Internship report at Mazars

Final year internship

2016 - 2017

Different Machine Learning Techniques in Finance

September 12, 2017

Author:
Riyaz MOUHAMAD

Supervisor:
Mathieu Schneider

Acknowledgements

As the saying goes, good premises do not entail good stories. Yet, this report would certainly not have come to its successful end without the help, support and trust of colleagues, friend and family. First of all, I would sincerely thank my supervisor Mathieu Schneider for his countless support, guidance. He helped me to re-discover the world of data science. Thanks to him for his advice on content and design and for making this challenging endeavor an effective and fruitful work.

I am grateful to my senior manager *Xavier Larrieu* for the freedom and the trust I was granted throughout this internship. He contributed to create and maintain a pleasant and stimulating working environment for the quantitative team. I would like to thank *Grégoire Thiercelin* for the innumerable conversations that we had about Finance and bodybuilding. I came to know about so many new things, I am truly grateful to them.

I would also like to thank all the professors from the M2MO for providing us a high-quality education and especially Annie Millet for having put her trust in me. I cannot express enough thanks to my professor from my engineering school for their continued support and encouragement: Hackim Boumaza, Ahmed Kebaier, Laurent Tournier, Emmanuel Audusse and all the great and very supporting professors of Sup Galilée.

Thanks to my parents. The countless time you helped throughout my journey at school. Your encouragement when the times got rough and provided everything that I have needed even though I know you have struggled a lot.

Last but not least, I would not be in this position without the trust of my director *Olivier Lafitte*, I am forever grateful for his unconditional support, valuable advice, and knowledge. He always makes sure that we, students of the "MACS", reach our goal. I hope in coming future, we will continue our endless conversation.

When you know a thing, to hold that you know it; and when you do not know a thing, to allow that you do not know it—this is knowledge. - Confucius, The Analects

Presentation of the Internship

The internship has taken place at **Mazars** UK, an independent audit organisation in the United Kingdom. The company has over 1700 employees and around 140 partners. It also specialized in consulting on financial services (FS consulting). I was working in the quantitative finance team within FS consulting. The quantitative team is composed of actuaries, quantitative analysts and statisticians in charge of instrument valuations, risk management, internal model assessment.

During the internship, I was charged to test different machine learning algorithms on a dataset. More generally, I was expecting to work on different matters related to data science. The senior manager Xavier Larrieu intend to build a data science team within the quantitative finance team in order to develop an expertise on this matter. Many subjects related to data science are on going. Thus in order to achieve these requirements, I was asked to have a deep knowledge on the different Machine Learning techniques. Hence Mathieu Schneider, the supervisor of the internship, proposed to split the internship into two phases:

- In the first half, an extensive knowledge of the various algorithms will be acquired. This step intends to have both a mathematical and a practical point of view of each algorithm. Furthermore, I was asked to concentrate my work first on basic techniques and after on more advanced ones as deep learning algorithms.

- In the second half, we needed to put all these algorithms into actions. He proposed to use a dataset related to direct bank marketing. This dataset comes with an article where the authors had some result. The end of the work is to have a better performance than the one in the paper.

Nevertheless, I want to highlight the fact that I was led to work also on different matters other than Machine Learning. In fact, I ought to work on valuation services as the pricing of financial instruments or risk assessment and management.

Finally, this internship was only a premise of a bigger project. In fact after this experience, I was fully operational to work in full time with a French investment bank on V@R shocks detection.

Contents

Introduction

Nowadays, most records and observations are captured electronically from devices connected to the internet. This, in principle, allows investors to access a new source of a competitive dataset with the available alternative data sources as well as the application of new quantitative techniques of Machine Learning to analyse these data. However, new datasets are often bigger in variability and volume compared to an old dataset as stock prices for instance. Alternative datasets could be found as data generated by people (social media, search trends, product review, ...) or data generated by sensors (satellite image data, ship locations, ...). In most of the case, we need to analyse these data before we used them in trading strategies for example. Even with a large traditional dataset, simple methods as linear regression often lead to over-fitting or inconsistent results. That is where Machine Learning comes in.

In last decade, there has been a massive development in the field of pattern recognition (uncovering relationship between variables). These methods are known Machine Learning techniques, and they are part of a more broader family in Computer Science and Statistics Machine Learning techniques enables analysis of large and unstructured datasets. To illustrate this consider a self-driving car which learns for being initially driven by a human driver; further, as it drives itself, in reinforces its learning and improves it with experience. In finance, one can view Machine Learning as a way to uncover relationships between variables, where given historical data, the algorithms forecast outcomes out of sample. In the study of Machine Learning, we find three principal branches: Supervised Learning, Unsupervised Learning, reinforcement learning. There been a huge development in deep learning. The year 2016 saw a multitude application of deep learning in real life: Google Home, Apple Siri, Samsung Bixby, Amazon Echo which relied mostly on the deep learning algorithms. The reader should notice there is a lot of hype around Machine Learning, researchers estimate that only 0.5% of the data available is being analyzed in [6]. Moreover, in implementing Machine Learning in Finance, it is more important to understand the data and the signals, *i.e.* the economics behind the data, than to be able to develop sophisticated algorithms. We have too many concepts in Machine Learning may sound plausible but will not lead to viable strategies in Finance. Moreover, the provided framework in this report is not a "ready-to-use" method for a given dataset.

One point is clear: techniques of Machine Learning yielded some spectacular results when applied to problems of pattern recognition, automation of complex tasks (driving a car), image processing and natural language processing. What is the application of Machine Learning in Finance, and how do these methods differ from each other.

In the first chapter of this report, we provide some basic definition of Machine Learning. In the second chapter, we provide an initial framework to understand some Machine Learning techniques. Finally, in the third chapter we try to apply these methods to a given dataset and try to compare the efficiency

Chapter 1

Different Machine Learning algorithms

Machine Learning is the study of methods for building computer programs that automatically improve and adapt their performance through experience.

The goal of machine learning is to discover methods by which the machine will provide its own model based on examples. Dietterich [3] mentioned four categories situations in which it is not easy for software engineers to design the software for solving a problem. In particular, there are problems where phenomena are changing rapidly, and we want programs whose adapt their behaviour in this environment. In finance, for example, people would like to predict the future evolution of the stock market. These practices change very frequently so that if a programmer could write a program which has a good prediction power, it has to be rewritten at every change of the market. A learning program can relieve the programmer of multiple tasks such as constantly modifying and calibrating a set of learned prediction rules. In this chapter, we will discuss the different classes of machine learning algorithms that are used.

Terminology Before getting into the core of a learning system, we need first to define the basic notions. An *example* (also called an *instance*) is the data used by an algorithm to learn how to classify. For example, if we consider the credit card frauds detection, then clients are examples. An example is represented by a group of attributes known as *variables* or *features*. The category that we are trying to forecast is the *label*. The latter could be either categorical or numerical. *Classification* aims to replicate decision making in order to develop such predictive analytics. Classifications algorithms work well for problems with well-defined boundaries. Basically, the classification process is to learn past experiences and use this to evaluate new inputs by matching them with previously observed patterns. This process is suitable for categorical outputs. *Regression* is one of the most important and widely used machine learning and statistics tools available nowadays. It allows to forecast from data by discovering the relationship between variables and a real valued response. This method is used in a massive number of fields ranging from predicting stock prices to understanding gene regulatory networks. We can classify the different type of learning. Some common situations are:

- *Supervised Learning*: Given the desired outputs, the machine's goal is to learn to produce the correct output given a new input. In supervised learning, the response variable is available which provides the desired action corresponding to the data. Example: to identify fraudulent or none authorized bahaviour quickly.

- *Unsupervised learning*: The purpose of the machine is to build a model of input that can be used for reasoning, forecasting things, decision making. In unsupervised learning no teacher is available. The machine discovers only patterns in the data using a collection of examples. This is called exploratory learning. Example: To highlight any relevant relationship between the operating and financial history of a company and its performance on the stock market.

- *Active learning*: Here not only a teacher is available, the program has the freedom to ask the teacher for suitable perception-action example pairs which will help the program to improve its performance. Consider a news recommender system which tries to learn a user's preference and

categorize news articles as interesting or uninteresting to the user. The system may present a particular article (of which it is not sure) to the user and ask whether it is interesting or not.

- *Reinforcement learning*: The machine can also produce actions which affect the state of the world, and receive rewards (or punishment). The goal is to learn to act in a way that maximizes rewards in the long term. In reinforcement learning, the computer could interact with the user. Instead of directly providing the desired action corresponding to a perception, he returns reward and punished to the program for its action corresponding to an example.

Chapter 2

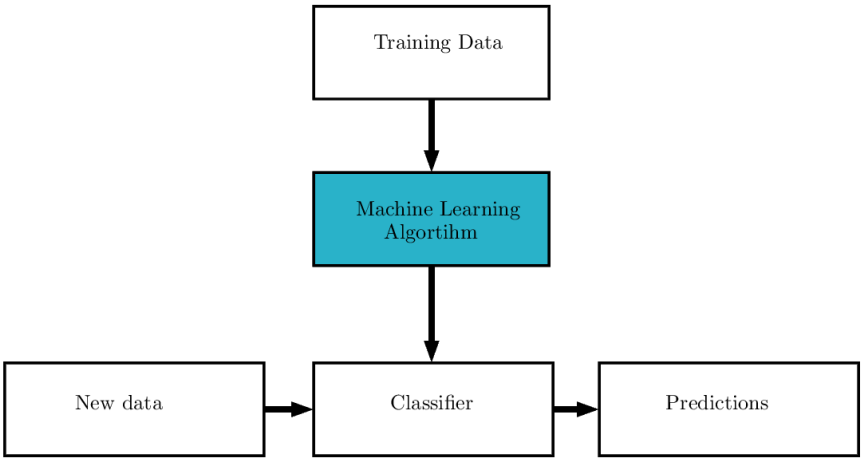
Modelling

Contents

(Bibliographic sources.: I. Goodfellow and Y. Bengio and A. Courville [4]; I. H. Witten, E. Frank, M. A. Hall and C.J. Pall, [5])

To establish notations for future use, we define $\mathbf{x}_i$ as input variables, also called input **features** or **design matrix**, typically $\mathbf{x}_i = (x_{i1}, \ldots, x_{im})^t \in \mathbb{R}^m$ and y_i as an output variable known as **target** variable that we are trying to predict. The vector $\mathbf{x}_i$ could be either real or categorical vector. A pair $(\mathbf{x}_i, y_i)$ is called a **training example** and denotes the ensemble of the training set by $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ We will also use $\mathcal{X}$ which is the space of input values, and $\mathcal{Y}$ the space of output values. Let X be a $n \times m$ matrix and $Y = (y_1, \ldots, y_n)^t$. Given our data, our class of learning is supervised learning. Basically, supervised learning is what statisticians do almost all the time. The term "supervised" refers to the fact that $y_i's$ are available, in contrast to unsupervised learning. Supervised learning more typically refers to a topic that is less familiar but is the focus of this chapter: classification.

The task is to learn a function $h : \mathcal{X} \mapsto \mathcal{Y}$ so that $h(\mathbf{x}_i)$ is a "good" predictor for the related value of y_i. The function h is called the **hypothesis** function for historical reasons. The picture below sums up our idea:



When the response variable is continuous, such as the stock market for example, we call the learning problem a **regression** problem. When y can take discrete values, typically $y \in \mathbb{N}$, we call it a **classification** problem. For $i \in 1, \ldots, n$, we want to explain $\mathbf{x}_i$ from y_i. For this purpose, we introduce a new family of models called: **parametric** models. They suppose the hypotheses h has a particular form, unlike **non-parametric** models. The following table resuming our thoughts of this section:

Statistics	Computer Science	Meaning
classification	supervised learning	predicting Y from X
data	training sample	$(X_1, Y_1), \ldots, (X_n, Y_n)$
covariates	features	the design matrix X
classifier	hypothesis	a function h which map: $\mathcal{X} \to \mathcal{Y}$
estimation	learning	finding a good classifier

2.1 Generalized Linear Model

2.1.1 Linear Regression

To perform supervised learning, we must decide how we represent the hypotheses h. As the natural choice, we decided to estimate Y as a linear function of X:

$$i \in 1, \ldots, n \; h_\theta(\mathbf{x}_i) = \sum_{j=1}^{m} \theta_j x_{ij} = \theta^t \mathbf{x}_i$$

where $\theta \in \mathbb{R}^n$.

Here, θ are the **weights**, also called **parameter**. They are parametrizing the space of linear functions mapping from $\mathcal{X}$ to $\mathcal{Y}$. Given a training set, we want to learn the parameters θ. A natural choice is to choose $h(\mathbf{x}_i)$ close y_i. For this goal, we have to take a measure which is going to quantify the error we make and help us to find θ. For example, we can take the l_2 norm and define the **cost** function known as the **residual sums of squares**:

$$\forall \mathbf{x}_i \in \mathbb{R}^n, \; y_i \in \mathbb{R}, \; J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (h_\theta(\mathbf{x}_i) - y_i)^2$$

Minimizing this function using the training set, we should be able to find optimal θ. Note that the optimization problem, we have posed here, presents one global optimum. Indeed, J is a convex quadratic function.

2.1.2 Probabilistic interpretation

Let us suppose that the response variables and the inputs are linked via the following equation:

$$y_i = h(\mathbf{x}_i) + \epsilon_i \; i \in \{1, \dots, n\}$$

where ϵ is distributed $i.i.d$ according to a Gaussian distribution: $\epsilon \sim \mathcal{N}(0_n, \sigma^2.I_n)$ This implies that:

$$h_\theta(x) = \mathbb{E}[Y|\theta; X = x] = \int y f(y|\theta; x) dy$$

We can thus write the linear regression model as follows

Definition 1. *The simple linear regression model*

$$y_i = \theta^t \mathbf{x}_i + \epsilon_i$$

where $\mathbb{E}[\epsilon_i|\mathbf{x}_i] = 0$ *and* $\mathrm{Var}[\epsilon_i|\mathbf{x}_i] = \sigma^2$

Furthermore, the distribution of y_i can also write as $y_i|\mathbf{x}_i; \theta \sim \mathcal{N}(\theta^t \mathbf{x}_i, \sigma^2)$. In order to estimate the parameters θ regarding $\mathbf{x}_i$, one could use the **likelihood** function.

$$L(\theta) = L(\theta; X, Y) = \mathbb{P}(Y|X; \theta)$$

We recall that a good choice of parameters is a choice which maximizes the probability in the training set.

Note that by the independence assumption of ϵ, we can rewrite as follows:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{n} \mathbb{P}(y_i|\mathbf{x}_i; \theta) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \theta^t \mathbf{x}_i)^2}{2\sigma^2}\right) \end{aligned}$$

Using the **maximum likelihood** estimators (MLE), one could find an estimation of the parameters:

$$\sup_{\theta \in \mathbb{R}^n} L(\theta)$$

Instead of maximizing $L(\theta)$, we could also maximize any strictly increasing function of $L(\theta)$. In particular, we maximize the **log-likelihood** to make it a little bit easier:

$$\begin{aligned} l(\theta) &= \log\left(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \theta^t \mathbf{x}_i)^2}{2\sigma^2}\right)\right) \\ &= \sum_{i=1}^{n} -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y_i - \theta^t \mathbf{x}_i)^2 \\ &= -\frac{1}{2}\log(2\pi\sigma^2)n - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \theta^t \mathbf{x}_i)^2 \end{aligned}$$

Consequently,

$$\sup_{\theta} \; l(\theta) = \inf_{\theta} \; \frac{1}{2}\sum_{i=1}^{n}(y_i - \theta^t \mathbf{x}_i)^2$$

Thus, maximizing $l(\theta)$ leads back to minimizing $J(\theta)$.

2.1.3 Logistic Regression

So far we have assumed that y_i is real valued. **Logistic regression** is a parametric method for regression when $y_i \in \{0, 1\}$. Intuitively, it makes no sense for $h_\theta(\mathbf{x}_i)$ to take values larger than 1 or smaller than 0 when the target variable y is between $\{0, 1\}$. For this purpose, we choose our hypotheses $h_\theta(\mathbf{x}_i)$ as follows:

$$\forall i \in \{0, \dots, 1\} \quad h_\theta(\mathbf{x}_i) = \frac{1}{1 + \exp\left(-\theta^t \mathbf{x}_i\right)}$$

The function $x \mapsto \frac{1}{1+\mathrm{e}^x}$ is called the **logistic function** or the **sigmoid function**. Others functions could be taken for the hypothesis function. We have supposed for the linear regression that y_i follows a Gaussian distribution. Now y_i are binary, this model asserts that y_i are $i.i.d$ Bernoulli random variables:

$$\begin{aligned}
\mathbb{P}(y_i = 1 | \mathbf{x}_i; \theta) &= h_\theta(\mathbf{x}_i) \\
\mathbb{P}(y_i = 0 | \mathbf{x}_i; \theta) &= 1 - h_\theta(\mathbf{x}_i)
\end{aligned}$$

Rewriting this, we obtain:

$$\mathbb{P}(y_i | \mathbf{x}_i; \theta) = (h_\theta(\mathbf{x}_i))^{y_i} (1 - h_\theta(\mathbf{x}_i))^{1-y_i}$$

As we did in the previous section, we compute the likelihood:

$$\begin{aligned}
L(\theta) &= \prod_{i=1}^{n} \mathbb{P}(y_i | \mathbf{x}_i; \theta) \\
&= \prod_{i=1}^{n} (h_\theta(\mathbf{x}_i))^{y_i} (1 - h_\theta(\mathbf{x}_i))^{1-y_i}
\end{aligned}$$

As before, we take the logarithm of the likelihood in order to simply:

$$\begin{aligned}
l(\theta) &= \log(L(\theta)) \\
&= \sum_{i=1}^{n} y_i \log(h(\mathbf{x}_i)) + (1 - y_i) \log((1 - h_\theta)(\mathbf{x}_i)
\end{aligned}$$

The maximum likelihood estimator $\widehat{\theta} = (\widehat{\theta}_1, \dots, \widehat{\theta}_m)^t$ cannot be found in a closed form. Thus, we have to perform an iterative algorithm.

2.2 Model

For a better approach to the exponential family, see Antoniadis et al. (1992).

We have seen the simple regression and also the logistic regression. In the regression, we had $Y|X; \theta \sim \mathcal{N}(\mu, \sigma^2)$ and for the logistic regression $Y|X; \theta \sim \mathcal{B}(\phi)$ for some definitions of μ and ϕ as the function of X and θ. These methods are special cases of a broader family of models, called Generalized Linear Models (GLMs). Suppose now that an insurance company would like to build a model to estimate the number of car accidents, we know that the Poisson distribution usually gives a good model for this problem.

The Exponential Family We assume that the observations belong to the exponential family with probability density function:

$$f(y_i) = \exp\left(\frac{y_i \theta_i - b(\theta_i)}{a_i(\phi)} + c(y_i, \phi)\right) \tag{2.1}$$

where θ_i and ϕ are the parameters and $a_i(\phi)$, $b(\theta_i)$ and $c(y_i, \phi)$ are known functions. For some distribution, the function a_i will be:

$$a_i(\phi) = \frac{\phi}{\omega_i}$$

where the weights ω_i are known from the observations; ϕ is known as the *dispersion parameter*. We can also write the *canonical form* of the exponential family:

$$f(y_i;\eta) = b(y_i)\exp(\eta^T(y_i) - a(\eta))$$

Here, η is named the **canonical parameter** of the distribution; $T(y_i)$ is the **sufficient statistic** and $a(\eta)$ is the **logpartition function**. The quantity $\exp(-a(\eta))$ basically plays the role of normalization constant.

So we have all the tools necessary to introduce the GLM. For this model we will make the three assumptions to predict the value of some about the conditional distribution of y_i given $\mathbf{x}_i$ and about our model:

1. $Y|X;\theta \sim$ ExponentialFamily(η) and the x_i's are $i.i.d$
2. Our goal is to predict $\mathbb{E}[T(y)|x]$ knowing x. This means we want the prediction $h(\mathbf{x}_i)$ output by our learned hypothesis h to satisfy $h(\mathbf{x}_i) = \mathbb{E}[y_i|\mathbf{x}_i]$
3. The parameter η and the inputs $\mathbf{x}_i$ are linked linearly: $\eta = \theta^t\mathbf{x}_i$.

When the hypothesis function makes the linear predictor η as the canonical parameter θ_i, it is named the *canonical link*. The canonical link for the normal distribution is the identity function. One can remark that the canonical link for the Poisson distribution is the log function. The canonical link for the binomial distribution is the logistic function. This leads to some natural pairings:

Error	Link function
Normal	Identity
Poisson	Log
Binomial	Logistic

2.2.1 Gradient Descent

This section presents the gradient descent algorithm. We want to choose θ so as to minimize the log-likelihood $l(\theta)$. TO do so, let us use a search that starts with some "initial guess" for θ, and that changes over and over to θ to make $l(\theta)$ smaller, until we converge to a value of θ to that minimizes $l(\theta)$. The update rules write as follows:

$$\theta_{j+1} := \theta_j - \alpha\frac{\partial J(\theta)}{\partial\theta_j}$$

This method is going through every example in the whole training set on every step and it is known as **batch gradient descent**. We remark the gradient descent will converge to a local minimum in general unless the optimization problem has a global optimal.

2.3 Bayes Classifier

Our goal is to find a classification rule h that makes accurate predictions. For this purpose, let us introduce some definitions.

2.3.1 Model

Definition 2. *The **error rate** of a classifier h is*

$$L(h) = \mathbb{P}(h(X) \neq Y)$$

*and the **empirical error rate** is*

$$\widehat{L}_n(h) = \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\{h(\mathbf{x}_i)\neq y_i\}}$$

We consider that $\mathcal{Y} = \{0, 1\}$. Let

$$r(x) = \mathbb{E}(Y|X = x) = \mathbb{P}(Y = 1|X = x)$$

be the **regression** function. Thanks to Bayes' theorem we have

$$\begin{aligned} r(x) &= \mathbb{P}(Y = 1|X = x) \\ &= \frac{\mathbb{P}(X|Y = 1)\mathbb{P}(Y = 1)}{\mathbb{P}(X|Y = 1)\mathbb{P}(Y = 1) + \mathbb{P}(X|Y = 0)\mathbb{P}(Y = 0)} \end{aligned} \tag{2.2}$$

Definition 3. *The **Bayes classification rule** h^* is*

$$h^*(x) = \begin{cases} 1 \text{ if } r(x) > \frac{1}{2} \\ 0 \text{ otherwise} \end{cases}$$

*The set $\mathcal{D}(h) = \{x : \ \mathbb{P}(Y = 1|X = x) = \mathbb{P}(Y = 0|X = x)\}$ is called the **decision boundary**.*

We have another interesting result with Bayes classifiers:

Theorem 2.3.1. *The Bayes rule is optimal, that is, if h is any other classification rule then $L(h^*) \leq L(h)$.*

The Bayes classifiers depend on many unknown quantities. We have to use the learning set in order to estimate them. We present here three main approaches:

1. *Emperical Risk Minimization*: Choose a set of classifiers in $\mathcal{H}$ and find the optimal $h^* \in \mathcal{H}$ which minimizes the loss $L(h)$.

2. *Regression* Find an estimation of the regression function $\widehat{r}$ and set:

$$h^*(x) = \begin{cases} 1 \text{ if } \widehat{r}(x) > \frac{1}{2} \\ 0 \text{ otherwise} \end{cases}$$

3. *Density estimation* One could find an estimation of (2.2). First, estimate $\mathbb{P}(X|Y = 0)$ from $\mathbf{x}_i$'s for which $y_i = 0$, estimate $\mathbb{P}(X|Y = 1)$ from $\mathbf{x}_i$'s for which $y_i = 1$. Second, set $\mathbb{P}(Y = k) = \frac{1}{n}\sum_{i=0}^{n} \mathbf{1}_{y_i=k}$ for $k = \{0, 1\}$. Given these estimations, one could compute easily:

$$\widehat{h}(x) = \begin{cases} 1 \text{ if } \widehat{r}(x) > \frac{1}{2} \\ 0 \text{ otherwise} \end{cases}$$

We can also generalize the Bayes classifiers when $\mathcal{Y} = \{0, \ldots, K\}$ for $K \in \mathbb{N}$:

Theorem 2.3.2. *Suppose that $\mathcal{Y} = \{0, \ldots, K\}$. The optimal rule is*

$$\begin{aligned} h(x) &= \max_k \ \mathbb{P}(Y = k|X = x) \\ &= \max_k \ \pi_k f_k(x) \end{aligned}$$

where $\mathbb{P}(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum r \pi_r f_r(x)}$ with $\pi_k = \mathbb{P}(Y = k)$, $f_k(x) = f(X = x|Y = k)$.

2.3.2 Linear and Gaussian classifiers

The first approach to classification is too use the density estimation method. Suppose that $\mathcal{Y} = \{0, 1\}$ and that $f_0 = f(x|Y = 0)$ and $f_1 = f(x|Y = 1)$ are both multivariate Gaussian:

$$f_k(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k)\right\}, k = 0, 1$$

Thus, $X|Y = 0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ and $X|Y = 1 \sim \mathcal{N}(\mu_1, \Sigma_1)$.

Theorem 2.3.3. *If $X|Y = 0 \sim \mathcal{N}(\mu_0, \Sigma_0)$ and $X|Y = 1 \sim \mathcal{N}(\mu_1, \Sigma_1)$, the optimal Bayes rule is*

$$h^*(x) = \begin{cases} 0 \text{ if } r_1^2 < r_0^2 + 2\log(\frac{\pi_1}{\pi_0}) + \log(\frac{|\Sigma_0|}{|\Sigma_1|}) \\ 1 \text{ otherwise} \end{cases}$$

where

$$r_i^2 = (x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i), i = 1, 2$$

*is the **Manalabhobis distance**. A different way of expressing the Bayes' rule is*

$$h^*(x) = \max_k \delta_k(x)$$

where

$$\delta_k(x) = -\frac{1}{2}\log(|\Sigma_k|) - \frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + \log(\pi_k)$$

In practice, μ_k, Σ_k and π_k are estimated using the training set in place of the true value. The decision boundary of the above classifiers is quadratic. Hence, this procedure is called **quadratic discriminant analysis (QDA)**. Setting $\Sigma_0 = \Sigma_1 = \Sigma$, a simplification occurs and the decision boundary is a linear so this procedure is called **linear discrimination analysis (LDA)**.

2.4 Tree-Based methods

We saw in the previous section two classifications methods which assumed a parametric structure on the available data. Now, we will see some non-parametric structure methods. Trees are classification methods that partition the space of input values $\mathcal{X}$ into disjoint pieces and then classify the observation $\mathbf{x}_i$ inot the partition element each of which fall in. As the name implies, the classifier can be represented as a tree.

2.4.1 Decision Tree

This section introduces widely used in classification technique, the **decision tree** classifier. A decision tree algorithm try to model the data using a tree representation where each internal node corresponds to a feature and every terminal node corresponds to a class. The tree has three types of nodes:

- A **root node** that have no incoming edges and zero or more outgoing edges.
- **Internal nodes** which have exactly one incoming edge and two or more outgoing edges.
- **Leaf** or **terminal** nodes which has exactly one incoming edge and no outgoing edges.

In this method, each leaf is assigned a class. The non-terminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate instances that have different characteristics

The power of this algorithm is that once the tree is built, classifying an unseen example is straightforward. From the root node, we apply the condition to the example and follows the appropriate branch based on the outcome of the test. Indeed we apply the test to the example $\mathbf{x}_i$ and then using the outcome of the test, we choose the suitted branch. According the outcome of the test, we will go to another internal node for which a new test is applied or directly to a leaf node.

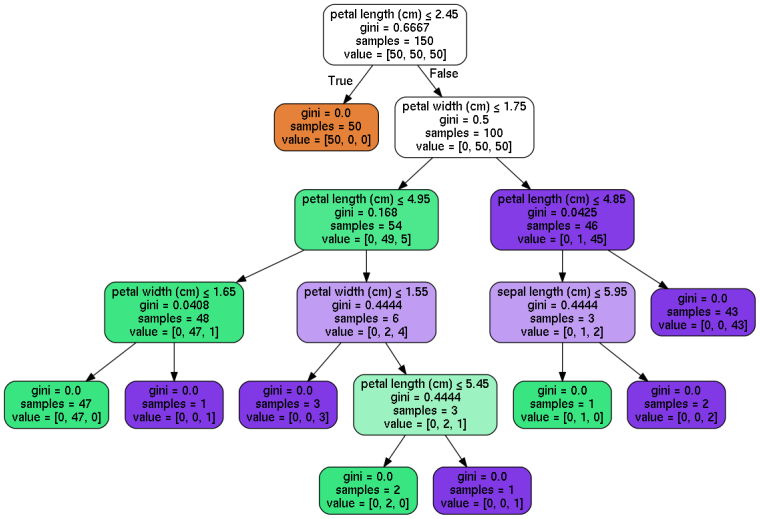


Figure 2.1: Example: Decision tree for iris dataset. This dataset is multiclass dataset presented by Ronal Fisher in 1936. The purpose of this data is to quantify the morphologic variation of Iris flowers of three related species.

Building a Decision Tree

Decision trees are constructed by analyzing a set of training examples for which the class labels are known. Afterwards, they can classify previously unseen instances. In principle, there is no unique decision tree for a given data set. In fact, we could find a large range of decision tree which fit the data set. However some of the trees are more accurate than others, the problem is then to find the optimal tree. But it is computationally infeasible because of the exponential size of the search space. However, powerful algorithms have been create to build a accurate decision tree in a reasonable amount of time. The main idea is to use a greedy strategy that grows a decision tree by making a series of locally optimum decisions about which attribute to use for splitting the data. One of these algorithms is **Hunt's algorithm**, which uses the basis of different existing decision tree algorithms (CART, ID3 and C4.5).

Hunt's Algorithm In Hunt's algorithm, a decision is grown in a recursive way by partitioning the training set into homogeneous subsets. Let D_t be the training set that are joint with the node t and $\mathcal{Y} = \{y_1, y_2, \ldots, y_n\}$ be class labels. The Hunt's algorithm is the following:

Step 1: If all the examples in D_t belong to the same class y_t, then t is a leaf node labelled as y_t.

Step 2: If D_t contains examples that belong to more than one class, an **attribute test condition** is selected to partition the records into smaller subset. A child node is created according to the outcome of the test. Afterwards the algorithm is recursively applied to each child node. Hunt's algorithm works very well if all the possible combinations are present in the training set and if each combination has a unique label. These assumptions are too constraining in most practical situations. Additional conditions are needed.

Decision tree induction algorithms have to provide a method in order to express an attribute test condition and its corresponding outcomes for different attribute types.

- **Binary Attributes** The test for a binary attribute generates two possible outcomes.
- **Nominal Attributes** Since a nominal attribute can have many values, the test can be expressed in two ways. If it is a multi-way split, the number of nodes is the number of distinct values for the corresponding attribute. For example, if an observation has three distinct categories: *Married, single, divorced* there is multiple ways to split. If it is a binary split, then we have to group the attributes as in Figure (**??**).

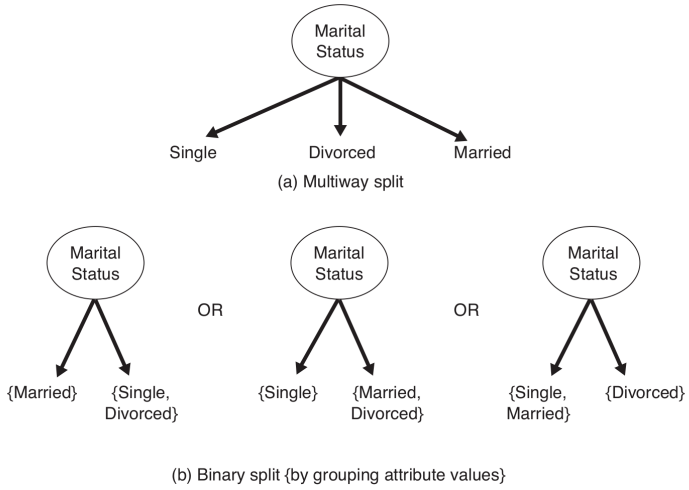


Figure 2.2: Test conditions for nominal attributes

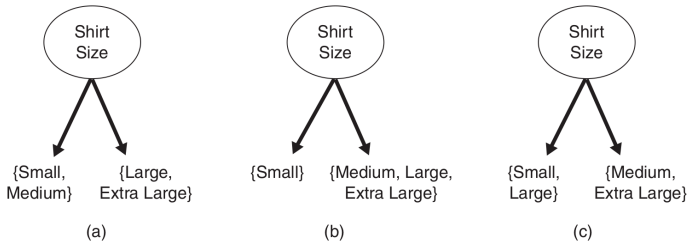


Figure 2.3: Different ways of grouping ordinal attribute values

- **Ordinal Attributes** Ordinal attributes can also produce multi ways splits. These attributes could be grouped into difference sets as long as it preserves the order.
- **Continuous Attributes** The test condition can be expressed as a comparison test for a given $(C < v)$ with binary output where $v \in \mathbb{R}$, or a range query with outputs of the form $v_i \geq C < v_{i+1}$. For the binary tree, the decision tree algorithm look after all possible split position v, and selects the one that produces the one which produces the most homogeneous sub sample. In case of a binary tree the algorithm must consider all possible split position v and choose the one that produces the best subset.

Here we will describe a popular decision tree method called Classification and regression tree algorithm (CART).

Regression Trees Let us consider a regression problem with continuous responses Y and inputs $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2\}$ with $i = \{0, 1\}$ $\mathbf{x}_i \in \mathbf{R}^n$, each taking values in the unit interval. To simplify matters, we restrict our attention to recursive binary partitions like that in We first split the space into two regions and model the response by the mean of Y in each region. We choose the variable and split-point to achieve the best fit. The one or both regions are splits into two more regions, and we repeat this process until some stopping rule is applied. For example, in the following figure the first region is split by $\mathbf{x}_1 = t_1$, then the region $\mathbf{x}_1 \geq t_1$ is split at $\mathbf{x}_1 > t_1$ is divided at $\mathbf{x}_1 = t_3$. Finally, the region $\mathbf{x}_1 > t_3$ is separated at $x_2 = t_4$. At the end, we have five different regions $R_1, \ldots, R_5$ shown in the following figure. This regression model predicts Y with a constant c_m in region R_m:

$$\widehat{f}(\mathbf{x}) = \sum_{m=1}^{5} c_m \mathbf{1}_{\{(\mathbf{x}_1, \mathbf{x}_2) \in R_m\}}$$

c_m are the estimated values of outcome Y in region R_m. This same model can be represented as a binary tree. The entire dataset is located at the top of the tree and each observation falls either into

the left branch or right branch satisfying the condition at the node. As we said previously, the leaves of the trees correspond to the regions $R_1, \ldots, R_5$. We have to say that the major key of the recursive binary tree is its interpretability. The feature space is fully represented by a single tree.

If we take as our criterion minimization the sum of squares $\sum_i (y_i - f(\mathbf{x}_i))^2$, taking the derivative regarding c_k, we can see that

$$\widehat{c}_m = \frac{1}{M} \sum_i y_i \text{ if } \mathbf{x}_i \in R_m$$

Now finding the best binary partition in terms of minimum variance is commonly computationally infeasible. Hence we could proceed with a greedy method. Starting at the root of the tree, consider a splitting feature j and cut point v and define the half-region:

$$R_1(j, v) = \{\mathbf{x} | \mathbf{x}_j \geq v\} \text{ and } R_2(j, v) = \{\mathbf{x} | \mathbf{x}_j > s\}$$

Afterwards, we are looking for the splitting variable j and split point v that solve:

$$\min_{j,v} \left[\min_{c_1} \sum_{\mathbf{x}_i \in R_1(j,v)} (y_i - c_i)^2 + \min_{c_2} \sum_{x_i \in R_2(j,v)} (y_i - c_2)^2 \right]$$

For $\forall (j, v)$, the internal minimization is:

$$\widehat{c}_1 = \frac{1}{N_1} \sum_{i=1}^{N_1} y_i \text{ and } \widehat{c}_2 = \frac{1}{N_2} \sum_{i=1}^{N_2} y_i$$

where $i = \{0, 1\}$ $N_1 = \#(\mathbf{x}_i \in R_1(j, v))$ and $N_2 = \#(\mathbf{x}_i \in R_2(j, v))$.

The idea behind is to get as much as possible two homogeneous groups in order to choose Y, that is why we minimize the variance in (2.4.1). More generally, one can define for m region:

$$\begin{aligned} N_m &= \#\{x_i \in R_m\}, \\ \widehat{c}_m &= \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \widehat{c}_m)^2 \end{aligned} \tag{2.3}$$

where T represents the tree. The function Q_m is called **impurity measure**.

For each splitting variable, the value of the split point v can be computed very quickly and hence by scanning through all of the inputs, determination of the best pair (j, s) is feasible. After founding the best split, we separate the data into two different regions and repeat the splitting process on each of the two regions until we attain a stopping rule.

Classifications Trees If the target is a categorical outcome taking for example $Y = \{1, \ldots, K\}$. The previous algorithm that we described would not change so much besides the splitting criteria the tree. For regression, we used the mean squared error (MSE) node impurity measure $Q_m(T)$ defined in (2.3), but this is not suitable for classification problem since the response variable is categorical. In a node m, representing a region R_m with N_m observations, let

$$\widehat{p}_{mk} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} \mathbb{1}_{y_i = k}$$

the proportion of class k observations in node m. We classify the observations in node m to class $k(m) = \text{argmax}_k \widehat{p}_{mk}$, the majority class in node m. There is a multiple measure $Q_m(T)$ of node impurity which includes the following:

- Misclassification error: $\frac{1}{N_m} \sum_{i \in R_m} \mathbb{1}_{\{y_i \neq k(m)\}} = 1 - \max_k \widehat{p}_{mk}$

- Gini index: $\sum_{k \neq k'} \widehat{p}_{mk}\widehat{p}_{mk'}$
- Cross-entropy or deviance: $-\sum_{k \neq k'} \widehat{p}_{mk} \log(\widehat{p}_{mk})$

For $K = 2$, if p is the proportion in the second class, these three measures are equal respectively to $1 - max(p, 1-p)$, $2p(1-p)$ and $-p\log(p) - (1-p)\log(1-p)$. We can see the following figure the three different measure. They are all similar except that cross-entropy and the Gini index are differentiable, and hence more suitable for numerical optimization.

The Gini index can be interpreted in two ways. In fact, rather than classifying observations to the majority class in the node, we could classify them to class k with probability $\widehat{p}_{mk}$. Then the expected training error rate of this rule in the node is $\sum k \neq k' \widehat{p}_{mk}\widehat{p}_{mk'}$ -the Gini index. Similarly, if we code each observation as 1 for class k and zero otherwise, the variance over the node of this $0-1$ response is $\widehat{p}_{mk}(1-\widehat{p}_{mk})$. Summing over classes k again gives the Gini index.

Advantages and Disadvantages of using Decision Trees

Decision trees offer multiple advantages over other methods. They are:

- **Visualization**. The interpretation of graphic is very helpful in the comprehension of the data set and the outcome dependencies.
- **Efficient**.

Although this algorithm is simple and intuitive, it has some limits.

2.5 Random Forests

So far our attention has focused on obtaining a specific tree for a given problem (in the context of classification or regression). However, Decision trees are models and hence they are subject to model misspecification. We can state that choosing a single tree and not giving some appreciation of its *mean squared error* (MSE) or other impurity measures gives a false notion of how good the classifier is. As we have said earlier, it is unclear how to estimate a standard error or bias for a tree, it is important to look into techniques that average over trees because as we will see it can reduce the variability that we could have with a single tree. One of these techniques is: Random Forests.

Random Forests build an extensive collection of *de-correlated* trees and then averages them. The essential idea is to average many noises as possible hence reduce the variance. Trees are ideal candidates for this since they can capture complex interaction structure in the data. If grown deep enough, they have a relatively low bias. This method was invented by Breiman[1] (2001) and substantially developed by Breiman and Cutler (2004)

Operationally , the random forest work as:

The main distinction between random forests for classification and regression is:

- When used for classification, the method gets a class vote from each tree and then classifies using majority vote.
- When used for regression, the predictions from each tree at a target point x are simply averaged.

There are two important *features* of Random forests that make the method particularly efficient.

[1]Leo Breiman (January 27, 1928 – July 5, 2005) was a distinguished statistician at the University of California, Berkeley. He was the recipient of numerous honors and awards and was a member of the United States National Academy of Science. - *Wikipedia*

Algorithm 1 Random forest for Regression or Classification

1: **for** i = 1 to B **do**
2: Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
3: Construct a Tree T_i from the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until a given stopping rule is attained:
- Select m variables at random from the p variables. - Choose the best splitting point among the m. - Split the node into two daughter nodes.
4: Return the ensemble of trees $\{T_b\}_1^B$
5: To make a prediction for test sample x:

- *Regression*: $\widehat{f}_{rf} = \frac{1}{B}\sum_{i=1}^{B} T_i(x)$
- *Classification*: Let $\widehat{C}_b(x)$ be the class prediction of the i-th tree. Then $\widehat{C}_{rf}(x)$ = majority vote $\{\widehat{C}_b(x)\}_1^B$

Out of Bag samples The first trick is *"out of bag"*(OOB) error estimate. It is a technique to get an estimate of the misclassification error. Indeed, the estimate is often claimed to be unbiased when it is corresponding to bootstrap samples. The idea is the following: *For each observation $z_i = (x_i, y_i)$, construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear.* For each data point, roughly $\frac{1}{3}$ of the bootstrap samples will not contain it, so approximately $\frac{2}{3}$ of the trees generate can be used.

Let us see what happened in the classifications case. If k be the class that gets the most votes for the i-th data point. The proportion of times that k differs from the class of the initial data point is the out-of-bag error estimate. Once the OOB error stabilizes, the training can be terminated.

Variable Importance Another significant feature is the variable importance. Recall that, at each step in a decision tree, the recursive partitioning examines all p variables to determine the *best split variable*. By contrast, random forests pick $\sqrt{p}$ of the variable at random, taking the best split among them afterwards. The improvement in the split-criterion at each split in each tree is the measure attributed to the splitting variable. It is accumulated over all the trees in the model separately for each variable. The candidate split-variable selection increases the chance that any single variable gets included in a random forest.

Random forests use the OOB samples in order to estimate the relative importance of, say, the j-th explanatory variable. When the b-th tree is grown, the OOB samples are passed down the tree and the prediction accuracy is recorded.The accuracy dropping as a result of this permuting is averaged over all trees and is used as of the importance of variable j in the random forest. For the classification case, random forest runs each observation through all the trees for which the observation is out of bag and counts the number of votes for the correct class. This method could also be be applied on the observations rather than on features. One can look at observations and create a local measure of importance specific to each observation in the training sample. So basically the procedure take an observation i, then run it down all the trees for which it is out-of-bag. It repeats this process taking randomly and permuted features j, and looks at the difference in the number of correct votes.

Analysis of Random Forests So far, we reviewed some great features of Random Forests. In this section, we will present multiple results that assure random forests will work well. However, we restrict our scope to the regression function. For the classification case, on can refer to [1] as most of the results in Random forests are due to Breiman.

We suppose that we have *i.i.d* sample $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $(\mathbb{R}^n \times \mathbb{R})$-valued random variable satisfying $\mathbb{E}Y^2 < \infty$. We recall that our goal is to estimate the regression function $r(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$ using the training sample. We will say that a regression function estimating r_n is consistent if $\mathbb{E}[|r_n(\mathbf{X}) - r(\mathbf{X})|^2] \to 0$ as $n \to \infty$. Formally, a random forest consists of a collection of random regression trees denoted by $\{r_n(\mathbf{X}, \Theta_m), m \geq 1\}$ where $\Theta = \Theta_1, \ldots, \Theta_m$ are *i.i.d*. These trees are then

aggregated in order to have:

$$\overline{r}(\mathbf{X}) = \mathbb{E}_{\Theta}[r_n(\mathbf{X}, \Theta)] \tag{2.4}$$

where $\mathbb{E}_{\Theta}$ is the expectation regarding the random variable Θ. Usually to estimate (2.4), we utilize the Monte-Carlo methods, *i.e.* by generating M (large enough) random regression trees and taking the average of the individual outcomes. The randomizing variable Θ_m characterizes the m-th random tree how we select the different features and how the successive cuts are performed such as the selection and the position of the coordinate to split Intuitively, reducing m will minimise the correlation between any pair of trees in the ensemble, thus decreasing the variance.

The mean squared generalization error for any numerical predictor is:

$$E[(Y - r(\mathbf{X}))^2]$$

We have the following results

Theorem 2.5.1. *Let n be the number of the training samples, we have:*

$$\mathbb{E}[(Y - \overline{r}(\mathbf{X}))^2] \xrightarrow[n\to\infty]{a.s} \mathbb{E}[(Y - r(\mathbf{X}))^2] \tag{2.5}$$

One could notice that in right-hand side of (2.5) is the error term that we denote $PE^*(forest)$ - the generalization error of the random forest. We define the generalization error of a tree as:

$$PE^*(tree) = \mathbb{E}_{\Theta}[\mathbb{E}[(Y - r(\mathbf{X}, \Theta))^2]]$$

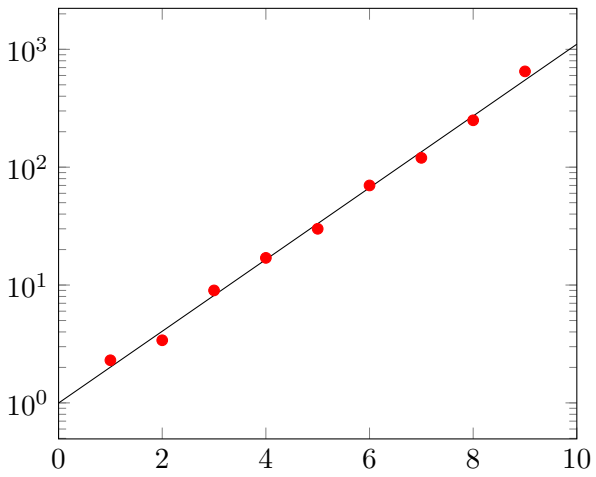
Theorem 2.5.2. *Assume that $\forall\Theta$, $\mathbb{E}[Y] = \mathbb{E}[r(\mathbf{X}, \Theta)]$, we have then*

$$PE^*(forest) \leq \rho PE^*(tree)$$

where ρ is the weighted correlation between the residual $Y - r(\mathbf{X}, \Theta)$ and $Y - r(\mathbf{X}, \Theta')$ with Θ, Θ' are independent.

2.6 Kernel Methods

In the previous section, we had seen multiple methods for predicting numerical values when the inputs space $\mathcal{X}$ was $\mathbb{R}^n$. Basically, we know how to do this:



However, the real data are often more complicated, they are not always real values or categorical values. For instance, the inputs could be graphs, chemical formulas and so on.

In this section, we are going to see kernel methods which help us to extend linear methods to a high-dimensional, unstructured and real-world data. The strength of these methods is that they lie on a **rigorous** mathematical framework.

2.6.1 Reproducing Kernel Hilbert Space

Our motivations here is to develop algorithms without making any assumptions regarding the data.

Let us roll some definitions. As we did before, we suppose that our inputs are denoted $\mathcal{X}$.

Definition 4. *A positive definite (p.d) kernel on a set $\mathcal{X}$ is a function k: $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$.*

Symmetry:

$$\forall \mathbf{x}, \tilde{\mathbf{x}} \in \mathcal{X},\ k(\mathbf{x}, \tilde{\mathbf{x}}) = k(\tilde{\mathbf{x}}, \mathbf{x})$$

Positivity condition:

$$\forall n \in \mathbb{N},\ \textit{suppose } (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathcal{X}^n \textit{ and } \alpha \in \mathbb{R}^n,\ \textit{then } \sum_i \sum_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$$

Remark 2.6.1.

- *Equivalently, we can say the kernel is p.d if and only if the kernel matrix* $\mathbf{K}$*, defined by* $(\mathbf{K})_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$*, is p.d*
- $\mathbf{K}$ *is always a squared matrix whatever the nature of the data: the algorithm will work regardless the type of the data (vectors, graphs, ...)*

Here we provide some example of p.d kernels:

Lemma 2.6.1. *Let Φ: $\mathcal{X} \to \mathbb{R}^d$. Then the function k: $\mathcal{X}^2 \to \mathbb{R}$ defined by:*

$$\forall \mathbf{x}, \tilde{\mathbf{x}} \in \mathcal{X},\ k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \Phi(\mathbf{x}), \Phi(\tilde{\mathbf{x}}) \rangle_{\mathbb{R}^d}$$

is a p.d kernel

Proof. Let $\alpha \in \mathbb{R}^d$.

- Symmetry:

$$\begin{aligned} k(\mathbf{x}, \tilde{\mathbf{x}}) =& \langle \Phi(\mathbf{x}), \Phi(\tilde{\mathbf{x}}) \rangle \\ =& \langle \Phi(\tilde{\mathbf{x}}), \Phi(\mathbf{x}) \rangle = k(\tilde{\mathbf{x}}, \mathbf{x}) \end{aligned}$$

- P.D:

$$\begin{aligned} \alpha^T K \alpha =& \sum_i \sum_j \alpha_i \alpha_j \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle \\ =& || \sum_i \alpha_i \Phi(\mathbf{x}_i) ||^2 \geq 0 \end{aligned}$$

□

The following result resumes our result above:

Theorem 2.6.1 (Aronszajn, 1950)**.** *k is p.d kernel if and only if there is a Hilbert space H and a feature map $\Phi : \mathcal{X} \to H$ such that*

$$\forall\ (\mathbf{x}, \tilde{\mathbf{x}}) \in \mathcal{X},\ k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \Phi(\mathbf{x}), \Phi(\tilde{\mathbf{x}}) \rangle$$

Proof. See appendix □

Remark 2.6.2. *Thus, a positive definite kernel provides:*

- *a function space of $\mathcal{X} \in \mathbb{R}$*
- *a norm for this space*

- *a feature map* Φ*:* $\mathcal{X} \to H, k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \Phi(\mathbf{x}), \Phi(\tilde{\mathbf{x}}) \rangle$

We observe that H *could be infinite.*

It brings us to the construction of the reproducing kernel Hilbert space (RKHS). Let $\langle \cdot, \cdot \rangle_H$ the inner product in H

Definition 5 (RKHS)**.** *Let* $\mathcal{X}$ *be a set and* $H \subset \mathbb{R}^{\mathcal{X}}$ *is the space of functions mapping* $\mathcal{X}$ *into* $\mathbb{R}$ *via:*

$$\Phi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}} \text{ where } x \mapsto k(\cdot\,, x)$$

The function $k : \mathcal{X}^2 \to \mathbb{R}$ *is called the reproducing kernel (r.k.) of* H *if*

- H *contains all functions of the form*

$$\forall \mathbf{x} \in \mathcal{X}, k(\cdot, \mathbf{x})$$

- *For* $\mathbf{x} \in \mathcal{X}$ *and* $f \in H$ *the reproducing property holds:*

$$f(\mathbf{x}) = \langle f, k(\cdot, \mathbf{x}) \rangle_H$$

(i.e. $k(\cdot, \mathbf{x})$ *corresponds to "Dirac" at* $\mathbf{x}$*)*

If a r.k. exits, then H *is called a RKHS.*

Definition 6 (RKHS)**.** *Let* $\mathcal{X}$ *be a set and* $H \subset \mathbb{R}^{\mathcal{X}}$ *is the space of functions mapping* $\mathcal{X}$ *into* $\mathbb{R}$*:*

$$H := \left\{ \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot) : n \in \mathbf{N}, \mathbf{x}_i \in \mathcal{X}, \alpha_i \in \mathbb{R}, i = 1, \ldots, n \right\}$$

H *is a Reproducing Kernel Hilbert Space equipped with a dot product* $\langle \cdot, \cdot \rangle_H$*. If it exits a function* $k : \mathcal{X}^2 \to \mathbb{R}$ *, called the reproducing kernel (r.k.) of* H *with the following properties:*

- *for all elements* $\mathbf{x} \in \mathcal{X}, k(\mathbf{x}, \cdot)$ *belongs to* H
- *For* $\mathbf{x} \in \mathcal{X}$ *and* $f \in H$ *the reproducing property holds,i.e.:*

$$\forall f \in H, \langle f, k(\mathbf{x}, \cdot) \rangle_H = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) = f(\mathbf{x})$$

(i.e $k(\mathbf{x}, \cdot)$ *corresponds to "Dirac" at* $\mathbf{x}$*)*

If a r.k. exits, then H *is called a RKHS.*

We can also stipulate another definition of RKHS using the **Riesz** representation:

Theorem 2.6.2. *The Hilbert space* $H \subset \mathbb{R}^{\mathcal{X}}$ *is a RKHS iff for* $\mathbf{x} \in \mathcal{X}$

$$\begin{aligned} F &: H \to \mathbb{R} \\ &f \mapsto f(\mathbf{x}). \end{aligned}$$

is continuous.

A positive definite kernel could be seen as of an inner product after embedding the input space $\mathcal{X}$ in some Hilbert space. Such p.d kernel define a **metric** on $\mathcal{X}$

The following result gives us the uniqueness of r.k:

Proposition 2.6.1. *If* H *is a RKHS then there exists a unique function* Φ *of* H *such that* $\forall \mathbf{x} \in \mathcal{X}$*,* $\forall f \in H,\ f(\mathbf{x}) = \langle \Phi(x), f \rangle$*.* k *is called a* ***reproducing kernel***

Example of kernels Let us see some example of kernels. We already presented some positive definite kernels, and now intend to do so for non-exhaustive list of p.d kernels.

Linear kernel Take $\mathcal{X} = \mathbb{R}^d$ and the **linear kernel**: $k(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle_{\mathbb{R}^d}$ Here H is linear function space which we associate the euclidean norm $||x||_{\mathbb{R}^d} = \sqrt{(\langle \mathbf{x}, \mathbf{x} \rangle)_{R^d}}$ and $\Phi(\mathbf{x}) = \mathbf{x}$.

Polynomial kernel Take $\mathcal{X} = \mathbb{R}^d$, $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^r$ where $r \in \mathbb{N}$.

$$k(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{p} x_i y_i \right)^r = \sum_{i_1 + \cdots + i_p = r} \frac{r!}{i_1! \dots i_p!} (x_1 y_1)^{i_1} \dots (x_p y_p)^{i_p} \text{ (multinomial theorem)}$$

Thus the feature map is $\Phi(\mathbf{x}) = \left(\frac{r!}{i_1! \dots i_p!} \right)^{\frac{1}{2}} (x_1)^{i_1} \dots (x_p)^{i_p}$. Finally $H = \{$ homogeneous polynomial function of degree p $\}$. We observe that $\dim H \approx p^r$, it is a huge space. The kernel allows to manipulate a large space without paying the cost.

We will see now some applications of kernels and RKHS in Machine Learning. But first, let us introduce some key results. A family of powerful algorithms for data analysis using p.d. kernels rely on two theoretical results:

- The kernel trick based on the representation of p.d. kernels as inner products.
- The representation theorem

Kernel tricks The following result is fundamental in the construction of algorithms. It has huge practical applications.

Proposition 2.6.2. *Every algorithm based on finite dimension using only inner product could be replaced by the inner product of any p.d. kernel.*

Example 1. *For instance, consider a simple classification problem where $\mathcal{Y} = \{-1, 1\}$ and so the training set is $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$. The algorithm is to compute the mean of two classes in the feature space:*

$$c_{+1} = \frac{1}{n_+} \sum_{i=1}^{n} \Phi(\mathbf{x}_i) \mathbb{1}_{\{y_i = +1\}} \text{ and } c_{-1} = \frac{1}{n_-} \sum_{i=1}^{n} \mathbb{1}_{\{y_i = -1\}} \Phi(\mathbf{x}_i)$$

where n_+ and n_- corresponds respectively to the number of positive examples and negative examples. Thus we assign a new point $\Phi(\mathbf{x})$ to the class whose mean is closest The prediction could be written as (we suppose that the closest means have the same distance to the origin):

$$y = \text{sign}(\langle \phi(\mathbf{x}), c_{+1} \rangle - \langle \Phi(\mathbf{x}), c_{-1} \rangle)$$

This could be rewritten as follows:

$$y = \text{sign} \left(\frac{1}{n_+} \underbrace{(\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i) \rangle)}_{k(\mathbf{x}, \mathbf{x}_i)} \mathbb{1}_{\{y_i = +1\}} - \frac{1}{n_-} \underbrace{(\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i) \rangle)}_{k(\mathbf{x}, \mathbf{x}_i)} \mathbb{1}_{\{y_i = -1\}} \right) \tag{2.6}$$

This substitution is called the kernel trick. There is example of p.d. kernels which can be evaluated efficiently even though they correspond to dot products in infinite dimensional dot product spaces.

Reprensenter Theorem We know that the RKHS H is a space function and for $f \in H$ $||f||_H$ it measures the smoothness of the function f. Given a training set $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ a natural way to do a regression is to find the minimum of the following objective function

$$\frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(\mathbf{x}_i)) + \lambda ||f||_H^2 \tag{2.7}$$

where the f is the loss function. The following theorem shows that the (2.7) could be expressed in terms of kernel expansions as for large class of optimization problems.

Theorem 2.6.3. *Let H be a RKHS, $(\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathcal{X}^n$ and $\Psi : \mathbb{R}^{n+1} \to \mathbb{R}$ strictly increasing with respect to the last variables then each minimisers $f \in H$*

$$\min_{f \in H} J(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n), ||f||_H)$$

admits the following representation of the form:

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

Proof. See appendix □

Example 2. *We take (2.7) and suppose that $\mathbf{u} \mapsto l(y, \mathbf{u})$ is the mean squared error, we have then:*

$$J(f(x_1), \ldots, f(\mathbf{x}_n), ||f||_H) = \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2 + \lambda ||f||_H^2 \tag{2.8}$$

One observes that we are in kernel ridge regression case. Thanks to representer theorem, we know that the solution (2.8) can be expanded as:

$$f^*(\mathbf{x}) = \sum_{i} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

Thus the minimisation rewrites as follows:

$$\begin{aligned} f^* &= \arg\min_{f \in H} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i))^2 + \lambda ||f||_H^2 \\ &= \arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} (K\alpha - \mathbf{y})^T (K\alpha - \mathbf{x}) + \lambda \alpha^T K \alpha \end{aligned} \tag{2.9}$$

This function being convex and differentiable w.r.t α. Its minimum can therefore be found by setting the gradient to zero. For $\lambda > 0$, the unique solution (2.9) is

$$\alpha = (K + \lambda n I)^{-1} \mathbf{y}$$

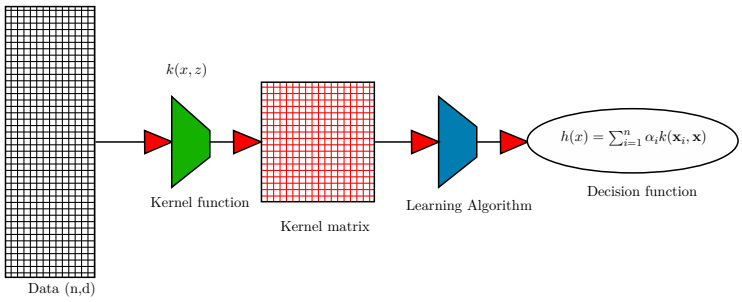


Figure 2.4: Processing chain of kernel methods

2.6.2 Application: Support Vector Machine

Here we are interested on the binary approximation, *i.e.* finding a decision rule in order to distinguish classes and this with the help of the hyperplane $h : \mathcal{X} \to \mathcal{Y}$, $h(\mathbf{x}) = \langle \mathbf{w}, x \rangle + w_0$. We suppose that $\mathcal{X}$ has a dimension equal to d and $\mathcal{Y} = \{-1, 1\}$. We use the following decision rule: $g(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + w_0)$. The first intuition that explains why we choose this decision rule is to represent the positive and negative training examples as below:

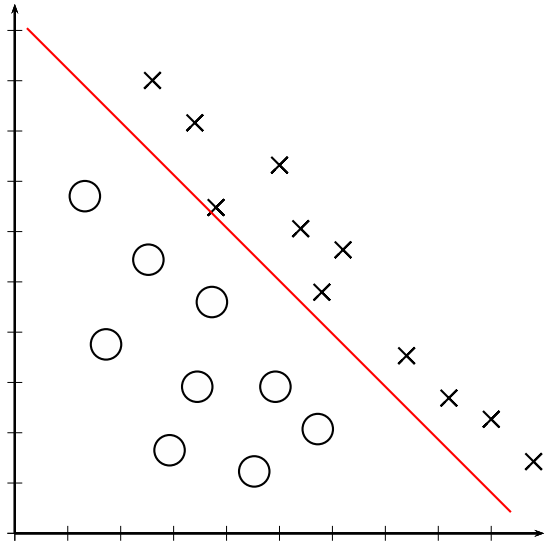


Figure 2.5: Representation of the classification

As we said in the Linear Regression section, a "good classifier" has to minimize the number of misclassification using the training set $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$. *Vapnik et al.* [4] decided to rather look to a confidence criterion, in other words, to consider the margin separating the example of the class "+1" and "-1" as in Figure (2.5). The distance of a new point $\mathbf{x}'$ to the hyperplane h is given by: $\langle \mathbf{w}, \mathbf{x}' \rangle + w_0$ as the vector $\mathbf{w}$ is orthogonal to the plan. However, in order to compare many hyperplanes with each other, we want to impose some sort of a normalization. From now on, we will consider $(\mathbf{w}/\|\mathbf{w}\|, w_0/\|\mathbf{w}\|)$. As there is an infinite number of margins which separates into two classes, we have to seek one of these margins which to do this at best. One way to formalize it, is to define the optimal margin:

$$\arg\max_{\mathbf{w}, w_0} \min\{\|\mathbf{x} - \mathbf{x}_i\| : \mathbf{x} \in \mathbf{R}^d, (\mathbf{w}^T\mathbf{x} + w_0) = 0, i = 1, \ldots, i = 1, \ldots, n \tag{2.10}$$

Transforming (2.10) into a nicer one, we have

$$\begin{array}{ll} \min & \frac{1}{2}^2 \|\mathbf{w}\| \\ \text{subject to} & y_i(\mathbf{w}^T\mathbf{x}_i + w_0) \geq 1 \; i = 1, \ldots, n. \end{array} \tag{2.11}$$

The previous problem is called the primal optimization problem and implies $(d+1)$ parameters. This problem could be solved using a quadratic programation method. However, the complexity of (2.11) is $\mathcal{O}(d^3)$. This problem becomes hard to solve when d is more than few hundreds. Fortunately, using the **Lagrange duality** we can reformulate the problem to solve it easier.

Dual Formulation In the optimization theory, we say that an optimization problem has a dual formulation when the objective function and the constraint are strictly convex. Thus solving the dual

optimization problem is equal to the primal (strong duality). Let us write the Lagrangian of the problem (2.11):

$$\alpha \in (\mathbb{R}_+)^n,\ \mathcal{L}(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \alpha_i y_i (\mathbf{w}^T \mathbf{x}_i + w_0)$$

Computing the gradient of $\mathcal{L}$ and setting it to zero:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \text{ and } \frac{\partial \mathcal{L}}{\partial w_0} \tag{2.12}$$

We find:

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \mathbf{w}^* = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{2.13}$$

We check here the representation theorem which stipulates the vector parameter $\mathbf{w}*$ is the linear combination of the training set $\mathbf{x}_1, \ldots, \mathbf{x}_n$. More precisely, the Karush-Kuhn-Tucker (KKT) conditions show that only the point lying on decision boundary will count, *i.e.* $\langle \mathbf{w}^*, \mathbf{x}_i \rangle + w_0^* = \pm 1$. These points where the Lagrange multipliers α is non-null is called *support vectors*. One noticed that the number of support vector is lower that the size of the training set, it will reduce the cost of the resolution. Substituting (2.13) in (2.12), we remove all the primary variables and we obtain the dual formulation:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \text{subject to} \quad & \alpha_i \geq 0 \quad i = 1, \ldots, n. \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{2.14}$$

The corresponding hyperplane solution writes:

$$h^*(\mathbf{x}) = \langle \mathbf{w}^*, \mathbf{x} \rangle + w_0^* = \sum_{i=1}^{n} \alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + w_0^*$$

In the case of non-separable case, we are going to use the theory we saw previously. One noticed that the problem (2.15) depends on the inner product (without computing it). Using the non-linear future map Φ, we can use the kernel trick. Equivalently, we can replace all $\mathbf{x}$ by $\Phi(\mathbf{x})$ in our previous equations. Thanks to the kernel trick, the dual optimization problem could rewrites:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to} \quad & \alpha_i \geq 0 \quad i = 1, \ldots, n. \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \end{aligned} \tag{2.15}$$

This transformation leads to huge reducing cost of computations because computation $k(\mathbf{x}_i, \mathbf{x}_j)$ is much more easier than $\langle \Phi(x_i), \Phi(x_j) \rangle$ especially when the feature space is high dimensional

2.6.3 Example: Iris dataset

We applied the SVM method to the Iris dataset. We remark that we can separate the data set into 3 different class. Nonetheless, this separation has to be non-linear in order to be effective.

For several values of regularization parameter, we observe that high values of C encourage the over-fitting as in Figure (2.7)

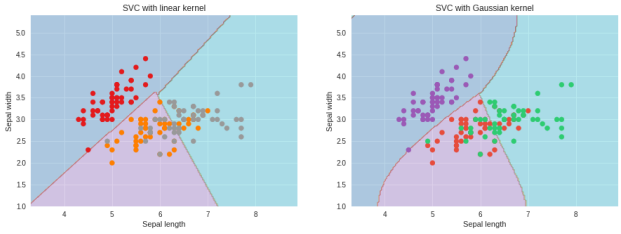


Figure 2.6: Visualization of the different clusters

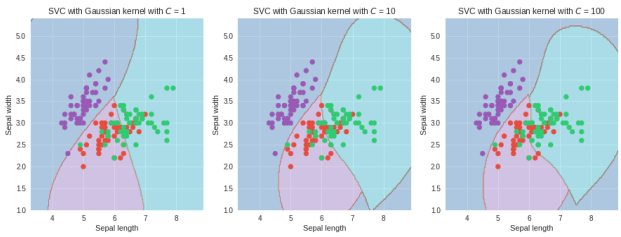


Figure 2.7: Visualization of the regularization parameter C impact

2.7 Neural Networks

Neural Networks or often called **Multilayer Layer Perceptrons** (MLP) are the tremendous illustration of deep learning models. Here we describe the most widely used "vanilla" neural networks as there is a huge type of neural network. There has been a sort of *hype* surrounding neural networks, making them at the same time *magical* and *mysterious*. The goal of the neural network is the same all the models we have seen before: to approximate some function f^* which can forecast the response variable. For instance, the classification task is to find f^* such that $y = f^*(\mathbf{x})$ where y is the category. These models are feed forward model because information flow through different layer where $\mathbf{x}$ is evaluated through intermediate computations to finally compute y. The neural networks are called **networks** because they are representing using an ensemble of many different functions.

Model Let us see how we are going to represent the hypothesis function. At very basic level, each neuron is doing a simple task as a regression taking an input $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Outputs of each **activation unit** z_m are created from linear combination of the inputs and then the target y_k^i is modelled using the linear combination of the z_m^i: Let j be the number of the layer, we define the output sequence of each layer j by:

$$z^{j+1} = \Theta^j a^j$$
$$a^{j+1} = g(z^{j+1})$$

where $a^1 = \mathbf{x}$

In this case, the layer $j = 1$ is referred to as the **first layer** and the layers $j = 2, \ldots, n-1$ is called the **hidden layers** and g is known as the activation function. We will discuss shortly the choice of the activation function for each layer.

One way to understand the neural networks is to see how a simple activation **unit** work. An activation unit is equal to a linear model such as the linear regression or the logistic regression. As we saw in the previous section Linear Models, these models may fit efficiently and reliably. However, the defective aspect is that the model capacity is limited to linear function and thus it could not catch the relation between two features. We can represent a non-linear function of $\mathbf{x}$ to overcome this void, transform the input $\mathbf{x}$ by $\Phi(\mathbf{x})$. Equivalently, we can apply the kernel trick described in (2.6.2) to create a non-linear mapping of the inputs. Hence, taking outputs of a given layer as inputs for a the adjacent layer could facilitate the discovering of relations and improve predictions.

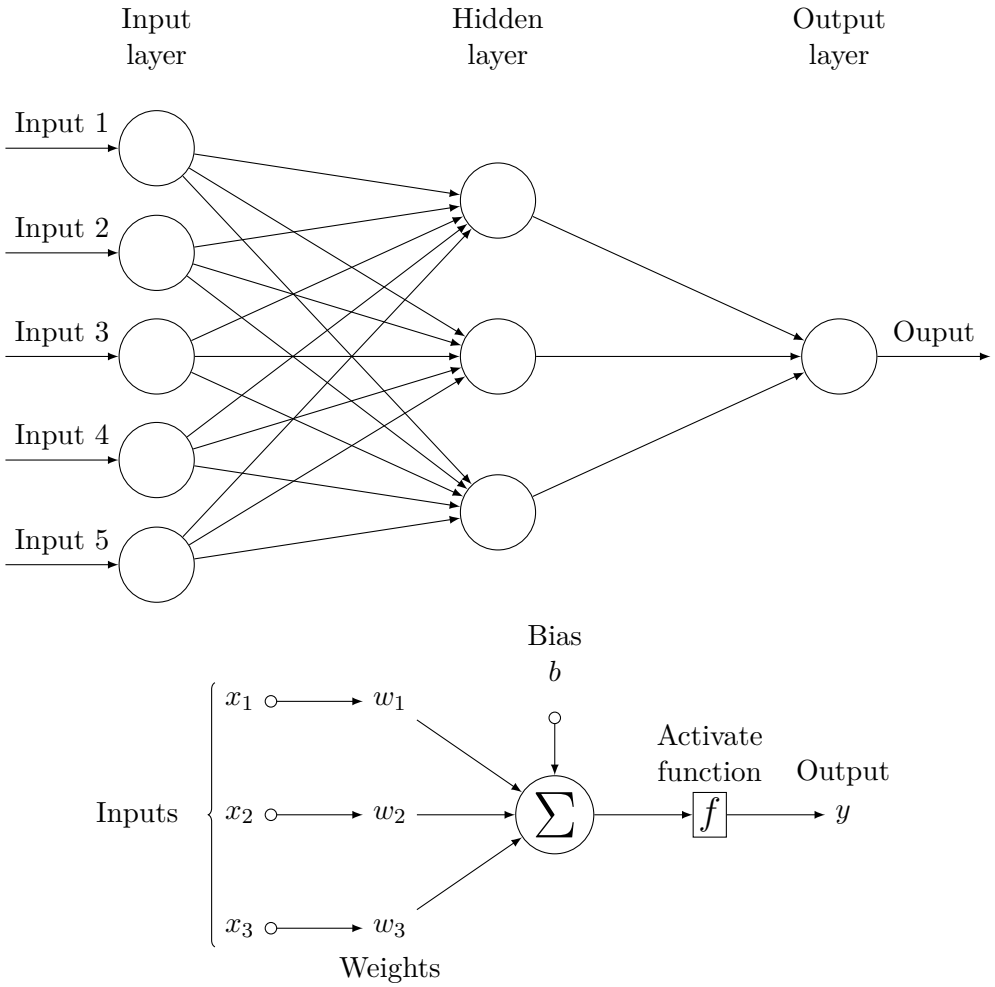


Figure 2.8: An illustration for a simple Neural Network with a single hidden layer

Fitting Neural Networks The reader could notice that as for each model we have unknown parameters, in our case *weights*, that we want to find in order to fit the training set well. We denote $(h_\theta)_k$ as being a hypothesis that results in the k-th output. For regression, we use the sum of squared as loss function:

$$J(\Theta) = \sum_{i=1}^{n}\sum_{k=1}^{K}(y_{ik} - (h_\theta(\mathbf{x}_i))_k$$

For classification, we use either cross entropy or the sum of squared as loss function:

$$J(\Theta) = -\sum_{i=1}^{n}\sum_{k=1}^{K} y_{ik}\log(h_\theta(\mathbf{x}_i))_k$$

and the corresponding classifier is $G(x) = argmax_k fk(x)$ In other to avoid over-fitting the training set, we will need some regularizations: this is done by adding a penalty term to the objective function. The usual to way to minimize $J(\Theta)$ is by *Gradient Descent*, called *back-propagation* in this setting. Thanks to the composition of the model, the gradient is computed using the chain rule.

Over-fitting Because of the high number of weight in Neural networks, it increases the chance of over-fitting. A method to avoid this is to use *weight decay* which is an alternative to ridge regression used in linear model. Thus, the objective function rewrites as:

$$\tilde{J}(\Theta) = J(\Theta) + \lambda R(\Theta)$$

2.8 Validations

Validation is the key to making real progress in Machine Learning. In the previous part, we encountered many methods but we do not know how to compare one with another. We need a good indicator of performance but using only the training set would not be a good idea. We introduced at the beginning two sets: training, test set but we did not give any explanations on them. As it may sounds we need two different sets to evaluate the predictive power of the model.

2.8.1 Training and Testing

For classifications problems, it is natural to measure classifer's performances towards the *error rate* also said *accuracy*. The error is the proportion of errors made over a set of examples. An error is the wrongly classified example. Of course, what we are interested in is the performance on new data. Why? Because the classifier has been learning form the very same training data so any estimate of performance will be optimistic and not really exploitable. The error rate on the training set is called the *re-substitution error*. Although it is not a reliable predictor of the true error rate on unseen data. To tackle this problem, we need to split the dataset into two sets and train the model on one and test it on the other one. In general, 70% of the data can be used as a training set, and the remaining 30% can be used for the validation process.

Often some models involve two stages, one to come up with a basic structure and the second to optimize parameters involved in that structure. In such situations, we need three sets: the *training* data, the *validation* data, the *test* data. The training data will be used to learn, the validation set is used to optimize parameters of the model and the test set to calculate the accuracy of the final, optimized method. Each of three sets must be different, the intersection has to be null. The *accuracy* seems to be a good indicator of performance. However, the predictor is not adapted for all data.

Metric: Accuracy Let us introduce some definitions:

- **True Positive error(TP)** is a result that indicates a given condition exists, when it does.
- **True Negative error(NP)** a result that indicates that a condition does not hold, when it does not.

- **False Positive error (FP)** is a result that indicates a given condition exists, when it does not.
- **False negative error (FN)** is a result that indicates that a condition does not hold, while in fact it does.

Accuracy is the proportion of correct results, either true positive or true negative. Formally, this can be stated as:

$$\text{Accuracy} : \frac{TP + TN}{TP + TN + FP + FN}$$

Sensitivity (*resp.* **Specificity**) is the proportion of true positive (*resp.* true negative)

$$\text{Sensistivity} : \frac{TP}{TP + FN} \quad \text{Specificity} : \frac{TN}{TN + FP} \tag{2.16}$$

By our previous definitions, the accuracy measure is not adapted for every situation. Especially in our case where the main goal is to predict a minor class. Indeed the Figure (3.1) shows us the proportion of people which accepted the deposit is lower than the proportion who rejected the deposit. Using the accuracy metric we will always have a good result because the number of TN will be higher than TP as there is many examples where the individuals have rejected the deposit. Thus, we will have to use another metric: **ROC Curve**.

Metric: ROC Curves ROC Curves are used when the model is trying to select samples of test examples that have a high proportion of positives. The acronym stands for Receiver Operating Characteristic (ROC). The name came from signal detection to characterize the trade-off between hit rate and false alarm rate over a noisy channel. ROC Curves describe the performance of a classifier without regard to class distribution or error costs. They plot sensitivity over the (1-specificity). We define for that purpose the true positive rate and false positive rate:

$$\text{TPR} : \frac{TP}{TP + FN} \quad \text{FPR} : \frac{FP}{TN + FP}$$

As one can remark, TPR is equal to sensitivity and FPR is equivalent to (1 – specificity). TPR and FPR together fix a point in the ROC curve and the position of a point in this curve shows the trade-off between sensitivity and specificity. Thus the location of the point describes whether the classification is useful or not. An ideal situation is a point yielding to the coordinate $(0, 1)$. This ideal points indications the classifier has sensitivity of 100% and specificity of 100%. A graphical presentation of what we described is below.

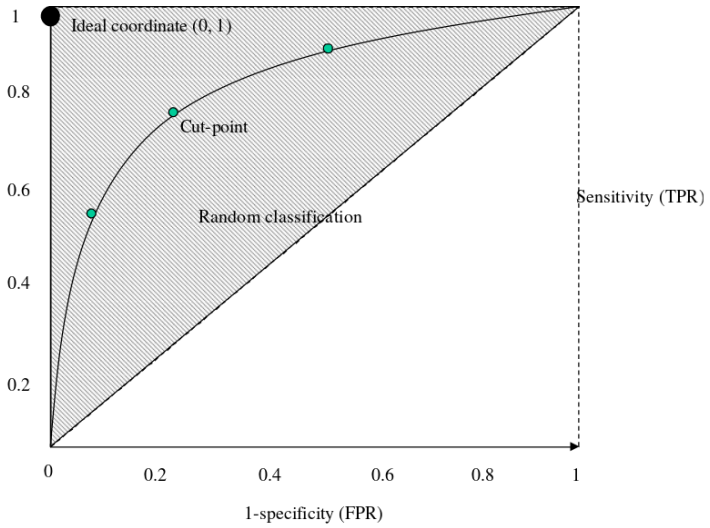


Figure 2.9: Hypothetical ROC Curve

The interpretation of ROC Curve is: closer the point of ROC Curve to the ideal coordinate, the more accurate the classification is. Closer the points on the ROC curve to the diagonal, less accurate

the classification is. The Area Under ROC Curve (AUC) provides a way to measure the accuracy of a classification test. The classifier is more accurate when the area is larger.

$$\text{AUC} \ : \int_0^1 ROC(t)dt$$

AUC Range	Classification
$0.9 < \text{AUC} < 1.0$	Excellent
$0.8 < \text{AUC} < 0.9$	Good
$0.7 < \text{AUC} < 0.8$	Worthless
$0.6 < \text{AUC} < 0.7$	Not Good

Table 2.1: Accuracy classification by AUC for a classification

The table (**??**) shows how to interpret the AUC, it will help us to compare different models.

2.8.2 Cross-Validation

In the previous section, we discussed splitting the data into two sets. However, we may be unlucky: the sample used for training (or testing) might not be representative of data. In general, you cannot tell whether a sample is representative or not. One can perform this simple check: each class in the entire dataset should be represented in approximately the right proportion in the two subsets. If this is not the case, the classifier will learn the over-presented class in the training set, and it would give mediocre results in the test set. Instead, we should ensure that the random sampling is done in a way that guarantees each class is properly represented in both training and test sets. This procedure is called *stratification*.

More generally, one can split the data into k different set or *folds* of the data, this method is called *cross-validation*. This method directly estimated the error $\mathbb{E}[L(\mathbf{y}, \hat{f}(\mathbf{X}))]$, the average generalization error when the method $\hat{f}(\mathbf{X})$ is applied to an independent test sample from the joint distribution of $\mathbf{X}$ and $\mathbf{y}$.

K-Fold Cross-Validation Ideally, if we had enough data we would set a side a validation set and use it to measure the performance of our prediction model. K-fold cross-validation uses sets of the full data to fit the model and a different set to test it. For instance, when K=5 the scenario looks like this:

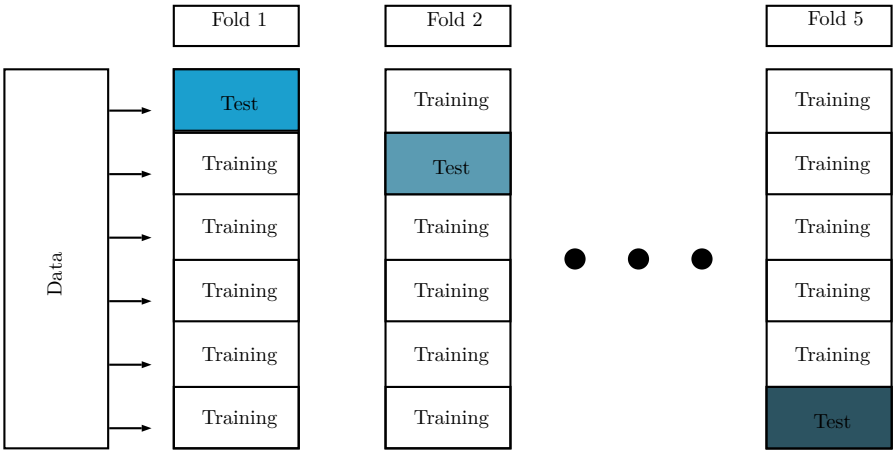


Figure 2.10: K-fold splitting with $K = 5$

For the K th part, we fit the model to other $k - 1$ parts of data, and calculates the prediction error of the fitted model when predicting the kth part of data. We do this for $k = 0, \ldots, K$ and combines the K estimates of prediction error. Let $\kappa : \{1, \ldots, N\} \mapsto 1, \ldots, K$ be an indexing function

that shows the partition to which observation i allocated by the randomization. Let $\hat{f}^k(x)$ be the fitted function, computed with the k-th part of the data removed. Then the cross-validation estimate of prediction error is

$$\mathrm{CV}(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}^{\kappa(i)}(\mathbf{x}_i)$$

In general, one chose $K = 5$ or 10 according the size of data. The case $K = N$ is known as *leave-one-out* cross-validation.

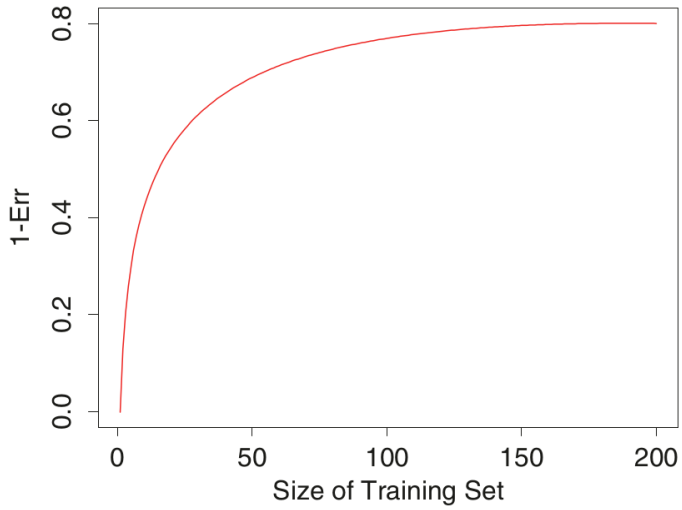


Figure 2.11: Theoretical learning curve for a classifier: a plot of $1Err$ versus the size of the training set N

Chapter 3

Applications

Contents

So far we have seen some strong and robust algorithms in supervised learning. We choose a dataset to compare these algorithms and see which one is more efficient. We want to highlight the fact that this work is done for a given dataset and should not be used as "generic" approach for other datasets. The data we choose here is related to direct marketing campaigns (phone calls) of a Portuguese banking institution [7]. The goal is to predict if the client is going to subscribe to a long-term deposit. The choice of this particular dataset is motivated by its ability to be used either in classifications algorithm and regression algorithms because the variable to predict is binary. Prior to start with this work, let us talk about the coding language that we used: *Python*. This choice is due to its easy way to implement algorithms. Python is an object oriented programming language, multi-paradigm and multi-platform. It encourages structured imperative feature. It is endowed of strong typing, automatic memory management. The strong point of Python is the availability of large libraries for a wide range disciplines. Especially for scientific purposes, we have *Numpy, Scipy*. In case of machine learning, we have *scikit-learn*, *Theano*, *Tensor-flow*. Tensor-flow is an ecosystem and in practice, you would not use directly it. There is a library built on top of Tensor-flow called *Keras* that we are using for this work.

3.1 Data Analysis

3.1.1 Data description

The data is collected from a Portuguese bank who used its own contact centre to do directed marketing. The dominant marketing channel was the telephone with an interlocutor. Moreover, each advertisement was managed in an integrated process and the results for all channels were assembled. The collected dataset is related to 17 campaigns occurred between 2008 and 2010 to a total of 45211 contacts. During these phone campaigns, an attractive deposit long-term deposit with real interest

rates. Hence, for each contact they registered a large number of attribute and if there was a success or not. The obtained data are most of them related to personal information of the client. (**??**)

Features	**Description**
	Personal information
Age	Age of the client (Numeric)
Marital status	Married, single, divorced (Nominal)
Sex	Male or Female (Nominal)
	Bank Client Information
Annual balance	in Euro currency (Numeric)
Debt card	Yes or No
Loans in delay	Yes or No (Nominal)
	Last Contact Information
Agent	Human that answered the call
Date and time	Referring to when the contact was made
	Visualizations Information
Number of times the client has seen the product in the home banking site	
Result of the last campaign if another contact was made	
Days since the last contact in other campaigns	

Table 3.1: Dataset features descriptions

Fortunately, we have no missing value that makes things easier. Let us see the behaviour of all features according to the target variable. One can find more specific information regarding the data in the appendix.

As we can guess, we have more rejections than subscriptions:

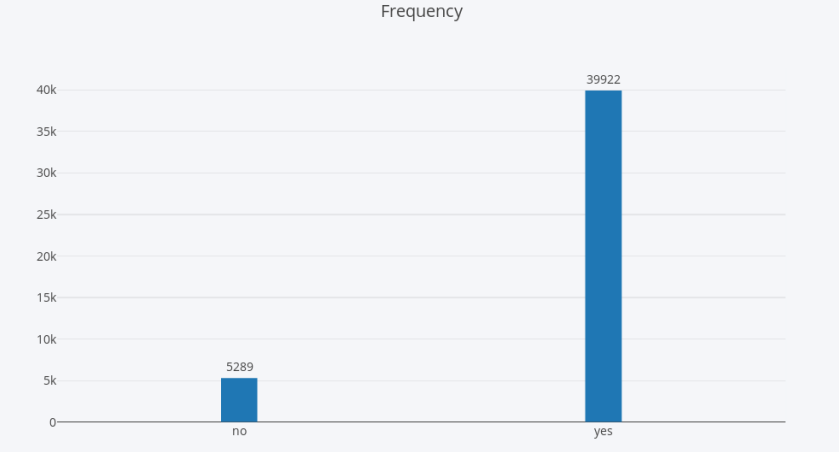


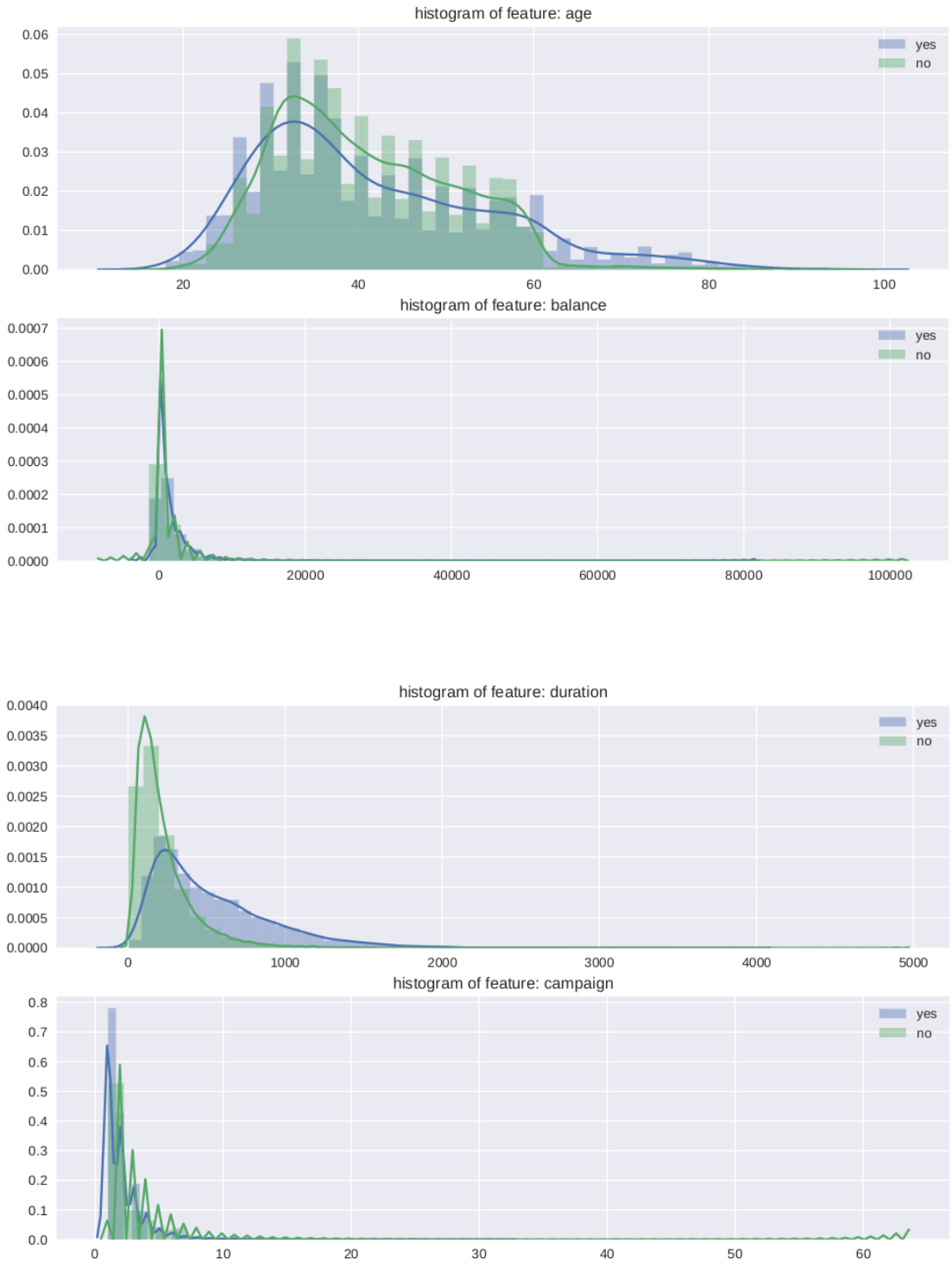
Figure 3.1: Frequency for each class: Number of accepted and rejected the deposit

Our dataset is not so unbalanced if this is the case we should do some preprocessing work because the accuracy score to evaluate the regression is going to be wrong, we are going to assign all the example to the majority class. In the bank point of view, we are more interested in the people who are going to accept the long-term deposit. However, here we provide some strategies to tackle this problem:

- Collect more data
- Change the performance metric (we will discuss it later)
- Re-sample the dataset: create two homogeneous class in the training class

Let us see how the distribution of numerical features is:

Figure 3.2: Different histograms according to the features



3.1.2 Data preparation

In this section, we will discuss the form that allows a machine learning algorithm to learn important features of the dataset quickly. We give a concrete example to illustrate the previous point. Consider modelling the rise/fall of a stock price, to train the model we may have data over several years from different companies and such data will not be useful to predict a share price (few data per companies) but the trained model on this data will reveal some interesting basic information. Indeed, we will

deduce that two principal data are needed the stock price and the earnings. The algorithm will be able to learn that the rise/fall of the stock price depends upon the ratio price-to-earnings but to do so it will require both time and data. However, the users already know this, it would have helped the algorithm if we had removed the column of price and that of earnings and had inserted a column consisting of the ratio. This example is referred to by the general injunction that one should add domain knowledge into the training data set. This parenthesis is made, in our case we will not need to add further features to make good predictions. Nevertheless, many algorithms do not support non-numerical value in dataset. Unfortunately, we have categorical and binary value in our dataset Table (**??**). In instance-based learning, categorical features can be treated as numeric by defining

Table 3.2: Example: Marital Status

	Marital Status
1	Married
2	Divorced
3	Single
4	Divorced

the "distances" between two nominal values that are the same, for instance, as 0 and between two different nominal values as 1 and this regardless of the actual values involved. This can be achieved for categorical features: replace a k nominal feature by k synthetic binary attributes, one for each value indicating whether the attribute has that value or not.

Table 3.3: Example: Martial Status feature after transformation

	MaritalStatus_Married	MaritalStatus_Divorced	MaritalStatus_Single
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	1

This distance is insensitive to the feature values involved because the information of the feature is not lost. Only "same" or "different" information is encoded, not the hues of difference that be associated with various outcomes of the features. A more interesting distinction can be made if the nominal values have weights reflecting their relative importance. For example, being a senior officer ("*blue-collar*") in "Job" feature tend to accept the long term deposit than a student, so more weight will be put on these attributes "blue-collar" than "student". If the value of the features are ordered, more issues will arise but fortunately we do not have these kinds of values.

This data encoding is done using the class `LabelEncoder`. After this task, we extend the number of features to 39.

```
'job[T.blue-collar]', 'job[T.entrepreneur]', 'job[T.housemaid]',
'job[T.management]', 'job[T.retired]', 'job[T.self-employed]',
'job[T.services]', 'job[T.student]', 'job[T.technician]',
'job[T.unemployed]', 'marital[T.married]', 'marital[T.single]',
'education[T.secondary]', 'education[T.tertiary]',
'contact[T.telephone]', 'month[T.aug]', 'month[T.dec]', 'month[T.
    feb]',
'month[T.jan]', 'month[T.jul]', 'month[T.jun]', 'month[T.mar]',
'month[T.may]', 'month[T.nov]', 'month[T.oct]', 'month[T.sep]',
'poutcome[T.other]', 'poutcome[T.success]', 'age', 'default', '
    balance',
```

```
'housing', 'loan', 'day', 'duration', 'campaign', 'pdays', '
    previous',
'pdays_yes'
```

Furthermore, some algorithms need more preparation than others like SVM or Neural Networks. In many scenarios, the different features have different scales and the model may not be comparable to each other. For example, the dataset has two continuous features: *balance*, *age*. They are drawn on a different scales each other. The latter feature is typically orders of magnitude larger than the former. As a consequence, any aggregate function computed on these features will be dominated by the feature of larger magnitude. To address this problem, the common process is the *standardization*. Consider the case where the j-th variable has the mean μ_j and the variance σ_j^2. Then the new attribute $\mathbf{z}$ rewrites:

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}$$

A second approach is to reduce the variables to $[0, 1]$ and for that one can do the following:

$$z_{ij} = \frac{x_{ij} - \min_j x_{ij}}{\max_j x_{ij} - \min_j x_{ij}}$$

This approach has a critical point the minimum and maximum values are extreme value outliers because of some mistake in data aggregation. For that reason, Standardization is more robust to such scenarios.

3.2 Validation

We divided the dataset into two different datasets called *training set* and *testing set*. We recall that the training set is used to train the algorithm and the testing set is used to measure the prediction power of the algorithm. Using the function `train_test_split` of `model_selection` class, we can easily divide the dataset **randomly**. This function will apply the *stratified* method introduced in the *Validation* section, ensuring that the training and testing set is well balanced. Setting the `test_size = 0.20` and `random_state = 42`, the shape of the two sets are: `X_test:(9043, 43), X_train: (36168, 43)`. With respect to parameter tuning, we will use cross-validation to assess it.

Classification scoring We will also use other score measures in addition to ROC Curve and the AUC. Here is a non-exhaustive list:

- **Precision**: The precision is the quantity $\frac{TP}{TP+FP}$. We recall TP is the number of true positives and FP the number of false positives. The precision could be seen as the ability of the classifier not to categorize as positive a sample that is negative
- **Recall**: The recall is the ratio $\frac{TP}{TP+FN}$ where FN the number of false negatives. The recall is the ability of the classifier to find all the positive samples.
- **F1-Score**: It can be interpreted as a weighted harmonic mean of the precision and recall. The scoring reaches its best value at 1 and worst score at 0.

3.3 GLM: Logistic regression

Running the logistic regression without any optimization, we have the following result:

Receiver Operating Characteristic Curve We recall that the AUC plots the False Positive Rate (FPR) versus the True Positive Rate (TPR) and allows identifying how good is the class discrimination: the higher the better, with the ideal model having a value of 1.0. With a base logistic regression:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=
    True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
```

We have:

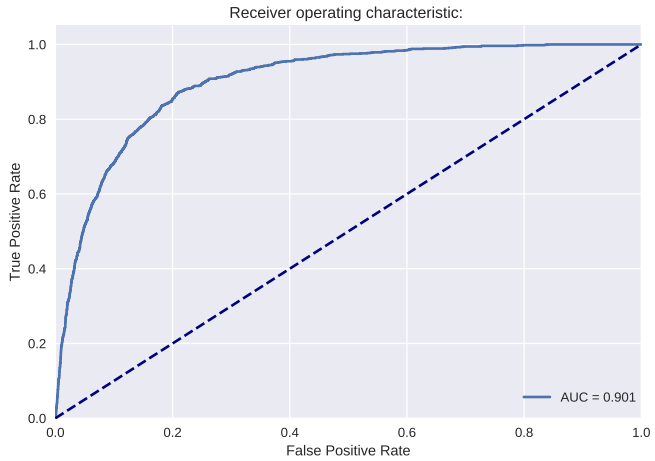


Figure 3.3: ROC Curve for a base logistic model. AUC: 0.90

We observe that the logistic regression is an encouraging outcome. It has a good predictive ability. However, we can improve a little bit this result using the selection model. Using step forward selection, we select only 31 features out to 40 based on AIC/BIC criterion and the AUC is 0.91. As we used probabilities for the predictions, thanks to ROC Curve we can fix a threshold to choose a class. Here the maximum of the ROC Curve is attained for $s = 0.21$. In overall, using other metrics as F1-Score and accuracy score, we have:

Model	F1-Score	Accuracy Score	Recall	AUC
Logistic regression	0.89	0.88	0.87	0.90

Table 3.4: Other metrics with the threshold $s = 0.21$

After few optimizations(adding regularization terms), we remark minor changes of predictive power for the Logistic regression. We will use this model as a basis to compare other models.

3.4 Bayes Classifiers

We want now to see how the Bayes classifiers perform on our dataset. We use the same training set and testing set as for the logistic regression. As one can guess, the Gaussian classifiers will not work with this dataset because it contains:

- Categorical

- Bernoulli
- Normal

Thus to tackle this problem, we use the very assumption of using Bayes classifiers, the features are independent. Consequently, we can do the following:

- Build a Bayes classifiers for each of the categoric features separately with dummy variable and multinomial Bayes.
- Build a Bayes classifier for all of the Bernoulli features at once
- Repeat the previous procedure for the Gaussian features

By the definition of independence, the probability for an example is the product of the probabilities of examples by these classifiers.

Thus, we have the following result: We find approximately the same AUC compared to the article

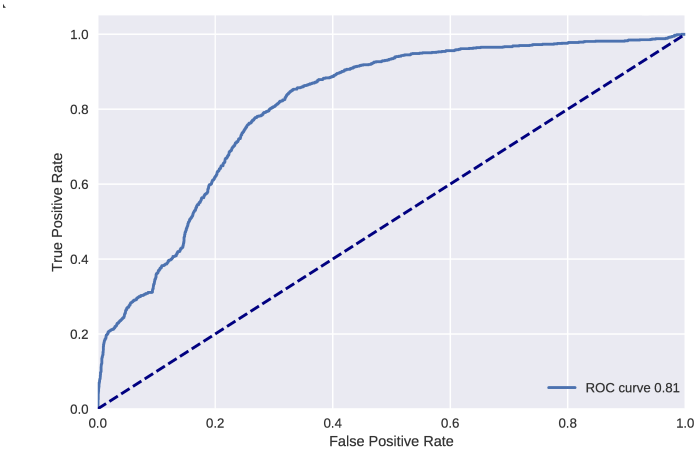


Figure 3.4: ROC Curve for Bayes classifiers. AUC: 0.81

[7]. Nevertheless, logistic regression overtakes Bayes classifiers.

3.5 Tree-Based methods

One of the vast property of tree based algorithm is that it does not need any preprocessing work on dataset as scaling, normalizing, creating dummy variables. We are going see the application of decision tree.

3.5.1 Decision Tree

Applying a base decision tree model:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=
    None,
max_features=None, max_leaf_nodes=None,
min_impurity_split=1e-07, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
presort=False, random_state=None, splitter='best')
```

As we can see, the classifier has many parameters. A small description is needed as we can going to try several values for these parameters.

- *Criterion*: The function used to measure the quality of a split. We have the choice between two criteria: `gini` for the Gini impurity and `entropy` for the information gain. (See (2.3))
- *Splitter*: The strategy used to split at each node. The strategies are: `best` to choose the best split and `random` to choose the best random split
- *Max depth*: The maximum depth of the tree.
- *min samples split*: The minimum number of samples to split. We will consider only binary trees. It will be fixed to `2`.
- *Max features*: The number of features to consider when looking for the best split. We have described this parameter in our model review.
- *Max leaf nodes*: Fix the number of a leaf node in best fashion way.

Another property of decision tree is that it allows us to visualise each node. The decision rules are easy to follow, they lead to a definite decision and enable the possibility to visualize how an instance falls into a class. For simplicity, we have limited the depth to 6.

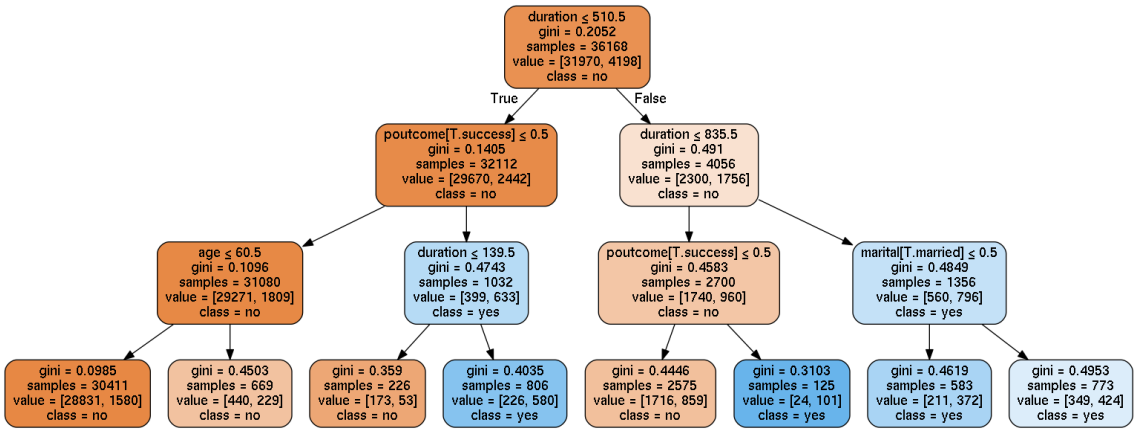


Figure 3.5: Tree visualization for `max_depth = 6`

The decision tree also gives us features which really counts in the decision process: Duration, poutcome (outcome of the previous marketing campaign), age.

In the base model, we plotted the ROC curve:

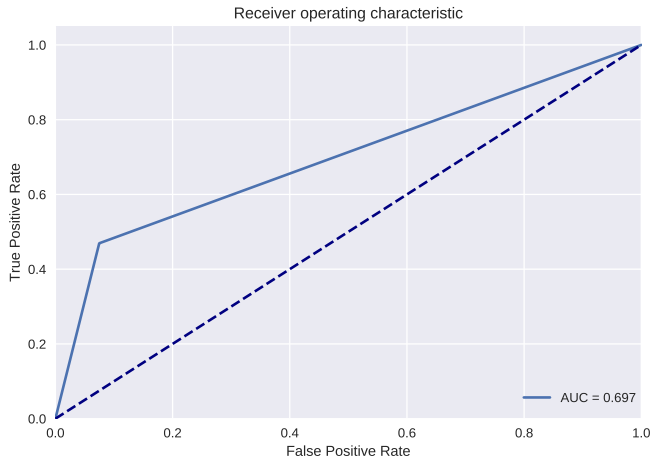


Figure 3.6: ROC curve for the Decision tree classifier with default parameter. AUC: 0.70

We remark the low performance of the decision tree compared to logistic regression. Before performing the optimization of the algorithm, let us see how the algorithm over-fit the training set.

As we know the number of samples required to populate the tree doubles for each additional level the tree grows to. In the following graph, we plotted multiple depths from 1 to 100 and we deduce that the optimal depth is 6, this process is called post-pruning. More the depth is greater more the algorithm over-fit the training set. However, we observe the depth growth lead to accuracy and AUC stagnation in the training set after depth = 25.

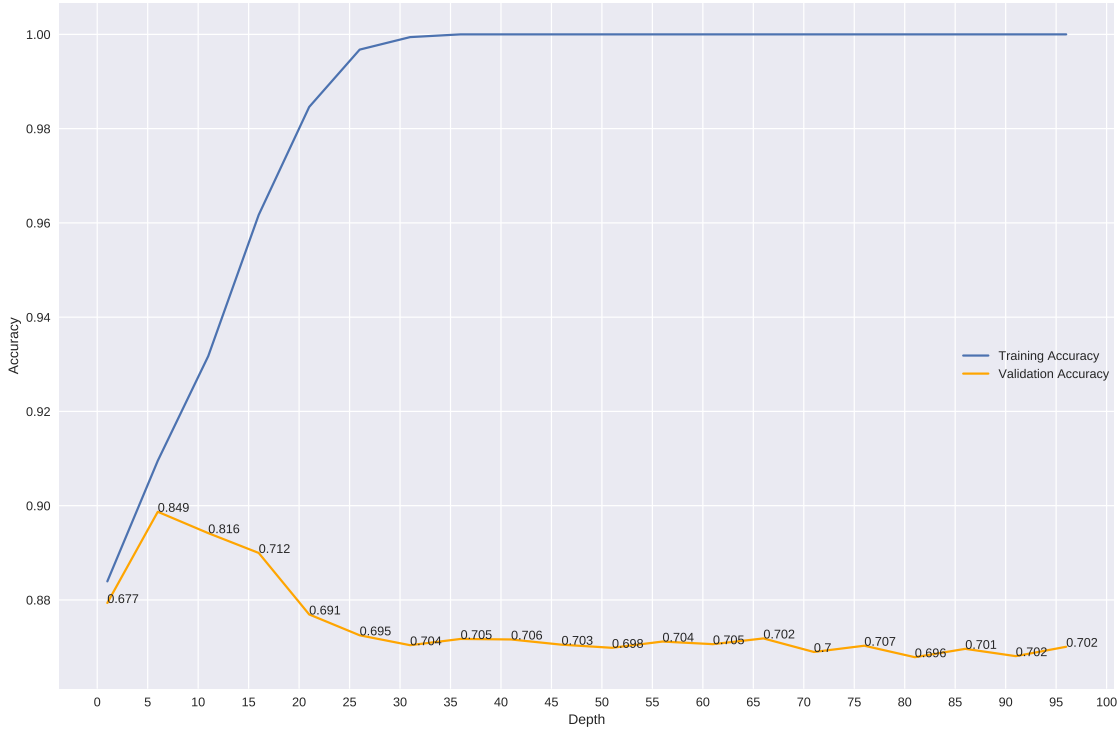


Figure 3.7: Evolution of the accuracy regarding the depth of the tree. Annotations are AUCs

Tuning Decision Tree algorithm Until now, we have set each parameter independently to others. The function `GridSearchCv` practice an exhaustive search over specified parameter values for an estimator. Cross-validated grid-search optimizes the parameters of the estimator over a parameter grid. Therefore, we found that the top parameters are:

```
top_params_tree = {'criterion': 'gini',
                   'max_depth': 8,
                   'max_features': 9,
                   'min_samples_leaf': 3,
                   'min_samples_split': 2,
                   'splitter': 'best'
                  }
```

We made significant improvement with the GridSearch function. It enables us to find the top parameter to optimize the predictive power of the tree classifier regarding a scoring function, in our case the AUC of the ROC curve.

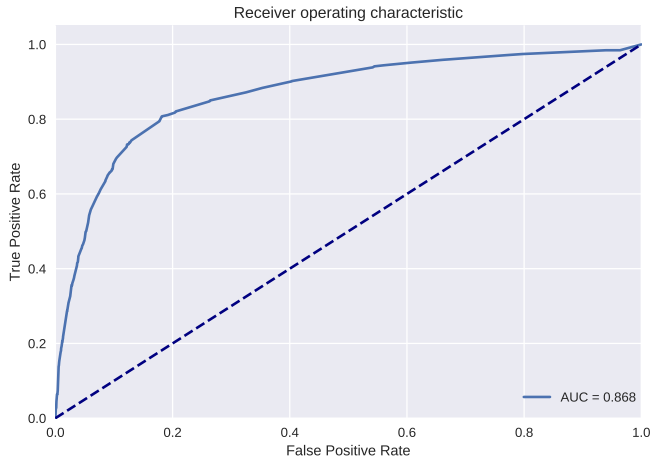


Figure 3.8: ROC curve for the Decision tree classifier with top parameters. AUC: 0.86

Model	F1-Score	Accuracy Score	Recall	AUC
Decision Tree	0.83	0.87	0.85	0.90

Table 3.5: Other metrics with the threshold $s = 0.20$

3.5.2 Random Forest

We move to another tree based classifier: Random Forest. We recall that Random Forests build an extensive collection of *de-correlated* trees, and then averages them. The essential idea is to average many noisy instances and, hence reduce the variance. Therefore, it supposes to give better results than decision tree. Random Forest enables us to see which features are the most used in the fitting process.

The previous graph shows the use of forests of trees to evaluate the importance of the feature. The blue bars are the percentage of feature importance along with their inter-trees variability.

Tuning Random Forest algorithm The performance of the Random forest with the default parameter is quite high. It does better than decision tree with an optimal parameter. Let us see if we can improve this with Grid search.

We found that top parameter are:

```
top_params_forest = {'bootstrap': False,
        'min_samples_leaf': 5,
        'n_estimators': 100,
        'min_samples_split': 2,
        'max_features': 8,
        'max_depth': 9}
```

We want to highlight GridSearch is very costly in terms of CPU computations because it tries each combination of parameters for different values. Having an idea of the range of the parameter could make easier the research and even more when we have many parameters. For instance, we saw in Figure (3.7) the AUC is acceptable when the depth is from 2 to 15.

Model	F1-Score	Accuracy Score	Recall	AUC
Random Forest	0.89	0.88	0.85	0.91

Table 3.6: Other metrics with the threshold $s = 0.20$

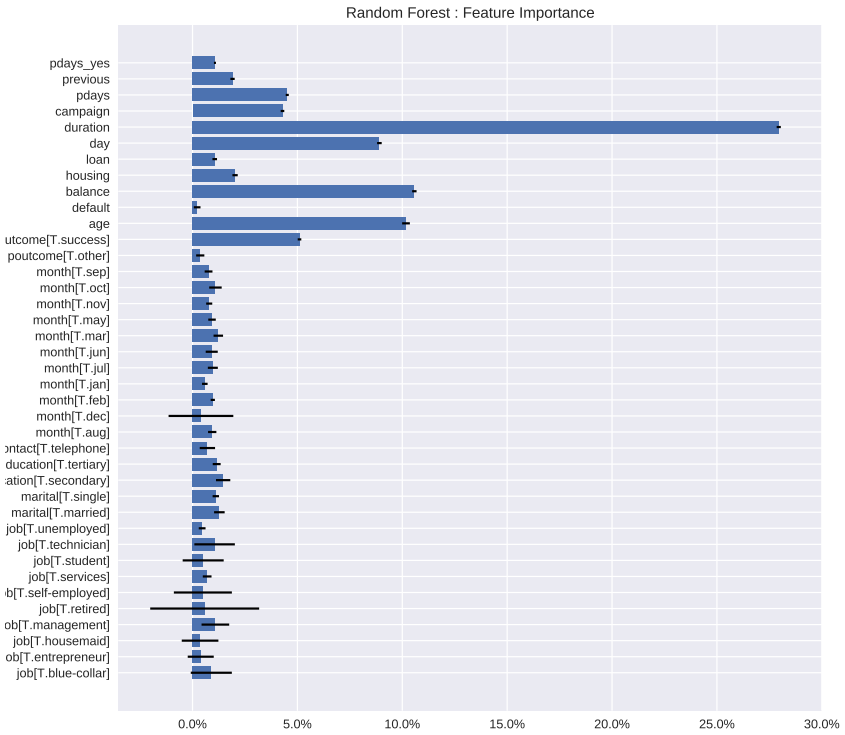


Figure 3.9: Features Importance in Random Forest

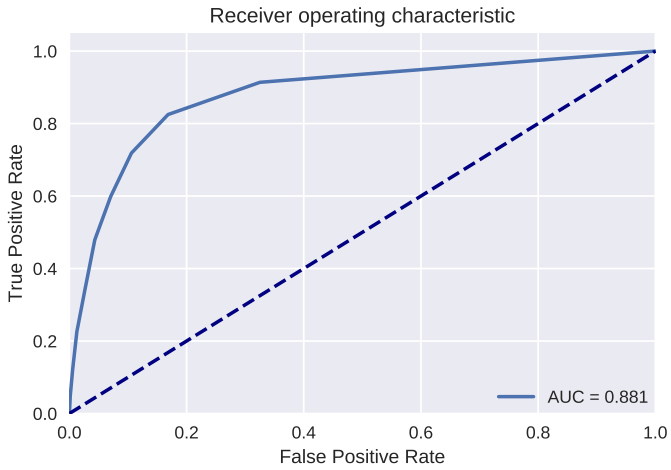


Figure 3.10: ROC curve for Random forest classifier with default parameter. AUC: 0.88

3.6 Support Vector Machine

In the article [7], the authors have observed that SVM is the most performing algorithm. In this section, we will verify their result. However, we need to perform some preprocessing work on dataset before giving it to the algorithm. In fact, all practical implementations of SVMs have strict requirements for training and testing (prediction). The first requirement is that the data should be numerical. Furthermore, Support Vector Machine algorithms are not scale-invariant, so it is highly recommended to scale the data. The scaling is done using the function `StandardScaler` present in the class `preprocess`. reduce The base model is created via the function:

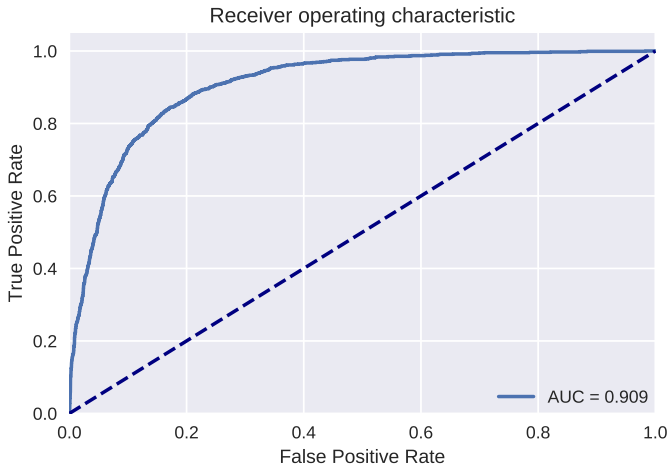


Figure 3.11: ROC curve for the Random forest classifier with top parameter. AUC: 0.91

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
max_iter=-1, probability=True, random_state=None, shrinking=True,
tol=0.001, verbose=False)
```

Without any optimization, we have the following ROC Curve:

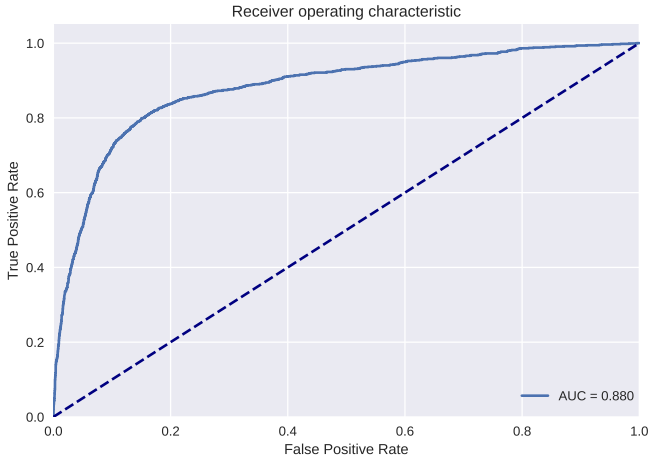


Figure 3.12: ROC curve for the SVC. AUC: 0.86

The main goal of SVM is to separate the data into different sets. We reduce therefore the dimension of the data to 2 in order to visualize the result. As already outlined earlier in Figure (3.9), two features explained at least 40% of the entire dataset: *duration, balance*. The important features found in the random forest are almost the same that we get from the forward selection.

The result is acceptable but it could be improved by choosing the correct parameter. Indeed, we may have chosen the wrong kernel (Gaussian kernel for the model above) or a rough regularization parameter. That brings us, for the optimization part.

Parameter tuning Similar to any machine learning algorithm, we need to choose/tune hyperparameters for these models. The relevant parameters to tune here are: C, the penalty parameter or the error term. We have seen in the model review this acts as a regularization parameter for SVM and γ for kernel coefficient (only for `'rbf'`, `'poly'` and `'sigmoid'` kernels). In above example, we used a default value of γ.

First, let us understand the impact of C and γ parameters have on SVM models. As seen in Figure(2.7), we find that higher value of γ, it will try to fit exactly the training set, *i.e.* generalization error and cause over-fitting issues. The parameter C controls the trade-off between smooth decision boundary and classifying the training instances correctly.

In the following graph, we observe ROC Curve with the optimized SVM classifier:

The performance of this model is very encouraging. We almost achieved the same result given on the paper [7].

Model	F1-Score	Accuracy Score	Recall	AUC
SVM	0.87	0.87	0.85	0.90

Table 3.7: Other metrics with the threshold $s = 0.20$

3.7 Neural Networks

SVMs results are very comforting and until now it is the best algorithm suited for our dataset. Neural Networks are another kind of supervised learning as we have seen in the model review. We will see in this section how well neural networks perform and if it can beat SVM. We recall that our model is built on **Keras**. Keras allows you to quickly and simply design and train the neural network and deep learning models.

After creating the model, we are going to use scitkit-learn to evaluate the model using **stratified K-fold cross validation** rather than a splitting the dataset into two sets. We recall that stratified K-fold is a resampling technique that provides an estimate of the performance of the model as the ROC curve. It results in a robust estimate of performance *Stratified* means that it will look at output values and balances the number of instances that belong to each class. It already the case using the function `train_test_split`.

We begin by defining the baseline model. Our model will have only a single hidden layer with all the neurons connected. The weights are initialized using a Gaussian random number. We use rectifier activation function. In the last layer, we use a sigmoid function in order to produce a probability output in the range of 0 to 1 that can easily and automatically be converted to probabilities. We can refer to Figure (**??**) where there will be 39 activation units in the hidden layer.

Furthermore, we used a logarithmic loss function (binary_cross-entropy) during training, the preferred loss function for binary classification problem. In our computation we fixed $K = 10$, thus the model will be created ten times for the 10-fold cross validation being performed.

As we did for SVM, we need to perform data preparation and an effective data preparation scheme when building neural networks is standardization done by the class `StandardScaler`. The data is then rescaled such that the mean-value for each feature is 0 and the standard deviation is 1. This preserves Gaussian and Gaussian-like distributions.

It is better in practice to train the model using standardization data using cross-validation rather than performing the standardization in the entire data. This makes Standardization a significant step in data preparation in the cross-validation process and it prevents the algorithm having the knowledge of "unseen" data during validation.

Model	F1-Score	Accuracy Score	Recall	AUC
Neural Network	0.90	0.88	0.87	0.91

Table 3.8: Other metrics with the threshold $s = 0.21$

Tuning Neural Network algorithm There are many parameters to tune on a neural network, such as activation functions, optimization methods weights, the initialization and so on.

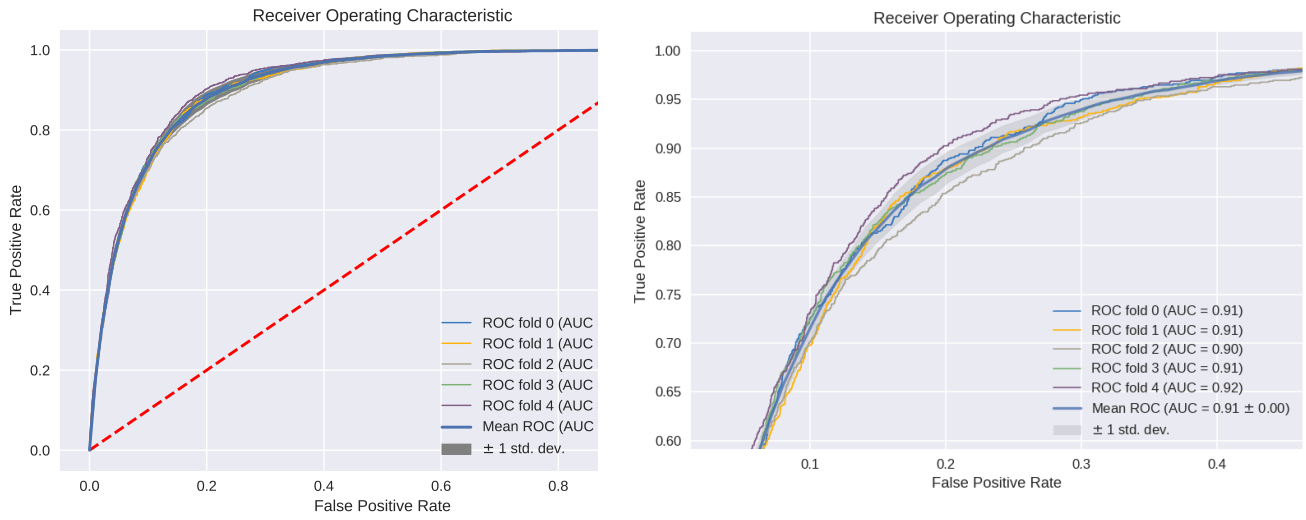


Figure 3.13: ROC curve achieved with cross-validation $K = 5$

One aspect that may have an out-sized effect is the structure of the network. In this section, we take a look at two experiments on the structure of the network:

Making the network smaller In the first run of the model, we set 39 activation units as of the dataset dimension. In order to reduce the redundancy in the input variable, we can force a type for feature extraction by restraining the representational space in the first hidden layer. Making a drastic reduction of number unit in the hidden layer will put pressure on the network during training to extract the most important structure in the input data to model.

Running this example provides by cross-validation the following result: Accuracy = 89.47% (0.35%).

We can see that we have a very slight improvement in the mean estimated accuracy and an important reduction in the standard deviation of the accuracy scores.

This is a great result because we are doing slightly better on a network of half the size, which in turn takes half the time to train.

The number of layer increment More layers in neural network offer more opportunity for the network to extract key features and recombine them in useful non-linear ways. In Keras adding layers is quite easy (one line). The following is the result is got by adding one new layer to the network that introduces another hidden layer with 30 neurons after the first hidden layer. The idea here is to give a chance to the network to stock important information in a large space (compared to the number of features) before being bottlenecked and forced to divide the representational capacity, much like we did in the experiment above with the smaller network.

Running this example provides by cross-validation the following result: Accuracy = 91.25% (0.38%).

3.8 Summary

In overall terms, the results of all the classifiers that we have encountered were very satisfying. But beyond this results, there are some aspects that we have to take in account in order to compare one model against another. First, each algorithm needs some preprocessing work. For instance, SVM and Neural networks need scaled inputs to ensure good performance while Random Forests or Decision Tree do not need this sort of work.

Furthermore, the fitting process could be time-consuming and costly in terms of computations for some model that we saw even if they perform very well. After running multiple models from

GLM, we observed a straightforward fitting process. This may be explained in part by the stringent constraints of GLM models. This is also the case with Bayes Classifiers. SVM and Neural Networks require vast computing power. The complexity for SVM can oscillate between a squared and a cubic term in the number of examples depending on how we choose the regularization parameter. The complexity of Neural Networks depends mostly on the structure of the network. A large hidden layer or an important number of activation units could increase drastically the computations. Furthermore, we find out that the calibration process for Logistic regression and Bayes classifiers are faster than for other methods. The number of parameters for other models as random forests or svm is high, so finding the good combination of parameters which maximize the accuracy may be time consuming.

	Logistic regression	Naïve Bayes	Decision Tree	Random Forest	SVM	Neural Network
F1-score	0.89	0.83	0.89	0.89	0.87	0.90
Recall	0.88	0.80	0.89	0.87	0.90	0.89
Accuracy	0.88	0.89	0.87	0.88	0.88	0.90
AUC	0.90	0.81	0.90	0.91	0.91	0.92

Table 3.9: Performance of each algorithm for different measures

Finally, we summarize the performance related to the predictive power of each model in Table (**??**). We observed that Neural Networks performed for each measure. However, we have done an important work of calibration for this model. Finding the corresponding structure of the network was not straightforward. We think that using a single model to forecast the response variable is not sufficient. One of the reasons is that each algorithm has its own particularities and offer different features, for instance Random Forests could give a list of important features.

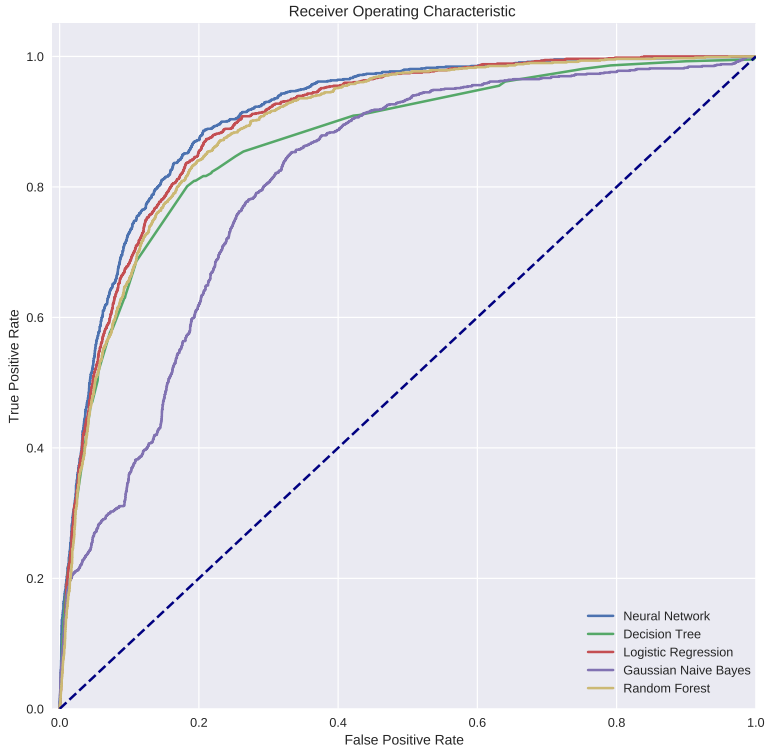


Figure 3.14: ROC Curves for all the models

Conclusion and perspectives

By and large, Machine Learning is still an open field of research for which many questions are still left unanswered. In this report, we have revisited multiple algorithms, consistently calling into question each and every part of these algorithms, in order to shed new light on their learning capabilities.

In first part, we reviewed these algorithms from the more simple towards the more complex in the context of classification and regression tasks. In particular the case of tree based algorithms, our analysis reviewed assignment rules, stopping criteria and splitting rules, theoretically motivating their design and purpose whenever possible. We have also seen fundamental methods in Machine Learning as SVM and Neural Networks. The theory behind SVM has been deeply analysed, hence showing their good computational performance and scalability to larger problems. Finally, the first part of this work has been concluded by the review of Neural Network, highlighting and discussing considerations that are critical as for the over-fitting issues, yet easily over-looked, for guaranteeing good computational performance.

In part II of this report, we analysed and discussed classifications methods using Machine Learning techniques, based on publicly available financial data, with the aim of addressing the problem of bank direct marketing campaigns. In particular, we used financial and recent data from a Portuguese bank. In effect, each algorithm that we reviewed has proven to be of great value, since obtained predictive performance increased. Using a sensitivity analysis, we measured the importance of each feature, and such knowledge can be used by managers to improve campaigns (*e.g.* scheduling campaigns to specific months). Machine Learning is already widely employed in the Pharmaceutical and Medical Sciences, Robotics, Oceanography, Image Recognition and numerous other domains, but still the applications in Finance is quite recent. Our top three performing classifier families were the Neural Network, the Support Vector Machine and Logistic regression, a result which is consistent with Machine Learning classification results using non-financial data reported in [2].

There are a number of further directions can be investigated starting from this project. The first one is to explore other creative and effective methods that might yield even better performance on direct marketing. Second, we might enhance the data by collecting more client based data, in order to check if high-quality predictive models can be achieved without contact - based information.

In conclusion, Machine Learning algorithms should not be considered as a black-box tool, but as a methodology rooted by a rational thought process that is entirely reliant on the problem we are trying to solve.

Appendix

Further data description

Outlier:

Data_point > (Q3 * 1.5) **is** said to be outlier where Q3 **is** 75% Quantile !

Age:

* Average age **in** the dataset **is** ~41 with std of 10.61
* Min. age **is** 18
* Max. age **is** 95
* quantile 75%(percentile) refers that 75 percentage have 49 **or** less age.
* As the maximum age **is** 95, there **is** great chance that its a outlier "49*(3/2) = 73.5". So anything greater than 73.5 **is** outlier.

Balance:

* Average balance **in** the dataset **is** 1528.53 with std of 3255.41. The standard deviation **is** quite huge, it shows the wide spreading accross the dataset.
* Min. balance **is** −6847
* Max. balance **is** 81204
* quantile 75%(percentile) refers that 75 percentage of the people have 1708 **or** less balance.
* **while** comparing with 75% quantile, 81204 **is** very huge **and** its a outlier data point.

Duration:

* Average duration of the people speaking **in** the dataset **is** (approx)371 with std of 347, as standard deviation **is** quite huge it means that duration **is** wide spread across the dataset.
* Min. duration **is** 2
* Max. duration **is** 3881
* quantile 75%(percentile) refers that 75 percentage of the people spoke **for** 496 seconds **or** less.
* **while** comparing with 75% quantile, 3881 **is** a outlier data point .

Pdays:

* Average no. of days passed after the client was contacted **from** previous campaign **in** the dataset **is** (approx)51.33 with std of 108.75.
* Min. pdays **is** −1
* Max. pdays **is** 854
* quantile 75%(percentile),**for** 75% of records it **is** 20.75 days, which means the Client was frequently contacted.

Campaign:

* Average no. of contacts performed during the current campaign **for** a client **in** the dataset **is** (approx)2.50 with std of 2.72.
* Min. balance **is** 1
* Max. balance **is** 63
* quantile 75%(percentile),**for** 75% of records, 3 times the client has been contacted **in** the current campaign **for** a client.
* **while** comparing with 75% quantile,63 **is** a outlier data point.

Previous:

* Average no. of contacts performed before this campaign **for** a client **in** the dataset **is** (approx)0.83 with std of 2.29.
* Min. balance **is** 0.
* Max. balance **is** 58
* quantile 75%(percentile),**for** 75% of records, 1 times the client has been contacted before this campaign.
* **while** comparing with 75% quantile,58 **is** a outlier data point.

A more detailled description of the data:

Accepted deposit

	age	balance	day	duration	campaign	\
count	5289.000000	5289.000000	5289.000000	5289.000000	5289.000000	
mean	41.670070	1804.267915	15.158253	537.294574	2.141047	
std	13.497781	3501.104777	8.501875	392.525262	1.921826	
min	18.000000	−3058.000000	1.000000	8.000000	1.000000	
25%	31.000000	210.000000	8.000000	244.000000	1.000000	
50%	38.000000	733.000000	15.000000	426.000000	2.000000	
75%	50.000000	2159.000000	22.000000	725.000000	3.000000	
max	95.000000	81204.000000	31.000000	3881.000000	32.000000	

	pdays	previous	y
count	5289.000000	5289.000000	5289.0
mean	68.702968	1.170354	1.0
std	118.822266	2.553272	0.0
min	−1.000000	0.000000	1.0

25%	−1.000000	0.000000	1.0
50%	−1.000000	0.000000	1.0
75%	98.000000	1.000000	1.0
max	854.000000	58.000000	1.0

Rejected deposit

	age	balance	day	duration	campaign	\
count	39922.000000	39922.000000	39922.000000	39922.000000	39922.000000	
mean	40.838986	1303.714969	15.892290	221.182806	2.846350	
std	10.172662	2974.195473	8.294728	207.383237	3.212767	
min	18.000000	−8019.000000	1.000000	0.000000	1.000000	
25%	33.000000	58.000000	8.000000	95.000000	1.000000	
50%	39.000000	417.000000	16.000000	164.000000	2.000000	
75%	48.000000	1345.000000	21.000000	279.000000	3.000000	
max	95.000000	102127.000000	31.000000	4918.000000	63.000000	

	pdays	previous	y
count	39922.000000	39922.000000	39922.0
mean	36.421372	0.502154	0.0
std	96.757135	2.256771	0.0
min	−1.000000	0.000000	0.0
25%	−1.000000	0.000000	0.0
50%	−1.000000	0.000000	0.0
75%	−1.000000	0.000000	0.0
max	871.000000	275.000000	0.0

Proof of theorem 2.6.1

[$\Leftarrow$] is trivial. Let us show that: $\forall\ (\mathbf{x}, \tilde{\mathbf{x}})\ k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle \Phi(\mathbf{x}), \Phi(\tilde{\mathbf{x}}) \rangle$ is definite positive. Let $\alpha \in \mathbb{R}^d$, we have:

$$\begin{aligned} \alpha^T K \alpha &= \sum_i \sum_j \alpha_i \alpha_j \langle \Phi(\mathbf{x}_i), \mathbf{x}_j \rangle \\ &= || \sum_i \alpha_i \Phi(\mathbf{x}_i) ||^2 \geq 0 \end{aligned}$$

The symmetry of this inner product follows form the symmetry of K.
[$\Rightarrow$]

1. Let $\mathcal{F}_0$ subspace spanning by all the vectors $k(\cdot, \mathbf{x}), \mathbf{x} \in \mathcal{X}$.
2. we define an inner product on $\mathcal{F}_0$ by: $\langle \sum_i \alpha_i k(\cdot, \mathbf{x}_i), \sum_j \beta_j k(\cdot, \mathbf{y}_j) \rangle = \sum_{i,j} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j)$. It is equivalent to define an inner product on each generated elements by $\langle k(\cdot, \mathbf{x}), k(\cdot, \mathbf{y}) \rangle$ This latter is of course an inner product because:
   - k is bilinear and symmetric on $\mathcal{F}_0$
   - Let $f \in \mathcal{F}_0$, $f = \sum_i^n \alpha_i k(\cdot, \mathbf{x}_i)$ and $||f||^2_{\mathcal{F}_0} = \sum_{i,j} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0$
   - Obviously we have $||f||_{\mathcal{F}_0} = 0 \Rightarrow f = 0$. Thanks to Cauchy-Swartz inequality we have $|f(x)| \leq ||f||_{\mathcal{F}_0} k(\mathbf{x}, \mathbf{x})^{\frac{1}{2}}$

Proof of theorem 2.6.3

Let $f \in H$ and $H_D = \text{span}\{\sum_i \alpha_i k(\cdot, \mathbf{x}_i) | \alpha \in \mathbb{R}^n\}$. Let $f_H \in H_D$ and $f_\perp \in H_D^\perp$ such that $f = f_D + f_\perp$. Then:

$$\forall\, i, \quad f(\mathbf{x}_i) = f_D(\mathbf{x}_i) + f_\perp(\mathbf{x}_i)$$

with $f_\perp(\mathbf{x}_i) = \langle f_\perp, k(\cdot, \mathbf{x}_i)\rangle = 0$. Using the Pythagoras theorem, we have:

$$\|f\|_H^2 = \|f_D\|_H^2 + \|f_\perp\|^2 \tag{1}$$

Thus we have the following:

$$\begin{aligned} J(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n), ||f||_H^2) &= J(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n), \|f_D\|_H^2 + \|f_{perp}\|^2) \\ &\geq J(f(\mathbf{x}_1)_D, \ldots, f(\mathbf{x}_n)_D, \|f_D\|_H^2) \end{aligned}$$

Therefore

$$\inf_{f \in H} J(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n), ||f||_H^2) = \inf_{f \in H_D} J(f(\mathbf{x}_1), \ldots, f(\mathbf{x}_n), ||f||_H^2)$$

Bibliography

[1] L. Breiman. Random forests. *Statistics Department, Berkeley*, 2001.

[2] Delgado. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research 1*, 2014.

[3] Dietterich. Machine learning. in: Nature encyclopedia of cognitive science. *Macmillan*, 2011.

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2015.

[5] M. A. Hall I. H. Witten, E. Frank and C.J. Pall. *Data Mining, Practical Machine Learning Tools and Techniques*. Elsevier, 2017.

[6] A. Regaldo. The data made me do it. *MIT Technology Review*, 2013.

[7] P. Cortez S. Moro and P. Rita. A data-driven approach to predict the success of bank telemarketing. decision support systems. *Elsevier*, 2014.