POLITECNICO DI MILANO

School of Industrial and Information Engineering
Master of Science in Computer Engineering

POLITECNICO
MILANO 1863

DEVELOPMENT OF MACHINE LEARNING ALGORITHMS FOR LONG-TERM ROOMS RENTALS PRICING

Supervisor: Prof. Marcello RESTELLI
Co-supervisor: Francesco TROVÒ Ph.D.
Ing. Alessandro NUARA
Ing. Giulia ROMANO

Master Thesis dissertation of:
Giovanni Maria GIANOLA Matr. 894107

Academic year 2018-2019

Ringraziamenti

Vorrei ringraziare tutte le persone che mi sono state vicine durante il mio percorso universitario e che mi hanno permesso di ottenere questo splendido risultato.
Innanzitutto, un sentito ringraziamento al mio relatore Marcello perché per primo mi ha avvicinato e appassionato al mondo dell'intelligenza artificiale e mi ha permesso di svolgere questo lavoro di tesi. Grazie ai miei collaboratori: Francesco, Alessandro e Giulia, per i vostri consigli, per la vostra disponibilità e per avermi accolto nel vostro gruppo di ricerca.
Un ringraziamento alla mia famiglia che non ha mai smesso di credere in me e mi ha sempre supportato nelle mie decisioni durante tutto il percorso di studi. Grazie al loro instancabile sostegno, sia morale che economico, contribuendo alla mia formazione personale.
Infine, ringrazio tutti i miei amici e compagni che hanno condiviso con me momenti belli e brutti e hanno avuto un peso determinante nel conseguimento di questo risultato.

Grazie.

Giovanni

Sommario

Il mercato degli affitti a lungo termine di stanze nelle grandi città ha conosciuto un incremento considerevole negli ultimi anni, registrando in alcune città un aumento della domanda superiore all'80%. Data l'offerta di stanze ridotta e un flusso di utenti ancora contenuto, fino a qualche anno fa i pochi player sul mercato esistenti potevano permettersi di gestire manualmente il pricing delle stanze. Tuttavia, la contemporanea crescita del numero di stanze da affittare e la presenza di una concorrenza sempre più ampia hanno reso la gestione del processo di pricing attraverso sistemi automatici una necessità improrogabile. Questo ha portato DoveVivo, la prima e la più grande azienda italiana di co-living, ad investire in soluzioni tecnologiche per automatizzare il processo di pricing delle stanze. Questo progetto, nato dalla collaborazione con il Politecnico di Milano, si propone di creare modelli e algoritmi capaci di catturare le peculiarità di questo problema e di generare in maniera automatica degli schemi di pricing per le stanze in affitto. L'approccio seguito per sviluppare tali modelli è quello data-driven, motivato dal fatto che l'azienda target di questo studio possiede dei dati storici, i quali possono essere utilizzati per estrarre le informazioni necessarie per costruire modelli accurati. Il nostro contributo consiste nel fornire tutti gli strumenti di data analysis e machine learning necessari per trasformare i dati grezzi raccolti in informazioni utili per supportare il processo di decision making aziendale.

Proponiamo due diverse formulazioni del problema. Inizialmente vogliamo replicare il lavoro manuale eseguito dal team di pricing interno a DoveVivo, costruendo modelli che minimizzano l'errore stimato del prezzo di listino. Infine, partendo dall'idea che maggiore è il prezzo assegnato ad una stanza, maggiore sarà la probabilità che questa rimanga sfitta per un più lungo periodo di tempo, proponiamo un secondo approccio che si basa sull'identificare ed assegnare il più alto prezzo ad una stanza che minimizzi il numero di giorni di sfitto.

Dai nostri esperimenti abbiamo ottenuto dei risultati molto soddisfacenti a parere degli esperiti di pricing dell'azienda. Cercando di emulare l'assegnamento

dei prezzi che fa DoveVivo attualmente, otteniamo un errore di circa il 3% rispetto a quanto fatto dal team di pricing correntemente. Mentre i risultati della seconda formulazione ci mostrano come sia possibile incrementare i prezzi attuali di listino, fino ad un massimo del 5%, senza rischiare di avere periodi significativi di sfitto.

Abstract

The market for long-term rental of rooms in large cities has observed a considerable increase in recent years, recording in some cities an increase in demand of more than 80%. Given the reduced offer of rooms and a still limited flow of users, until some years ago, the few existing players in the market could afford to manage the room pricing manually. However, the simultaneous growth in the number of rooms to rent and the presence of ever stronger competition have made the management of the pricing process through automatic systems an essential need. This led DoveVivo, the first and largest Italian co-living company, to invest in technological solutions to automate the room pricing process. This project, born from the collaboration with the Politecnico di Milano, aims to create models and algorithms able to capture the peculiarities of this problem and automatically generate pricing schemes for rented rooms. The approach followed to develop these models is the data-driven one, motivated by the fact that the target company of this study has a large database of historical data, which can be used to extract the information necessary to build an accurate model. Our contribution is to provide all the data analysis and machine learning tools necessary to transform the raw data collected by DoveVivo into useful information to support the corporate pricing process.

We propose two different formulations of the problem. Initially, we want to replicate the manual work performed by the internal pricing team at DoveVivo, building models that minimize the estimated error of the listing price. Then, we propose a second approach that is based on identifying and assigning the best price to a room that minimizes the number of vacancy days. From the experiments conducted on real-world data we obtained satisfactory results: we get an error of about 3% when estimating the price set by the DoveVivo pricing team and show that it is possible to increase the current listing prices, up to a maximum of 5%, without risking to have days of vacancy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Milan has marked a new record in the rents domain, with requests that on average for a two-room apartment have reached almost $1,300$ euros per month, with an increase of 5.8% in the semester from March to September 2019, as emerged from data collected by the Immobiliare.it and Mioaffitto portals.[1] The prices increase are justified by a demand that continues to grow and marks +4.2% on a half-yearly basis and to which the offer is unable to keep up: compared to March 2019, the properties offered for rent fell by 3.2%. The rent demand, in Milan, first increased by nearly 80% in 2018, while the offer decreased, the number of leased properties, for example, decreased by 10%.[2] Since 2015, rental prices have increased by 22%, making Milan the most expensive city in Italy, with an average of $17,5$ €$/m^2$. Until twenty years ago, the contracts were concluded directly with the owners, who advertised their properties through paper ads or by relying on agencies, facing, in this case, high costs of commission. Today, however, owners and users can make rental contact more easily through online platforms. Some of these companies behave actively, renovating and renting rooms (DoveVivo), others replacing them in the role of the real estate intermediary. In this area, there is still no leading company, but the business is expanding very quickly.

In the face of the strong growth experienced by DoveVivo in recent years, automating the pricing procedure has become a tangible need. From 2013 to 2017 DoveVivo recorded an average turnover growth rate of 38%, currently manages a thousand houses in 5 cities and stipulates around 1400 contracts per year.[3] The expected growth for the next few years requires technological

[1] https://www.ilsole24ore.com/art/affitti-corsa-milano-altre-citta-aumenti-contenuti-AC6eZEj?fromSearch.

[2] https://www.ilsole24ore.com/art/affitti-milano-boom-domanda-e-canoni-rialzo-AE27TE6E.

[3] https://www.dovevivo.it/.

support capable of managing the complexity of the problem by analyzing the numerous parameters that characterize the environment, in which DoveVivo offers its services.

This project, born from the collaboration with the Politecnico di Milano, aims to capture the peculiarities of the problem and automatically generate pricing schemes for rented rooms via machine learning models and algorithms. The approach followed to develop these models is the data-driven one, motivated by the fact that the target company of this study, DoveVivo, has historical data, which can be used to extract useful information in order to build an accurate pricing model.

We start by focusing on the data provided by DoveVivo gathered in the past few years. The initial phase concerns a process of inspection, cleaning, transformation and integration of data intending to obtain useful information. Our work follows two main directions:

- the first approach aims to emulate the manual work done by DoveVivo's pricing team and then generalize the pricing method. These methods take into account various characteristics of the room and flat, as well as geographical factors such as coordinates and neighborhood;

- the second approach takes advantage of the contract history of each room. We analyze the past contracts of each room and generalize the price trend also for future samples. In this case, the criterion used is to minimize the number of vacancy days of a room given a price.

The predictions provided by these models are intended as a suggestion for the company itself useful for decision-making.

The thesis is structured in the following way:

- in Chapter 2 we provide a general description of DoveVivo. We give a general idea of the company's *business model* and the used approach for the rooms' pricing;

- In Chapter 3 we provide two mathematical representations of the problem of finding the right price to assign to a room;

- in Chapter 4 we provide the theoretical foundation on which this work is based. We present some key concepts from the machine learning field, learning algorithms for regression and classification problems and the metrics we used to evaluate our work;

- in Chapter 5 we firstly show some basic techniques to handle and pre-process data to be used for machine learning models. Then, we provide the pipeline's steps used for each problem definition: from raw data to a clean dataset;
- in Chapter 6 we present the dataset we used for this work, showing its main characteristics and limitations, and we provide some experiments to validate our choices;
- in Chapter 7 we draw conclusions, summarize the main results of our work, and propose some future developments.

Chapter 2

State of the Art

In this chapter, we give a general perspective of DoveVivo, presenting its business model based on the concept of co-living. Then we describe the process followed by a potential client to rent a room and the process used by the company's pricing team to price a new room. Finally, we provide an overview of some works on the pricing problem present in the literature.

2.1 Business Model

DoveVivo managed to understand in advance what the needs of the market were, and at the same time identified a gap in the sector, completely fragmented. The company has started an industrialization process by proposing an offer conceived and created following a model that goes from researching the property to marketing the room product. The model satisfies the need to offer co-living solutions to an increasingly large pool of potential students and off-site workers and, at the same time, to cope with the management and rental of a large number of properties. As shown in the Figure 2.1 the business model starts, in fact, from the research and analysis of mainly large flats ($\approx 80/100\ m^2$) and their context to enter into a standard long-term lease agreement with the corresponding owner ($8-10$ years) with authorization to sublease. Throughout the period, DoveVivo guarantees a constant income for the owners and eliminates the associated costs. DoveVivo's team of architects, designers and technicians renovate and furnish the apartments with new and trendy furniture, modern and functional appliances and accessories. DoveVivo also takes care of all the paperwork, insures the property against damages and handles all the property management responsibilities. From that moment, the model develops along a structured process that allows home seekers to use the dovevivo.it website to select a new home based

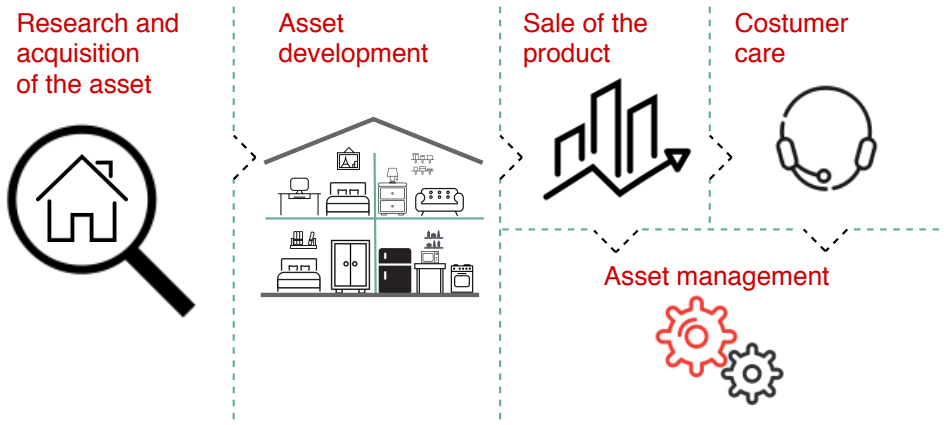


Figure 2.1: DoveVivo's business model.

on the type of needs, the area and the preferences related to the housemates. DoveVivo represents, for the entire duration of the contract, the point of reference for tenants and owners, offering complementary services to the rent: fee including utilities, 24-hour assistance, ordinary maintenance, rapid and professional management of extraordinary maintenance interventions, relationship with condominium administrations and between co-tenants, a dedicated app for a smart and direct relationship with tenants, loyalty card with discounts and benefits. The purpose of this integration is to offer a service, supported by structured processes, digital tools and a strong customer care orientation, which fully satisfies the needs of the targets involved and which allows us to accompany them from brand consideration to loyalty and final satisfaction.

2.2 Rental Process

Despite this rapid expansion, the price of available rooms is still established manually, using the experience of the sales team. The growth forecast for the next few years, however, makes it necessary to create adequate technological support, capable of automatically optimizing the pricing process.

Currently, the rental process followed by a user to go from a possible renter to a tenant, illustrated in Figure 2.2, takes place through the following phases:

- the potential customer gets in touch with DoveVivo through the website, social networks (e.g., Facebook) or by going directly to the company's headquarter. The user also has the option to virtually inspect

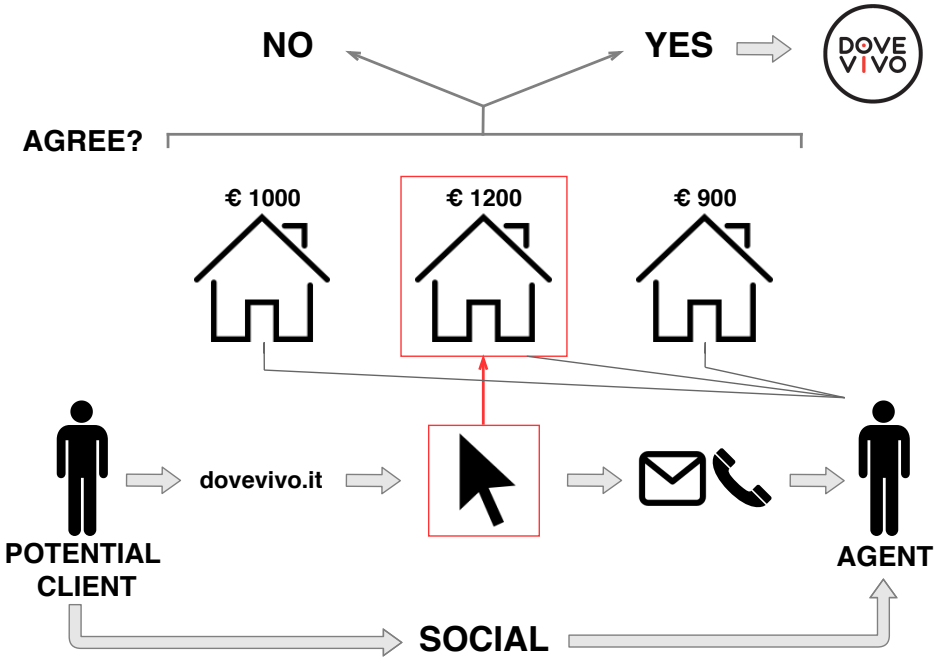


Figure 2.2: Renting process.

some sample flats through the website;

- customer interested in seeing the flat in person can arrange an appointment with a DoveVivo agent who proceeds by scheduling a visit to the desired room. The agent may also propose some other room, up to three, some of which may not be present among those published online on the website;
- finally, if the customer is interested in one of the rooms visited, he proceed with the signing of the pre-contract and, then, of the contract.

2.3 Pricing process

Currently, room prices are assigned manually by evaluating the following characteristics:

- **room**: square footage, bed size, balcony, private bathroom, walk-in closet, single/double, electric stove;
- **apartment**: number of tenants, the ratio of bathrooms to tenants, dimension of common areas, presence of a living room, new or renovated bathrooms, floor, washing machine, dishwasher, wi-fi;

- **building**: status of the building, reception, lift, neighborhood, public transport, services (for example supermarket and pharmacy).

Some of these features are not stored in the databases, for instance, electric stove, presence of a living room, new or renovated bathrooms, status of the building, services. The only way to confirm the presence and the quality of these services is by checking the room of interest in person or using photos of the room. The sales department creates a list of fixed prices which are then modified, taking into account the time of year when the room is vacant. Given the high demand, the price applied might vary over time, for instance, during September, the price may be higher than the one in June. The list of prices is based on the months in which there is a high demand. The goal is to be able to sign an indeterminate contract in these months to maximize profit. One of the techniques currently used to manage the months with the least demand is to provide a discount on the rent or to stipulate fixed-term contracts with a lower price, which last up to those periods with high demand. At this point, the tenant can choose whether to stay by signing an indeterminate contract with a higher price, equal to the listing price. There are two possible scenarios in which a room needs a new price: DoveVivo acquires a new apartment and, therefore, a new price must be assigned to the rooms, or the renter leaves a room, and its price must be updated. The processes currently used to price a room in the two cases are as follows:

- price estimation for a new bedroom s:
  - check the price $p(s_1)$ of s_1, the closest DoveVivo's room to s;
  - observe and compare the characteristics of s and those of s_1;
  - get an estimation of the price $p(s)$ through a comparison with s_1 (for example, evaluating present and absent factors and modifying the price accordingly).
- price update of a room s:
  - the prices $p(s)$ are updated periodically based on market trends;
  - the prices $p(s)$ are increased by an additional $\Delta(s)$ based on the room's furniture.

Note that at the moment both these operations are performed manually, relying on the experience of the pricing team present at DoveVivo. This approach clearly does not scale well in the case the turnover of the renters and acquired rooms becomes larger and larger.

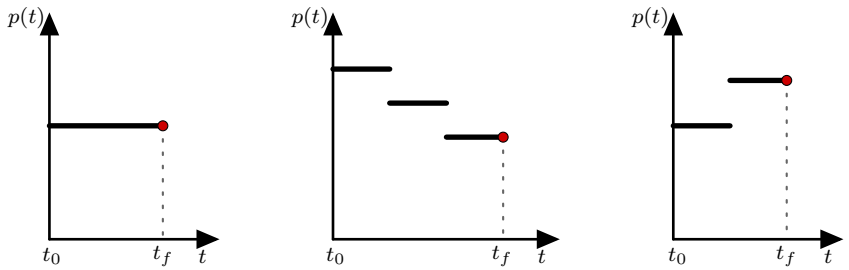


Figure 2.3: Online prices trends.

2.4 Publication of an online price

The phase that goes from the online publication of a price for a room to the signing of a new contract, can take place with 3 different scenarios. Figure 2.3 shows the trend over time of the price published online for a given room. The price is made visible at the instant t_0 while t_f is the instant in which a user agrees to rent that bedroom at the corresponding price. The first plot represents the case in which DoveVivo proposes a price which, after a certain time, is accepted by the customer. The second graphic shows the case in which the proposed price is not accepted by any customer for a certain period and therefore is lowered hoping to find an available user. The third graphic, in contrast, represents the situation in which DoveVivo offers a very low price and, given the massive influx of customers, decides to raise it. The first case seems to be the most common. Price changes represented by the second and third graphics are not currently traced. The data exposed by the databases are the listing price and the contract price which is the one accepted by the user as a monthly fee, that is, the one at time t_f. In general, DoveVivo's intention is to avoid price changes during the negotiations phase which goes from the online publication date of a room to the signature of its lease. They would like to set a fixed and appropriate price for each room before publishing it online.

2.5 Related Works

One of the main reasons that push the real estate agencies to resort and invest in complex technologies in the pricing field is the economic growth in Italy after 2015. The Real Estate Market report of Caldirola and Martino

(2016) analyze the trend of the Italian real estate market and in particular how this growth is related to the increase in investments in the retail sector. In traditional pricing problems, are usually analyzed scenarios in which the seller owns a *limited* (Babaioff et al., 2015) or *unlimited* (Trovò et al., 2018; Trovo et al., 2015) quantity of identical goods. In our case, however, we rarely have to price two products (bedrooms) with the same characteristics. Therefore, it is not possible to accurately estimate the demand curve. An alternative solution, is provided by Ye et al. [2018], who offer a tool that helps landlords on *Airbnb* to estimate an excellent daily price. This solution cannot be used in our setting as it does not incorporate an estimate of the period of stay, crucial for determining the earnings in our specific case. In fact, it assumes that users' periods of stay are dictated by factors that cannot be controlled by the owner, while experts in the sector have confirmed that staying in a rented room for an extended period is greatly influenced by the price of the room (Gibbs et al., 2018). More generally, we can model our problem as a posted-price auction Chawla et al. (2010), in which the price of an asset changes dynamically over time. However, these algorithms require knowledge of parameters that are unknown in real applications, such as the probability distribution of values assigned by users to goods. The growth of the real estate market in recent years has made this sector increasingly competitive. For this reason, together with the non-disclosure of data for commercial purposes, we are not aware of the pricing systems of other private companies.

Chapter 3

Problem Formulation

In this chapter, we provide the formulation of the pricing problem that we are going to tackle from a mathematical perspective. We start by defining the potential renter and how he/she evaluates a room, introducing the concept of *Reservation Price*. Then, we give an overview of the room evaluation phases followed by a user before renting a room. We also provide a high-level description of the update mechanism of the user's *utility* of a room, which is a value that represents a user's intention to sign a lease for a specific place. Finally, we provide two different formulations of the problem of finding the right price to assign to a room exploiting DoveVivo's data, more specifically:

- maximizing the revenue;
- minimizing the number of vacancy days.

DoveVivo collects several data about rooms such as location, prices, and features, historical contracts and flats characteristics. We focused our analysis on the rooms located in Milan. Each apartment is assigned to a geographical area defined by DoveVivo. Each area broadly corresponds to a set of neighborhoods in Milan.

3.1 User Formulation

To better illustrate the evaluation process of a property, we formally define what we mean by a potential renter and what are the steps she/he takes before signing a rental contract.

3.1.1 Reservation Price

Each user has a personal evaluation of a property which, we assume, is the price he/she is giving to the rental of a specific bedroom. Every user

establishes his/her *reservation price*, that is, the maximum price that he/she is willing to pay to obtain that good. Below we define the reservation price v_{ji} and the u_{ji} utility of the i-th user for the j-th room:

$$v_{ji} = f(x_{j1}, x_{j2}, \ldots, x_{jM}), \tag{3.1}$$

$$u_{ji} := v_{ji} - p_j, \tag{3.2}$$

where p_j is the price assigned to the j-th bed, x_{jk} is the k-th feature, in a set of M different features, of the j-th bed.

Formally, before entering the website, the user i-th has a reservation price v_{jk} for a generic room S_k in a set of possible S rooms. The rooms currently available are included in the set $D \subset S$ and have prices $p_1, \ldots, p_K$. The i-th user is interested in the room S_k if and only if its utility value u_{ji} is positive. Suppose that the user behaves optimistically, that is if he/she knows partial information about a room S_k, he/she assigns the maximum possible evaluation for a room in S with characteristics consistent with the room S_k. This way, the assessment of a user turns out to be a monotonous function weakly decreasing with the advancement of the phases and, due to this decrease, the user abandons the purchase if in a particular phase its utility is negative. The deal of a contract occurs only if the utility of the room S_k selected on the website is positive in the signing contract phase.

Since users assign a different price for each room, we model the set of possible users through a distribution. If we knew the real distribution of users for the evaluation of a specific room, it would be enough to recommend a price slightly lower than the maximum reservation price to have a good trade-off between cost and time it takes to sell the room. Although, in a realistic scenario, it is not possible to know this information. Therefore, we have to estimate the distribution of users.

Assume, for simplicity, that users follow a given Gaussian distribution. First, we choose a price to assign to a room, and, then, the number of users that are coming to evaluate the offer is extracted from the distribution. We face two possible situations:

- If the user's reservation price is below the price chosen for the room, the room is not chosen by the user;
- Otherwise, if the reservation price is extracted above the chosen price, the room is taken.

Intuitively, the higher the price we assign to the room, the longer it will take to be rented by a user. That means that the more we increase the room price, the lower is the probability of extracting a user with a suitable reservation

price, and consequently, it takes more time to rent it. On the other hand, if we assign a low price, the higher is the probability to sample a useful user, the time to rent the room is reduced accordingly, but the profit will likely decrease. This is, in a nutshell, the problem of selecting the optimal price, i.e., the tradeoff between revenues and time spent in waiting for the suitable user.

3.1.2 Room Evaluation Process

The evaluation process of a room by a potential renter is divided into three *phases* in which he or she acquires more and more information gradually.

1. In the first phase, the user gets in touch with DoveVivo through the website. Here the client can observe the map of the selected city and the markers placed on it that indicate the position and price of each room. From these first two characteristics (price and geographical area), the user starts to create a first reservation price v_{ji} for each room j observed. Although, this assessment is based on incomplete information and will, therefore, be intended to modify with the progress of the phases and information acquired. At this point, it is possible to calculate the utility u_{ji} of a certain room. If the utility is greater than zero, the user is prompted to obtain further information. Otherwise, the user will discard the room and focus his/her interest on those who can still provide him/her with some positive utility. If all the observed rooms provide a negative utility, the user abandons the search and will not proceed in the following phases.

2. In the second phase, the user selects the rooms that in the first phase had provided a positive utility. In this way, always through the site, the features of the room are disclosed, for instance, the room and flat square footage, the presence of a balcony, the number of roommates. This new piece of information helps the user to update the reservation price by getting v'_{ji}. Again the new reservation price can give a positive or negative utility u'_{ji}. In the second case, the user discards the room.

3. In the last phase, the user obtains all the information on the room after a personal visit. The user creates a final opinion of the room by observing a different set of features of the room, such as the neighborhood, the people who are currently living there, the roommates, the aesthetics of the building. The reservation price v''_{ji} is the final one. If still u''_{ji} is positive for a certain room, it means that the user, observing all its characteristics, agrees to rent it.

We model optimistic users by assuming that if he/she does not know a specific feature of any room, his/her utility assumes the best values contextually. As the user passes through phases, he/she becomes more aware of the characteristics of any room and refines the first impressions of the room. The evaluation of the user turns out to be a monotonous function that decreases marginally with the progression of the phases and, as a result of this decrease, the user abandons the purchase if its utility is negative during a particular stage.

If we knew the users' evaluations for a specific room, we would have to recommend a price slightly lower than the maximum reservation price. In a real scenario, it is not possible to know this information. So we try to estimate v_{ji} as well as possible with the available data. The given *contract price* can undoubtedly be considered a lower bound of the user's reservation price.

3.2 Formulation

3.2.1 Maximize the Revenue

The idea is to emulate the pricing process carried out by DoveVivo given a data set of rooms. Formally, given an $N \times M$ dataset $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ corresponding to N rooms, where $\mathbf{x}_n$ represent the vector of the M features of the rooms and the corresponding continuous target values $\mathbf{t} = \{t_1, \ldots, t_N\}$ indicating the observed listing price of the rooms, we provide an estimation of the unknown mapping $f : \mathbf{X} \to \mathbf{t}$. We want to minimize the estimation error, that is, the expected value of the reconstruction error over the input space. As a proxy of this unknown quantity, we choose as a *loss function* the empirical expected value $L(\mathbf{w})$ of the *squared error*:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (f(x_n, \mathbf{w}) - t_n)^2. \tag{3.3}$$

For each room n, we have a vector of features $\mathbf{x}_n = \{x_{n,1}, \ldots, x_{n,M}\}$ that describe the apartment and the area in which it is located. It includes:

- *categorical features*, such as the apartment address or the room type;
- *binary features*, representing the presence or absence of a characteristic such as reception, air conditioning or cellar;
- *numerical features*, such as the square footage, the number of rooms in the apartment or the floor.

The target t_n represents the listing price of the n-th room, assigned by DoveVivo. This is a regression problem and our goal is to build a predictive model, exploiting different available data. The model aims to suggest DoveVivo employees a listing price for an unseen new room in the Milan area, maximizing the revenue while finding a price as close as possible to users' *reservation prices*.

3.2.2 Minimize the Number of Vacancy Days

The same problem can be formulated differently if we already have information about past rent contract for the room: try to find the best price that allows to rent a room in the shortest possible time. In this circumstance, an excellent way to generalize the listing price of a room is by minimizing the number of vacancy days.

The lack of a large number of data leads us to a more effective formulation of the problem that takes into account the history of the contracts of the rooms. In addition to the above-mentioned features specific of the room, we also have:

- the start and end date of the contract;
- the days of vacancies occurred before signing the contract;
- a price associated with each contract.

Taking into account the contracts' characteristic combined with the features of the room as in the previous formulation, we want to predict a binary target value that identifies whether a room generates a vacancy or not.

We have an $N \times M$ dataset of inputs $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$ called design matrix. $\mathbf{z}_i = (\mathbf{x}_i, \mathbf{y}_i, t_i)$ where i is a the index corresponding to a contract for a specific room in the dataset. $\mathbf{x}_i$ denotes the features, $\mathbf{y}_i$ represents the contracts data and t_i is the price assigned to the room i at the time of contract $\mathbf{y}_i$. The target value s_i, which represents the vacancy, can assume two different class of values:

- $s_i \in \{0, 1\}$ in this case we have a binary target value denoting whether or not the room generated vacancy.
- $s_i \in [0, +\infty)$ in this case we have a continuous target value denoting the number of vacancy days.

we want to determine the unknown mapping $\tilde{f} : \mathbf{Z} \to \mathbf{s}$. According to the type of target, there are two distinct classes of problems: the former one is a classification problem; the latter one is a regression problem.

Chapter 4

Theoretical Background

4.1 Machine Learning

Machine Learning (ML) is a field of Artificial Intelligence (AI) that provides systems with the capability to automatically learn and improve from experience without being explicitly programmed (Bishop, 2006). Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

> *"A computer program is said to*
> *learn from experience E*
> *with respect to some class of tasks T*
> *and performance measure P,*
> *improves with experience E"* Mitchell et al. (1997)

The process of learning starts with observations or samples, such as examples, direct experience, or instruction, to search for patterns in data and make better decisions in the future based on the samples that we provide. The purpose is to allow computers to learn automatically without human intervention or assistance and adjust actions accordingly.

Technically, ML is the systematic study of algorithms and statistical models that computer systems use to accomplish a specific task without using precise instructions, relying on patterns and deduction instead. Machine learning algorithms build a mathematical model based on sample data, known as training data, to make predictions or decisions. Machine learning algorithms are used in a wide variety of applications, such as email filtering, computer vision and stock prediction, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task. The study of mathematical optimization delivers methods, theory and application do-

mains to the field of machine learning. In its application across business problems, machine learning is also referred to as predictive analytics.

4.1.1 Supervised Learning

Supervised learning is the largest, most mature, most widely used sub-field of machine learning. Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs Russel (2007). The data is known as training data and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and a matrix represents the training data. Through iterative optimization of an objective function, supervised learning algorithms aim to estimate the unknown model that maps known inputs to known outputs Mohri et al. (2018). An optimal function will allow the algorithm to determine the output for unseen inputs correctly.

Definition 1. *Given a **training set** of N example input-output pairs*

$$(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N), \tag{4.1}$$

where each y_j was generated by an unknown function $y = f(x)$, supervised learning aims to find a function $h(\cdot)$ that approximates the true function $f(\cdot)$.

Here x and y can be any numerical value. The function $h(\cdot)$ is a hypothesis. Learning is a search through the space of possible hypotheses for one that will perform well, even on new examples beyond the training set. To measure the accuracy of a hypothesis, we give it a test set of examples that are distinct from the **training set**. We say a hypothesis generalizes well if it correctly predicts the value of y for different examples. Sometimes the function f is stochastic, it is not strictly a function of x, and what we have to learn is a conditional probability distribution, $P(Y|x)$. When the output y is one of a finite set of values, the learning problem is called classification and is called binary classification if there are only two values (true or false, for example). When y is a continuous value, the learning problem is called regression. Technically, solving a regression problem is finding a conditional expectation or average value of y, because the probability that we have found precisely the right real-valued number for y is 0.

4.2 Linear Models for Regression

The purpose of regression is to predict the value of one or more continuous target variables. Given a training data set comprising N observations x_n, where $n \in \{1, \ldots, N\}$, together with corresponding target values y_n, the goal is to predict the value of t for a new value of x. In the simplest approach, this can be done by directly constructing an appropriate function $y(x)$ whose values for new inputs x constitute the predictions for the corresponding values of t in such a way as to minimize the expected value of a suitably chosen loss function. A common choice of loss function for real-valued variables is *the squared loss*, for which the conditional expectation of t gives the optimal solution.

4.2.1 Linear Basis Function Models

The simplest linear model for regression is the one that involves a linear combination of the input variables

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \ldots + w_D x_D, \tag{4.2}$$

where $\mathbf{x} = (x_1, \ldots, x_D)^T$. This is often simply known as linear regression. The key property of this model is that it is a linear function of the parameters $w_0, \ldots, w_D$. It is also, a linear function of the input variables x_i, and this imposes significant limitations on the model. Therefore is possible to extend the class of models by considering linear combinations of fixed nonlinear functions of the input variables, of the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}), \tag{4.3}$$

where $\phi_j(x)$ are known as basis functions. By denoting the maximum value of the index j by $M-1$, the total number of parameters in this model will be M. The parameter w_0 allows for any fixed offset in the data and is sometimes called a *bias* parameter. It is often convenient to define an additional dummy 'basis function' $\phi_0(x) = 1$ so that

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}), \tag{4.4}$$

where $\mathbf{w} = (w_0, \ldots, w_{M-1})^T$ and $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_{M-1})^T$. By using nonlinear basis functions, we allow the function $y(\mathbf{x}, \mathbf{w})$ to be a nonlinear function of the input vector $\mathbf{x}$. Functions of this form are called linear models, however,

because this function is linear in $\mathbf{w}$. It is this linearity in the parameters that will greatly simplify the analysis of this class of models. There are many other possible choices for the basis functions, for example

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}, \tag{4.5}$$

where the μ_j govern the locations of the basis functions in input space, and the parameter s governs their spatial scale. These are usually referred to as *Gaussian* basis functions. Another possibility is the *sigmoidal* basis function of the form

$$\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right), \tag{4.6}$$

where $\sigma(a)$ is the logistic sigmoid function defined by

$$\sigma(a) = \frac{1}{1+\exp(-a)}. \tag{4.7}$$

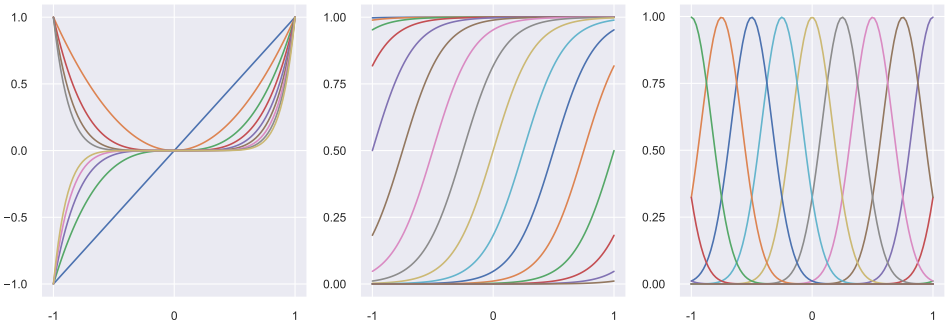


Figure 4.1: Examples of basis functions, showing polynomials (on the left), sigmoid functions (center), and Gaussian basis (on the right).

4.2.2 Loss function for Regression

The loss function is a method of evaluating how well specific algorithm models the given data. If prediction deviates too much from actual results, loss function will arise a huge number. It is needed to define the loss error function $L(t, y(\mathbf{x}))$ where t is the output and y is what your model predicts. Once the loss function is defined, the expected value is computed concerning a conditional probability distribution P. P is a function of the input x and the output t, the distribution that is expected to have over test samples. The *average* , or expected loss is given by:

$$\mathbb{E}[L] = \int\int L(t, y(\mathbf{x}))p(\mathbf{x}, t)d\mathbf{x}dt. \tag{4.8}$$

A common choice is the *squared loss function*, where L is the difference between t and $y(x)$ to the power of 2.

$$\mathbb{E}[L] = \int \int (t - y(\mathbf{x}))^2 p(\mathbf{x}, t) d\mathbf{x} dt. \tag{4.9}$$

The optimal solution (if we assume a completely flexible function) is the *conditional average*:

$$y(\mathbf{x}) = \int t p(t|\mathbf{x}) dt = \mathbb{E}[t|\mathbf{x}]. \tag{4.10}$$

4.2.3 Direct Approach: Minimising Least Squares

Unlike a probabilistic approach, direct optimization one simply tries to search the space of the models to minimize the loss function. This means finding a regression function $y(x)$ directly from the training data. Given a data set with N samples, consider the following error function:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, \mathbf{w}) - t_n)^2. \tag{4.11}$$

This is half the *residual sum of squares* (RSS), also called *sum of squared errors* (SSE) It can also be written as the sum of the l_2-norm of the vector of residual errors.

$$RSS(\mathbf{w}) = \|\boldsymbol{\epsilon}\|_2^2 = \sum_{i-1}^{N} \epsilon_i^2. \tag{4.12}$$

In mathematics, an expression is said to be a *closed-form* if it can be expressed analytically in terms of a finite number of certain "well-known" functions. Typically, these functions are defined to be elementary functions constants or elementary operations of arithmetic. The *Ordinary least square* (OLS) is an example of a closed-form solution to estimate the parameter of a linear regression model. Let us introduce the matrix Φ, which is the matrix with the number of rows equal to the number of samples N, and a column for each feature M. Let us write RSS in matrix form with $\Phi = (\boldsymbol{\phi}(\mathbf{x_1}), \ldots, \boldsymbol{\phi}(\mathbf{x_N}))^T$, also called *design matrix* ($\Phi \in \mathbb{R}^{N \times M}$), and $\mathbf{t} = (t_1, \ldots, t_N)^T$

$$L(\mathbf{w}) = \frac{1}{2} RSS(\mathbf{w}) = \frac{1}{2} (\mathbf{t} - \Phi\mathbf{w})^T (\mathbf{t} - \Phi\mathbf{w}). \tag{4.13}$$

We have rewritten the L formula in matrix notation. Now we have to find the minimum of the function, which means that the gradient with respect to

w must be zero and the ascend point (curvature) must be positive. Compute the first and second derivatives.

$$\frac{\partial L(\mathbf{w})}{\partial \mathbf{w}} = -\Phi^T(\mathbf{t} - \Phi\mathbf{w}), \tag{4.14}$$

$$\frac{\partial^2 L(\mathbf{w})}{\partial \mathbf{w}\partial \mathbf{w}^T} = \Phi^T\Phi. \tag{4.15}$$

The ascend is $\Phi^T\Phi$ which is symmetric, semi-positive (all the item values are non-negative). If they are all positive it means you have only one minimum. If they have zeros, it means that you have more than one solution. Assuming $\Phi^T\Phi$ nonsingular, the weights of the model:

$$\hat{\mathbf{w}}_{OLS} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{t}. \tag{4.16}$$

Theorem 1. *(Gauss-Markov Theorem) The least squares estimate of* $\mathbf{w}$ *has the smallest* ***variance*** *among all linear* ***unbiased*** *estimates.*

It follows that the least-squares estimator has the lowest Mean Square Error (MSE) of all linear estimators with no bias. However, there may exist a biased estimator with smaller MSE.

4.2.4 Regularized least squares

In linear models, overfitting is typically associated with large weight values. To avoid overfitting (Bülhmann and van der Geer, 2011), the loss function can be modified by adding a component related to the complexity of the model:

$$L(\mathbf{w}) = L_D(\mathbf{w}) + \lambda L_W(\mathbf{w}), \tag{4.17}$$

where $\lambda \in \mathbb{R}^+$ is the regularization coefficient that controls the relative importance of the *data-dependent error* $L_D(\mathbf{w})$ and the *regularization term* $L_W(\mathbf{w})$.

By introducing the parameter λ, the optimization becomes a trade-off between the minimization of the data error and the complexity of the model. The model complexity is the order of magnitude of the weights. The regularisation parameter controls this trade-off: high λ values lead to a strong regularisation, that is to prefer simple models, while low λ values imply a low regularisation, that is more complex models. The choice of $L_W(\mathbf{w})$ determines the type of regularisation. Among the most common regularisation choices we find:

- *Ridge Regression*: $L_W(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$;

- *Lasso Regression*: $L_W(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|_1$;

where $\|\mathbf{w}\|_2 = \sqrt{\sum_{j=0}^{M-1} w_j^2}$ and $\|\mathbf{w}\|_1 = \sum_{j=0}^{M-1} |w_j|$. Ridge regularization, also known as *Tikhonov regularization* or $l2$, shrinks the coefficients and it helps to reduce the model complexity and multi-collinearity. So lower the constraint (low λ) on the features, the model will resemble linear regression model. In this case the loss function is still quadratic in parameters $\mathbf{w}$, therefore, exists a closed-form solution. The vector of the optimal parameters becomes:

$$\hat{\mathbf{w}} = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T \mathbf{t}, \tag{4.18}$$

where I is the identity matrix.

On the other hand, lasso regression not only helps in reducing over-fitting, but it can help us in feature selection. This type of regularisation ($l1$) for λ significantly high can lead to zero coefficients forming a sparse model. For example, some of the features are entirely neglected for the evaluation of output. Just like Ridge regression, the regularisation parameter λ can be controlled.

4.2.5 Non-Negative Least Square

The estimation of parameters in an OLS problem is not always that straightforward because in many real-world problems the underlying parameters represent quantities that can take on only non-negative values, for example, amounts of materials, chemical concentrations, pixel intensities, to name a few. In such a case, the least-squares problem must be modified to include non-negativity constraints on the model parameters (Chen and Plemmons, 2010). The resulting problem is called Non-negative Least Squares (NNLS), and is formulated as follows:

Problem 1. *(NNLS Problem) Given a matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and the set of observed values given by* $\mathbf{b} \in \mathbb{R}^n$*, find a non-negative vector* $\mathbf{x} \in \mathbb{R}^n$ *to minimize the functional* $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$*, for example*

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) = \frac{1}{2} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2, \\ &\textit{subject to } \mathbf{x} \geq 0. \end{aligned} \tag{4.19}$$

Here $\mathbf{x} \geq 0$ means that each component of the vector $\mathbf{x}$ should be non-negative, and $\|.\|^2$ denotes the $l2$ norm.

The first widely used algorithm for solving this problem is an *active-set method* published by *Lawson and Hanson (1995)* in their book *Solving Least*

Squares Problems. In pseudocode, this algorithm looks as follows (Chen and Plemmons, 2010):

Algorithm 1 `Non Negative Least Square - NNLS`

Input $\mathbf{A} \in \mathbf{R}^{m \times n}, \mathbf{b} \in \mathbf{R}^m$
Output $\mathbf{x}^* \geq 0$ such that $\mathbf{x}^* = \arg\min \|\mathbf{Ax} - \mathbf{b}\|^2$.

function NNLS($\mathbf{A} \in \mathbf{R}^{m \times n}$, $\mathbf{b} \in \mathbf{R}^m$)
  Initialization: $P = \emptyset$, $R = \{1, 2, \ldots, n\}$, $\mathbf{x} = \mathbf{0}$, $\mathbf{w} = \mathbf{A^T}(\mathbf{b} - \mathbf{Ax})$
  repeat
    $j = \arg\max_{i \in r}(w_i)$
    Include the index j in P and remove it form R
    $\mathbf{s}^P = [(\mathbf{A}^P)^T \mathbf{A}^P]^{-1} (\mathbf{A}^P)^T \mathbf{b}$
    repeat
      $\alpha = -\min_{i \in P}[x_i/(x_i - s_i)]$
      $\mathbf{x} \coloneqq \mathbf{x} + \alpha(\mathbf{s} - \mathbf{x})$
      Update R and P
      $\mathbf{s}^P = [(\mathbf{A}^P)^T \mathbf{A}^P]^{-1} (\mathbf{A}^P)^T \mathbf{b}$
      $\mathbf{s}^R = 0$
    until $\min(\mathbf{s}^P) > 0$
    $\mathbf{x} = \mathbf{s}$
    $\mathbf{w} = \mathbf{A^T}(\mathbf{b} - \mathbf{Ax})$
  until $R = \emptyset \vee [\max_{i \in R}(w_i) > tolerance]$
end function

Notation: The matrix $\mathbf{A}^P$ is a matrix associated with only the variables currently in the passive set P. It is proved by Lawson and Hanson that the iteration of the NNLS algorithm is finite. Given sufficient time, the algorithm will reach a point where the *Kuhn-Tucker* conditions are satisfied, and it will terminate (Chen and Plemmons, 2010).

4.2.6 Bounded - Variable Least Square

A generalization of NNLS is *bounded-variable least squares* (BVLS), with simultaneous upper and lower bounds $\alpha_i \leq x_i \leq \beta$ (Stark and Parker, 1995). This problem rise when is needed to apply a different constraint for each variable of a least-square problem, and is formulated as follows:

Problem 2. *(BVLS Problem) Given a matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and the set of observed values given by* $\mathbf{b} \in \mathbb{R}^n$*, find a constraints vector* $\mathbf{x} \in \mathbb{R}^n$ *to minimize the functional* $f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|^2$*, for example*

$$\min_{\mathbf{x}} f(\mathbf{x}) = \frac{1}{2} \|\mathbf{Ax} - \mathbf{b}\|^2, \quad \textit{subject to } \alpha_i \leq \mathbf{x} \leq \beta_i. \tag{4.20}$$

$\forall \alpha, \beta \in (-\infty, +\infty)$.

Here $\alpha_i \leq \mathbf{x} \leq \beta_i$. means that each component of the vector $\mathbf{x}$ should be in the range of (α, β), and $\|.\|^2$ denotes the $l2$ norm. This optimization problem is convex, hence a found minimum (if iterations have converged) is guaranteed to be global. BVLS uses an active set strategy similar to that of NNLS, except two active sets are maintained, one for variables at their lower bounds and one for variables at their upper bounds. The proof that BVLS converges to a solution of problem bvls follows that for NNLS (Lawson and Hanson, 1995).

4.3 Non-Parametric models

The methods seen so far use the training data to estimate a fixed set of parameters $\mathbf{w}$. That defines our hypothesis $h(x)$, and at that point, we can throw away the training data, because they are all summarised by $\mathbf{w}$. A learning model that summarises data with a set of parameters of fixed size (independent of the number of training examples) is called a *parametric model*. A *nonparametric model*, instead, is one that cannot be characterized by a limited set of parameters. For example, suppose that each hypothesis we generate retains within itself all of the training examples and uses all of them to predict the next example. Such a hypothesis family would be nonparametric because the effective number of parameters is unbounded; it grows with the number of examples. This approach is called *instance-based* learning or *memory-based* learning (Russel, 2007). This kind of learning involves storing the entire training set in order to make predictions for future data points. They typically require a metric to be defined that measures the similarity of any two vectors in input space, and are generally fast to 'train' but slow at making predictions for test data points.

4.3.1 Kernel Methods

Many linear parametric models can be re-cast into an equivalent *dual representation* in which the predictions are also based on linear combinations of a *kernel function* evaluated at the training data points (Bishop, 2006). For models which are based on a fixed nonlinear feature space mapping $\boldsymbol{\phi}(x)$, the kernel function is given by the relation:

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}'). \tag{4.21}$$

The kernel is a symmetric function of its arguments so that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$. The simplest example of a kernel function is obtained by considering the identity mapping for the feature space so that $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$, in which

case $k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T\mathbf{x}'$. We shall refer to this as the *linear kernel*. The concept of a kernel formulated as an inner product in a feature space allows us to build interesting extensions of many well-known algorithms by making use of the *kernel trick*. The idea is that, if we have an algorithm formulated in such a way that the input vector $\mathbf{x}$ enters only in the form of scalar products, then we can replace that scalar product with some other choice of kernel. There are numerous forms of kernel functions in common use. Many have the property of being a function only of the difference between the arguments, so that $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$, which are known as *stationary kernels* because they are invariant to translations in input space. A further specialization involves *homogeneous kernels*, also known as *radial basis functions*, which depend only on the magnitude of the distance (typically Euclidean) between the arguments so that $k(\mathbf{x}, \mathbf{x}') = k(\|\mathbf{x} - \mathbf{x}'\|)$.

4.3.2 Sparse Kernel Machine

Consider kernel-based algorithms that have sparse solutions, so that predictions for new inputs depend only on the kernel function evaluated at a subset of the training data points. One of the most popular kernel-based algorithms is *support vector machine* (SVM), which became popular in some years ago for solving problems in classification, regression, and novelty detection. An important property of support vector machines is that the determination of the model parameters corresponds to a convex optimization problem, and so any local solution is also a global optimum (Bishop, 2006).

The two-class classification problem using linear models of the form

$$y(\mathbf{x}) = \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}) + b, \tag{4.22}$$

where $\boldsymbol{\phi}(\mathbf{x})$ denotes a fixed feature-space transformation, with bias parameter b explicit. The training data set comprises N input vectors $x_1, \ldots, x_N$, with corresponding target values $t_1, \ldots, t_N$ where $t_n \in \{-1, 1\}$, and new data points x are classified according to the sign of $y(\mathbf{x})$. We assume that the training data set is linearly separable in feature space, so that by definition there exists at least one choice of the parameters $\mathbf{w}$ and b such that the function satisfies $y(\mathbf{x}_n) > 0$ for points having $t_n = +1$ and $y(x_n) < 0$ for points having $t_n = -1$, so that $t_n y(\mathbf{x}_n) > 0$ for all training data points. There may exist many such solutions that separate the classes correctly. The support vector machine approaches this problem through the concept of the *margin*, which is defined to be the smallest distance between the decision boundary and any of the samples. In support vector machines, the decision boundary is chosen to be the one for which the margin is maximized. The

margin is given by the perpendicular distance to the closest point x_n from the data set, and we wish to optimize the parameters w and b in order to maximize this distance. Thus the maximum margin solution is found by solving

$$\mathbf{w}^* = \arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n [t_n(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n) + b)] \right\}. \tag{4.23}$$

Direct solution of this optimization problem would be very complex, and so we shall convert it into an equivalent problem that is much easier to solve. The optimization problem then simply requires that we maximize $\|\mathbf{w}\|^{-1}$, which is equivalent to minimizing $\|\mathbf{w}\|^2$, and so we have to solve the optimization problem

$$\begin{aligned} &\arg\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 \\ &\textit{subject to } t_n(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x_n}) + b) \geq 1,\ n \in \{1, \ldots, N\}. \end{aligned} \tag{4.24}$$

This is an example of a quadratic programming problem in which we are trying to minimize a quadratic function subject to a set of linear inequality constraints.

In order to solve this constrained optimization problem, we introduce Lagrange multipliers $a_n \geq 0$, with one multiplier and for each of the constraints, giving the Lagrangian function:

$$L(\mathbf{w}, b, a) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n \big\{ t_n(\mathbf{w}^T \phi(\mathbf{x_n}) + b) - 1 \big\}, \tag{4.25}$$

where $a = (a_1, \ldots, a_N)^T$. Note the minus sign in front of the Lagrange multiplier term, because we are minimizing with respect to w and b, and maximizing with respect to a. Setting the derivatives of $L(\mathbf{w}, b, a)$ with respect to w and b equal to zero, we obtain the following two conditions:

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n), \tag{4.26}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{4.27}$$

Eliminating w and b from $L(\mathbf{w}, b, a)$ using these conditions then gives the *dual representation* of the maximum margin problem in which we maximize:

$$\tilde{L}(a) = \sum_{n=1}^{N} -\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{M} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m), \tag{4.28}$$

subject to the constraints

$$a_n \geq 0, \ n = 1, \ldots, N,$$
$$\sum_{n=1}^{N} a_n t_n = 0. \tag{4.29}$$

Here the kernel function is defined by $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$. Again, this takes the form of a quadratic programming problem in which we optimize a quadratic function of a subject to a set of inequality constraints. The classification of new points with the train model is

$$y(\mathbf{x}) = sign\left(\sum_{n=1}^{N} \alpha_n t_n k(\mathbf{x}, \mathbf{x}_n) + b \right). \tag{4.30}$$

4.3.3 SVMs for regression

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The *Support Vector Regression* (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because the output is a real number, it becomes challenging to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. However, the main idea is always the same: to minimize error, to individualize the *hyperplane*, which maximizes the margin, keeping in mind that part of the error is tolerated.

4.4 Tree Based Algorithms

Tree-based algorithms are considered to be one of the best supervised learning methods which are mostly used. Tree-based algorithms enhance predictive models with high accuracy, stability and ease of interpretation. They map nonlinear relationships quite well, unlike linear models. They are adaptable to solve any problem (classification or regression) at hand. Methods such as decision trees, random forest, gradient boosting are used popularly in all forms of data science and analytics issues.

4.4.1 Decision Tree

The decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems. It works

for both categorical and continuous input and output variables. We call C the set of classes or labels. The purpose of the classifier is to assign a label $j \in C$ to an input vector of features $\mathbf{x}$. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on the most significant *splitter* in input variables.

The type of decision tree is based on the type of target variable we have. It can be of two kinds:

- **Categorical Variable Decision Tree**: Decision Tree, which has a categorical target variable then it is called the Categorical Variable decision tree. Such as binary target value;
- **Continuous Variable Decision Tree**: Decision Tree, which has a continuous target variable, then it is called Continuous Variable Decision Tree.

A decision tree is a classifier expressed as a recursive partition of the instance space. The decision tree consists of nodes that form a Rooted Tree; namely, it is a Directed Tree with a node called a *root* that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is referred to as an *internal node*. All other nodes are called *leaves* (also known as *terminal nodes* or *decision nodes*) (Rokach and Maimon, 2008). In a decision tree, each internal node splits the instance space into two or more sub-spaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned according to the value of the attribute. In the case of numeric attributes, the condition refers to a range. Each leaf is assigned to one class representing the most appropriate target value.

Figure 4.2 provides an example of a decision tree that predicts whether or not a potential customer will respond to a direct mailing. Internal nodes are represented as circles, whereas leaves are denoted as triangles. Two or more branches may grow out from each internal node. Each node corresponds with a specific characteristic, and the branches correspond with a range of values. These ranges of values must be mutually exclusive and complete. These two properties of *disjointness* and *completeness* are essential since they ensure that each data instance is mapped to one instance. Instances are classified by navigating them from the root of the tree down to a leaf according to the outcome of the tests along the path. We start with the root of a tree; we consider the characteristic that corresponds to the root, and then we define to which branch the observed value of the given characteristic corresponds.

Then, we consider the node in which the given branch appears. We repeat the same operations for this node until we reach a leaf. Each node is labeled with the attribute it tests, and its branches are labeled with its corresponding values. In the case of numeric attributes, decision trees can be geometrically interpreted as a collection of *hyperplanes*, each orthogonal to one of the axes.

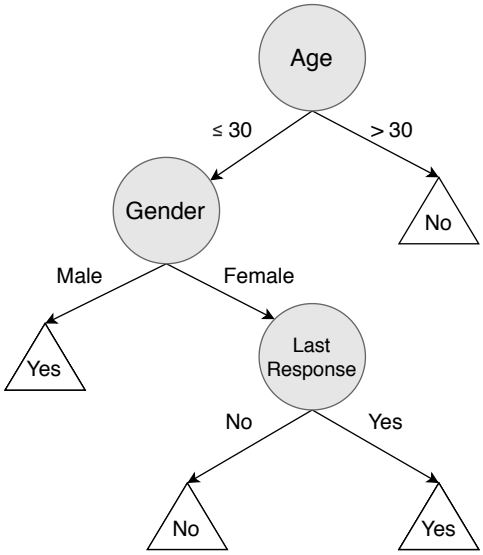


Figure 4.2: Decision tree example on direct mailing response.

A decision tree divides the space into axis-parallel boxes and associates each box with the most frequent label in it. It begins by finding the best horizontal split and the best vertical split (best in the sense of yielding the lowest *misclassification* rate). Therefore, by adopting a greedy strategy, we keep the split that gives the lowest error and move on to develop the tree's branches. The procedure is recursively applied to each of the resulting branches. Complicated decision trees might have a limited generalization capability, for example: although it seems to classify all training instances correctly, it fails to do so in new and unseen instances. In some cases, we should prefer simple trees over complicated ones even if the latter seems to outperform the former in the training set.

Regression models map the input space into a real-value domain. For instance, a regressor can predict the demand for a specific product given its characteristics. Formally, the goal is to examine $y|\,\mathbf{x}$ for a response y and a set of predictors $\mathbf{x}$. The basic idea is to combine decision trees and linear regression to forecast numerical target attributes based on a set of input attributes. These methods perform induction utilizing an efficient recursive partitioning algorithm. The choice of the best split at each node of the tree

is usually guided by a *least squares error* criterion.

4.4.2 Decision Forests

The main idea of an ensemble methodology is to combine a set of models (often called *weak learners*), each of which solves the same original task, in order to obtain a better composite global model, with more accurate and reliable estimates or decisions than can be obtained from using a single model (Rokach and Maimon, 2008).

Definition 2 (Strong Learner)**.** *A strong learner is an inducer that is given a training set consisting of labeled data and produces a classifier which can be arbitrarily accurate.*

Definition 3 (Weak Learner)**.** *A weak learner produces a classifier which is only slightly more accurate than random classification.*

Most of the time, a weak learner model perform not so well by themselves either because it has a high bias (low degree of freedom models, for example) or because it has too much variance to be robust (high degree of freedom models, for example). Then, the idea of ensemble methods is to try reducing bias and or variance of such weak learners by combining several of them in order to create a keen learner (or ensemble model) that achieves better performances. One crucial point is that our choice of weak learners should be coherent with the way we aggregate these models. If we choose base models with low bias but high variance, it should be with an aggregating method that tends to reduce variance whereas if we choose base models with low variance but high bias, it should be with an aggregating method that tends to reduce bias. We can mention two major kinds of meta-algorithms that aims at combining weak learners:

- **bagging**: that often considers homogeneous weak learners learn them *independently* from each other in parallel and combines them following some kind of deterministic averaging process. Bagging will mainly focus on getting an ensemble model with less variance than its components. This approach is based on *bootstrapping*. This statistical technique consists in generating samples of size B (called bootstrap samples) from an initial data set of size N by randomly drawing with replacement B observations. First, we create multiple bootstrap samples so that each new bootstrap sample will act as another independent data set drawn from the true distribution. Then, we can fit a weak learner for each of these samples and finally aggregate them such that

we kind of "average" their outputs and, so, obtain an ensemble model with less variance than its components;

- **boosting**: that often considers homogeneous weak learners learn them *sequentially* in a very adaptive way (a base model depends on the previous ones) and combines them following a deterministic strategy. Boosting will mainly try to produce a strong model less biased than their components (even if variance can also be reduced). Each model in the sequence is fitted, giving more importance to observations in the data set that were badly handled by the previous models in the sequence. Intuitively, each new model focuses its efforts on the most difficult observations to fit up to now, so that we obtain, at the end of the process, a strong learner with lower bias. Boosting, like bagging, can be used for regression as well as for classification problems. *Adaptive Boosting*, *Gradient Boosting* (Friedman, 2001) and *Newton Boosting* (Mason et al., 2000) are three implementation of boosting algorithms. Both of theme fits a weak classifier to weighted versions of the data iteratively. At each iteration, the data is reweighted such that misclassified data points receive larger weights than others. The resulting model can be written as follows:

$$\hat{f}(x) \equiv \sum_{m=1}^{M} \hat{\theta}_m \hat{c}_m(x), \tag{4.31}$$

where $\hat{c}_m(x) \in \{-1, 1\}$ are the weak classifiers and clear classifications are given by $\hat{c}_m(x) = sign(\hat{f}(x))$ (Freund et al., 1996).

4.4.3 Random Forest

Random forests are an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees (Ho, 1995). The random forest approach is a bagging method where deep trees, fitted on bootstrap samples, are combined to produce an output with lower variance. However, random forests also use another trick to make the multiple fitted trees a bit less correlated with each other: when growing each tree, instead of only sampling over the observations in the data set to generate a bootstrap sample, we also sample over features and keep only a *random* subset of them to build the tree.

Random forests use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features.

This process is sometimes called "*feature bagging*". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are robust predictors for the response variable (target output), these features will be selected in many of the trees, causing them to become correlated.

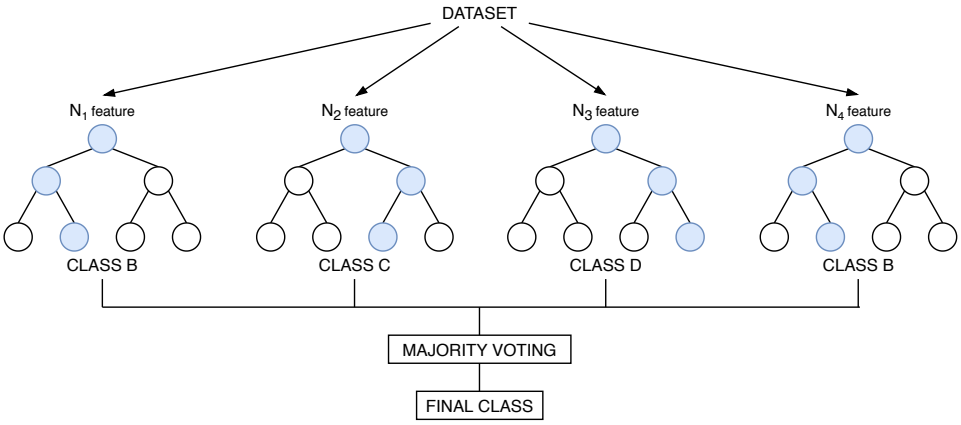


Figure 4.3: Example of Random Forest classifier.

4.4.4 Extra Trees

Adding one further step of randomization yields extremely randomized trees, or Extra Trees. While similar to ordinary random forests in that they are an ensemble of individual trees, there are two main differences: first, each tree is trained using the whole learning sample (rather than a bootstrap sample), and second, the top-down splitting in the tree learner is randomized. Instead of computing the locally optimal cut-point for each feature under consideration (based on, for example, Gini impurity), a random cut-point is selected.[1] This value is selected from a uniform distribution within the feature's empirical range (in the tree's training set). Then, of all the randomly generated splits, the split that yields the highest score is chosen to split the node (Geurts et al., 2006).

[1]Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. The Gini impurity can be computed by summing the probability p_i of an item with label i being chosen times the probability $\sum_{k \neq i} p_k = 1 - p_i$ of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

4.4.5 eXtreme Gradient Boosting

eXtreme Gradient Boosting or XGBoost is an optimized distributed gradient boosting open-source library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM or MART) that solve many data science problems in a fast and accurate way.[2] Nielsen (2016) in his work details how weight function affects the determination of the fit and shows that the tree boosting can be seen to take the *bias-variance trade-off* directly into consideration during fitting. The neighborhoods are kept as large as possible in order to avoid increasing variance unnecessarily and only made smaller when complex structure seems apparent. While coming to a high dimensional problem, tree boosting "beats" the *curse of dimensionality* by not relying on any distance metric. Also, the similarity between data points is learned from the data through adaptive adjustment of neighborhoods. This makes the model immune to the curse of dimensionality. Also, more deep-rooted trees help to capture the interaction of the features. Thus there will be no need to search for appropriate transformations. Thus, with the benefits of boosting tree models, for example, adaptively determined neighborhoods, MART and XGBoost, in general, should make a better fit than other methods. They can perform automatic feature selection and capture high-order interactions without breaking down. By comparing MART and XGBoost, while MART does only set an equal number of terminal nodes across all trees, XGBoost set a T_{max} and regularisation parameter to make the tree deeper while still keeping the variance lower. The Newton boosting used by XGBoost is likely to learn better structures compared to MART's gradient boosting. XGBoost also includes an extra randomization parameter, for example, column subsampling, this helps to reduce the correlation of each tree even further.

4.5 Model Evaluation

The "No Free Lunch" theorem states that there is no one model that works best for every problem. (Wolpert and Macready, 1997) The assumptions of a great model for one problem may not hold for another problem, so it is common in machine learning to try multiple models and find one that works best for a particular problem. This is especially true in supervised learning; *validation* or *cross-validation* is commonly used to assess the predictive accuracies of multiple models of varying complexity to find the best model.

[2] https://xgboost.ai/about.

Multiple algorithms could also train a model that works well; for example, linear regression could be trained by the normal equations or by gradient descent. In this section, we introduce which techniques to use to evaluate different prediction models. While training a model is a crucial step, how the model generalizes on unseen data is an equally important aspect that should be considered in every machine learning pipeline. We need to know whether it works and, consequently, if we can trust its predictions.

4.5.1 Cross-Validation

Cross-validation is a technique that involves partitioning the original observation data set into a training set, used to train the model, and an independent set used to evaluate the analysis (Kohavi et al., 1995). The most common cross-validation technique is *k-fold cross-validation*, where the original data set is partitioned into k equal size subsamples, called folds (Bishop, 2006). The k is a user-specified number, usually with 5 or 10 as its preferred value. Each k-th times, one of the k subsets is used as the test set/validation set, and the other $k-1$ subsets are put together to form a training set. The error estimation is averaged over all k trials to get the total effectiveness of our model. Formally, randomly divide training data into k equal parts: $D_1, \ldots, D_k$, for each part i learn model $y_{D \setminus D_i}$ using data points not in D_i. Estimate the error of $y_{D \setminus D_i}$ on a validation set D_i:

$$L_{D_i} = \frac{k}{N} \sum_{(\mathbf{x}_n, t_n) \in D_i} (t_n - y_{D \setminus D_i}(\mathbf{x}_n))^2. \tag{4.32}$$

k-fold cross validation error is average over data splits

$$L_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} L_{D_i}. \tag{4.33}$$

As can be seen, every data point gets to be in a test set exactly once and gets to be in a training set $k-1$ times. This significantly reduces *bias*, as we are using most of the data for fitting, and it also significantly reduces *variance*, as most of the data is also being used in the test set. Interchanging the training and test sets also adds to the effectiveness of this method.

4.5.2 Evaluation Metrics

Model evaluation metrics are required to quantify model performance. The choice of evaluation metrics depends on a given machine learning task (such

as classification, regression, ranking, clustering, topic modeling, among others). Some metrics, such as precision-recall, are useful for multiple tasks. Supervised learning tasks such as classification and regression constitutes a majority of machine learning applications. The basic classification or regression pipeline works as follows:

- we start by some initial configuration of the model and predict the output based on some input;
- the predicted value is then compared with the target, and the measure of our model performance is taken;
- then the various parameters of the model are adjusted iteratively inorder to reach the optimal value of the *performance metric*.

Classification Metrics

Given an input vector $\mathbf{x}$, the purpose of the classification is to assign it to one of the discrete classes C_k where $k \in \{1, \ldots, K\}$ (Bishop, 2006). Generally, we assume that classes are disjointed in such a way that each input is assigned to one and only one class. The space of the input variables is, therefore divided into decision-making regions whose edges are called decision-making surfaces. When we talk about linear models for classification, we mean that *decision surfaces* are linear functions of input $\mathbf{x}$, they are $(D-1)$-dimensional *hyperplanes* defined within the D-dimensional space of the inputs. The other way around, nonlinear models create decision surfaces given by nonlinear functions of the inputs. The performances of a classifier are measured using the *confusion matrix* and the percentage of inputs correctly classified.

Figure 4.4 shows the confusion matrix of a binary classifier (two possible classes $1 = True$, $0 = False$):

- true Positive (TP): number of positive class data classified correctly;
- true Negative (TN): number of negative class data classified correctly;
- false Negative (FN): number of positive class data wrongly classified as negative class;
- false Positive (FP): number of negative class data wrongly classified as positive class.

The most commonly used indicators to measure classifiers performance are the following:

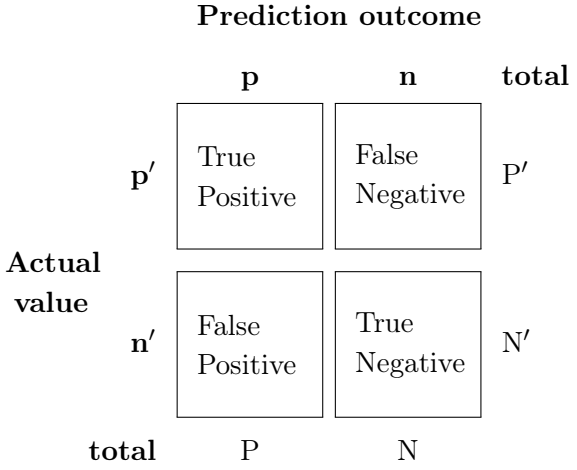


Figure 4.4: Confusion matrix.

- **Accuracy**: the number of correct predictions made by the model over all kinds of predictions made.

$$\begin{aligned} accuracy &= \frac{number\ of\ correctly\ classified\ predictions}{total\ number\ of\ predictions} \\ &= \frac{TP+TN}{TP+FN+FP+TN}. \end{aligned} \tag{4.34}$$

  Accuracy is a good measure when the target variable classes in the data are nearly balanced;

- **Precision**: is the fraction of relevant instances among the retrieved instances.

$$precision = \frac{TP}{TP+FP}. \tag{4.35}$$

  Precision focuses on minimising False positives;

- **Recall**: is the fraction of the total amount of relevant instances that were retrieved.

$$recall = \frac{TP}{TP+FN}. \tag{4.36}$$

  Recall focuses more on minimising False Negatives;

- **F1 score**: the harmonic mean of the recall and Precision.

$$f1\_score = 2\frac{Precision \cdot Recall}{Precision + Recall}. \tag{4.37}$$

Regression Metrics

Regression task is the prediction of the state of an outcome variable at a particular time point with the help of other correlated independent variables (Bishop, 2006). The regression task, unlike the classification task, outputs continuous value within a given range. The most commonly used indicators to measure regressors performance are the following:

- **Mean Squared Error** (MSE): is the average of the squared difference between the target value and the value predicted by the regression model. As it squares the differences, it penalizes even a small error which leads to over-estimation of how bad the model is. It is preferred more than other metrics because it is differentiable and hence can be optimized better. It if defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2, \tag{4.38}$$

  where n is the number of predictions, $\mathbf{Y}_i$ is the vector of observed values of the variable being predicted and $\hat{\mathbf{Y}}$ is the predicted values;

- **Root Mean Squared Error** (RMSE) : is the square root of the averaged squared difference between the target value and the value predicted by the model. It is preferred more in some cases because the errors are first squared before averaging, which poses a high penalty on large errors. This implies that RMSE is useful when large errors are undesired. It if defined as follows:

$$RMSE = \sqrt{MSE} \tag{4.39}$$

$$= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2}, \tag{4.40}$$

  where n is the number of predictions, $\mathbf{Y}_i$ is the vector of observed values of the variable being predicted and $\hat{\mathbf{Y}}$ is the predicted values;

- **Mean Absolute Error** (MAE): is the absolute difference between the target value and the value predicted by the model. The MAE is more robust to *outliers* and does not penalise the errors as extremely as MSE. MAE is a linear score which means all the individual differences are weighted equally. It if defined as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\mathbf{Y}_i - \hat{\mathbf{Y}}_i|, \tag{4.41}$$

where n is the number of predictions, $\mathbf{Y}_i$ is the vector of observed values of the variable being predicted and $\hat{\mathbf{Y}}$ is the predicted values.

Chapter 5

Proposed Algorithms

The following section introduces the techniques and methodologies coming from the machine learning field, which have been developed and used to handle the challenges associated with pricing long-term room rents. Data analytics using machine learning relies on an established suite of events, also known as the *data analytics pipeline* (L'heureux et al., 2017). Figure 5.1 shows a representation of the pipeline based on the work of Labrinidis and Jagadish (2012), along with the three types of manipulations:

- *Data manipulations*: the first manipulations to be attempted in an effort to apply machine learning to a dataset is to try to modify the data in order to select only the relevant information. This modification takes place in the data pre-processing stage of the pipeline. Two intuitive data manipulations for learning are *dimensionality reduction* and *instance selection*. Additionally, *data clearing* is another important aspect of data manipulations, concerning to pre-processing steps such, as noise and outlier removal;

- *Processing manipulations*: to improve machine learning performance, processing manipulations focus on modifying how data are processed and stored. Here, the term storage refers not only to physical storage on a permanent medium but also to how data are represented in memory. Processing manipulations can happen during three phases of the data analytics pipeline: *data transformation*, *data storage*, and *data analysis*;

- *Algorithm manipulations*: include approaches that modify existing algorithms, with or without applying new paradigms. Many algorithms are used to accomplish different business goals. The better features are used, the better the predictive power is. After cleaning the data and

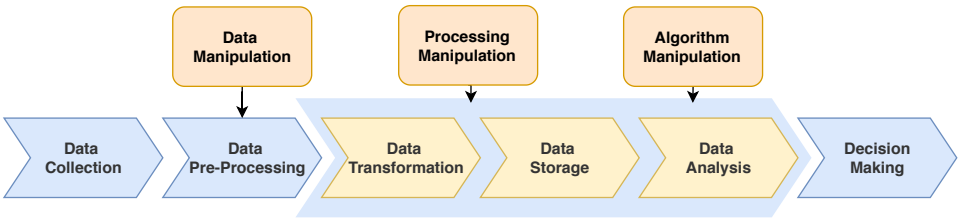


Figure 5.1: Data analysis pipeline.

finding what features are most important, using a model as a predictive tool enhance the business *decision making*.

In this chapter, we mainly focus on giving a theoretical explanation and justification of the technique used for data manipulation.

5.1 Pre-processing Techniques

Data gathered in datasets can present multiple forms and come from many different sources. Data directly extracted from relational databases or obtained from the real world is completely raw: it has not been transformed, cleansed or changed at all. Therefore, it may contain errors due to wrong data entry procedures or missing data, or inconsistencies due to ill-handled merging data processes. Many pre-processing techniques have been designed to overcome the problems present in such real-world datasets and to obtain a final, reliable and accurate dataset to apply a machine learning technique later (Zhang et al., 2003). Gathering all the data elements together is not an easy task when the examples come from different sources, and they have to be merged in a single dataset. Integrating data from different databases is usually called data integration. If some attributes are calculated from the others, the datasets will present a large size, but the information contained will not scale accordingly: detecting and eliminating *redundant attributes* is needed. Having a consistent dataset without measurable inconsistencies does not mean that the data is clean. Errors like *missing values* or uncontrolled *noise* may be still present. A *data cleaning* step is usually needed to filter or correct the wrong data. Otherwise, the knowledge extracted by a machine learning algorithms will be barely accurate. Ending up with a consistent and almost error-free dataset does not mean that the data is in the best form for an algorithm. Some algorithms work much better with normalized attribute values. Others are not able to work with nominal valued attributes or benefit from small transformations in the data. Data *normalization* and data *transformation* techniques have been designed to adapt a dataset to the

needs of the machine learning algorithm that will be applied afterward. Note that eliminating redundant attributes and inconsistencies may still yield a large dataset that will slow down the process. The use of data *reduction* techniques to transform the dataset are quite useful, as they can reduce the number of attributes or instances while maintaining as much information as possible. To sum up, real-world data is usually incomplete, unclean and inconsistent. Therefore, data pre-processing techniques are needed to improve the accuracy and efficiency of the subsequent learning technique used (García et al., 2015).

The rest of the chapter further describes the techniques used to perform the preparation of a generic dataset to replicate the analysis we do on DoveVivo's data. Formally, given a set X of rooms features and a target t denoting its listing price, which transformation and techniques should we use to have a ready-to-use dataset for prediction?

5.2 Data Cleaning

Real-world data tend to be incomplete, noisy, and inconsistent. Data Cleaning routines attempt to fill in *Missing Values* (MV), smooth out noise while identifying *outliers*, and correct *inconsistencies* in the data. Data can be *noisy*, having incorrect attribute values. Owing to the following, the data collection instruments used may be fault. Maybe human or computer errors occurred at data entry. Errors in data transmission can also occur. "Dirty" data can confuse the mining procedure. Although most mining routines have some procedures, they deal with incomplete or noisy data, which are not always robust. Therefore, a useful Data Pre-processing step is to run the data through some Data Cleaning routines.[1]

5.2.1 Dealing with Missing Values

Many existing, industrial and research datasets contain missing values in their attribute values. Intuitively an MV is just a value for an attribute that was not introduced or was lost in the recording process. There are various reasons for their existence, such as manual data entry procedures, equipment errors and incorrect measurements. The simplest way of dealing with MVs is to discard the examples that contain them. However, this method is practical only when the data contains a relatively small number of examples with MVs, and when analysis of the complete examples will not

[1] https://medium.com/easyread/basics-of-data-pre-processing-71c314bc7188

lead to serious bias during the inference (Little and Rubin, 2019). MVs make performing data analysis difficult. There are several types of missing values:

- **Missing Completely At Random** (MCAR): the mechanism that causes the missing data does not depend on both the data observed or the missing data;
- **Missing At Random** (MAR): the mechanism that causes the lack of the data depends on the observed data but not on the missing data;
- **Not Missing At Random** (NMAR): the mechanism that causes the missing data depends on the missing data itself.

The management of missing values can be carried out with different techniques (Graham et al., 2012) and depends strongly on the category to which they belong. The most straightforward and fastest computational approach is to discard all rows containing a missing value. Instead, if the data is not *MCAR*, this method could introduce bias in the final estimate. There is also a risk of excessively reducing the number of examples in the dataset. A computationally more demanding approach is to replace the missing values of a specific attribute with information obtained from the dataset itself, for example, with the mean, the median or with the results predicted by a regression having as output the attribute considered. It is also possible to keep track of the replaced values by reporting a column of dummy variables that indicate whether the value of the given attribute was missing for a particular example. This technique reduces variability and cannot be used for *NMAR*s. Some algorithms autonomously manage missing values according to predefined criteria. To treat *NMAR*s it is necessary to go back to the data source to get more information. The techniques adopted for this type of data are treated in depth in (Graham et al., 2012).

5.2.2 Dealing with Outliers

An *outlier* is a data point that is distant from other similar points. They may be due to variability in the measurement or may indicate experimental errors. If possible, outliers should be excluded from the dataset. However, detecting that anomalous instances might be very difficult, and is not always possible. Having outliers can produce several negative consequences in machine learning problems. Several approaches have been studied in the literature to deal with noisy data. Among them, the most important are:

- *Robust learners*: these are techniques characterized by being less influenced by noisy data. Examples of robust learners are *Support Vec-*

tor Machine and *C4.5* (Quinlan, 2014) algorithm. *C4.5* uses pruning strategies to reduce the chances to reduce the possibility that trees overfit to noise in the training data. However, if the noise level is relatively high, even a robust learner may have poor performances;

- *Data polishing methods*: they aim to correct noisy instances before training a learner. This option is only viable when datasets are small because it is generally time-consuming;

- *Noise filters* (Khoshgoftaar and Rebours, 2007): identify noisy instances which can be eliminated from the training data. These are used with many learners that are sensitive to noisy data and require data pre-processing to address the problem.

A practical way to recognize the outliers of a particular dataset is to visually detect them plotting the data.

Definition 4. *In descriptive statistics, a box plot is a method for graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending horizontally from the boxes (whiskers) indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram. Outliers are plotted as individual points.*

Figure 5.2 is an example of a boxplot showing the distribution of the rooms square footage. Outliers are plotted as diamonds in the plot, and the population is grouped and displayed as a box. This method allows us to eliminate automatically discarding all the data over a certain quantile, indicated in the figure as a vertical line. These data are classified as outliers.

5.2.3 Dealing with Inconsistent data and Duplicates

Real-world data is often incomplete, *inconsistent*, and lacking in certain behaviors or trends and is likely to contain many errors. They create variations on data that simply should not exist. An inconsistency problem is when different information is stored under the same variable. For example, the variable denoting the flat's floor stores only one ordinal number, but someone inputs the number in letters. Indeed, since "second" and "2" refer to the same value, they should be represented in the same way.

A dataset may also include data objects which are *duplicates* of one another. It may happen, for example, when the same person submits a form more than once. In most cases, the duplicates are removed not to give that

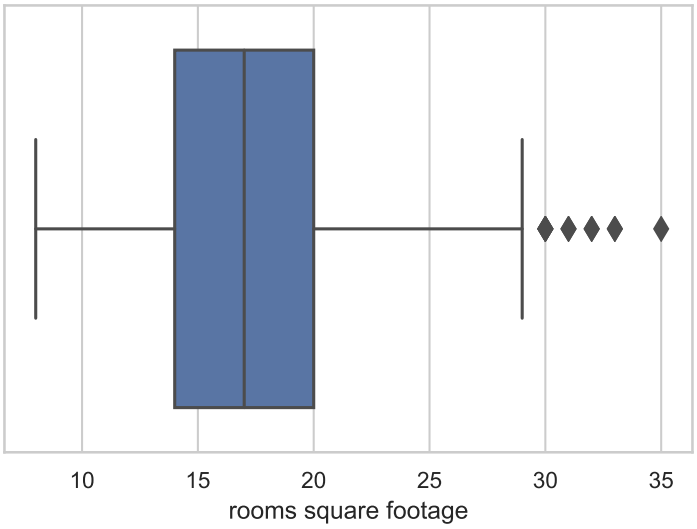


Figure 5.2: Boxplot of the distribution of room square footage.

particular data object an advantage or bias, when running machine learning algorithms.

5.3 Feature Encoding and Data Normalization

Feature Encoding and *Data Normalization* are two methods needed to prepare a raw dataset to be used as input to machine learning algorithms. The former is to transform categorical variables into continuous variables. There are different techniques based on specific needs, such as whether or not the hierarchy between the data is maintained. Data normalization is a method used to impose a range of values that the features must assume in order to make them independent of the unit of measurement used. In data processing, it is also known as Feature Scaling.

5.3.1 Feature Encoding and Discretization

The whole purpose of data pre-processing is to *encode* the data, that means modifying in such a way machine can understand it. We are now focusing on transforming categorical data to numerical data. *Feature encoding* performs transformations on the data such that it can be easily accepted as input for machine learning algorithms while still retaining its original meaning. There are some general criteria and rules which are followed when performing feature encoding (García et al., 2015).

	Name	City	Milan	Rome	Florence
0	Bob	Milan	1	0	0
1	Jhon	Rome	0	1	0
2	Bea	Milan	1	0	0
3	Al	Milan	1	0	0
4	Robe	Florence	0	0	1
5	Jack	Rome	0	1	0
6	Steve	Florence	0	0	1

Table 5.1: Example of One Hot Encoding method.

For Continuous variables:

- *Ordinal*: An order-preserving change of values. This approach, called *Label Encoding*, is straightforward and it involves converting each value in a column to a number. The notion of small, medium and large can be represented equally well with the help of a new function, that is, $\langle NewValue = f(OldValue)\rangle$ for example, $\{0, 1, 2\}$. Label encoding, also induce order/precedence in numbers which means that an algorithm might understand that data has some kind of hierarchy/order $0 < 1 < 2 < \ldots < 6$ and might give more weight to a feature in calculation than another. This means that if we are not interested in preserving the order, this is not the right method;

- *Nominal*: Any one-to-one mapping can be done which retains the meaning. The ordering issue caused by Label Encoding can be addressed in another common alternative approach called *One-Hot Encoding*. In this strategy, each category value is converted into a new column and assigned a 1 or 0 (notation for true/false) value to the column. Although this approach eliminates the hierarchy/order issues but does have the downside of adding more columns to the dataset. An example is presented in Table 5.1.

There are other two methods used for Numerical variables instead that are worth mentioning briefly:

- *Interval*: Simple mathematical transformation like using the equation $\langle NewValue = a\ OldValue + b\rangle$, a and b being constants. For example,

Fahrenheit and Celsius scales, which differ in their Zero values size of a unit, can be encoded in this manner.

- *Ratio*: These variables can be scaled to any particular measures while still maintaining the meaning and ratio of their values. Simple mathematical transformations work in this case as well, like the transformation $\langle NewValue = a\ OldValue + b \rangle$. For, length can be measured in meters or feet; money can be taken in different currencies.

The *discretization* process transforms quantitative data into qualitative data, that is, numerical attributes into discrete or nominal attributes with a finite number of intervals, obtaining a non-overlapping partition of a continuous domain. An association between each interval with a discrete numerical value is then established. In practice, discretization can be viewed as a data reduction method since it maps data from a vast spectrum of numeric values to a significantly reduced subset of discrete values (García et al., 2015). *Equal-Frequency* is an unsupervised discretization method and consists of separating all possible values into N number of bins, each having the same amount of observations. Intervals may correspond to quantile values. This technique can handle outliers and can be combined with categorical encoding.

5.3.2 Min-Max Data Normalization

Many machine learning algorithms search for patterns and trends in the data by comparing the features of data points. However, there is an issue when the features are on drastically different scales. Consider a rooms' dataset. Two potential features might be the square footage of the room, and the total number of bathrooms in the house. A machine learning algorithm could try to predict which price is most suitable for a specific room. However, when the algorithm compares data points, the feature with the larger scale will completely dominate the other. The goal of normalization is to make every data point have the same scale so that each feature is equally important. *Min-max normalization* is one of the most common ways to normalize data. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1. Let X be the random variable corresponding to an attribute and let $x_i, \ldots, x_n$ the random samples extracted from X. With the *range normalization* technique, each value is

scaled concerning the range of X or formally:

$$x'_i = \frac{x_i - \min_i\{x_i\}}{\max_i\{x_i\} - \min_i\{x_i\}}, \tag{5.1}$$

where x'_i is the normalized sample and the denominator of the fraction is the range of the set of samples.

5.4 Data Reduction

Data reduction techniques can be applied to achieve a reduced representation of the dataset; it is much smaller in volume and tries to keep most of the integrity of the original data (Han et al., 2011). In the case of data reduction, the data produced usually maintains the essential structure and integrity of the original data, but the amount of data is downsized (García et al., 2015). Therefore, the goal is to provide the mining process with a mechanism to produce the same (or almost the same) outcome.

One of the well-known problems in Data Mining is the *Curse of Dimensionality*, related with the usual high amount of attributes in data (García et al., 2015). Dimensionality becomes a severe obstacle for the efficiency of most of the machine learning algorithms because of their computational complexity. Indeed, the high dimensionality of the input increases the size of the search space exponentially and also increases the chance to obtain invalid models.

5.4.1 Feature Selection

Definition 5. *Feature Selection (FS) is a process that chooses an optimal subset of features according to a certain criterion.*

The criterion determines the details of evaluating feature subsets. The selection of the criterion must be made according to the purposes of FS. For example, an optimal subset could be a minimal subset that could give the best estimate of predictive accuracy (García et al., 2015).

Generally, the objective of FS is to identify the features in the dataset, which are essential, and discard others as redundant or irrelevant. Since FS reduces the dimensionality of the data, machine learning algorithms, can operate faster and obtain better outcomes. The main reason for this achieved improvement is mainly raised by a smoother and more compact representation of the target concept. Reasons for performing FS may include:

- removing irrelevant data;

- increasing predictive accuracy of learned models;
- improving learning efficiency, such as reducing storage requirements and computational cost;
- reducing the complexity of the resulting model description, improving the understanding of the data and the model.

There are various methodologies and techniques used to subset the feature space and help learning models perform better and efficiently:

- *Filter methods* select features independently of any machine learning algorithms. Features are selected based on their scores in various statistical tests for their correlation with the outcome variable. For instance, using Pearson's Correlation (Biesiada and Duch, 2007). This set of methods do not remove multicollinearity. It is necessary to deal with multicollinearity of features as well before training models;
- *Wrapper methods*, try to use a subset of features and train a model using them. Based on the inferences that we draw from the previous model, we decide to add or remove features from the subset. The problem is essentially reduced to a search problem. These methods are usually computationally costly in terms of time-consuming. Some common examples of wrapper methods are forward feature selection, backward feature elimination, recursive feature elimination. They are defined as follows:
  - *Forward Selection*: Forward selection is an iterative method in which we start with having no feature in the model. In each iteration, we keep adding the feature which best improves our model till an addition of a new variable does not improve the performance of the model. Algorithm 2 provides details for Sequential Forward Feature set Generation. Here, the best feature f is chosen by $FindNext(F)$. f is added into S and removed from F, growing S and shrinking F. There can be two stopping criteria, either when the F set is empty or when S satisfies U. Adopting only the first criterion, we can obtain a ranked list of features;
  - *Backward Elimination*: In backward elimination, we start with all the features and removes the least significant feature at each iteration which improves the performance of the model. We repeat this until no improvement is observed on the removal of features;

  - *Recursive Feature elimination*: It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination;

- *Embedded methods* combine the qualities' of filter and wrapper methods. It is implemented by algorithms that have their built-in feature selection methods. Some of the most popular examples of these methods are LASSO and RIDGE regression which have inbuilt penalization functions to reduce overfitting.

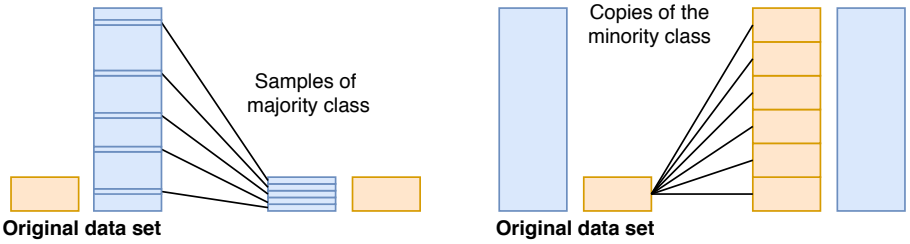
Algorithm 2 `Sequential Forward Feature set Generation - SFG`

```
1: function SFG(F - full set, U - measure)
2:     initialize: S = {}                          ▷ S stores the selected features
3:     repeat
4:         f = FindNext(F)
5:         S = S ∪ {f}
6:         F = F − {f}
7:     until S satisfies U or F = {}
8:     return S
9: end function
```

5.5 Feature Sampling

Imbalanced classifications pose a challenge for predictive modeling (Japkowicz and Stephen, 2002). Most of the machine learning algorithms used for classification were designed with the assumption of an equal number of examples for each class. This results in models that have poor predictive performance, specifically for the minority class. Over and under-sampling methodologies have received significant attention to counter the effect of imbalanced datasets (Chawla, 2009).

A widely adopted technique for dealing with highly unbalanced datasets is called resampling. It consists of removing samples from the majority class (undersampling) and/or adding more examples from the minority class (oversampling). *Undersampling* is the process where you randomly delete some of the observations from the majority class to match the numbers with the minority class. *Oversampling* is the process that samples data for the minority class to match the number of observations in the majority class. Figure 5.3 shows an example of how undersampling and oversampling work.



(a) Undersampling *(b) Oversampling*

Figure 5.3: Example of Undersampling and Oversampling.

The key principle here is that the sampling should be done in such a manner that the sample generated should have approximately the same properties as the original dataset, meaning that the sample is representative of the original distribution. This involves choosing the correct sample size and sampling strategy. *Simple Random Sampling* dictates that there is an equal probability of selecting any particular entity. It has two main variations as well:

- *Sampling without Replacement*: As each item is selected, it is removed from the set of all the objects that form the total dataset;
- *Sampling with Replacement*: Items are not removed from the total dataset after getting selected. This means they can get selected more than once.

Despite the advantage of balancing classes, these techniques also have their weaknesses. The most straightforward implementation of oversampling is to duplicate random records from the minority class, which can cause overfitting. In undersampling, the most straightforward technique involves removing random records from the majority class, which can cause loss of information.

5.6 First Problem Formulation - Data Pipeline

In this paragraph, we describe the general pipeline used to obtain a dataset ready to be applied as input to machine learning algorithms. We want to prepare a dataset to accomplish the regression task of predicting a room price maximizing the revenue. Taking advantage of the data analysis techniques described so far and solving some of the problems that may arise during this pre-processing phase.

We assume to have a dataset of rooms X composed of different features and a price associated which is our target t. The problem we have to solve is to build a model that generalizes the mapping from the features of the rooms to the final price assigned.

5.6.1 Data Cleaning

The first step on every pre-processing pipeline is the data quality assessment. We must follow some steps to clean the data, transform it and to integrate it with additional information. First of all, our work is based on a specific city: Milan, we make this choice because prices vary a lot from one city to another, so it is good to consider only one place at a time. The features to consider are of different nature:

- features that characterize the room, for example, square footage, type of bed (single bed or double bed), type of room (single room or double room), whether or not it has a private bathroom;

- features that represent the apartment in which the room is located, such as, the number of other single or double rooms in the apartment, the floor, the presence or absence of a balcony, air conditioning, washing machine or lift;

- features that represent the area in which the room is located: geographic coordinates, neighborhood or proximity to services such as means of transport.

In order to feed a machine learning algorithm, the data must be clean; this means that they must be free of inconsistencies, missing values and outliers. In the previous paragraphs, we present several techniques to solve these problems; it is important to note:

- duplicate data must be deleted;

- eliminate samples with missing values if these values cannot be somehow derived from other sources or recalculated. It is reasonable to assume that these missing values are *MCAR* or *MAR* and, therefore, we delete the rows that contain them. If for a specific room, the data on the square footage is missing, and this information can not be obtained in any other way, e.g., from another data source, we opted to remove it;

- eliminate features that contain a large number of missing values not attributable to the dataset; this also helps for the problem of *curse of dimensionality*;
- machine learning algorithms take numeric data as input. Therefore, it is necessary to transform all categorical variables into numeric ones. We used the methods mentioned earlier of Feature Decoding:
  - **Label Encoding** for categorical features such as the floor, the type of room (single or double) or the type of bed (single or double);
  - **One-Hot Encoding** for categorical features where we do not want to guarantee the numeric order. For example, to divide the dataset into clusters, we create a binary column for each cluster, assigning a one in the column corresponding to the zone to which the sample belongs;
- from our analysis, information regarding the geographical area in which the house is located is essential. Therefore, it is highly recommended that, among the features of a room, the geographical coordinates, i.e., latitude and longitude, are present.

At this point, it is necessary to filter the data to eliminate all those samples that have been entered incorrectly in the raw dataset. This data can be trivially eliminated by filtering, feature by feature, the values for predefined intervals. For instance, the square footage of a room is commonly from 5 to 50 square meters. All values outside this range are most likely *outliers*. Alternatively, an excellent way to manually inspect them is by defining a threshold quantile using a box plot, as shown in Figure 5.2. All the values outside the leftmost and the rightmost hinge are considered outliers.

5.6.2 Feature Engineering

It is possible to integrate the initial dataset with additional information regarding the position of the house concerning other city's places of interest. Our analysis shows how the introduction of the data about the proximity of the room to public transport may favor the evaluation of a room positively. Another decisive factor from our analysis was to understand how the price fluctuates concerning the proximity to the city center. Therefore, we carried out a transformation of variables from Cartesian coordinates to polar coordinates.

Haversine formula

The distance between two points uses the *haversine* formula to calculate the great-circle distance between two points. That is the shortest distance over the earth's surface, giving an "as-the-crow-flies" *distance* between the points (ignoring any hills they fly over), or formally:

$$\begin{aligned} a &= \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\ \sin^2\left(\frac{\Delta\lambda}{2}\right), \\ c &= 2\arctan(\sqrt{a}, \sqrt{(1-a)}), \\ distance &= Rc, \end{aligned} \tag{5.2}$$

where ϕ is latitude, λ is longitude, R is earth's radius (mean radius = $6.371km$), $\arctan(y, x)$ indicates the angle in radiant between the positive half-axis of the X of a Cartesian plane and a point of coordinates (x, y) lying on it.

Relative Bearing

On the other hand, the angle between a specific room and the city center is calculated using the *relative bearing* which, in the naval context, refers to the angle between the craft's forward direction and the location of another object:

$$\begin{aligned} \Theta = &\arctan(\sin(\Delta\lambda)\cos(\phi_2), \\ &\cos(\phi_1)\sin(\phi_2) - \cos(\phi_1)\cos(\phi_2)\cos(\Delta\lambda)), \end{aligned} \tag{5.3}$$

where ϕ_1, λ_1 is the start point, ϕ_2, λ_2 the end point, and $\Delta\lambda$ is the difference in longitude.

Room style

Another factor that we consider essential to integrate into the dataset is the information on how old the apartment is. This information is not present in the database so we rely on the "age" of the room which can be obtained considering the number of years from today until the starting date of the contract with the owner. Older the room, the more likely it has been renovated before.

Normalization

Finally, having at this point, a clean and balanced dataset, we just have to normalize the data to eliminate any difference in scale and unit of measure

between the features. The Min-Max normalization formula in Equation 5.1 re-scales the values of a feature in a range $[0, 1]$. In this way, each feature is equally important.

5.7 Second Problem Formulation - Data Pipeline

In this section, we see in detail what are the main reasons that lead us to a second formulation of the pricing problem based on a classifier. The latter predicts whether, at the end of a contract, a certain bed will remain vacant for some days or not. Next, we show what features a contract dataset must have for our purpose. The pre-processing phase starts from the reconstruction of the contract history up to the creation of new features to introduce seasonality in the data.

5.7.1 Motivation

What we create is a classifier that takes as input a dataset Z and returns a binary value relative to vacancy. Each sample in the dataset corresponds to a contract. It is composed of the room's features, as in the previous formulation, historical data referring to the contract and the monthly fee assigned to the room from the contract to the time. The classifier's output can take two values:

- 1: there will be days of vacancy between one contract and the next;
- 0: there will be no vacancy days at the end of the current contract.

Our reasoning starts with practical evaluations. A room with recognizable features by users will be more requested than the others; therefore less likely to go through vacant periods Given the strong demand for the rental market in Milan, we are induced to raise the prices of rental properties. However, we wonder how much we can modify it without causing a consequent vacancy period, and we proceed as follows:

1. consider a wide range of prices centered on the current listing price and discretize it, thus obtaining a vector of prices for each bedroom.
2. for each room, we give the classifier the room's characteristics, which are fixed, and, one at a time, the prices of the vector created in the previous point. In this way, we obtain a binary vector that predicts the vacancy for each proposed price;

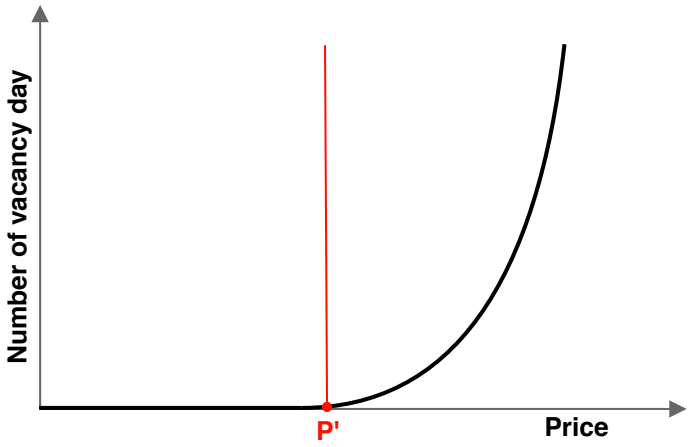


Figure 5.4: Trend of the vacancy period compared to the rental price.

3. observe the binary vector and select the highest price among those that do not give vacancy or among those that, given as input to the classifier, return an output equal to zero.

We expect the classifier to have monotonous increasing behavior or that the vector returned for each room is composed of a sequence of zeros followed by a sequence of ones. Intuitively this trend of the vector means that, by raising the price of a given bedroom, the difficulty in finding a new interested user by the end date of the contract increases. The trend of the duration of the vacancy period concerning the price changes is shown in Figure 5.4: vacancies increase as the price increases. P' represents the maximum price that can be assigned to the room that guarantees not having a vacancy period following the end of the contract. Figure 5.5 shows the monotonous behavior a classifier should have in this scenario as we increase the price of a room: it returns 0 if the price is below P', hence 1 denotes the room will likely have vacancy.

Figure 5.6 shows an example illustrating this methodology. In this example, we have chosen to assign prices included in the range $P \pm 10\%P$, where P is the listing price currently in use. In particular, we discretized this interval into sub-intervals of amplitude 10€, and we have discovered, observing the vector of the outputs of the classifier, that the ideal price P' is 580€.

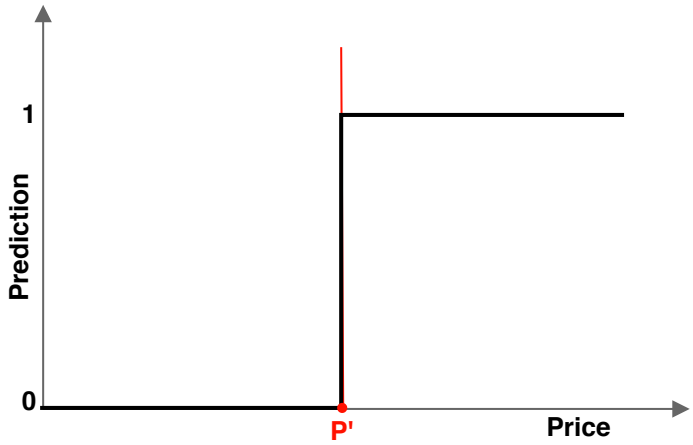


Figure 5.5: Monotonous classifier behaviour compared to the rental price.

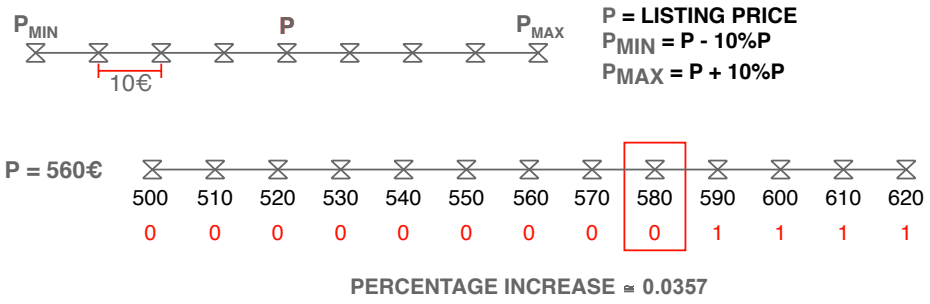


Figure 5.6: Example of methodology that uses the classifier to predict vacancy.

5.7.2 Data Cleaning

In this section, we firstly, describe in detail which characteristics the dataset must-have. Then, we show the pre-processing steps to follow to create a dataset for the classification problem. The features to be considered for this formulation are of different types:

- features concerning the characteristics of the room;
- features concerning the characteristics of the contract, including the listing price of the room at the time of the contract.

As we have already said, each sample of this dataset corresponds to a contract for a rented room. In particular, each contract contains:

- contract start and end date;

- date of the stipulation of the contract which corresponds to the date on which the tenant signs the contract;
- contract cancellation date which corresponds to the date on which the tenant decides to leave the room;
- a binary value relating to the status of the contract: active or inactive;
- the monthly fee for the room;
- a unique identifier for the contract and the room.

The dataset cleaning phase starts with the data assessment. As in the previous formulation, any type of missing and inconsistent data must be adjusted in such a way as not to create problems with the machine learning models we are going to use.

5.7.3 Contract History Reconstruction

A crucial step in guaranteeing the chronological correctness of the data is the reconstruction of the contract history. This procedure is mainly used to detect and possibly correct inconsistencies in the data. Therefore, it must be ensured that the contract history does not contain any type of inconsistency for a specific room. This means that there can not be two different contracts active simultaneously for the same room. Similarly, the end date of a contract cannot happen later than the beginning of the next contract. These inconsistencies are usually due to human input errors in databases. Where possible, it is always better to fix these inconsistencies without eliminating the data. For instance, if two chronologically contiguous contracts are partially overlapping, as shown in Figure 5.7, it is possible to terminate the first contract before the beginning of the next one. On the other hand, if by mistake two or more contracts have the same start and end validity dates, it is better to eliminate all the duplicates.

5.7.4 Feature Engineering

Generally, room rents contracts are not stationary over the year. For instance, a period in which the fees usually increase is close to the beginning of the university semesters, i.e., September and February. For the study of seasonal patterns, we introduced new numerical features. It is possible to describe the seasonal pattern as one sine and one cosine function (Stolwijk

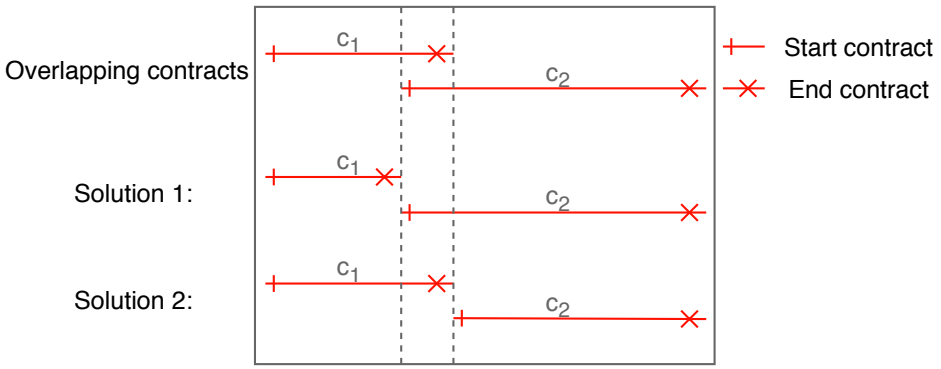


Figure 5.7: Overlapping contracts.

et al., 1999):

$$SinData_i = \sin\left(\frac{2\pi x_i}{365}\right), \tag{5.4}$$

$$CosData_i = \cos\left(\frac{2\pi x_i}{365}\right), \tag{5.5}$$

where i represents the i-th sample, and x_i is the day of the year of the starting date contracts of the i-th contract.

We also introduced:

- the number of years from today to the beginning of the contract with the owner of the room (called passive contract date);
- the year in which the contract started;
- the duration of each contract in days;
- the number of vacancy days between one contract and the next one;
- the percentage increase in monthly fees between one contract and the next one.

5.7.5 Data Filtering

We proceed by filtering the data. We only consider contracts that have a valid contract end date which corresponds to considering only concluded contracts. Finally, we assign to each sample a binary value starting from the number of vacancy days:

- 1: if the number of vacancy days exceeds 10, that amount being significant for business purposes;

- 0: otherwise.

This last feature is the target on which we train our classification model.

5.7.6 Data Sampling for Monotone Classifier

The main characteristic of the model we build is that it has to follow the trend of a monotonous function. This means that as a given value increases (listing price), the target will be increasingly inclined to return a positive value (vacancy). At the same time, if the price drops, the prediction will be more likely to be negative. The idea is that if a contract with certain characteristics and with a price p generates vacancy, then increasing the price will continue to produce vacancy. Taking advantage of the oversampling technique, we sample all the contracts with label one without replacement, as seen in Section 5.5, and assign a new price higher by a small percentage compared to the initial one. In this way, we also increase the number of samples of the dataset.

Data Integration

From our analysis, it is essential to integrate the room data with additional information taken from other sources. An in-depth geographical analysis allows us to identify and cluster rooms with similar particularities. This allows us to build targeted and more specific models for each individual cluster. Taking advantage of the division into neighborhoods of the city already performed in the dataset, we are able to analyze a sub-group of rooms at a time and create an ad hoc models. In the case of data provided by DoveVivo, exploiting a grouping of rooms on a geographical basis allowed us to notice and fix some inconsistencies. In particular, the company assigns a specific area of Milan to each room, and this region was not consistent with the actual position of the room. Other types of information to be integrated into the rooms dataset are the presence of services such as supermarkets, pharmacies and public transport nearby.

Chapter 6

Experiments

In the following chapter, we analyze in detail the data provided by DoveVivo for both problem formulations presented in Section 3.2. After a first qualitative and quantitative analysis of the data, we show how they have been cleaned, transformed, and used. We take advantage of the pre-processing techniques introduced in the previous chapter, detailing what problems we have encountered and how we have overcome them, what information we had and what we integrated to strengthen the model and improve performance. Finally, we show the results we obtain by comparing different learning models, exploiting, therefore, supervised learning techniques introduced in Chapter 4 both for regression and classification. To evaluate the performances achieved by the various models, we use the evaluation metrics presented in Section 4.5.2.

6.1 First Problem Formulation - Experiment

6.1.1 Data Collection and Feature Description

We take the data of the DoveVivo's rooms by querying their data storage systems via remote connection. The data are structured in different tables, so we have to join multiple instances to get a first dataset to analyze. We mainly need four tables:

- **clu_postoletto**: all the room information resides;
- **clu_appartamento**: contains information about the apartment;
- **clu_città**: includes all the information on the city in which the apartment is located;
- **clu_locazione**: includes information on the contracts for each room.

Then, we perform a first filtering of the data directly from the query. We do not use samples where the information is missing. We only examine rooms currently owned by DoveVivo, since there may be rooms in the system that no longer belong to the company. Given the considerable difference in the trend of the real estate market between one city and another, our analysis focus only on the city of Milan. This allows us to have a discrete pool of data following a similar price trend. We only take into consideration the single or double rooms in Milan intended to private audiences (B2C contracts) currently under the responsibility of DoveVivo.

Features Description

After a first screening of the data, we obtain a dataset of 31 features and 2540 samples, where each sample corresponds to a specific room. In this section, we observe in detail the features that comprise the dataset. They are:

- **data_stipula**: is the date on which the last currently active contract for a room is signed. For our analyzes, this field is used to filter data in such a way to eliminate inconsistencies. For example, we check that every stipulate date is before the contract start date;
- **prezzo_listino**: listing price associated with each room which may differ from the contract price. It is a continuous numeric field expressed in euros and will constitute the target for regression models;
- **prezzo_utenze**: price of the utilities of a house, usually the value is around 40-50 euros and must be added to the listing price to obtain the final price;
- **mq_stanza**: continuous value that represents the square footage of each room in square meters;
- **num_singole**: represents the total number of single rooms in the examined apartment;
- **num_doppie**: represents the total number of double rooms in the examined apartment;
- **num_bagni_app**: represents the total number of bathrooms in the examined apartment;
- **tipo_stanza**: this field represents the type of a room. The field can assume two values: "Stanza singola" (single room) and "Stanza doppia" (double room);

- The Table 6.1 lists the 14 binary features that represent the presence or absence of benefits in the home, with the corresponding number of positive occurrences:

Table 6.1: Binary Features.

Field Name	Benefit Name	Positive occurrences
bagno_privato	private restroom	2488
fb_balcone	balcony	804
fb_giardino	garden	5
fb_wifi	Wi-Fi connection	2541
fb_cantina	cellar	41
fb_parcheggio_bici	bicycle park	1502
fb_parcheggio_moto	motorcycle park	4
fb_parcheggio_auto	car park	0
fb_lavatrice	washing machine	2541
fb_portineria	reception	1695
fb_condizionatore	air-conditioner	21
fb_terrazzo	terrace	5
fb_lavastoviglie	dish washer	2386
fb_ascensore	elevator	2013

- **lat_lon**: represent the geographical coordinates of each room. the field has the following format $(latitude, longitude)$;
- **id_zona** / **nome_zona**: they represent the zone to which the room belongs. DoveVivo assigns to each room an area that broadly corresponds to agglomerate some contiguous neighbourhoods of the city of Milan. In total 19 areas cover the entire surface of the municipality;
- **piano**: represents the floor on which the apartment is located inside an apartment building. It is a positive numeric value.
- **tipo_letto**: this data gives us information on the type of bed in a

room. This field can assume three different values: "Singolo" (single bed), "Matrimoniale" (King-size bed), and "Una piazza e mezzo" (Queen-size bed);

- **data_inizio_passivo**: this field represents the date on which the contract involving DoveVivo and the owner of an apartment started. This data is used to indicate how old an apartment is.

6.1.2 Data Cleaning and Data Exploration

Duplicate and Inconsistency Management

Consistency check for the date of the contracts' stipulation: we eliminate all mistakenly entered dates in the database. If there is a date corresponding to a future day, the corresponding sample is discarded. Furthermore, if selecting only the active contracts, two rows are corresponding to the same room, we keep the one with the most recent contract stipulation date.

Handle categorical variables

In the data provided by DoveVivo, there are some categorical variables to be handled. Let us see in detail how each categorical variable was processed.

- **tipo_stanza**: this field assumes two categorical values to indicate whether the sample considered is a single or double room. We apply *Label Encoding* (Section 5.3) to transform the categorical variable into numeric:
  - single room: 1;
  - double room: 0;
- **lat_lon**: this field is composed of two strings divided by a comma: the first member corresponds to the latitude of the room, while the second one corresponds to the longitude. Therefore, we isolate the two values and finally create two new distinct variables in the dataset with the latitude and longitude values;
- **nome_zona**: this field contains 19 unique names to indicate the areas in which the city of Milan is divided by DoveVivo. We apply *One-Hot Encoding* (Section 5.3) to map the categorical attribute into 19 binary variables, each of which describes a specific area;
- **tipo_letto**: Figure 6.1 shows how the values associated with the tipo_letto variable are distributed. Given the deficient number of

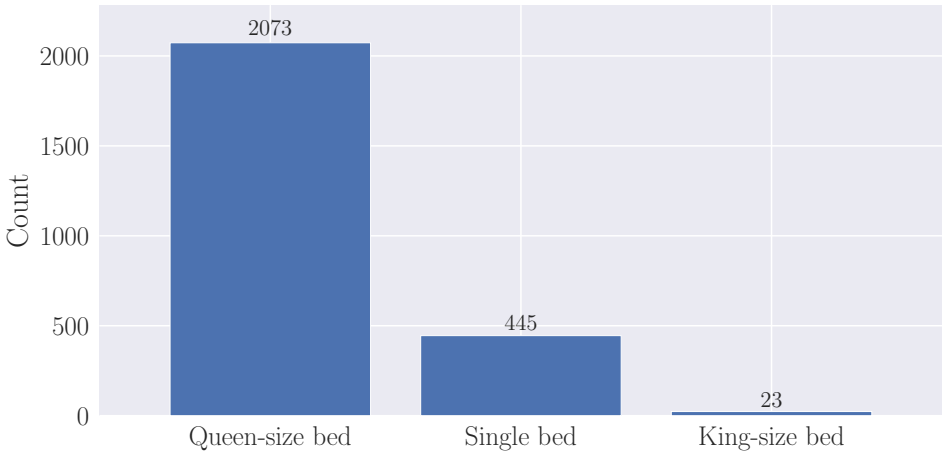


Figure 6.1: Bed type variable distribution.

king-size beds, we decided to consider them together with the queen-size bed. Then apply *Label Encoding* (Section 5.3) to transform the categorical variable into numeric:

  - double bed: 1;
  - single bed: 0;

- **data_inizio_passivo**: we decided to keep this field to indicate how old a room is. We calculate for each room the number of year from today to the passive start date;

- **piano**: in most cases, this variable contains numeric values that indicate the floor of the room. Some values, however, are shown in string format: for example, 'three' to indicate the 3rd floor. In this phase, we have translated natural language into numbers. Instead, all those inconsistent values that do not indicate a positive integer have been eliminated.

Handle data outliers

Outlier is a term used in statistics to define, in a set of observations, an anomalous and aberrant value. Figure 6.2 shows the boxplots of some of the numeric variables that we examine, more specifically listing price, floor, room and apartment square footage. To eliminate these values far from other observations, we have imposed ranges of values within which the sample is considered acceptable. Outliers should be eliminated or, if possible, corrected

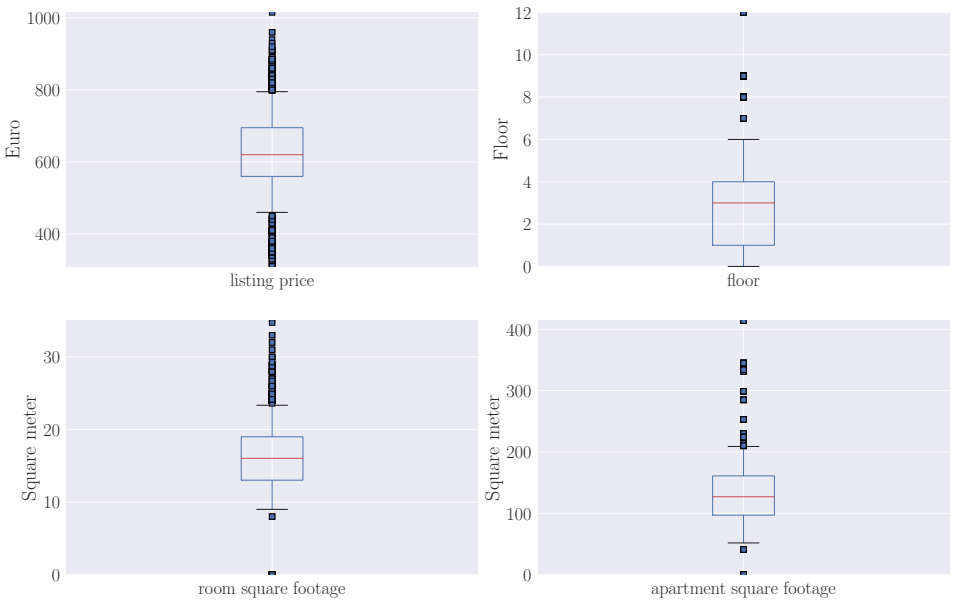


Figure 6.2: Numeric variables boxplots.

if data are incorrectly reported. If, on the other hand, they are data relating to rooms with characteristics that make them very different from all the others, we must identify the corresponding rooms, price them by hand and study them individually. In both cases, the pre-processing phase identifies the outliers. It eliminates them from the dataset so that they are not used for training the model and it exports them in specific files from which they can then be examined separately. The ranges outside which the data have been considered outliers are the following:

- contract price $\in [100, 5000]$;
- price list $\in [100, 5000]$;
- floor $\in [0, 20]$
- square meters room $\in [7, 50]$;
- square meters apartment $\in [30, 700]$.

Numerical Variables Transformation and Feature Engineering

Let us discuss how we deal with the problem from a geographical point of view. A preliminary analysis we conduct shows that the proximity to the center of Milan is directly proportional to the listing price. In Figure 6.3, the

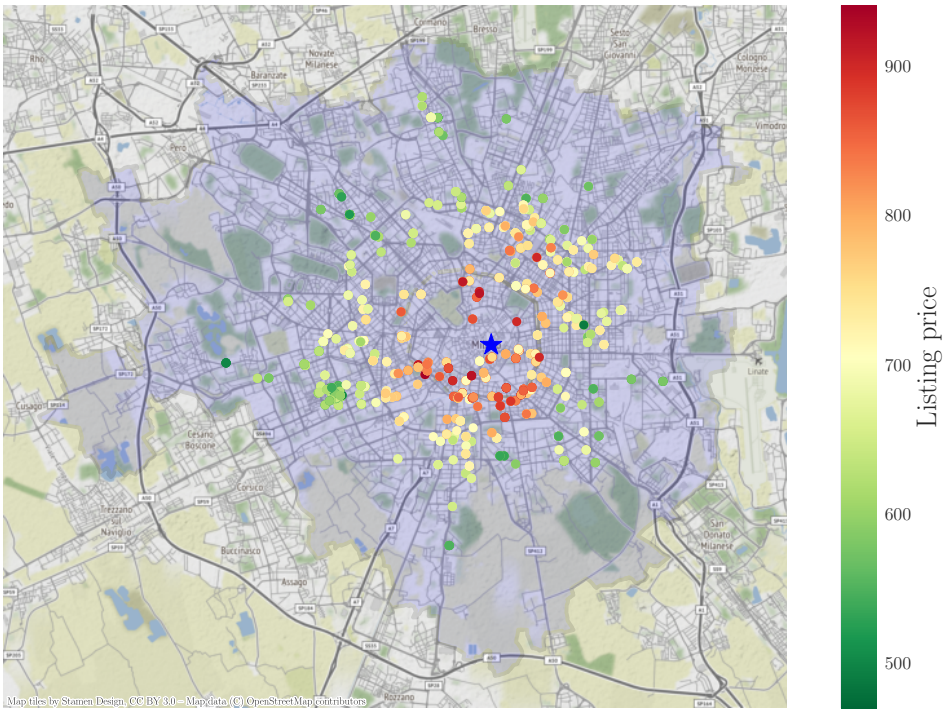


Figure 6.3: DoveVivo's rooms over Milan's surface.

rooms of DoveVivo are displayed on the area of the municipality of Milan. The blue star represents the Duomo of Milan, which is the reference point that we used as the center of the city. We transform the continuous numerical variables of latitude and longitude into a new reference system composed of radius and angle. Specifically, the radius indicates the distance of each apartment from the Duomo of Milan. The angle indicates the inclination with respect to the north. The new coordinates are added to the dataset. The distance between two points on the Earth's surface is calculated using the *Haversine* formula (Equation (5.2)), while the angle is calculated using the *Relative Bearing* formula (Equation (5.3)).

On the website of the municipality of Milan is possible to consult and use a large amount of data concerning various aspects of the city, e.g., environment, energy, health, transport.[1] From a detailed spatial analysis of the areas indicated by DoveVivo for each room, we noticed some inconsistencies between the actual position of the room and the area set by the company. These are the main steps that allow us to fix them:

- we downloaded the coordinates of the vertices of the polygons that

[1] https://dati.comune.milano.it/.

form the districts of Milan. The latter are 89, many more than the areas indicated by DoveVivo;

- we manually agglomerate the districts of Milan to obtain 19 macro-areas such as those declared by the company. Figure 6.4 shows the Milan surface divided into the 19 regions;
- we represent DoveVivo's rooms on the map of Milan given the coordinates;
- using the *convex hull* method, we have traced the smallest convex set on the sheet that encloses all the rooms area by area taking as reference the zone indicated by DoveVivo in the data. It can be easily seen from Figure 6.5 that some rooms are not spatially located in the correct area;
- exploiting the geographical coordinates of the rooms and the vertices of the regions, we recalculate the area to which each room belongs. We modify the label nome_zona of each room with the zone to which it belongs. Figure 6.6 shows the final results after the re-positioning of each room.

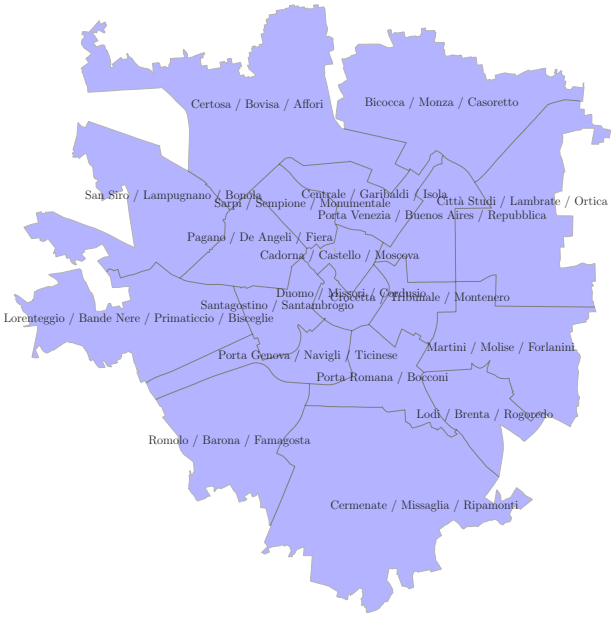


Figure 6.4: DoveVivo's regions, each with its own label.

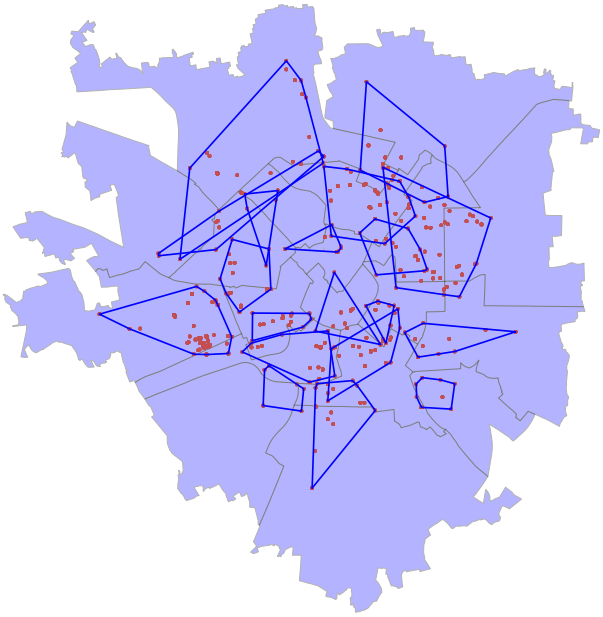


Figure 6.5: DoveVivo's rooms over initial Milan's regions.

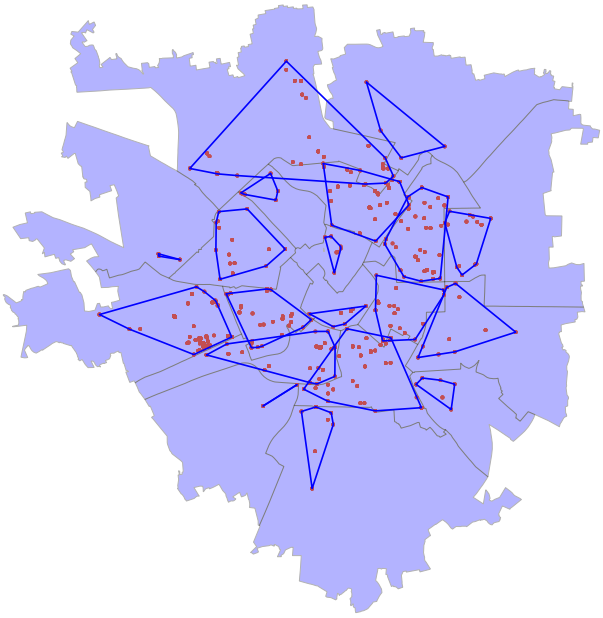


Figure 6.6: DoveVivo's rooms over newly-defined Milan's regions.

In a big city, intuitively, the value of an apartment increases with the proximity to the public means of transport. Therefore, we introduced to the dataset, for each room, the closeness in meters from the nearest metro station. To do this, we use the coordinates of the metro stations provided by the city of Milan. We calculate the distance between each room with all the metro stops, and for each of them, we keep the smaller gap. Distance is calculated using the *Haversine* formula in Equation (5.2).

However, living near a to subway station is not always considered positive, especially in areas more distant from the downtown. This is because the stations are noisy places and sometimes are the center of areas of urban decay. To verify our idea, we decide to discretize the distance in 4 bands, exploiting the *Equal-Frequency discretization* in Equation (5.3.1). Each block contains approximately the same number of rooms, obtaining the following labels:

- $0\ m \leq d_1 < 300\ m$;
- $300\ m \leq d_2 < 650\ m$;
- $650\ m \leq d_3 < 1000\ m$;
- $d_4 \geq 1000\ m$;

where d_i are the 4 ranges of distances. We replace each continuous distance value with the corresponding label. Since the proximity of the stations is not always considered a positive factor, we do not preserve the order for all the four batches by assigning a value from 1 to 4. We use the *One-Hot-Encoding* method (Section 5.1) to create four new binary variables to add to the rooms dataset. In this way, the models assign a different weight to each distance based on what they learn area by area.

The last operations we carry out are the elimination of all irrelevant binary features, that is, containing a number of ones greater than 95% concerning the total, or vice versa with a number of ones less than 5%. This is because their contribution can affect the performance of the final model. Machine Learning algorithms do not consider a feature if almost all, or none, of the samples, have it. Therefore, we eliminate garden, wi fi, cellar, motorcycle parking, car parking, washing machine, air conditioning, terrace and dishwasher. We decide to keep the information on the private bathroom as very relevant to the prediction of the final price. Finally, using the min-max formula in Equation (5.3.2). We normalize all the data in the range $[0, 1]$, in this way, all the variables have the same importance.

Data Visualization

After the pre-processing phase, we obtain a clean dataset with no missing, inconsistent or outlier values. We obtain a matrix consisting of 2398 unique rooms and 43 numeric features. The latter are composed as follows:

- listing price;
- nineteen zones: binary features result of the *One-Hot Encoding* process (Section 5.1);
- nine features that characterize the apartment. Four numeric: number of single and double rooms, number of bathrooms and the number of years from the start of the passive contract. Five binary features: balcony, bike parking, floor, lift and concierge;
- four features that characterize the room. One numerical: square footage. Three binaries: single or double room, private bathroom, type of bed;
- eight features that indicate the geographical position of the room: four binary features to indicate the distance of the nearest metro, distance from the Duomo of Milan, angle, latitude and longitude.

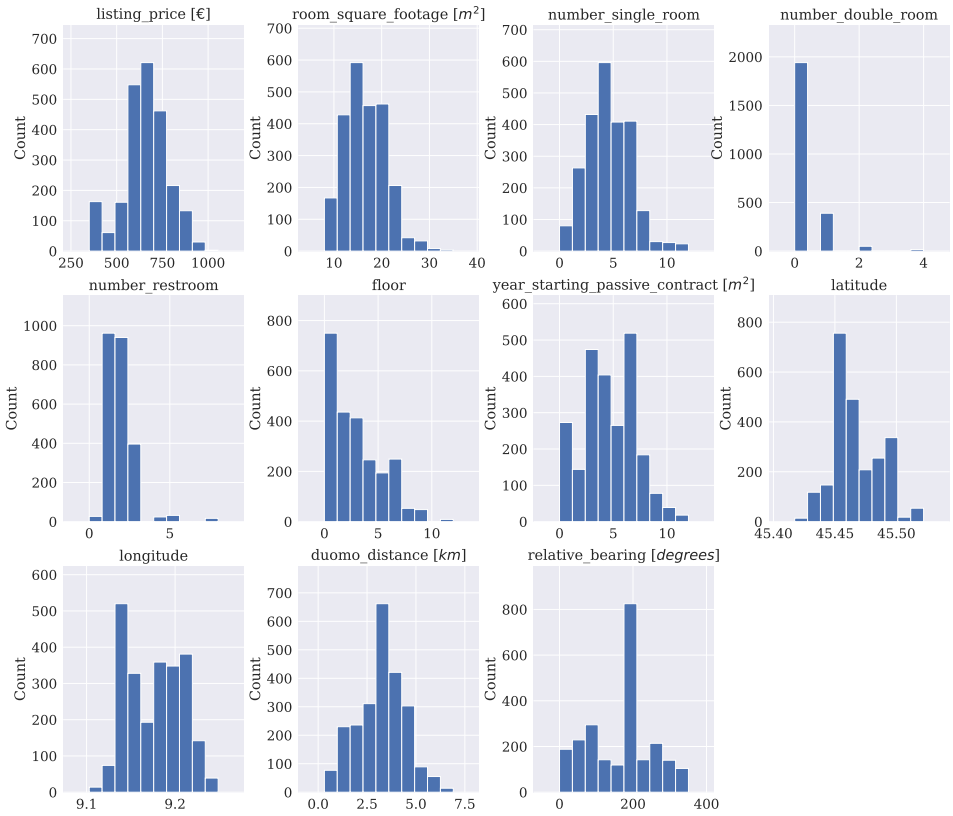


Figure 6.7: Data distribution plot.

Figure 6.7 shows the *data distribution* for each individual numeric feature. Figure 6.8 shows the *correlation matrix*, which is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. Looking at it, we see that most numeric and categorical variables have a low correlation so they can all be used as input to a regression model.

6.1.3 Modelling

The purpose of this first group of models is to replicate DoveVivo's current pricing strategy automatically. More precisely, we are looking for a model capable of predicting prices that reflect those currently assigned by the sales team. The goal of the prediction is to approximate the current listing prices accurately. Moreover, this tool can be used to isolate those rooms for which the predicted listing price is very different from the current one. Indeed, there could exist some rooms whose price has been set according to specific criteria, which, therefore, must be priced according to ad-hoc criteria. Otherwise, if

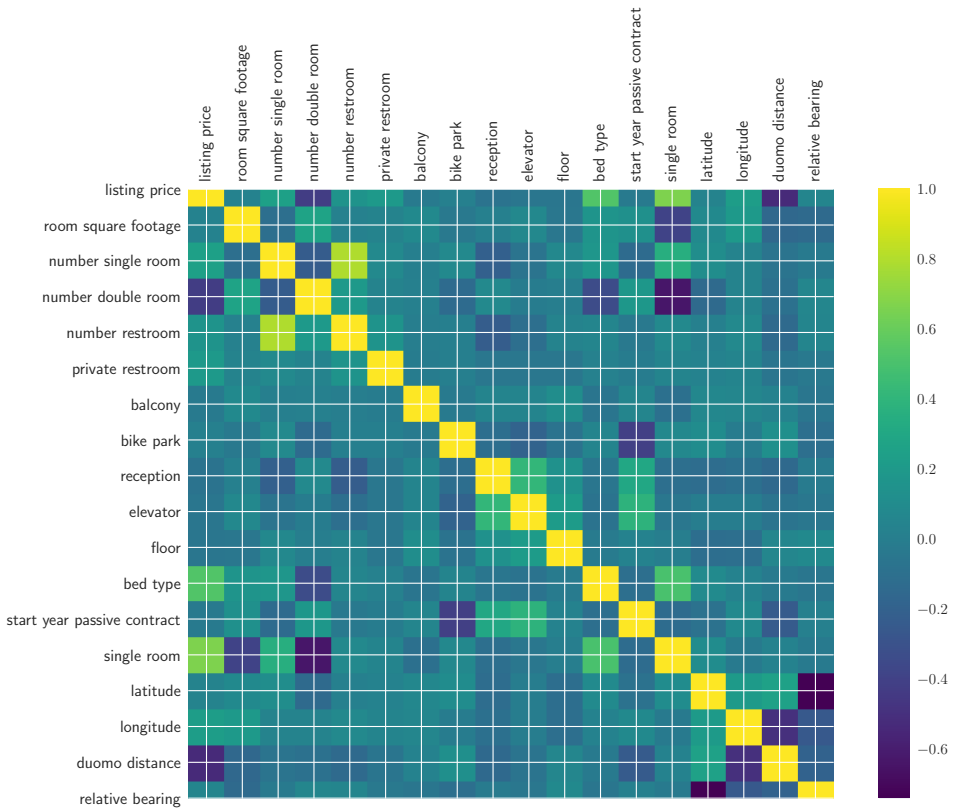


Figure 6.8: Correlation matrix heatmap for rooms' dataset.

these rooms do not have such particular characteristics as to justify the gap between the predicted and the listing price, it is possible to consider the price returned by the model as a suggestion of an increase or decrease in the value to assign. We now present the various regression models that we have applied to DoveVivo datasets.

The regression models we designed are:

- **Setup 1**
  Zone Model with Latitude and Longitude: we have built a different model for each zone. According to the area where a given room is located, a specific model is applied. Among the inputs of the dataset containing only active contracts, the latitude and longitude attributes were chosen to indicate the apartment coordinates;

- **Setup 2**
  Zone Model with Radius and Angle: we built a different model for each zone. According to the area where a given room is located, a specific

model is applied in which, among the inputs of the dataset containing only the active contracts, the attributes radius and angle have been chosen to indicate the coordinates of the apartment;

- **Setup 3**
  Unique Model with Latitude and Longitude: a unique model for the whole dataset. The attributes that express the position of the apartment are latitude and longitude;

- **Setup 4**
  Unique Model with Radius and Angle: a unique model for the whole dataset. The attributes that express the position of the apartment are radius and angle.

We built both parametric models such as linear regression and non-parametric regressions based on decision trees, ensemble of decision trees, and Support Vector Machine. In parametric modeling training data are used to set the weights of the model, then, during the test phase, the parameters are used to obtain predictions on unseen test points. Conversely, in non-parametric modeling, predictions are made directly as a function of the training data, so they are needed at test time. The Machine Learning models we tested are those that are described in detail in Chapter 4.

Parametric Models

In our experiments, we tested four parametric linear regression models using three different optimizers:

- **Ordinary Least Square** (OLS) (Section 4.2.3): chooses the parameters of a linear function of a set of variables by the principle of least squares, i.e., minimizing the sum of the squares of the differences between the observed dependent variable in the given dataset and those predicted by the linear function;

- **Non-Negative Least Square** (NNLS) (Section 4.2.5): is a type of constrained least squares problem where the coefficients are not allowed to be negative. Each component of the vector of parameters should be non-negative;

- **Bounded Variable Least Squares** (BVLS) (Section 4.2.6): is a generalization of the NNLS method with simultaneous upper and lower fixed bounds for each parameter to be calculated. We also tried this approach exploiting second-order polynomial variables.

The idea of exploiting optimizers different than the classic OLS arises from the need to impose a predetermined sign on the parameters of the regression. Indeed, it is natural to think that some features should not provide a negative contribution to the final price. In the same way, it is not exact to claim that all the features assign a positive or zero supplement to the prediction. Using BVLS, it is possible to set a range of values that the parameter corresponding to each variable can take. In this way, it is possible to assign a positive range of values to the reception, square footage or single room features while a negative range of values to features like the number of locals in the apartment and distance from the Duomo.

Training and Hyperparameters Tuning

Generally, after the training phase, we need to test the model performances on data that has not been used before. In such cases, the solution is to split the dataset into two subsets. One for training and the other for testing; this is done before training the model. We use 75% of the data for training and the remaining 25% for testing.

An l_2 regularization factor is applied to each model tested to avoid overfitting problems (Section 4.2.4). In order to achieve better performance, reducing the loss function, we tuned some parameters of the models. In the specific case of the algorithms based on the ensemble of decision trees. Typically, in machine learning, the objective function is multidimensional because it takes in a set of model hyperparameters. For simple functions in low dimensions, it is possible to find the minimum loss by creating a grid of input values (i.e, performing *grid search*) and seeing which one yields the lowest loss. The main parameters we tuned using a grid search are:

- number of trees;
- the minimum number of samples required to split an internal node;
- the minimum number of samples required to be at a leaf node;
- the maximum depth of the tree.

Performance Metrics and Models Comparison

The metrics that we use to evaluate the performance of the various models and then to compare them are *RMSE*, *MEA* and *ME*. Each error was calculated by 10-fold cross-validation, as described in Section 4.5.1. Regarding the setups that exploit zone models, the calculation of the overall RMSE error was carried out as follows:

- the dataset is first divided into 19 subsets, one for each zone;
- the data are then divided into train and test. A model is built for each subset;
- for each zone $z \in \{0, \dots, 18\}$ we calculate the mean square error MSE_z as follows:

$$MSE_z(y, \hat{y}) = \frac{\sum_{i=0}^{n_z-1}(y_i^z - \hat{y}_i^z)}{n_z}, \tag{6.1}$$

  where n_z is the number of samples in the dataset coming from zone z, y_i is the i-th observed values of the variable being predicted and $\hat{y}_i$ is the predicted values. Using the *k-fold cross-validation* with number of folds $n_f = 10$, as described in Section 4.5.1, we have the crossvalidated MSE:

$$\begin{aligned} MSE_{z\_cv}(y, \hat{y}) &= \frac{\sum_{k=0}^{n_f-1} MSE_z(y, \hat{y})}{n_f} \\ &= \sum_{k=0}^{n_f-1} \sum_{i=0}^{n_z-1} \frac{(y_i^z - \hat{y}_i^z)}{n_z n_f}; \end{aligned} \tag{6.2}$$

- the sum of quadratic error for a specific region SSE_{z_cv} is;

$$SSE_{z\_cv} = n_z \; MSE_{z\_cv}(y, \hat{y}); \tag{6.3}$$

- the total *RMSE* becomes:

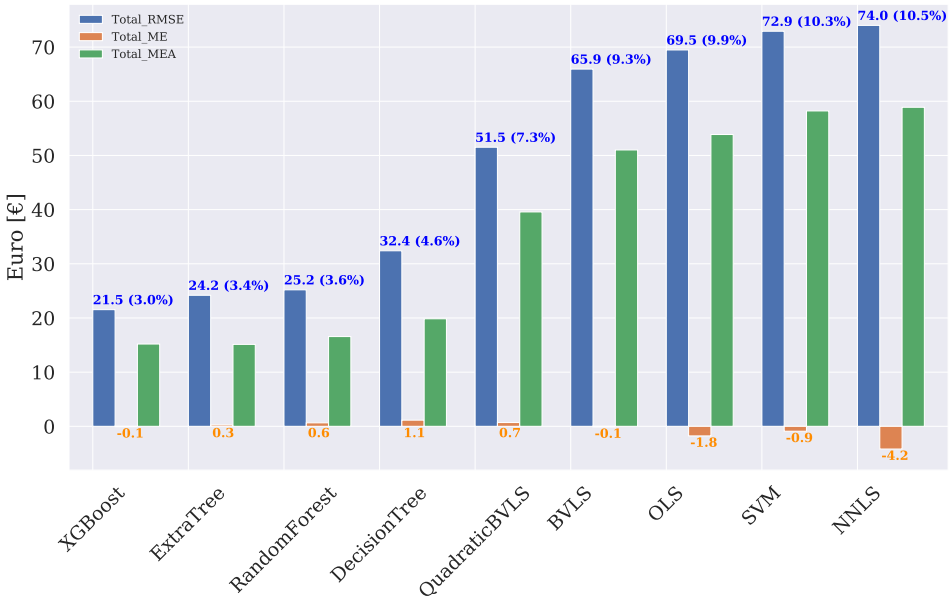
$$RMSE_{cv} = \sqrt{\frac{\sum_{z=0}^{n_Z-1} SSE_{z\_cv}}{\sum_{z=0}^{n_Z-1} n_z}}, \tag{6.4}$$

  where $n_Z = 19$ is the number of different zones the rooms are divided in.

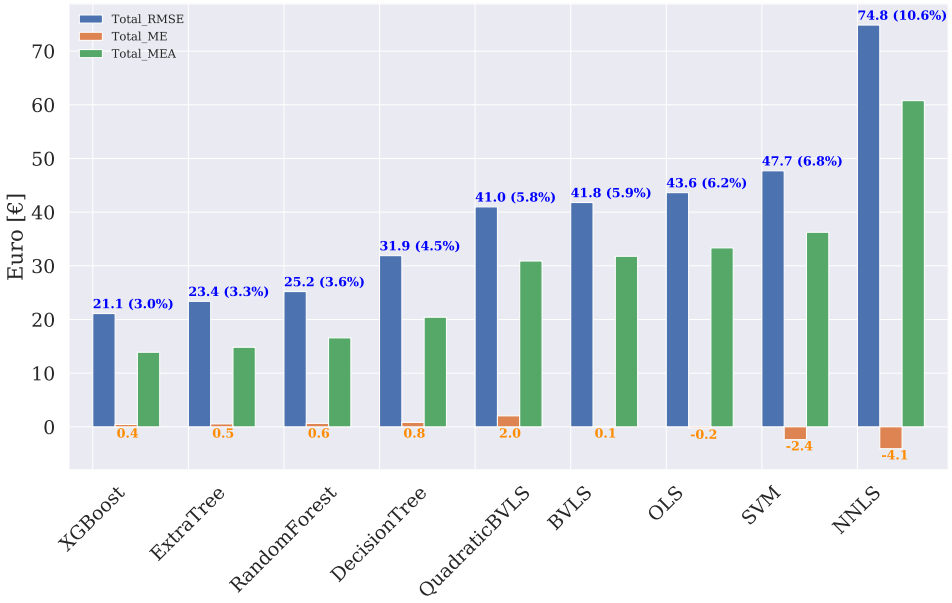
The same approach is applied for each regression metric.

6.1.4 Results

The results of the previously presented models are described in Figures 6.9a, 6.9b, 6.10a, and 6.10b. The figures show the performances of the models sorted by *Root Mean Square Error* (RMSE). As described in Section 4.5.2, the lower is this value, the lower is the average error the model makes in predicting new values. As far as parametric models are concerned, a zone approach performs much better. Indeed, regarding the *BVLS* and *OLS* bar in the graphics in Figures 6.10a and 6.10b, we get an error in terms of *RMSE*



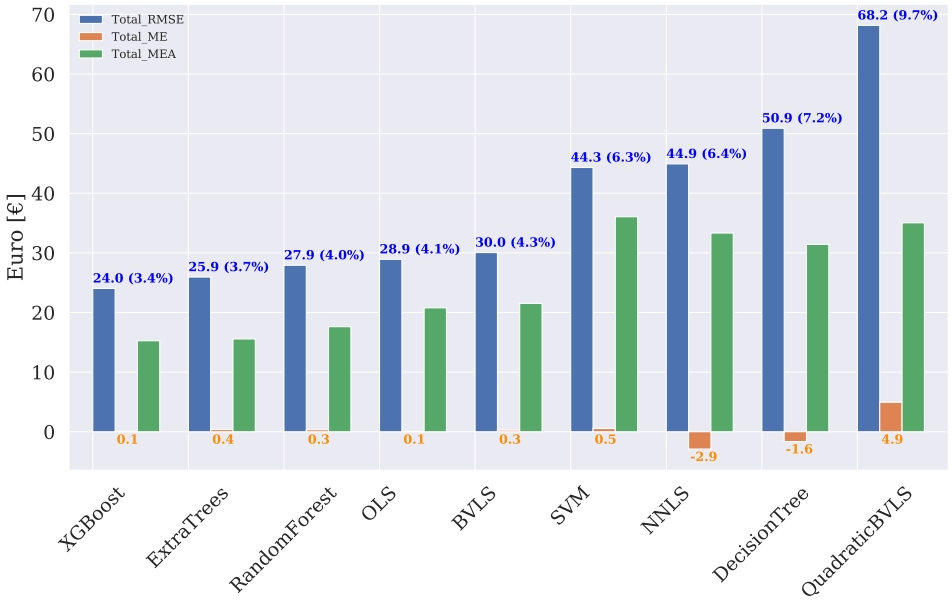
(a) Regression models without zones using latitude and longitude.



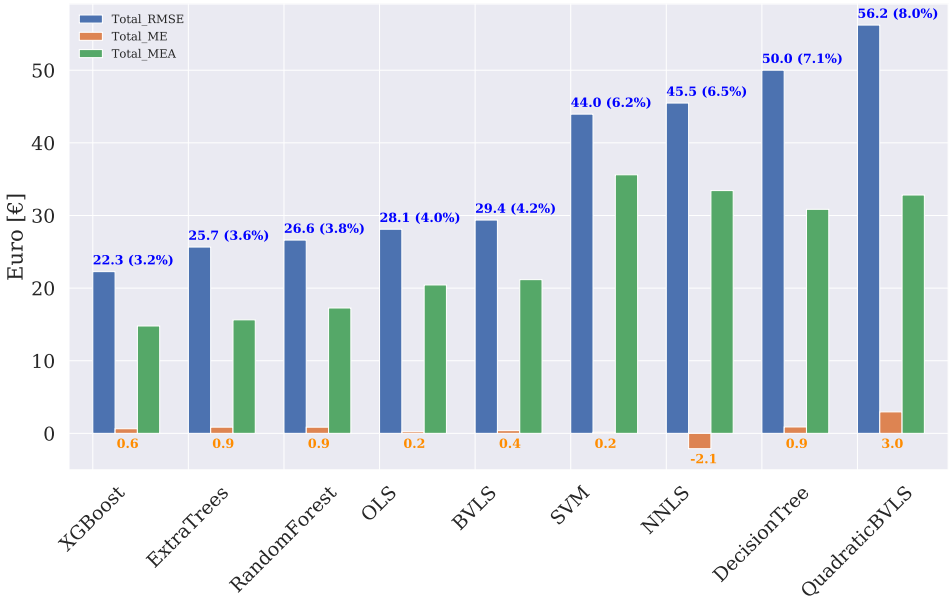
(b) Regression models without zones using distance and angle.

Figure 6.9: Regression models without zones.

from 4 to 4.3 percentage points. The RMSE percentage is calculated on the price range obtained by the difference between the maximum and the minimum price offered by DoveVivo in its pricing list. This delta price is around



(a) Regression models with zones using latitude and longitude.



(b) Regression models with zones using distance and angle.

Figure 6.10: Regression models with zones.

700 euros. We have transformed this percentage value into the corresponding amount in money: the prices returned by the parametric models of *BVLS* and *OLS* differ on average about 29 euros from those in the DoveVivo price

list. The lowest *RMSE* error is the one provided by a decision tree ensemble models: *XGBoost*, which is described in Section 4.4.5. The results confirm it to be the best model in terms of average error for all our experiments. In particular, in the setup of models without zones taking advantage of the distance and the angle, shown in Figure 6.9b, *XGBoost* obtains a percentage error of about 3 percentage points corresponding to 21.10€. These prices are close to those offered by DoveVivo they can be used as a suggestion to follow to create the new price lists while those that deviate a lot can be seen as a suggestion of price variation.

6.2 Second Problem Formulation - Experiment

6.2.1 Data Collection and Feature Description

Like the previous formulation, the data necessary for our analyses are collected through SQL queries from DoveVivo's database. As explained in Section 3.2.2, for this second formulation, we also need data regarding room contracts. For these data, we need to access the following database tables:

- **clu_postoletto**: containing all the room information;
- **clu_locazione**: includes information on the contracts for each room.

We perform a first filtering of the data through the query. We only consider the contracts of the rooms of the city of Milan with the end date of the contract greater than the start date.

Features Description

From the query, we obtain a dataset of 11 features and 9825 samples, where each row corresponds to a contract, active or concluded, of a room with a tenant. In this section, we analyze in detail which variables we extract to create in our dataset:

- **clu_datastipula**: is the date on which a new customer signs the lease contract with DoveVivo for a bed;
- **clu_datainiziocontratto**: it is the effective date on which the lease begins;
- **clu_datadisdetta**: it is the date on which a tenant of a room informs DoveVivo about his/her desire to leave the room;

- **clu_datascadenza**: it is the effective end date of the contract stipulated between a customer and DoveVivo. Contracts usually have a standard duration of 4 years; if the tenant wants to leave the room earlier, he must inform the company a few months in advance;
- **clu_starting passive**: this is the date on which the rental contract between DoveVivo and the owner of an apartment begins. This field is used to indicate how old an apartment is;
- **clu_canonemensile**: represents the monthly contract price at which the room is rented during all the duration of the contract;
- **clu_modellocontratto**: it is a binary field that contains only two values of string type about the kind of contract:
  - CNT-DET: indicates that the contract is short-term, that is, lasting a few months;
  - CNT-IND: indicates that this is an indeterminate type of contract. Indeterminate means a standard $4 + 4$-year lease;
- **st_contratto**: is a binary field indicating the status of the contract:
  - Active: indicates that the contract is currently active and in progress;
  - End of Validity: indicates that the contract is concluded;
- **st_postoletto**: is a binary field indicating the status of the room:
  - Active: indicates that the room is still under the responsibility of DoveVivo;
  - Inactive: indicates that the room is no longer operable by DoveVivo.

6.2.2 Data Cleaning and Data Exploration

We want to build an algorithm able to predict the binary label regarding the vacancy of a room given a specific price. In this section, we present our analysis on raw data to see how the dataset should be, before moving on to the algorithm modeling phase. The first step is to reconstruct the DoveVivo's *contracts history*. The purpose of this descriptive analysis is to:

- Identify and possibly eliminate inconsistencies and outliers;

- Reconstruct the history of all the contracts stipulated for each bed by calculating the elapsed period between the checkout date and the start date of the subsequent contract, i.e., the period of stay and the rental period.

The phase of data cleaning and data inconsistency resolution is of fundamental importance. The matrix $\mathbf{Z}$ corresponds to all the available contracts where $\mathbf{z}_i$ is a vector representing the generic contract i-th, with $i \in \{1, \ldots, n_c\}$, where n_c is the number of contracts. We list the type of inconsistencies we can find in the dataset and the solutions we use to fix them:

- to analyze a single room at the time, we group the rows of the dataset by **id_postoletto** and order them by the date of the beginning of the contract;
- if two contracts are partially overlapping, i.e, if the contract $\mathbf{x}_i^j$ expiration date is greater than the starting date of the contract $\mathbf{x}_{i+1}^j$. Where j denotes the j-th room in the dataset, the expiration date of the contract $\mathbf{x}_i^j$ is moved one day before the contract starting date of $\mathbf{x}_{i+1}^j$. Figure 5.7 presents an example of overlapping contracts and how to fix them;
- in case that two or more consecutive contracts, for the same room, are completely overlapping, we keep only one of them;
- for each contract $\mathbf{x}_i$ that the stipulation date is earlier than the start date. If not, we move the stipulation date before the start of the contract.

Data Filtering

The data we analyzed has been collected over the past ten years. Over time, the storage criteria changed and generate unwanted errors. Therefore, usable data are less than those available collected by DoveVivo, and only the most recent can be analyzed. The filtering phase is used to eliminate all the values: *inconsistent*, *outliers* or *missing*. More specifically:

- we remove all contracts $\mathbf{x}_i$ with a monthly fee lower than 200 or higher than 1000 euros;
- we only keep *terminated contracts*. That is, where the field **st_contratto** is equal to 0, we drop the contract since it lacks some of the necessary information (censored data);

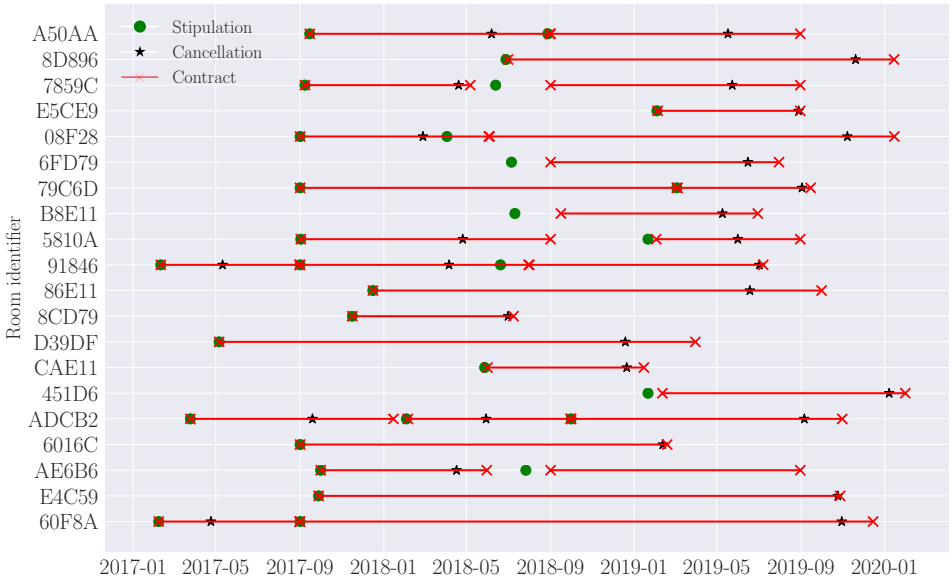


Figure 6.11: Contract history reconstruction.

- we select *indeterminate contracts* since during a preliminary analysis we detected that they have a completely different distribution;

- we keep contracts that last at least 30 days. There is no duration field in the dataset, but it can be calculated considering the contract expiring date of the generic $\mathbf{x}_i$ minus the start date of the same contract.

Figure 6.11 shows a depiction of some of the contracts of DoveVivo's rooms on a temporal axis. Each row represents a room: the duration of the contract is marked with two red **X**s joined by a **line**, the date of the conclusion of the contract by a green **dot** while the date of termination by a black **star**. We can observe that for some rooms there is a period of a few months between one contract and the subsequent, while for others, this time is almost zero.

Data Integration and Feature Engineering

We integrate the dataset obtained so far with the data regarding the rooms. The selection of the proper features to be included has been lead by the previous analysis described in Section 6.1, which allowed us to get a clean dataset with all the DoveVivo operating rooms of Milan. We merge the two datasets (using the variable **id_postoletto** as an identifier of the room). Unlike the first formulation, the model we create to classify vacancies needs

a binary target not included in the initial data. As for the classifier label, we create other features to give the model more robustness:

- **clu_sfitto**: this field indicates the number of days a room has been without tenants before being rented. To calculate the vacancy of a generic room we must consider two consecutive contracts $\mathbf{x}_i^j$ and $\mathbf{x}_{i+1}^j$. **clu_sfitto** is calculated by the difference between the contract starting date of $\mathbf{x}_{i+1}^j$ and the expiring date of $\mathbf{x}_i^j$. We denoted a sample as an outlier and therefore eliminated it if the sample with **clu_sfitto** less than 0 or greater than 100 days, where the upper bound have been selected by asking the opinion of DoveVivo's expert team;

- **clu_sfitto_cla**: the classifier needs a binary target, we discretise the **clu_sfitto** field to obtain two classes:

  - **class 1**: if we have at least ten days of vacancy;
  - **class 0**: otherwise;

- **sindata** and **cosdata**: attributes to allow the model to capture seasonal patterns, calculated through the formulas in Equations (5.4) and (5.5);

- **anno_contratto**: it is the year of the beginning of the contract; $\mathbf{x}_i^j$;

- **mese_inizio_passivo**: to indicate how old an apartment is, we introduce the number of months from the **clu_datainiziopassivo** to the present day;

- **clu_aumperc**: this field indicates the percentage increase in monthly fee that a room has, compared to the previous contract. If the price rises, the increase also increases;

- **NiNb**: maximum number of tenants per number of bathrooms in the apartment. Given a generic room $\mathbf{x}$, we have:

$$NiNb = \frac{nsr + 2ndr}{nb},$$

  where nsr and ndr are the number of single and double rooms in an apartment, respectively, and nb is the number of bathrooms present in the flat.

The prices listed in the database under the label **clu_canonemensile**, refer to the contract prices that a tenant has undertaken to pay monthly for the contract duration. DoveVivo gave us a small dataset containing the old

listing prices. Ordinarily, the company updates prices once or twice a year. The price list provided is composed as follows:

- **id_postoletto**: unique code of each room;
- Lists of prices divided by time, the date to be considered is the stipulation of the contract:
  - **ListingPrices2016** if the stipulation date is before 1-May-2017;
  - **ListingPrices2017** if the stipulation date is before 1-May-2018;
  - **ListingPrices2018-1** if the stipulation date is before 1-Nov-2018;
  - **ListingPrices2018-2** if the stipulation date is before 1-Jul-2019;
  - **ListingPrices2019** if the stipulation date is after 1-Jul-2019;

We integrate this new information by adding the **clu_prezzolistino** feature in the dataset. When an **id_postoletto** or a corresponding price is missing in the old prices dataset, we choose to use the monthly contract fee as the listing price instead.

Class Balance analysis and Data Normalization

The ratio between the number of positive classes, in the target feature, compared to the total is about $1 : 10$. This means that only one sample out of 10 is labeled as having been vacated. In Section 5.5, we discussed how to deal with a class unbalance problem. In particular, we use a variant of the oversampling technique to obtain two results. First, we want to guarantee the monotonicity of the prediction that makes our model and then we want to make the classes less unbalanced. Starting from the idea that if a contract with a price p generates vacancy, then the same contract with a price $p+10\%$ will inevitably create vacancy. In this way, we sample without replacement all the rows of the dataset having a positive target, update the value of the price and the percentage increase accordingly.

We drop all the rooms and contract identifiers and dates, to obtain a dataset composed of numeric features. Finally, using the min-max formula in Equation (5.3.2). We normalize all the data in the range $[0, 1]$, in this way, all the variables have the same importance.

Data Visualization

At this point, we got a dataset of 8522 contracts of 1742 unique rooms, 22 features and a target ratio of $1 : 5$:

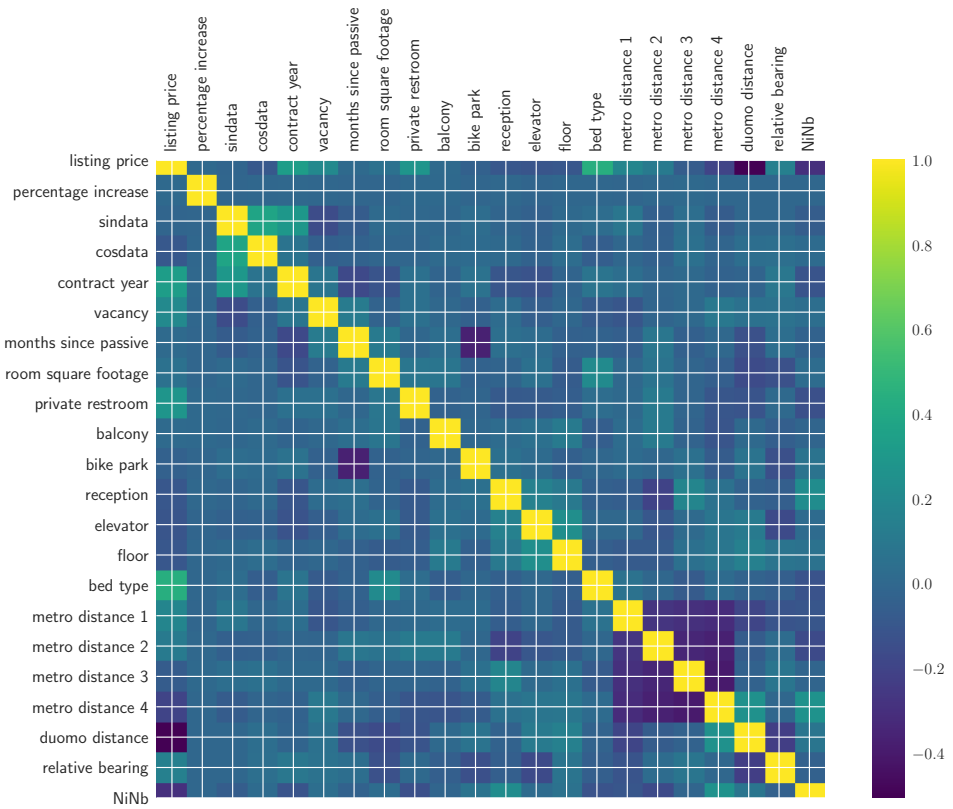


Figure 6.12: Correlation matrix heatmap contracts dataset.

- vacancy, target of the problem;
- five numeric features that represent the contract: percentage increase, sindata, cosada, contract year and monthly fee;
- sixteen features representing the room and apartment of interest. Ten binaries: private bathroom, balcony, bike parking, elevator, bed type, reception and four attributes to indicate the proximity to the nearest metro. Six numerical features: number of months from the beginning of the passive contract, room square footage, floor, distance and angle from the Duomo, NiNb.

Figure 6.12 shows the *correlation matrix*, which is a table showing correlation coefficients between couples of numerical variables. The table allows us to see which pairs of attributes have the highest correlation. Looking at it, we see that most numeric and categorical variables have a correlation factor close to zero so they can all be used as input to a classification model.

6.2.3 Modelling

The purpose of this formulation, as described in Section 3.2.2, is to create a classifier capable of determining whether a room generates vacancy. Given a monthly fee for a room, the model must be able to classify the room supporting a monotonous behavior. This means that as a price increases for a particular room, the prediction will be more and more likely to be positive, hence the room will generate vacancy. Figure 5.5 shows the behavior that the model has to pursue. Two non-parametric models are considered: *Decision Tree Classifier* and *XGBoost Classifier*. They are detailed in Section 4.4.1 and 4.4.5, respectively.

Training and Hyperparameters Tuning

The training phase is made on 75% of the available data while the test phase on the remaining 25%. The two models we use in this experiment, *Decision Tree* and *XGBoost*, have several hyperparameters that need to be tuned to achieve optimal performances.[2]

We use the random search (Bergstra and Bengio, 2012) approach together with 10-fold *cross-validation* to find the best parameters for the models. The hyperparameters that need to be set are the following:

- Decision Tree Classifier:
  - **min_samples_split**: the minimum number of samples required to split an internal node of the tree;
  - **min_samples_leaf**: the minimum number of samples required to be at a leaf node;
  - **max_feature**s: the number of features to consider when looking for the best split;
  - **max_depth**: the maximum depth of the tree.
- eXtreme Gradient Boosting Classifier:
  - **eta**: step size shrinkage used in update to prevents overfitting, correspond to the learning rate;
  - **max_depth**: maximum depth of a tree;
  - **lambda**: L_2 regularization term on weights;

[2]We used the implementation of these methods available in Python. Specifically, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html and https://xgboost-clone.readthedocs.io/en/latest/parameter.html.

Table 6.2: Classification models evaluation with their standard deviation.

	XGBoost	Decision Tree
Accuracy	**0.941** (±0.003)	0.739 (±0.024)
Recall	**0.910** (±0.019)	0.446 (±0.033)
Precision	**0.822** (±0.017)	0.595 (±0.059)
F1-score	**0.863** (±0.006)	0.507 (±0.022)

– **min_child_weight**: minimum sum of instance weight (hessian) needed in a child.

For both models considered, it is also possible to assign a different weight for each class. Associating a different cost to each class allow to penalize one category compared to the other during the training phase. *Cost-sensitive learning* approaches are designed with the idea that an expensive cost is imposed on a classifier when a misclassification happens during the training phase (Ali et al., 2015). These penalties can bias the model to pay more attention to the minority class.

Performance Metrics and Models Comparison

The selected models are evaluated with the standard classification metrics described in Section 4.5.2. The results obtained in terms of *accuracy*, *recall*, *precision* and *f1*, are calculated exploiting 10-fold *cross-validation*.

6.2.4 Results

The results for this task are reported in Table 6.2. The *XGBoost* model achieves better overall performance on every evaluation metric compared to a single *Decision Tree*. We verify the results obtained by examining the XGBoost classifier with a test dataset, following the procedure illustrated in Section 5.7.1. From the dataset of the rooms, we consider a cost range for each room centered on the listing price p_i, where i indicates a generic room of the dataset. For each room, we construct a price vector $\mathbf{p}_i$ with range $[p_i - 5\% \cdot p_i, p_i + 5\% \cdot p_i]$ and a uniform step size of 5 €. So we create a new set of rooms by replicating each sample as many times as the length of the new price vector $\mathbf{p}_i$. We add the features concerning the contracts to the new test rooms dataset, to obtain a matrix of samples suitable for our model. These features are:

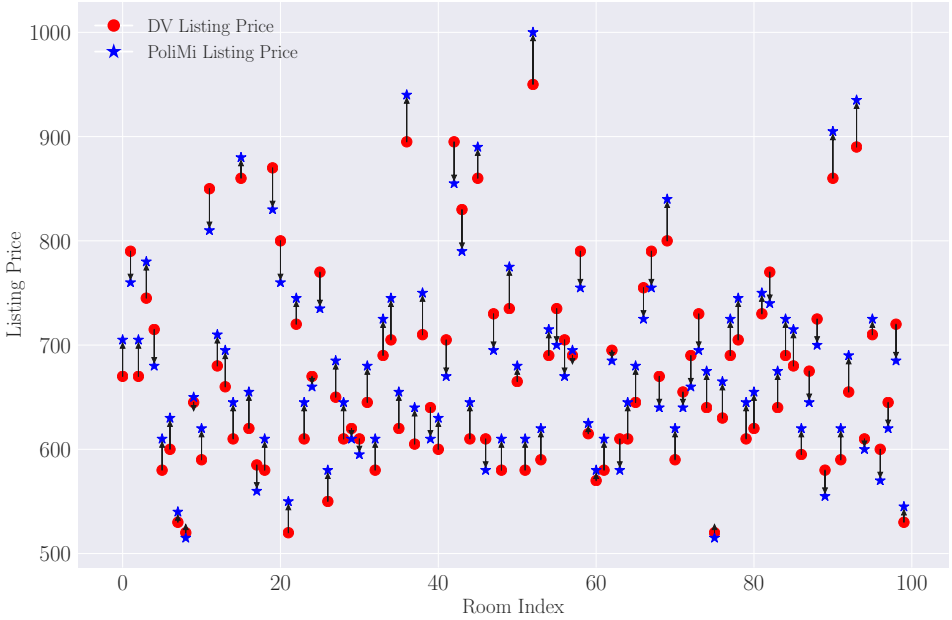


Figure 6.13: Plot of listing price increment - Single rooms.

- contract year equal to 2020, we want to predict a new room;
- **sindata** and **cosdata**, computed on 1st January 2020;
- price increase percentage, given the new prices, this field must be re-calculated based on the list price assigned by DoveVivo;
- **NiNb**;
- number of months since passive contracts.

In this way, we obtain a dataset of 33664 samples consisting of 2398 unique rooms. At this point, we can use the classifier to predict the vacancy of each room vector. The result is a monotonous binary vector for each room as a function of the assigned prices. Zero if the corresponding amount will not generate vacancy, one if the room will likely have a vacancy. We assign to each place the price $\hat{p}_i$ associated with the first occurrence of one in the vector $\mathbf{p}_i$. The room, therefore, obtains a positive price increment if the predicted value is higher than the listing price proposed by DoveVivo.

Figure 6.13 shows a sub-sample of 100 single rooms with the associated increment or reduction in price. A black arrow denotes the price change from the one assigned by DoveVivo, red dot, to the amount proposed by our classifier, blue star.

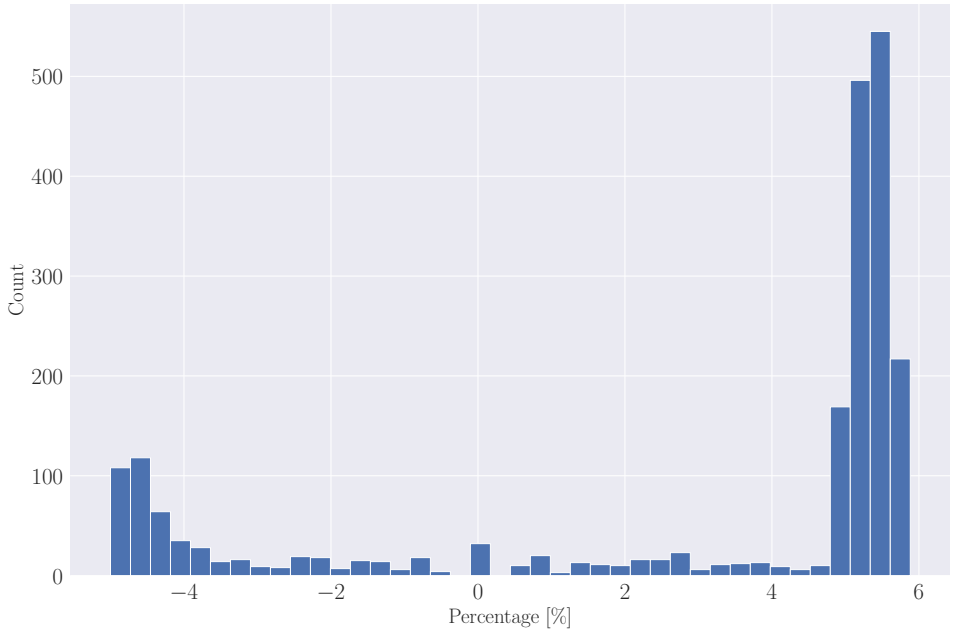


Figure 6.14: Cumulative histogram of listing price percentage increment - Single rooms.

Finally, Figure 6.14 shows the percentage increase in the listing price of the classifier compared to those offered by DoveVivo. These new proposed prices are intended as a suggestion. From these results, it can be understood that for some rooms, it is possible to raise the price to 6% without causing a loss, as they do not generate vacancies.

Chapter 7

Conclusion and Future Development

In this work, we aim to create *models* and *algorithms* capable of capturing the peculiarities of the pricing problem and automatically generate pricing schemes for the rooms rented by *DoveVivo*. The substantial growth of the real estate market and the increase in demand for rental rooms in Milan in recent years has made the automation of the pricing process a necessity. This situation has pushed DoveVivo to explore different solutions and eventually to invest in infrastructures that can support and help in managing the complexity of the pricing problem. Our study and analysis took into consideration numerous parameters and variables that are typical of the marketplace and environment in which the company offers its services and are based on the data collected by the company during the last 10 years. We proposed two different data-driven approaches to provide tools to support the DoveVivo's pricing strategy: the first approach replicates the current process of pricing that is based on the market value of the rooms set up by a pricing team in DoveVivo. This price of a new room is generalized by collecting information of similar rooms in the designated geographical area; the second approach analyzes the past contracts and value of each room and define a price that minimizes the time between a contract and another

Initially, we focused on the analysis and processing of the raw data provided by DoveVivo: cleaning them, filtering them, refining them and integrating them with new information. The pre-processing phase allowed us to create two different datasets to deal with the two different approaches. Finally, for both formulations, different supervised machine learning models were selected and compared. The experiments we have carried out have shown how to obtain an RMSE of 3% (about 22€) exploiting *XGBoost* Re-

gressor on DoveVivo rooms' data. While a classifier has revealed us how for some rooms it is possible to raise the current listing price to a maximum of 5% without risking having vacancy days and therefore a financial loss.

7.1 Limitations and Future Works

The first limitation in this research work was the lack of a large amount of data available for the analysis. Problems of small dataset are various and mainly regarding high variance: *over-fitting* becomes much harder to avoid, *outliers* become much more dangerous, and *noise*, in general, becomes a real issue. The analysis proposed for our formulations exploit samples supplied only by DoveVivo, which could produce *biased* results in terms of inputs generalization. The best way to solve this problem is to find *additional data* to enlarge and integrate the current dataset of Milan rooms. Further data could be provided by real estate companies and online platforms operating in the Milan area other than DoveVivo. There is the possibility that the data gathered from different sources could be incompatible with the one of DoveVivo, mainly because of the different way of collecting and storing them. However, they would be relevant in carrying out analysis regarding the *trend* of the *rental market*.

Another limitation is the absence of information regarding the *furnishings* of the apartments. This data is critical, considering that the DoveVivo's pricing team takes into account the style and type of furnishings of each home it owns. Unfortunately, this information is not stored in the databases that have been provided to us and therefore, it cannot be retrieved yet. It would be useful to start recording the dates of *renovation* and *modernization* of the apartments, to obtain more accurate and time-dependent prediction results.

For what concerns possible *future works*, many interesting directions can be taken to improve the work we have already done:

- *Integrate* into the room dataset other information regarding, for example, public surface transports or primary goods services such as supermarkets;

- *Aggregate* the rooms data of a city, by areas or neighbourhoods as we have done, allows a more detailed and accurate analysis. In the same way, it is interesting to try to cluster the rooms according to other judgment methods: points of interest, services, proximity to universities or urban areas in heavy expansion;

- Study and introduce *indicators* of past and present market or social trends to build more *robust*, *flexible* and *time-dependent* models;
- Finally, alternative supervised models could be introduced and tested. For example, in this work no in-depth analysis was carried out with *Deep Learning* models exploiting *neural networks*.

Bibliography

A. Ali, S. M. Shamsuddin, A. L. Ralescu, et al. Classification with class imbalance problem: a review. *Int. J. Advance Soft Compu. Appl*, 7(3): 176–204, 2015.

M. Babaioff, S. Dughmi, R. Kleinberg, and A. Slivkins. Dynamic pricing with limited supply. *ACM Transactions on Economics and Computation (TEAC)*, 3(1):1–26, 2015.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(Feb):281–305, 2012.

J. Biesiada and W. Duch. Feature selection for high-dimensional data—a pearson redundancy based filter. In *Computer recognition systems 2*, pages 242–249. Springer, 2007.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

P. Bülhmann and S. van der Geer. Statistics for high dimensional data. *Statistics (New York). Springer-Verlag, Berlin*, 2011.

E. Caldirola and A. Martino. Real estate market overview italy | 2016, 2016.

N. V. Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 875–886. Springer, 2009.

S. Chawla, J. D. Hartline, D. L. Malec, and B. Sivan. Multi-parameter mechanism design and sequential posted pricing. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 311–320, 2010.

D. Chen and R. J. Plemmons. Nonnegativity constraints in numerical analysis. In *The birth of numerical analysis*, pages 109–139. World Scientific, 2010.

Y. Freund, R. E. Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer, 1996.

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

S. García, J. Luengo, and F. Herrera. *Data preprocessing in data mining*. Springer, 2015.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

C. Gibbs, D. Guttentag, U. Gretzel, J. Morton, and A. Goodwill. Pricing in the sharing economy: a hedonic pricing model applied to airbnb listings. *Journal of Travel & Tourism Marketing*, 35(1):46–56, 2018.

J. W. Graham, P. E. Cumsille, and A. E. Shevock. Methods for handling missing data. *Handbook of Psychology, Second Edition*, 2, 2012.

J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

T. K. Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

T. M. Khoshgoftaar and P. Rebours. Improving software quality prediction by noise filtering techniques. *Journal of Computer Science and Technology*, 22(3):387–396, 2007.

R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14:2, pages 1137–1145. Montreal, Canada, 1995.

A. Labrinidis and H. V. Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

C. L. Lawson and R. J. Hanson. *Solving least squares problems*, volume 15. Siam, 1995.

R. J. Little and D. B. Rubin. *Statistical analysis with missing data*, volume 793. John Wiley & Sons, 2019.

A. L'heureux, K. Grolinger, H. F. Elyamany, and M. A. Capretz. Machine learning with big data: Challenges and approaches. *IEEE Access*, 5:7776–7797, 2017.

L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean. Boosting algorithms as gradient descent. In *Advances in neural information processing systems*, pages 512–518, 2000.

T. M. Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

D. Nielsen. Tree boosting with xgboost-why does xgboost win" every" machine learning competition? Master's thesis, NTNU, 2016.

J. R. Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

L. Rokach and O. Z. Maimon. *Data mining with decision trees: theory and applications*, volume 69. World scientific, 2008.

S. Russel. Artificial intelligence. a modern approach/russel s., norvig p, 2007.

P. B. Stark and R. L. Parker. Bounded-variable least-squares: an algorithm and applications. *Computational Statistics*, 10:129–129, 1995.

A. Stolwijk, H. Straatman, and G. Zielhuis. Studying seasonality by using sine and cosine functions in regression analysis. *Journal of Epidemiology & Community Health*, 53(4):235–238, 1999.

F. Trovo, S. Paladino, M. Restelli, and N. Gatti. Multi–armed bandit for pricing. In *Proceedings of the European Workshop on Reinforcement Learning (EWRL)*, 2015.

F. Trovò, S. Paladino, M. Restelli, and N. Gatti. Improving multi-armed bandit algorithms in online pricing settings. *International Journal of Approximate Reasoning*, 98:196–235, 2018.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

P. Ye, J. Qian, J. Chen, C.-h. Wu, Y. Zhou, S. De Mars, F. Yang, and L. Zhang. Customized regression model for airbnb dynamic pricing. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 932–940, 2018.

S. Zhang, C. Zhang, and Q. Yang. Data preparation for data mining. *Applied artificial intelligence*, 17(5-6):375–381, 2003.