process on its cross-validation portion of the training set, as described in the next section.

7.2.2 Experimental design

We shall discuss separately the experimental designs used to learn $f(\mathbf{x})$ and $g(\mathbf{x})$.

The CCA model $f(\mathbf{x})$ solves a binary classification problem: to each input \mathbf{x}_i^t is associated

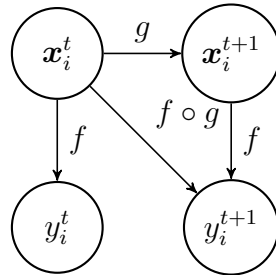


Figure 7.2: A visual representation of the temporal structure assumed in the collected data. When the two functions f (CCA) and g (PEP) are learned, the FCA model $f \circ g$ is able to predict the evolution of the disease course for future time points y_i^{t+1} .

an output y_i^t that encodes the corresponding MS disease course (*RR* or *SP*) with a binary label. We split the data set in three temporal chunks, namely *training*, *validation* and *test* sets, consisting of all samples collected at time points $t = 1, 2, 3$, $t = 4$ and $t = 5, 6, 7, 8$, respectively. Accordingly, we used 1853 samples for training $f(\mathbf{x})$, 398 for validation leaving the remaining 448 for test. Five candidate models for $f(\mathbf{x})$ are fitted on 20 Monte Carlo (MC) random sampling of the training set each time keeping $\frac{1}{4}$ of the samples aside [Molinaro et al., 2005]. For each MC sampling the fitting procedure is performed on the remaining $\frac{3}{4}$ of the samples and it includes an inner parameter optimization via grid-search cross-validation [Hastie et al., 2009]. In particular, we require the MS course prediction to be based on a reduced number of variables (see Section 7.2.3), therefore we enforce sparsity in each candidate model. Leveraging on the MC strategy, we rank the variables according to their selection frequency [Barbieri et al., 2016; Meinshausen and Bühlmann, 2010]. Once a variable ranking is achieved for each candidate model, the list of selected variables is identified by thresholding the corresponding ranking with the threshold that maximizes the accuracy on the validation set. Finally, the last training step consists in fitting each candidate model on the union of training and validation sets taking only into account the corresponding reduced subset of selected variables. The final CCA model $\hat{f}(\mathbf{x})$ is chosen as the one that performs better on the previously unseen test set in terms of accuracy, precision, recall and F_1 score.

On the other hand, learning the PEP model $g(\mathbf{x})$ implies solving a multiple-output regression problem and each input \mathbf{x}_i^t is associated with the output vector \mathbf{x}_i^{t+1} . Therefore, we can only consider samples at time point t with an available follow-up at the next time point $t + 1$, which reduces the overall number of available samples. The data set splitting is consistent with the one followed for learning $f(\mathbf{x})$, although there is no need for a separate validation set, as learning $g(\mathbf{x})$ does not require any variable selection process. We used the samples collected at time points $t = 1, 2, 3, 4$ for training and those at $t = 5, 6, 7, 8$ for test, resulting in 1737 and 254 samples, respectively. The fitting procedure includes an inner parameter optimization via grid-search cross-validation. Each candidate model is a function $g : \mathbb{R}^{165} \rightarrow \mathbb{R}^k$ where k is the number of variables selected by the best CCA model. The final PEP model $\hat{g}(\mathbf{x})$ is chosen as the candidate model that performs better on the previously unseen test set in terms of mean absolute error (MAE).

The predictive capability of the FCA model $\hat{f} \circ \hat{g}(\mathbf{x})$ is finally evaluated on the test set. The CCA model $\hat{f}(\mathbf{x}_i^t)$ predicts the MS course \hat{y}_i^t from the *PCO* data vector $\hat{\mathbf{x}}_i^t$ that, in turn, is predicted by the PEP model $\hat{g}(\mathbf{x}_i^{t-1})$. We shall notice here that the predictions $\hat{f} \circ \hat{g}(\mathbf{x}_i^t) = y_i^{t+1}$ for $t = 8$ are foreseeing possible *RR* to *SP* transitions that are beyond our data observation, hence predictions at the last time point cannot be used to assess the FCA model performance. Therefore, its performance is evaluated only on 220 test samples.

7.2.3 Learning $f(\mathbf{x})$

We imposed $f(\mathbf{x})$ to be sparse. This requirement is helpful from two distinct respects: a) the performance of the predictive model may increase thanks to a reduced effect the course of dimensionality [Hastie et al., 2015] and b) the identification of a reduced subset of meaningful *PCOs* provides interpretability of the results for the clinicians. In order to achieve such sparse model, we take advantage of two main variable selection strategies: embedded and wrapper methods [Guyon and Elisseeff, 2003]. When using embedded methods, we exploited the sparsity inducing penalties of EN to take into account possible correlation between *PCO* variables

and of SLR to benefit from the renowned classification capability of the logistic loss function. We applied the RFE wrapper method to two tree-based learning machines (RF and GB) that are capable of capturing nonlinear relationship between input and output and are intrinsically well-suited to deal with categorical/ordinal variables. We also explored the use of RFE with SVM, as in [Guyon et al., 2002].

7.2.4 Learning $g(x)$

As no prior information on the relationship between PCOs evaluated at different time points was available, to learn $g(x)$ we investigated on the use of both linear and nonlinear models.

Concerning the linear models, we explored two different solutions: NNM and MTEN. The first imposes a low-rank prior on the result. The second is a natural multiple-output extension of EN, hence it induces a row-structured sparsity pattern on the solution where collinear variables are more likely to be included in the model together. For nonlinear prediction, we resorted to the state-of-the-art MLP approach.

7.3 Results

We shall discuss separately the results achieved in terms of CCA, PEP and FCA models.

Regarding CCA, the GB method outperforms the other candidate models reaching accuracy 0.900, precision 0.936, recall 0.899 and F_1 score 0.917, as shown in Figure 7.3a. Therefore we chose it as CCA model $\hat{f}(x)$. Insights on the use of PCOs for MS assessment are provided by the sparsity of the CCA model induced by the RFE schema. The 31 selected variables are reported in Table ???. Comparing the full list of *PCO* questionnaires of Table 7.1 with Table ??, we observe that each *PCO* used in this study is represented at least once, except EDINB, and the most represented is FIM. We also see that, whenever possible, the model tends to select aggregate scores (total and subtotal) rather than single items. This is consistent with the clinical practice, where neurologists are more likely to assess patient’s health status by using the aggregate scores, rather than the single questions. Quite surprisingly, the recent number of relapses is the only additional information not selected by the model. Finally, we note that all the domains that are known to be affected by the disease are well covered: mobility (upper and lower limbs), cognition, emotional, fatigue, bladder and psychosocial. The heatmap in Figure 7.3b shows the Hamming distance estimated across the list of variables selected by the five CCA candidate models. Interestingly, tree-based methods are more prone to select similar variables with respect to linear methods. As expected, the sparsity induced by the ℓ_1 -norm of SLR allows the method to achieve a list of variables similar to the one obtained by SVM-RFE, while the list obtained by ENET includes collinear variables and it is significantly different from the others.

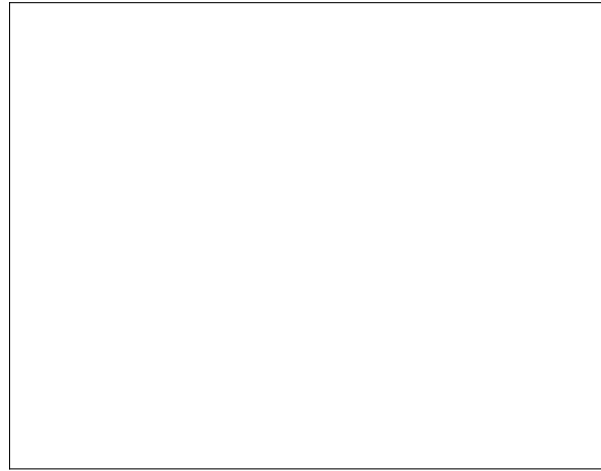
Regarding PEP, MTEN outperforms the other candidate models in terms of MAE ($\text{MAE}_{\text{MTEN}} = 0.095$, $\text{MAE}_{\text{NNM}} = 0.102$, $\text{MAE}_{\text{MLP}} = 0.105$), hence we select it as our PEP model $\hat{g}(x)$.

Finally, the FCA model $\hat{f} \circ \hat{g}(x)$, obtained by combining MTEN and GB achieves the following performance scores on the 220 test samples: accuracy 0.841, precision 0.900, recall 0.824 and F_1 score 0.860.

In this work we proposed a novel temporal model based on patient-centered outcomes and



(a)



(b)

Figure 7.3: A visual representation of the results obtained from the CCA model. On the left panel (a) we show the classification performance achieved on the test set by the candidate models. Precision, recall and F_1 score are estimated considering SP as the positive class. As GB outperforms the other methods on each performance metric, it is chosen as CCA model. On the right panel (b) a heatmap displays the distance between the lists of variables selected by each model in terms of their hamming distance. **{break figure in two}**

machine learning for disease form prediction in multiple sclerosis. In particular, we address the tasks of current course assignment, PCOs evolution prediction and future course assignment. The model is built on a collection of PCOs acquired on a cohort of individuals enrolled in an ongoing funded study (*DETECT-MS PRO*). PCOs data are typically used to corroborate evidence provided by quantitative exams, in our case the absence of clear MS disease form predictors makes the information extracted from PCOs data the only available resource. The proposed temporal model was able to correctly assign the current MS form and to foresee future ones with accuracy of 90.0% and 84.1%, respectively. This demonstrates that PCOs can effectively be used as MS disease course predictor. In the next future, we plan to further investigate on the predictive capabilities of the proposed model with longer temporal horizons and to compare it with different approaches, such as probabilistic graphical models. Given the achieved promising results, the proposed model is soon going to be validated in clinical practice, where it will assist the clinicians involved in this study to foresee possible disease course transition and to

take important decisions concerning treatment and therapies that can substantially improve the quality of life of their patients. In the context of neurodegenerative diseases, clinicians typically use PCOs data to corroborate evidences coming from standard quantitative exams [Black, 2013]. Interestingly, in our case the absence of clear *SP* predictors makes the information extracted from PCOs data the only available resource. In the era of precision medicine, the problem of predicting MS course evolution still relies on stressful exams and clinical judgement. To the best of our knowledge, this is the first attempt to solve this delicate task leveraging on patient-friendly measures and machine learning.

8 Temporal model for glucose predictions

Over the past decade, continuous glucose monitoring (CGM) has proven to be a very resourceful tool for diabetes management. To date, CGM devices are employed for both retrospective and online applications. Their use allows to better describe the patients' pathology as well as to achieve a better control of patients' level of glycemia. The analysis of CGM sensor data makes possible to observe a wide range of metrics, such as the glycemic variability during the day or the amount of time spent below or above certain glycemic thresholds. However, due to the high variability of the glycemic signals among sensors and individuals, CGM data analysis is a non-trivial task. Standard signal filtering solutions fall short when an appropriate model personalization is not applied. State-of-the-art data-driven strategies for online CGM forecasting rely upon the use of recursive filters. Each time a new sample is collected, such models need to adjust their parameters in order to predict the next glycemic level. In this paper we aim at demonstrating that the problem of online CGM forecasting can be successfully tackled by personalized machine learning models, that do not need to recursively update their parameters.

8.1 Diabetes care

Diabetes is a chronic metabolic disorder affecting nearly 400 million of individuals worldwide. The number of diabetic patients is increasing and it is expected to reach almost 600 million in the next future [?]. According to the *World Health Organization* [?], the global prevalence of diabetes among adults has nearly doubled in the last few decades, rising from 4.7% in 1980 to 8.5% in 2014. If not treated correctly, diabetes may cause several permanent complications, such as visual impairment and kidney failure [?].

Hypoglycemia is a severe risk in diabetes therapy. The mean incidence of hypoglycemia in patients with type 1 diabetes (T1D) is 1-2 events per week, while severe hypoglycemia occurs 0.1-1.5 episodes per year [?]. Moreover, hypoglycemia interferes with the quality of life and increases the risks of cardiovascular events in type 2 diabetes (T2D) patients [?]. On the other hand, hyperglycemia associates with an increased risk of diabetes complication as well.

The most common glucose monitoring solutions are self blood glucose meters and continuous glucose monitoring systems (CGM). CGM devices are minimally-invasive, and can be used in daily life for retrospective or online applications [?]. CGM systems measure interstitial glucose concentration at fixed time intervals, enabling an accurate observation of glycemic variability during the day as well as the ratio of time spent in hypo/hyperglycemia. When CGM is employed online, an improvement of the therapy can be achieved by embedding in the system a tool that foresees glucose levels using suitable time-series models trained on past CGM data [?]. In this case, alarms can be generated when the glucose concentration exceeds the normal range [?].

To model patient-specific blood glucose (BG) levels, several approaches integrating various information were proposed [??]. However, the problem of glycemic level prediction is still

challenging, due to the high CGM signal variability among patients and acquisition devices. In this paper, assuming that CGM signals have structural information on time-changes of BG concentration [??], we explore the performance of a collection of purely data-driven techniques for time-series analysis.

Data-driven approaches, as opposed to models driven by an *a-priori* description of the underlying phenomena, are capable of modeling input/output relationships without requiring prior knowledge on the field of use. This family of approaches can be successfully employed to obtain reliable predictions without modeling complex, and possibly not completely understood, environments. Data-driven models are widely applied in real-world scenarios often employing a moving-window paradigm, i.e., the system keeps track of the last w acquisitions using them to forecast future values.

In this chapter, we focus on two main groups of data-driven methods for online time-series forecasting: *recursive filters* and *machine learning* models. The first group includes linear stationary models, such as Autoregressive Moving Average (ARMA), non stationary models, such as Autoregressive Integrated Moving Average (ARIMA) and adaptive filtering techniques, such as the Kalman Filter (KF). These methods are well-established and extensively used, but they require recursive parameters adjustment for every new collected sample [?]. The second group comprises regularized kernel methods, such as Kernel Ridge Regression (KRR) and deep learning methods, such as Long Short-Term Memory networks (LSTM). In the recent past, machine learning methods showed very promising results in the context of time-series forecasting [?Schmidhuber et al., 2005]. To achieve a predictive model, they need to *learn* the model parameters from a training set. This learning process is done only once and it conveys a model that does not require further parameter adjustments to predict future values. This remarkable advantage makes machine learning models more suitable to be delivered on embedded portable systems.

Our work aims at understanding whether purely data-driven machine learning methods can be successfully employed to forecast glucose values of diabetic patients.

8.2 Methods

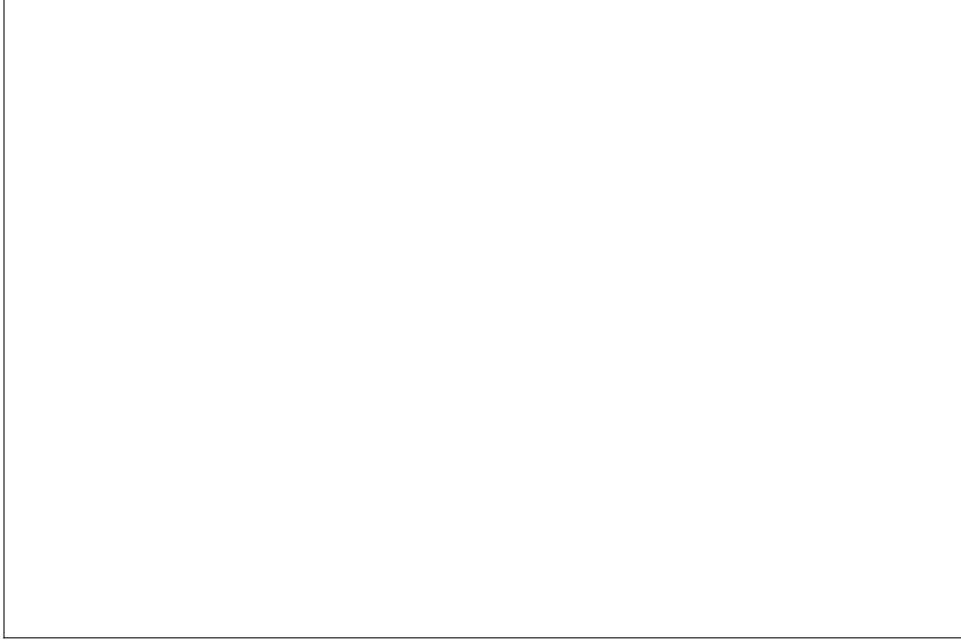
Given a set of data \mathcal{S} , we refer to data-driven models with hyperparameters θ as $\mathcal{M}_{\mathcal{S}}(\theta)$. Given the time-series $y(t)$ we aim at predicting $y(t + \Delta T)$, where ΔT is some *prediction horizon*.

A well-known issue of CGM sensor data analysis is that signal properties, such as the SNR, may vary among devices and individuals [?]. In order to achieve an accurate forecast of the glucose level of a given individual from past samples, any prediction model $\mathcal{M}_{\mathcal{S}}(\theta)$ must be *personalized*. Such model personalization procedure is two-fold: *hyperparameters* (θ) *optimization*, also known as model selection, and *parameter estimation*, or model fitting. In this work we fixed a common strategy for hyperparameters optimization whilst the actual model fitting procedure was defined differently according to the model.

8.2.1 Experimental Setting

The hyperparameters optimization strategy can be described as follows. Given $y(t)$ for time points $t = 1, \dots, T$, we split the time-series in two chunks: an initial *burn-in* of $T' = 300$ CGM

Figure 8.1: An example of two glycemic profiles obtained from T1D and T2D patients. The glucose target range is set between 70 mg/dL and 140 mg/dL (dashed lines). The yellow area at the left hand side of the plot is the initial *burn-in* interval used for model *personalization*.



observations and the *experiment set* made of the subsequent $T - T'$ samples (see Figure 8.1). For each model $\mathcal{M}_S(\theta)$, and for each subject, we use the burn-in samples to optimize the hyperparameters θ by grid-search. In other words, the best hyperparameters θ^* are chosen in order to minimize an index $J(\theta)$ estimated on cross-validation (CV) splits and defined differently according to the model. In the context of time-series, the CV training split consists only of observations occurred before the corresponding validation samples; such CV flavor is sometimes referred to as *rolling forecasting origin* [?]. The grid-search CV scheme promotes the identification of models capable of returning robust predictions. Once the best hyperparameters θ^* are identified, the model $\mathcal{M}_S(\theta^*)$ is fitted on the whole burn-in and it is used to perform online forecasting on the data points of the experiment set (i.e., for $t = T' + 1, \dots, T$). For each personalized model $\mathcal{M}_S(\theta^*)$ we calculate the accuracy for prediction horizons ΔT of 30, 60, and 90 minutes. The performance is estimated by *Root Mean Square Error*

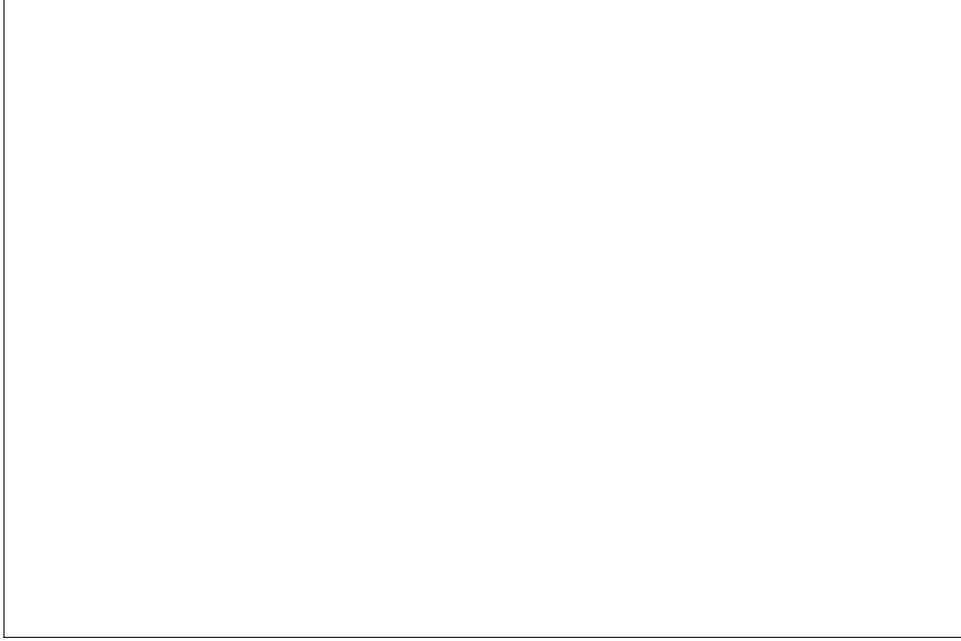
$$\text{RMSE} = \sqrt{\frac{\sum_t (y(t + \Delta T) - \hat{y}(t + \Delta T))^2}{n}}$$

and *Mean Absolute Error*

$$\text{MAE} = \frac{\sum_t |y(t + \Delta T) - \hat{y}(t + \Delta T)|}{n}$$

assuming n samples in the experiment set. Fixing some notation, we use $w = 36$ as the size of the window in moving-window paradigm and we indicate with $\{\mathbf{x}_i, y_i\}_{i=1}^N$ the input/output pairs for machine learning model. Each input \mathbf{x}_i is a w -dimensional vector made of the CGM samples acquired at times $t = t_{i-w}, \dots, t_i$, while the corresponding output y_i is the CGM acquisition at time t_{i+1} .

Figure 8.2: Plots reporting the distributions of LBG1 (left) and HBG1 (right) for T1D and T2D. Green areas at the bottom of each plot represents low risk of hypo/hyperglycemia events.



8.2.2 Autoregressive Integrated Moving Average

ARIMA methods can be used to perform linear forecasting of non stationary time-series assuming that its d -th difference is a stationary ARMA process [?]. The output of an ARMA process, with white input noise $u(t) \sim \mathcal{N}(0, \sigma^2)$, can be expressed as in Equation (8.1).

$$y(t) = - \sum_{k=1}^p a_k y(t-k) + \sum_{k=0}^q b_k u(t-k) \quad (8.1)$$

The output of an ARMA(p, q) model can be seen as the sum of p autoregressive and q moving average terms. This strategy requires the modeled processes to be stationary. On the other hand, when a time-series can be considered stationary after d differentiations, ARIMA(p, d, q) models can be used. This is the case for non stationary time-series that exhibit local stationary behavior. In general, (p, d, q) are unknown and we will consider them as hyperparameters of the ARIMA model. Here, the CV index for ARIMA models was defined as $J(\boldsymbol{\theta}) = \text{AIC}(\mathcal{M}_{\mathcal{S}}(\boldsymbol{\theta})) + \bar{\varepsilon}_{\text{cv}}$, where AIC stands for Akaike Information Criterion [?] and $\bar{\varepsilon}_{\text{cv}}$ is the average CV squared error [?]. We refer to [?] for a detailed description of ARIMA model fitting. The application of this class of models to predict CGM sensor data was also explored in [??].

8.2.3 Kalman Filter

The KF addresses the problem of estimating the state $\mathbf{x} \in \mathbb{R}^d$ of a discrete-time process governed by the linear stochastic difference equation $\mathbf{x}(t+1) = F\mathbf{x}(t) + \mathbf{w}(t)$ with measurements $\mathbf{y} \in \mathbb{R}^k$, $\mathbf{y}(t) = H\mathbf{x}(t) + \mathbf{v}(t)$, where $\mathbf{w}(t)$ and $\mathbf{v}(t)$ are independent random variables representing state and measurement noise, respectively [?]. It is usually assumed that $\mathbf{w}(t) \sim \mathcal{N}(\mathbf{0}, Q)$ and $\mathbf{v}(t) \sim \mathcal{N}(\mathbf{0}, R)$, with both Q and R unknown. In the context of CGM sensor data prediction, we can safely assume that the state space is two-dimensional, hence $x_1(t) = u(t)$,

$x_2(t) = u(t - 1)$ where the unknown signal $u(t)$ is described *a-priori* as an integrated random-walk $u(t) = 2u(t - 1) - u(t - 2) + w(t)$ as in [?]. Consequently, the state space transition matrix F can be written as $F = \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix}$, while the measurement vector is $H = \begin{bmatrix} 1 & 0 \end{bmatrix}$. The process noise covariance is $Q = \begin{bmatrix} \lambda^2 & 0 \\ 0 & 0 \end{bmatrix}$ and the measurement noise covariance is $R = \sigma^2$, as in [?]. Both λ^2 and σ^2 are unknown and we will consider them as hyperparameters of the KF forecasting strategy, while for the CV index we resort to $J(\theta) = \bar{\varepsilon}_{cv}$. The application of KF to predict CGM sensor data was also explored in [??].

8.3 Data

We acquired CGM samples from a group of 148 T1D and T2D patients wearing the iPro[®]2 Professional CGM sensor (Medtronic), which reported glucose values every 5 minutes. Patients were monitored for up to 7 days in free living conditions, keeping track of their treatments. From this initial cohort, we excluded the 18 individuals which acquisitions lasted for less than 3.5 days as well as the 24 time-series that presented artifacts due to incorrect use of the CGM acquisition device. Hence, our final dataset comprises 106 subjects of which 72 T1D and 34 T2D. On average, glycemic variability is relatively high, with 170.7 ± 70.0 mg/dL for T1D and 158.4 ± 43.6 mg/dL for T2D. Figure 8.1 shows two examples, one for each diabetes type. For each patient, the risk of hypo/hyperglycemia was determined by computing the Low Blood Glucose Index (LBGI) and the High Blood Glucose Index (HBGI), defined as in [?]. These two indices are based on the frequency and extent of hypo- and hyper-glycemic episodes. Figure 8.2 shows the LBGI and HBGI distributions for T1D and T2D.

8.4 Results

Taking into account the available information on treatments, we divided the dataset into four groups, namely T1D with microinfusion pump (32 subjects), T1D with insulin injection (40 subjects), T2D with rapid acting insulin (10 subjects) and T2D with other therapies (24 subjects). For each group, we applied all forecasting models whose performance, expressed in term of MAE and RMSE, is presented in Table 8.1. The forecasting errors clearly increase with the prediction horizon as the errors accumulate at each forecasted time step. KRR achieves the most accurate prediction overall and for almost every group of patients, although ARIMA results are comparable. Moreover, its loss of accuracy from the first to the last prediction horizon is lower than for ARIMA. Figure 8.3 displays an example of time-series forecast obtained with KRR showing the three different prediction errors at 30, 60 and 90 minutes.

8.5 Conclusions and Future Works

In this work we compared the performance of two types of data-driven strategies for online CGM forecasting: recursive filters and machine learning models. We also proposed a general procedure for model personalization based on cross-validation with hyperparameters optimization via grid-search. Finally, we showed that reliable CGM predictions can be obtained with machine learning models that do not need to recursively adjust their parameters along time. In the future, to

Figure 8.3: Online time-series forecasting obtained by KRR model. *One-step-ahead prediction* (left): the green solid line shows the available samples; the dashed line represents one-step-ahead predictions that are obtained by applying the model on a moving window of 36 time-points. *Open-loop forecast* (right): with passing time, the moving-window incorporates an increasing number of predicted points, accumulating errors; the dashed line represents forecast with a prediction horizon of 90'. Absolute prediction errors are evaluated with respect to future measures (red solid line).



improve the performance on long prediction horizon, we plan to investigate on sparse data representations with deeper architectures or regularized dictionary learning approaches.

Table 8.1: Prediction errors (std) of the forecasting models for increasing prediction horizons overall and on the four groups. *MP* = Microinfusion Pump, *II* = Insulin Injection, *RAI* = Rapid Acting Insulin, *Other* = Other therapies. Bold numbers indicate best result.

ARIMA						
	MAE mg/dL			RMSE mg/dL		
	30	60	90	30	60	90
Overall	11.5 (5.2)	25.6 (12.5)	37.2 (20.3)	16.7 (7.9)	37.1 (19.3)	53.8 (31.8)
T1D - <i>MP</i>	13.3 (3.6)	30.3 (7.4)	45.3 (12.5)	19.5 (5.4)	43.9 (10.9)	64.9 (18.1)
T1D - <i>II</i>	13.4 (6.3)	31.0 (16.2)	46.1 (27.1)	19.7 (10.6)	45.1 (27.0)	66.6 (45.7)
T2D - <i>RAI</i>	11.0 (3.3)	23.8 (6.5)	33.2 (8.5)	16.2 (4.8)	34.1 (8.3)	46.9 (10.9)
T2D - <i>Other</i>	10.0 (5.1)	21.4 (10.4)	29.2 (12.5)	13.6 (6.0)	29.6 (12.7)	41.3 (16.5)

KF						
	MAE mg/dL			RMSE mg/dL		
	30	60	90	30	60	90
Overall	46.8 (23.2)	50.8(25.4)	159.6 (43.5)	58.3 (27.7)	63.1 (30.3)	169.9 (47.1)
T1D - <i>MP</i>	56.6 (15.9)	59.5 (15.2)	163.5 (26.7)	70.5 (19.0)	74.3 (18.4)	177.5 (29.9)
T1D - <i>II</i>	59.9 (24.5)	67.9 (26.7)	179.2 (38.9)	74.4 (28.7)	83.5 (31.3)	193.8 (40.4)
T2D - <i>RAI</i>	48.6 (20.4)	50.9 (20.8)	195.2 (63.1)	60.4 (22.9)	63.6 (24.1)	204.4 (63.8)
T2D - <i>Other</i>	33.0 (12.1)	35.6 (15.8)	145.0 (30.4)	41.3 (14.2)	44.3 (17.7)	150.1 (31.8)

KRR						
	MAE mg/dL			RMSE mg/dL		
	30	60	90	30	60	90
Overall	11.1 (4.3)	25.1 (10.4)	35.2 (15.4)	15.5 (6.7)	33.6 (13.8)	45.8 (19.7)
T1D - <i>MP</i>	12.9 (3.4)	30.2 (8.4)	43.1 (12.6)	17.9 (5.1)	40.2 (10.9)	56.0 (16.1)
T1D - <i>II</i>	13.1 (4.9)	30.3 (10.6)	43.1 (14.5)	18.6 (8.4)	40.5 (14.5)	55.8 (18.6)
T2D - <i>RAI</i>	11.2 (2.9)	24.1 (5.5)	33.2 (7.1)	16.0 (4.7)	32.7 (7.5)	43.7 (9.5)
T2D - <i>Other</i>	8.7 (2.3)	19.8 (6.9)	27.7 (11.8)	11.9 (3.4)	26.5 (8.9)	36.2 (14.4)

LSTM						
	MAE mg/dL			RMSE mg/dL		
	30	60	90	30	60	90
Overall	19.9 (10.6)	44.0 (25.9)	61.3 (36.5)	26.2 (13.4)	55.0 (30.0)	74.7 (41.3)
T1D - <i>MP</i>	22.9 (9.1)	51.6 (21.5)	71.6 (27.0)	30.6 (11.6)	65.7 (27.0)	89.8 (33.5)
T1D - <i>II</i>	24.5 (12.9)	56.5 (33.1)	81.4 (45.8)	31.2 (14.9)	68.8 (35.9)	97.0 (48.4)
T2D - <i>RAI</i>	18.3 (6.7)	39.4 (14.8)	54.1 (21.0)	25.7 (10.1)	51.6 (21.7)	67.7 (29.1)
T2D - <i>Other</i>	16.8 (8.9)	35.0 (17.7)	47.0 (26.5)	22.2 (12.1)	43.4 (19.0)	56.5 (27.3)

9 Conclusion

A Appendix

As already pointed out at the beginning of Chapter 4, ML is a cross-disciplinary field and the statistical tools used in literature to describe models and algorithms heavily depend on the academic background of the author. This can make the approach to ML fascinating and somewhat cumbersome at the same time.

The goal of this appendix is to shed light on some of the statistical tools and definitions that are typically left unsaid, or given for granted, by most of the authors. In particular, in the following sections insightful statistical details on the formulation of the supervised learning problem expressed in Equation (4.2) will be provided.

A.1 Useful theorems and definitions

This first section lists the theorems and the definitions that are useful for the comprehension of the following sections.

Theorem 1 (Law of the unconscious statistician) *Given two continuous random variables $(a, b) \in A \times B$ with joint probability distribution $p(a, b)$, the expected value of the function $g(a, b)$ can be stated as follows.*

$$\mathbb{E}[g(a, b)] = \iint_{A \times B} g(a, b) p(a, b) da db$$

Theorem 2 (Bayes rule) *Given A and B two events with probability $P(A)$ and $P(B) \neq 0$, the conditional probability of observing A given that B is true is*

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

where $P(B|A)$ is the probability of observing B given that A is true.

Definition 1 (Well-posed problem) *A problem is **well-posed** if its solution: (i) exists, (ii) is unique, (iii) depends continuously on the data (e.g. it is stable).*

Definition 2 (Ill-posed problem) *A problem is **ill-posed** if it is not well-posed.*

Definition 3 (Likelihood function) *Let α be a continuous random variable with probability distribution $p(\alpha, \phi)$ depending on the parameter ϕ ; then the function $\mathcal{L}(\phi|\bar{\alpha}) = p(\bar{\alpha}, \phi)$ is the likelihood function of ϕ given that $\bar{\alpha}$ is the outcome of α .*

A.2 Empirical risk minimization

In Section 4.1 we introduced the concept of supervised learning as the branch of ML in which predictive models are trained on labeled data. The final goal of supervised learning is to find a function of the input variables $f : \mathcal{X} \rightarrow \mathcal{Y}$ that provides a *good* approximation of the output y . In order to measure the adherence between predictions $\hat{y} = f(\mathbf{x})$ and actual output y , we introduced the concept of *loss function* $L(\hat{y}, y)$, see Table 4.1. For a fixed choice of the loss, the ideal estimator, also known as the *target function*, $f^*(\mathbf{x})$ is the minimizer of the (true) expected risk $\mathcal{E}(f)$ in a rather *large* class of functions \mathcal{F} .

$$f^*(\mathbf{x}) = \operatorname{argmin}_{f \in \mathcal{F}} \mathcal{E}(f) \quad (\text{A.1})$$

Applying the law of the unconscious statistician, stated in Theorem 1, the expected risk $\mathcal{E}(f)$ can be written as in Equation (A.2), where (\mathbf{x}, y) are two random variables with joint probability distribution $p(\mathbf{x}, y)$.

$$\mathcal{E}(f) = \mathbb{E}[L(f(\mathbf{x}), y)] = \iint_{\mathcal{X} \times \mathcal{Y}} L(f(\mathbf{x}), y) p(\mathbf{x}, y) d\mathbf{x} dy \quad (\text{A.2})$$

In real situations, a direct computation of $\mathcal{E}(f)$ is unfeasible as the joint probability distribution $p(\mathbf{x}, y)$ is unknown. Although, we assume to be provided with a collection of input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ that are supposed to be sampled *i.i.d.* from $\mathcal{X} \times \mathcal{Y}$ according to $p(\mathbf{x}, y)$. In statistical learning theory, as introduced by Vapnik [Vapnik, 2013], the dataset \mathcal{D} can be used to build a stochastic approximation of $\mathcal{E}(f)$ called *empirical risk* $\mathcal{E}(f_{\mathcal{D}})$ and defined in Equation (A.3).

$$\mathcal{E}(f_{\mathcal{D}}) = \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i), y_i) \quad (\text{A.3})$$

As \mathcal{D} is drawn according to the probability distribution $p(\mathbf{x}, y)$, our hope is that the empirical risk can be used as a proxy for the expected risk, hence $\mathcal{E}(f_{\mathcal{D}}) \approx \mathcal{E}(f)$. The solution of the supervised learning problem is then found by *Empirical Risk Minimization* (ERM), defined in Equation (A.4).

$$\hat{f}(\mathbf{x}) = \operatorname{argmin}_{f \in \mathcal{F}} \mathcal{E}(f_{\mathcal{D}}) = \operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i), y_i) \quad (\text{A.4})$$

In practice, minimizing $\mathcal{E}(f_{\mathcal{D}})$ instead of $\mathcal{E}(f)$ comes at a price. The central problem is whether the first is a good approximation of the second. For instance, when $p(\mathbf{x}, y)$ is too *complex*, the number of examples is too small and/or the class of functions \mathcal{F} is *too large*, $\hat{f}(\mathbf{x})$ will be far from the target function $f^*(\mathbf{x})$, even when its empirical error is 0. In real circumstances, it is impossible to control the true probability distribution and it is often extremely difficult to collect a very large number of examples. The only element we can control is the class of functions \mathcal{F} and, in particular, its *size*. Since Tikhonov [Tikhonov, 1963] it is known that, for an arbitrary function space \mathcal{F} , the ERM problem is ill-posed (see Definition 2). A possible way to ensure

well-posedness is to impose a constrain that restricts the function space. Hence, the constrained ERM problem assumes the form in Equation (A.5), where $\lambda \neq 0$.

$$\begin{aligned} \operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i), y_i) \\ \text{subject to } \mathcal{R}(f) < \frac{1}{\lambda} \end{aligned} \quad (\text{A.5})$$

Applying the *Lagrange multipliers technique*¹ Equation (A.5) can be finally written as Equation (A.6).

$$\operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i), y_i) + \lambda \mathcal{R}(f) \quad (\text{A.6})$$

The penalty term $\mathcal{R}(f)$ acts as regularizer² and, according to its definition, it can ensure well-posedness of the problem and it can enforce different interesting properties on the achieved solution, see Section 4.1.1. It can also be shown that the use of appropriate regularizes promotes generalization, hence increases our chance to find a $\hat{f}(\mathbf{x})$ close to $f^*(\mathbf{x})$.

According to the choice made for $L(\cdot)$ and $\mathcal{R}(\cdot)$, the minimization problem posed in Equation (A.6) can have very different properties; it can be convex or non-convex, it can include differentiable as well as non-differentiable terms. A rigorous review of the most common optimization methods for ML is beyond the scope of this thesis and can be found here [Boyd and Vandenberghe, 2004; Bach et al., 2012; Sra et al., 2012; Nesterov, 2013].

A.3 Maximum likelihood estimation

In this section we will see a different approach to tackle the supervised learning problem. Once again, let the training data be made of input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, with $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}, \forall i = 1, \dots, n$. This approach relies on the expression of the uncertainty over the value of y with a probability distribution $p(y|\mathbf{x}, \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \Theta$. Applying Definition 3, and assuming that the samples are drawn *i.i.d.*, we can write the likelihood function for $\boldsymbol{\theta}$ as in Equation (A.7).

$$\mathcal{L}(y|\mathbf{x}, \boldsymbol{\theta}) = \prod_{i=1}^n p(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \quad (\text{A.7})$$

Equation (A.7) can be considered as the probability of observing the output y_i , given the input \mathbf{x}_i and the parameters $\boldsymbol{\theta}$ ($\forall i = 1, \dots, n$). This statistical setup suggests a strategy to obtain an estimate for $\boldsymbol{\theta}$ known as *Maximum Likelihood Estimation* (MLE), see Equation (A.8).

¹for a thorough description of this technique, see Appendix E of Bishop's book [Bishop, 2006]

² $\mathcal{R}(f)$, in general, can be thought as $\mathcal{R}(f) = \|f\|_K^2$ where $\|\cdot\|_K^2$ is the norm defined by the kernel K in a *Reproducing Kernel Hilbert Space* \mathcal{H} [Evgeniou et al., 2000]

$$\hat{\theta}_{MLE} = \underset{\theta \in \Theta}{\operatorname{argmax}} \mathcal{L}(y_i | \mathbf{x}_i, \theta) = \underset{\theta \in \Theta}{\operatorname{argmax}} \prod_{i=1}^n p(y_i | \mathbf{x}_i, \theta) \quad (\text{A.8})$$

Instead of maximizing the likelihood it is often convenient to minimize the *negative log-likelihood*³. Equation (A.8) can then be rewritten as Equation (A.9).

$$\hat{\theta}_{MLE} = \underset{\theta \in \Theta}{\operatorname{argmin}} - \sum_{i=1}^n \log p(y_i | \mathbf{x}_i, \theta) \quad (\text{A.9})$$

Moreover, if some prior knowledge on θ is available, it is possible to incorporate it in the form of a prior distribution $p(\theta)$. Applying the Bayes rule (Theorem 2) it is possible to write the *posterior distribution* $p(\theta | y, \mathbf{x})$ as in Equation (A.10).

$$p(\theta | y, \mathbf{x}) = \frac{p(y | \mathbf{x}, \theta) \cdot p(\theta)}{p(y | \mathbf{x})} \quad (\text{A.10})$$

The normalizing constant in Equation (A.10) $p(y | \mathbf{x})$ is independent from θ . It is known as the marginal likelihood and it can be estimated as in Equation (A.11).

$$p(y | \mathbf{x}) = \int p(y | \mathbf{x}, \theta) \cdot p(\theta) d\theta \quad (\text{A.11})$$

Equation (A.10) suggest a new strategy to achieve an estimate for θ that takes into account the prior distribution. This criterion is stated in Equation (A.12) and it is known as *Maximum A Posteriori* (MAP).

$$\hat{\theta}_{MAP} = \underset{\theta \in \Theta}{\operatorname{argmax}} p(\theta | y, \mathbf{x}) = \underset{\theta \in \Theta}{\operatorname{argmax}} p(y | \mathbf{x}, \theta) \cdot p(\theta) \quad (\text{A.12})$$

Finally, given that $p(y | \mathbf{x}, \theta)$ is the likelihood of θ (see Definition 3), we can assume *i.i.d.* samples and apply the negative log-likelihood trick to rewrite Equation (A.12) as in Equation (A.13).

$$\hat{\theta}_{MAP} = \underset{\theta \in \Theta}{\operatorname{argmin}} - \left[\sum_{i=1}^n \log p(y_i | \mathbf{x}_i, \theta) + \log p(\theta) \right] \quad (\text{A.13})$$

Fixing the two distributions, the predictive model can be achieved solving the minimization problem in Equation (A.13). For the solution of this minimization problem the same observations provided at the end of the last section for Equation (A.6) hold.

A.4 ERM vs MLE/MAP

The goal of this last section is to show that the approaches described in Section A.2 and in Section A.3 look very different, but they actually are two sides of the same coin.

³approaching information theory or deep learning literature, the negative log-likelihood is often referred to as *cross-entropy*

Assuming that we have the usual collection of *i.i.d.* samples $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} \forall i = 1, \dots, n$, our aim here is to learn a good input-output relationship $f : \mathcal{X} \rightarrow \mathcal{Y}$. The function f may depend from some parameters that, for ease of writing, will be temporary omitted. If we decide to proceed by MLE, we can find \hat{f}_{MLE} solving the optimization problem in Equation (A.14).

$$\hat{f}_{MLE} = \operatorname{argmax}_{f \in \mathcal{F}} \prod_{i=1}^n p(y_i | \mathbf{x}_i, f) \quad (\text{A.14})$$

Applying the negative log-likelihood trick, Equation (A.14) can be rewritten as Equation (A.15)⁴.

$$\hat{f}_{MLE} = \operatorname{argmin}_{f \in \mathcal{F}} -\frac{1}{n} \sum_{i=1}^n \log p(y_i | \mathbf{x}_i, f) \quad (\text{A.15})$$

As we can see here, we are naming *negative log-likelihood* what in Section A.2 was called *loss function*. In fact, using $L(f(\mathbf{x}), y) = -\log p(y | \mathbf{x}, f)$ Equation (A.15) can be rewritten as in Equation (A.16), which is the ERM problem.

$$\hat{f}_{ERM} = \operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i), y_i) \quad (\text{A.16})$$

In Section A.2 we have seen that introducing $\mathcal{R}(f)$ reduces the space of functions \mathcal{F} and prevents the achieved solution from overfitting. Intuitively, the same effect can be achieved by introducing a prior $p(f)$ as in the MAP estimate. Following Equation (A.13) we can write Equation (A.17).

$$\hat{f}_{MAP} = \operatorname{argmin}_{f \in \mathcal{F}} -\frac{1}{n} \left[\sum_{i=1}^n \log p(y_i | \mathbf{x}_i, f) + \lambda \log p(f) \right] \quad (\text{A.17})$$

Finally, as for Equation (A.16), we can express the penalty as $R(f) = -\frac{\lambda}{n} \log p(f)$ and Equation (A.17) becomes Equation (A.18), which is the classical *Loss + Penalty* formulation of the ERM problem.

$$\hat{f}_{ERM} = \operatorname{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n L(f(\mathbf{x}_i), y_i) + \lambda R(f) \quad (\text{A.18})$$

In this section an intuitive explanation of the connection between two popular supervised learning approaches is provided. For a more rigorous overview on ERM and MLE/MAP we refer to [Hastie et al., 2009] and to [Rasmussen and Williams, 2006], respectively.

⁴the two minimization problems $\min_x f(x)$ and $\min_x \alpha f(x)$ have the same solution if α is a constant that does not depend from x

A.4.1 Linear regression revisited

To clarify the connection between ERM and MLE/MAP we can revisit the simple linear regression problem.

Once again, we have a collection of *i.i.d.* samples $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n = (X, \mathbf{y})$, where $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y} \forall i = 1, \dots, n$ and our aim is to learn an input-output relationship $f : \mathcal{X} \rightarrow \mathcal{Y}$. Moreover, we assume that the outputs y_i is affected by additive Gaussian noise, hence $y_i = f(\mathbf{x}_i) + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$. Interestingly, this corresponds to the assumption that f is modeling the mean of the outputs y_i , while its standard deviation σ_n remains unknown, therefore $y_i \sim \mathcal{N}(f(\mathbf{x}_i), \sigma_n^2)$ ($\forall i = 1, \dots, n$). In Section A.2 we have seen that the ERM solution can be estimated as in Equation A.16. For the sake of simplicity we can restrict to the Ridge Regression case (see Section 4.1.1), *i.e.* we look for a model that can be written as $\hat{y} = f(\mathbf{x}) = \mathbf{x}^T \hat{\mathbf{w}}$ minimizing the square loss $L(\hat{y}, y) = \frac{1}{n} \|\mathbf{y} - X\mathbf{w}\|_2^2$ penalized by the ℓ_2 -norm $\mathcal{R}(\mathbf{w}) = \|\mathbf{w}\|_2^2$. Therefore, the minimization problem is stated in Equation (A.19).

$$\hat{\mathbf{w}}_{\ell_2} = \underset{\hat{\mathbf{w}} \in \mathbb{R}^d}{\operatorname{argmin}} J(\mathbf{y}, X, \mathbf{w}) = \underset{\hat{\mathbf{w}} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2n} \|\mathbf{y} - X\mathbf{w}\|_2^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \quad (\text{A.19})$$

In Section A.4 we have seen that the regularized minimization problem corresponds to a MAP estimate with an appropriate choice for negative log-likelihood and prior distribution (on \mathbf{w}) which correspond to loss function and regularization penalty, respectively. So, considering $J(\mathbf{y}, X, \mathbf{w})$ as a negative log-posterior and factoring out λ we can write

$$\exp \left[-J(\mathbf{y}, X, \mathbf{w}) \right] \propto \exp \left[-\frac{1}{2n\lambda} \|\mathbf{y} - X\mathbf{w}\|_2^2 \right] \cdot \exp \left[-\frac{1}{2} \|\mathbf{w}\|_2^2 \right]$$

which can be seen as

$$p(\mathbf{w}|\mathbf{y}, X) = p(\mathbf{y}|X, \mathbf{w}) \cdot p(\mathbf{w})$$

where $p(\mathbf{y}|X, \mathbf{w}) = \mathcal{N}(X\mathbf{w}, n\lambda I)$ and $p(\mathbf{w}) = \mathcal{N}(0, I)$. So, in this probabilistic interpretation, the variance of the noise affecting the output \mathbf{y} plays the role of the regularization parameter $\lambda \approx \sigma_n^2$.

A.4.2 Logistic regression revisited

In this section we revisit binary classification via logistic regression from a probabilistic perspective.

In binary classification problems we are provided with a collection of input-output pairs $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n = (X, \mathbf{y})$, where $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \{+1, -1\}$, $\forall i = 1, \dots, n$. Once again we are looking for a model $f : \mathcal{X} \rightarrow \mathcal{Y}$ that associates each input sample with its corresponding class. For the sake of simplicity we restrict to the case of linear functions $\hat{y} = f(\mathbf{x}) = \mathbf{x}^T \hat{\mathbf{w}}$.

The main idea behind logistic regression is to use a loss function having a $[0, 1]$ range to estimate the probability that a sample \mathbf{x}_i belongs to one of the two classes. As suggested by its name, the function of choice is the logistic $\sigma(z) = [1 + \exp(-z)]^{-1}$.

In this context, we model our outputs y_i as Bernoulli random variables, which implies that

$$P(y = 1|\mathbf{x}) = \sigma(-f(\mathbf{x})) = \frac{1}{1 + \exp(-f(\mathbf{x}))}$$

and

$$P(y = -1|\mathbf{x}) = \sigma(f(\mathbf{x})) = \frac{1}{1 + \exp(f(\mathbf{x}))}$$

consequently, we can write the general form as in Equation (A.20).

$$P(y = \pm 1|\mathbf{x}) = \frac{1}{1 + \exp(-yf(\mathbf{x}))} \quad (\text{A.20})$$

Therefore, assuming a Gaussian prior on the weights $\mathbf{w} \sim \mathcal{N}(0, I)$, we can perform a MAP estimate of \hat{f} solving the problem in Equation (A.21).

$$\hat{\mathbf{w}}_{\text{LR}} = \underset{\hat{\mathbf{w}} \in \mathbb{R}^d}{\operatorname{argmax}} \prod_{i=1}^n p(\mathbf{w}|y_i, \mathbf{x}_i) = \underset{\hat{\mathbf{w}} \in \mathbb{R}^d}{\operatorname{argmax}} \prod_{i=1}^n \frac{1}{1 + \exp(-y_i \mathbf{x}_i^T \mathbf{w})} \cdot \exp\left(-\frac{1}{2} \|\mathbf{w}\|_2^2\right) \quad (\text{A.21})$$

Log-transforming Equation (A.21), and applying some elementary linear algebra, we can write Equation (A.22).

$$\hat{\mathbf{w}}_{\text{LR}} = \underset{\hat{\mathbf{w}} \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{2n\lambda} \sum_{i=1}^n \log [1 + \exp(-y_i \mathbf{x}_i^T \mathbf{w})] + \frac{\lambda}{2} \|\mathbf{w}\|_2^2 \quad (\text{A.22})$$

The minimization problem expressed in Equation (A.22) is known as Regularized Logistic Regression. It can be casted in the regularization framework of Equation (A.18) where the logistic loss function $L(f(\mathbf{x}), y) = \log [1 + \exp(-yf(\mathbf{x}))]$ is penalized by the ℓ_2 -norm.

List of Figures

3.1	Drew Conway's Data Science Venn Diagram ⁵	5
4.1	The Internet popularity over the past five years of two terms: <i>data science</i> and <i>machine learning</i> . The vertical axis represents the number of Google searches of an input term normalized with respect to its maximum (source: Google Trends).	8
4.2	An overview on the most common loss functions for regression (a) and classification (b) problems plotted against the corresponding prediction error.	11
4.3	An example of underfit (a), overfit (b) and optimal fit (c) for a nonlinear regression problem. The data are a downsampled version ($f_s = 546 \text{ Hz}$) of the first observation of gravitational waves from a binary black hole merger detected on September 14 th , 2015, 09:50:45 UTC at LIGO Hanford (WA).	12
4.4	An example of underfit (a), overfit (b) and optimal fit (c) for a nonlinear binary classification problem. Each data point is a pulsar candidate randomly sampled from the High Time Resolution Universe Survey (South) dataset. The data are standardized and their dimensionality is reduced by the t-SNE algorithm [Van der Maaten and Hinton, 2008].	13
4.5	A pictorial representation of the vector $\hat{\mathbf{w}}_{\text{OLS}}$ obtained fitting an OLS model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (<i>i.e.</i> directly proportional to the output) and a negative (<i>i.e.</i> inversely proportional) weight are represented in blue and red, respectively.	15
4.6	A pictorial representation of the vector $\hat{\mathbf{w}}_{\ell_2}$ obtained fitting a ridge regression model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (<i>i.e.</i> directly proportional to the output) and a negative (<i>i.e.</i> inversely proportional) weight are represented in blue and red, respectively.	17
4.7	A pictorial representation of the vector $\hat{\mathbf{w}}_{\ell_1}$ obtained fitting a Ridge model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (<i>i.e.</i> directly proportional to the output) and a negative (<i>i.e.</i> inversely proportional) weight are represented in blue and red, respectively.	19
4.8	Profiles of the Lasso coefficients for the aging problem as λ decreases. The vertical dashed line represents the optimal value $\hat{\lambda}_{\text{cv}}$ estimated by grid-search (5-fold) cross-validation.	19
4.9	A pictorial representation of the vector $\hat{\mathbf{w}}_{\ell_1\ell_2}$ obtained fitting a Elastic-Net model on 74 randomly selected training samples of $\mathcal{D}_{\text{aging}}$. Variables associated with positive (<i>i.e.</i> directly proportional to the output) and a negative (<i>i.e.</i> inversely proportional) weight are represented in blue and red, respectively.	21

4.10	Profiles of the Elastic-Net coefficients for the aging problem as λ decreases. The vertical dashed line represents the optimal value $\hat{\lambda}_{cv}$ estimated by grid-search (5-fold) cross-validation.	22
4.11	A comparison of the value of the ℓ_1 and ℓ_2 norms of the weights obtained by OLS, ridge, Lasso and Elastic-Net.	23
4.12	Pictorial representation of the contour lines of the square loss in a 2D regression problem with various penalties: (a) ordinary least squares (no penalty), (b) ridge regression (ℓ_2 -norm penalty), (c) the Lasso (ℓ_1 -norm penalty) and finally (d) the Elastic-Net (ℓ_1 - and ℓ_2 -norm penalties).	24
4.13	A pictorial representation of the vector $\hat{\mathbf{w}}_{SVR}$ obtained fitting a SVR model on 74 randomly selected training samples of \mathcal{D}_{aging} . Variables associated with positive (<i>i.e.</i> directly proportional to the output) and a negative (<i>i.e.</i> inversely proportional) weight are represented in blue and red, respectively.	26
4.14	Pictorial representation of the kernel trick. Panel (a) shows a 2D classification problem where the input-output relationship is nonlinear. Panel (b) shows the <i>kernel explosion</i> , <i>i.e.</i> the projection of the 2D problem in a higher (3D) dimensional space in which the two classes are linearly separable, as shown in Panel (c).	27
4.15	Two examples of decision tree applications for: (a) 2D multiclass classification problem, (b) 1D regression problem.	29
4.16	The graph structure learned from the aging problem (see Section 4.1.1.1). The maximum depth of the tree is 3 and it is chosen via grid-search cross-validation.	31
4.17	A pictorial representation of the feature importance achieved by a RF model with 500 trees growing on 4, out of 12, features.	33
4.18	The effect of the number of trees in a RF model in terms of MAE, panel (a), and EV, panel (b). As expected, larger forests do not lead to overfit, <i>i.e.</i> the training error does not reach zero.	33
4.19	A pictorial representation of the feature importance achieved by a GB model with learning rate 0.1 after 31 boosting iterations. Each tree has maximum depth of 11 and it grows on 4, out of 12, features.	34
4.20	The effect of the number of boosting iterations in a GB model (learning rate 0.1) in terms of MAE, panel (a), and EV, panel (b). As expected, increasingly large boosting iterations lead to perfect overfit, <i>i.e.</i> zero training error.	35
4.21	A pictorial representation of the structure of a MLP with two hidden layers having four and three hidden units, respectively. According to the nature of the output layer, this network topology can be adopted either for regression or binary classification problems starting from raw samples in a three-dimensional space.	36
4.22	An example of ROC analysis on the MNIST dataset [LeCun et al., 2010]. In this example a linear SVM is trained on only 1% of the dataset, where 200 additional noisy features were added to make the classification problem harder.	45
4.23	An example of Silhouette analysis where a 10 clusters k -means runs on a randomly sampled 80% of the MNIST dataset [LeCun et al., 2010].	47

5.1	Three different 2D projections of the samples of the GEO gene expression data set used in this work. Projections on the left (a), middle (b) and right (c) panes are obtained via linear PCA, Gaussian PCA and isomap, respectively. The color of each point corresponds to the actual tissue type, while the background color is automatically learned by the k -means clustering algorithm. White hexagons correspond to cluster centroids.	56
5.2	An example of hierarchical trees visualization learned by two ADENINE pipelines on mucosa (a) and CRC (b) samples. Each probe set is color coded according to the corresponding sublist. This visualization provides insights on the underlying structure of the measured gene expression level.	57
7.1	An overview of the <i>PCO</i> data set used in this study. The left panel (a) shows a bar chart of the number of MS patients in each disease form at different examinations. The right panel (b) presents a representation of the distribution of the total amount of acquisitions (3137), divided according to the disease form. {break figure in two}	61
7.2	A visual representation of the temporal structure assumed in the collected data. When the two functions f (CCA) and g (PEP) are learned, the FCA model $f \circ g$ is able to predict the evolution of the disease course for future time points y_i^{t+1}	62
7.3	A visual representation of the results obtained from the CCA model. On the left panel (a) we show the classification performance achieved on the test set by the candidate models. Precision, recall and F_1 score are estimated considering <i>SP</i> as the positive class. As GB outperforms the other methods on each performance metric, it is chosen as CCA model. On the right panel (b) a heatmap displays the distance between the lists of variables selected by each model in terms of their hamming distance. {break figure in two}	65
8.1	An example of two glycemic profiles obtained from T1D and T2D patients. The glucose target range is set between 70 mg/dL and 140 mg/dL (dashed lines). The yellow area at the left hand side of the plot is the initial <i>burn-in</i> interval used for model <i>personalization</i>	69
8.2	Plots reporting the distributions of LBGI (left) and HBGI (right) for T1D and T2D. Green areas at the bottom of each plot represents low risk of hypo/hyperglycemia events.	70
8.3	Online time-series forecasting obtained by KRR model. <i>One-step-ahead prediction</i> (left): the green solid line shows the available samples; the dashed line represents one-step-ahead predictions that are obtained by applying the model on a moving window of 36 time-points. <i>Open-loop forecast</i> (right): with passing time, the moving-window incorporates an increasing number of predicted points, accumulating errors; the dashed line represents forecast with a prediction horizon of 90'. Absolute prediction errors are evaluated with respect to future measures (red solid line).	72

List of Tables

4.1	Definition of the loss functions for regression (top) and classification (bottom) problems represented in Figure 4.2.	11
4.2	Popular kernel functions. RBF stands for Radial Basis Function.	28
4.3	Overview of the matrix norms used for multiple-output regression.	29
4.4	A prototypical confusion matrix for a binary classification problem. The two classes are encoded as ± 1 , the number of true positive examples is n_+ , whereas the number of true negative examples is n_- ; hats denote estimated values. The extension for multiclass problems is straightforward, once a positive class is defined.	43
5.1	Pipeline building blocks currently available in ADENINE.	51
7.1	The set of available PCOs. The first 6 are self-reported, while the last 5 are administered by trained medical staff. In our analysis all PCOs were used, with the exception of <i>EDSS</i>	60
8.1	Prediction errors (std) of the forecasting models for increasing prediction horizons overall and on the four groups. <i>MP</i> = Microinfusion Pump, <i>II</i> = Insulin Injection, <i>RAI</i> = Rapid Acting Insulin, <i>Other</i> = Other therapies. Bold numbers indicate best result.	73

List of Abbreviations

AIC Akaike Information Criteria

AMI Adjusted Mutual Information

API Application Program Interface

ARI Adjusted Rand Index

AUC Area Under the Curve

BIC Bayesian Information Criteria

CART Classification And Regression Trees

CE Cross-Entropy

CNN Convolutional Neural Network

CT Computerized Tomography

DNA DeoxyriboNucleic Acid

DSL Domain Specific Language

EDA Exploratory-Data Analysis

ERM Empirical Risk Minimization

EV Explained Variance

FISTA Fast Iterative Shrinkage-Thresholding Algorithm

FPR False Positive Rate

GB Gradient Boosting

GUI Graphical User Interface

GWAS Genome-wide Association Studies

i.i.d. Independent and Identically Distributed

MAE Mean Absolute Error

MAP Maximum A Posteriori

MCC Matthews correlation coefficient

MLE Maximum Likelihood Estimation

MLP Multi-Layer Perceptron

MRI Magnetic Resonance Imaging

MSE Mean Squared Error

OLS Ordinary Least Squares

OVA *One-vs-All*

OVO *One-vs-One*

PCO Patient Centered Outcomes

PET Positron Emission Tomography

PP Primary-Progressive Multiple Sclerosis

PR Progressive-Relapsing Multiple Sclerosis

R² Coefficient of determination

RF Random Forests

RNN Recurrent Neural Network

ROC Receiver Operating Curve

RR Relapsing-Remitting Multiple Sclerosis

RSS Residual Sum of Squares

SP Secondary-Progressive Multiple Sclerosis

SPECT Single-Photon Emission Computed Tomography

SVC Support Vector Classification

SVM Support Vector Machine

SVR Support Vector Regression

TPR True Positive Rate

Bibliography

- Aben, N., Vis, D. J., Michaut, M., and Wessels, L. F. (2016). Tandem: a two-stage approach to maximize interpretability of drug response models based on multiple molecular data types. *Bioinformatics*, 32(17):i413–i420. [Not cited.]
- Abraham, G., Kowalczyk, A., Zobel, J., and Inouye, M. (2013). Performance and robustness of penalized and unpenalized methods for genetic prediction of complex human disease. *Genetic Epidemiology*, 37(2):184–195. [Not cited.]
- Alexandrov, L. B., Nik-Zainal, S., Wedge, D. C., Aparicio, S. A., Behjati, S., Biankin, A. V., Bignell, G. R., Bolli, N., Borg, A., Børresen-Dale, A.-L., et al. (2013). Signatures of mutational processes in human cancer. *Nature*, 500(7463):415–421. [Not cited.]
- Altmann, A., Toloşi, L., Sander, O., and Lengauer, T. (2010). Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347. [Not cited.]
- Angermueller, C., Pärnamaa, T., Parts, L., and Stegle, O. (2016). Deep learning for computational biology. *Molecular systems biology*, 12(7):878. [Not cited.]
- Appenzeller, T. (2017). The ai revolution in science. *Science*. [Not cited.]
- Argyriou, A., Evgeniou, T., and Pontil, M. (2008). Convex multi-task feature learning. *Machine Learning*, 73(3):243–272. [Not cited.]
- Arnould, C., Vandervelde, L., Batcho, C. S., Penta, M., and Thonnard, J.-L. (2012). Can manual ability be measured with a generic abilhand scale? a cross-sectional study conducted on six diagnostic groups. *BMJ open*, 2(6):e001807. [Not cited.]
- Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics. [Not cited.]
- Aupperle, R. L., Beatty, W. W., Shelton, F. d. N., and Gontkovsky, S. T. (2002). Three screening batteries to detect cognitive impairment in multiple sclerosis. *Multiple Sclerosis*, 8(5):382–389. [Not cited.]
- Bach, F., Jenatton, R., Mairal, J., Obozinski, G., et al. (2012). Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106. [Not cited.]
- Baldassarre, L., Rosasco, L., Barla, A., and Verri, A. (2012). Multi-output learning via spectral filtering. *Machine learning*, 87(3):259–301. [Not cited.]
- Ball, G. H. and Hall, D. J. (1967). A clustering technique for summarizing multivariate data. *Systems Research and Behavioral Science*, 12(2):153–155. [Not cited.]
- Barbieri, M., Fiorini, S., Tomasi, F., and Barla, A. (2016). PALLADIO: a parallel framework for robust variable selection in high-dimensional data. *PyHPC2016 conference, IEEE proceedings*. [Not cited.]

- Barrett, T., Wilhite, S. E., Ledoux, P., Evangelista, C., Kim, I. F., Tomashevsky, M., Marshall, K. A., Phillippy, K. H., Sherman, P. M., Holko, M., et al. (2013). Ncbi geo: archive for functional genomics data sets—update. *Nucleic acids research*, 41(D1):D991–D995. [Not cited.]
- Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202. [Not cited.]
- Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* [Not cited.]
- Bergamaschi, R., Montomoli, C., Mallucci, G., Lugaresi, A., Izquierdo, G., Grand’Maison, F., Duquette, P., Shaygannejad, V., Alroughani, R., Grammond, P., et al. (2015). Bremso: A simple score to predict early the natural course of multiple sclerosis. *European journal of neurology*, 22(6):981–989. [Not cited.]
- Bishop, C. M. (2006). Pattern recognition. *Machine Learning*. [Not cited.]
- Black, N. (2013). Patient reported outcome measures could help transform healthcare. *BMJ (Clinical research ed)*, 346:f167. [Not cited.]
- Borg, I. and Groenen, P. J. (2005). *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media. [Not cited.]
- Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press. [Not cited.]
- Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32. [Not cited.]
- Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press. [Not cited.]
- Buehlmann, P. (2006). Boosting for high-dimensional linear models. *The Annals of Statistics*, pages 559–583. [Not cited.]
- Button, K. S., Ioannidis, J. P., Mokrysz, C., Nosek, B. A., Flint, J., Robinson, E. S., and Munafò, M. R. (2013). Power failure: why small sample size undermines the reliability of neuroscience. *Nature Reviews Neuroscience*, 14(5):365–376. [Not cited.]
- Candès, E. J. and Recht, B. (2009). Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717. [Not cited.]
- Cardozo, L., Staskin, D., Currie, B., Wiklund, I., Globe, D., Signori, M., Dmochowski, R., MacDiarmid, S., Nitti, V. W., and Noblett, K. (2014). Validation of a bladder symptom screening tool in women with incontinence due to overactive bladder. *International urogynecology journal*, 25(12):1655–1663. [Not cited.]
- Chen, X., He, J., Lawrence, R., and Carbonell, J. G. (2012). Adaptive multi-task sparse learning with an application to fmri study. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 212–223. SIAM. [Not cited.]
- Chen, Y., Li, Y., Narayan, R., Subramanian, A., and Xie, X. (2016). Gene expression inference with deep learning. *Bioinformatics*, 32 12:1832–9. [Not cited.]

- Chollet, F. (2018). *Deep learning with Python*. Manning Publications. [Not cited.]
- Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619. [Not cited.]
- Consortium, E. P. et al. (2004). The encode (encyclopedia of dna elements) project. *Science*, 306(5696):636–640. [Not cited.]
- Dagenais, E., Rouleau, I., Demers, M., Jobin, C., Roger, É., Chamelien, L., and Duquette, P. (2013). Value of the moca test as a screening instrument in multiple sclerosis. *The Canadian Journal of Neurological Sciences*, 40(03):410–415. [Not cited.]
- De Mol, C., De Vito, E., and Rosasco, L. (2009a). Elastic-net regularization in learning theory. *Journal of Complexity*, 25(2):201–230. [Not cited.]
- De Mol, C., Mosci, S., Traskine, M., and Verri, A. (2009b). A regularized method for selecting nested groups of relevant genes from microarray data. *Journal of Computational Biology*, 16(5):677–690. [Not cited.]
- De Souto, M. C., Jaskowiak, P. A., and Costa, I. G. (2015). Impact of missing data imputation methods on gene expression clustering and classification. *BMC bioinformatics*, 16(1):64. [Not cited.]
- Demšar, J., Curk, T., Erjavec, A., Gorup, Č., Hočevár, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., et al. (2013). Orange: data mining toolbox in python. *The Journal of Machine Learning Research*, 14(1):2349–2353. [Not cited.]
- Deng, H. and Runger, G. (2013). Gene selection with guided regularized random forest. *Pattern Recognition*, 46(12):3483–3489. [Not cited.]
- Dickens, A. M., Larkin, J. R., Griffin, J. L., Cavey, A., Matthews, L., Turner, M. R., Wilcock, G. K., Davis, B. G., Claridge, T. D., Palace, J., et al. (2014). A type 2 biomarker separates relapsing-remitting from secondary progressive multiple sclerosis. *Neurology*, 83(17):1492–1499. [Not cited.]
- Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231. [Not cited.]
- Evgeniou, T., Pontil, M., and Poggio, T. (2000). Regularization networks and support vector machines. *Advances in computational mathematics*, 13(1):1–50. [Not cited.]
- Fiorini, S., Tomasi, F., Squillario, M., and Barla, A. (2017). Adenine: a hpc-oriented tool for biological data exploration. In *Computational Intelligence methods for Bioinformatics and Biostatistics, CIBB*. [Not cited.]
- Fiorini, S., Verri, A., Tacchino, A., Ponzio, M., Bricchetto, G., and Barla, A. (2015). A machine learning pipeline for multiple sclerosis course detection from clinical scales and patient reported outcomes. In *Engineering in Medicine and Biology Society (EMBC), 2015 37th Annual International Conference of the IEEE*, pages 4443–4446. IEEE. [Not cited.]

- Flachenecker, P., Kümpfel, T., Kallmann, B., Gottschalk, M., Grauer, O., Rieckmann, P., Trenkwalder, C., and Toyka, K. (2002). Fatigue in multiple sclerosis: a comparison of different rating scales and correlation to clinical parameters. *Multiple sclerosis*, 8(6):523–526. [Not cited.]
- Franchignoni, F., Tesio, L., Ottonello, M., and Benevolo, E. (1999). Life satisfaction index: Italian version and validation of a short form1. *American journal of physical medicine & rehabilitation*, 78(6):509–515. [Not cited.]
- Frey, B. J. and Dueck, D. (2007). Clustering by passing messages between data points. *science*, 315(5814):972–976. [Not cited.]
- Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232. [Not cited.]
- Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378. [Not cited.]
- Garg, R. P., Dong, S., Shah, S. J., and Jonnalagadda, S. R. (2016). A bootstrap machine learning approach to identify rare disease patients from electronic health records. *CoRR*, abs/1609.01586. [Not cited.]
- Giovannoni, G., Butzkueven, H., Dhib-Jalbut, S., Hobart, J., Kobelt, G., Pepper, G., Sormani, M. P., Thalheim, C., Traboulsee, A., and Vollmer, T. (2016). Brain health: time matters in multiple sclerosis. *Multiple Sclerosis and Related Disorders*, 9:S5–S48. [Not cited.]
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press. [Not cited.]
- Gramfort, A., Kowalski, M., and Hämmäläinen, M. (2012). Mixed-norm estimates for the m/eeg inverse problem using accelerated gradient methods. *Physics in medicine and biology*, 57(7):1937. [Not cited.]
- Granger, C., Cotter, A., Hamilton, B., Fiedler, R., and Hens, M. (1990). Functional assessment scales: a study of persons with multiple sclerosis. *Archives of physical medicine and rehabilitation*, 71(11):870–875. [Not cited.]
- Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*. [Not cited.]
- Gui, J. and Li, H. (2005). Penalized cox regression analysis in the high-dimensional and low-sample size settings, with applications to microarray gene expression data. *Bioinformatics*, 21(13):3001–3008. [Not cited.]
- Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182. [Not cited.]
- Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422. [Not cited.]
- Halko, N., Martinsson, P.-G., and Tropp, J. A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288. [Not cited.]

- Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning*, volume 2. Springer. [Not cited.]
- Hastie, T., Tibshirani, R., and Wainwright, M. (2015). *Statistical learning with sparsity: the lasso and generalizations*. CRC Press. [Not cited.]
- He, D., Kuhn, D., and Parida, L. (2016). Novel applications of multitask learning and multiple output regression to multiple genetic trait prediction. *Bioinformatics*, 32(12):i37–i43. [Not cited.]
- Helmstaedter, M., Briggman, K. L., Turaga, S. C., Jain, V., Seung, H. S., and Denk, W. (2013). Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500(7461):168–174. [Not cited.]
- Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67. [Not cited.]
- Hoggart, C. J., Whittaker, J. C., De Iorio, M., and Balding, D. J. (2008). Simultaneous analysis of all snps in genome-wide and re-sequencing association studies. *PLoS genetics*, 4(7):e1000130. [Not cited.]
- Honarmand, K. and Feinstein, A. (2009). Validation of the hospital anxiety and depression scale for use with multiple sclerosis patients. *Multiple Sclerosis*. [Not cited.]
- Hughey, J. J. and Butte, A. J. (2015). Robust meta-analysis of gene expression using the elastic net. *Nucleic Acids Research*, 43(12):e79. [Not cited.]
- Ishwaran, H., Kogalur, U. B., Blackstone, E. H., and Lauer, M. S. (2008). Random Survival Forests. *The Annals of Applied Statistics*, 2(3):841–860. [Not cited.]
- Jack, C. R., Bernstein, M. A., Fox, N. C., Thompson, P., Alexander, G., Harvey, D., Borowski, B., Britson, P. J., L Whitwell, J., Ward, C., et al. (2008). The alzheimer’s disease neuroimaging initiative (adni): Mri methods. *Journal of magnetic resonance imaging*, 27(4):685–691. [Not cited.]
- Jacob, L., Obozinski, G., and Vert, J.-P. (2009). Group lasso with overlap and graph lasso. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML ’09*, pages 433–440, New York, NY, USA. ACM. [Not cited.]
- Jolliffe, I. (2002). *Principal component analysis*. Wiley Online Library. [Not cited.]
- Joly, A., Schnitzler, F., Geurts, P., and Wehenkel, L. (2012). L1-based compression of random forest models. In *20th European Symposium on Artificial Neural Networks*. [Not cited.]
- Jung, K., Dihazi, H., Bibi, A., Dihazi, G. H., and Beißbarth, T. (2014). Adaption of the global test idea to proteomics data with missing values. *Bioinformatics*, 30(10):1424–1430. [Not cited.]
- Kolker, E., Higdon, R., Haynes, W., Welch, D., Broomall, W., Lancet, D., Stanberry, L., and Kolker, N. (2012). Moped: model organism protein expression database. *Nucleic acids research*, 40(D1):D1093–D1099. [Not cited.]
- Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957. [Not cited.]

- Kulkarni, V. Y. and Sinha, P. K. (2012). Pruning of Random Forest classifiers: A survey and future directions. In *2012 International Conference on Data Science Engineering (ICDSE)*, pages 64–68. [Not cited.]
- Kursa, M. B. (2014). Robustness of Random Forest-based gene selection methods. *BMC Bioinformatics*, 15:8. [Not cited.]
- Kurtzke, J. F. (1983). Rating neurologic impairment in multiple sclerosis an expanded disability status scale (edss). *Neurology*, 33(11):1444–1444. [Not cited.]
- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444. [Not cited.]
- LeCun, Y., Cortes, C., and Burges, C. J. (2010). Mnist handwritten digit database. *AT&T Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2. [Not cited.]
- Lee, S., Zhu, J., and Xing, E. P. (2010). Adaptive multi-task lasso: with application to eqtl detection. In *Advances in neural information processing systems*, pages 1306–1314. [Not cited.]
- Leung, M. K., Xiong, H. Y., Lee, L. J., and Frey, B. J. (2014). Deep learning of the tissue-regulated splicing code. *Bioinformatics*, 30(12):i121–i129. [Not cited.]
- Lewis, J. M., De Sa, V. R., and Van Der Maaten, L. (2013). Divvy: fast and intuitive exploratory data analysis. *The Journal of Machine Learning Research*, 14(1):3159–3163. [Not cited.]
- Liu, S., Dissanayake, S., Patel, S., Dang, X., Mlsna, T., Chen, Y., and Wilkins, D. (2014). Learning accurate and interpretable models based on regularized random forests regression. *BMC Systems Biology*, 8(3):S5. [Not cited.]
- Lublin, F. D., Reingold, S. C., Cohen, J. A., Cutter, G. R., Sørensen, P. S., Thompson, A. J., Wolinsky, J. S., Balcer, L. J., Banwell, B., Barkhof, F., et al. (2014). Defining the clinical course of multiple sclerosis the 2013 revisions. *Neurology*, 83(3):278–286. [Not cited.]
- Lusa, L. et al. (2015). Boosting for high-dimensional two-class prediction. *BMC bioinformatics*, 16(1):300. [Not cited.]
- Ma, J., Sheridan, R. P., Liaw, A., Dahl, G. E., and Svetnik, V. (2015). Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 55(2):263–274. [Not cited.]
- Ma, S. and Huang, J. (2007). Additive risk survival model with microarray data. *BMC bioinformatics*, 8(1):192. [Not cited.]
- Mamoshina, P., Vieira, A., Putin, E., and Zhavoronkov, A. (2016). Applications of deep learning in biomedicine. *Molecular pharmaceuticals*, 13(5):1445–1454. [Not cited.]
- Marx, V. (2013). Biology: The big challenges of big data. *Nature*, 498(7453):255–260. [Not cited.]
- Masecchia, S., Coco, S., Barla, A., Verri, A., and Tonini, G. P. (2015). Genome iny model of metastatic neuroblastoma tumorigenesis by a dictionary learning algorithm. *BMC medical genomics*, 8(1):57. [Not cited.]

- Mayr, A., Binder, H., Gefeller, O., Schmid, M., et al. (2014). The evolution of boosting algorithms. *Methods of Information in Medicine*, 53(6):419–427. [Not cited.]
- McNeish, D. M. and Stapleton, L. M. (2016). The effect of small sample size on two-level model estimates: A review and illustration. *Educational Psychology Review*, 28(2):295–314. [Not cited.]
- Meier, L., Van De Geer, S., and Bühlmann, P. (2008). The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71. [Not cited.]
- Meinshausen, N. and Bühlmann, P. (2010). Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473. [Not cited.]
- Meyer-Moock, S., Feng, Y.-S., Maeurer, M., Dippel, F.-W., and Kohlmann, T. (2014). Systematic literature review and validity evaluation of the expanded disability status scale (edss) and the multiple sclerosis functional composite (msfc) in patients with multiple sclerosis. *BMC neurology*, 14(1):58. [Not cited.]
- Min, S., Lee, B., and Yoon, S. (2016). Deep learning in bioinformatics. *arXiv preprint arXiv:1603.06430*. [Not cited.]
- Molinaro, A. M., Simon, R., and Pfeiffer, R. M. (2005). Prediction error estimation: a comparison of resampling methods. *Bioinformatics*, 21(15):3301–3307. [Not cited.]
- Muhr, M. and Granitzer, M. (2009). Automatic cluster number selection using a split and merge k-means approach. In *Database and Expert Systems Application, 2009. DEXA'09. 20th International Workshop on*, pages 363–367. IEEE. [Not cited.]
- Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press. [Not cited.]
- Nelson, E. C., Eftimovska, E., Lind, C., Hager, A., Wasson, J. H., and Lindblad, S. (2015). Patient reported outcome measures in practice. *Bmj*, 350:g7818. [Not cited.]
- Nesterov, Y. (2013). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media. [Not cited.]
- Ng, A. Y., Jordan, M. I., Weiss, Y., et al. (2002). On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856. [Not cited.]
- Nowak, G., Hastie, T., Pollack, J. R., and Tibshirani, R. (2011). A fused lasso latent feature model for analyzing multi-sample acgh data. *Biostatistics*, page kxr012. [Not cited.]
- Okser, S., Pahikkala, T., Airola, A., Salakoski, T., Ripatti, S., and Aittokallio, T. (2014). Regularized machine learning in the genetic prediction of complex traits. *PLoS Genet*, 10(11):e1004754. [Not cited.]
- Oldfield, R. C. (1971). The assessment and analysis of handedness: the edinburgh inventory. *Neuropsychologia*, 9(1):97–113. [Not cited.]
- Parmenter, B., Weinstock-Guttman, B., Garg, N., Munschauer, F., and Benedict, R. H. (2007). Screening for cognitive impairment in multiple sclerosis using the symbol digit modalities test. *Multiple Sclerosis*, 13(1):52–57. [Not cited.]

- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. [Not cited.]
- Pelleg, D., Moore, A. W., et al. (2000). X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, volume 1, pages 727–734. [Not cited.]
- Peng, B., Wang, L., and Wu, Y. (2016). An error bound for l_1 -norm support vector machine coefficients in ultra-high dimension. *The Journal of Machine Learning Research*, 17(1):8279–8304. [Not cited.]
- Prechelt, L. (1998). Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer. [Not cited.]
- Qi, Y. (2012). Random forest for bioinformatics. In *Ensemble machine learning*, pages 307–323. Springer. [Not cited.]
- Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*, volume 1. MIT press Cambridge. [Not cited.]
- Ross, D. A., Lim, J., Lin, R.-S., and Yang, M.-H. (2008). Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1-3):125–141. [Not cited.]
- Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65. [Not cited.]
- Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326. [Not cited.]
- Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*. [Not cited.]
- Scalfari, A., Neuhaus, A., Daumer, M., Muraro, P. A., and Ebers, G. C. (2014). Onset of secondary progressive phase and long-term evolution of multiple sclerosis. *Journal of Neurology, Neurosurgery & Psychiatry*, 85(1):67–75. [Not cited.]
- Schmidhuber, J., Wierstra, D., and Gomez, F. (2005). Evolino: Hybrid neuroevolution/optimal linear search for sequence learning. In *Proc. of IJCAI*. [Not cited.]
- Schölkopf, B., Smola, A., and Müller, K.-R. (1997). Kernel principal component analysis. In *Artificial Neural Networks—ICANN’97*, pages 583–588. Springer. [Not cited.]
- Schulz, W. (2005). *Molecular biology of human cancers: an advanced student’s textbook*. Springer Science & Business Media. [Not cited.]
- Service, R. F. (2017). Ai is changing how we do science. get a glimpse. *Science*. [Not cited.]
- Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge university press. [Not cited.]
- Shawe-Taylor, J. and Sun, S. (2011). A review of optimization methodologies in support vector machines. *Neurocomputing*, 74(17):3609–3618. [Not cited.]

- Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905. [Not cited.]
- Smola, A. J. and Schölkopf, B. (1998). *Learning with kernels*. GMD-Forschungszentrum Informationstechnik. [Not cited.]
- Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222. [Not cited.]
- Sra, S., Nowozin, S., and Wright, S. J. (2012). *Optimization for machine learning*. Mit Press. [Not cited.]
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958. [Not cited.]
- Stekhoven, D. J. and Bühlmann, P. (2011). Missforest—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118. [Not cited.]
- Tang, Z., Shen, Y., Zhang, X., and Yi, N. (2017). The spike-and-slab lasso cox model for survival prediction and associated genes detection. *Bioinformatics*, page btx300. [Not cited.]
- Tenenbaum, J. B., De Silva, V., and Langford, J. C. (2000). A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323. [Not cited.]
- Teramoto, R. et al. (2009). Balanced gradient boosting from imbalanced data for clinical outcome prediction. *Statistical applications in genetics and molecular biology*, 8(1):1–19. [Not cited.]
- Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288. [Not cited.]
- Tibshirani, R. et al. (1997). The lasso method for variable selection in the cox model. *Statistics in medicine*, 16(4):385–395. [Not cited.]
- Tikhonov, A. (1963). Solution of incorrectly formulated problems and the regularization method. In *Soviet Math. Dokl.*, volume 5, pages 1035–1038. [Not cited.]
- Toga, A. W. and Dinov, I. D. (2015). Sharing big biomedical data. *Journal of big data*, 2(1):7. [Not cited.]
- Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B. (2001). Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525. [Not cited.]
- Tutz, G. and Binder, H. (2006). Generalized additive modeling with implicit variable selection by likelihood-based boosting. *Biometrics*, 62(4):961–971. [Not cited.]
- Tutz, G. and Binder, H. (2007). Boosting ridge regression. *Computational Statistics & Data Analysis*, 51(12):6044–6059. [Not cited.]
- Uitdehaag, B. (2014). Clinical outcome measures in multiple sclerosis. *Handb Clin Neurol*, 122:393–404. [Not cited.]

- Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85. [Not cited.]
- Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media. [Not cited.]
- Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [Not cited.]
- Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416. [Not cited.]
- Vukusic, S. and Confavreux, C. (2003). Prognostic factors for progression of disability in the secondary progressive phase of multiple sclerosis. *Journal of the neurological sciences*, 206(2):135–137. [Not cited.]
- Waldmann, P., Mészáros, G., Gredler, B., Fuerst, C., and Sölkner, J. (2013). Evaluation of the lasso and the elastic net in genome-wide association studies. *Frontiers in genetics*, 4:270. [Not cited.]
- Witten, D. M. and Tibshirani, R. (2009). Covariance-regularized regression and classification for high dimensional problems. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):615–636. [Not cited.]
- Wu, T. T., Chen, Y. F., Hastie, T., Sobel, E., and Lange, K. (2009). Genome-wide association analysis by lasso penalized logistic regression. *Bioinformatics*, 25(6):714–721. [Not cited.]
- Wu, T. T. and Lange, K. (2008). Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, pages 224–244. [Not cited.]
- Yu, D., Huber, W., and Vitek, O. (2013). Shrinkage estimation of dispersion in negative binomial models for rna-seq experiments with small sample size. *Bioinformatics*, 29(10):1275–1282. [Not cited.]
- Zhou, Z.-H. (2012). *Ensemble methods: foundations and algorithms*. CRC press. [Not cited.]
- Zhu, J., Rosset, S., Tibshirani, R., and Hastie, T. J. (2004). 1-norm support vector machines. In *Advances in neural information processing systems*, pages 49–56. [Not cited.]
- Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429. [Not cited.]
- Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320. [Not cited.]