UNIVERSITÀ DEGLI STUDI DI GENOVA

DIPARTIMENTO DI MATEMATICA

Machine Learning Techniques Applied to Telecommunication Data

Stefania Cazzato

Supervisors:
Prof. Anna Maria Massone
Dr. Matteo Santoro

Co-supervisor:
Prof. Michele Piana

Academic Year 2017/2018

To my mother and my father,
who keep on fighting against monsters.

Contents

Introduction

Machine Learning is one of the richest and widest field of studies throughout the world. Its applications range from speech recognition [1] to computer vision [23], from medical diagnosis [19] to robot control [21]. In the field of finance and business, it has been successfully applied to the stock exchange market [16], to energy and consumption analyses [2], to customer data [17] and to many other areas. Nowadays, this trend is accelerating, thanks to the increasing amounts of data becoming available, the higher computer processing power and the development of modern deep learning models.

In this thesis, Machine Learning has been mainly applied to data belonging to a specific business area: telecommunication networks. Indeed, this thesis is the result of an industrial PhD that was carried out at the premises of Camelot Biomedical System srl (https://www.camelotbio.com), a software company specialized in business and professional counseling. Two main problems have been passed by three different telecommunication companies (commonly referred to as TELCOs) to Camelot Biomedical Systems:

- *Churn prevention*. The main goal of churn prevention is to identify customers who are about to transfer their business to a competitor operator. Churn is an important issue for wireless telecommunication industry, because it can have damaging consequences for the sales of a company, leading to an increased need of attracting new customers;

- *Predictive Maintenance*. In telecommunication domain, predictive maintenance aims at preventing the break down of the global infrastructure providing access

to the mobile network of the company. Maintenance costs have a deep impact to the financial wealth of a company, which is forced to expensive last-minute repairs, as well as providing a disservice for its customers in the event of a break down. For these reasons, an efficient system of predictive maintenance is desired by most of the telecommunication companies.

My thesis work has addressed the two problems described above and proposes solutions that are the result of the activity done to select the most appropriate Machine Learning methods when applied to the analysis of this kind of data.

The data that are analyzed in this thesis can be considered as an example of *Big Data*, both for their dimension, which ranges to some hundreds of MB to dozens of GB, and variety. Both the collection and storage of such data, commonly referred to as *data ingestion*, played an important role in the development of the thesis. Once the data were ingested, a great effort has been devoted to the *modeling* of the problem and the search of a suitable *learning algorithm* that could provide good predictions. Both the churn prevention and the predictive maintenance problems have been modeled as *binary classification* problems, and the corresponding most suitable learning algorithms have been selected through an operation of *model assessment* comprising a wide range of learning algorithm families, all belonging to the world of Supervised Machine Learning: from Ridge regression [15] to kernel Support Vector Machines [8], from random forests [20] to gradient boosting [10].

The performed are integral part of three different projects carried out for as many telecommunication companies (two of them concerning churn prevention and one of them concerning predictive maintenance). For privacy reasons, the name of the companies has to remain anonymous, and the content of the data can not be directly reported. The final goals of these projects have been the delivering of the resulting predictions, which are used by the TELCOs to enhance their business plans, as well as the writing of a collection of technical reports.

From a more scientific point of view, this thesis shows that the application of Machine Learning techniques to a huge amount of data, specifically concerning the prob-

lems of churn prediction and predictive maintenance in the telecommunication field, can lead to meaningful and exploitable results. In different occasions, predictions produced by this work were given to the TELCOs with monthly occurrence and were used by the companies in their promotional campaigns.

This thesis is structured as follows:

- The first chapter gives the mathematical foundations of *Supervised Machine Learning*, introducing the *statistical learning framework* and its main features, the theory behind four families of *machine learning algorithms* and the most important learning steps that were followed in the learning problems described in the next chapters.

- The second and the third chapters are devoted to the projects related to *churn prevention*. In particular, in the project described in the second chapter the goal is to train a learning algorithm capable of predicting the customers about to churn to a competitor operator. The chapter describes the data given by the TELCO to perform the churn prevention, the preliminary analyses performed on this data and the mathematical modeling of the problem, as well as the different experimental settings on which the learning algorithms were run. The second project focused on churn prevention is described in the third chapter. A new set of data was provided for the learning problem, and many experimental settings were tested with as many learning algorithms. Again, the chapter provides an overview of the dataset, the evidences given by the preliminary analyses on the data and a description of all the experiments undertaken to retrieve the lists of future churners.

- Finally, the fourth chapter is focused on the project related to *predictive maintenance*. This project was carried out in two different steps, with a significant increasing in the amount of data provided by the TELCO between the first and the second run. The chapter describes the data provided for both these subprojects, as well as the undertaken preliminary analyses and the mathematical modeling of the problem. It provides also the corresponding results of both the subpro-

jects, and the way the first subproject's outcome influenced the definition of the experimental setting during the second one.

The final chapter provides the conclusions and possible developments deriving from this thesis.

As a final remark, I would like just to mention the fact that during my PhD activity I had the opportunity to verify the power and versatility of Machine Learning techniques in different application fields, as well as telecommunications. In particular I contributed to the development of a machine learning-based system for the prediction of energy consumptions in market and stores chains, and I have used pattern recognition techniques to perform cell detection in biological microscope images.

Chapter 1

Supervised machine learning

'*Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed*' is a most famous quote from Arthur Samuel in 1959 [24] that provides the first definition to the concept of *Machine Learning*. It was a general definition, implying a question: what does it mean that a computer *learns*?

Tom Mitchell answered this question in 1997 [22], by stating: '*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E*'.

This definition is the base of all the mathematical theories that represent the foundations of Machine Learning. There are many different types of Machine Learning techniques. One crucial distinction must be drawn between *supervised* and *unsupervised* learning.

In *Supervised Learning* the machine is *trained* on a pre-defined set of *training examples*, which then facilitate its ability to reach an accurate conclusion when given new data. Conversely, in *Unsupervised Learning* the machine is given a bunch of data and must find patterns and relationships therein, without previous examples to learn from.

This thesis focuses on *Supervised Machine Learning* techniques, as they have been used to solve the learning problems described in the next chapters. For a good review of Unsupervised Machine Learning algorithms refer to [3], [13] and [14].

This chapter provides the mathematical theory under Supervised Machine Learn-

ing. In Section 1.1 the main objects of this theory, such as the *learning algorithm* and the *training data*, are given, followed by the definitions of the most used learning paradigm (i.e. Empirical Risk Minimization) and some abstract concepts such as *learnability*. The next sections provide the most commonly used algorithmic approaches to Supervised Machine Learning, i.e. the *linear predictors* (Section 1.2), the *Support Vector Machines* (Section 1.3) and a brief explanation of the concepts behind the *ensemble methods* (Section 1.4), as well as the *decision tree* algorithms (Section 1.5). Section 1.6 describes the procedure known as *model assessment*. Finally, Section 1.7 explicits the procedure followed throughout the next chapters and gives the list of specific learning algorithms used during this thesis. Most of the material is derived from [3] and [25].

1.1 Basic concepts

The problems described in this thesis belong to the world of supervised machine learning. From a mathematical point of view, supervised learning falls under the *statistical learning framework*. The main object is a *learning algorithm* that, from a set of 'labeled training data', is able to predict the correct targets for unseen instances and thus generalizes beyond the training data.

Formally, a learning algorithm can be seen as a map $\mathcal{A} : U_{n\in\mathbb{N}}(\mathcal{X}\times\mathcal{Y})^n \to \mathcal{Y}^{\mathcal{X}}$ that locates a function $\tilde{f} : \mathcal{X} \to \mathcal{Y}$ with *domain set* $\mathcal{X}$ and *target set* $\mathcal{Y}$. When the target set $\mathcal{Y}$ is a *discrete* set, the problem addressed by the learning algorithm is called a *classification problem*; when $\mathcal{Y}$ is *continuous*, it is called a *regression problem*. The special case of $\mathcal{Y} = \{0, 1\}$ is called *binary classification*.

The input and output of a learning algorithm can be defined as this:

- input: **Training data**. It is a finite sequence $S = ((x_1, y_1), \ldots, (x_n, y_n))$ of pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. Usually the points $x_i \in \mathcal{X}$ are called *input variables* or *features*, and $y_i \in \mathcal{Y}$ are referred to as the *targets* (or *labels* in case of classification) associated to x_i. The pairs (x_i, y_i) belonging to the training set are treated as values of random variables $(\mathcal{X}_i, \mathcal{Y}_i)$ that are identically and independently dis-

tributed according to some probability measure P over $\mathcal{X} \times \mathcal{Y}$. It is assumed that P governs a distribution D of both the training data and the future, yet unseen instances of data point (x, y).

- output: **Learner**. It is a function $h : \mathcal{X} \to \mathcal{Y}$, that aims at predicting the correct target $y \in \mathcal{Y}$ for an arbitrary input $x \in \mathcal{X}$. It is usually referred to as *hypothesis*. The hypothesis returned by a learning algorithm upon receiving the training data S is denoted as $\mathcal{A}(S)$.

To measure the distance between $h(x)$ and the respective y the *generalization error* of h must be defined. Intuitively, the generalization error is the probability that, given an instance x generated from $\mathcal{X}$ by the probability distribution D, the label $h(x)$ does not equal the value of $f(x)$. Formally:

Def. 1.1.1 (Generalization error)**.** The *generalization error* of a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ with respect to a probability distribution D over $\mathcal{X} \times \mathcal{Y}$ is defined as:

$$L_D(h) = \int_{\mathcal{X}\times\mathcal{Y}} L(y, h(x))dP(x, y) \tag{1.1}$$

where P is the probability measure associated to D and $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ a chosen *loss function*.

In a classification problem, Eq. 1.1 can be written as:

$$L_D(h) := \mathbb{P}_{x\sim D}[h(x) \neq y] = D(\{x \ : \ h(x) \neq y\}). \tag{1.2}$$

The generalization error is also known as *risk* or *true error*. The loss functions L that can be chosen vary according to the nature of the learning problem, being it a classification or regression problem. For classification problems a common choice is the *0-1 loss function* defined as

$$L(y, y') = 1 - \sigma_{y,y'}. \tag{1.3}$$

In regression problems, if $\mathcal{Y} = \mathbb{R}$ a common choice is the *squared-loss function*, i.e.

$$L(y, y') = |y' - y|^2. \tag{1.4}$$

The aim of a learning algorithm is to retrieve the hypothesis h which *minimizes* its generalization error without knowing a priori the probability distribution D and the related probability measure P.

1.1.1 Empirical Risk Minimization and Overfitting

Since the probability distribution D and the related probability measure P are not known a priori, the generalization error of a given hypothesis h defined in Eq. 1.1 cannot be computed directly.

The only data known to the learning algorithm are the training examples belonging to the training set S. As a consequence, a new measure of the error associated to the hypothesis y, called the *empirical error*, can be defined:

Def. 1.1.2 (Empirical error)**.** The *empirical error* of a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ on a given training set $S = ((x_1, y_1), \ldots, (x_n, y_n))$ according to a chosen loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ is defined as:

$$L_S(h) := \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(x_i)). \tag{1.5}$$

As before, the empirical error is also known as *empirical risk*.

The approach of minimizing the empirical error $L_S(h)$ is called *Empirical Risk Minimization*, or ERM for short.

The ERM paradigm is very natural, however it may not lead to good results. The following classification example, taken by [25], shows a case of an hypothesis that performs well on the training set but is associated to a poor true error.

Let us assume that $\mathcal{X}$ is the square shown in Figure 1.1 and the data points are labelled 1 inside the blue square, and 0 otherwise. The area of the inner square is 1, while the global area of the outer square is 2.

Taking an arbitrary training set S of points belonging to $\mathcal{X} \times \mathcal{Y}$, an hypothesis h that minimizes the empirical error is the following:

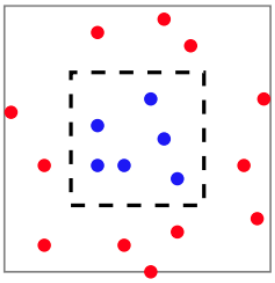


Figure 1.1: Example of binary classification on a given $\mathcal{X}$. The data points inside the inner square are labelled 1, the other ones 0. Picture taken from [25].

$$h_S(x) = \begin{cases} y_i & \text{if } \exists x_i \in S_{\mathcal{X}} \text{ such that } x_i = x, \\ 0 & \text{otherwise} \end{cases}$$

Clearly, the empirical error $L_S(h_S)$ is always equal to 0, so this hypothesis could be chosen by an ERM algorithm. However, this hypothesis returns label 1 only on a finite set of points, thus the true error $L_D(h_S)$ is equal to 1/2.

When the difference between the true error associated to an hypothesis h, $L_D(h)$, and the empirical error $L_S(h)$ is large it is said that the learning algorithm $\mathcal{A}$ suffers of *overfitting*: intuitively, the hypothesis returned by the ERM algorithm is too focalized on the training set S, and it is not able to *generalize* to other points belonging to the space $\mathcal{X} \times \mathcal{Y}$.

A common way to overcame this problem is to restrict the learning algorithm to a subspace of $\mathcal{Y}^{\mathcal{X}}$, i.e. to a specific set of predictors $\tilde{f} : \mathcal{X} \to \mathcal{Y}$. This set is called an *hypothesis class* and it is usually defined as $\mathcal{H}$. Restricting a learning algorithm to a given class $\mathcal{H}$ means that the output hypothesis $h = \mathcal{A}(S)$ must belong to $\mathcal{H}$ and must minimize the empirical error over S. Formally

$$h = \mathcal{A}_{\mathcal{H}}(S) \in \operatorname*{argmin}_{\tilde{f} \in \mathcal{H}} L_S(h)$$

This restriction is usually referred to as *inductive bias*. The class $\mathcal{H}$ should be based on some a priori information concerning the problem, and ideally it should be a class

where the overfitting can not occur.

When there exists an hypothesis $h^* \in \mathcal{H}$ such that its generalization error is null, i.e. $L_D(h^*) = 0$, it is said that the *realizable assumption* is met. Note that under the realizable assumption, there is probability 1 that the empirical error associated to an hypothesis h^* such that $L_D(h^*) = 0$ is such that $L_S(h^*) = 0$ for a training set S drawn from distribution D over $\mathcal{X} \times \mathcal{Y}$.

The usage of a priori information in order to solve the learning problem is a necessary step that must be always undertaken. This is stated in a well-known theorem called the *No free lunch* theorem, of which the most famous version is the one concerning binary classification:

Theorem 1.1.3. *Let $\mathcal{A}$ be a learning algorithm that performs binary classification using the 0-1 loss function and with a given domain set $\mathcal{X}$. Let S be a training set of size n smaller than $|\mathcal{X}|/2$. Then there exists a distribution D over $\mathcal{X} \times \{0, 1\}$ such that:*

- *there exists a function $g : \mathcal{X} \to \{0, 1\}$ such that $L_D(g) = 0$,*
- *over the choice $S \sim D^n$ we have that $L_D(A(S)) \geq \frac{1}{8}$ with a probability at least 1/7.*

This theorem states that for each learner there exists at least a task in which the learner fails. The usage of the a priori knowledge about the specific learning task is necessary to avoid those distributions that would lead to a failure. In the cases that are going to be discussed, this information is used to restrict the algorithm to a given class $\mathcal{H}$.

1.1.2 The bias-complexity trade-off

The usage of ERM algorithm with inductive bias may limit the problem of overfitting but has a cost. Let $\tilde{L}$ be the so-called *Bayes risk*, defined as

$$\tilde{L} = \inf_h L_D(h)$$

where the infimum is taken over $\mathcal{Y}^{\mathcal{X}}$. Let then

$$\tilde{L}_{\mathcal{H}} = \inf_{h \in \mathcal{H}} L_D(h)$$

be the optimal performance of a learning algorithm restricted on class $\mathcal{H}$, i.e. the infimum of the generalization error over a restricted hypothesis class $\mathcal{H}$. And let $\tilde{h}$ be an hypothesis inside class $\mathcal{H}$ that minimizes the empirical error, i.e.

$$\tilde{h} \in \mathcal{H} \ : \ L_S(\tilde{h}) \leq L_S(h) \ \forall h \in \mathcal{H}.$$

Note that both $\tilde{L}$ and $\tilde{L}_{\mathcal{H}}$ do not depend on the training set S.

The difference between the true error of a given hypothesis h and the Bayes risk can be decomposed in this way:

$$L_D(h) - \tilde{L} = \underbrace{(L_D(h) - L_D\tilde{h}))}_{\textit{optimization error}} + \underbrace{(L_D\tilde{h}) - \tilde{L}_{\mathcal{H}})}_{\textit{estimation error}} + \underbrace{(\tilde{L}_{\mathcal{H}} - \tilde{L})}_{\textit{approximation error}} \tag{1.6}$$

The three errors that appear in Eq. 1.6 can be thus described:

- the *optimization error* measures the goodness of the hypothesis h compared to a minimizer of the empirical error, $\tilde{h}$, in terms of the generalization (or true) error;
- the *estimation error* measures how well the hypothesis $\tilde{h}$ that minimizes the empirical error performs relative to a true minimizer of the generalization error in class $\mathcal{H}$ (intuitively, it is the cost that has to be payed for using a minimizer of the empirical error instead of the true error);
- finally, the *approximation error* (or *bias*) does not depend upon the hypothesis h nor the training data, and it quantifies how well the hypothesis class $\mathcal{H}$ is suited for the problem taken into account.

Let us assume that the learning algorithm produces an ideal output hypothesis h such that the associated optimization error is null. The performance of such h still depends upon the approximation and estimation error.

The approximation error can be decreased by taking bigger hypothesis classes $\mathcal{H}$ that better approximate the true setting $\mathcal{Y}^{\mathcal{X}}$. The estimation error depends upon both the class $\mathcal{H}$ and the training set S: it increases with the 'complexity' of $\mathcal{H}$ and decreases with the size of the training set S, i.e. more complex classes $\mathcal{H}$ need a greater number of sampling data in order to reduce the estimation error. The concept of 'complexity' of a class is discussed in Section 1.1.3.

The trade-off between the approximation and the estimation error is called the *bias-complexity trade-off*. Let's fix the training set S. Choosing a complex class $\mathcal{H}$ decreases the approximation error but at the same time might increase the estimation error, leading to an overfitting. On the other hand, choosing a simple class $\mathcal{H}$ decreases the estimation error but might increase the approximation error, leading to an underfitting.

The need to minimize the generalization error associated to an hypothesis h by controlling two contradictory factors was formalized within the context of a new induction principle, the so-called *Structural Risk Minimization* (SRM) principle. The idea behind SRM is to consider a sequence of hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3 \ldots$ of increasing complexity and to optimize the empirical error plus another term, a penalty term, that takes into account the complexity of the class (for a more thorough explanation of this paradigm see [25]).

However, in many practical situations the SRM approach does not lead to satisfying results. A more practical approach is the usage of a procedure called *validation*, which is described in Section 1.6.1.

1.1.3 Learnability and complexity

The training set has always been assumed to be random, so there is the possibility that it may not be a fair sample of the underlying distribution D: for example, in a classification problem all training data may be drawn from a subset of all the classes. Furthermore, even if the training set does in fact faithfully represent D, it is still a discrete set, and may not reflect some fine details of the distribution.

For binary classification problems, these issues lead to the definition of *Probably*

Approximately Correct (PAC) *Learnability* for an hypothesis class:

Def. 1.1.4 (PAC Learnability)**.** A hypothesis class $\mathcal{H}$ is *PAC learnable* if there exist a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathcal{A}$ with the following property: for every $\epsilon,\ \sigma \in (0,1)$ and for every D distribution over $\mathcal{X} \times \mathcal{Y}$, if the realizable assumption holds with respect to $\mathcal{H}$ and D, then the output hypothesis $h = \mathcal{A}(S)$ retrieved by taking as input a training set S of a number $m \geq m_{\mathcal{H}}(\epsilon, \sigma)$ of i.i.d. examples generated by D, is such that

$$L_D(h) \leq \epsilon$$

with probability of at least $1 - \sigma$.

In this definition, ϵ can be interpreted as the *accuracy* associated to the hypothesis h, and σ as its *confidence*. The function $m_{\mathcal{H}}$ represents the *sample complexity* of learning $\mathcal{H}$, i.e. the number of samples required to guarantee a probably approximately correct solution. It depends on both the accuracy, the confidence and the properties of the class $\mathcal{H}$. For example, it can be proved (see [3], [25]) that, in the context of binary classification, every *finite* hypothesis class $\mathcal{H}$ is PAC learnable with sample complexity

$$m_{\mathcal{H}}(\epsilon, \sigma) \leq |\frac{\log |\mathcal{H}|/\sigma}{\epsilon}|$$

There are also infinite classes that turn out to be learnable.

By relaxing the realizable assumptions the *Agnostic PAC Learnability* is defined:

Def. 1.1.5 (Agnostic PAC Learnability)**.** A hypothesis class $\mathcal{H}$ is *agnostic PAC learnable* if there exist a function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathcal{A}$ with the following property: for every $\epsilon,\ \sigma \in (0,1)$, for every D distribution over $\mathcal{X} \times \mathcal{Y}$ the output hypothesis $h = \mathcal{A}(S)$ retrieved by taking as input a training set S of a number $m \geq m_{\mathcal{H}}(\epsilon, \sigma)$ of i.i.d. examples generated by D and labeled by f, is such that

$$L_D(h) \leq \min_{h' \in \mathcal{H}} L_D(h') + \epsilon$$

with probability of at least $1 - \sigma$.

It can be proved (see [25]) that an hypothesis class that is *uniform converging* is agnostic PAC learnable. The definition of uniform convergence for a class $\mathcal{H}$ is given in the Def. 1.1.6.

Def. 1.1.6 (Uniform Convergence)**.** Given a domain set $\mathcal{X}$ and a target set $\mathcal{Y}$, an hypothesis class $\mathcal{H}$ has the uniform convergence property with respect to a loss function L if there exists a function $m_{\mathcal{H}}^{UC} : (0,1)^2 \to \mathbb{N}$ such that for every $\epsilon, \sigma \in (0,1)$ and for every probability distribution D over $\mathcal{X} \times \mathcal{Y}$, if a training set S has size $m \geq m_{\mathcal{H}}^{UC}(\epsilon, \sigma)$ and contains samples drawn i.i.d. according to D, then

$$|L_S(h) - L_D(h)| \leq \epsilon \quad \forall\, h \in \mathcal{H} \tag{1.7}$$

with probability of at least $1 - \sigma$.

When a training set S satisfies the property described in Eq. 1.7 it is said that S is ϵ- *representative*.

Remaining in the field of boolean categorization, there exists a property called the Vapnik-Chervonenkis dimension (*VC-dimension*) [27] [4] for each hypothesis class $\mathcal{H}$ which correctly characterizes its learnability. Before introducing this concept it is necessary to define the *shattering* of a class $\mathcal{H}$:

Def. 1.1.7. Given an hypothesis class $\mathcal{H}$ and a finite subset $C \subset \mathcal{H}$, it is said that $\mathcal{H}$ shatters C if the restriction of $\mathcal{H}$ to C, i.e. $\mathcal{H}_C = \{(h(c_1, \dots h(c_m)) \;:\; h \in \mathcal{H}\}$ is equal to all functions that go from C to $\{0,1\}$.

Example: let $\mathcal{H}$ be the class of threshold functions over $\mathbb{R}$, i.e.

$$\mathcal{H} = \{h_a : \mathbb{R} \to \{0,1\} \text{ such that } h_a(x) = 1 \text{ if } x < a,\ 0 \text{ otherwise, for } a \in \mathbb{R}\}$$

Take a set $C = \{c_1\}$. Taking $a = c_1 + 1$, then $h_a(c_1) = 1$, and taking $a = c_1 - 1$ it is $h_a(c_1) = 0$. Therefore, the restriction $\mathcal{H}_C$ contains the set of all functions from C to $\{0,1\}$, so $\mathcal{H}$ shatters C. If instead the set $C = \{c_1, c_2\}$ where $c_1 \leq c_2$ is taken, there is no $h \in \mathcal{H}$ such that $h_a(c_1) = 0$ and $h_a(c_2) = 1$, because any threshold that assigns

label 0 to c_1 must assign label 0 to c_2 as well. Therefore not all functions from C to $\{0,1\}$ are included in $\mathcal{H}_C$, and so $\mathcal{H}$ does not shatter C.

With the notion of shattering, the VC-dimension is defined as this:

Def. 1.1.8 (VC-dimension)**.** The VC-dimension of a hypothesis class $\mathcal{H}$ is the maximal size of a set $C \subset \mathcal{X}$ that can be shattered by $\mathcal{H}$. If $\mathcal{H}$ can shatter sets of arbitrarily large size, it is said that $\mathcal{H}$ has infinite VC-dimension.

In the previous example, the class of threshold functions has VC-dimension equal to 1.

The PAC learnability of hypothesis classes associated to binary classification using VC-dimension is summed up in a theorem called the *Fundamental theorem of binary classification*:

Theorem 1.1.9 (Fundamental theorem of binary classification)**.** *Let $\mathcal{H}$ be a hypothesis class from a domain $\mathcal{X}$ to $\{0,1\}$ associated to a learning algorithm $\mathcal{A}$ and let the loss function be the 0-1 loss function. Then the following are equivalent:*

- *$\mathcal{H}$ has the uniform convergence property (Def. 1.1.6);*
- *$\mathcal{H}$ is PAC learnable (Def. 1.1.4);*
- *Any $h = \mathcal{A}(S)$ with S training set returned following the ERM paradigm is a successful PAC learner for $\mathcal{H}$;*

The proof can be found in [25]. There is also a version of this theorem (called the *quantitative version*) that directly quantifies the sample complexity.

This theorem is applied to binary classification. A similar result holds for some other learning problems, such as regression with the absolute loss or the squared loss. However, the theorem does not hold for all learning tasks.

1.2 Linear predictors

A common family of hypothesis classes applied to learning algorithms is the family of *linear predictors*.

Def. 1.2.1. *Linear predictors* L_d is a set of functions, where each function

- is parameterized by a $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$,
- takes as input a vector $\mathbf{x} \in \mathbb{R}^{\mathbf{d}}$ and
- returns as output the scalar $\langle \mathbf{w}, \mathbf{x} \rangle + b$,

i.e

$$L_d = \{h_{\mathbf{w},b} : \mathbb{R}^d \to \mathbb{R} \text{ such that } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

where

$$h_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b = \left(\sum_{i=1}^{d} w_i x_i \right) + b.$$

The parameter b can be incorporated inside $\mathbf{w}$ by considering $\mathbf{w}' = (b, w_1, \ldots, w_d) \in \mathbb{R}^{d+1}$ and $\mathbf{x}' = (1, x_1, \ldots x_d) \in \mathbb{R}^{d+1}$. This leads to the homogeneous linear functions that, in this chapter, are included inside the definition of linear predictors.

The different hypothesis classes of linear predictors are constructed by composing a function $\phi : \mathbb{R} \to \mathcal{Y}$ with the functions inside L_d. Many learning algorithms rely on linear predictors, because they can usually be learned efficiently, are intuitive, easy to interpret and fit the training data well in many learning problems.

In the following sections some examples of hypothesis classes of linear predictors are provided, as well as their associated most common learning algorithms.

1.2.1 Halfspaces

The hypothesis class of halfspaces was designed to solve binary classification problems, where $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{+1, -1\}$. In this class, the composing function ϕ is the *sign* function:

$$\mathcal{H}_d = \text{sign} \circ L_d = \{\mathbf{x} \mapsto sign(h_{\mathbf{w},b}(\mathbf{x})) \ : \ h_{\mathbf{w},b} \in L_d\}$$

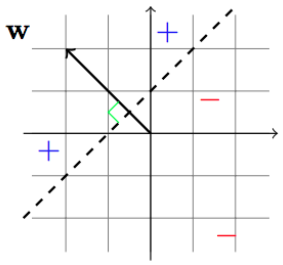


Figure 1.2: Example of halfspace associated to vector $\mathbf{w}$. Picture taken from [25].

It can be proved (see [25] [13]) that the complexity of these hypothesis classes, i.e. their VC-dimension, is directly proportional to the size d of the domain set $\mathbb{R}$.

A geometric illustration of this class with $d = 2$ is given in Figure 1.2. Each hypothesis forms a halfplane bounded by a hyperplane that is perpendicular to the vector $\mathbf{w}$ and intersects the vertical axis at the point $(0, -b/w_2)$.

The ERM procedure is usually implemented with such hypothesis classes when the learning problem is *separable*, i.e. all positive labelled points can be divided from the negative labelled points by a hyperplane. In this case, the empirical risk is minimized with the 0-1 loss function, and the way the minimizing operation is carried out leads to different possible learning algorithms, such as *Linear Programming* (LP) and the *Perceptron algorithm*.

1.2.2 Linear Regression

In linear regression the domain set $\mathcal{X}$ is a subset of $\mathbb{R}^d$ and the target set $\mathcal{Y}$ is the set of real numbers $\mathbb{R}$. It follows that the composing function ϕ that produces the associated hypothesis class is the identity function, i.e. the hypothesis class is equal to L_d. An analysis of the complexity associated to this hypothesis class can be found in [25].

There are many loss functions that can be associated to linear regression. As stated in Section 1.1, one of the most used is the *squared-loss function*. For this choice, the empirical error is called *Mean Squared Error*, and it is defined as this:

$$L_S(h) = \frac{1}{n}\sum_{i=1}^{n}\left(h(\mathbf{x_i}) - y_i\right)^2 \tag{1.8}$$

The application of the ERM procedure to linear regression leads to the *Least Square* algorithm:

$$\mathcal{A}(S) = \underset{\mathbf{w}}{\operatorname{argmin}}\, L_S(h_{\mathbf{w}}) = \underset{\mathbf{w}}{\operatorname{argmin}}\, \frac{1}{n}\sum_{i=1}^{n}\left(\langle \mathbf{w}, \mathbf{x}_i\rangle - y_i\right)^2$$

The problem is solved by computing the gradient of the objective function and compared it to zero. See [25], [3] or [13] for a complete exposure.

1.2.3 Logistic Regression

In logistic regression $\mathcal{X} = \mathbb{R}^d$, and the target set $\mathcal{Y}$ is equal to the interval [0,1] of $\mathbb{R}$. Even though the target set is continuos, logistic regression is used for classification problems, as the hypotheses $h(\mathbf{x})$ are interpreted as the *probability* that sample $\mathbf{x}$ is equal to 1.

The hypothesis class associated to logistic regression has as composing function a particular *sigmoid* function $\phi_{\text{sig}} : \mathbb{R} \to [0, 1]$ that is called *logistic function*:

$$\phi_{\text{sig}}(z) = \frac{1}{1 + \exp^{-z}}$$

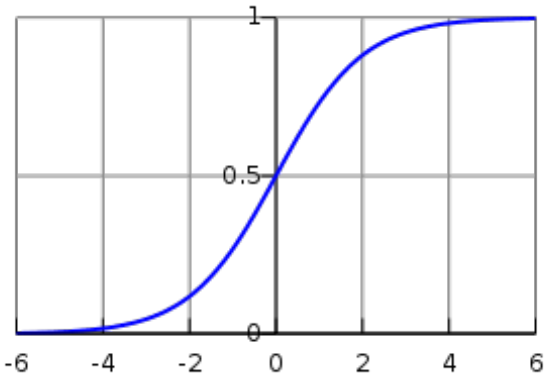


Figure 1.3: The logistic function.

Consequently the hypothesis class can be defined as:

$$\mathcal{H}_d = \phi_{\text{sig}} \circ L_d = \{\mathbf{x} \mapsto \phi_{\text{sig}}(h_{\mathbf{w},b}(\mathbf{x})) \ : \ h_{\mathbf{w},b} \in L_d\}$$

The predictions on the halfspace hypothesis and the logistic hypothesis are similar when $|\langle \mathbf{w}, \mathbf{x} \rangle|$ is large. When $|\langle \mathbf{w}, \mathbf{x} \rangle|$ is close to 0 the logistic function is close to 1/2. Intuitively, the logistic hypothesis is not sure about the value of the label so it guesses that the label is $\text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$ with probability slightly larger than 50%.

The loss function usually associated to the logistic function is the *logistic loss function*:

$$L(h_{\mathbf{w}}(\mathbf{x}), y) = \log(1 + \exp(-y h_{\mathbf{w}}(\mathbf{x}))) = \log(1 + \exp(-y \langle \mathbf{w}, \mathbf{x} \rangle))$$

For this choice of the loss function, the empirical error is thus defined:

$$L_S(h_{\mathbf{w}}) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle \mathbf{w}, \mathbf{x}_i \rangle)) \tag{1.9}$$

The advantage of this choice is that the logistic loss function is a *convex* function with respect to $\mathbf{w}$, and so the ERM problem can be solved efficiently using standard methods.

The actual labelling is performed by fixing a given *threshold* $s \in (0, 1)$ and labelling 1 all sample points with probability over the threshold as positive, 0 otherwise:

$$h_{01}(\mathbf{x}_i) = \begin{cases} 1 & \text{if } h(\mathbf{x}_i) \geq s, \\ 0 & \text{otherwise} \end{cases} \tag{1.10}$$

1.2.4 Regularized Loss Minimization and stability

Regularized Loss Minimization (RLM) is a learning paradigm that can be used instead of ERM within the context of linear predictors. In this learning rule, the aim is to minimize the empirical error and a new *regularized function* $R : \mathbb{R}^d \to \mathbb{R}$ applied to the parameters vector $\mathbf{w}$, i.e.

$$\mathcal{A}(S) = \underset{\mathbf{w}}{\operatorname{argmin}} \left(L_S(h_{\mathbf{w}}) + R(\mathbf{w}) \right) \tag{1.11}$$

Intuitively, the regularization function measures the complexity of the hypothesis, and the learning algorithm searches for a balance between a low empirical error and a simple output hypothesis.

There are many regularized functions that can be used. One of the simplest ones is the *Tikhonov regularization*, $R(\mathbf{w}) = \lambda \|\mathbf{w}\|^2$ where $\lambda > 0$ is a scalar and the norm is the ℓ_2 norm, $\|\mathbf{w}\|^2 = \sum_{i=1}^{d} w_i^2$. With this regularizer, Eq. 1.11 becomes

$$\mathcal{A}_S = \underset{\mathbf{w}}{\operatorname{argmin}} \left(L_S(h_{\mathbf{w}}) + \lambda \|\mathbf{w}\|^2 \right) \tag{1.12}$$

Equation 1.12 can be seen as an example of the SRM paradigm, where the sequence of hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3 \ldots$ is defined as $\mathcal{H}_i = \{\mathbf{w} \ : \ \|\mathbf{w}\|^2 \leq i \in \mathbb{N}\}$, if the sample complexity of each class $\mathcal{H}_i$ depends on i. The application of the Tikhonov regularization to linear regression leads to the *ridge regression*, defined in Section 1.7.1.

In the RLM paradigm, the application of the Tikhonov regularizer gives the learning algorithm $\mathcal{A}$ an important property: *stability*.

Intuitively, $\mathcal{A}$ is *stable* if a small change in the input does not lead to great changes in the output. This concept can be formalized in this way: let $S = (z_1, z_2, \ldots z_n)$ be a training set, and let $S^{(i)}$ be another training set where element z_i is replaced with another sample z', i.e. $S^{(i)} = (z_1, \ldots z_{i-1}, z', z_{i+1}, \ldots z_n)$. Let $h = \mathcal{A}(S)$ be the hypothesis with input training set S and $h^i = \mathcal{A}(S^{(i)})$ the one with input training set $S^{(i)}$. Then the learning algorithm $\mathcal{A}$ is stable if, taken sample z_i, the difference between the loss of the hypothesis h and h^i calculated on point z_i is small, i.e. $L(h(z_i), z_i) - L(h^i(z_i), z_i)$ is small.

It can be proved (see [13]) that, by taking a *convex* and either *Lipschitz* [1] or smooth

[1] A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is $\rho-$Lipschitz over a $C \subset \mathbb{R}^n$ if for every $\mathbf{x}, \mathbf{y} \in C$ it is:

$$\|f(\mathbf{x}) - f(\mathbf{y})\| \leq \rho \|\mathbf{x} - \mathbf{y}\|.$$

loss function L for learning algorithm $\mathcal{A}$, the application of Tikhonov regularization to RLM leads to a stable algorithm, and that the stability is controlled by parameter λ.

1.3 Support Vector Machines and Kernel Methods

In the next sections the main concepts behind Support Vector Machines [8] and kernel methods [5] applied to binary classification are given. Support Vector Machines can be used also to solve regression problems (leading to Support Vector Regression, SVR), but since all the problems described in the next chapters revolve around binary classification, this second aspect was not studied in details. See [13], [3] for more informations.

1.3.1 Support VectorMachines

Support Vector Machines (SVM) is a learning paradigm used for learning linear hypothesis (see Section 1.2) when the domain space $\mathcal{X}$ has a high dimension.

The basic concept behind SVM is the *margin*. Let the training set S be *linearly separable*, i.e. there exists an halfspace defined by an hyperplane $(\mathbf{w}, b)$ such that $\text{sign}(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) = y_i \ \forall i = 1, 2, \dots n$. All the halfspaces that satisfy this condition are hypotheses that solve the ERM paradigm (see Section 1.5 and 1.2.1).

A linearly separable training set has many such halfspaces, so the learning algorithm needs some criteria in order to choose one of them. Looking at Figure 1.4, intuitively the algorithm should choose the halfspace associated to the black hyperplane instead of the green one.

The concept of *margin* is defined in order to formalize this idea: the margin of a hyperplane $(\mathbf{w}, b)$ with respect to a training set S is the minimal distance between a point in the training set and the hyperplane. It can be proved that this quantity is equal to $\min_i |\langle \mathbf{w}, \mathbf{x_i} \rangle + b|$ (see [25]).

Two different conditions on the margin of a hypothesis lead to two different learning algorithms: Hard SVM and Soft SVM. Hard SVM returns the hypothesis associated to the largest possible margin, i.e.

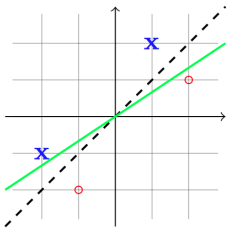


Figure 1.4: Graphical concept of SVM: the black dotted halfspace performs a better splitting of the training data than the green one.

$$\mathcal{A}_{\text{hard}}(S) = \underset{(\mathbf{w},b):\|\mathbf{w}\|=1}{\operatorname{argmax}} \left(\min_i |\langle \mathbf{w}, \mathbf{x_i} \rangle + b| \right) \quad \text{such that} \quad y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) > 0 \ \forall i$$

When the training set is linearly separable, the previous equation becomes

$$\mathcal{A}_{\text{hard}}(S) = \underset{(\mathbf{w},b):\|\mathbf{w}\|=1}{\operatorname{argmax}} \left(\min_i y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \right)$$

which is equivalent to

$$\begin{aligned} &\mathcal{A}_{\text{hard}}(S) = \underset{(\mathbf{w},b)}{\operatorname{argmin}} \|\mathbf{w}\|^2 \\ &\text{such that} \quad y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1 \ \forall i \end{aligned} \tag{1.13}$$

The Soft SVM is a relaxation version of the Hard SVM, and can be applied when the training set is not separable. The hard constraint $y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1$ is replaced with $y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1 + \zeta_i$, where $\zeta_1, \zeta_2, \ldots \zeta_n$ are positive variables that measure how much the constraint is violated. Soft SVM then aims at minimizing both the margin and the average of the ζ_i, introducing a trade-off parameter λ:

$$\begin{aligned} &\mathcal{A}_{\text{soft}}(S) = \min_{(\mathbf{w},b,\zeta)} \left(\lambda \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{n} \zeta_i \right) \\ &\text{such that} \quad y_i(\langle \mathbf{w}, \mathbf{x_i} \rangle + b) \geq 1 + \zeta_i \text{ and } \zeta_i \geq 0 \ \forall i \end{aligned} \tag{1.14}$$

It can be proved (see [25]) that the Soft SVM equation can be re-written as a regularized loss minimization (RLM) problem (see Section 1.2.4) by taking as loss function the *hinge loss function*

$$L(h_{\mathbf{w},b}(\mathbf{x}), y) = \max\{0, 1 - y(h_{\mathbf{w},b}(\mathbf{x})\}) = \max\{0, 1 - y(\langle \mathbf{w}, \mathbf{x}\rangle)\}$$

The name 'Support Vector Machine' comes from the interpretation of the hypothesis produced by the Hard SVM equation Eq. 1.13 in the homogeneous case, i.e. when $b = 0$. Let $h_{\mathbf{w}_0,b_0} = \mathcal{A}_{\text{hard}}(S)$ be the output hypothesis of the Hard SVM learning algorithm, with associated parameters $\mathbf{w}_0$ and b_0. Then vector $\mathbf{w}_0$ is *supported* by the sample points that are at distance $1/\|\mathbf{w}_0\|$ from the separating hyperplane, i.e.

$$\mathbf{w}_0 = \sum_{i \in I} \alpha_i \mathbf{x}_i$$

where $I = \{i \ : |\langle \mathbf{w}_0, \mathbf{x}_i\rangle = 1|\}$. Vectors $\{\mathbf{x}_i \ : \ i \in I\}$ are called the *support vectors*.

1.3.2 Embeddings and Kernels

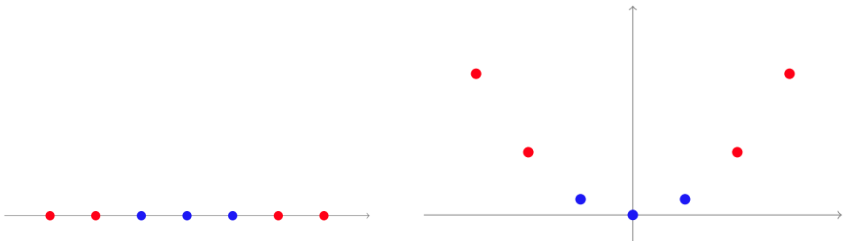
In the SVM paradigm described in the previous section, Hard SVM requires that the training set is linearly separable. Since in most of practical problems the data do not satisfy this constraint, the Soft SVM algorithm was developed.

However, it may happen that a not-linearly separable training set S becomes linearly separable when *mapped* into a higher dimensional space by a map $\psi : \mathbb{R} \to \mathbb{R}^m$. An example is shown in Figure 1.5, where a training set in $\mathbb{R}$ is mapped according to $x \mapsto (x, x^2) \in \mathbb{R}^2$.

This operation is referred to as *embedding*, and the range of the map ψ is usually referred to as *feature space*. The feature space is usually an *Hilbert* space [2]. The learning algorithm is trained on the new training set $S' = ((\psi(\mathbf{x}_1), y_1), (\psi(\mathbf{x}_2), y_2), \ldots (\psi(\mathbf{x}_n), y_n))$, and the label of a test point $\mathbf{x}_i$ becomes $h(\psi(\mathbf{x}_i))$.

A frequently used mapping is the *polynomial mapping*. Considering the 1-dimensional case, i.e. $\mathcal{X} = \mathbb{R}$, learning a k degree polynomial $p(x)$ can be done by applying the map $\psi : x \mapsto (1, x, x^2, \ldots x^k)$ and moving the problem into the feature space $\mathbb{R}^{k+1}$.

[2] A Hilbert space is an inner product space $(H, \langle \cdot, \cdot \rangle)$ such that the induced Hilbertian norm is complete.



(a) (b)

Figure 1.5: The advantage of embedding in a higher-dimensional space: a non-linearly serapable set becames separable after applying the mapping $x \mapsto (x, x^2)$.

The equations related to the SVM optimization problem that were given in the previous section can be generalized in the following expression:

$$\min_{\mathbf{w}} \left(g(\langle \mathbf{w}, \psi(\mathbf{x}_1)\rangle, \dots \langle \mathbf{w}, \psi(\mathbf{x}_n)\rangle) + R(\|\mathbf{w}\|)\right) \tag{1.15}$$

where $g : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary function and $R : \mathbb{R}_+ \to \mathbb{R}$ is a monotonically non-decreasing function. Indeed, Eq. 1.13 related to Hard SVM can be derived by letting $R(a) = a^2$ and g such that

$$g(a_1, \dots a_n) = \begin{cases} 0 & \text{if } \exists\ b \text{ such that } y_i(a_i + b)\ \forall i, \\ \infty & \text{otherwise} \end{cases}$$

and Eq. 1.14, related to Soft SVM, can be derived by letting $R(a) = \lambda a^2$ and $g(a_1, \dots a_n) = \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i a_i\}$.

The *Representer Theorem* (see [25]) states that there exists an optimal solution of Eq. 1.15 that lies in the span of $\{\psi(x_1), \dots \psi(x_n)\}$, i.e.

Theorem 1.3.1 (Representer Theorem)**.** *Let ψ be a map from $\mathcal{X}$ and an Hilbert space. Then there exists $\alpha \in \mathbb{R}^n$ such that*

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \psi(\mathbf{x}_i)$$

is the optimal solution of Eq. 1.15.

Proof. Let $\mathbf{w}^*$ be an optimal solution of Eq. 1.15. Since it belongs to an Hilbert space, it can be re-written as

$$\mathbf{w}^* = \sum_{i=1}^{n} \alpha_i \psi(\mathbf{x}_i) + \mathbf{u}$$

where $\langle \mathbf{x}_i, \mathbf{u} \rangle = 0 \ \forall i$. Consider $\mathbf{w} = \mathbf{w}^* - \mathbf{u}$. Then $\|\mathbf{w}^*\|^2 = \|\mathbf{w}\|^2 + \|\mathbf{u}\|^2$, so $\|\mathbf{w}\|^2 \leq \|\mathbf{w}^*\|^2$ and consequently $R(\|\mathbf{w}\|^2) \leq R(\|\mathbf{w}^*\|^2)$ because R is nondecreasing.

Now, for all i it is

$$\langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle = \langle \mathbf{w}^* - \mathbf{u}, \psi(\mathbf{x}_i) \rangle = \langle \mathbf{w}^*, \psi(\mathbf{x}_i) \rangle$$

so it follows that

$$g(\langle \mathbf{w}, \psi(\mathbf{x}_1) \rangle, \ldots \langle \mathbf{w}, \psi(\mathbf{x}_n) \rangle) = g(\langle \mathbf{w}^*, \psi(\mathbf{x}_1) \rangle, \ldots \langle \mathbf{w}^*, \psi(\mathbf{x}_n) \rangle).$$

This proves that the objective function of Eq. 1.15 computed on $\mathbf{w}$ cannot be greater than the objective function computed on $\mathbf{w}^*$, so $\mathbf{w}$ is an optimal solution. q.e.d.

Thanks to the Representer Theorem, $\langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle$ can be re-written as

$$\langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle = \left\langle \sum_j \alpha_j \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \right\rangle = \sum_j \alpha_j \langle \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \rangle$$

and $\|\mathbf{w}\|^2$ as

$$\|\mathbf{w}\|^2 = \left\langle \sum_j \alpha_j \psi(\mathbf{x}_j), \sum_j \alpha_j \psi(\mathbf{x}_j) \right\rangle = \sum_{i,j=1}^{n} \alpha_i \alpha_j \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle$$

These equations lead to the introduction of the concept of *kernel*, defined as an inner product [3] inside the feature space $K(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$.

[3] Let H be a complex vector space. An *inner product* on H is a function, $\langle \cdot, \cdot \rangle : H \times H \to \mathbb{C}$, s.t.

1. $\langle ax + by, z \rangle = a\langle x, z \rangle + b\langle y, z \rangle$, i.e. $x \mapsto \langle x, z \rangle$ is linear.
2. $\overline{\langle x, y \rangle} = \langle y, x \rangle$.
3. $\|x\|^2 \equiv \langle x, x \rangle \geq 0$ with equality $\|x\|^2 = 0$ iff $x = 0$.

Using kernels, the problem stated in Eq. 1.15 can be solved by the equivalent problem

$$\min_{\alpha \in \mathbb{R}^n} g\left(\sum_{j=1}^{n} \alpha_j K(\mathbf{x}_j, \mathbf{x}_1), \ldots \sum_{j=1}^{n} \alpha_j K(\mathbf{x}_j, \mathbf{x}_n)\right) + R\left(\sqrt{\sum_{i,j=1}^{n} \alpha_i \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)}\right)$$

This new problem does not need elements inside the (high dimensional) feature space, all it needs is the values of the kernel function, which can be placed inside an $n \times n$ matrix G : $G_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ that is usually referred to as *Gram matrix*. Using the Gram matrix, Eq. 1.14 associated to Soft SVM can be re-written as

$$\min_{\alpha \in \mathbb{R}^n} \left(\lambda \alpha^T G \alpha + \frac{1}{n} \sum_{i=1}^{n} \max\left\{0, 1 - y_i (G\alpha)_i\right\}\right)$$

This equation can be written as quadratic programming and hence can be solved efficiently.

1.4 Ensemble Methods

Ensemble methods [10] [26] train multiple learners to solve the same problem. Their goal is to combine the predictions of several estimators built with a given learning algorithm in order to improve generalizability and robustness over a single estimator. Ensemble learning is also called *committee-based learning*, or learning *multiple classifier systems.*

Figure 1.6 shows a common architecture for ensemble methods. There is a number of *base learners*, each generated from training data by a *base* learning algorithm. Usually base learners are homogeneous, i.e. of the same type, leading to *homogeneous ensembles*, even though there are also some *heterogeneous ensembles* with different learning algorithm for their base learners.

An ensemble is usually constructed in two steps, i.e. generating the base learners, and then combining them into a strong learner that can make very accurate predictions.

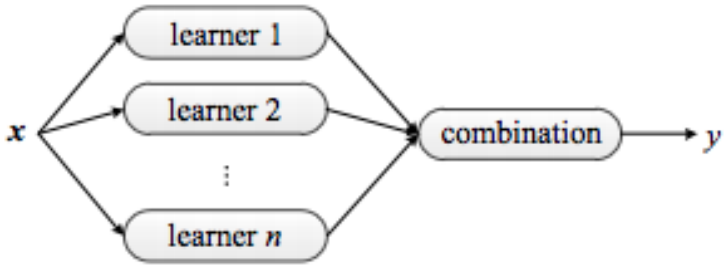


Figure 1.6: Common ensemble architecture.

Generally the combination step can be performed in two different ways, each leading to a wide family of ensemble methods:

- building sequential estimators leads to *sequential* ensemble methods, or *boosting methods*.
- building several estimators independently and then averaging their predictions according to given rules leads to *parallel* ensemble methods, or *averaging methods*;

An example of averaging method is the *random forest* algorithm [20], which will be described in Section 1.5.4. In the following section the attention is focused on the theory behind boosting methods.

1.4.1 Boosting Methods

Boosting methods combine a *sequence* of many *weak learners* to obtain a single *strong learner*, usually referred to as *commitee*. A weak learning algorithm is one whose error is only slightly better than random guessing. This concept is formalized by the $\gamma-$*Weak Learnability*:

Def. 1.4.1. A learning algorithm $\mathcal{A}$ is a γ- *weak-learner* for a hypothesis class $\mathcal{H}$ if there exists a function $m_{\mathcal{H}} : (0,1) \to \mathbb{N}$ such that for every $\delta \in (0,1)$, for every distribution D over $\mathcal{X}$ and for every labeling function $f : \mathcal{X} \to \{\pm 1\}$, the algorithm

returns a hypothesis $h = \mathcal{A}(S)$ such that, with probability of at least $1 - \delta$,

$$L_D(h) \leq \frac{1}{2} - \gamma.$$

Consequently, an hypothesis class $\mathcal{H}$ is γ- *weak-learnable* if there exists at least one γ- *weak-learner* associated to the class.

The implementation of a boosting algorithm usually starts with taking a simple hypothesis class B and applying a weak learning algorithm that follows the ERM paradigm. The learning algorithm has to be an efficient implementation, and must produce an hypothesis h that satisfies the requirement stated in Def. 1.4.1.

An example of simple hypothesis class is the *decision stumps* over $\mathbb{R}^d$, defined as

$$\mathcal{H}_{DS} = \{\mathbf{x} \mapsto \text{sign}(\theta - x_i) \cdot b \ : \ \theta \in \mathbb{R}, i \in \{1, 2, \dots d\}, b \in \{\pm 1\}\}$$

It can be proved (see [25]) that there exists a fast implementation of the ERM procedure that retrieves the minimizer of the empirical error inside this hypothesis class, which in this case becomes

$$L_S(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{[h(\mathbf{x}_i) \neq y_i]}$$

The following step is to *boost* this efficient weak learner into an efficient strong learner. In one of the most famous boosting algorithms, AdaBoost (which states for Adaptive Boosting) [11], this operation is performed in a sequence of following steps. At step t the booster:

- defines a distribution D^t over the training set S, where D^t in practice is a probability vector of $\mathbb{R}^n$, i.e. $\sum_{i=1}^{n} D_i = 1$ and $D_i \geq 0 \ \forall i$;
- passes the training set S and the new distribution to the weak learner, which returns a 'weak' hypothesis h_t whose error is equal to

$$L_{S,D^t}(h_t) = \sum_{i=1}^{n} D_i^t \mathbb{1}_{[h_t(\mathbf{x}_i) \neq y_i]}$$

and is at most $1-\gamma$ with probability $1-\delta$ (note that L_{S,D^t} is equal to L_S if $D^t = \{1/n, \ldots, 1/n\}$);

- assigns this hypothesis a weight $w_t = \frac{1}{2}\log(1/L_{S,D^t} - 1)$, which is inversely proportional to the error of h_t;
- finally, it updates the distribution D^t by increasing the entries associated to higher values of the error $L(h_t(x_i), y_i)$, in order to 'force' the next iteration to focus on such points.

The output of AdaBoost is a strong learner that is based on the weighted sum of all the weak learners h_t. It can be proved (see [25]) that the empirical error associated to the output hypothesis decreases exponentially fast with the number of boosting steps.

Two observations must be made: first of all, the assumption on the weak learners is that the output hypothesis has an error of at most $1/2 - \gamma$ with probability $1-\delta$. This means that the weak learner can fail with probability δ. Using the union bound, over T iterations the probability that the weak learner never fails is $1 - \delta T$. However, it can be proved that the dependence of the sample complexity on δ can always be logarithmic in $1/\delta$, and consequently δT can be assumed to be small.

The second observation is that all these considerations are given with respect to the empirical error, not the generalization error. Also in this case it can be proved for hypothesis class with given properties that the estimation error (see Section 1.1.2) is small, and so the distance between the empirical error and the generalization error is small.

1.5 Decision Trees

A decision tree is an hypothesis $h : \mathcal{X} \to \mathcal{Y}$ that predicts the target of an input point $\mathbf{x}$ by following a path from a root node to a leaf node based on a sequence of splitting rules.

Intuitively the splitting rules should be decided according to one of the given paradigms, e.g. the ERM; however the computational costs associated to this kind of operations are

very high, and so decision tree algorithms are based on *heuristics*, like the greedy approach, where each rule (or node) is constructed separately. In this way a decision trees 'grows' from node to node and returns a global output hypothesis that is not guaranteed to be the optimal solution of the learning problem, but that works reasonably well in practice.

A decision tree learning algorithm partitions the domain set into a collection of cells, and assign a leaf of the decision tree to each cell. The hypothesis class is therefore a collection of piece-wise functions, with as many pieces as the number of splitting subsets (which, if the size of the decision tree is allowed to be arbitrary, can be infinite).

Figure 1.7 shows a partition of the domain set $\mathcal{X}$, a subset of $\mathbb{R}^2$, by lines that are parallel to the coordinate axes. The output hypothesis h is usually modeled as a constant function on each partition element.

The binary tree of Figure 1.7 represents the same model in a single binary tree.

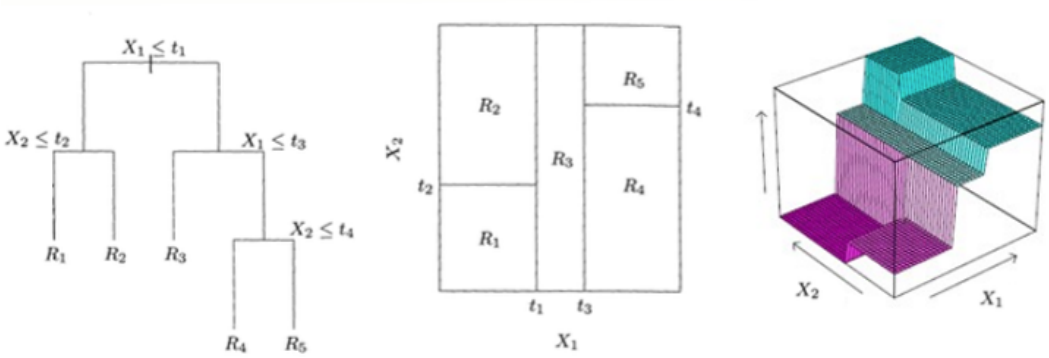


Figure 1.7: Example of a partition of a two-dimensional feature space, along with the corresponding decision tree and a perspective plot of the prediction surface.

Next sections describe a common tree-based method for regression and classification called CART [6]. Most of the material is retrieved from [13].

1.5.1 Regression trees

To grow a *regression* tree T_0 the domain set is split recursively in smaller regions until some stopping rule is applied. The learning algorithm has to automatically decide what

the splitting variables, and corresponding split points, should be. For example, let's assume that $\mathcal{X}$ is partitioned into M regions R_1, R_2, $\dots R_M$ and the output hypothesis is defined as

$$h(x) = \sum_{m=1}^{M} c_m I\{x \in R_m\} \tag{1.16}$$

Starting with all of the data, let's consider a splitting variable j and a split point s. Both j and s are chosen according to a desired criterion of minimization. For example, choosing the sum of squares criterion $\sum (y_i - h(x_i))^2$ as criterion of minimization, the best set of constants $\hat{c}_m$ is the average of y_i in region R_m

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

and so the best pair (j, s) has to solve

$$\min_{j,s} \left(\sum_{x_i \in R_m(j,s)} (y_i - \hat{c}_m)^2 \right). \tag{1.17}$$

where $R_m(j, s)$ is the region generated using variable j and split point s.

Since this computation is generally infeasible, a greedy algorithm is used, and j and s are chosen to solve

$$\min_{j,s} \left(\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right) \tag{1.18}$$

where R_1 and R_2 are half-planes defined as:

$$R_1(j, s) = \{x | x_j \leq s\} \quad \text{and} \quad R_2(j, s) = \{x | x_j > s\}$$

For any choice of j, it is easy to see that the inner minimization is solved by

$$\begin{aligned} \hat{c}_1 &= \frac{1}{N_1} \sum_{x_i \in R_1(j,s)} y_i, \\ \hat{c}_2 &= \frac{1}{N_2} \sum_{x_i \in R_2(j,s)} y_i \end{aligned} \tag{1.19}$$

The split point s can be found very quickly and so the determination of the best pair (j, s) is feasible. Once it is found, the data is partitioned in two resulting regions and the process is repeated over each region.

The number of levels of the tree is a tuning parameter that should be adaptively chosen from data. There are different criteria that can be chosen; for example, a node may be split only if:

- the decrease of the error is over a chosen threshold, or
- the size of the node is over a chosen size, or
- the level of the three is under a chosen top value.

It must be noted that sometimes splits are not restricted to be of the form $x_j \leq s$, but can be a linear combination of the form $\sum a_j x_j \leq s$. Weights a_j and split points s are usually optimized to minimize the cost complexity criterion (see next section).

1.5.2 Cost complexity pruning

The procedure described in the previous section is likely to produce a very large tree. In this case the empirical error may be small, but the generalization error is likely to be very high, i.e. the tree may be over-fitting the training data. One way to solve this problem is applying a *pruning* to the tree, constructing a smaller tree that would have a similar empirical error but a lower true error.

Usually the pruning operation is based on cross validation (see Section 1.6.2). The idea is to generate a sequence of trees which are successively smaller to the point of having a tree with just the root node. Then the best tree is chosen as the tree in the sequence that gives the smallest error in the validation data, according to a chosen criterion.

The most known criterion for pruning is called the *cost-complexity criterion*. It can be interpreted as the error in the prediction of a tree plus a penalty factor for the size of the tree.

Let $T \subset T_0$ be a subtree of T_0 with $|T|$ terminal nodes. The cost complexity criterion is defined as

$$C_\alpha(T) = \sum_{m=1} |T| N_m Q_m(T) + \alpha |T| \tag{1.20}$$

where $Q_m(T)$ is called the *node impurity measure* and α is a penalty cost.

Impurity is a measure of how badly the observations at a given node fit the model. In a regression tree, for example, the node impurity measure may be the mean absolute error or the mean squared error within that node:

$$\begin{aligned} \hat{c}_m &= \frac{1}{N_m} \sum_{x_i \in R_m} y_i, \\ Q_m(T) &= \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2, \end{aligned} \tag{1.21}$$

Node impurity measures for classification trees will be discussed in the next section. It must be noted that during the growing of the tree, the criterion used for splitting data can be seen as a node impurity measure: in the previous section, for example, Eq. 1.17 uses the node impurity measure of Eq. 1.21.

It can be proved that for each α there is a unique subtree $T_\alpha \subseteq T_0$ that minimizes the cost complexity criterion. Parameter α is a tradeoff between the tree size and its goodness on data, where $\alpha = 0$ means there is no penalty for having too many nodes in a tree and the best tree using the cost complexity criterion is the full grown unpruned tree T_0. Usually the best $\hat{\alpha}$ is chosen through a five- or ten-folds cross validation.

The final output tree is $T_{\hat{\alpha}}$.

1.5.3 Classification trees

For *classification* trees differences lie in the criteria for splitting nodes and pruning trees, i.e. in the node impurity measure $Q_m(T)$ introduced in the previous paragraph.

Let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{1.22}$$

be the proportion of label k inside a node m with N_m observations. Some common node impurity measures $Q_m(T)$ are:

- Misclassification error:

$$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq \text{argmax } \hat{p}_{mk}) = 1 - \hat{p}_{mk}(m)$$

- Gini index:

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- Cross-entropy or deviance:

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}.$$

For example, if there are only two classes and p is the proportion of the second class we have $1 - \max(p, 1-p)$, $2p(1-p)$ and $-p \log p - (1-p) \log(1-p)$ respectively, as shown in Figure 1.8. For all cases, the maximum value of $Q_m(T)$ is reached when the class distribution is uniform, i.e. there is an equal number of members of the first and second class inside the node ($p = 0.5$), while the minimum values are attained when all records belong to the same class ($p = 0$ or $p = 1$).

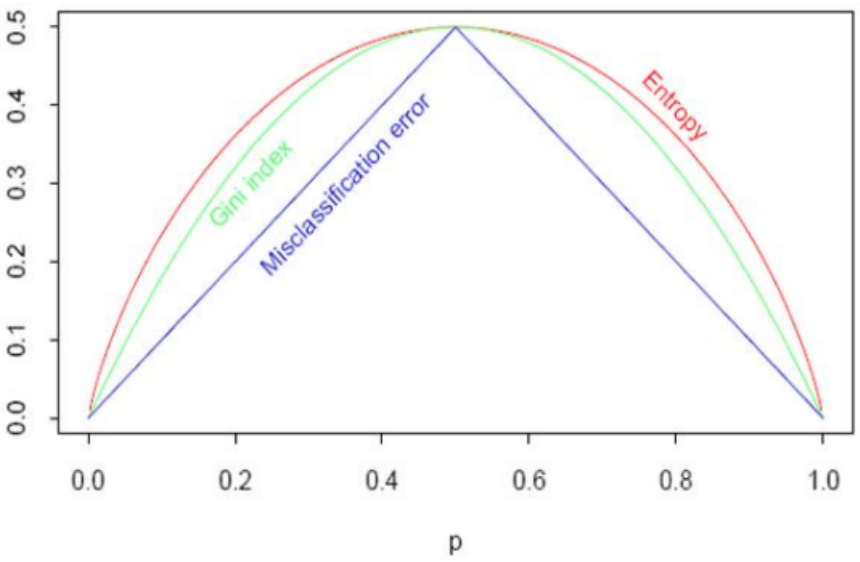


Figure 1.8: Example of common node impurity measures for a two-class classification, as a function of the proportion p of the second class.

Cross-entropy and Gini index are differentiable and more sensitive to changes in node probabilities, so they are usually chosen over the misclassification rate as splitting node criteria. During cost-complexity pruning any of these measures can be used, but usually the misclassification measure is chosen.

1.5.4 Random Forest

The *Random Forest* learning algorithm [20] is an example of parallel ensemble (see Section 1.4) where the weak learners belong to the family of decision trees. The idea behind random forest is to construct a collection of different decision trees and then averaging the predictions of each decision tree.

Each decision tree is built in a different way to ensure a wide range of cases. Usually there are two main elements that are settled for each decision tree:

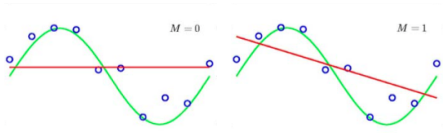
- the training set provided to the decision tree is not the original training set S, but a subset S' drawn from S with replacement, i.e. a *bootstrap sample*;
- the splitting strategies does not take into account all d features of a point $\mathbf{x} \in \mathcal{X}$, but is restricted to only a random subset of k features.

This random procedure usually sees the increasing of the bias, but thanks to the averaging step the variance decreases, usually compensating the bias.

1.6 Model Selection

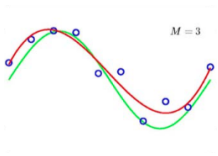
The restriction of the learning algorithm to an hypothesis class $\mathcal{H}$ is one of the key points of a learning problem. In practice, most of learning algorithms are designed to be applied to an hypothesis class $\mathcal{H}$ that includes only one type of function, or *model*, e.g. the polynomial functions, the exponential functions etc. Each model can also have a number of internal *parameters* that define the hypothesis of $\mathcal{H}$: e.g., for the polynomial model a parameter can be the degree r of the polynomial. For this reason, this task is usually referred to as *model selection* or *model assessment*.

A classical example of model selection is given by considering the problem of learning a one dimensional regression function, $f : \mathbb{R} \to \mathbb{R}$, that fits a training set S similar to the one provided in Figure 1.9.

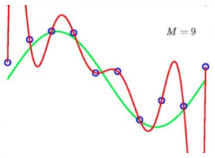


(a) $M = 0$

(b) $M = 1$



(c) $M = 3$



(d) $M = 9$

Figure 1.9: Example of polynomial fitting with polynomials of different degree M.

In this example we consider the learning algorithm to be restricted to polynomial functions. A free parameter in this scenario is the degree M of the polynomials. Figure 1.9 shows the resulting fitting hypothesis h for four different values of M. It is easy to see that a small degree leads to an h which does not fit the training data well, i.e. it has a large estimation error. However, the hypothesis associated to a polynomial of degree 9, while nulling the estimation error, is clearly overfitting, since it is associated to a high approximation error.

For both small and high values of M, the generalization or true error is high. A practical estimation of the true error can be retrieved by following a procedure called *validation*.

1.6.1 Train, test and validation set

The idea behind validation is very simple: the easiest way to evaluate the true error associated to an hypothesis h is sampling an additional set of examples, which must be *independent* from the training set S, and computing the empirical error on this new *validation set*. It can be proved (see [25], [3]) that this new empirical error is a better

estimate of the empirical error associated to h, at the price of needing an additional sample of data.

Validation can be naturally used to perform the model selection. The procedure is the following: a set of different hypothesis classes $\mathcal{H}_1, \mathcal{H}_2, \dots \mathcal{H}_m$ is defined and associated to as many learning algorithms $\mathcal{A}_1, \mathcal{A}_2, \dots \mathcal{A}_m$. Each algorithm is then trained on the training set S, and a set of output hypothesis $h_1, h_2, \dots h_m$ such that $h_i = \mathcal{A}(S)\ \forall i = 1, 2, \dots m$ is retrieved (in the example associated to Figure 1.9, the learning algorithms are associated to polynomial functions of fixed degree M and some of the resulting h_i are shown in red in Figure 1.9). Finally, a new validation set is constructed and each h_i is applied in order to compute the corresponding empirical error. The chosen h_i is the one associated to the lower empirical error.

It is proved (see [25]) that the error on the validation set approximates the true error as long as the number of considered hypothesis classes is not too large: in this case, the risk of overfitting increases.

1.6.2 k-fold cross validation

In many practical situations it is not possible to construct a new validation set: all the available data are included in the training set S, and they are too 'precious' to be discarded in the training phase. The *k-fold cross validation* is a technique that was developed in order to retrieve a good estimation of the true error without 'wasting' too many data.

In k-fold cross validation the training set is split into k subsets, or *folds*, of size $\#S/k$ (assume that $\#S$ is a multiple of k for simplicity). Then, for each fold, a learning algorithm is trained on the union of the other folds, *holding out* the k fold as validation set. The approximation of the true error associated to the learning algorithm is computed as the average of all the resulting approximations.

This procedure can be summarized in the following way: given a training set S of $\#S = n$ data points, a set of m learning algorithms $\mathcal{A}_1, \mathcal{A}_2, \dots \mathcal{A}_m$ ad a number k of folds,

1. Partition S into folds $S_1, S_2, \dots S_k$
2. For each $\mathcal{A}_i \in \{\mathcal{A}_1, \mathcal{A}_2, \dots \mathcal{A}_m\}$:

   For each $j = 1, 2, \dots, k$:

   $h_{i,j} = \mathcal{A}_i(S_j)$

   $\text{error}_i = \frac{1}{k} \sum_{j=1}^{k} L_{S_j}(h_{i,j})$
3. Output: $h^* = \mathcal{A}_{i^*}(S)$ where i^* is such that $\text{error}_{i^*} = \text{argmin}_i(\{\text{error}_i \text{ for } i = 1, 2, \dots m\})$

The procedure described in the previous section was an example of 1-fold cross validation, or simply called cross validation. The special case where $k = \#S$ is called the *leave-one-out* cross validation.

Usually, alongside a training and a validation set a third set, called *test set*, is used during the model assessment. This set, independent from the other two, is used after the assessment of the best model to test its performance and obtain a finale estimation of the true error of the chosen hypothesis.

1.7 Solving a Classification Problem

This thesis presents three different binary classification problems. The procedure followed to find an acceptable learner, or classifier, was always the same, and can be surmised in the following passages:

1. Import and interpretation of the data
2. Preliminary analyses
3. Construction of derived features
4. Model assessment
5. Prediction

6. Analysis of the results

First, the data provided by the TELCO were taken, interpreted and studied through a set of *preliminary analyses*; then a new set of derived features was constructed and added to the data. Once the training set was ready, different learning algorithms were compared using the *cross validation* procedure, thus performing a proper model assessment. Finally, the resulting learning algorithm was applied and the results were studied. Usually, the output of the algorithm led to new predictions in different experimental settings, where the feature variables were changed or restricted to given values.

1.7.1 Classification learning algorithms and setting

This section provides an overview of the classification algorithms used in the following chapters. All the implementations were retrieved from the sklearn library ([31]) and were performed in python ([33]) environmental language. For every algorithm a list of the parameters that were used to perform cross validation is given. The parameters are named according to those inside the sklearn library.

Ridge Regression The *Ridge Regression* learning algorithm is used to solve linear regression (see Section 1.2.2). Its learning rule is obtained by applying the Tikhonov regularization (see Section 1.2.4) to linear regression with the squared loss.

Def. 1.7.1 (Ridge Regression)**.**

$$\mathcal{A}(S) = \underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}} \left(\lambda\|\mathbf{w}\|^2 + \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(\langle \mathbf{w}, \mathbf{x}_i\rangle - y_i)^2 \right) \tag{1.23}$$

The solution of Eq. 1.23 is obtained by computing the gradient of the objective and set it to 0:

$$(2\lambda n I + A)\mathbf{w} - \mathbf{b} = 0 \tag{1.24}$$

where

$$A = \left(\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T \right) \qquad \text{and} \qquad \mathbf{b} = \sum_{i=1}^{n} y_i \mathbf{x}_i$$

Matrix A is a positive semidefinite matrix, so the matrix $2\lambda n I + A$ has all its eigenvalues bounded below by 2λ and is therefore invertible. The solution of ridge regression becomes:

$$\mathbf{w} = (2\lambda n I + A)^{-1} \mathbf{b} \tag{1.25}$$

Ridge Regression was used to perform classification following a procedure analogous to Logistic Regression, see Section 1.2.3.

The parameters that were cross validated were:

- **alpha**: it corresponds to the γ parameter of Eq. 1.23. It controls the level of regularization.

Support Vector Machines with RBF kernel This learning algorithm belongs to the family of Support Vector Machines where a kernel is applied (see Sections 1.3). The implemented kernel is the *Gaussian* kernel, known also as the *Radial Basis Functions* (RBF) kernel and defined as:

$$K(\mathbf{x}, \mathbf{x}') = e^{-\frac{\|\mathbf{x}-\mathbf{x}'\|}{2\sigma}} \tag{1.26}$$

Intuitively, the RBF kernel sets the inner product in the feature space to be close to 0 if the original data points inside the domain space are far away from each other, and close to 1 if they are near, with parameter σ that quantifies the closeness.

The parameters that were cross validated were:

- **gamma**: it corresponds to the σ parameter of Eq. 1.26. It defines how much influence a single training example has.

Decision trees This learning algorithm is equal to the one described in Section 1.5.

The parameters that were cross validated were:

- **max_depth**: it represents the maximum depth of the tree.
- **min_samples_split**: it represents the minimum number of samples that are required in order to split an internal node.
- **max_features**: it represents the number of features taken into account when looking for the best split.

Random Forests This learning algorithm is equal to the one described in Section 1.5.4.

The parameters that were cross validated were:

- **nr_estimators**: it represents the number of trees in the forest
- **max_depth**: it represents the maximum depth of the tree.
- **min_samples_split**: it represents the minimum number of samples that are required in order to split an internal node.
- **max_features**: it represents the number of features taken into account when looking for the best split.

Gradient Tree Boosting The *Gradient Tree Boosting Classifier* (GTBC) is a boosting learning algorithm (see Section 1.4.1) applied to decision trees. In the learning problems described in this thesis, the loss function that has been optimized is equivalent to the one of logistic regression described in Section 1.2.3.

The parameters that were cross validated were:

- **nr_estimators**: it represents the number of trees taken by the ensemble
- **max_depth**: it represents the maximum depth of the tree.
- **min_samples_split**: it represents the minimum number of samples that are required in order to split an internal node.
- **max_features**: it represents the number of features taken into account when looking for the best split.

1.7.2 Evaluation: ROC curves

All the learning algorithms described in the previous section performed a logistic regression (see Section 1.2.3), i.e. solved a regression problem where the output represented the probability that a given sample was positive labelled. The output is then binarized according to a given threshold s as described in Eq. 1.10

In case of logistic regression, the evaluation of a prediction can be achieved using a *Receiver Operator Characteristic Curve*, or ROC curve. To define such a curve however it is necessary to introduce the concept of *confusion matrix* and the main indicators associated to binary classification.

In binary classification a label can be either 0 or 1. This leads to a total of 4 possible combinations for the couple $(y_i, h_{01}(\mathbf{x}_i))$, where $y_i \in \{0, 1\}$ is the label, $\mathbf{x}_i \in \mathcal{X}$ is a sample point and $h_{01}(\mathbf{x}_i)$ is the prediction of the label associated to $\mathbf{x}_i$ according to the classifier h_{01} derived from the output hypothesis h:

- $(y_i = 1,\ h_{01}(\mathbf{x}_i) = 1)$: both the prediction and the original label were Positive; in this case $h_{01}(\mathbf{x}_i)$ is called a *true positive* sample (TP).
- $(y_i = 0,\ h_{01}(\mathbf{x}_i) = 0)$: both the prediction and the original label were Negative; in this case $h_{01}(\mathbf{x}_i)$ is called a *true negative* sample (TN).
- $(y_i = 1,\ h_{01}(\mathbf{x}_i) = 0)$: the classifier assigned label 0 to an original positive value; in this case $h_{01}(\mathbf{x}_i)$ is called a *false negative* sample (FN).
- $(y_i = 0,\ h_{01}(\mathbf{x}_i) = 01)$: another example of misclassification, this time where the classifier assigned label 1 to an original negative value; in this case $h_{01}(\mathbf{x}_i)$ is called a *false positive* sample (FP).

The number of TP, TN, FN and FP samples retrieved by a classifier h_{01} are usually reported inside a matrix, called *Confusion Matrix* (see Table 1.1), where the rows represents the original labels of the sample points, and the columns the predicted labels:

From the confusion matrix a list of four indicators can be retrieved for each classifier h_{01}:

	$h_{01}(\mathbf{x}_i)$: 0 $(h(\mathbf{x}_i) < s)$	$h_{01}(\mathbf{x}_i)$: 1 $(h(\mathbf{x}_i) > s)$
y_i: 0	TN	FP
y_i: 1	FN	TP

Table 1.1: Confusion Matrix

- *PRECISION* $= \frac{TP}{(TP+FP)}$.

  It describes how many of the positively classified were relevant.

- *RECALL* = *SENSITIVITY* $= \frac{TP}{(TP+FN)}$.

  It describes how good a test is at detecting the positives.

- *SPECIFICITY* $= \frac{TN}{(TN+FP)}$.

  It describes how good a test is at avoiding false alarms.

- *ACCURACY* $= \frac{TP+TN}{(TP+TN+FP+FN)}$.

  It describes how much the test was accurate.

For a perfect classifier, $FP = FN = 0$, so the confusion matrix is a diagonal matrix and the value of all four indicators is 1. In a real context, FP and/or FN are not null.

The sensitivity is also called the *True Positive Rate* (TPR), while the opposite of the specificity, i.e. *1 - SPECIFICITY*, is called the *False Positive Rate* (FPR):

$$\text{FPR} = 1 - \text{S}PECIFICITY = 1 - \frac{TN}{(TN+FP)} = \frac{FP}{(TN+FP)}$$

The ROC curve studies the behavior of the True Positive Rate and the False Positive Rate according to the threshold s that produced the related classifier. The output is a curve such as the one described in Figure 1.10.

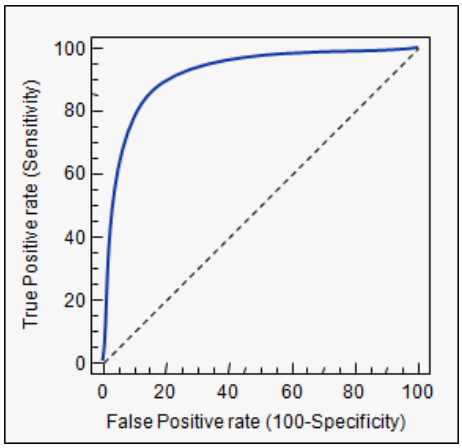


Figure 1.10: Example of ROC curve

It is a monotone increasing function, mapping (0, 1) to (0, 1). A perfect classifier is such that the TPR and the FPR are both equal to 1 for every choice of the threshold s, leading the ROC curve to follow the left and upper borders of the positive unit quadrant. Conversely, a completely uninformative prediction is such that the TPR and the FPR are always equal, leading the ROC curve to be equal to the diagonal line. As a consequence, the more the ROC curve approaches the border of the positive unit quadrant, the better is the classifier performance.

A way to measure the proximity of the ROC curve to the border of the positive unit quadrant is the area between the curve and the x axis, called the *Area Under the Curve* or AUC. A classifier with AUC=1 is a perfect classifier, and one with AUC = 0.5 is the equivalent of the random classifier.

The AUC is a common indicator for the performance of a binary classifier, and it was used in the classification problems described in the following chapters. Indeed, it was usually used as a scoring parameter for the cross validation operation.

Chapter 2

Churn Prevention #1

In the telecommunication world, *churn customer* is defined as the occurrence when a customer of a telecommunication company, later referred to as *TELCO*, is not satisfied with the terms of its contract and chooses to leave the company, usually switching to a competitor operator. To prevent churning, TELCOs draw up new offers that provide more advantages to their customers and, consequently, reduce the income of the company. In order to avoid a great loss of income, those offers are not proposed to all costumers, but are reserved only to those that are about to terminate their contract, or *churn*.

The aim of *churn prevention* is to locate these potential churners with a certain advance, usually of the order of a couple of months. The prediction has to be accurate, since false positive errors will incur in offering more advantageous conditions to customers that are not about to churn, with a consequent reduction in the income from customers, while false negative errors may cause the loss of a real churner.

Usually, churn prevention involves statistical techniques applied to given data associated to the client, e.g. traffic data. Results however have been limited because of the complexity of the problem. In recent years, Machine Learning techniques have been applied to telecommunication data, both belonging to the supervised ([18]) and the unsupervised paradigm ([7], [12]).

The problem of churn prevention has been addressed for two different companies.

In this chapter, the analyses performed on data given by the first company, TELCO1, are reported. Section 2.1 describes the data provided by the company and the preliminary analyses carried out on such data. Section 2.2 provides the modeling of the problem and the sample strategy used to carry out the predictions. Finally, in Section 2.3 all the experiments and related results are reported: an initial phase of *model assessment* was followed by a wide range of predictions and analyses that were subscribed to and approved by the TELCO, leading to a systematic process of churn detection that has been carried out for over 2 years.

2.1 Data analysis

2.1.1 Type and storing of the data

The data provided by TELCO1 consisted of a set of monthly-based indicators related to the activities of all the sim cards owned by the TELCO's customer. For each month, the batch of data associated to the sim cards that were active during that month contained over 80 variables that described the usage of the sim card and included:

- *basic information* on the sim card, e.g. the anonymized id of the owner of the sim (which will be referred to as *client*), the date of activation and, in case of a sim card that churned to the TELCO, the operator from which it churned,
- *deactivation information*, present in case of deactivation of the sim card, such as the deactivation date, the cause of deactivation and, if the cause is a churn, the operator to which the sim card churned,
- *usage data*, e.g. the length of phone calls and the bytes of internet data,
- *incomes* derived from the traffic data,
- *terminal data*, that report the number of active and non-active terminals (i.e. devices connected to the network system) owned by the client, and

- *complaints and assistance data* associated to the number of calls given and received by the assistance number.

The most important features, as well as their description, are listed in Table 2.2. Variables in section *Deactivation* are null if the sim card was not deactivated during the month of reference. In the *Usage* section, data associated to different operators *X* were provided.

Each batch of data was stored inside a given number of excel files (*.xlsx), whose contents were imported and stored inside a non-relational MONGO database (see [32]).

Some of these variables, such as the id of the client owning the sim card and the date of activation, are not directly related to the reference month of the batch, and thus remain constant inside each batch. Checks were performed before the import to ensure the integrity of the data and correct possible errors.

2.1.2 Preprocessing of the data and derived features

Some of the variables provided by the TELCO had to be preprocessed in order to be used as training data in a supervised learning setting. The first example of preprocessing was the *boolean categorization* of all variables of type 'string': for all possible string values associated to such a variable, a list of boolean features was created, all set to *false* except the actual value for the sample data. An example of this operation for variable *CHURN_IN_OP* is shown in Table 2.1.

Another preprocessing operation concerned all variables of type 'date': they were replaced with the number of months spent between the month of the date and the month of reference for the data. This operation can be formalized as this:

Def. 2.1.1. Let d_{ref} be the reference month of the data, with M_{ref} being the actual month and Y_{ref} year of reference, and d a variable of type 'date' whose month is M_d and year Y_d. Then d is processed as follow:

$$d = ((Y_{\text{ref}} - Y_d) \cdot 12) + (M_{\text{ref}} - M_d)$$

Variable	Description	Type
Basic Informations		
CLIENT_ID	id of the owner of the sim card	string
SEGMENT	segment of the client, based on the nr of associated sim cards. It can be SEG1 or SEG2	string
CLASS	class of the sim card: either CLASS#1 or CLASS#2	string
TYPE	type of sim card: either *voice* or *data*	string
CONTRACT	id of the type of contract	string
ACT_DATE	date of activation	date
CHURN_IN_OP	operator from which the sim card churned from (null if there was no churn)	date
Deactivation		
DEACT_DATE	date of deactivation	date
DEACT_CAUSE	cause of deactivation	string
CHURN_OUT_OP	operator where the sim card churned to (null if the cause of deactivation is not churn)	date
Usage		
IN_VOICE_X	minutes of incoming calls received by a number belonging to X operator	number
OUT_VOICE_X	minutes of outgoing calls made to a number belonging to X operator	number
IN_SMS_X	nr of incoming sms received by a number belonging to X operator	number
OUT_SMS_X	nr of outgoing sms sent to a number belonging to X operator	number
VOL_DATA	Bytes of internet data	number

Incomes		
VOICE_INC	income derived from calls	number
SMS_INC	income derived from sms	number
DATA_INC	income derived from internet usage	number
Terminal		
NR_TERMINAL	total number of terminals owned by the client owning the sim card	number
NR_TERMINAL_ACT	total number of active terminals owned by the client owning the sim card	number
Complaints and assistance		
NR_ASSIST	nr of calls carried out to receive assistance	number
NR_COMPL	nr of carried out complaints	number
NR_COMPL_FRAUD	nr of complaints carried out because of fraud event	number

Table 2.2: Most important variables provided for each sim card.

This process led to the creation of two variables of great importance:

- ***AGEING_SIM***, derived from variable *ACT_DATE*, that provided the 'age' of the sim card, and
- ***MONTHS_TO_DEACT***, derived from variable *DEACT_DATE*, that provided the number of months before the deactivation of the sim card.

It must be noted that, according to Def. 2.1.1, the values of *AGEING_SIM* are positive, whereas those of *MONTHS_TO_DEACT* are negative. For sim cards that were still active, the date *DEACT_DATE* was fixed at January 1995, leading to a positive value for *MONTHS_TO_DEACT*. An example of this operation is shown in Table 2.3.

Alongside the preprocessed variables, two new sets of *derived variables* were constructed. The first set included *variations* in the usage variables associated to the sim card, both in terms of voice/sms and internet data. Usually these variations concerned

SIM_ID	CHURN_IN_OP
SIM#1	OP#2
SIM#2	OP#1
SIM#3	None
SIM#4	OP#5

→

SIM_ID	CHURN_IN_OP#1	CHURN_IN_OP#2	...	CHURN_IN_OP#5
SIM#1	0	1	...	0
SIM#2	1	0	...	0
SIM#3	0	0	...	0
SIM#4	0	0	...	1

Table 2.1: Example of boolean categorization performed on the non-numerical values of variable *CHURN_IN_OP*. Similar categorizations were performed on all variables of type 'string'.

SIM_ID	Ref month	ACT_DATE	DEACT_DATE
SIM#1	2015-07	2011-03-15	2015-09-14
SIM#2	2015-07	2009-07-15	2015-08-17
SIM#3	2015-07	2014-06-01	Null
SIM#4	2015-07	2014-04-29	2015-09-01

→

SIM_ID	Ref month	AGEING_SIM	MONTHS_TO_DEACT
SIM#1	2015-07	52	-2
SIM#2	2015-07	72	-1
SIM#3	2015-07	13	126
SIM#4	2015-07	15	-2

Table 2.3: Date variables were replaced with numerical variables expressing the temporal lag (in months) between the date and the month of reference of the data. When a date was not available, a generic date referring to January 1995 was considered for the computations.

the usages associated to the month of reference and those of the previous 6 months, and were computed through different statistical operators (mean, standard deviation, median).

The second set featured a number of derived variables associated to the *client* owning the sim card. The presence of variable *CLIENT_ID* allowed to carry out an *aggregation* of different variables, including the number of sim cards associated to the client, the number of activations and deactivations occurred during the last N months (usually 3 months were considered) and data concerning the age of the client. A complete overview of this set is reported in Table 2.4.

Variable	**Description**
CLIENT_NR_VOICESIM	total number of voice sim cards
CLIENT_NR_DATASIM	total number of data sim cards
CLIENT_ACT_N	number of activated sim cards in the last N months
CLIENT_CHURN_IN_N	number of activated sim cards that churned from another operator in the last N months
CLIENT_DEACT_N	number of deactivated sim cards in the last N months
CLIENT_CHURN_OUT_N	number of deactivated sim cards that churned to another operator in the last N months
CLIENT_VAR_VOICESIM	variation in the number of voice sim cards
CLIENT_VAR_DATASIM	variation in the number of data sim cards
MIN_CLIENT_AGE	age of the youngest sim card
MAX_CLIENT_AGE	age of the oldest sim card

Table 2.4: Aggregated variables derived for each client.

2.1.3 Preliminary analyses

Before starting the modeling of the problem and the predictive phase, all data were analyzed and checked up with TELCO1 in order to ensure the correctness of the import. Sims were divided according to their segment, as well as their type and their class. Particular care has been reserved to those sim cards belonging to segment SEG1, according to the wishes of the TELCO.

For privacy reasons, the contents of the performed analyses cannot be reported directly. Some evidences for segment SEG1 however were:

- the number of sim cards belonging to CLASS#1 increased almost linearly throughout the year, whereas those of type CLASS#2 decreased;
- the usages, both in terms of voice, sms and data, had different behaviors but shared some common patterns, i.e. a sensible increase during the first months of the year and a negative peak during August;
- more than half of the clients owned a single sim of type VOICE;
- there was a small amount of sim cards with no usage associated ('silent' sim cards).

As for the churners, the number of sim cards that churned to another operator varied noticeably from month to month, but no regular trend could be found. Analyses suggested that the actual period of the year was not highly meaningful for the purpose of churn prevention, and that the attention should be focused on the history of a sim card as well as of its owner. This supported the idea of approaching each experimental setting on a monthly base, as described in the following section.

2.2 Modeling of the problem and labeling strategy

This problem was modeled as a *direct classification* of the churning of single sim cards. The input data points were identified with the monthly collection of original and derived

variables associated to a sim card and referring to a given month of reference. As for the labeling strategy, a sim card was considered a churner, i.e. was positive labelled, if its deactivation occurred within a fixed number of months N_{month}, usually set to 2.

Formally, the described labeling strategy can be defined as this:

Def. 2.2.1 (Labeling of a sim card)**.** Let (s, d) be a point identified by a sim card s at a given month of reference d, with M_d and Y_d being the associated month and year respectively. Fixing N_{month}, the point (s, d) is labelled as following:

$$churner(s,d) = \begin{cases} 1 & \text{if } 1 \leq MONTHS\_TO\_DEACT(s,d) \leq N_{\text{month}}, \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $MONTHS\_TO\_DEACT(s, d)$ is defined according to Def. 2.1.1.

Following a discussion with the client, it was agreed to analyze the two segment of sim cards, SEG1 and SEG2, separately. These segments included clients with very different types of contract and number of associated sims, and the strategies employed by the TELCO to deal with possible churners differed too. During all the experimental phase of the project, analyses and predictions were carried out only on sim cards belonging to segment SEG1. For this reason, in the following sections data are always referred to that segment. Once the procedure of prediction was fixed, analyses were extended also to the other segment of the dataset.

2.3 Results

An initial operation of *model assessment* led to the best type of *classifier*, which turned out to be the *random forest* classifier. From that moment on, the type of classifier remained unchanged.

The experimental setting was essentially the same throughout all the predictions. Fixing a reference month M, to predict the churners of months $M + 1$ and $M + 2$ the model was initially trained on a training set referred to month $M - 2$, and then it was

applied to the test set M. In some occasions the training set was partitioned according to the values of a given variable (i.e. variable *AGEING_SIM*, see Section 2.3.3) in a set of smaller training sets. Only at the end of the analyses it was decided to expand the training set to other two months, $M-3$ and $M-4$, in order to improve the number of samples belonging to some datasets (see Section 2.4).

The output of each prediction was a list of sim cards ordered according to the score of the prediction, i.e. the probability that they churned in the next two months. To evaluate the goodness of the prediction, alongside the ROC curve's AUC (see Section 1.7.2) the percentage of churners in the top K list was compared to the percentage of churners in all the monthly dataset (baseline). The ratio between these two percentages is referred to as *lift*.

2.3.1 Model assessment

The first run of experiments aimed at finding the best type of model to be used for the classification of churning sim cards. Four types of model were considered:

- Decision Tree,
- Linear Ridge Regression,
- Kernel SVM (RBF),
- Random Forest.

For each model, the related parameters were tuned via a cross validation using the AUC of the ROC curves as measure of performance (see Section 1.7.2). Fixing a month M, the model assessment went as follows (see Figure 2.1):

1. the cross validation was carried out on month $M-4$, identifying the best parameters for each type of model,
2. the resulting parameters were used to train four different classifiers on month $M-2$, and

3. tests were performed on month M, where the AUCs were used to identify the best type of model.

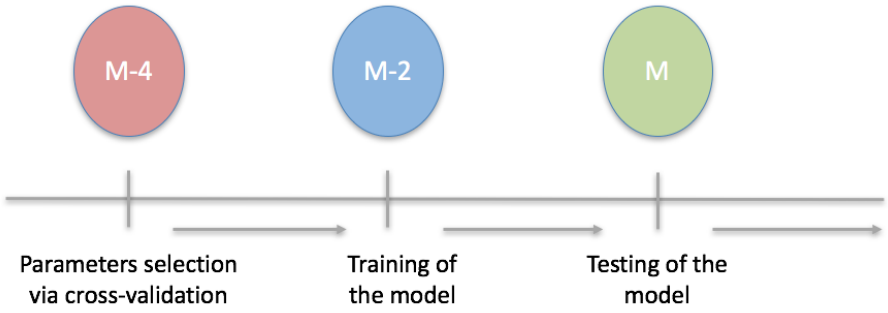


Figure 2.1: Usage of monthly dataset during the operation of model assessment.

The data used to carry out this model assessment comprised all sim cards belonging to segment SEG1.

Results are reported in Figure 2.2 and Table 2.5.

From this analysis, the best kind of model turned out to be the Random Forest model. All next experiments were therefore carried out using this classifier. A deeper study of its parameters selection was also performed, whose results are reported in Table 2.6. For the Random Forest classifier, the parameters that were cross-validated were the maximum depth, the number of estimators, the maximum number of features and the minimum number of samples split. It can be seen that there is a good stability, especially regarding the maximum depth and the minimum number of samples split.

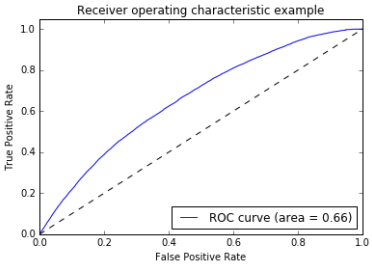
2.3.2 Preliminary results

Following the model assessment, a first run of experiments was undertaken to estimate the average goodness of the predictions. Again, the datasets comprised only sim cards belonging to segment SEG1.

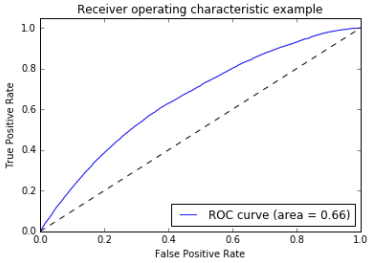
Table 2.7 shows the AUCs associated to different predictions, as well as the number of churners identified in the Top 10K list (true positive samples).

Table 2.8 shows the performances of the predictions in terms of the lift. As it can

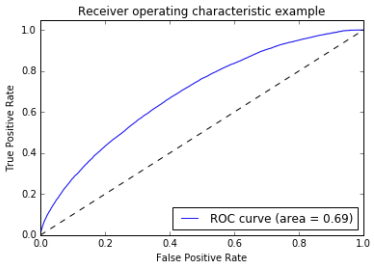
Model assessment



(a) Linear Ridge Regression



(b) Kernel SVM - RBF



(c) Random Forest

Figure 2.2: Test ROC curves associated to the predictions obtained by different types of model. The reference month is September 2015. The dataset comprised all sim cards of segment SEG1

.

Reference month	DT	Ridge	RBF-SVM	RF
2015-05	0.67	0.65	0.68	0.71
2015-07	0.67	0.67	0.66	0.69
2015-09	0.65	0.66	0.66	0.69

Table 2.5: Comparison of the AUC of the ROC curves related to best models of type Decision Tree (*DT*, Linear Ridge regression (*Ridge*), Kernel SVM of type RBF (*RBF-SVM*) and Random Forest (*RF*). Column 'Reference month' describes the testing month of the experiment. The best parameters of each model were found using cross validation.

Month of test	Maximum depth	Number of estimators	Maximum features	Minimum samples split	AUC
2015-04	15	150	20	500	0.718
2015-05	15	50	20	500	0.713
2015-06	15	150	25	500	0.705
2015-07	15	150	30	500	0.693
2015-09	15	150	30	500	0.695

Table 2.6: Best parameters of Random Forest classifier found through cross validation and associated AUCs.

be seen, there is a meaningful increase of churners inside the Top 10K sims highlighted by the classifier.

Tables 2.9 and Figure 2.3 are related to the experiment with reference test month January 2016: Table 2.9 reports the lifts in the Top 1K, 2K, 5K and 10K lists, while Figure 2.3 shows the behavior of the percentage of churners throughout all the data. In this last Figure, sim cards are ordered in the x-axis according to the score computed by the classifier, and the horizontal line represents the baseline of the dataset. The detail of the Top 10K sims is highlighted in the upper right corner.

A deeper analysis showed that the classifier was able to identify also a good number of sim cards that churned *after* two months. While sim cards churning after 3 or 4 months were considered negative examples by the employed scoring function (see Section 2.2), intercepting early sign of churning maybe considered as a feature of the model. Table 2.10 shows the monthly distribution of churners associated to the experiment with reference test month January 2016. As it was to be expect, the lift decreases the further we went from the reference month. Note that, at the time of the experiment, the data associated to May 2016 were partial.

Since the classifier was of type Random Forest, it was possible to retrieve the list of the features mostly used by the model to carry out the predictions. Even though

Train month	Test month	AUC	Churners in Top 10K
2015-05	2015-07	0.700	1327
2015-07	2015-09	0.706	1710
2015-09	2015-11	0.710	1735
2015-11	2016-01	0.711	1848

Table 2.7: Preliminary prediction results.

Train month	Test month	% Churners (baseline)	% Churners in Top 10K	Lift
2015-05	2015-07	3.34%	13.27%	3.97
2015-07	2015-09	4.54%	17.10%	3.76
2015-09	2015-11	4.82%	17.35%	3.59
2015-11	2016-01	4.95%	18.48%	3.73

Table 2.8: Comparison between lifts associated to predictions on sim cards with all types of class.

Top K list	% Churners (baseline)	% Churners in Top K	Lift
Top 10K	4.95%	18.48%	3.73
Top 5K	4.95%	22.26%	4.50
Top 2K	4.95%	26.40%	5.33
Top 1K	4.95%	29.70%	6.00

Table 2.9: Behavior of the lift at the increasing of the Top K list. Data associated to reference month January 2016.

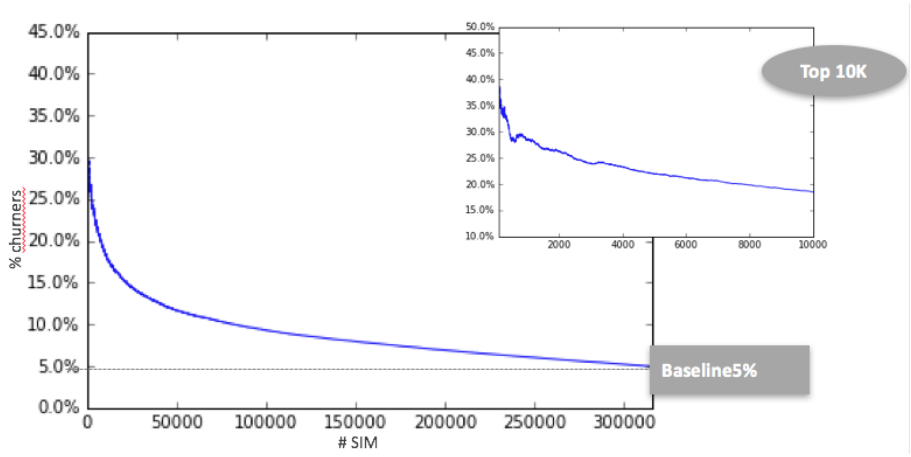
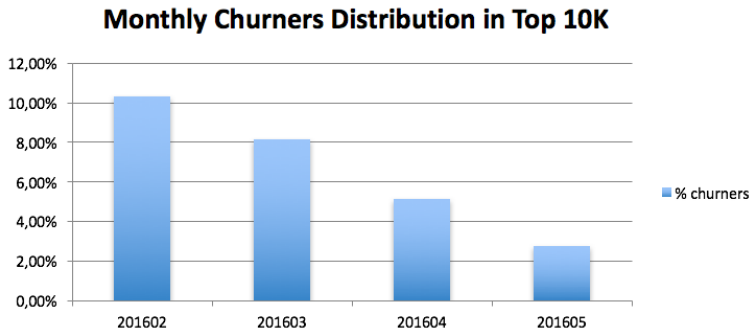


Figure 2.3: Percentage of the churners according to the score of the sim cards, with detail of the Top 10K list. Data associated to reference month January 2016.



	2016-02	2016-03	2016-04	2016-05*
Nr Churners	1033	815	512	277
% Churners in Top 10K	10.3%	8.1%	5.1%	2.7%
% Churners (baseline)	2.3%	2.6%	1.9%	1.1%
% Lift	4.5	3.1	2.7	2.5

Table 2.10: Monthly distribution of churners associated to the experiment with reference test month January 2016.
* partial data.

their ranking changed from experiment to experiment, some turned out to be of high importance in most of the cases, i.e. those concerning the *ages* of both sim cards and

client and the number of sim cards owned by the client. Variable $AGEING\_SIM$ in particular had an interesting behavior: in the Top 10K it turned out that the ages of the sim cards were concentrated around 25 months, as shown in Table 2.11 for the experimental dataset referred to January 2016 (the other datasets provided analogous results). When these evidences were given to the TELCO, they reported that 25 months was a common deadline for many contractual agreements. This fact led to the decision to split the experimental dataset according to the age of sim cards. These experiments and related results are shown in the next Section.

	Top 10K	**Total population**
mean	25.5	26.8
std	6.5	34.2
25%	23.0	7.0
50%	25.0	15.0
75%	27.0	32.0

Table 2.11: Statistics on variable $AGEING\_SIM$ (number of months) in Top 10K list compared to those of the total population. Data associated to reference month January 2016.

2.3.3 Splitting according to the age of sim cards

Following the evidences observed in the first run of experiments, it was decided in agreement with the TELCO to split the datasets according to the values of variable $AGEING\_SIM$. The aim was to isolate those sim cards with age near the deadline of their contractual agreement, which usually was around 25 months. These sim cards were divided from the others restricting variable $AGEING\_SIM$ inside a range of $[22, 28]$ months. This operation split the dataset in three different sets, as shown in Figure 2.4: sim cards whose age was *under* 22 months, *between* 22-28 months and *over* 28 months.

In the first experiments, the two outer datasets were merged, leading to a partition of

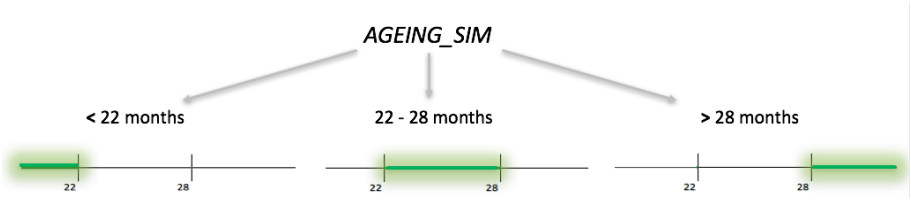


Figure 2.4: Splitting of datasets according to the values of variable $AGEING\_SIM$.

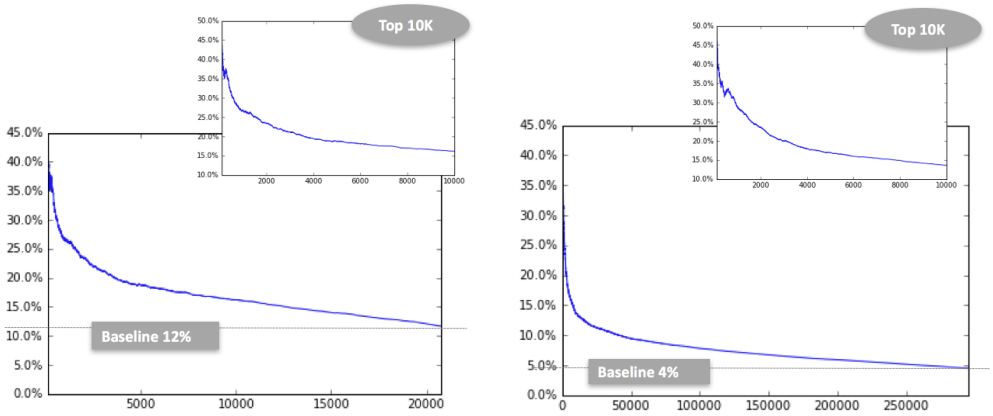
only two elements. The resulting two models had different performances: even though the numbers of correct churners identified in the Top 10K lists were similar, for the dataset comprising sim cards with age inside the range $[22, 28]$ the baseline was high, usually over 10%, thus leading to smaller values for the lifts. Table 2.12 and Figure 2.5 shows the result obtained for the experimental settings associated to January 2016. Similar results were found for the other months.

Later on, the 'outer' dataset was split in two different parts, and a different model was trained on each of them. This splitting turned out to be more performant, especially if the Top list was restricted to the first 2K samples, and was used in a more systematic way to deliver monthly results (as described in Section 2.4). The disadvantage of this strategy was the small number of samples available for some of the sets, especially that associated to the inner range [22, 28] months. To overcome this limitation, the training was usually performed not only on the data associated to month $M - 2$, but also to months $M - 3$ and $M - 4$.

Tables 2.13, 2.14, 2.15 and 2.16 show the results obtained from datasets associated to different reference months. Here the training month was always of only one month. Note that data associated to March 2016 were partial.

2.3.4 Other secondary experiments

Different sets of experiments were undertaken before stabilizing on the training procedure described in the previous sections. They did not lead to significant increases in the performances, and even though they sometimes highlighted interesting insights of the problem, they were not considered meaningful enough to be carried on systematically.



(a) Inside range [22, 28] (b) Outside range [22, 28]

Figure 2.5: Percentage of the churners according to the score of the sim cards, with detail of the Top 10K list. Data associated to reference month January 2016.

Inside range $[22, 28]$

Nr samples	20756
Nr churners	2420

In Top 10K

Nr churners	1620
% churners	16.20%
Baseline	11.65%
Lift	1.39

Outside range $[22, 28]$

Nr samples	295139
Nr churners	13220

In Top 10K

Nr churners	1366
% churners	13.66%
Baseline	4.46%
Lift	3.05

Table 2.12: Statistics on model performances. Data associated to reference month January 2016.

< 22 months	
Nr samples	204150
Nr churners	6950

22 - 28 months	
Nr samples	21456
Nr churners	2397

> 28 months	
Nr samples	91803
Nr churners	4297

In Top 2K	
Nr churners	348
% churners	17.4%
Baseline	3.3%
Lift	5.14

In Top 2K	
Nr churners	501
% churners	25.0%
Baseline	11.1%
Lift	2.24

in Top 2K	
Nr churners	395
% churners	19.7%
Baseline	4.6%
Lift	4.21

Table 2.13: Statistics on model performances. Data associated to reference month December 2015.

< 22 months	
Nr samples	204127
Nr churners	8446

22 - 28 months	
Nr samples	20756
Nr churners	2420

> 28 months	
Nr samples	91012
Nr churners	4774

In Top 2K	
Nr churners	434
% churners	21.7%
Baseline	4.1%
Lift	5.24

In Top 2K	
Nr churners	442
% churners	22.1%
Baseline	11.6%
Lift	1.89

in Top 2K	
Nr churners	396
% churners	19.8%
Baseline	5.2%
Lift	3.77

Table 2.14: Statistics on model performances. Data associated to reference month January 2016.

< 22 months	
Nr samples	206164
Nr churners	7966

In Top 2K	
Nr churners	393
% churners	19.6%
Baseline	3.8%
Lift	5.08

22 - 28 months	
Nr samples	19728
Nr churners	2260

In Top 2K	
Nr churners	407
% churners	20.3%
Baseline	11.4%
Lift	1.78

> 28 months	
Nr samples	90044
Nr churners	4402

in Top 2K	
Nr churners	396
% churners	18.4%
Baseline	4.8%
Lift	3.77

Table 2.15: Statistics on model performances. Data associated to reference month February 2016.

< 22 months	
Nr samples	208974
Nr churners	5236

In Top 2K	
Nr churners	342
% churners	17.1%
Baseline	2.5%
Lift	6.8

22 - 28 months	
Nr samples	19876
Nr churners	1542

In Top 2K	
Nr churners	351
% churners	17.5%
Baseline	7.7%
Lift	2.26

> 28 months	
Nr samples	88531
Nr churners	3048

in Top 2K	
Nr churners	341
% churners	17.0%
Baseline	3.4%
Lift	4.95

Table 2.16: Statistics on model performances. Partial data associated to reference month March 2016.

Splitting by Class. Before the splitting based on variable $AGEING\_SIM$, different experiments were carried out dividing the sim cards according to their variable $CLASS$. The results of this procedure are shown in Table 2.17. It turned out that the performances were better for sim cards of class $CLASS\#2$, but significantly worse for those belonging to $CLASS\#1$. After a confrontation with the TELCO it was decided to keep the entire dataset for future analyses, thus leaving this splitting strategy behind.

Study of the number of Complaints. Once the splitting strategy based on variable $AGEING\_SIM$ was fixed, a throughout analysis of the behaviors of the other variables was carried out. The aim was to find some empirical indicators that could confirm the goodness of the predictive models, and at the same time help the experts of the TELCO to have new insights in the problem of churn prevention.

The variables associated to the number of complaints, NR_COMPL, showed some interesting patterns that differed from split to split. In Figure 2.6, sim cards in the x-axis are ordered according to the score retrieved by the predictions. It can be seen that:

- for sim cards with $AGEING\_SIM < 22$ months, two different clusters of sim cards with higher values of NR_COMPL were localized at the extremes of the scores;
- for sim cards with $AGEING\_SIM$ inside range [22, 28] months, there seemed to be a positive correlation between the score and the value of NR_COMPL, at least following the first 2K top sim cards;
- for sim cards with $AGEING\_SIM > 28$ months, lower scores seemed to be associated with some irregular peaks of NR_COMPL.

These behaviors were common in four different datasets, as reported also in Figures 2.7 and 2.8. Experiments were undertaken to try to produce positive correlations also in the splits associated to the 'outer' sets. In the end, this result was accomplished by considering only sim cards associated to clients with less than 40 sims. Figure 2.9, 2.10

$CLASS\#1 + CLASS\#2$

Train Month	Test Month	AUC	Top 10K churners
2015-03	2015-05	0.72	1971
2015-04	2015-06	0.69	1557
2015-05	2015-07	0.69	1308
2015-07	2015-09	0.70	1677
2015-09	2015-11	0.69	2105

$CLASS\#1$

Train Month	Test Month	AUC	Top 10K churners
2015-03	2015-05	0.65	696
2015-04	2015-06	0.64	639
2015-05	2015-07	0.65	773
2015-07	2015-09	0.65	997
2015-09	2015-11	0.63	1156

$CLASS\#2$

Train Month	Test Month	AUC	Top 10K churners
2015-03	2015-05	0.75	2463
2015-04	2015-06	0.72	1536
2015-05	2015-07	0.70	1210
2015-07	2015-09	0.70	1604
2015-09	2015-11	0.74	1991

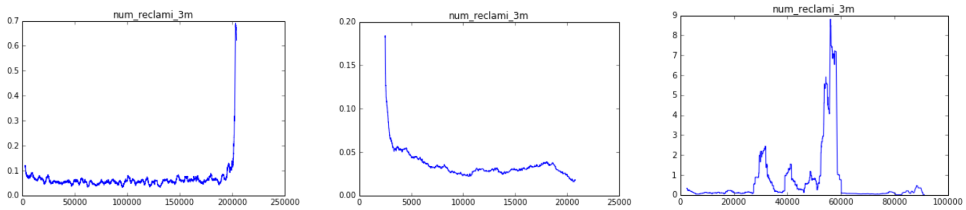
Table 2.17: Performances on dataset split according to variable $CLASS$.

and 2.11 show that in the first split the second cluster of sim cards disappeared, while in the third split the irregular peaks were replaced by a more linear behavior. However, the performances of these experiments were unstable, and soon they were left out.

2.4 Conclusions

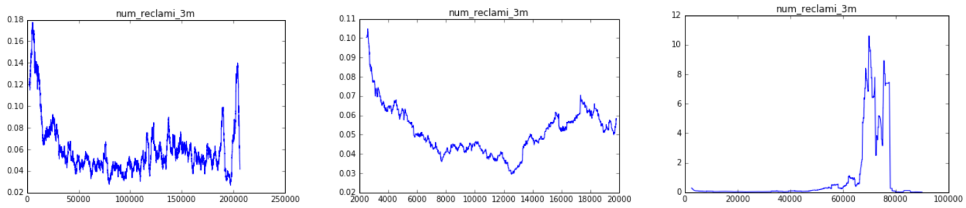
The splitting strategy based on variable $AGEING\_SIM$ turned out to be the most performing and appreciated by the TELCO, and was used for over two years to deliver monthly lists of potential churners. A new set of data associated to sim cards belonging to segment SEG2 was provided, and after some analyses a Random Forest classifier with slightly different parameters was fixed to perform predictions.

Each month, a total of six different models were trained of the datasets associated to different segments and ageing values. To improve the robustness, after some months it was decided to increase the training datasets by including samples not only related to month $M-2$, but also months $M-3$ and $M-4$. Once the predictions were made, lists containing the scores associated to each sim cards were produces and delivered to the TELCO.



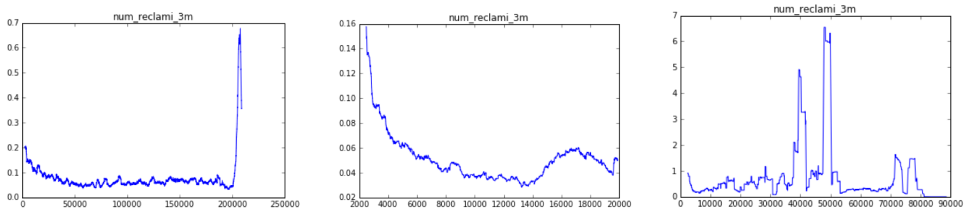
(a) < 22 months (b) Inside range [22, 28] (c) < 22 months

Figure 2.6: Behavior of variable NR_COMPL according to the score of the sim cards. Data associated to reference month January 2016.



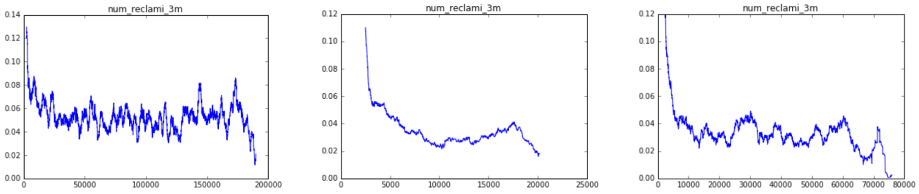
(a) < 22 months (b) Inside range [22, 28] (c) < 22 months

Figure 2.7: Behavior of variable NR_COMPL according to the score of the sim cards. Data associated to reference month February 2016.



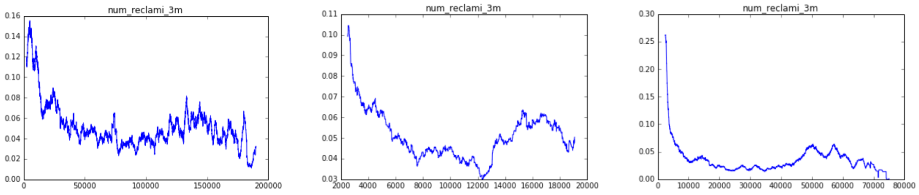
(a) < 22 months (b) Inside range [22, 28] (c) < 22 months

Figure 2.8: Behavior of variable NR_COMPL according to the score of the sim cards. Data associated to reference month March 2016 (partial data).



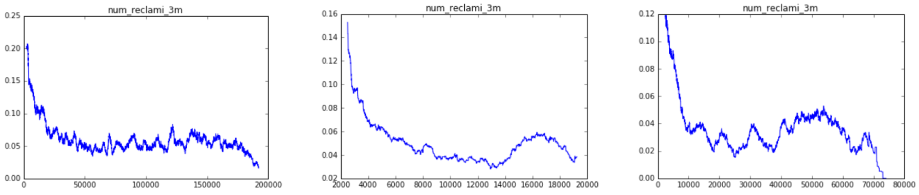
(a) < 22 months (b) Inside range [22, 28] (c) < 22 months

Figure 2.9: Behavior of variable NR_COMPL according to the score of the sim cards for dataset restricted to clients with less than 40 sim cards. Data associated to reference month January 2016.



(a) < 22 months (b) Inside range [22, 28] (c) < 22 months

Figure 2.10: Behavior of variable NR_COMPL according to the score of the sim cards for dataset restricted to clients with less than 40 sim cards. Data associated to reference month February 2016.



(a) < 22 months (b) Inside range [22, 28] (c) < 22 months

Figure 2.11: Behavior of variable NR_COMPL according to the score of the sim cards for dataset restricted to clients with less than 40 sim cards. Data associated to reference month March 2016 (partial data).

Chapter 3

Churn Prevention #2

The problem addressed by TELCO2 from the high level perspective greatly resembled the one exposed in the previous chapter. The company provided monthly batches of data associated to sim cards, and asked for the lists of most likely churners. Many of the data associated to sim cards overlapped those exposed in Section 2.1, and the general behavior of the population had many common traits.

For all these reasons, the evidences discovered during the analyses of TELCO1's data were considered as a base for the classification procedures described in this chapter. One of the major differences between the two dataset was the magnitude of the considered data. In this project, every month around one million sim cards had to be processed and used to train the classifiers, leading to computational time's considerations that had to be taken into account throughout the project.

In this chapter, Section 3.1 describes the data provided by the company, as well as the preliminary analyses, the formal modeling of the problem and the followed labeling strategy, which was equal to the one given in the previous chapter. In Section 3.2 a set of subsections reports all the performed experiments and related results.

3.1 Data analysis

3.1.1 Type and storing of the data

The data provided by TELCO2, as for TELCO1, consisted of batches of monthly data arranged into four different excel files (*.xlsx):

- File#1: it contained *basic information* on the sim cards, such as the type of the contract, the segment of the sim card, personal informations on the owner of the sim card (client), as well as the fees and all informations regarding the *activation* and the *deactivation*.
- File#2: it contained all data concerning the *usage* of the sim cards during the month of reference, i.e. voice and sms data, divided according to the operator of destination, and internet usages.
- File#3: it contained information related to all the changes of phone plan occurred since the activation of the sim card.
- File#4: it contained a complete history of the contacts occurred between the sim card and the *assistance number* of the TELCO, which dates back to the date of activation of the sim card. There are two possible types of contact, *type_1* and *type_2*.

A list of the most important features is provided in Table 3.1.

All files had as primary keys the *id* of the sim card and the *reference month* (for data in File#4 the reference month was derived from the dates). For the first two files, this second key was always equal to the reference month of the batch, while in the other two files it changed according to the single change of plan or contact described by the data. These keys allowed the *merging* of these different data in the preprocessing phase (see next Section).

Unlike what has been done for TELCO1, this time data were not imported inside a database structure: since the files were already well-structured and tidy, it was decided

Variable	Description	Type
***Features from* File#1**		
CLIENT_ID	id of the owner of the sim card	string
SEGMENT	segment of the sim card.	string
OPTION	contract option of the sim card.	string
DEACT_CAUSE	cause of deactivation	string
EXPIRY_DATE	date of expiration	date
MONTHLY_FEE	fee on reference month	number
ACT_DATE	date of activation	date
CHURN_IN_OP	operator from which the sim card churned from (null if there was no churn)	string
DEACT_DATE	date of deactivation (January 1900 if sim still active at the end of the month)	date
CHURN_OUT_OP	operator where the sim card churned to (null if there was no churn to another operator)	string
***Features from* File#2**		
MINUTI_IN_X	minutes of incoming calls received by a number belonging to X operator	number
MINUTI_OUT_X	minutes of outgoing calls made to a number belonging to X operator	number
IMPORTO_MIN_OUT_X	revenues from outgoing calls made to a number belonging to X operator	number
NUM_SMS	nr of outgoing sms	number
IMPORTO_SMS	revenues from outgoing sms	number
NUM_MMS	nr of outgoing mms	number

IMPORTO_MMS	revenues from outgoing mms	number
VOLUME_DATA	bytes of internet data	number
IMPORTO_DATA	revenues from internet usage	number
***Features from* File#3**		
DES_PLAN	new phone plan	string
DATE_CP	date of the change of phone plan	date
***Features from* File#4**		
TYPE_1_DATE	date of the contact of type *type_1*	date
TYPE_2_DATE	date of the contact of type *type_2*	date

Table 3.1: Most important variables listed according to the file of storage. Each feature is associated to a sim card and, with the exception of those stored inside File#4, a reference month. For features provided by File#2, data associated to different operators *X* were provided.

to keep them as main source of data, importing the useful ones at the beginning of each analysis.

3.1.2 Preprocessing of the data and derived features

For each reference month used during the analysis the associated batch of four files described in the previous section had to be imported. The data provided by each type of file had to be preprocessed separately in order to be used for the predictions. Many of the operations described in this section were performed also on the data associated to TELCO1 (see Section 2.1.2).

The data which underwent most preprocessing were those related to the basic informations of the sim cards, stored inside File#1. The most important preprocessing steps were:

- the *boolean categorization* of all variables of type 'string'. In this problem some variables such as *DES_PLAN* (see Table 3.1) had a wide range of different values, most of which rarely used. Such values were merged inside a common value,

usually referred to *OTHER*. An example of this operation can be seen in Table 3.2.

- the *replacement* of all variables of type 'date' with the number of months passed between the date and the month of reference for the variable. The formal definition of this process is stated in Def. 2.1.1, provided in Section 2.1.2. As in the problem associated to TELCO1, from variable *ACT_DATE* it was derived a new variable ***AGEING_SIM*** describing the 'age' of the sim card, and from variable *DEACT_DATE* it was derived variable ***MONTHS_TO_DEACT*** providing the number of months before the deactivation of the sim card. Again, the values of *AGEING_SIM* are positive, and those of *MONTHS_TO_DEACT* are negative if *DEACT_DATE* is not January 1900 (i.e. there was a deactivation of the sim card). It must be noted that also for variable *ACT_DATE* some values were equal to January 1900. An example of this operation is shown in Table 3.3.

- the *filling* of invalid values. Invalide values included both null values (which occurred mainly inside File#2) and high months range that derived from dates with value equal to January 1900. A proper filling value was chosen according to the variable that needed to be filled (i.e. zeros for variables associated to usages, and fixed positive or negative numbers for date variables).

Thanks to variable $CLIENT\_ID$ a set of *derived features* associated to the owner of the sim card was created. They were:

1. the number of *owned* sim cards in the current and in the last 2, 3 and 4 months,
2. the number of *activated* sim cards in the current and last 3 months,
3. the number of *churn in* sim cards, i.e. the sim cards activated because of a churn from another operator, in the current and in the last 3 months,
4. the number of *deactivated* sim cards in the current and in the last 3 months,
5. the number of *churn out* sim cards, i.e. the sim cards deactivated because of a churn to another operator, in the current and in the last 3 months, and

SIM_ID	OPTION
SIM#1	OPTION_2
SIM#2	OPTION_1
SIM#3	OPTION_26$^{(rare)}$
SIM#4	OPTION_19$^{(rare)}$

$\rightarrow$

SIM_ID	OPTION_1	OPTION_2	...	OPTION_OTHER
SIM#1	0	1	...	0
SIM#2	1	0	...	0
SIM#3	0	0	...	1
SIM#4	0	0	...	1

Table 3.2: Example of boolean categorization performed on the non-numerical values of a variable called *OPTION*. Note that options 26 and 19 were rarely used for this variable, so they were merged inside a common option *OPTION_OTHER*, Similar categorizations were performed on all variables of type 'string'.

SIM_ID	Ref month	ACT_DATE	DEACT_DATE
SIM#1	2016-08	2011-02-07	1900-01-01
SIM#2	2016-08	1900-01-01	2016-10-06
SIM#3	2016-08	2014-05-18	2016-11-12

$\rightarrow$

SIM_ID	Ref month	AGEING_SIM	MONTHS_TO_DEACT
SIM#1	2016-08	66	1399
SIM#2	2016-08	1399	-2
SIM#3	2016-08	29	-3

Table 3.3: Replacement of date variables with the monthly temporal lag between the month of the date and the month of reference of the data. The date January 1900 in the original data was associated to unknown or null values.

6. the number of *change plans* in the current and in the last 3 months.

From the first derived variables also the *variations* in the number of owned sim cards were computed. A global list of derived variables, as well as the original variables from which they were originated, is given in Table 3.4.

Because of the experience accumulated with TELCO1, in the first instance it was decided not to include derived features associated to the variations of the usages variables. In almost all experiments described in the previous chapter those variables had a low rank and were not used by the classifier to detect potential churners. The results exposed in Section 3.2 do not include these derived features, except for a set of experiments listed in Section 3.2.4. In this case, the outcomes confirmed previous evidences, and this line of action was thus discarded.

3.1.3 Preliminary analyses

For this problem TELCO2 provided at first data that collected from June 2016 to June 2017. On these data a set of preliminary analyses were carried out in order to retrieve some first evidences and understand the environment in which predictions had to be done. The following results are derived from the aggregation over all provided months of the data inside the four types of files (see Section 3.1.1). For this reason in this section the reference to a file File#X between File#1, File#2, File#3 and File#4 relates to all monthly files of type File#X provided by the client.

The first issue that was addressed was the overlap between sim cards belonging to the different files. The sim cards listed in File#1 were considered as the experimental base, and checks were performed to see if they had data associated to usages (File#2), change of plans (File#3) and contacts to the assistance number (File#4). It turned out that, on a total of 1.05 millions registered sim cards:

- the 14,29% (over 150K) had no usage data (missing in File#2),
- the 95,92% (slightly more than 1 million) had registered no change of plan (missing in File#3), and

Variable	Derived from	Description	X values
Nr_sim_X_mese	SIM_ID	nr activated sim cards in the last X months	1, 3
Nr_new_in_X_mese	SIM_ID ACT_DATE	nr activated sim cards in the last X months	1, 3
Nr_new_port_in_X_mese	SIM_ID ACT_DATE CHURN_IN_OP	nr churn in sim cards in the last X months	1, 3
churn_in_X_mese	SIM_ID DEACT_DATE	nr deactivated sim cards in the last X months	1, 3
churn_port_in_X_mese	SIM_ID DEACT_DATE CHURN_OUT_OP	nr churn out sim cards in the last X months	1, 3
CP_X_mese	SIM_ID DATE_CP	nr change of phone plans in the last X months	1, 3
sim_var_prec	Nr_sim_1_mese Nr_sim_2_mese	variation in the last month	-
sim_var_last	Nr_sim_1_mese Nr_sim_4_mese	variation in the last 3 months	-

Table 3.4: List of derived variables created from the aggregation on feature $CLIENT\_ID$. The aggregation was performed over different months, leading to associated derived features. The months taken into account are listed in column *X values*, where X refers to the month according to current month (i.e. 1 states for current month, 2 for the last month, 3 for the last but two months, etc).

- the 78,30% (over 820K) had no registered contact with the assistance number (missing in File#4).

Globally, only slightly less than 21K sim cards were common to all files. This limited overlapping was in part caused by File#4, where no sim card associated to

a month belonging to 2017 had a counterpart in File#1 (this happened because of a different encoding of sim cards ids during 2017).

It was agreed with TELCO2 to remove all sim cards with no usage data, and to discard variables derived from File#4 until new files related to 2017 could be provided. This is the reason why most of the analyses in Section 3.2 do not have features related to the number of contacts to the assistance number.

For privacy reason, the results of the preliminary analyses could not be included in this thesis. Some evidences however were:

1. most clients (97,50%) owned less than 10 sim cards, the 2,45% owned from 10 to 50 sim cards and only the 0,05% owned more than 50 sim cards;

2. the number of new activations and deactivations did not follow regular monthly patterns;

3. the main reason behind the deactivations of sim cards was a voluntary churn out to another operator;

4. sim cards belonging to different segments had churn statistics that differ significantly from each other;

5. the mean age of a sim card turned out to be around 1.5 - 2 years;

6. most of the revenues were associated to internet usage;

7. the main number of change of plans was around 1.003.

3.1.4 Modeling of the problem and labeling strategy

This problem was modeled as the one stated by TELCO1, i.e. a *direct classification* of the churning of single sim cards based on monthly features. The labeling strategy followed the definition stated in Def. 2.1.1, where data points were labelled as positive if the churn occurred within a fixed number of months from the month of reference.

3.2 Results

Since the experimental setting was very similar to the one described in the previous chapter and associated to data provided by TELCO1, the classifiers of type Decision Tree, Linear Ridge Regression and Kernel SVM (RBF) were not tested. The classifier that was taken into account alongside Random Forest was the *Gradient Boosting* classifier. These two models were compared during a phase of *model assessment*, with parameters derived from a *cross validation*.

The Gradient Boosting classifier proved to be the best classifier for these data, as well as the most performing in terms of computational times. The results are shown in Table 3.5.

Training month	**Testing month**	**RF**	**GB**
2016-10	2016-12	0.755	0.758
2016-11	2017-01	0.766	0.765
2016-12	2017-02	0.745	0.750
2017-01	2017-03	0.744	0.751
2017-02	2017-04	0.758	0.760

Table 3.5: Comparison of the AUC of the ROC curves related to best models of type Random Forest (*RF*) and Gradient Boosting (*GB*). The best parameters of each model were found using cross validation.

3.2.1 Preliminary results

In the first batch of experiments, no splitting on any given features was considered. As stated in Section 3.1.3, variables referring to the contacts to the assistance number were discarded because of coherence problems within the data. Also, derived variables associated to the variations in the usages of sim cards were omitted.

The experimental setting echoed the one used in the previous chapter. Taking M as the reference month, the training set consisted of data related to month $M-2$, on which

the classifier was trained in order to predict churn sim cards over the next two months. As a consequence of the derived variable computed through operations of aggregation described in Section 3.1.2, to compute a training set on month $M-2$ data were needed as far as month $M-5$: for this reason, even though TELCO2 provided data related to one year, only 5 experiments could be carried out, from December 2016 to April 2017.

The outcome of the experiments can be seen in Table 3.6. The AUC of Roc Curve's (see Section 1.7.2) was always over 0.75, with little variations throughout the test sets.

Train month	Test month	AUC	Churners in Top 2K
2016-10	2016-12	0.758	659
2016-11	2017-01	0.765	682
2016-12	2017-02	0.750	594
2017-01	2017-03	0.751	450
2017-02	2017-04	0.760	554

Table 3.6: Preliminary prediction results.

As for the lifts, lifts associated to the Top 2K sim cards were considered. The results can be seen in Table 3.7. Starting from a baseline around 2.6%, in the Top 2K sim cards found by the classifier the percentage of churners was always over the 20%, resulting in lifts higher then 8.8. The behavior of the lift associated to single experiments over all the data can be seen in Figure 3.2.

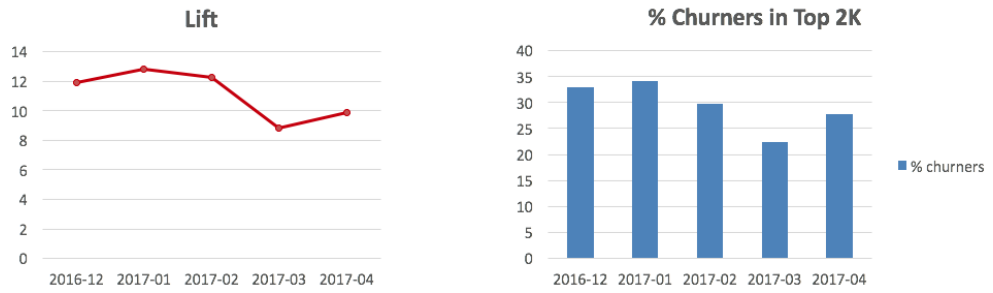
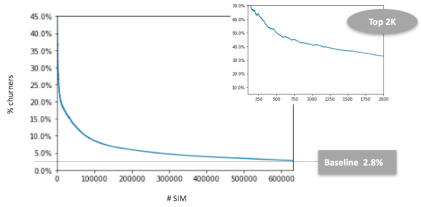
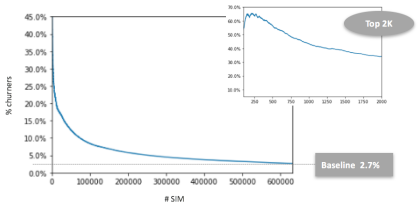


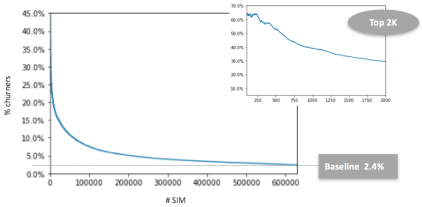
Figure 3.1: Behaviors of lifts and percentages of churners detected in the Top 2K list.



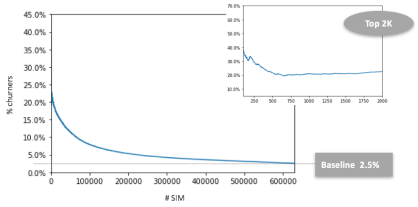
(a) Test month 2016-12



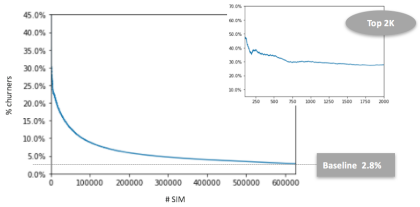
(b) Test month 2017-01



(c) Test month 2017-02



(d) Test month 2017-03



(e) Test month 2017-04

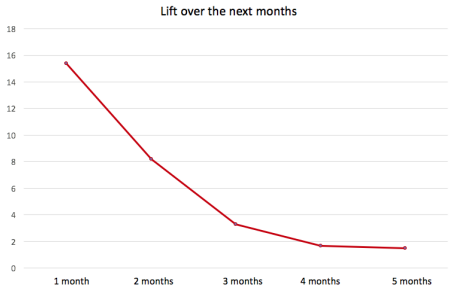
Figure 3.2: Percentage of the churners according to the score of the sim cards, with detail of the Top 2K list.

Train month	**Test month**	**% Churners (baseline)**	**% Churners in Top 2K**	**Lift**
2016-10	2016-12	2.77%	32.95%	11.9
2016-11	2017-01	2.66%	34.10%	12.8
2016-12	2017-02	2.42%	29.70%	12.2
2017-01	2017-03	2.55%	22.50%	8.8
2017-02	2017-04	2.80%	27.70%	9.9

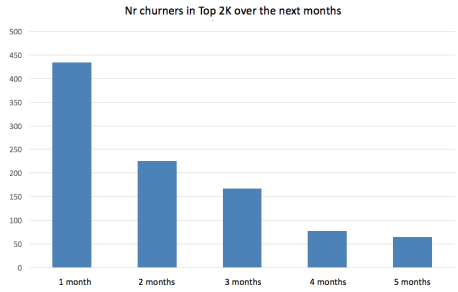
Table 3.7: Comparison between lifts associated to predictions.

Further analyses showed that the classifier associated to test month December 2016 included inside the Top 2K also a relevant number of sim cards that churned in the months belonging to 2017. Specifically, performances were still relevant considering the third month, see Table 3.8.

(a) Lift



(b) Number of churners

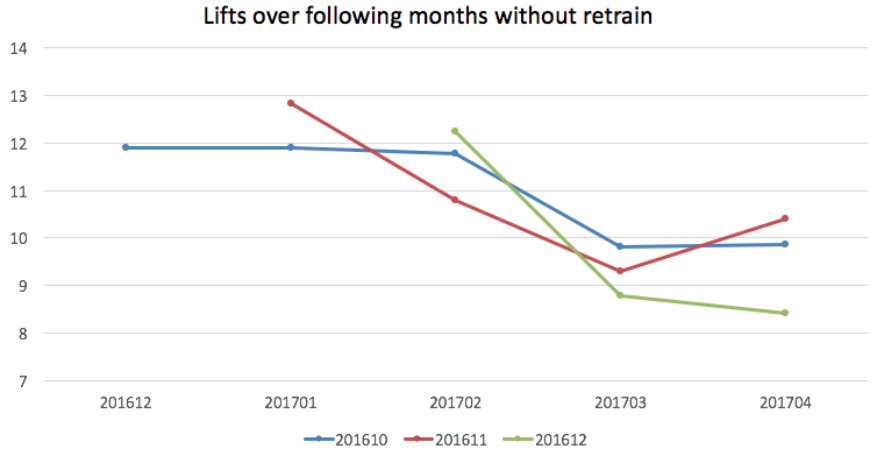


	Nr Churners in Top 2K	**% Churners (baseline)**	**% Churners in Top 2K**	**Lift**
M+1	434	1.40%	21.70%	15.4
M+2	225	1.37%	11.25%	8.2
M+3	166	2.53%	8.30%	3.3
M+4	77	1.91%	3.85%	1.7
M+5	64	2.50%	3.20%	1.5
M+6	58	2.28%	2.90%	1.3

Table 3.8: Behavior of lifts and percentages of churners over the next months for sim cards inside the Top 2K list. Data associated to test month M equal to December 2016.

Another studied element was the performance of a model trained on month $M-2$ applied to test sets associated to months following month M. The resulting lifts are shown in Table 3.9. As it can be seen, performances decreased the further we went from the training set. An exception was the model trained on November 2016, which had an increase of performances on test set April 2017, but this was considered a random event. This result confirmed the necessity to retrain the model every month in order to keep

high the quality of predictions.



		Test month				
		2016-02	**2017-01**	**2017-02**	**2017-03**	**2017-04**
Train month	**2016-10**	11.90	11.89	11.77	9.81	9.86
	2016-12		12.83	10.80	9.30	10.41
	2017-01			12.24	8.79	8.41

Table 3.9: Lifts obtained on following test months without retrain.

As for the top features used by the classifier, it turned out that the most important ones were those associated to the age of the sim cards (fact that led to the experiments described in the next section) and to the expiry date. This last variable was used as splitting variable during a following phase of the project, after having consulted the experts of TELCO2 on how to best define the splitting intervals for its values. This analysis is described in Section 3.2.5.

3.2.2 Splitting according to the age of sim cards

Following the evidences provided in the previous section, and also the ones obtained for the problem associated to TELCO1 data and described in the previous chapter, the behavior of variables $AGEING\_SIM$ was studied inside the Top 2K list retrieved by

the classifiers. Results are reported in Table 3.10. As it can be seen, in this experimental setting the age of sim cards gravitated towards a fixed value too, being around 31 months.

	Mean age in Top 2K	**Mean age in total population**
2016-12	31.5	23.15
2017-01	27.9	23.4
2017-02	32.4	23.6
2017-03	34.4	23.9
2017-04	29.2	24.5

Table 3.10: Mean values ofvariable $AGEING\_SIM$ (number of months) in Top 2K list compared to those of the total population. Data associated to reference month January 2016.

This led to a new set of experiments where the data were split according to the values of variable $AGEING\_SIM$. The threshold values were 25 and 35 months, and through them three subsets were created, as seen in Figure 3.3.

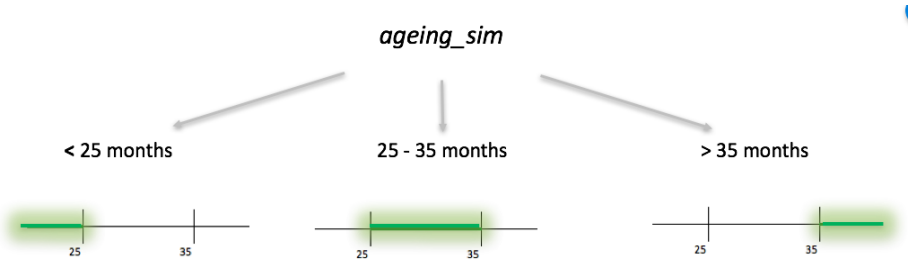
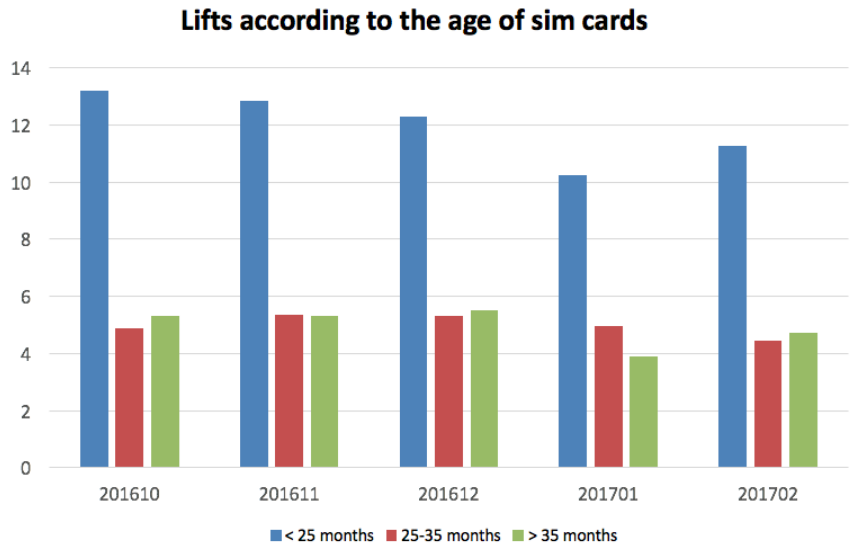


Figure 3.3: Splitting of datasets according to the values of variable $AGEING\_SIM$.

Results are reported inside Table 3.11. The set with higher performances was always the first one, with the age of sim cards under 25 months, where lifts were always over 10, while in the other two subsets the lifts were similar and remained under 6. More details are given for the experiments associated to months December 2016 and January 2017 inside Table 3.12 and Table 3.13.

It must be noted that the percentages of churners retrieved inside the second subset were always higher then those associated to the third subset: in fact, they were similar to those of the first subset. The small values for the lift were caused by the higher baselines of this subset: considering only the sim cards with an age around 30 months was already a step towards the localization of churners, and this limited the performances of the classifier.



Train month	Test month	Lift for sim card with age < 25	Lift for sim card with age in [25, 35]	Lift for sim card with age > 35
2016-10	2016-12	13.20	4.87	5.30
2016-11	2017-01	12.84	5.35	5.32
2016-12	2017-02	12.30	5.31	5.52
2017-01	2017-03	10.25	4.97	4.00
2017-02	2017-04	11.26	4.44	4.73

Table 3.11: Lifts associated to subset with sim cards of different ages (Top 2K).

3.2.3 Splitting according to the segment

Based on some feedbacks from the client, another variable that was studied as potential splitting variable was $SEGMENT$. Sim cards belonging to different segments were treated differently by TELCO2, and even though the segment of belonging was not one of the main features produced by the classifier there was an interest in the performances of classifiers attuned to specific segments.

There were three possible values for $SEGMENT$. The associated subsets had different shapes, as reported in Table 3.14: the first segment was the most numerous one, followed by the second and the third segment, respectively.

The resulting lifts according to this split of the dataset are reported in Table 3.15. The third segment was always the worst performing one, while the other two had similar performances.

Considerations echoing those in previous section must be made: in the third segment the percentages of correct churners retrieved in the Top 2K list were very similar to those of the first segment. The different performance was due to higher values for the baseline. Conversely, in the second segment very small values for the baseline led to high lifts, even though the percentages of correct churners were much smaller.

3.2.4 Inclusion of contacts to assistance number and usage-derived variables

In the analyses described in previous sections two set of features were left out: variables provided by files of type File#4 (see Section 3.1.1) and associated to the contacts with the assistance number of TELCO2, because of an error in the encoding of sim cards for data related to 2017 (as stated in Section 3.1.3), and variables derived from the usage of sim cards which were described in Section 3.1.2.

By the time the analysis described in previous section was in progress, the new set of files of type File#4 was provided. This allowed the inclusion of related variables during the training of predictive models.

The results, which are reported in Table 3.18, show that the performances with and

< 25 months	
Nr samples	423549
Nr churners	8997

25 - 35 months	
Nr samples	78641
Nr churners	4307

> 35 months	
Nr samples	120110
Nr churners	4182

In Top 2K	
Nr churners	549
% churners	27.4%
Baseline	2.1%
Lift	13.20

In Top 2K	
Nr churners	533
% churners	26.6%
Baseline	5.5%
Lift	4.87

in Top 2K	
Nr churners	369
% churners	18.45%
Baseline	3.5%
Lift	5.30

Table 3.12: Statistics on model performances. Data associated to reference month December 2016.

< 25 months	
Nr samples	431540
Nr churners	8786

25 - 35 months	
Nr samples	80214
Nr churners	4059

> 35 months	
Nr samples	115610
Nr churners	3934

In Top 2K	
Nr churners	523
% churners	26.1%
Baseline	2.0%
Lift	12.844

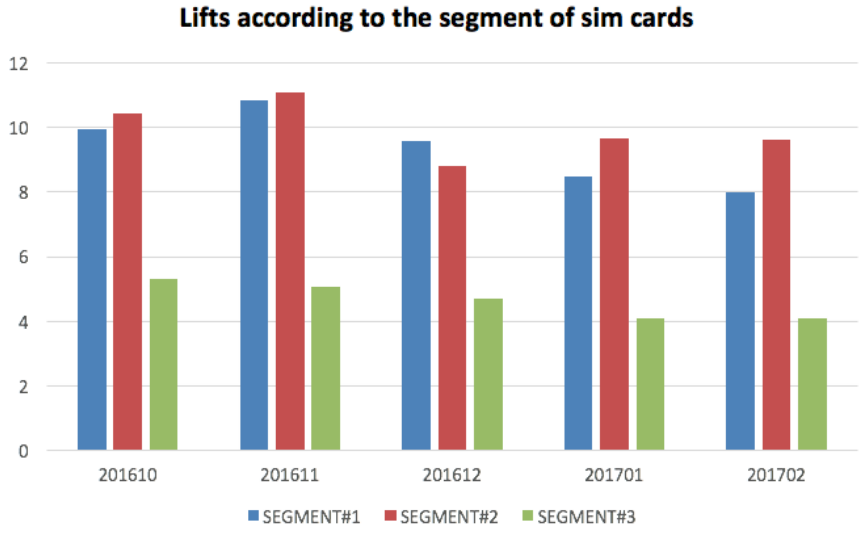
In Top 2K	
Nr churners	542
% churners	27.1%
Baseline	5.1%
Lift	5.355

in Top 2K	
Nr churners	350
% churners	17.5%
Baseline	3.3%
Lift	5.318

Table 3.13: Statistics on model performances. Data associated to reference month January 2017.

Reference month	SEGMENT#1	SEGMENT#2	SEGMENT#3
2016-12	57.6%	25.5%	16.9%
2017-01	59.0%	25.2%	15.8%
2017-02	60.6%	24.8%	14.6%
2017-03	62.1%	24.8%	13.1%
2017-04	63.0%	24.7%	12.3%

Table 3.14: Distribution of sim cards according to their segment.



Train month	Test month	Lift for SEGMENT#1	Lift for SEGMENT#2	Lift for SEGMENT#3
2016-10	2016-12	9.94	10.42	5.31
2016-11	2017-01	10.84	11.11	5.08
2016-12	2017-02	9.57	8.81	4.69
2017-01	2017-03	8.48	9.67	4.00
2017-02	2017-04	8.01	9.62	4.07

Table 3.15: Lifts associated to subset with sim cards of different segments (Top 2K).

without those new features were very similar even across all following months. Indeed, usually the presence of those new features led to a slightly decrease of the lifts.

Alongside this new analysis it was decided to try to include the derived variables from the usage of sim cards. Results are shown in Table 3.19 and are summarized in Figure 3.4. Again, the performances remained unchanged.

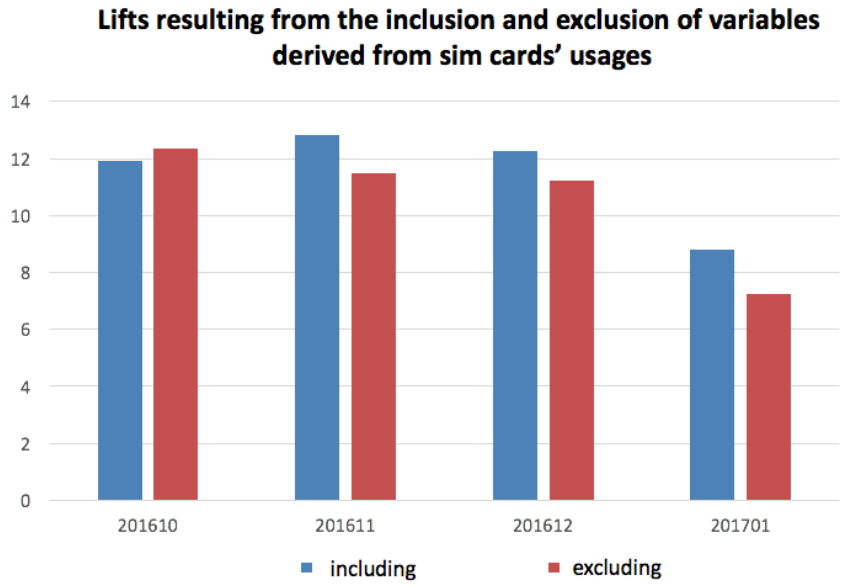


Figure 3.4: Comparison between the number of churners and the lifts associated to Top 2K sim cards retrieved by classifier including and excluding variables derived from sim cards' usages.

SEGMENT#1	
Nr samples	363674
Nr churners	10240

In Top 2K	
Nr churners	560
% churners	28.0%
Baseline	2.8%
Lift	9.941

SEGMENT#2	
Nr samples	160897
Nr churners	1474

In Top 2K	
Nr churners	190
% churners	9.5%
Baseline	0.9%
Lift	10.42

SEGMENT#3	
Nr samples	106826
Nr churners	5772

in Top 2K	
Nr churners	574
% churners	28.7%
Baseline	5.4%
Lift	5.31

Table 3.16: Statistics on model performances. Data associated to reference month December 2016.

SEGMENT#1	
Nr samples	371932
Nr churners	9810

In Top 2K	
Nr churners	572
% churners	28.6%
Baseline	2.6%
Lift	10.84

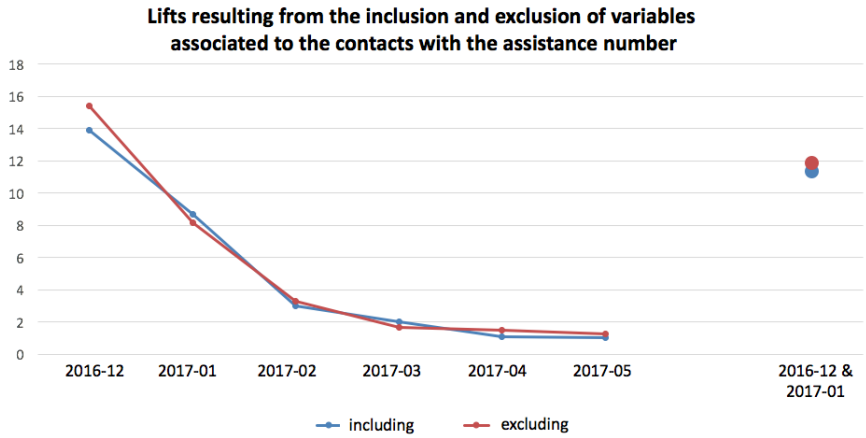
SEGMENT#2	
Nr samples	159614
Nr churners	1472

In Top 2K	
Nr churners	205
% churners	10.2%
Baseline	0.9%
Lift	11.11

SEGMENT#3	
Nr samples	99752
Nr churners	5497

in Top 2K	
Nr churners	560
% churners	28.0%
Baseline	5.5%
Lift	5.08

Table 3.17: Statistics on model performances. Data associated to reference month January 2017.



	With new features		Without new features	
	Nr churners	**Lift**	**Nr churners**	**Lift**
M+1	390	13.89	434	15.4
M+2	238	8.71	225	8.2
M+3	151	2.99	166	3.3
M+4	78	2.04	77	1.7
M+5	56	1.12	64	1.5
M+6	49	1.07	58	1.3
M+1 and M+2	628	11.34	659	11.90

Table 3.18: Comparison between the number of churners and the lifts associated to Top 2K sim cards retrieved by classifier including and excluding variables associated to the contacts with the assistance number. Data associated to test month M equal to December 2016.

	2016-12		**2017-01**		**2017-02**		**2017-03**		**2017-04**	
	Incl.	**Excl.**	**Incl.**	**Excl.**	**Incl.**	**Excl.**	**Incl.**	**Excl.**	**incl.**	**Excl.**
M+1	10.9	15.4	9.3	16.9	10.3	17.9	7.0	11.5	9.1	12.3
M+2	5.4	8.2	5.0	8.4	3.7	5.7	3.4	6.6	4.1	7.4
M+3	3.3	3.3	2.3	2.1	2.6	2.4	2.8	2.7	-	-
M+4	1.5	1.7	1.7	1.5	1.6	1.6	-	-	-	-
M+5	1.0	1.5	1.5	1.3	-	-	-	-	-	-
M+6	1.0	1.3	-	-	-	-	-	-	-	-
M+1 and M+2	12.3	11.9	11.5	12.8	11.2	12.2	7.2	8.8	9.0	9.9

Table 3.19: Comparison between the lifts associated to Top 2K sim cards retrieved by classifier including and excluding variables derived from sim cards' usages. In the first line the reference month M is provided.

The reason behind these results lies in the features the classifiers chose in order to predict churners. In both these new experimental settings, some of the new features were included among the middle-ranked features used by the classifiers. This led to some changes in the predictions, but did not alter significantly the performances.

3.2.5 Splitting according to the expiry date

As state in Section 3.2.1, one of the top-ranked feature was associated to the expiry date of the sim card, variable *EXPIRY_DATE*. Following a discussion with the TELCO, it was agreed to try a splitting based on thresholds -3 and 3 months, as described in Figure 3.5. For this experiment new data were provided, including months till September 2017. Because of this, it was decided to *expand* the training tests from the single month $M-2$ to the three months $M-2$, $M-3$ and $M-4$.

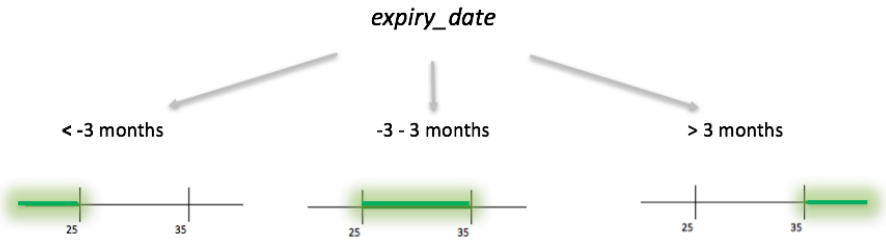


Figure 3.5: Splitting of datasets according to the values of variable $EXPIRE\_DATE$.

Table 3.20 shows the results obtained for test month May 2017. The analyses included also a further splitting on the segment of the sim cards. In Table 3.21 the details referred to the global datasets are provided.

The best lift was the one associated to higher values for the expiry date, a result obtained because of a low baseline compared to the other. This set was also the most numerous one. In the other sets, in spite of high baselines the lifts were quite good, with a percentage of churners in the Top 2K between 30% and 40%.

	SEGMENT#1	**SEGMENT#2**	**SEGMENT#3**	**Global**
EXPIRY_DATE < -3 months	1.35	3.71	1.81	3.66
EXPIRY_DATE in [-3, 3]	1.99	2.95	1.94	2.98
EXPIRY_DATE > 3 months	3.58	3.80	2.21	4.10

Table 3.20: Lifts associated to subset with sim cards of different segments and expiry date (Top 2K). Data associated to reference month May 2017.

< -3 months		-3 - 3 months		> 3 months	
Nr samples	55923	Nr samples	64825	Nr samples	496203
Nr churners	4599	Nr churners	8553	Nr churners	11517
In Top 2K		In Top 2K		in Top 2K	
Nr churners	602	Nr churners	786	Nr churners	186
% churners	30.1%	% churners	39.3%	% churners	9.3%
Baseline	8.2%	Baseline	13.2%	Baseline	2.2%
Lift	3.66	Lift	2.97	Lift	4.09

Table 3.21: Statistics on model performances. Data associated to reference month May 201y.

3.2.6 Studies regarding client aggregation

In all the analyses described in previous sections, the outputs of the predictions were list of sim cards ordered according to the score retrieved by the classifier. The performances discussed so far were always centered around the single sim cards, and did not take into account their owners, i.e. the clients of the TELCO.

A question was asked by TELCO2: what were the performances of the classifiers

in terms of customers? By contacting the customers, or clients, owning the sim cards in the Top K lists retrieved by the predictive algorithms, how many of them were clients with sim cards that were actually going to churn in the next few months?

To answer this questions, a new set of analyses, based on the client aggregation of sim cards, was performed. Starting from the outcome of predictions that followed the procedure described in previous section, the aim was to reconstruct a list of *top churning clients* and associated sim cards, and evaluate the performances of the classifier both in terms of churning sim cards and churning clients. The reference month for these analyses was January 2018, based on new data provided by the client.

The premises for these analyses were the following:

1. a client was assumed to be a *churning client* if it had *at least* a sim card that churned;

2. the churning clients had to be ordered according to a proper *client score* derived from the scores of the sim cards owned by the client;

3. once the clients were ordered according to their client score, performances had to be evaluated on both the actual churning sim cards and churning clients related to the Top K clients of the list.

It must be noted that, since the number of sim cards related to each client was not constant, the number of sim cards associated to the Top K churning clients wasn't constant either.

Client's scores. The first step of the analysis was to define a set of new *client scores* to be associated to the clients of the dataset in order to retrieve a list of *top churning clients*. A total of 8 different scores were constructed, whose definitions are reported inside Def. 3.2.1:

Def. 3.2.1 (Client scores)**.** Let d be a fixed month of reference. Let C be a client owning the sim cards $s_1, s_2, \ldots s_n$ belonging to the dataset associated to month d and containing n sim cards. Let $p(s)$ be the prediction score of a sim card s according to

a given classifier. We define $c_1 = p(s_1), c_2 = p(s_2), \dots c_n = p(s_n)$ the prediction scores of the sim cards owned by C, and $r_1, r_2, \dots r_n$ their rank (where rank equal to N is associated to the sim card with highest prediction score). Let also $P_K(C)$ be the percentage of sim cards owned by C inside the Top K list of the classifier, i.e. the percentage of sim cards whose rank is over $N - K$.

The following *client scores* associated to C are defined:

- *baseline* score: it is equal to the highest prediction scores of the owned sim cards, i.e.

$$\text{baseline}(C) = \max\{c_i, \text{for } i = 1, 2, \dots, n\}. \tag{3.1}$$

- *sum* score: it is equal to the sum of the prediction scores of the owned sim cards, i.e.

$$\text{sum}(C) = \sum_{i=1}^{n} c_i \tag{3.2}$$

- *mean* score: it is equal to the mean of the prediction scores of the owned sim cards, i.e.

$$\text{mean}(C) = \frac{\sum_{i=1}^{n} c_i}{n} \tag{3.3}$$

- *rank* score: it is equal to the mean of the ranks of the owned sim card, i.e.

$$\text{rank}(C) = \frac{\sum_{i=1}^{n} r_i}{n} \tag{3.4}$$

- *sum_rank* score: it is equal to the mean between scores *sum* and *rank*, i.e.

$$\begin{aligned} \text{sum\_rank}(C) &= \frac{1}{2}\sum_{i=1}^{n} c_i + \frac{1}{2}\frac{\sum_{i=1}^{n} r_i}{n} \\ &= \frac{1}{2}\text{sum}(C) + \frac{1}{2}\text{rank}(C) \end{aligned} \tag{3.5}$$

- *mean_rank* score: it is equal to the mean between scores *mean* and *rank*, i.e.

$$\begin{aligned} \text{mean\_rank}(C) &= \frac{1}{2}\frac{\sum_{i=1}^{n} c_i}{n} + \frac{1}{2}\frac{\sum_{i=1}^{n} r_i}{n} \\ &= \frac{1}{2}\text{mean}(C) + \frac{1}{2}\text{rank}(C) \end{aligned} \tag{3.6}$$

- *sum_rank_top* score: it is equal to the mean of two values, the percentage of owned sim cards inside the Top K list of the classifier, and the score *sum_rank*, i.e.

$$\begin{aligned} \text{sum\_rank\_top}(C) &= \frac{1}{2}P_K(C) + \frac{1}{2}\left(\frac{1}{2}\sum_{i=1}^{n} c_i + \frac{1}{2}\frac{\sum_{i=1}^{n} r_i}{n}\right) \\ &= \frac{1}{2}P_K(C) + \frac{1}{2}\text{sum\_rank}(C) \end{aligned} \tag{3.7}$$

- *mean_rank_top* score: it is equal to the mean of two values, the percentage of owned sim cards inside the Top K list of the classifier, and the score *mean_rank*, i.e.

$$\begin{aligned} \text{mean\_rank\_top}(C) &= \frac{1}{2}P_K(C) + \frac{1}{2}\left(\frac{1}{2}\frac{\sum_{i=1}^{n} c_i}{n} + \frac{1}{2}\frac{\sum_{i=1}^{n} r_i}{n}\right) \\ &= \frac{1}{2}P_K(C) + \frac{1}{2}\text{mean\_rank}(C) \end{aligned} \tag{3.8}$$

In the analyses described in this section, only scores *baseline*, *sum*, *mean*, *sum_rank_top* and *mean_rank_top* are considered.

Experiments. Following the reorder of the clients according to the defined client scores, the attention was focused on the sim cards owned by the Top 100, 200, 500 and 1K churning clients. The performances are reported in Table 3.22. In the table, the value inside *Baseline churning clients* refers to the number of churning clients found by considering only the prediction scores of the Top 2K sim cards.

Tables 3.23 - 3.26 list also the number of common clients (not necessary real churners) retrieved by the different client scores. As it can be seen, the scores associated to the *top* score highlighted many common clients, while the other methods were focused on different ones.

Threshold	Baseline churning clients	client score	Nr churning clients	% churning clients	Lift	Nr sim cards	Nr churning sim cards	% churning sim cards
100	0.08	baseline	45	0.45	5.81	1379	111	0.08
		mean	47	0.47	6.06	323	66	0.20
		sum	43	0.43	5.55	6433	487	0.08
		mean_rank_top	48	0.48	6.19	137	67	0.50
		sum_rank_top	42	0.42	5.42	483	68	0.14
200	0.08	baseline	102	0.51	6.58	2156	228	0.11
		mean	90	0.45	5.81	534	128	0.24
		sum	87	0.43	5.61	9872	989	0.10
		mean_rank_top	86	0.43	5.59	265	109	0.41
		sum_rank_top	86	0.43	5.59	583	112	0.19

Threshold	Baseline churning clients	client score	Nr churning clients	% churning clients	Lift	Nr sim cards	Nr churning sim cards	% churning sim cards
500	0.08	baseline	245	0.49	6.32	3954	506	0.13
		mean	210	0.42	5.42	1213	348	0.29
		sum	175	0.35	4.52	16248	1568	0.10
		mean_rank_top	219	0.44	5.65	1636	413	0.25
		sum_rank_top	220	0.44	5.68	3087	376	0.12
1000	0.08	baseline	448	0.45	5.78	6682	942	0.14
		mean	350	0.35	4.52	2259	621	0.27
		sum	314	0.31	4.05	23658	2327	0.10
		mean_rank_top	423	0.42	5.49	3507	783	0.22
		sum_rank_top	433	0.43	5.59	7296	938	0.13

Table 3.22: Performances associated to Top K churning clients computed according to the related client score. The value inside *Baseline churning clients* refers to the number of churning clients found by considering only the prediction scores of the Top 2K sim cards. Data associated to reference test month January 2018.

The following considerations can be made:

- the score *baseline* identified a great number of churning clients, which however had an overall small number of associated sim cards. The precision was usually very low.
- the scores based on the mean, i.e. *mean* and *mean_top_rank*, favored again clients with few owned sim cards. Even though the number of churning sim cards was smaller than the one associated to the *baseline* score, the precision was higher, especially for *mean_top_rank* at low thresholds.
- the score *sum* conversely identified clients with an high number of owned sim cards, most of which however were not churning sim cards, leading to low precisions. Score *sum_top_rank* fixed this behavior, but resulted in a number of churning clients and sim cards under those of the *baseline* score.

In conclusion:

- if the aim is to find the greatest number of churning clients: use *baseline* score;
- if the aim is to find the greatest number of churning sim cards: use *sum* score;
- if the aim is to find the most precise list of churning sim cards: use one of *mean* or *mean_rank_top* scores.

3.3 Conclusions

During this project monthly lists of potential churners were provided to the TELCO at the start of every month. The employed splitting strategy was the one associated to variable *EXPIRY_DATE* because of a direct request of the TELCO, and on some occasions a greater number of splits were computed beside the three described in Section 3.2.5 (again because of the TELCO requests). The analysis of client aggregation described in Section 3.2.6 was also appreciated and used to provide lists of churning clients.

	baseline	sum	sum_rank_top	mean	mean_rank_top
baseline	100	12	24	25	22
sum	12	100	2	1	0
sum_rank_top	24	2	100	76	75
mean	25	1	76	100	97
mean_rank_top	22	0	75	97	100

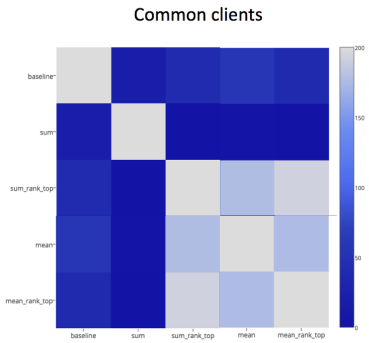


Table 3.23: Common churning clients - Threshold 100.

	baseline	sum	sum_rank_top	mean	mean_rank_top
baseline	200	20	40	57	38
sum	20	200	2	4	0
sum_rank_top	40	2	200	177	193
mean	57	4	177	200	176
mean_rank_top	38	0	193	176	200

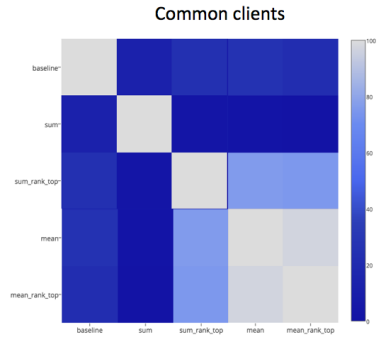


Table 3.24: Common churning clients - Threshold 200.

	baseline	sum	sum_rank_top	mean	mean_rank_top
baseline	500	63	212	150	204
sum	63	500	28	18	31
sum_rank_top	212	28	500	315	448
mean	150	18	315	500	327
mean_rank_top	204	31	448	327	500

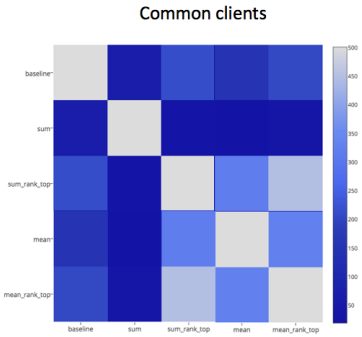


Table 3.25: Common churning clients - Threshold 500.

	baseline	sum	sum_rank_top	mean	mean_rank_top
baseline	1000	180	835	353	673
sum	180	1000	143	58	111
sum_rank_top	835	143	1000	400	786
mean	353	58	400	1000	603
mean_rank_top	673	111	786	603	1000

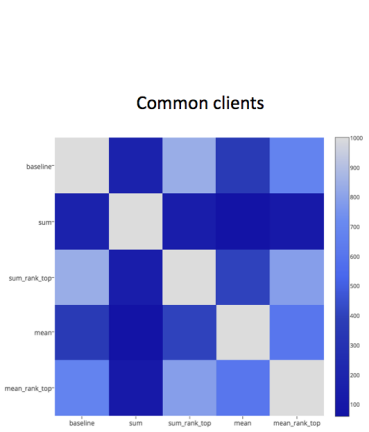


Table 3.26: Common churning clients - Threshold 1000.

Chapter 4

Predictive Maintenance

The problem of *predictive maintenance* is a common problem to all TELCOs. Unlike churn prevention, it does not concern the customers of the company, but it is focused on the management and maintenance system that provides access to the mobile network of the company.

The global infrastructure of a TELCO is based on a great number of *antennas*, placed on strategical locations throughout the country. Each antenna is divided in *nodes* and *cells*, and the latter are connected each with an *apparatus*, given by a *supplier*, that provides the network coverage within its range.

Whenever a disfunction or a system error occurs inside a cell, the corresponding apparatus generates an *alarm signal* that is collected by the cell's operating system. Alarms can last for a small time and then disappear, meaning that the corresponding cell has been able to compensate for the disfunction, or persist until a repair technician arrives and fixes the problem. During this period, the cell and sometimes the entire antenna is *out of order*, providing a disservice for the client of the TELCO and forcing the latter to expensive last-minute repairs.

The aim of predictive maintenance is to predict the rising of an alarm signal in advance in order to perform proper maintenance and avoid disservice and last-minute repairs. It follows that these predictions must be highly *accurate*, since false alarms could lead to a number of unnecessary interventions and result in a loss of money.

The usage of a cell is monitored through a wide number of *basic counters*. They represent a source of synchronous, raw data that report the condition of a given cell at a given time. From these counters, the experts in the field have derived a set of higher level *performance measures*, or *KPIs*, that can sum up the state of each cell, node and antenna at a given time within a window of 15 minutes.

Following the demands of the TELCO, the analyses focused on these higher-leveled counter. As stated by the TELCO, these KPIs have been designed to highlight the causes behind each single alarm event, and give the repair technicians a detailed diagnosis in order to fix the cell. Because of this, KPIs' behavior proved able to provide an *early warning* in the order of minutes that predicts the imminent alarm signal. The problem was that no explicit relations had been found between the measure of one or more KPI and the occurrence of an alarm event within an hour or more. The company desired a longer notice because it could allow the technicians to undertake the necessary prevention steps and avoid the fault of the cell.

Consequently, the formulation of the problem posed by the client was to find out a mathematical relationship between the KPIs and the future alarm signal, in order to predict on each cell the alarm signal from the behavior of the KPIs in the recent past with a high interval of confidence.

This project has been carried out during two different occasions, or 'runs'. Each time, the amount of data given by the company increased, in order to provide more information to the model and retrieve more accurate predictions. To describe the data provided by the TELCO at each run three dimensions are introduced:

- *supplier* (or *vendor*): it relates to the number of different suppliers the cells under analysis are connected to. The TELCO has a total of three different suppliers: V1, V2 and V3;

- *geography*: it relates to the number of cells included in the analysis and their location across Italy;

- *time*: it relates to the number of months the data cover;

Using these dimensions, the data provided at each run can be summarized as in Table 4.1. In the first run data of higher level, coming from cells belonging to the regions of Lazio, Lombardia, Piemonte, Liguria, Toscana and Emilia Romagna, related to a single supplier and covering a single month was analyzed. During the second run the time coverage increased from 1 to 6 months and the cells under exam included also those connected to the other two suppliers. The geographical setting changed, comprising the regions of Marche, Umbria, Liguria, Friuli Venezia-Giulia, Trentino Alto-Adige, Lazio and Sicilia.

	Supplier	**Geography**	**Time**
First run	V1	Liguria, Lombardia, Piemonte, Toscana, Lazio, Emilia Romagna	October 2015
Second run	V1, V2, V3	Marche, Umbria, Lazio, Liguria, Friuli Venezia-Giulia, Trentino Alto-Adige, Sicilia	April - September 2016 (6 months)
Third run	V1	Piemonte, Campania	May, September and October 2016

Table 4.1: Description of the data provided at each run of the project.

In this chapter the analyses performed on both the first and second run are provided. The following sections provide the main steps that were followed, starting from the import and storing of the data (Section 4.1), the preliminary analyses on the occurrences of the alarms (Section 4.2) and the behavior of the KPIs (Section 4.3). The list of the used derived variables is given in Section 4.4. The modeling of the problem as a *classification problem* and the following labeling strategy are described in Section 4.5 and Section 4.6. Finally, Section 4.7 and Section 4.8 describe the experiments and related results of first and second run respectively.

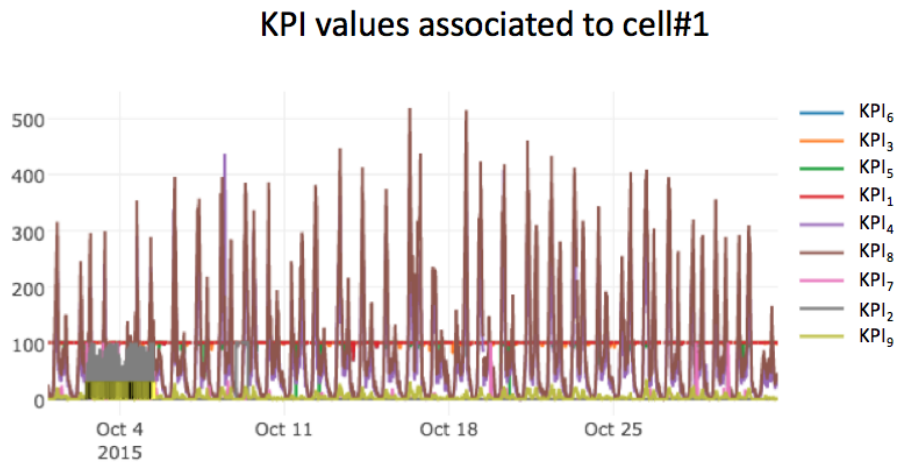


Figure 4.1: Example of KPI values over a single cell.

4.1 Type and storing of the data

The TELCO provided two different types of data to address this problem, stored inside two different datasets.

The first dataset contained nine KPIs, ordered sequences of numerical data that described the status of every cell during a given time range. KPIs are an example of synchronous data, and are collected with a time frequency of 15 minutes, or quarter.

The second dataset contained the collection of all alarm events occurred on the cells of the network system. This data is asynchronous, and it is generated by the apparatus inside each cell and connected to a given supplier, or vendor. For each event a set of related information is provided, of which only the following entries were used:

- **antenna**: string, it identifies the antenna containing the cell;
- **node**: string, it identifies the node inside the antenna containing the cell;
- **first_event**: timestamp, temporal coordinates of the beginning of the alarm;
- **last_event**: timestamp, temporal coordinates of the end of the alarm;

The first step that had to be undertaken before starting the analyses was the merging of these two datasets in order to construct a *time series* for each cell of the network.

The keys *first_event* and *last_event* were used to associate every quarter with the presence or not of an alarm event, thus transforming the second dataset in a collection of synchronous data. Formally:

Def. 4.1.1. Let c be a cell and A the set of alarms registered on cell c described as:

$$A = \{a \ : \ a \text{ alarm on } c\}$$

where each alarm a lasts from $a_{\textit{first\_event}}$ to $a_{\textit{last\_event}}$. Given a fixed quarter q, the variable *alarm* associated to cell c at time q is defined as:

$$alarm((c,q)) = \begin{cases} 1 & \text{if } \exists \ a \text{ alarm on } c : a_{\textit{first\_event}} <= q <= a_{\textit{last\_event}} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

At the end of this process, the **history** of each cell appeared as described in Table 4.2.

CELL	**quarter**	KPI_1	KPI_2	...	KPI_8	KPI_9	alarm
cell#1	**01-10-2015 00:00:00**	1.2	1.1	...	4.6	5.8	0
cell#1	**01-10-2015 00:15:00**	1.5	1.0	...	4.7	5.2	0
cell#1	**01-10-2015 00:30:00**	1.6	1.0	...	4.2	5.7	1
cell#1	**01-10-2015 00:45:00**	1.3	0.9	...	4.5	5.8	1

Table 4.2: Example of data points used to solve the problem of predictive maintenance. Variables related to the antenna and node associated to the cell are also given.

Each item can be univocally identified by a couple (CELL, quarter), making these two variables a valid *multi-index* for the database. Consequently, from now on a data point of the time series will be defined as the collection of variables related to a given couple of cell and quarter.

For each cell a number of additional geographic information was provided, such as the antenna and the node that contained the cell as well as the geographic coordinates of the node. These information were not used directly during the prediction, but provided

a mean to created meaningful subsets of cells and retrieve also *meteorological features* that were added to the analysis (see Section 4.8.7).

All data were stored inside a non-relational MONGO database (see [32]).

4.2 Data Analysis - Alarmed cells

In the first phase of the project, the historical data of all cells under exam were analyzed with standard statistical techniques. This study had two main goals: select a *meaningful subset* of data on which concentrate the following analyses and better *understand* the problem under investigation, in order to retrieve important indicators on the type of models that could be used for the predictions.

The chosen criterion to select a subset of cells on which train a prediction model was the availability of alarm events of the given cell. Indeed, in both runs a great number of the cells did not registered a single alarm event during the time covered by the data. In most cases, these cells were *filtered* from the dataset because they could not provided significant information on the occurrence of the alarm.

Different studies were undertaken on the remaining cells with the aim to find a set of *pathological cells* that could be used to make predictions. In the end, the final criterions used in the first run were different from those used in second run, in part because of the different amount of data that was provided (which enabled, in the second run, a number of more targeted analyses), in part thanks to the accumulated experience brought by the first run.

The following sections provide the alarm-related analyses and the consequently filtering strategies used on the datasets associated to the first and second run.

4.2.1 First Run

As stated in Table 4.1, the data provided during the first run covered only a single month (October 2015) and comprised cells connected to a single vendor out of three.

The original data consisted of 51813 cells located inside the regions of Lazio, Lombardia, Piemonte, Liguria, Toscana and Emilia Romagna. Of these cells, preliminary

analyses showed that only 18431 had registered at least one alarm event during the month of October, so the other 33382 had to be discarded. Table 4.3 describes some characteristics of the remaining cells. Please note that in this table more contiguous alarmed quarters are considered as a unique alarm event.

Indicator	**Values**
Number of alarmed cells	18431
Mean number of alarms per cell	2.9
Standard deviation of the number of alarms per cell	4.2
Cells with less than 2 alarms	50%
Cells with less than 3 alarms	75%
Cells with less than 6 alarms	90%
Maximum number of alarms of a single cell	94

Table 4.3: Some statistical indicators on the population of the first run. More contiguous alarmed quarters are considered as a unique *alarm event*.

The main number of alarms proved to be slightly more than 2, and the 90% of the cells registered a maximum of 6 alarm events. For these cells, the number of alarms was so low that they were considered random events, not able to provide useful information to the predictive model. As a consequence, the analysis was limited to the 1863 cells that registered a minimum number of 6 alarms (*pathological cells*), which included about the 40% of the total number of alarms. The maximum number of distinct alarms registered on a single cell was 94, triggered by 6 different cells but belonging to the same node.

4.2.2 Second Run

After the conclusion of the first run, arrangements were made to continue the project of predictive maintenance and a second run of experiments was defined. Since the short temporal range of the previous data proved to be one of the most limiting factor for the

analysis, this time the TELCO provided KPIs' values for a total of 6 months, ranging from April to September 2016. Moreover, cells associated to other two vendors were included inside the project, thus increasing the total number of available samples.

The new KPIs' values provided by the TELCO were related to cells belonging to seven different regions: Liguria, Marche, Umbria, Friuli Venezia-Giulia, Trentino Alto-Adige, Lazio and Sicilia. Because of the results obtained during the first run (see Section 4.7), cells belonging to these regions were considered from the start as different sources and their data were analyzed separately.

The first performed analysis concerned the alarmed cells inside the datasets. As in the first run, a great number of cells did not registered any alarm during the provided months. The remaining cells provided a number of alarmed quarters as reported inside Table 4.4.

The magnitude of the number of alarmed quarters remained constant for most of the regions, with the exceptions of Liguria and Lazio, where a couple of months registered some peaks. Lazio and Sicilia were the most populated regions, and thus they had a greater number of alarmed quarters with respect to the other regions.

Going to a cell-wise perspective, the mean number of alarms registered by the cells during all the 6 months of data is reported in Table 4.5. Lazio and Sicilia provided the higher values for this indicator, stating that their cells were not only more numerous, but also more prone to malfunctioning. Note that in this table the number of alarm events, not the number of alarmed samples, are analyzed.

After the preliminary analyses on the alarms, the values of the KPIs' time series were studied. The results are shown in detailed the next section. However, two main anomalies led to a consistent filtering of the dataset:

1. The values of KPI_1 proved to be always the duplicated of a second KPI, whose identity changed according to the considered regions but remained constant over the months. The couples of duplicated KPIs are reported in Table 4.6.

   Because of this anomaly, KPI_1 had to be *discarded* from the analysis, and so the learning setting had to be reduced from 9 to 8 KPIs.

Month	Liguria	Trentino Alto-Adige	Friuli Venezia-Giulia	Marche and Umbria	Lazio	Sicilia
April	2240	1048	1092	1128	9513	13700
May	4041	801	1360	1826	12301	19666
June	2952	788	1543	1730	16561	18032
July	2546	984	1580	3256	20222	15838
August	2675	808	1050	1264	10417	15237
September	4227	588	1114	1036	15287	10521

Table 4.4: Monthly alarmed quarters registered for each region.

Region	Mean number of alarms
Liguria	8.61
Marche	6.70
Umbria	10.27
Friuli Venezia-Giulia	5.65
Trentino Alto-Adige	4.28
Lazio	11.0
Sicilia	11.37

Table 4.5: Mean number of alarm events registered during 6 months and divided by region.

Region	Duplicated KPIs	
Liguria	KPI_1	KPI_6
Marche and Umbria	KPI_1	KPI_2
Friuli Venezia-Giulia	KPI_1	KPI_2
Trentino Alto-Adige	KPI_1	KPI_2
Lazio	KPI_1	KPI_6
Sicilia	KPI_1	KPI_2

Table 4.6: Couples of duplicated KPIs. These couples differ from region to region but remain constant over the months.

2. In the single region of Sicilia the values associated to KPI_3 and KPI_5 were almost completely without numerical values (see Section 4.3), and thus not usable in the analysis. This led to a decrease of available features during the learning process associated to the region. In agreement with the TELCO, the dataset of Sicily was *discarded* from the analysis and will not be dealt with in the following part of the chapter.

4.3 Data Analysis - KPIs' behavior

The nine KPIs provided by the client are an example of time series with a time frequency of 15 minutes. The expert of the field, when providing these data, explained that these KPIs had been derived from a collection of basic counters related to the status of the cell with the specific purpose of highlight the causes behind each single alarm event. From these facts it immediately became clear that the behavior of KPIs when the alarm is approaching was of paramount importance and could influence greatly the setting of the prediction environment. An accurate analysis of these time series had to be performed before starting any experiment.

Preliminary studies highlighted a first complication: most of KPIs presented *null* and *non numeric* values. These values could occur for three main reasons:

- for technical reasons, it was not possible to retrieve from the antenna the basic counters used to calculate the KPI value;
- the basic counters were available, but their processing inside the KPI formula rose an exception (e.g. a division by zero);
- the node was out of service and its traffic had been redirected to another node of the network.

It followed that their presence near or during an alarm event was to be expected. Further analysis however showed that these values could be frequently registered also during the correct functioning of the cell, and had to be dealt with (and, in the case of the second run, led to the dropping of an entire region's data, as stated in Section 4.2.2).

During the first run, it was decided in agreement with the TELCO to *discard* these non-numerical values. This choice reduced considerably the amount of data available, and probably undermined the performances of the final predictions. For these reasons, during the second run a different strategy was undertaken, as stated in Section 4.3.1.

Once the problem of non-numerical values was dealt with, the nine different time series were analyzed, with particular care for the points preceding an alarm event. The first step was the analysis of mean and variance of the KPIs series in correspondence, in the absence and in approach of an alarm. This analysis comprises both inter-cell (i.e. relative to all the cells of the dataset) and intra-cell (i.e. relative to a single cell) statistics, and is summed up in Figure 4.2. These results come from the data of the first run, but the conclusions they lead to could be applied also to the data of the second run.

There are three main behaviors for the KPI:

1. the mean values remain constant during all the period of observation and the variance is small. These stable KPI seem not to be able to 'feel' the presence of an alarm, and consequently are presumably little useful for predictive purposes. KPI with this behavior are KPI_1, KPI_3, KPI_5 and KPI_6.
2. the mean values during an alarm are quite different from those registered in absence of an alarm, but the high variance of the data in fact 'conceals' this trend

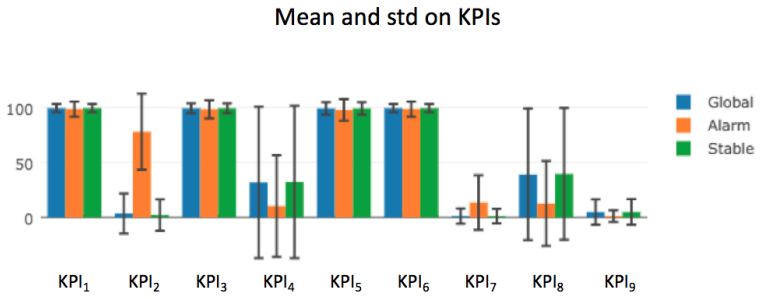


Figure 4.2: Mean and standard deviations on KPIs' time series. Blue bars are related to all the samples, the orange ones to alarmed samples and the green ones to those in absence of an alarm.

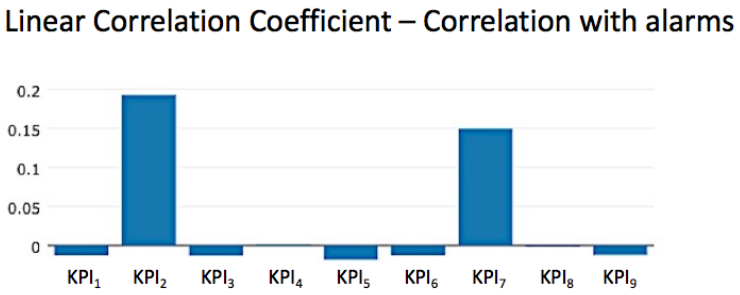


Figure 4.3: Correlation between KPIs' values and the occurrence of an alarm event.

and makes the KPI unstable. KPIs with this behavior are KPI_4, KPI_8 and KPI_9.

3. both the mean values and the variances differ greatly if there is an alarm or not, making this behavior the more useful of the three. KPIs with this behavior are KPI_2 and KPI_7.

This simple but telling analysis showed that KPI_2 and KPI_7 were the time series with higher probability of containing useful information for future prediction. The correlation between the KPIs' values and the presence of an alarm, shown in Figure 4.3, confirmed this idea.

It is important to note that these results, even if relevant in finding the most telling KPIs, correlated events that were *instantaneous*. The correlation between KPIs' values and an alarm that started some quarter in the immediate future were much smaller, and decreases rapidly with the time shift. This analysis confirmed what stated by the expert

of the field, i.e. that the mere numerical values of the KPIs could give an *early warning* in the order of minutes of the imminent alarm, but they could not be used in a contest of *predictive maintenance*. For this goal it was necessary to undertake analysis of the *temporal trend* of the KPIs at the approach of the alarm.

The trend analysis was limited to KPI_2 and KPI_7. Figure 4.4 shows the mean values in the 200 quarters before an alarm of these two KPIs (the number of quarters is reported in the horizontal axis).

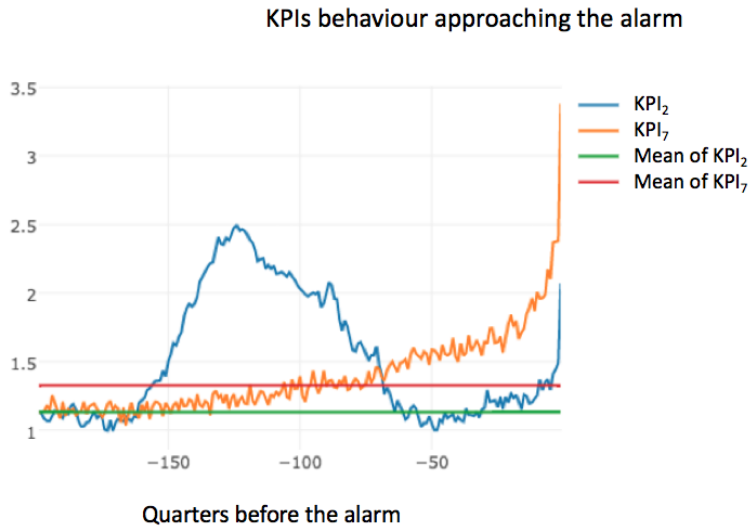


Figure 4.4: Behavior of KPI_2 and KPI_7 approaching an alarm.

KPI_2 started to deviate from its mean values around 14 hours before the occurrence of the alarm, and in the last half an hour values reached up to 12 times the mean. Conversely, KPI_7 had a quicker and more pronounced uphill trend, that made significant only the last three quarters before the event, but between the 40 and 20 hours before the alarm there was another region of higher values that could prove useful for the predictions. Even though the study of the variance trend, shown in Figure 4.5, proved that these variations were mostly concealed by the natural variance of the data, they could always be meaningful for the predictions.

The main conclusions of these preliminary analyses were these:

- some KPIs, i.e. KPI_2 and KPI_7, had a well defined behavior in the approach of

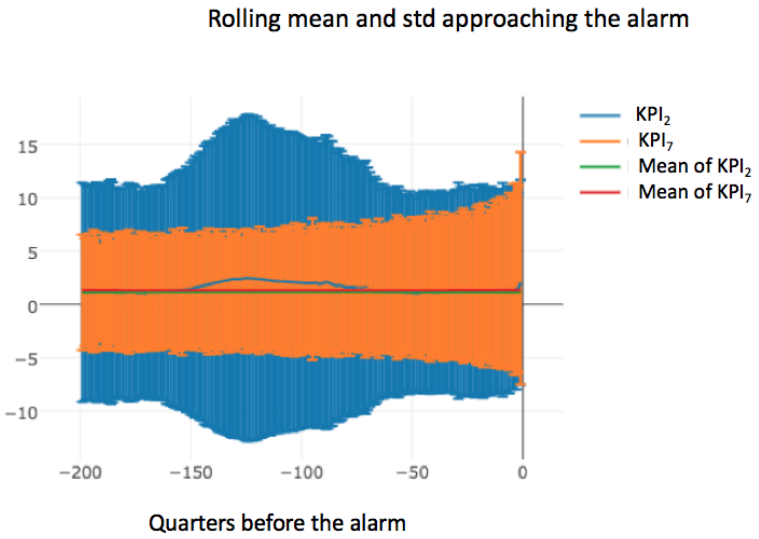


Figure 4.5: Rolling mean and standard deviation of KPI_2 and KPI_7 approaching an alarm.

an alarm that had a potential exploitation for prediction;

- the usage of only the absolute values of the KPIs was not useful in a contest of predictive maintenance. To ensure a good prediction, these values had to be accompanied by more sophisticated data, such as those derived from the frequency analysis of the KPIs.

4.3.1 Non-numerical values in second run

In the dataset associated to the second run, the analysis of non-numerical values inside the KPIs' time series highlighted, besides the anomaly described in Section 4.2.2, different behaviors throughout the regions, as reported in Table **??**. For some KPIs, such as KPI_3 and KPI_7, the great number of non-numerical values seemed to be independent from the region of the cell. For the other KPIs however percentages changed greatly from region to region.

This result was another evidence that the behavior of the cells was not constant inside the global dataset, and confirmed the necessity of dealing with each region separately. It showed also that the decision taken during the first run of dropping samples with non-numerical values from the dataset was too limiting, since it would have led to

KPI	Liguria	Trentino Alto-Adige	Friuli Venezia-Giulia	Marche and Umbria	Lazio	Sicilia
KPI_2	6.42%	5.69%	1.33%	10.25%	1.25%	0.40%
KPI_3	31.56%	31.76%	23.36%	27.17%	20.03%	99.77%
KPI_4	6.42%	5.69%	1.33%	10.25%	1.25%	0.40%
KPI_5	60.56%	67.11%	63.32%	60.96%	49.03%	99.77%
KPI_6	32.66%	7.73%	2.68%	10.80%	20.65%	5.84%
KPI_7	50.15%	50.15%	45.61%	42.71%	36.89%	28.70%
KPI_8	6.42%	6.21%	1.57%	10.35%	1.25%	0.40%
KPI_9	6.42%	5.69%	1.33%	10.25%	1.25%	0.40%

Table 4.7: Percentages of non-numerical entries inside KPIs' values in September 2016. See the anomaly reported for Sicilia concerning KPI_3 and KPI_5.

a discharge of too many samples. For this reason, non-numerical values were processed in the following way:

1. for each KPI and for each type of non-numerical value that appeared inside the KPI time series, a new boolean feature that stated the presence or not of that particular value during a given quarter was constructed;

2. in the KPIs time series, non-numerical values were replaced with a *linear interpolation* of the nearest values of the KPI (the filling option was considered but discarded).

Table 4.8 provides an example of this *boolean categorization*.

4.4 Derived variables

As stated in Section 4.3, the nature of the KPIs was such that the usage of only their numerical values could not provide sufficient information to ensure a good prediction

quarter	KPI
10:45	1
11:00	1
11:15	#ND1
11:30	2
11:45	2
12:00	#ND2

$\rightarrow$

quarter	KPI	KPI_ND1	KPI_ND2
10:45	1	0	0
11:00	1	0	0
11:15	1.5	1	0
11:30	2	0	0
11:45	2	0	0
12:00	2	0	1

Table 4.8: Example of boolean categorization performed during second run on non-numerical values of a given KPI.

of the alarms. For this reason, a set of *filters* was defined, each applied to the time series using one-dimensional convolution algorithms. Three main typologies of filters were used:

- filters belonging to the generic family of *wavelet* (see [9]). For this problem, the proposed filters were: cosine, Ricker (Mexican Hat), Morlet, Bartlett, Exponential (Poisson), Gaussian, Kaiser, Parzen and Triangular. For each filter different window lengths (or radius) were used, and the centering point ranged on the quarters between 6 and 18 hours before the alarm event. It is important to note that parameters defining the window length had to be chosen with great care, taking into account the minimum number of quarters that a point had to be far from an alarm event in order to be considered connected to the alarm n_{min} described in Section 4.6 and the minimum number of quarters to be discarded after an alarm, $\hat{f}$;
- the rolling windows of some basic statistical elements such as mean, median and standard deviation;
- the pure lag of the two most important KPIs, KPI_2 and KPI_7, taken from 3 to 9 hours in the past.

Each of these derived features was codified according to the filter from which it was

generated. For each type of filter there was a proper code:

1. *Wavelet filters.* Example of feature:

   KPI_2_WGAU5s48

   Description: first there is the name of the KPI on which the filter was applied (in this case, KPI_2). The letter W states that the filter belongs to the family of wavelets filters, and it is followed by the acronym of the specific filter (in this case, the Gaussian filter). The number immediately following the acronym refers to the number of points used in the computation, i.e. the window lengths or radius (in this case, 5). Finally, the letter s identifies the number of quarters that precedes the current quarter and that was used as the centre of the convolution filter (in this case 48 quarters, that is 12 hours).

2. *Statistical filters.* Example of variable:

   KPI_5_RSTDw8s12

   Description: the name of the KPI (in this case, KPI_5) is followed by the letter R, that states that the current filter is either a rolling mean, a rolling median or a rolling standard deviation. The actual filter is defined by the acronym immediately following (in this case, a standard deviation). The parameters of the rolling window are then defined: letter w represents the window length of the filter in number of quarters (in this case, 8), while letter s identifies, similar to what was done for the wavelet filters, the actual centre of the window (in this case, located 3 hours before the time of the current point).

3. *Lags.* Example of variable:

   KPI_7_Q36

   Description: letter Q states that this value is the mere lag of the value of the KPI

(in this case, KPI_7). The actual lag time is described in number of quarters (in this case, 9 hours).

The result of these operations was the construction of a great number of features (over one thousand), that had to be selected during the phase of model selection described in next section. In the second run, two other types of features were included in the analysis: the *vendor* related to the cell and the *day of the week* to which the single point belonged to.

Because of the great number of features associated to each data point, usually the training of the model was preceded by a step of *feature selection* that highlighted the most important features to be given to the predictive models. In this contest, the chosen 'shrinkage' operator was the Elastic Net operator (see [30]). Different levels of shrinkage have been considered, forcing the model to deal with more or less variables. Feature selection proved to be a useful help to retrieve better predictions, as reported in Section 4.7.3.

4.5 Modeling of the problem and labeling of the data

In order to solve this learning problem, two different statistical models were taken into accounts.

The first model, schematized in Figure 4.6, was a *direct classification* of the data point. Each point could be labeled according to the presence of an alarm event within a given range of quarters. The advantages of this approach were its simplicity and the high performance of the available algorithms that could be used during the training phase.

The second model, schematized in Figure 4.7, was based on a *regression* analysis on the KPIs' values followed by a *classification* based on threshold parameters over the predicted values. This approach exploited the results of the explorative analysis regarding the behavior of some KPIs near the alarm event and their ability to lead to high accurate classification. The main problem was the necessity to find an auto-regression

	KPI_1	...	KPI_d	Alarm(t+Δ)
$Cell_1$	T_1	T_1	T_1	0
	...	...	...	...
	T_n	T_n	T_n	1
$Cell_m$	T_1	T_1	T_1	0
	...	...	...	...
	T_n	T_n	T_n	1

Figure 4.6: Scheme of the first model (direct classification).

algorithm capable of predict the KPIs' values with due advance.

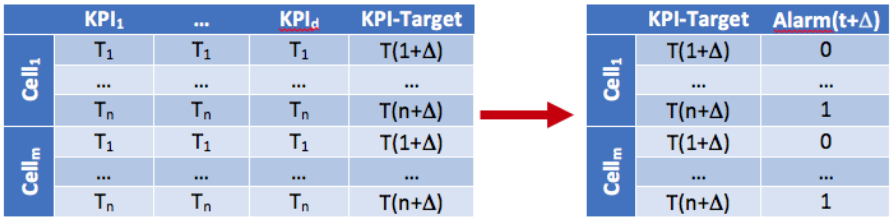


Figure 4.7: Scheme of the second model (regression followed by classification).

After some preliminary tests, it was decided to follow the first model, the direct classification of the data point. This choice led immediately to the necessity to find an appropriate *labelling strategy* for the data.

4.6 Labeling and filtering of the data

The aim of the project was to predict an alarm inside a cell *with a reasonable advance* on its occurrence. In accordance with the chosen model described in previous section, this operation was carried out by identifying those points that *preceded* the alarm event with a reasonable time and classifying them as *positive labels*. Conversely, the data point far away from an event were labeled as *negative*.

Referring to Table 4.2, this labeling of the data was based on the values of vari-

able *alarm*. Two main temporal parameters had to be defined: the *minimum* and the *maximum number of quarters* n_{min} and n_{max} that a point had to be distant from an alarm event in order to be considered connected to the alarm. The only constrain was that n_{min} had to be greater than 0, otherwise the related window included alarmed data points.

The formal definition of the positive labels associated to a given alarm was this:

Def. 4.6.1 (Positive labels associated to an alarm)**.** Let (c, q) be a point identified by a cell c at a given quarter q, and let q_a be the quarter associated to the start of an alarm event a, i.e. $\textit{alarm}(c, q_a) = 1$ and $\textit{alarm}(c, q_{a-1}) = 0$. The *positive labels* associated to the alarm a are defined as those points (c, q) inside the time window

$$T_1(c, a) = \big\{(c, q) \text{ such that } q \in \{q \ : \ q_{a-n_{max}} \leq q \leq q_{a-n_{min}}\}\big\} \tag{4.2}$$

Consequently, the labeling strategy could be thus defined:

$$\text{label}(c, q) = x(c, q) = \begin{cases} 1 & \text{if } \exists \ a \text{ alarm s.t. } (c, q) \in T_1(c, a) \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

It must be noted that with this definition a set of contiguous points with variable *alarm* equal to 1 were considered as a single alarm event and thus led to the creation of a single set of positive labels.

Once the labeling strategy had been fixed, the data had to be *filtered* in order to obtain a coherent and homogeneous dataset. Three different types of data had to be filtered:

1. *Alarmed points*: those points in concomitance of an alarm event. This set was defined as

$$F_1 = \big\{(c, q) \ : \ \textit{alarm}(c, q) = 1\big\} \tag{4.4}$$

2. *Pre-alarm points*: those point that immediately precede an alarm event. This set

was defined as

$$F_2 = \{(c,q) \text{ such that } \exists\ a \text{ alarm } : alarm(c, q_{a-1}) = 0 \wedge alarm(c, q_a) = 1 \wedge q_{a-n_{min}} < q < q_a\} \tag{4.5}$$

3. *Post-alarm points*: those point that immediately follow an alarm event. This set was defined as

$$F_3 = \{(c,q) \text{ such that } \exists\ a \text{ alarm } : alarm(c, q_{a-1}) = 1 \wedge alarm(c, q_a) = 0 \wedge q_a < q \leq q_{a+\hat{f}}\} \tag{4.6}$$

where parameter $\hat{f}$ represented the minimum number of quarters to be discarded after an alarm.

Data points belonging to set F_3 had to be discarded because the behavior of some KPIs continued to be anomalous after the ending of an alarm event, and this proved to affect negatively the following predictions.

In the end, starting from the complete history of a cell the dataset used for the analysis could be schematized as in Figure 4.8.

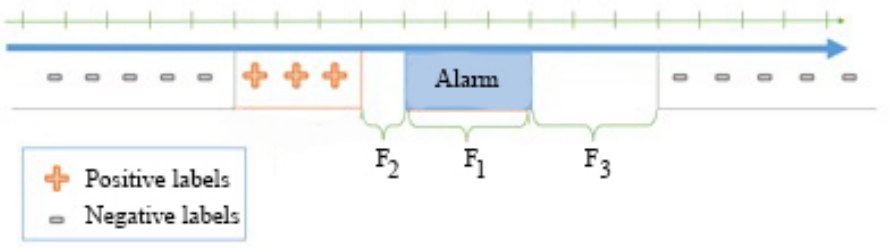


Figure 4.8: Labeling strategy. Samples belonging to F_1, F_2 and F_3 are discarded.

4.7 First run - Results

4.7.1 Model Assessment

The training data were passed to a classification model for the model assessment phase.In this experiment only the *Ridge classifier* was used. Its parameter *alpha* was *tuned* through a procedure of *cross validation* described in Section 1.6.2. Three different splitting strategies for the construction of train, validation and test set were considered:

- *Casual splitting*: the global set of cells was divided into three random disjointed sets with different proportion (at least the 30% of the cells had to belong to the test set).

- *Geographical splitting*: cells were divided according to the geographical proximity, i.e. the belonging to a common node. In this way, all data points referred to a single cell could belong only to one of the three sets of train, validation and test.

- *Temporal splitting*: data points were divided according to temporal criterions, i.e. training and validation data were extracted from the first 20 days of the months, and test data belonged to the remaining 10 days.

The majority of the experiments followed the temporal splitting strategy, despite the short time range of the dataset. The reason behind this choice rested mainly in the advantage of creating a predictive system that could be retrained automatically from week to week. Some experiments were also performed following the geographical splitting, but they turned out to be a minority due to the limited amount of data.

Because of the limited amount of samples to work with and the intrinsically *imbalance* of the dataset, in all test a further operation of rebalancing was performed. The number of negative labeled samples was reduced of a given factor (usually 4) by a random removal.

4.7.2 Preliminary results

To evaluate the performance of the experiments ROC curves were used, as well as tables reporting the sensitivity and precision of the prediction at fixed thresholds for the output of the classifier (see Section 1.7.2 for more details).

In each experiment, the goal was to classify correctly the presence of an alarm at least 3 hours in advance. Consequently, the minimum and maximum number of quarters n_{max} and n_{min} described in Section 4.6 were fixed as these: $n_{min} = 12$ quarters and $n_{max} = 72$ quarters. The minimum number of quarters to be discarded after an alarm $\hat{f}$ was set at 48 quarters.

The first experiment comprised all the cells of the dataset. The resulting ROC curve of the test set is reported in Figure 4.9.

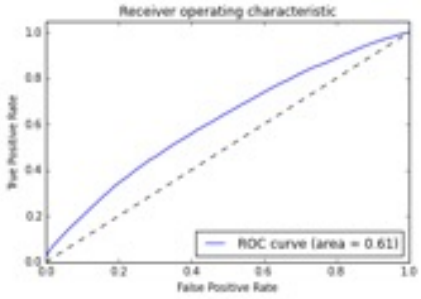


Figure 4.9: Test ROC of dataset including all cells.

As it can be seen, the ROC curve is flattened on the bisector of the plane, and its behavior on the extremes showed that the precision of the corresponding prediction was too low to be considered acceptable for the TELCO's demands.

4.7.3 Geographical restriction

Because of these low performances, a new set of experiments was designed and performed. The idea was to restrict the dataset of cells from all around Italy to progressively smaller regions centered around a fixed point, such as the centre of a big city. Six cities were considered, one for each region (Rome for Lazio, Milan for Lombardia, Turin for Piemonte, Genoa for Liguria, Florence for Toscana and Bologna for Emilia

Romagna) as well as three different radii (10, 20 and 30 km). Because of the limited amount of data, some experimental settings proved to be too small to retrieve a meaningful analysis.

Radius (km)	**City**					
	Rome	**Milan**	**Turin**	**Genoa**	**Florence**	**Bologna**
10	201	127	74	50	99	15
20	297	234	140	184	201	146
30	385	383	289	411	298	285

Table 4.9: Number of alarmed cells located near the centre of different towns.

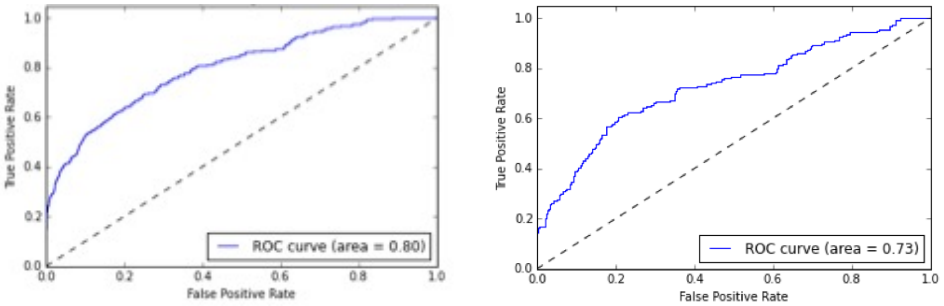
Rome proved to be the city with the best performances. Figure 4.10 shows the ROC curve associated to the train and test set of an experiment containing cells within a radius of 10 km from the center of the city of Rome, which proved to be the one with best performances. A feature selection was performed before the training of the model.

As it can be seen, the behavior of the ROC curves is better than the one of the curve associated to the global setting, especially in the classification of positive samples. This is confirmed by the results in Table 4.10, where at different thresholds the precision is over 0.52 out of a baseline of 0.21. As for the negative samples, on a restricted number of cells the classification was highly accurate (over 0.9 on the 13% of the total population). These performances drop significantly as the distance from the centre of the city increases (see Figure 4.11).

The impact of the feature selection performed before the training of the model was evaluated comparing the results of two experiments settled around Rome, the first being carried out without the aforementioned selection. The results are reported in Figure 4.12.

In the second experiment, 409 features out of the original 1200 were selected by the process of feature selection. Table 4.11 reports the ordered list of the 35 most important features extracted with the Elastic Net operator.

PM - Results of first run: Rome



(a) Rome - Train set (b) Rome - Test set

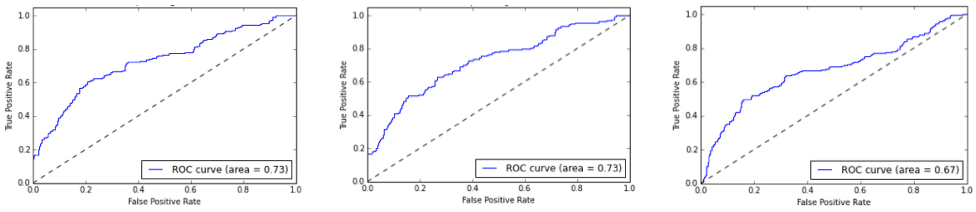
Figure 4.10: Train and test ROC of best performing experiment.

Precision baseline: 0.21.

	Threshold	% of Population	Precision
Positive	0.8	10%	0.82
	0.5	13%	0.70
	0.2	25%	0.53
Negative	-0.8	5%	1.0
	-0.5	40%	0.92
	-0.2	70%	0.89

Table 4.10: Performances obtained at different thresholds for the output of the classifier.

Behavior at the increasing of the radius.



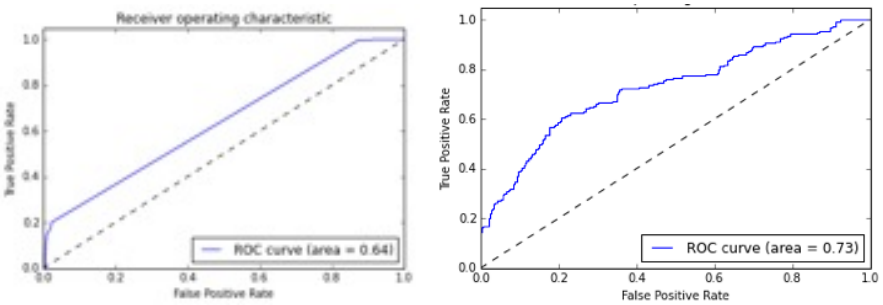
(a) Rome - 10 km (b) Rome - 20 km (c) Rome - 30 km

Figure 4.11: Test ROC curves at the increasing of the radius.

Rank	Feature
1	KPI_2_WSYM213s48
2	KPI_2_RSTDw8s48
3	KPI_2_WCOIF111s48
4	KPI_2_WCOS5s48
5	KPI_2_WHAAR6s48
6	KPI_2_WTRI5s48
7	KPI_2_RMEDw8s48
8	KPI_2_WKAI5s48
9	KPI_2_WBAR9s48
10	KPI_2_WEXP9s48
11	KPI_2_WGAU5s48
12	KPI_2_WMOR9s48
13	KPI_2_WBAR5s48
14	KPI_2_Q35
15	KPI_2_WGAU13s48
16	KPI_2_WPAR5s48
17	KPI_2_WKAI13s48
18	KPI_2_WTRI13s48

Rank	Feature
19	KPI_2_WCOS13s48
20	KPI_2_WDB311s48
21	KPI_2_WEXP5s48
22	KPI_2_WEXP13s48
23	KPI_2_WRICK13s48
24	KPI_2_WBAR13s48
25	KPI_2_WPAR13s48
26	KPI_2_Q20
27	KPI_7_WBAR9s48
28	KPI_7_WHAAR6s48
29	KPI_2_WMOR13s48
30	KPI_2_Q22
31	KPI_7_WCOIF111s48
32	KPI_7_WSYM213s48
33	KPI_2_Q33
34	KPI_2_Q21
35	KPI_7_RSTDw8s48

Table 4.11: Top 35 features extracted with feature selection during an experiment settled around Rome. To interpret the name of the feature see Section 4.4.



(a) ROC obtained considering all 1200 features

(b) ROC obtained considering only the 409 features extracted by the feature selection procedure

Figure 4.12: Evaluation of feature selection in the setting of Rome.

The following observations can be made:

1. the two KPIs with the most important contribute were the ones identified in the preliminary analysis, i.e. KPI_2 and KPI_7;

2. the majority of features derives from a convolution filter that belongs to harmonic analysis;

3. a great number of features were computed starting from 12 hours before the actual data point (features with *s48*) and an amplitude (described by the letter w) greater then 5 quarters;

4. the mere numerical values of the KPIs have an impact on the model, as stated by the presence of both the median and standard deviation features (*RMED* and *RSTD* respectively).

In most of the other experiments, the usage of feature selection proved to be positive as it led to better ROC curves and more precise predictions. It must be noted that KPI_2 and KPI_7 were selected by the elastic net operator in the majority of cases (thus confirming their importance in the problem), but sometimes other KPIs proved to be important too. For example, in the city of Milan, where feature selection had a great

impact on performances as shown in Figure 4.13, features derived from KPI_5 turned out to be as meaningful as those derived from the other two KPIs.

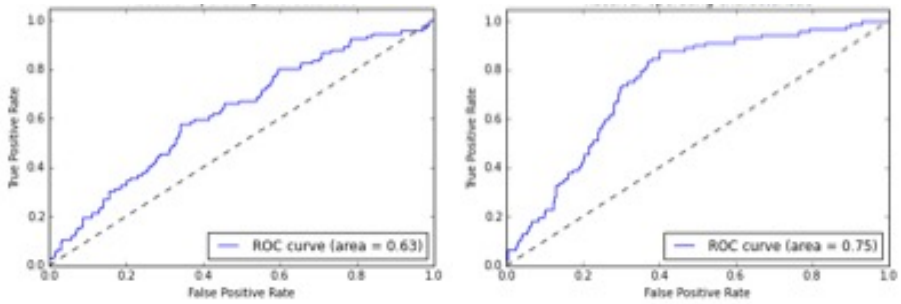


Figure 4.13: ROC curves obtained on the setting of Milan considering 917 features (left) and 704 features (right) extracted by the feature selection procedure.

In the end, the following conclusions were drawn:

1. predictions were most performant when carried out 3 hours before the alarm event. Further anticipations lead to a decreasing of accuracy, and at 6/9 hours the prediction seemed to be almost random;

2. analyzing all cells in a single run did not lead to good predictions. It was best to aggregate cells which were located in a limited geographical area, such as near the centre of a town, and then train separate models. Smaller areas were generally associated to better performances.

3. feature selection proved to be an important step to increase the quality of predictions, and had to be performed for each experimental setting.

4.8 Second Run - Results

4.8.1 Geographical selection strategies

The first run of the project highlighted the need to aggregate the cells according to their location, in order to find clusters of cells that were affected by common events, and thus could be modeled.

Following this demand, three different strategies were defined:

1. *Regional selection*. Each cluster contained cells belonging to the same region (e.g. Lazio);

2. *Centre-Radius selection*. It was a generalization of the selection performed during the first run. Each cluster contained cells that were within X km from a fixed geographical point, located by its latitude and longitude coordinates.

3. *Grid selection*. This selection was a union of the other two strategies. Each region was enclosed inside a rectangle whose vertex were defined by the minimum and maximum latitude and longitude of the points inside the region. Over this rectangle a *grid* was then set, and over each point of this grid the Centre-Radius selection of cells was applied, as well as the constraint that the cells belonged to the given region. In order to ensure the total coverage of the region, the radius was fixed as the step of the grid. Figure 4.23 shows an example of this procedure, highlighting three different points were the analysis could be carried out separately.

Figure 4.14: Example of grid selection of the cells. For each point of the grid, the Centre-Radius selection is performed. The radius equals the step of the grid.

This strategy was developed to manage the different densities of cells inside the

greater cities and belonging to rural areas. In particular:

- around cities the greater number of cells led to the usage of a grid with small step (*fine grid*);
- on the contrary, in rural areas the density of cells was smaller, so the grid was set with a bigger step (*coarse grid*).

4.8.2 Model Assessment

The operation of model assessment in this second run was more convoluted. The Ridge classifier was abandoned, and in its stand two different families of classifier were considered: the *random forests* and *gradient boosting* models. In each case, a set of parameters was tuned in order to find the most performing classifier, being:

- number of estimators, minimum samples split and maximum depth for random forest models,
- number of estimators, minimum samples split and maximum depth for gradient boosting models.

The tuning of the parameters for each model was performed through a cross validation approach, specifically using a 3-fold cross validation (see Section 1.6.2). After some preliminary analyses, the learning algorithm was always chosen as the gradient boosting classifier, but cross validation was performed at the beginning of each experiment.

Thanks to the wider temporal range covered by the data, the *temporal splitting* strategy to divide data inside train and test set as described in Section 4.7.1 could be applied on each dataset without a pathological decrease of the population. Train, validation and test set were created according to the following rule:

- Training and Validation set: April - June
- Test set: July - September

To ensure an *homogeneity* between the two sets of cells, a further *filtering step* was undertaken: every experimental set comprised only those cells that registered a similar number of alarms during the months of training and testing. Cells were thus divided inside different *alarm ranges* in the form of [A_{min}, A_{max}], which included all cells that registered from A_{min} to A_{max} alarms during both the training and testing months. Some examples of alarm ranges are [1,5], [2,5] and [3-10].

4.8.3 Centre-Radius experiments

The first set of experiments was done selecting the cells for each dataset according to the Centre-Radius selection strategy, as done in the first run of the project. For each region a single *centre* was fixed over the biggest town, and a set of different *radii*, from 10 to 120 km, was considered. Only the alarm range [5,10] was taken into account, and in the labeling strategy the parameter n_{min} was fixed at 6 hours. Figure 4.15 and Table 4.13 report some results associated to the best performing predictions.

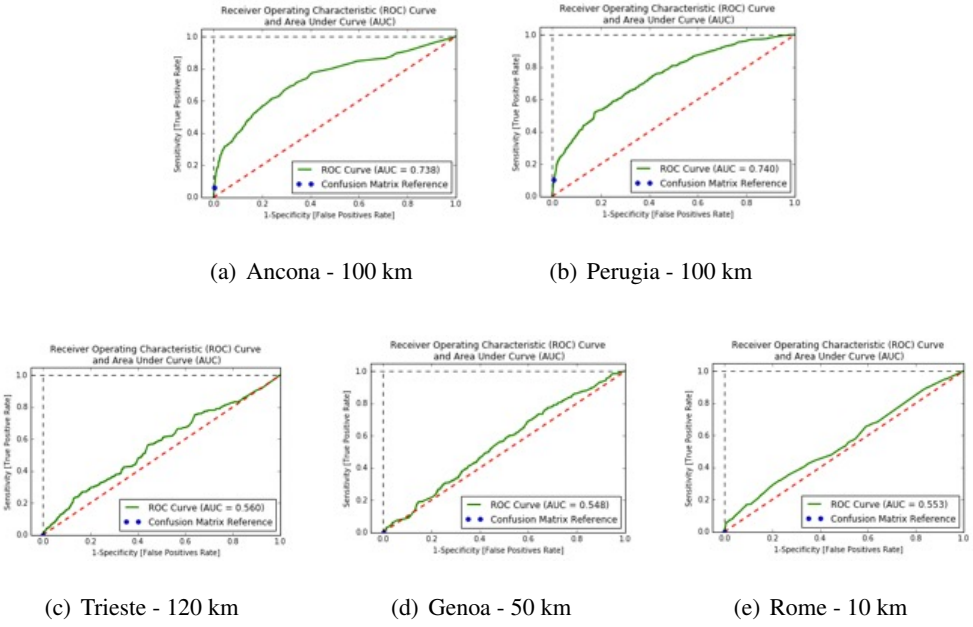
Both the cities of Ancona (Marche) and Perugia (Umbria) produced good ROC curves; with a radius of 100 km from the centre of the town, the predictions reached a precision of 14% and 11% respectively, starting from a baseline of 0,7%. For the other towns however (Genoa for Liguria, Trieste for Friuli Venezia-Giulia and Rome for Lazio), the precision was under the 5%, making the results completely unacceptable.

4.8.4 Regional selection

After the preliminary results, the attention was focused of the totality of the regions. Three different alarm regions were considered, [1,5], [2,5] and [3,10], and the parameter n_{min} associated to the labeling strategy was kept fixed at 6 hours.

The top value for precision was reached by Friuli Venezia-Giulia (29%), followed by Marche (16%) in alarm range [1,5]. It must be noted however that in Friuli Venezia-Giulia the number of predicted alarms was very small (24 positive predictions, divided into 17 FP and 7 TP). The classifiers related to Trentino Alto-Adige, Liguria and Lazio were unable to perform better than the random classification.

Centre-Radius experiments



(a) Ancona - 100 km (b) Perugia - 100 km

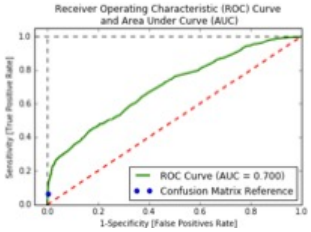
(c) Trieste - 120 km (d) Genoa - 50 km (e) Rome - 10 km

Figure 4.15: Test ROC curves associated to the predictions on data retrieved by the centre-radius selection strategy (best results per town).

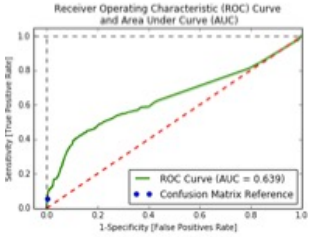
Town	Radius	Predicted alarms (TP)	Total alarms	Total samples	Precision
Ancona	100 km	245	4032	553680	0.137
Perugia	100 km	353	3480	447254	0.108
Trieste	120 km	2	7464	634179	0.027
Genoa	50 km	0	1968	136394	0.000
Rome	10 km	3	7464	634179	0.038

Table 4.12: Performances associated to the predictions reported if Figure 4.15.

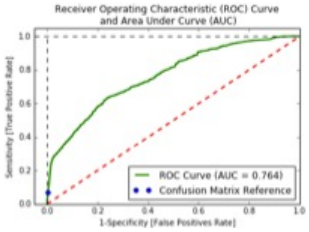
Region experiments



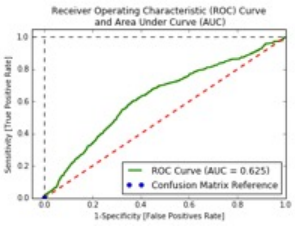
(a) Marche - alarm range 1-5



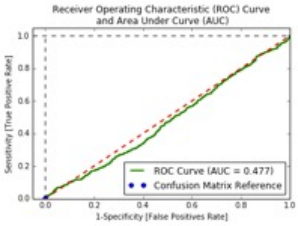
(b) Marche - alarm range 3-10



(c) Umbria - alarm range 3-10



(d) Friuli Venezia-Giulia - alarm range 2-5



(e) Trentino Alto-Adige - alarm range 2-5

Figure 4.16: Test ROC curves associated to the predictions on data retrieved by the region selection strategy.

Region	Alarm range	Predicted alarms (TP)	Total alarms	Total samples	Precision
Marche	[1,5]	343	5112	1301041	0.158
Marche	[3,10]	190	3432	609662	0.132
Umbria	[3,10]	262	3768	543523	0.111
Friuli V.G.	[2,5]	7	2687	616887	0.292
Trentino A.A	[2,5]	5	1296	348775	0.024

Table 4.13: Performances associated to the predictions reported if Figure 4.16.

To better understand the behavior of the classifiers a deeper analysis on the single cells was performed. Figure 4.17(a) shows the labeling and scoring of the data points referred to a particular cell belonging to the region of Marche, alarm range [1,5]. Each samples represent a couple (CELL, quarter), with quarters that run on the x-axis and comprise all the days belonging to the test set (July-September). Blue bars represent the label of the sample, which can be 1 or 0, and orange bars represent the score that the classifier assigned to the sample.

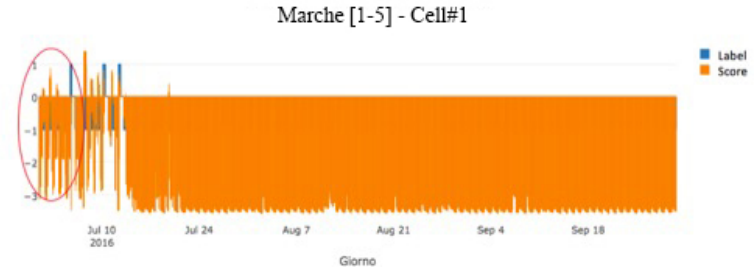
As it can be seen, the classifier was not able to correctly predict the three sets of positive labels (related to as many alarm events), but the behavior of the score is clearly uneven if compared with the remaining days of the dataset, suggesting that the classifier understood that some interferences with the cell were undergoing. The results reported in Figure 4.17(b) and Figure 4.17(c) belong to the same experimental setting, but are related to two different cells. The first one shows an example of good prediction, where the alarm was correctly predicted, while in the second one the score seems not to correlated with the two different alarms. Other examples can be found in Figure 4.18.

4.8.5 Grid selection

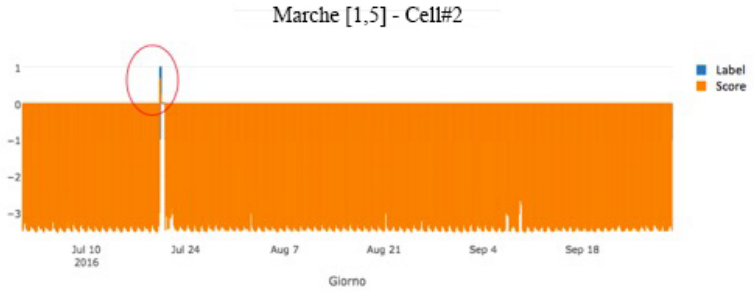
After the conclusion of the second set of experiments, and having received some feedbacks from the TELCO, the analysis focalized on the points inside each region located through the grid selection strategy.

As stated in Section 4.8.1, two different grids were used, a *fine* one to analyze cells around the bigger towns and a *coarse* one for the rural areas. To include a sufficient amount of cells, the wider alarm range [0,30] was considered. A difference from the previous experiment is the absence of a cross validation operation to tune the parameter of the model: the classifier was fixed as the gradient boosting model with parameters equal to those that were selected the most from the previous experiments, i.e. number of estimators = 30, minimum samples split = 5000 and maximum depth = 3.

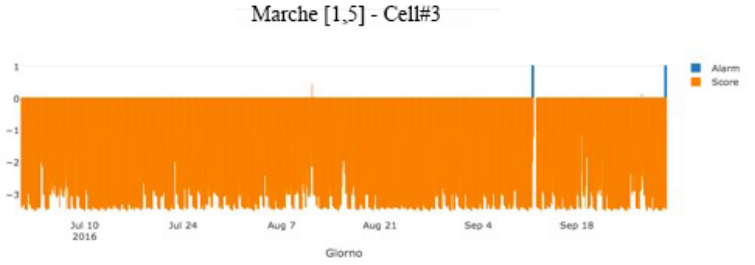
For the regions of Marche, Umbria, Friuli Venezia-Giulia and Trentino Alto-Adige the fine and coarse grid had a step of 30 km and 50 km respectively. In Liguria only



(a)

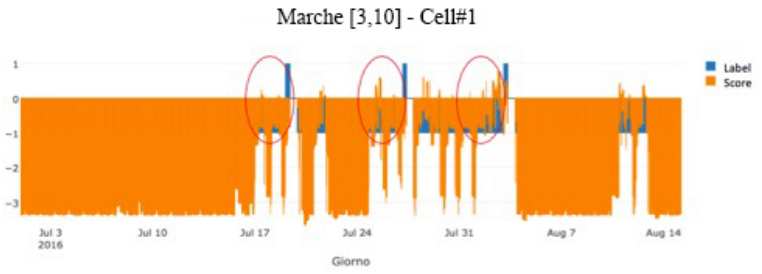


(b)

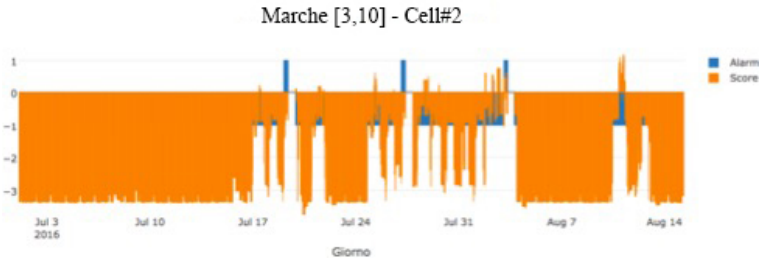


(c)

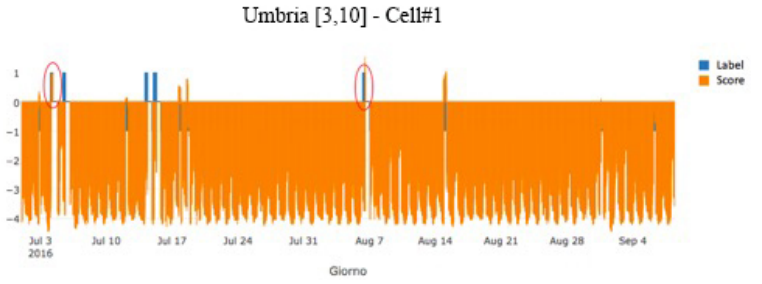
Figure 4.17: Behavior of scores and labels of three different cells belonging to Marche, alarm range [1,5].



(a)



(b)



(c)

Figure 4.18: Other examples of scores' behavior on some cells. Note that cells in subfigures 4.18(a) and 4.18(b) belong to the same node, and their behaviors, as well as their scores, are comparable.

one grid of 15 km was used due to the shape of the region, and in Lazio the presence of Rome enable the usage of a fine grid with step 3 km (the coarse grid was fixed with step 40 km).

For each region, the area under the ROC curve (AUC) obtained for every point of the associated grid was used to construct images such as those shown in Figure 4.19. The color scale goes from red to green as the value of AUC increases, thus highlighting the most performing experiments. The 'holes' refer to grid point where the associated cells were too few to perform an experiment.

Table 4.14 reports the best recall values registered for each region. As it can be seen, the best performance were always associated to points belonging to the fine grid. The most performant prediction was found in Liguria, where a point was associated to a recall of almost 13%. Marche also increased its performances, reaching a 7.4% of recall. Friuli Venezia-Giulia and Lazio performed badly (under 1%), and Umbria registered a significant drop from the results obtained in the previous set of experiments. For Trentino Alto-Adige no point was found with a recall over zero, so no data are reported.

Region	**Parameters of best prediction**		
	Grid step	**AUC**	**Recall/Sensitivity**
Marche	30 km	0.66	0.074
Umbria	30 km	0.689	0.026
Friuli V.G.	30 km	0.616	0.006
Trentino A.A	-	-	-
Liguria	15 km	0.675	0.129
Lazio	3 km	0.612	0.009

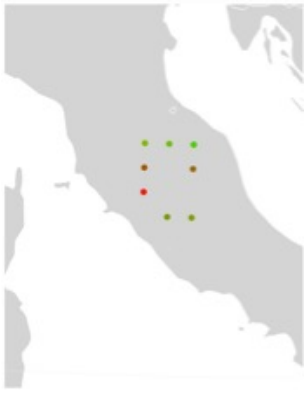
Table 4.14: Best performance obtained on each region. The performance is measured as the recall values.

Grid experiments

(a) Marche - 30 km

(b) Marche - 50 km

(c) Umbria - 30 km



(d) Umbria - 50 km

(e) Friuli Venezia-Giulia - 30 km

(f) Friuli Venezia-Giulia - 50 km

Figure 4.19: Area under the ROC curve (AUC) for each point of the grid. Regions: Marche, Umbria, Friuli Venezia-Giulia.

Grid experiments

(a) Trentino Alto-Adige - 30 km (b) Trentino Alto-Adige - 50 km

(c) Liguria - 15 km

(d) Lazio - 3 km (e) Lazio - 40 km

Figure 4.20: Area under the ROC curve (AUC) for each point of the grid. Regions: Trentino Alto-Adige, Liguria, Lazio.

Grid experiments - Marche

Sample #1

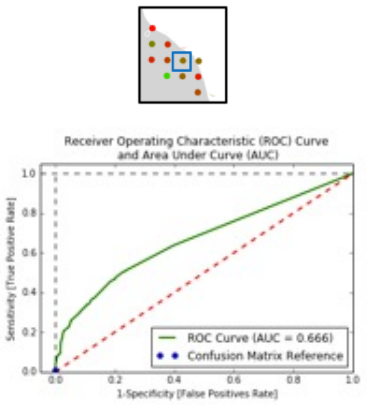


Figure 4.21: ROC and location of point inside the grid

TN = 754480 (99.56)	FP = 108 (0.01)
FN = 3238 (0.43)	TP = 26 (0.00)

Population: 757852	**Positive: 3264**
Accuracy	0.996
Precision	0.194
Recall/Sensitivity	0.008
Specificity	1.000

Table 4.15: Confusion matrix and performances.

Sample #2

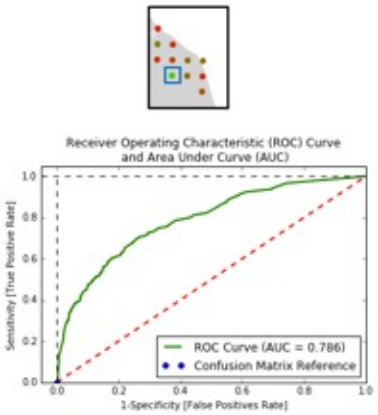


Figure 4.22: ROC and location of point inside the grid

TN = 300935 (99.32)	FP = 106 (0.3)
FN = 1968 (0.47)	TP = 0 (0.0)

Population: 303009	**Positive: 1968**
Accuracy	0.993
Precision	0.000
Recall/Sensitivity	0.000
Specificity	1.000

Table 4.16: Confusion matrix and performances.

4.8.6 Other results - aggregation and further prediction

In this section two other types of experiments are briefly reported, both performed during the first set of experiments (Section 4.8.3) but then left out because of the poor quality of predictions.

In the first type, the original values of KPIs and alarms were *aggregated* at given time intervals T before undergoing the process of labeling, filtering and construction of derived features. Examples of aggregations are the hourly aggregations (T=1h), daily quarters aggregations (T = 6h), and half daily aggregations (T=12h).

Figure 4.23: Example of temporal aggregation. The upper time line shows the KPIs' units (quarters). Taking an interval T, data are aggregated and a new smaller set of data points is obtained (lower line).

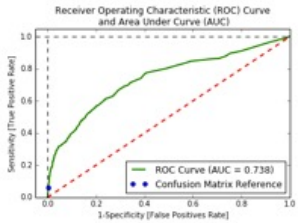
The daily quarters and half daily aggregation (T = 6h, 12h) proved to be unfeasible in most of the datasets. The only aggregation that was studied in deep was the hourly aggregation (T=1h). The setting is the same as in Section 4.8.3. Results and comparisons are reported in Figure 4.24.

In most of cases, the hourly aggregation proved to be counterproductive or irrelevant. Only in two occasions there was an increasing of the performances, i.e. for the cities of Rome and Trieste, but the results were still unusable.

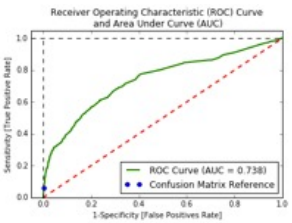
In the second type of experiments different values for the parameter n_{min}, i.e. the minimum number of quarters that a point had to be distant from an alarm event in order to be considered connected to the alarm, were considered. The goal was to see how the performances behaved when trying to anticipate the alarms. The experiments were performed in the cities of Ancona, Perugia and Genoa, were samples where selected following the centre-radius strategy (radius of 50 km) and belonging to the alarm range [3,10]. Results are reported in Figure 4.25 and Tables 4.17 4.18 .

In most cases, the values of the AUC decreased with the increasing of the parameter.

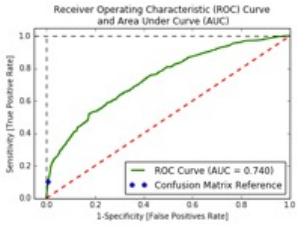
Hourly aggregation



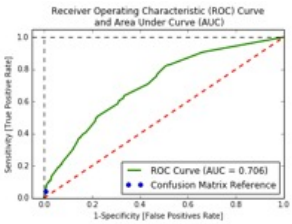
(a) Ancona, 100 km - T = 15m



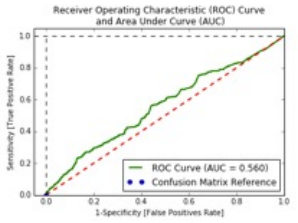
(b) Ancona, 100 km - T = 1h



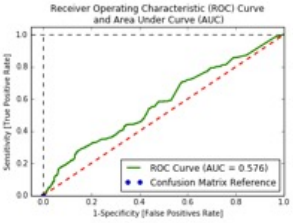
(c) Perugia, 100 km - T = 15m



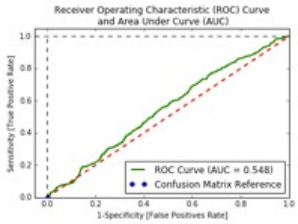
(d) Perugia, 100 km - T = 1h



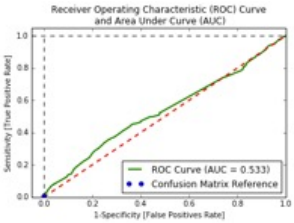
(e) Trieste, 120 km - T = 15m



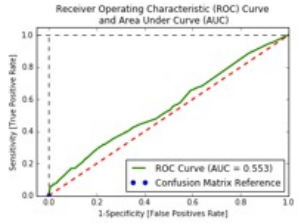
(f) Trieste, 120 km - T = 1h



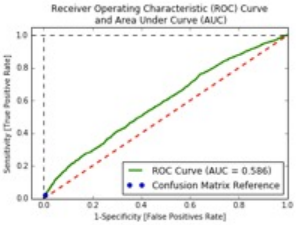
(g) Genoa, 50 km - T = 15m



(h) Genoa, 50 km - T = 1h



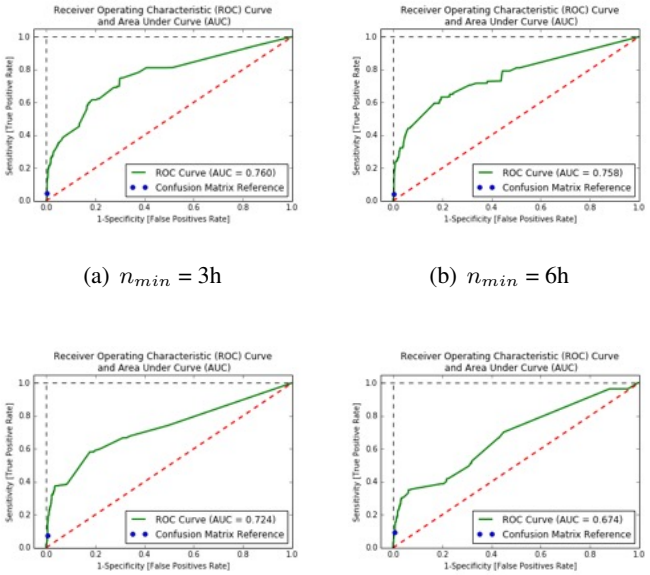
(i) Rome, 10 km - T = 15m



(j) Rome, 10 km - T = 1h

Figure 4.24: Comparison between predictions made on original samples (left) and aggregated samples (right).

Shift of parameter n_{min}



(a) n_{min} = 3h (b) n_{min} = 6h

(c) n_{min} = 12h (d) n_{min} = 24h

Figure 4.25: ROC curves at different values of parameter n_{min} for the town of Ancona.

TN = 390401 (99.44)	FP = 1198 (0.31)
FN = 940 (0.24)	TP = 44 (0.01)

TN = 388922 (95.62)	FP = 1173 (0.30)
FN = 1882 (0.48)	TP = 86 (0.02)

TN = 385118 (98.46)	FP = 2157 (0.55)
FN = 3582 (0.92)	TP = 291 (0.07)

TN = 380204 (97.62)	FP = 1851 (0.48)
FN = 6719 (1.73)	TP = 690 (0.18)

Table 4.17: Confusion matrix associated to experiments shown in Figure 4.25.

Town	n_{min} **= 3h**	n_{min} **= 6h**	n_{min} **= 12 h**	n_{min} **= 24h**
Ancona	0.035 (0.760)	0.068 (0.758)	0.119 (0.724)	0.272 (0.674)
Perugia	0.113 (0.851)	0.084 (0.823)	0.06 (0.743)	0.26 (0.709)
Genoa	0 (0.634)	0.38 (0.650)	0.303 (0.598)	1.0 (0.588)

Table 4.18: Performance of precision (AUC) associated to different values of n_{min}.

Figure 4.25 shows that also the shape of the ROC worsened, especially in right upper corner (negative labels). As for the precision, there seemed to be no apparent rule that told what anticipation led to the best performances. For these reasons, it was chosen to fix n_{min} at 6 hours, a compromise between a numerous dataset, good ROC behavior and mean precision performances.

4.8.7 Weather conditions

A further analysis was carried out to investigate a possible correlation of alarms with meteorological events. The rational behind this idea was that some intense weather phenomena, such as storms, cold temperatures or strong winds, could damage the cells located on given antennas and as a consequence lead to alarm events.

Weather informations were acquired thanks to the DarkSky.net service and then integrated inside the main dataset, using the node identifier and the quarter as indices. The node identifiers was used because inside the database the longitude and latitude coordinates are referred nodes, and then extended to all the cells belonging to the node. A list of all new features is given in Table 4.19.

Feature	**Type**
Type of precipitation	Categorical
Temperature	Numerical
Perceived Temperature	Numerical
Humidity	Numerical
Wind's speed	Numerical
Wind's direction	Numerical
Cloud's coverage	Numerical
Pressure	Numerical

Table 4.19: Meteorological variables extracted from DarkSky.net service.

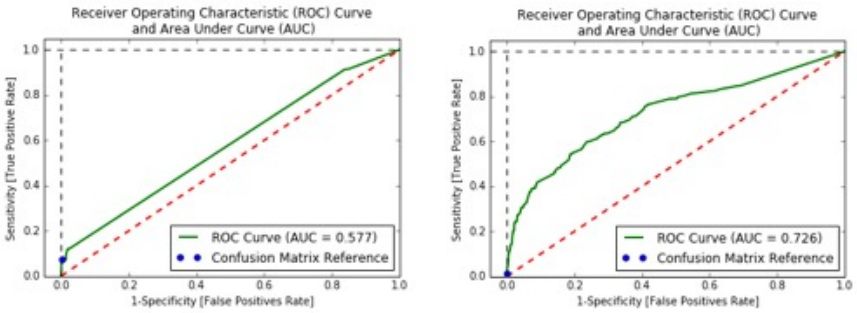
For the numerical new features, a set of derived features was created using the filters belonging to the *Statistical filters* (see Section 4.4). The categorical feature was

transformed in a set of boolean features following a procedure similar to that described in Section 4.3.1.

For the purposes of the experiment, only the historical data related to all the cells of Umbria with at list one alarm were downloaded. The experiment was performed under the following setup: dataset were constructed using the Grid selection strategy with grid step equal to 25 km, and the alarm range was fixed as [3,10].

In terms of precision, the results were discording. In some cases, the addition of weather variables did not produces any particular difference in the performances, as shown in Figure 4.26. In other cases, the effect was negative (Figure 4.27).

Weather experiment - Umbria



(a) Weather information included (b) Weather information excluded

Figure 4.26: Example of ROC with and without meteorological features

TN = 603312 (99.29)	FP = 730 (0.12)
FN = 3333 (0.55)	TP = 267 (0.04)

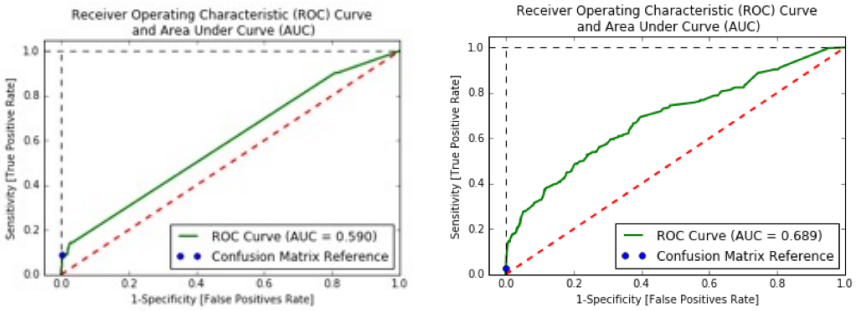
TN = 603909 (99.39)	FP = 133 (0.02)
FN = 3552 (0.58)	TP = 48 (0.01)

Population: 607642	**Positive: 3600**
Accuracy	0.993
Precision	0.268
Recall/Sensitivity	0.074
Specificity	0.999

Population: 607642	**Positive: 3600**
Accuracy	0.994
Precision	0.265
Recall/Sensitivity	0.013
Specificity	1.000

Table 4.20: Performances of predictions associated to Figure 4.26(a) (left) and Figure 4.26(b) (right).

Weather experiment - Umbria



(a) Weather information included (b) Weather information excluded

Figure 4.27: Example of ROC with (a) and without (b) meteorological features

TN = 498333 (99.02)	FP = 1855 (0.37)
FN = 2817 (0.56)	TP = 279 (0.06)

TN = 499944 (99.39)	FP = 244 (0.05)
FN = 3015 (0.60)	TP = 81 (0.02)

Population: 503284	**Positive: 3096**
Accuracy	0.991
Precision	0.131
Recall/Sensitivity	0.090
Specificity	0.996

Population: 503284	**Positive: 3096**
Accuracy	0.994
Precision	0.249
Recall/Sensitivity	0.026
Specificity	1.000

Table 4.21: Performances of predictions associated to Figure 4.26(a) (left) and Figure 4.26(b) (right).

4.9 Conclusions

This first run of experiments showed the potential of machine learning in the context of predictive maintenance. In some subsets of cells (25% of population) the sensitivity of the prediction reached over 50%, and as for the selection of negative labels (i.e. the prediction of a correct functioning of a cell) a precision near 90% was found over a population of the order of 70%.

These results, even though a lot higher than the ones associated to the random choice, are not useful in a real context of predictive maintenance. The presence of a great number of false alarms in particular would lead to a lot of unnecessary interventions that would result more expensive for the TELCO than those associated to last-minute maintenance. The problem was though to lie in the limited amount of both geographical and temporal data. A single month proved to be too short to train a performing prediction model, and the number of cells from which to take the samples was too small to lead to a good generalization.

The data provided during the second run led to slightly better results, indeed precisions over 15% were found on a significant percentage of the population, performing a geographical restriction of the training set according to one strategy or the other. However, they still were not sufficient to be of practical use. A third project was undergone, where the basic counters behind the KPIs were provided. The usage of these raw data somewhat increased the performance of the predictions, but mostly provided the TELCO's experts with more insights on the network system they used. A proposed strategy was the usage of other types of learning algorithm, belonging to the *neural networks*, as stated in [28].

Bibliography

[1] D. Amodei et al. (2016). Deep Speech 2 : End-to-End Speech Recognition in English and Mandarin. Proceedings of Machine Learning Research 48.

[2] J. Ll. Berral, I. Goiri, R. Nou, F. Juliá, J. Guitart, R. Gavaldá, J. Torres (2010). Towards energy-aware scheduling in data centers using machine learning. e-Energy '10 Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking, 215-224.

[3] C. M. Bishop (2006). Pattern Recognition and Machine Learning. Springer.

[4] A. Blumer, A. Ehrenfeucht, D. Haussler, M. K. Warmuth (1989). Learnability and the Vapnik-Chervonenkis Dimension. Journal of the Association for Computing Machinery. Vol. 36. No. 4, 929-965.

[5] B. Boser, I. Guyon, V. Vapnik (1992). A Training Algorithm for Optimal Margin Classifiers. InProceedings of the fifth annual workshop on Computational learning theory, Jul 1, 144-152, ACM.

[6] L. Breiman et all (1984). Classification and Regression Trees. Wadsworth, Belmont, CA.

[7] J. Burez, D. Van den Poel (2009). Handling class imbalance in customer churn prediction. Expert Systems with Applications 36 4626-4636

[8] C. Cortes, V. Vapnik (1995). Support-Vector Networks. Machine learning, Springer.

[9] I. Daubechies (1992). Ten lectures on wavelets. Siam.

[10] T.G. Dietterich (2000). Ensemble methods in machine learning. InInternational workshop on multiple classifier systems, Jun 21, 1-15, Springer, Berlin, Heidelberg.

[11] Y. Freund, R. E. Schapire (1996). Experiments with a new boosting algorithm. InIcml Jul 3, Vol. 96, 148-156

[12] X. Guo-en, J. Wei-dong (2008). Model of Customer Churn Prediction on Support Vector Machine. Systems Engineering - Theory & Practice, Volume 28, Issue 1, SETP, 28(1): 71-77.

[13] T. Hastie, R. Tibshirani and J. Friedman (2009). Elements of Statistical Learning. Springer.

[14] S. Haykin (1999). Neural Networks - A Comprehensive Foundation (2nd Edition). Pearson Prentice Hall.

[15] A.E. Hoerl, R.W. Kennard (1970). Ridge regression: Biased estimation for nonorthogonal problems. Technometrics. Feb 1;12(1):55-67.

[16] W. Huanga, Y. Nakamoria, S. Wang (2005). Forecasting stock market movement direction with support vector machine. Computers & Operations Research 32, 2513 - 2522

[17] S. C. Hui, G. Jha (2000). Data mining for customer service support. Information & Management 38, 1-13.

[18] S. Hung, David C. Yen, H. Wang (2006). Applying data mining to telecom churn management. Expert Systems with Applications 31 515-524.

[19] I. Kononenko (2001). Machine Learning for Medical Diagnosis: History, State of the Art and Perspective. Artificial Intelligence in medicine, Elsevier

[20] A. Liaw, M. Wiener (2002). Classification and regression by randomForest. R news, Dec 3, 2(3):18-22

[21] P Maes, RA Brooks (1990). Learning to Coordinate Behaviors. AAAI.

[22] T. Mitchell (1997). Machine Learning. McGraw Hill.

[23] E. Rosten, T. Drummond (2006). Machine learning for high-speed corner detection. European conference on computer vision, Springer

[24] A. L. Samuel (1959). Some studies in machine learning using the game of checkers. IBM Journal of research and development, 3(3), 210-229.

[25] A. Shalev-Shwartz and S. Ben-David (2014). Understanding Machine Learning - From Theory to Algorithms. Cambridge University Press.

[26] A. Smola and S.V.N. Vishwanathan (2008). Introduction to Machine Learning. Cambridge University Press.

[27] V. N. Vapnik, A. Ya. Chervonenkis (1971). On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. Theory of Probability & Its Applications. 16 (2): 264.

[28] G.M. Weiss (2002). Predicting Telecommunication Equipment Failures from Sequences of Network Alarms. To appear in W. Kloesgen and J. Zytkow (eds.), Handbook of Knowledge Discovery and Data Mining, Oxford University Press.

[29] M. M. Wolf (2018). Mathematical Foundations of Supervised Learning (growing lecture notes).

[30] H. Zou, T. Hastie (2005). Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society, Series B, 67(2):301-320.

[31] scikit-learn - Machine Learning in Python. Web site: https://scikit-learn.org/stable/.

[32] MongoDB Atlas. Web site: https://www.mongodb.com/.

[33] Python programming language. Web site:https://www.python.org/

Acknowledgements

This thesis has been written in an extremely difficult time, both for myself and my family. There is no doubt that, without the help and the support of many people, the sheets you are reading would not have been written. These people deserve all my thanks and gratitude, more than those I can express in this next few lines.

The first people I want to thank are my supervisors at University, prof. Michele Piana and especially Dr. Anna Maria Massone. Anna, thank you for your (great) patient. I know it has not been easy being my supervisor, and I know that without your support my PhD would almost certainly not be concluded.

I want to thank Dr. Matteo Santoro for having taken me after my graduation and included in the family of Camelot. Matteo, you have lead me along the path from the theoretical world to the one of practice and direct applications, and have been really understanding and kind throughout all these years. Thank you.

To my colleagues, Danilo, Andrea, Sofia, Curzio, Mario, Stefano, Agostino, Gaspare, Andrea, Federico, Marina, Alice, Francesco and Gabriele goes my gratitude for their support, the funny lunch times spent together and the occasional happy hours and dinners that were organized during these four years.

To my ex flat-mates, Selene and Nicola, I want to say thank you for the time spent together in that apartment with that really *laaarge* terrace. Now we have split to different regions, but I will always remember our dinners and cooking experiments.

Savio and One, you have taken me in with you whenever I needed a roof over my head - which, especially in the last year, happened *really* frequently. The new recipes that we tried on those occasions cannot repay you for all your support and kindness.

In Savona, thank you to Laura, Emily, Denise, Alessandro, Costanza and all my other friends for all the times we went out climbing or skiing, for the evenings at the boulder gym and the nights at Fronte's pub. Let's hope they will be followed by a long list of similar days.

A special thanks goes to Reforzo's family, Tiziana, Franco, Simone and Matteo, whom I consider part of my family and have proved to be true friends in the most difficult situations.

Finally, thank you to my mother and my father. Thanks for your support, your kindness and your understanding. Do believe that the worst is over, and that now you will be able to enjoy life. You have richly deserved it.

Stefania