Automatic analysis of high dimensional categorical variables in medical databases for the prediction of hospital bacteremia

Por
Jaime del Rey García

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Ingeniería Informática
Facultad de Informática

Directores : Óscar Garnica Alcázar y José Manuel Ruiz Giardín

Análisis automático de variables categóricas de alta dimensionalidad en bases de datos médicas para la predicción de bacteriemias hospitalarias

Madrid, 2020–2021

Authorization for dissemination and use

The authors of this work authorize the Complutense University of Madrid to use both the code and the report produced, solely for didactic purposes and mentioning the authors of the same.

Jaime del Rey García

Acknowledgements

First of all, thank Óscar and José Manuel for providing the dataset, sharing the idea and contributing to the study. A special thanks to Óscar again for his dedication and understanding throughout the process, he has made me feel that he was working side by side with me and contributed as a tutor at a level beyond academic. To all the people who have encouraged me to keep on going: to my father, my mother and my brother for their perseverance; and to my friends who have not hesitated to support me.

Sobre TEFLONX

Teflon X(cc0 1.0(documentación) MIT(código))es una plantilla de LaTeX creada por David Pacios Izquierdo con fecha de Enero de 2018. Con atribuciones de uso CC0.

Esta plantilla fue desarrollada para facilitar la creación de documentación profesional para Trabajos de Fin de Grado, Trabajos de Fin de Máster o Doctorados. La versión usada es la X

V:X Overleaf V2 with XeLaTeX, margin 1in, bib

Contacto
Autor: David Pacios Izquiero
Correo: dpacios@ucm.es
ASCII: ascii@ucm.es
Despacho 110 - Facultad de Informática

Contents

Page

1 Introduction **1**
- 1.1 Bacteremia . . . 1
- 1.2 Context . . . 3
- 1.3 Objectives . . . 3

2 Work organization **5**
- 2.1 Workplan . . . 5
  - 2.1.1 Development environment . . . 6
  - 2.1.2 Version control . . . 6

3 The Dataset **7**
- 3.1 Study of the data . . . 7
  - 3.1.1 Dirty categorical variables . . . 12
  - 3.1.2 Encoding String Categorical Variables . . . 14
- 3.2 Cleaning the data . . . 15
- 3.3 Processing the data . . . 18
  - 3.3.1 String similarity . . . 19
  - 3.3.2 K-NN . . . 22

4 Categories **25**
- 4.1 Category recognition . . . 25
  - 4.1.1 Choosing threshold value . . . 25
  - 4.1.2 Noise removal . . . 26
  - 4.1.3 Identifying variants of strings . . . 27
  - 4.1.4 Defining the classes . . . 29
  - 4.1.5 Evaluating the results . . . 30
- 4.2 Iterating the process . . . 34
- 4.3 Substituting categories . . . 35
  - 4.3.1 Dataset insertion . . . 35
- 4.4 Dataset preparation . . . 36

5 Bias and variance **39**
- 5.1 Avoiding Bias and Variance . . . 40
- 5.2 K-Fold Cross Validation . . . 41

6 Random Forest **43**
- 6.1 Fitting a Random Forest . . . 43

6.2 Evaluation metrics 45
6.2.1 Confusion matrix 47
6.2.2 ROC Graph 48
6.3 Weighing the features 50
6.4 Feature filtering 51

7 Conclusions **55**

8 Future improvements **57**

5 Literature and reference links **62**

Abstract

This project aims to continue and consolidate the study for the bacteriemia detection process and its diagnosis carried out by some faculty companions last year. A first glance through the analysis of numerical variables allowed a deeper understanding and the trace of an approach for a quick detection model. Now, categorical variables take relevance too in order to successfully achieve higher results in the classifier models.

The addition of categorical variables in classifier models has been around for at least five years due to the increase in computational capacity, and the benefits in the classifiers as direct consequence is clear. Yet, it is proven that, as complex and abstract as language is, classifiers do struggle when data with slang or abbreviations comes up for prediction, even if its linguistic register is heavily bounded, i.e. when strictly related to medical issues data is treated.

Throughout the study we will apply text cleaning and text processing methods to prepare the variables for use, since their format is heterogeneous and unsuitable to be processed by Machine Learning tools.

We will also apply the *string similarity* method to identify all those classes that can help in the algorithm classification process and we will assess the most suitable types of encoding for working with these variables.

Finally, we will apply the *Random Forest* Machine Learning algorithm on the set with techniques that allow us to avoid data learning bias and we will assess the results in terms of the success rates and the relevance of the variables in the decision-making process of the algorithm.

Keywords

- Bacteremia
- Comorbidity
- Predictive medicine
- Pathogenesis
- Dataframe
- Dirty category
- String similarity
- One hot encoding

- Adjacency matrix
- Adjacency list
- Binary encoding
- K-Nearest Neighbors (KNN)
- Bias and Variance
- K-Fold Cross Validation
- Random Forest
- ROC
- SHAP

Chapter 1

Introduction

1.1 Bacteremia

Bacteremia is a concrete case of **Bloodstream infections (BSIs)** where bacteria enter the bloodstream. Blood is a sterile environment, so when bacteria is detected the immune system arises a response to remove the external agent. Bacteria can enter the bloodstream as a complication of an infection or foreign bodies surpassing the skin as a defensive barrier, i.e. injuries ,burns or catheters. It is worth mentioning that surgery is a scenario where bacteria can enter the arteries or veins, thus they are exposed due to the nature of the procedure. Also, specific bacteria like resistant enterococcal species can cause bacteremia in patients who have had long hospital stays or frequent antibiotic use in the past.

Bacteremia is defined as either a primary or secondary process. In primary bacteremia, bacteria have been directly introduced into the bloodstream. Injection drug use may lead to primary bacteremia. In the hospital setting, use of blood vessel catheters contaminated with bacteria may also lead to primary bacteremia. Secondary bacteremia occurs when bacteria have entered the body at another site, such as the aforementioned cuts in the skin, or the mucous membranes of the lungs (respiratory tract), mouth or intestines (gastrointestinal tract), bladder (urinary tract), or genitals.

Bacteremia can be classified as monomicrobial or polymicrobial depending on the presence of microorganisms in the bloodstream. First is frequently found in patients with endocarditis or meningitis while the second one usually is diagnosed with necrosis of skin. Causes of bacteremia can additionally be divided into healthcare-associated (acquired during the process of receiving care in a healthcare facility) or community-acquired (acquired outside of a health facility, often prior to hospitalization).

The most common way to detect bacteremia is through blood cultures. A typical blood culture collection involves drawing blood into two bottles, which together form one ”culture”. One bottle is designed to enhance the growth of aerobic organisms, and the other is designed to grow anaerobic organisms.

Both bottles contain culture medium, a formula that supports the growth of microorganisms. These samples, named as blood cultures, are placed in an incubator for several days to allow microorganisms to multiply and form colonies. This new microbial colonies are

later tested for antimicrobial susceptibility to properly diagnose the bacteremia.

Figure 1.1: Blood culture examples
[1]

Sepsis, or septicemia, is a harmful response from the immune system to bacteria in the bloodstream that causes injuries to the organs of the patient [2]. Several immune responses may result in this condition, evolving into a life-threatening, potentially fatal scenario called septic shock, increasing the mortality rate of the bacteremia.

Usually the bacteria is quickly removed from the blood by means of no harm to the host, but it is indeed an infection disease which might lead to several important health consequences [3], depending on the bacteremia [4] reaching up to 50% of mortality rate.

Bacteremias are treated with antibiotics. Any patient with signs or symptoms of bacteremia or a positive blood culture should be started on intravenous antibiotics. The source of infection will determine the choice of antibiotics, as well as the symptoms, patient history with antibiotic use and allergies to antibiotics. The treatment is ought to be applied as soon as the bacteremia is diagnosed, were it not to be the case, a progression to sepsis could be produced, involving high mortality rates. However, blood cultures are not requested until sepsis, fever, or antibiotic resilience is detected, delaying the start of the treatment when needed.

Over the years the process of bacteremia diagnosis and recognition has improved remarkably from 5 to 6 days a few years ago to <24h in a recent study in 2019 [5]. Yet it was biased only to men and not so precise, the study estimates positive future results with clinical assessment. Although these are promising conclusions, being able to state and predict the treatment of the bacteremia before the results would lead to diminishing the mortality rate, along with the effect of the symptoms in the host.

The prognosis of bacteremia is so poor that it is rated as a high risk disease [6]. That is why the early diagnosis and the correct application of an effective treatment for bacteremia are so important. This leads to the point where the idea of training a prediction model to fasten the beginning of the treatment or, even the diagnosis, stands out; for it has been proven that early steps decrease the mortality rate when correctly diagnosed. Nowadays prediction models do not make a real difference: studies showing promising results happen to be made with short population samples or over homogeneous sample groups.

Bacteremia commonly starts in soft tissues which mainly involve **organs** that take part in the digestive, genitourinary and even the respiratory system, as it has been already

mentioned, such as lungs, stomach, or the skin. That is why it would be fair to consider and evaluate other symptoms when trying to predict bacteremia.

To sum up, bacteremia is a type of bloodstream infection that almost always requires treatment with antibiotics. It is diagnosed by blood cultures, which may take up to 5 or 6 days. Bacteremia is a process with different causes and a high mortality rate that benefits from early detection.

1.2 Context

Artificial Intelligence is a branch of computer science, specially influenced by probability and statistics, that studies, designs and applies algorithms for problem solving, the so-called Machine Learning algorithms.

Its popularity has been booming over the last decade, driven by the trajectory of the development of computer architectures and its computational capabilities. The rate at which computers have been operating recently is the key factor that promoted the improvement of Machine Learning algorithms to test models.

The goal of Machine Learning is to minimize the error of the results the models return, based on the parameters learnt. In order to do so, models are iteratively trained with sample datasets to improve the results, modifying when needed the values of the functions parameters.

Machine Learning can be classified into supervised and unsupervised learning. Supervised learning receives labeled datasets as input and the model should learn the function that best approximates the result to the output based on the training input; it usually weighs the most relevant parameters in the decision making process. On the other hand, unsupervised learning´s goal is to find patterns or similarities among data within datasets that allow to label and find the different classes in which the data can be divided. This learning process cannot validate the result of the output, for it has no labeled data to compare with.

Machine Learning models are benefited from great datasets with low missing data rates. The output is ought to be more accurate the more training is put into the model, yet it is important to avoid the extreme where overfitting is produced due to excess training with a single dataset, in which case the model loses accuracy with no training data.

1.3 Objectives

This project is the continuation of the study conducted last year about Machine Learning techniques for healthcare-associated bacteremias by other companions [7]. In such study categorical, ordinal and boolean variables were the goal, applying different data manipulation techniques; for instance; one-hot classifiers, separate-class methods or imputation methods.

Although accuracy in the prediction models might seem as the main goal, it is as important as the transparency of the model and the underlying algorithm. Were it not for this condition, the previous study might have been concluded, but this is a health related investigation project that is expected to be applied in real scenarios.

Thus, the target of this project is to find the most relevant variables for the detection and diagnosis of bacteremia by means of ML models, including the nominal categorical variables skipped in the previous study and improving the results of the predictions in the most explainable models.

Chapter 2

Work organization

2.1 Workplan

Before approaching the project, a weekly follow-up was agreed through which Óscar could evaluate the progress made, giving continuity to the work. In this first contact, the development environment, the tool used for version control, as well as for the work flow, were approved. Similarly, the use of the Goole Meets tool for communication and meetings, and the use of a Google Chat room for messages between meetings is set.

In the first instance, Óscar explained that it would be necessary reviewing the work carried out the previous year; reading of research on machine learning results with different coding methods applied to sparse matrices and understanding the set was given equal relevance of data with which to work and the evaluation of the variables marked as pending from the previous study.

From there, the work plan was established as follows:

(i) Create GitHub directory tree.

(ii) Get access to the previous year repository.

(iii) Reed the articles about *Missing Data* [8] from Yufen Ding and Jeffrey S. Simonoff, *Encoding High Cardinality Variables* [9] and *Similarity encoding for learning with dirty categorical variables* [10] from Patricio Cerda.

(iv) Install *Jupyter Notebook*.

(v) Install *Python 3*.

(vi) Get familiar with the previous study and the script used to work with the data features.

(vii) Evaluate the dataset variables.

(viii) Check with the physician the categorical variables to study.

(ix) Find in the resulting DataFrame object all distinct values for the study case variables.

(x) Modify the script according to the needs of the current features.

(xi) Create working environment workflow file with *pipenv*.

(xii) Cleaning accents and strange characters from the study features.

(xiii) Get all distinct values from the study variable *Otrascomor*.

(xiv) Depth search for text analysis techniques and tools.

(xv) Apply string similarity method to the resulting bag of words.

(xvi) Identify each of the different entities from the bag of words and replace them in the DataFrame for model evaluation.

(xvii) Check bag of words with 2 and 3 character minimum size abbreviations.

(xviii) Create substitution list for abbreviation replacement.

(xix) Run the script having applied the abbreviation substitution.

(xx) Complete the workflow with the categories obtained from running the script.

(xxi) Apply binary codification to all categories within the DataFrame for the study feature.

(xxii) Reorganize and clean the script.

(xxiii) Remove all predictor variables from the DataFrame before running Random Forest model.

(xxiv) Apply Random Forest to the DataFrame with binary codification.

(xxv) Run k-fold cross validation with 80/20 parameters for training and validation sets.

(xxvi) Apply Random Forest to the DataFrame with binary codification.

(xxvii) Retrieve graph with the weighted relevance of the features.

(xxviii) Retrieve ROC graph from the Random Forest execution.

(xxix) Write the project report.

2.1.1 Development environment

The development environment that has been used for this project is Jupyter Notebook, with the use of Python as programming language. Óscar emphasized the requirement to maintain a file with the latest stable version of the software and the libraries used necessary for the execution of the code. The use of pipenv, a production tool that creates virtual environments and maintains the build used during development, covers this need.

2.1.2 Version control

For version control, github was used which, in addition to storing the code versions, allows creating a follow-up by target cards according to the status of their completion, allowing control over all project tasks apart from development tasks.

Within the project, a folder structure was established to distinguish between documentation, code, patient data and results obtained.

Chapter 3

The Dataset

3.1 Study of the data

This chapter will develop all the information about the dataset, the treatment it has received for its correct manipulation, as well as its interpretation and subsequent preparation for its usage with Machine Learning tools.

The dataset is an anonymized dataset provided by José Manuel Ruiz, physician at the Hospital Universitario de Fuenlabrada, a 350-bed hospital with the following services: general surgery, urology, orthopaedic surgery, gynaecology and obstetrics, paediatrics, intensive care units (ICUs), haematology-oncology, internal medicine and cardiology. The database was gathered from 2005 to 2015, and it consists of 4357 anonymous patient records, a.k.a. instances, containing 117 features per patient, 49.3% female with age 65.1 $\pm$ 19.7, and 56.1% male with age 62.7 $\pm$ 20.2. Each instance contains demographic and medical data (medical history, clinical analysis, comorbidities, etc.) and the result of the blood culture, the feature to be predicted, which can take one of two values: bacteraemia and no bacteraemia. The database contains 2123 bacteraemia (51.3%), which includes aerobic, strict anaerobic and facultative anaerobic bacteria, and 2234 no bacteraemia (48.7%), including 1844 contaminations.The final classification of true bacteraemia was done in prospective time by an infectious disease physician, using all the previous data, including microbiological, clinical and analytical data.

The target variables in this project are those of nominal type from the list of variables, which result in the following: *desmotuci*, *uci*, *origin*, *otrascomor*. After a first evaluation, the origin variable was discarded due to its nature. This variable represents the origin of the bacteremia, which is a feature from the final diagnosis. Features that can only be obtained after the drawning of the blood culture should not be included because in the real process the physicians would not be able to count on that values. This would also contaminate the results of the Machine Learning models.

Tables 3.1 and 3.2 display the features in the dataset and the selected variables for the study.

Table 3.1: Features from the dataset

Features from the dataset		
periodo	Year of the study case	Ordinal
mes	Month of the study case	Ordinal
dia	Day of the study case	Scalar
edad	Age of the patient	Scalar
edada	Age per group	Ordinal
edada75	Age *goe* to 75	Scalar
edada85	Age *goe* to 85	Scalar
Diasdet	Detection time in days	Scalar
Prifrpos	First culture to grow	Scalar
microrga	Microorganism	Scalar
identif	Species of the bacteremia	Nominal
Microrgagrupo	Group of the microorganism	Scalar
anhonpol	Anaerobes	Ordinal
Anaerobio	Presence of anaerobes against all other bacteria	Ordinal
Hongos	Presence of fungs against all other bacteria	Ordinal
Stafcoag	Staphyloccocus coag	Scalar
Polimicr	Polimicrobian	Ordinal
microbpoli	Germs of microbial bacteria	Ordinal
gram	Gram stain	Ordinal
medio	True positive growth medium	Ordinal
frasae	Growth at least in aerobes	Scalar
frasanae	Growth at least in anaerobic flask	Scalar
frasextr	Bottles of extracted blood cultures	Scalar
contamin	Pollutant growth	Ordinal
mediocon	Growth medium of the pollutant microorganism	Ordinal
Antibiograma	Microorganism antibiogram	Nominal
Antiresist	Categorized antibiogram	Scalar
glucosa	Blood glucose	Scalar
Urea	Blood urea in mg/l	Scalar
creatin	Creatinine	Scalar
pcrcate	Categorical PCR	Scalar
pcr	PCR value	Scalar
leuc	Leukocytes	Scalar
hgb	Hemoglobin	Scalar
pmfn	PMN percentage	Scalar

Features from the dataset		
hbcateg	Categorical Hb	Scalar
plaqut	Platelets	Scalar
leucocit	Leukocytosis	Ordinal
trombope	Thrombopenia	Ordinal
Coagulación	Altered coagulation	Ordinal
so	Urine system analysis	Scalar
sedorina	Urine sediment	Scalar
diashosp	Days in hospital until bloodculture extraction	Scalar
lnghosp1m	Hospital admissions over the last month	Scalar
lnghosp12m	Hospital admissions of over 48h in the last year	Scalar
comentar	Comments	Nominal
COMORBIL	Comorbidity	Scalar
Diabetes	Diabetes	Scalar
Cardiopatía	Heart disease	Scalar
Enfresp	Chronic respiratory disease	Scalar
Neoplasia	Active neoplasia	Scalar
Insrenal	Renal insufficiency	Scalar
Hepatopatia	Liver disease	Scalar
Udvp	Parenteral drug addiction	Scalar
Alcoholismo	Alcoholism	Scalar
Otrascomor	Other comorbidities	Nominal
enfbasWeinst	Weinstein's underlying disease	Ordinal
esteroid	Steroids	Ordinal
drogadic	Drug addiction	Ordinal
antibiot	Antibiotics	Ordinal
inmunosu	Immunosuppressants	Ordinal
neutrope	Neutropenia	Ordinal
m_genitu	Genitourinary manipulations	Ordinal
m_respir	Respiratory manipulations	Ordinal
m_digest	Digestive manipulations	Ordinal
cirugia	Previous surgery	Ordinal
diagnost	Diagnosis of bacteremia	Ordinal
cateter	Days of last catheter placement	Scalar
cateter1	Type of catheter	Ordinal
Especialidad	Specialty where the bacteremia is produced	Scalar
servicio	Service where the bacteremia is produced	Scalar

Features from the dataset		
Ordinal		
urgencias	Blood cultures taken in the emergency room	Scalar
adquisic	Adquisition	Ordinal
durac	Days of fever febore blood culture	Scalar
tas	Systolic blood pressure	Scalar
tad	Diastolic blood pressure	Scalar
fc	Heart rate	Scalar
primtemp	First ER temperature with which blood cultures are drawn	Scalar
temporal	Temperature based on oral and axillary temperatures	Scalar
fiebre	Fever when blood cultures are drawn	Ordinal
fiebrePitt	Oral temperature classification in Pitt scale	Scalar
hipotens	Hypotension	Ordinal
Vasopre	Use of vasopressor agents at the time of bacteremia	Scalar
intubacion	Need for intubation at the time of bacteremia	Scalar
RCP	Cardiac resuscitation at the time of bacteremia	Scalar
Alerta	Consciousness at the time of bacteremia	Scalar
PITT	Pitt scale in number of ICU patients	Scalar
metastas	Metastasis	Ordinal
metasta1	Where the metastasis is produed, if any	Ordinal
evolucio	Evolution	Ordinal
muerte	Death	Ordinal
origen	Origin of bacteremia	Nominal
dxfinal	Final diagnosis	Scalar
origensos	Suspected origin of bacteremia at the time the blood cultures are drawn	Scalar
origenf	Origin of bacteremia in the final diagnosis	Scalar
origenva	Vascular origin	Scalar
atbempir	Empirical antibiotic treatment	Scalar
t_empiri	Empirical treatment adequate or inadequate	Ordinal
ttoesp	Specific adequate treatment	Scalar
t_especi	Specific treatment adequate or inadequate	Ordinal
diastto	Days of treatment until start of inadequate treatment	Scalar
t_quirur	Indication or not of surgical treatment	Ordinal
uci	Bloodcultures drawn in ICU	Nominal
ucidiashem	Days in ICU to blood culture drawn	Scalar
motuci	Reason of ICU admission	Scalar

Features from the dataset		
desmotuci	Reaso of ICU admission as text	Nominal
consfieb	Consult for fever	Scalar
sintomas	Fever symptoms	Scalar
stlocal	Locator syndrome	Scalar
dest	Destination	Scalar
vuelta_a	Return to ER	Scalar
tratamie	Antibiotic treatment	Scalar

Table 3.2: Selected features in the study.

Selected features	
desmotuci	Reason for admission to ICU
uci	Blood cultures from ICU
origin	Origin of the bacteremia
Otrascomor	Other comorbidities

A study of the variables is carried out individually to identify the characteristics of each variable. Starting with the variable *otrascomor* that contains the comorbidities with which each patient was admitted.

Comorbidities are additional disorders that patients present in addition to the disease for which they are admitted. This variable is characteristic because it can help to identify under what conditions a patient is more likely to have bacteremia.

This variable contains comorbidities as nouns separated by punctuation marks or other text characters. Comorbidities are not written in a homogeneous way, that is, some are written with diminutives, others contractions of different lengths of characters on the same word. The structure of the variable is itself a series from the pandas library where each index in the list represents the patient referred and the content within the list is a string detailing the comorbidities.

```
[1]: v_nominales['Otrascomor']
```

```
[1]:
0                deterioro cognitivo
1
2       CARDIOPATIA. EPILEPSIA
3                  valvulopatia
4              bcno, dm, hta, ci
                    ...
5389
5390                    Obesidad
5391
5392
5393
Name: Otrascomor, Length: 5394, dtype: object
```

Figure 3.1: Sample of the feature *Otrascomor*.

The typo of the variable is not homogeneous either, counting with upper and lower case indiscriminately. The separators, which indicate the end and the beginning of each comorbidity in succession, are also heterogeneous in a range from punctuation marks such as ',' or '.' to other types of elements such as '+' or 'e'.

Another attribute that can be observed is the appearance of spaces as prefixes and suffixes, in addition to the indexes of the series that have no content. There are also strings throughout the series that contain accents, apostrophes and other elements that pose difficulties for the representation of the text according to what formats may be used.

3.1.1 Dirty categorical variables

In the context of Machine Learning with natural language variables, dirty categories are the definition of non-curated data with high cardinality but redundancy: several categories reflect the same entity.

One of the main challenges with dirty categorical variables is to identify all the elements that are related and refer to the same entity or class. Without data cleaning, different string representations of the same category will lead to completely different results or sub-categories, not only because the different elements refer to the same category themselves but also because of errors such as typos that cause morphological variations.

From a data-integration standpoint, these categories may be seen as a data cleaning problem about entity resolution. Tasks such as deduplication, that tries to merge different variants of the same entity, seek to recognize different variants of the same entity, which may be the best case to apply as a preprocessing step. However, data cleaning usually requires human intervention and major costs in data analysis.

In this dataset almost all the examples are dirty and categorical for the variables of study. If two examples for the variable *Otrascomor* are taken, it can be understood that within this study there is a dirty categorical variable problem. To cater for different ways

information might appear, several entities are shown in Figures 3.2 and 3.3 for each of the possible subclasses.

```
[1]:  anemi
      anemia,
      anemia cronica,
      anemica cronica ferr,
      anemia ferropenica,
      anemia hemolitica,
      anemia megaloblastic,
      anemia microcitica,
      anemia n-n,
      anemia nn,
      anemia normocitica normocronica,
      anemia por deficit b12,
      anemia tr cronicos
```

Figure 3.2: Sample of the elements in the feature *Otrascomor* refering to anemia as dirty categories.

```
[1]:  alzheimer
      alzheimer avanzado,
      alzheimer evolucionado,
      alzheimer evolucionado (institucionalizada,
      alzheimer terminal,
      alzheimer.,
      alzheimer. asma,
      alzheimer. dm,
      alzheimer. enfermedad vascular cerebral,
      alzheimer. itus de repeticion.,
      ca. prostata. alzheimer avanzado. parkinson,
      demencia alzheimer,
      dm.alzheimer. tvp. itus de rep.,
      dtalzheimer avanzado,
      enfermedad de alzheimer,
      epoc. neumonia. alzheimer.,
      erc estadio 3. alzheimer.,
      hemorragia ceebral; e de alzheimer,
      hta; alzheimer
```

Figure 3.3: Sample of the elements in the feature *Otrascomor* refering to alzheimer as dirty categories.

In both examples, all the variables contain the word that we might consider as the entity

they all refer to, but some typos already stand out such as **anemi** vs **anemia**, or the difference between **alzheimer avanzado** and **alzheimer terminal**.

The goal, therefore, will be to provide the most faithful approximation to what the main categories should be, taking into account which of the variables refer as the same category too in order to replace all the dirty variables with one single entity in order to simplify the process of data processing and to improve the results for the Machine Learning models by lowering the amount of classes and attributes to take into account.

3.1.2 Encoding String Categorical Variables

String variables usually require vector representation when inside datasets that are thought to fit for statistical and Machine Learning models. This is when encoding or numerical vector representation of the entries takes place.

If we consider string entries as nominal unordered categories, the prediction of the Machine Learning model can be improved. In such a situation, it is necessary for the categories to be mutually exclusive and unrelated to each other with a fixed known set of possibilities.

The classic approach to encode categorical variables for statistical analysis is one-hot encoding as it creates vectors that agree with the general intuition of nominal categories: orthogonal and equidistant. However, for high-cardinality categories, one-hot encoding leads to feature vectors of high dimensionality, which struggles in big datasets increasing the number of categories and the computational complexity and cost.

In this project, the dataset has too many attributes and, as previously shown, there are several entries that may refer to the same category. Therefore, new approaches are introduced: in first place a new type of encoding called similarity encoding [9] and ultimately, how can it be applied to Machine Learning problems [10].

- Let $s_i \in S, i = 1..n$, the category corresponding to the i-th sample of a given training dataset. Given a string similarity $\text{sim}(s_i, s_j) : S \times S \to [0, 1]$, similarity encoding builds a feature map $x^{sim} \in \mathbb{R}^k$ as:

$$x_i^{sim} \stackrel{\text{def}}{=} [sim(s_i, s^{(1)}), sim(s_i, s^{(2)}), ..., sim(s_i, s^{(k)})] \in \mathbb{R}^k \tag{3.1}$$

  where $s^{(l)} \subseteq S, l = 1...k$, is the set of all unique categories in the training set chosen heuristically.

- Now, as categorical entities are not numerical, a matrix X is needed to build a feature map. Such table is a set $A_j, j = 1..m$, i.e. the column names. Each attribute has a domain $dom(A_j = D_j)$. A table is defined as a *relationr* on the scheme R where is specified by a set of tuples $t(i) : R \to U_{j=1}^m D_j, i = 1...n$. If A_j is a numerical attribute, then $\text{dom}(A_j) = D_j \subseteq \mathbb{R}$. If, on the other had, A_j is a categorical attribute represented by strings then $D_j \subseteq \mathbb{S}$, where $\mathbb{S}$ is the set of finite-length strings that can represent a variable.

  We define then a feature matrix X from the relation r that consist of replacing the

tuple elements $t^i(A_j), i = 1...n$ by feature vectors:

$$X_j^i \in \mathbb{R}^{P_j}, p_j \geq 1 \tag{3.2}$$

Using the same notation in case of numerical attributes, we can define $x_j^i = t^i(A_j) \in \mathbb{R}^{P_j}, p_j = 1$ and write the feature matrix X as:

$$X = \begin{bmatrix} x_1^1 & ... & x_m^1 \\ . & . & . \\ . & . & . \\ . & . & . \\ x_1^n & ... & x_m^n \end{bmatrix} \in \mathbb{R}^{nxp}, p = \sum_{j=1}^{m} p_j \tag{3.3}$$

In standard supervised-learning settings, the observations, represented by the feature matrix X, are associated with a target vector $y \in \mathbb{R}^n$ to predict.

3.2 Cleaning the data

Cleaning the data in language manipulation problems is usually a time and resource consuming task. Trying to get a pristine dataset would not be worth the effort. Yet, taking into account that we attempt to use Machine Learning and data processing tools it is important to ensure that at least it is coherent in format and language.

To clean the data, different tools will be used, provided by various python libraries. The idea is to create a bag of words and be able to define the categories from it using the string similarity method explained in the previous subsection.

At the beginning, the categorical variables of the dataset were separated from the others, these being the main objective of this project. Now the feature *Otrascomor* is selected for the first preprocessing pace.

The first step is to standardize the text format by eliminating all the characters that may give reading errors according to the text format, such as accents, diacritical marks or apostrophes.

The first approach is the use of regular expressions to replace characters that may present problems for Machine Learning models. The problem with this option is, in addition to not being able to be applied to series, the fact of having to list all the possible substitution options.

In a second approach, a library is used, **unidecode**, which maps the characters found in a text to ASCII characters, that is, universal format. This library has the drawback of not producing good results for oriental languages but, taking into account that it works with Spanish and seeks to eliminate accents (a result that can be obtained with the English keyboard), aforementioned library provides the solution and the desired result.

```
def remove_diacritics(x):
    """
    Argumentos clave:
    x    -- objeto tipo serie de string
    Devuelve: todas las cadenas sin tildes diacriticas
    """
    return unidecode.unidecode(x)

def espacios_lower(x):
    """
    Argumentos clave:
    x    -- objeto tipo lista de string
    Devuelve: lista string en minuscula eliminando los espacios en
    prefijo y sufijo de cada elemento
    """
    return x.strip(" ").lower()
```

Once a homogeneous format is achieved, it is necessary to identify the separators to be able to isolate the strings that potentially represent a class. To achieve this, a list is created with each separator identified in the variable and the following process is repeated: each element of the series is separated by the token that identifies the separator and the resulting strings are joined in a list object.

Two approaches were used initially: the use of commas as a separator and the use of commas and periods as separators. The reason is that they are the characters with that have the most frequency throughout the variable.

It was decided to differentiate between the bag with commas removed and the bag with commas and dots removed. This is not only because of the frequency of the separators, but also because some entities include the element '.' as part of the diminutive or contraction on some of the words embedded on it.

```
def separa(serie,separador):
    serie_separadas=[x.split(separador) for x in serie]
    return functools.reduce(operator.add,serie_separadas)
```

After the first outcome using the comma-separator bag of words, explained in the results chapter, separators that presented less frequency were included to maximize the potential categories that could appear in the resulting bag of words. In the example of the class alzheimer the last two entities share semicolon as the separator between two potential subclasses.

The next step is to set the variable as all uppercase or lowercase for the training of Machine Learning models: the sensitivity of these tools is high enough to present different results between texts with only one type of font size and texts with different type of font size.

Having achieved both, removing unnecessary spaces is the next step as Machine Learning models are sensitive to the characters in strings, including the empty character.

```
def bolsa_sin_comas(x):
    """
    Argumentos clave:
    x    -- objeto tipo lista de string
    Limpia y separa los elementos, genera una bolsa de palabras
    Devuelve: lista de string
    """
    serie_separadas = list(x.apply(remove_diacritics))
    for separador in [',',' ,',', ','y','+',' e ',' de ','-']:
        serie_separadas = separa(serie_separadas,separador)
    mapa_espacios = list(map(espacios_lower,serie_separadas))
    bolsa = list(filter(lambda a: a != '', mapa_espacios))

    return bolsa
```

It is worth to notice that the split function is sensitive enough to need specific positioning of the separators in case they may appear as prefixes or suffixes.

```
def bolsa_sin_comas_puntos(x):
    """
    Argumentos clave:
    x    -- objeto tipo lista de string
    Limpia y separa los elementos, genera una bolsa de palabras
    Devuelve: lista de string
    """
    serie_separadas = list(x.apply(remove_diacritics))
    for separador in [',',' ,',', ','.',' .','. ','y','+',' e ',' de ','-']:
        serie_separadas = separa(serie_separadas,separador)
    mapa_espacios = list(map(espacios_lower,serie_separadas))
    bolsa = list(filter(lambda a: a != '', mapa_espacios))

    return bolsa
```

These last functions make calls to the text cleaning functions on the initial series and creates the bag as a result of separating the words, finally returning a series object.

```
[1]:  1 con retinopatia
      1/2 l/d,
      2 stent,
      3,
      50 paq/ano,
      7,
      a,
      a intervenido,
      a los,
      a los 5 d se inicia fluco,
      aaa,
      abandono segui,
      abceso anal,
      ablacion por radiofrecuencia,
      absce,
      abscesos,
      acalasia,
      accidente cerebrovascular,
      acv,
      acv cardioembolico
      [3773 rows x 1 columns]
```

Figure 3.4: Sample of the elements in the bag of words.

Once the bag of words has been obtained, in order to identify the possible categories underlying the bag, all the repeated elements must be eliminated. For this reason, a filter is applied that maintains only one instance of each chain that appears in the list or bag of words. Another bag of words results from this operation, 20 examples of the result are shown in Figure 3.4 where commas and dots were taken as separators.

3.3 Processing the data

During the process of cleaning the data, a filter would normally be used to eliminate words that have no value in the context of study, either because of typing errors or because they are articles, determiners, connectors or words that do not represent a class themselves. These last type words are called 'stop words'. In this case this filter cannot be applied because by containing diminutives or abbreviations that, as we have explained, make up the dirty categories, the filter could eliminate entities that either identify or refer to a class.

Following what was stated about the similarity between character strings, for the preparation of the data the approach is to achieve, through the use of tools that facilitate the processing, to identify which strings are similar to each other in order to group them with different methods.

3.3.1 String similarity

The tool selected for this task is dirty_cat, a repository that has developed classes and functions to work with non-curated data.

The similarity or distance between chains is measured based on calculations made on n-grams. An n-gram is a subsequence of n elements of a character set. For example, a 3-gram of **absceso** is **abs**. In turn, these strings are identified as tokens, elements that identify a class. This n-gram similarity is based on splitting the two strings to compare in their character n-grams and calculating the Jaccard coefficient between these two sets [11]:

$$sim_{n-gram}(s_i, sj) = J(G(s_i), G(s_j)) = \frac{|Gs_i \cap Gs_j|}{|Gs_i \cap Gs_j|} \tag{3.4}$$

where $G(s)$, is the set of consecutive character n-grams for the string s.

To be able to work with variables in natural language processing, the treatment of tokens by n-grams is relevant. The tokens can identify a class through dictionaries. The creation of tokens is made by cleaning the input data. These tokens that refer to the same element but have a different n-gram structure, require a tool or process that allows obtaining the similarity between any two tokens. The goal is to compute the similarities between each of the entities in the bag with all the other entities and get the feature matrix explained in the section Encoding String Categorical Variables.

This tool creates such matrix. In Figure 3.5, a heat map over the feature matrix of an example containing jobs shows how similar they are. The matrix ought to be read either above or below the diagonal as it is symmetric. It is because all entities are compared against each other, therefore we should ignore the main diagonal, for each entity will always get a 1.0 similarity value with itself, and only read one of the halves as the other will be mirrored.

The entities Property Manager I and Property Manager II are the closest as their similarity reaches up above 0.7. On the other hand Police Cadet and Police Captain do not share such a high value, yet it is noticed that they share similarities with no other entities.

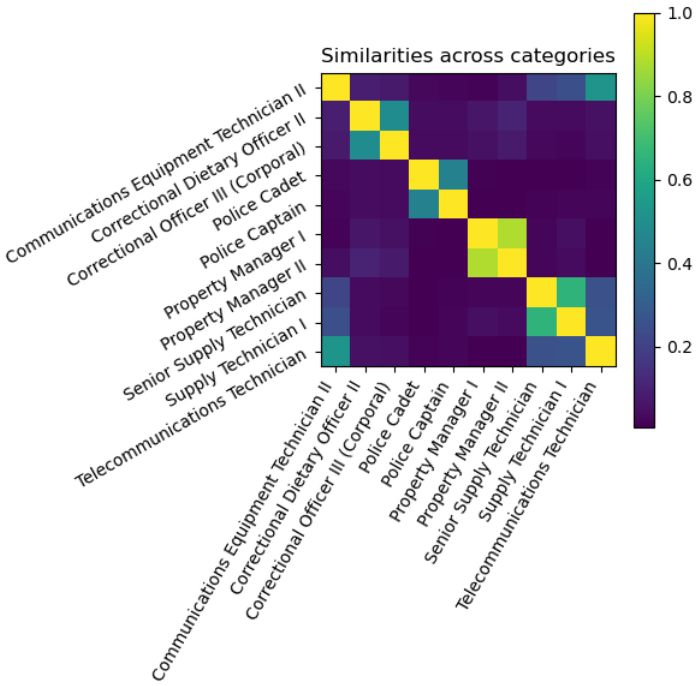


Figure 3.5: Example of a heat map over dirty categories of different works.

The feature matrix in Figure 3.5 is computed using, at first, the bag of words with commas as separators. The presence of a large number of categories calls for representing the relationships between them so the heat map is used to that end.

The results are promising after the evaluation of the first example shown in Figure 3.6, in which the map shows a high relationship value between all the chains that share the word *adenocarcinoma* in a heterogeneous way, such as *adnocarcinoma*, *adenoma* or *adenocea*. Adenocarcinoma is a term that refers to cancer. Although the different types of cancer affect the body in different ways, the result sets a clear guideline to establish *adenocarcinoma* as a class.

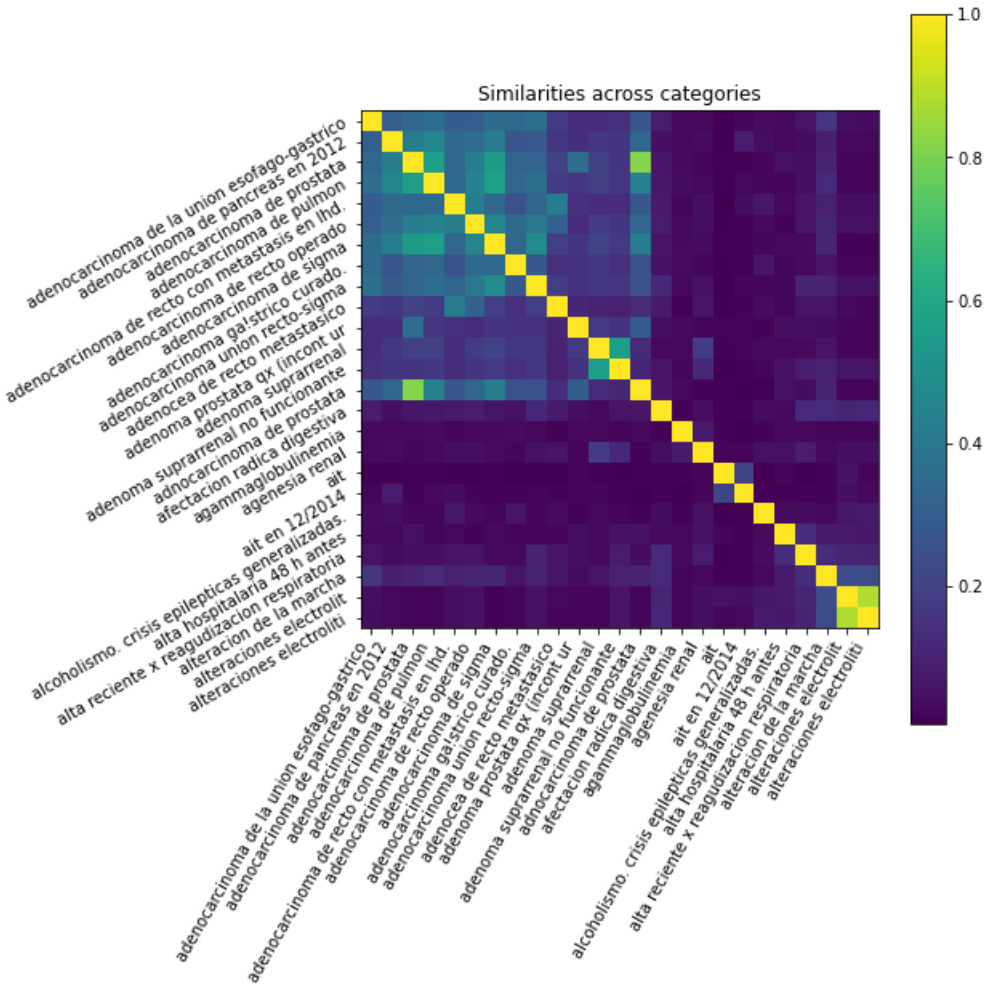


Figure 3.6: Example 1 of a heat map over 25 random dirty categories from the dataset.

However, after evaluating the second example depicted in figure 3.7, a problem appears that will accompany much of the rest of the project: the map does not show a similarity value not even close to 0.4 between chains such as *acv cardioembolico* and *acv con hemiparesia*, both being a type of stroke. One relationship that is striking is the low similarity value of *accidente cerebrovascular* and *acv*. Both chains refer to the exact same concept, being able to form the same class but according to the matrix they show almost no similarity.

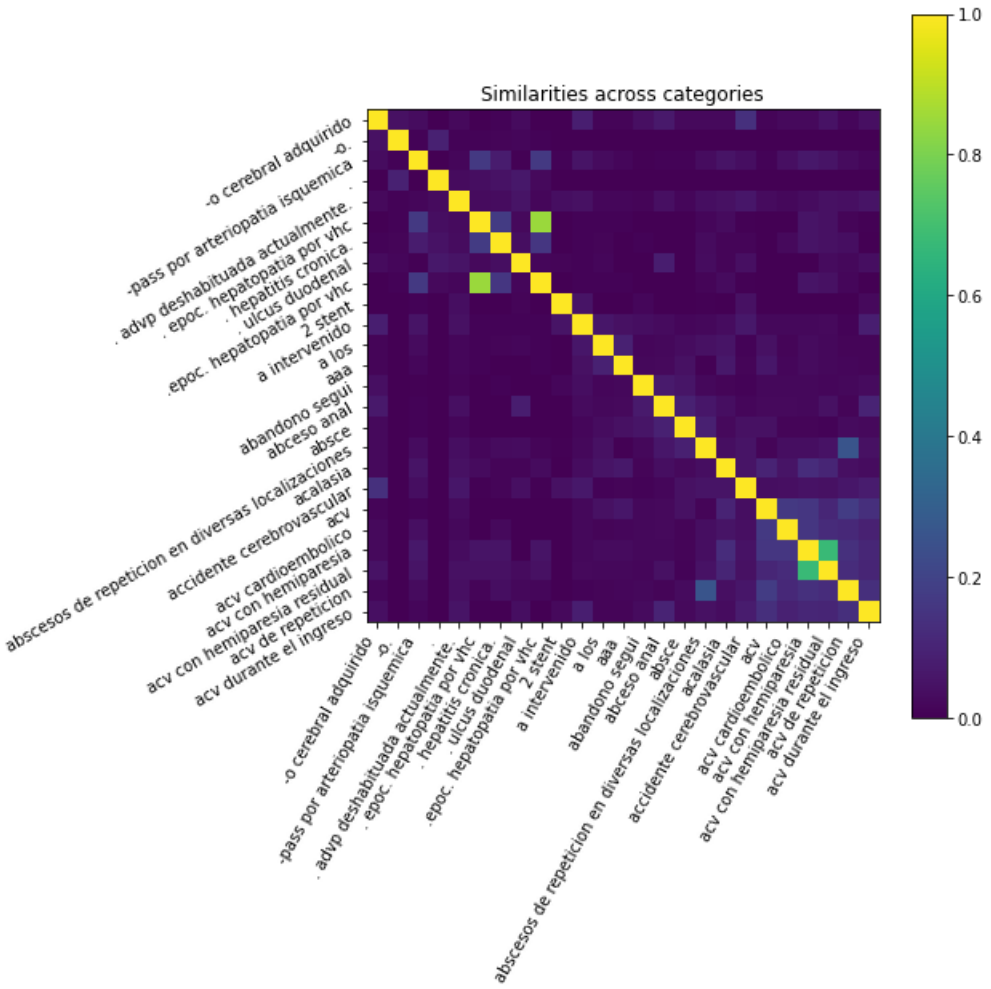


Figure 3.7: Example 2 of a heat map over 25 random dirty categories from the dataset.

3.3.2 K-NN

The K-NN algorithm is a supervised classification method that allows data subsets to be grouped by distance between the elements of each subset. This algorithm classifies each new entry in a class, according to the k neighbors closest to a certain group. To do this, it calculates the distance of the new entry to each data already existing in the model and orders those distances from lowest to highest to choose the group to which it belongs. This means that the algorithm will use the similarity between two strings to measure the distances. The group chosen will be the one that represents the shortest distance, or that group with the greatest representation in a greater longitudinal spectrum.

The number of neighbors of a given example allows adjusting the noise of a classification, including in each class the points of the hyperplane that are alike the most. However, significantly reducing the number of neighbors causes an effect that may not be desired: creating classes of elements that are not necessarily different.

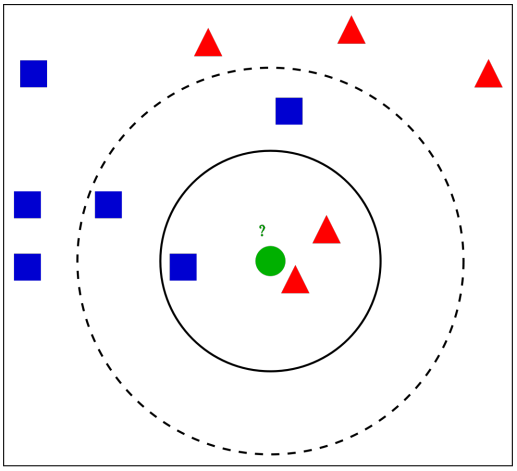


Figure 3.8: Example of the classes grouped relative to the number of neighbors [12]

K-NN is quite sensitive to:

a). The variable k, so that with different values of k we can also obtain very different results.

b). The similarity metric used, since this will strongly influence the closeness relationships that will be established in the algorithm construction process. The distance metric can contain weights that will help us calibrate the classification algorithm, making it, in effect, a custom metric.

Thus, the algorithm is highly benefited by the nature of the dataset with which it is going to work. The matrix already provides a **weighted metric** between each pair of elements of the same.

Fitting a K-NN model

The next step is to fit the model to the dataset and check if there is an estimate that the algorithm can make based on the values obtained in the similarity metric. The input for the value k is chosen to be 4. The objective is to have a glance to whether the algorithm is able to group strings that the heat map did not match but that actually refer to the same concept, that is why there is no investment of resources in finding the optimal number of neighbors.

```
from sklearn.neighbors import NearestNeighbors

nn_c = NearestNeighbors(n_neighbors=4).fit(transformed_values_c)
_, indices_ = nn_c.kneighbors(transformed_values_c[random_points])
indices = np.unique(indices_.squeeze())
```

Figure 3.9 shows of 20 selected items in the bag of words grouped. It is observed that chains such as *artrosis de columna*, *artritis reumatoide* or *posible arteritis de la tem-*

poral are located nearby. This result is promising since the *arthritis* class is the most representative element in both cases and they have been classified as similar.

On the other hand, it is observed in the same graph that elements such as *poliartrosis* or *polinosis* appear as similar elements and, although it is true that they share the prefix *poli*, they have widely different meanings: the first class, polyarthrosis is the definition of inflammation of several joints at the same time while polinois is an allergic disease that affects the eyes, nose and lungs produced as a reaction of the immune system to pollen.

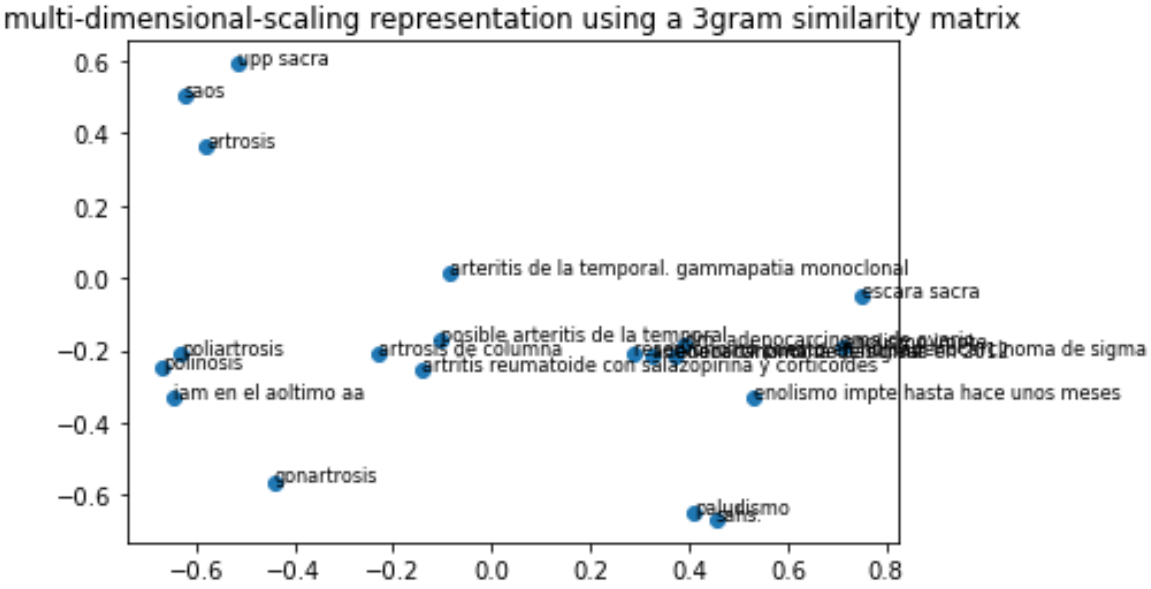


Figure 3.9: K-NN over the bag of words using k=4 neighbors

Conclusion of the method

The intuition behind this evaluation is that for strings with a low number of characters, the string similarity algorithm presents more noise, as the n-grams have greater weight over the total string.

Chapter 4

Categories

4.1 Category recognition

Despite the conclusion shown in the previous section, the elements of the bag of words are evaluated, reaching the conclusion that possibly a large number of examples have similarity values high enough to be able to work with them and obtain a reasonable number of categories that include the highest number of comorbidities exposed in the variable.

In order to face the problem from a Machine Learning point of view, the next step is to be able to identify all the categories using the matrix of values and to be able to merge all the similar strings into the one that identifies the class by adapting the dataset obtained up to now to adjust it to the needs of the study.

This process of adapting the dataset to the problem is manually coded as there is no tool to performs these tasks so customized as it is needed; therefore a series of functions are defined and developed that will be shown for each specific step. Each of the functions went through several tests on the dataset, so the process involved a great investment in time, this section being the bulk of the project

Based on the data obtained so far, the approach that is carried out is the following:

a). Set a relevance threshold for the similarity value.

b). Get rid of as much noise as possible.

c). Identify for each token all those that share a relevant similarity.

d). Set a class for each set of strings.

e). Replace each element in the original variable with its superclass.

4.1.1 Choosing threshold value

The idea of establishing a threshold value for the similarity of the strings is to be able to eliminate the largest number of candidates that do not present a relationship close to the identification as an entity with each of the strings of the bag of words.

This, to begin with, makes it possible to eliminate noise from the dataset by making the categories obtained represent as reliable a reflection as possible of how a professional in the field would interpret and group the dataset.

The value taken by this parameter is significantly relevant for the rest of the process, with the functions and results obtained being sensitive to the increase or decrease of the threshold.

Based on the first result obtained from evaluating the feature matrix from figure 3.6, a threshold value greater than 0.5 suggests that the resulting strings will necessarily be very similar to each other, ensuring a relatively small number of categories and faithful to the groups that make up each of the subsets of strings.

On the other hand, looking at the results of the second heat map on figure 3.7, the choice of such a high threshold value would rule out, in this specific example, all the strings that refer to different types of *acv*.

After assessing it with Óscar, the decision that seems the most appropriate is to establish different values for the threshold and test the execution of the entire process for each of them. Lastly, the results would be evaluated and the most convenient one would be chosen.

Set of threshold values:

0.2	0.3	0.4	0.5	0.65	(4.1)

4.1.2 Noise removal

The denoising process begins by using the matrix itself with the similarity values as the structure to work on.

Since the goal is that all those strings that do not respect the threshold are discarded, the process carried out is similar to applying a binary encoding to the matrix or, analogously, applying a one-hot encoding to each similarity vector in which the elements that meet the condition are kept.

Let x be the vector representing the first row of the matrix, with $x_i, i \in [0..m]$ being m the total size of the bag of words. We should remember that in each row of the matrix the token of the bag of words that represents the index of the row is compared with each of the other strings. We know that, being the first token, the element x_0 has the value of 1 because it is compared with itself.

Taking into account that the similarity values oscillate in a range of [0..1] we apply, having u as the threshold value and defining a function F on the vector such that $F(x_i) = 0, \forall x < u$.

This is the function shown below. Knowing that it is going to work with the elements that meet the condition, the function returns a list with the indexes of its row that have not been at 0.

```
def corte_valores(lista_categorias,matriz_similitud,corte):
    matrix = np.empty([len(matriz_similitud),len(matriz_similitud)])
    """
    Argumentos clave:
    lista_categorias    -- objeto tipo ndarray, contiene
    los valores de la bolsa de palabras
    matriz_similitud    -- objeto tipo ndarray, contiene para
    cada valor su valor de similitud con el resto (0..1)
    corte               -- valor decimal (0..1) para filtrar las palabras
    Pone a cero todos los valores en la matriz por debajo del corte
    Devuelve: lista => para cada categoría las posiciones de las categorías
    cuya similitud es relevante (según corte)
    """

    for i, value in enumerate(matriz_similitud):
        matrix[i] = list(map((lambda x: 0 if x < corte else x),matriz_similitud[i]))

    matriz_indices = []
    for i, value in enumerate(matrix):
        matriz_indices.append(list(np.nonzero(matrix[i])[0]))

    lista_corte = []
    for i, value in enumerate(matriz_indices):
        lista_corte.append([i,0,[j for j in matriz_indices[i]]])

    return lista_corte
```

It turns out that, as each element of the array has been compared with the rest of the elements in the same order, the values of the positions for each of the strings share the same reference, that is, if the element y has in the vector of similar tokens the index 37, this same index will represent the same string regardless of whether it appears in the vector x or z, where x, y, z are entities of the word bag and, thus, have a row assigned in the matrix. This is an advantage because it avoids having to save an object for each of the strings in the word bag, saving resources in time and memory.

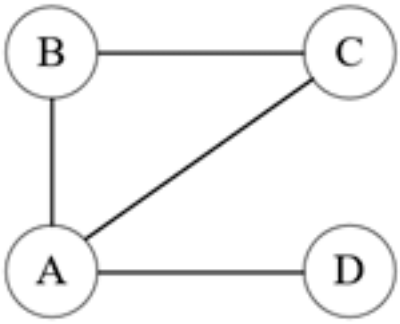
4.1.3 Identifying variants of strings

The next step is to identify all the strings that share similarity and establish a class to group them all.

Having applied the function F over the entire feature matrix, it remains in a state that can be interpreted as an adjacency matrix, giving the possibility of seeing the problem from a graph point of view.

If we establish that a graph is a pair of sets $G = (V, A)$ where V is the set of vertices and A is the set of edges as pairs of the form (u, v) such that $u, v \in V$, we define, then, $\forall u, v \in V$ exists a tuple of the form $(u, v) \in A$ if and only if $u[v] > 0$ or $v[u] > 0$.

The following illustration allows us to see how the matrix would be interpreted, if we consider that every element greater than 0 can be shown as 1.



	A	B	C	D
A	0	1	1	1
B	1	0	1	0
C	1	1	0	0
D	1	0	0	0

Figure 4.1: Example of graph with adjacency matrix
[13]

Taking the vertices and edges, we know that each edge (u, v) fulfills the condition that both u and v have a similarity value greater than or equal to the threshold and, therefore, they can be identified under the same category.

The function corte_valores returned a list of indices in which the values were not 0 after applying the F function on a string. Considering this list A' as an adjacency list, the process to follow is to go through $v, \forall v \in A'$ and mark the node as visited and changing the value of the node to that of the node from which the search starts, that is, updating the index value that heads the adjacency list by the index value that started from.

In the example below, an example adjacency list is considered:

$$\boxed{0} \rightarrow \boxed{0}\boxed{1}\boxed{4}\boxed{6} \tag{4.2}$$

If we follow the algorithm stated above, then we should visit all the nodes until the next result is achieved.

$$\boxed{0} \rightarrow \boxed{1}\boxed{7}\boxed{24}\boxed{146}\boxed{89} \tag{4.3}$$

$$\boxed{0} \rightarrow \boxed{4}\boxed{3} \tag{4.4}$$

$$\boxed{0} \rightarrow \boxed{6}\boxed{11}\boxed{43}\boxed{94}\boxed{159}\boxed{271} \tag{4.5}$$

For each of the nodes in the first adjacency list, they have been visited and changed the head to 0, which is the head of the starting node. It can still be associated to which element the adjacency list belonged as the first element for each token is always the token itself.

Initially, an in depth algorithm was run over each of the nodes of the graph, but string similarity is not a transitive property and it was discarded.

Therefore, if a node has already been visited, its adjacency list is not traversed since it would relate elements in a transitive way and would lead to an incorrect grouping by

classes. The following function shows the process, which returns the updated adjacency lists with the changed list head values. It is noteworthy to note that this process is only executed for those nodes that have more than one node in their adjacency list, since all will have at least one node: themselves.

```
def aplica_categorias(posiciones_equivalencia):
    """
    Devuelve las categorías únicas resultantes de aplicar
    a la bolsa de palabras los valores de similitud con umbral
    """
    for i, lista_similares in enumerate(posiciones_equivalencia):
        posiciones_equivalencia[i][1] = 1
        if len(lista_similares[2]) > 1:
            for pos in lista_similares[2]:
                if (posiciones_equivalencia[pos][1] == 0):
                    posiciones_equivalencia[pos][0] = posiciones_equivalencia[i][0]
                    posiciones_equivalencia[pos][1] = 1
                    #print(valores_cp[pos] + ' ' + str(posiciones_equivalencia[pos][1]))

    return np.array(posiciones_equivalencia,dtype='object')
```

4.1.4 Defining the classes

Once the connections between all the tokens have been established, the categories that will group the rest of the strings are defined, thus dealing with the problem of dirty categories.

Following the process established at the beginning of the section, it is necessary to define a map that allows identifying the category that defines or groups it for each class. With the data structures formed so far, the simplest option to develop is to establish a dictionary where the key is the index of the element to be consulted and the value is the category that groups it. This process is the one that is encoded in the following function, returning said dictionary.

```
def merge_categorias(info_posiciones_corte):
    """
    Argumentos clave:
    posiciones_equivalencia    -- lista de listas, contiene para cada categoría
    las posiciones de sus equivalentes según umbral
    En orden de aparición va etiquetando las categorías que comparten
    similitud como una sola
    Devuelve: lista => para cada categoría la categoría que es -superclase-
    """
    claves = np.arange(len(info_posiciones_corte))
    valores = info_posiciones_corte[:,0]
    diccionario = dict(zip(claves,valores))

    k = list(zip(claves,valores))
    lista_borra = []
    for i, tupla in enumerate(k):
        count = 0
        if (i == tupla[1]):
            for tupla2 in (k):
                if (i == tupla2[1]):
                    count = count + 1
        if count == 1:
            lista_borra.append(i)
    #borramos las que solo tienen una equivalencia ya que la mayoria
    son cadenas mal filtradas

    diccionario = dict(k)

    for i in lista_borra:
        diccionario[i] = -1

    return diccionario
```

In order to correct the process, we use an additional structure in which, for each unique element of the dictionary, we introduce the string it refers to, encoded in the following function, thus facilitating the evaluation of the results prior to substitution in the DataFrame.

4.1.5 Evaluating the results

In this subsection we will treat the results from the execution of the functions over the bag of words for the different threshold values.

The amount of tokens is huge for each of them to be compared, in exchange a token from the bag of words will be evaluated for all the executions. First, the resulting adjacency lists for the variable *acv de repeticion* using commas as separator.

Here are listed all the equivalent tokens for a threshold value equals to **0.2**. Several tokens

that do not refer to the same concept are included, even tokens like *itu de repecition*, which seem to be similar due to sharing the word *repeticion*.

'acv de repeticion':
abscesos de repeticion en diversas localizaciones,
acv de repeticion,
acv repeticion,
acva repeticion,
alzheimer. itus de repeticion.,
asma. neumonias de repeticion.,
broncoaspiraciones de repeticion,
celulitis de repeticion.hip,
cistitis de repeticion,
colangitis de repeticion,
coledocolitiasis de repeticion,
colicos renoureterales de repeticion,
cru de repeticion,
cru repeticion,
desnutricion,
fa. colangitis de repeticion.,
infecciones urinarias de repeticion,
itu de repeticion,
itu repetic,
itu repeticion,
itu repeticion (2/ano),
itus de repeticion,
itus de repeticion. dan alta sin tto atb.,
leishmaniasis cutanea. cistitis de repeticion.,
nefrolitiasis de repeticion,
neumonias de repeticion. cardiopatia isquemica.,
neumonias repeticion,
no fc. dm. fa. enf tromboembolica. itus de repeticion.,
pancreatitis aguda de repeticion,
pancreatitis de repeticion,
pna repeticio,
portador de svpermanente con itu repeticion,
rao de repeticion,
sepsis de origen urinario de repeticion.,
tetraparesia x acvrepeticion,
trombopenia cronica idiopatica. itus de repeticion.,
tvp de repeticion,
volvulo intestinal de repet

Next, all the equivalent tokens for a threshold value equals to **0.3**. We can find less tokens that do not share meaning but still the majority of the list does not relate to the concept of *acv*.

```
'acv de repeticion':
acv de repeticion,
acv repeticion,
acva repeticion,
alzheimer. itus de repeticion.,
asma. neumonias de repeticion.,
broncoaspiraciones de repeticion,
celulitis de repeticion.hip,
cistitis de repeticion,
colangitis de repeticion,
coledocolitiasis de repeticion,
colicos renoureterales de repeticion,
cru de repeticion,
cru repeticion,
fa. colangitis de repeticion.,
infecciones urinarias de repeticion,
itu de repeticion,
itu repeticion,
itu repeticion (2/ano),
itus de repeticion,
nefrolitiasis de repeticion,
neumonias repeticion,
pancreatitis aguda de repeticion,
pancreatitis de repeticion,
pna repeticio,
rao de repeticion,
tetraparesia x acvrepeticion,
tvp de repeticion
```

The following code snippet lists all the equivalent tokens for a threshold value equals to **0.4**, **0.5** and **0.65**.

The same problem that arised with the heat map still remains. With this token there is a clear choice to be made: the threshold value equals to **0.65** gets the cleanest set of tokens similar to the root token.

'acv de repeticion':
acv de repeticion,
acv repeticion,
acva repeticion,
cistitis de repeticion,
colangitis de repeticion,
cru de repeticion,
cru repeticion,
itu de repeticion,
itu repeticion,
itus de repeticion,
nefrolitiasis de repeticion,
pancreatitis de repeticion,
rao de repeticion,
tvp de repeticion

'acv de repeticion':
acv de repeticion,
acv repeticion,
acva repeticion,
cistitis de repeticion,
cru de repeticion,
itu de repeticion,
itus de repeticion,
rao de repeticion,
tvp de repeticion

'acv repeticion':
acv de repeticion,
acv repeticion,
acva repeticion

However, and as it was explained in previous sections, this problem remains for those tokens which root is a small set of characters. If we take, for example, the token *adenocarcinoma de colon* we can see how does the result change. This time, for simplicity, the difference between **0.3** and **0.5** threshold values will be shown.

'adenocarcinma de colon':
adenoca colon mtt,
adenocarcinma de colon,
adenocarcinoma colon,
adenocarcinoma colorectal,
adenocarcinoma con mtx hepaticas,
adenocarcinoma de colon,
adenocarcinoma de colon estadio iv.,
adenocarcinoma de la union esofago-gastrico,
adenocarcinoma de pancreas en 2012,
adenocarcinoma de prostata,
adenocarcinoma de pulmon,
adenocarcinoma de recto con metastasis en lhd.,
adenocarcinoma de recto operado,
adenocarcinoma de sigma,
adnocarcinoma de prostata,
carcinoma de pulmon,
desnutricion. adenocarcinoma de colon con metastasis hep,
dm. adenocarcinoma de ovario.,
dm. iam. adenoca. de colon.,
no pcr. dm. adenocarcinoma de recto

'adenocarcinoma de colon':
adenocarcinma de colon,
adenocarcinoma colon,
adenocarcinoma colorectal,
adenocarcinoma de colon,
adenocarcinoma de colon estadio iv.,
adenocarcinoma de prostata,
adenocarcinoma de pulmon,
adenocarcinoma de sigma

Continuing with the execution of the script, the next results to take into account are

the bags with the categories filtered and grouped. At this moment, the weight of the choice of the threshold value can be observed in the face of the number of categories that appear.

For instance, in the case of the execution with the value of **0.65**, the final number of categories resulted in 1295, while the execution with the values of **0.2** and **0.3** resulted in a total of 307 and 578 categories respectively. The assessment of this result is that none of the final category bags are eligible to continue due to the large cardinality of the dataset. Such a large number of categories would make the Machine Learning model very difficult to work with.

The solution that was carried out was to iterate the entire previous process on the resulting bags, seeking to get a second bag of categories over the first reduced one. This process was only feasible thanks to the dictionary of equivalences between the index of the tokens and the categories that grouped them.

4.2 Iterating the process

The only thing left was to repeat the process, so it was only needed to adjust the data structure of the newest bag of words in order to fit again in the functions defined in the previous section.

```
#Form the new bag of words from the resulting of the first process
nuevos_valores_cp_02 = pd.DataFrame(cats_cp_02)[0].unique()
nuevos_valores_cp_02 = pd.DataFrame([i for i in nuevos_valores_cp_02
if len(i)>2])[0].unique()
nuevos_transformed_values_cp_02 = nuevo_similarity_encoder_cp.fit_transform(
nuevos_valores_cp_02.reshape(-1, 1))
#Apply the threshold
n_pos_corte_cp_02 = corte_valores(nuevos_valores_cp_02,
nuevos_transformed_values_cp_02,0.2)
#Work with the adjacency lists
n_info_categorias_cp_02 = aplica_categorias(n_pos_corte_cp_02)
#Get the dictionary
n_mergeado_cp_02 = merge_categorias(n_info_categorias_cp_02)
#Create the new bag of words
n_cats_cp_02 = obtener_categorias(n_mergeado_cp_02,nuevos_valores_cp_02)
```

The result was not as expected: the bags of words for **0.2** and **0.3** had a number of categories below 100 but in the case of the second bag, which was the most promising, there were several tokens that did not fit properly such as *a los* or *aaa*; and some others that were still dirty categories like *accidentet cerebrovascular*, *acv* and *acv cardioembolico*.

On the other hand, the next bag of words which was holding a value of **0.4** for the threshold had 641 categories. Again, that was not acceptable.

4.3 Substituting categories

The resulting list of categories did not allow to address the problem that had been raised from the beginning without worsening the conditions, since these categories do not suppose reliable information from the point of view of someone who is trying to understand which comorbidities may be more significant while diagnosing bacteremia. Óscar then proposed creating a substitution list of all the categories that we knew were abbreviated from larger strings.

Being aware that the blind spot of the string similarity method was the calculation between strings of short length, the substitution of elements such as *acv* with *accidente cerebrovascular* or *hta* with *hiper tension arterial* was considered as the step to reach a solution.

After providing Óscar the list of substitutions by regular expressions, it was applied to the first bag of words prior to the execution of the data processing functions. Once there, the category recognition process was repeated and the functions were run on the dataset. The result was a bag of words of 175 categories applying all the separators and with a threshold value of **0.3** that, although they were numerous, seemed promising as they did not contain repetitions or a high level of noise.

Once the bag of words was obtained, it was left to perform the substitution of each token in the original variable by the category that corresponded to it. To do this, it was necessary to return to the original variable, apply the same separation filters used to obtain the bag of words and perform the substitution using the dictionary.

4.3.1 Dataset insertion

Machine Learning models work much better with nominal variables. By having replaced the categories, it was possible to break down each of the comorbidities suffered by each patient, but the set cannot be used as input to a model and expect reliable results. To do this, you just have to code the categories under the same criteria and apply it to the DataFrame.

Taking into account that each patient may have more than one comorbidity, the options are:

- Categorize the comorbidities and include for each patient those that suffered during admission.
- Apply one-hot encoding per patient.
- Apply binary coding for comorbidity.

Both the first type and the second are encoding methods that could possibly generate confusion during the learning process of the model. The use of the first would imply that for the variable there would be a list of numbers per patient, each number representing a comorbidity. This list may not be interpreted correctly since learning can be done on the order of the elements instead of the elements themselves. In the same way, the second encoding method, although it is more comprehensible for the model than a list of strings, would generate a vector of 0 and 1 for each category in a patient.

The third option, on the other hand, significantly increases the number of columns in the DataFrame but is much easier to understand for both models and people. Each category will form a new column in which each patient suffering from this comorbidity will have the row at 1.

```
def aniade_categorias_por_columnas(columna_categorias,
    bolsa_categorias,datFrame):
    for i, bar1 in enumerate(bolsa_categorias):
        col = [0 for x in range(len(datFrame))]
        for j, foo1 in enumerate(columna_categorias):
            for foo2 in foo1:
                if (bar1 == foo2):
                    col[j] = 1
        datFrame[bolsa_categorias[i]] = col
    return datFrame
```

4.4 Dataset preparation

Finally, before carrying out the tests with the model, the choice of variables remains. In the study carried out last year, they established which variables were the most relevant for optimizing the results obtained with the models. This includes the elimination of unusable variables, as well as those that can serve as predictors for the model and, as such, must be eliminated.

Table 4.1: Remaining features

Features selected by the algorithm				
Clasifica	edad	sexo	Polimicr	anhonpol
Anaerobio	Hongos	periodo	mes	dia
medio	microbpoli	frasae	frasanae	frasextr
COMORBIL	so	antibiot	diashosp	Especialid
hgb	Inghosp1m	plaqut	stlocal	pmfn
leuc	Inghosp12m	Neoplasia	Hepatopatia	Enfresp
Diabetes	Cardiopatia	Insrenal	Udvp	Alcoholismo
creatin	sedorina	enfbasWeinst	drogadic	inmunosu
esteroid	cirugia	neutrope	glucosa	sintomas
m_digest	m_respir	leucocit	trombope	m_genitu
m_vascul				

Nonetheless there is a modification to be carried out. The feature *COMORBIL* is a variable that indicates whether the patient had any comorbidity at the time of hospitalization. This feature has already no use, each comorbidity will be checked for each of

the patients as a binary variable, therefore $COMORBIL \subset Otrascomor$. The feature is not needed, the information that it brings to the dataset is spread and specified over the different columns for each of the comorbidities. Leaving the variable *COMORBIL* in the dataset would only add redundant data and one more dimension to the dataset features, so it is removed.

Chapter 5

Bias and variance

Supervised Machine Learning algorithms require a large volume of data that will be used during the training and testing or validation phases. The training phase in Machine Learning algorithms is where the algorithms find relationships or correlations in the data, depending on the problem and model, among the data or between the data introduced and the output expected. The testing phase consists on supplying a smaller set of data to check whether the predictions are accurate for each of the cases. It is used to measure the accuracy of the model.

The procedure described above is called inductive learning. The induction capacity of a model determines the level of precision that an algorithm has when trying to solve a problem similar to those provided as an example. The goal of any Machine Learning algorithm is to deduct the training data well to any domain of the problem. The purpose of the technique is to predict future actions on never-before-seen data.

The main causes of a model with unreliable accuracy are called overfitting or high variance and underfitting or high bias. As the name says, overfitting is produced when, during the learning phase, the model adjusts the weighs too tight for the training set introduced. The easiest way to recognize this situation is when a model has very good accuracy levels with the training data but surprisingly poor accuracy rates with the test data. This is usually due to the use of small datasets.

On the contrary, underfitting does not achieve high accuracy rates neither with the training nor the test datasets. This can be produced due to large sets of data during the training phase and short periods of training. The algorithms then lack of time for adjusting properly the weighs and results in a generic model where no result is ”too good” nor ”too bad”.

Figure 5.1: Bias and Variance
[14]

5.1 Avoiding Bias and Variance

Overfitting and underfitting directly affect the accuracy and reliability of the models we are working with. It is important to avoid this situations by testing the data we are going to work with [15].

- Ensure that we have a sufficient number of samples to both train the model and validate it.
- Subdivide our data set and keep a portion of it to test the model. This will allow us to evaluate the performance of the algorithm and will also allow us to easily detect the effects of overfitting or underfitting.
- Make sure that the test set is large enough to yield statistically meaningful results and is representative of the data set as a whole. In other words, do not pick a test set with different characteristics than the training set.
- The excessive number of attributes should be avoided, since it would generate a large number of dimensions in our model. This is because each attribute makes up one dimension of the model's sample space. The number of dimensions of the sample space must be proportional to the number of cases available to carry out the study, that is, the greater the number of attributes, the greater the number of case studies, or vice versa, in the case of having few case studies few attributes should be used.

In this project, we are using the dataset left from the study carried the previous year, which has the advantages of normalized data, missing data treatment and attribute filtering.

Figure 5.2 illustrates how each of the situations described above influence the decisions over a set of data and how should a model behave.

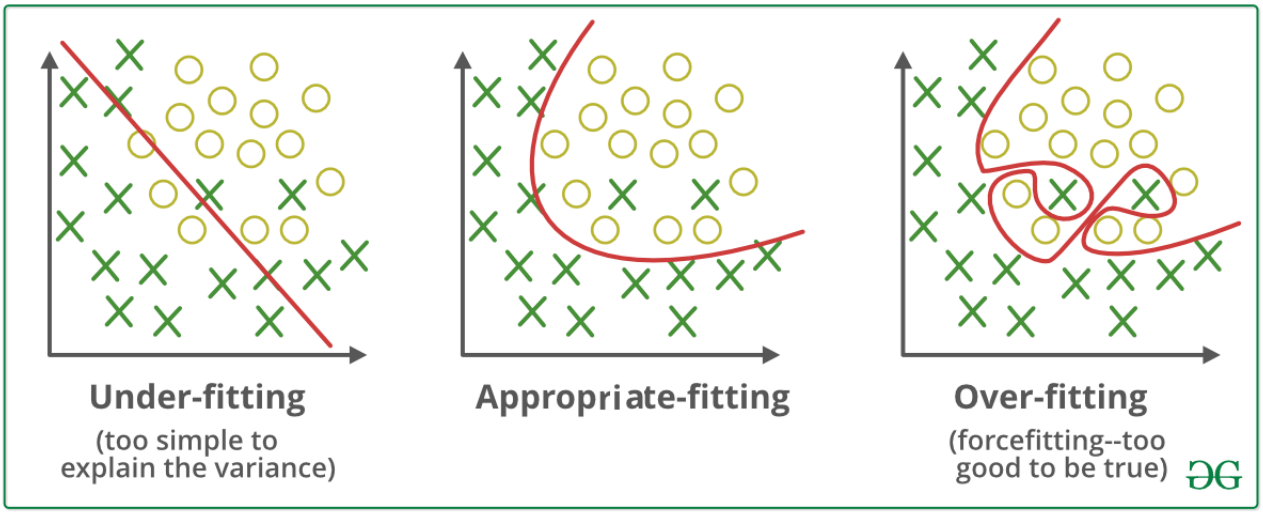


Figure 5.2: Examples of overfitting and underfitting [16]

5.2 K-Fold Cross Validation

Cross-validation is a resampling procedure used to evaluate Machine Learning models on a limited data sample. It allows generating different models from the same dataset. The technique divides into different subsets from the original set and generates a model so that each subset of data is used for both the training part and the validation part. More specifically, it randomly mixes the dataframe and subdivides it into equal groups.

Cross-validation is primarily used in applied Machine Learning to estimate the skill of a Machine Learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups.
- For each unique group:
  - Take the group as a hold out or test dataset.
  - Take the remaining groups as a training dataset.
  - Fit a model on the training set and evaluate it on the test set
  - Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

K-Fold Cross validation is one of many processes used to that end, but is what will be used in this study. Figure 5.3 illustrates the process.

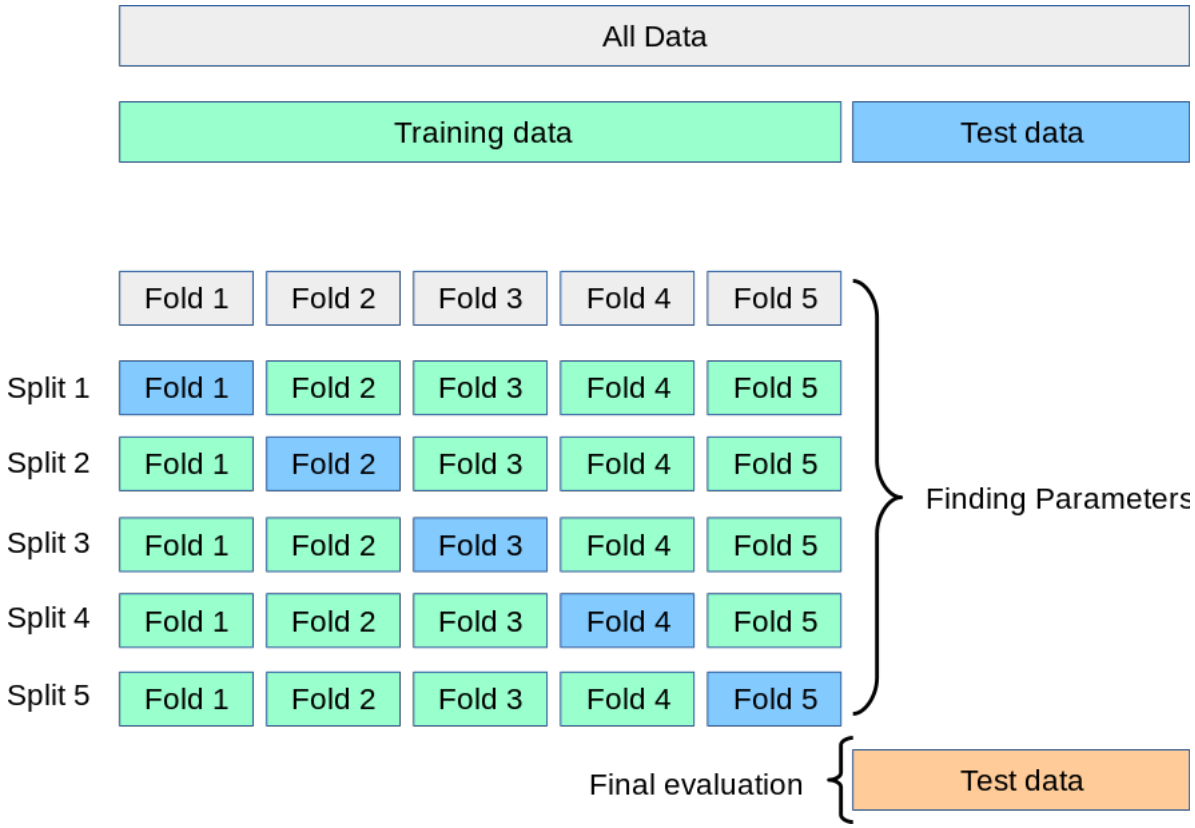


Figure 5.3: Examples of overfitting and underfitting
[17]

Chapter 6

Random Forest

The Random Forest algorithm is a supervised learning technique that includes different methods in the training phase.

It is a model frequently used to deal with overfitting and underfitting problems. This algorithm is used for solving regression and classification problems. It has a correct operation even without adjusting its own parameters and remains stable when new data is entered. On the other hand, it requires high processing times, it is difficult to interpret and small data frames are not processed optimally.

A Random Forest is an ensemble of decision trees combined with bagging. When using *bagging* [18], what is actually happening is that different trees see different portions of the data. The low correlation between models (trees) is the key. The reason for this effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). No tree sees all the training data. This causes each tree to be trained with different data samples for the same problem. In this way, when combining their results, some errors are compensated for others and we have a prediction that generalizes better.

When we using bagging, we also combine various Machine Learning models. Unlike other methods, the way to get errors to compensate for each other is that each model is trained with subsets of the training set. These subsets are formed by randomly choosing samples (with repetition) from the training set.

6.1 Fitting a Random Forest

To adjust the model based on this classifier, it is necessary to adjust the **n_estimators** parameter. This parameter represents the number of trees that will make up the model and on which each case study will be evaluated. In addition, the *random_state* parameter is set to be able to replicate the accuracy values of the model with the same input parameters.

This parameter helps to control the randomness of the algorithm when generating the decision trees. It is important while changing between datasets, parameters on the same dataset or reevaluating models.

First, to avoid overfitting and underfitting, we will split the data using 80% of the data for the training process and the remaining 20% for the model validation. Taking into account the amount of samples in the dataset and the amount of attributes for each sample, the 80-20 folding distribution leaves room for a correct fitting process.

```
from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test =  train_test_split(X, Y, test_size=0.2)
```

Once we have applied *K-Fold Cross Validation*, we can execute the code that will adjust the number of estimators to minimize the error. A range between 20 and 90 estimators is fixed. Now the model has to train using the training data from the *Cross Validation* method and then calculate the accuracy with the sets of data used for the test phase.

```
arrayPred = []
for ind in range(20,90):
    print(ind, end = ' ')
    RF = RandomForestClassifier(ind, random_state=0)
    RF.fit(X_Train, Y_Train)
    predicciones = RF.predict(X_Test)
    accuracy = accuracy_score(Y_Test, predicciones)
    arrayPred.append(accuracy)

    if accuracy > maxi_accuracy:
        maxi = ind
        maxi_accuracy = accuracy
        rf_max= RF
```

In Figure 6.1 we can see that overall there are very decent accuracy values, but the peak is reached for **54** estimators (the highest value in the record is placed in position 34, as the sample goes from 20 to 90 the total number of estimators is 34+20), a random_state value of **0** with an accuracy of over **0.940**. For this execution, *K-Fold* partition with 80% for training and 20% for testing was used. The results are shown for the training phase.

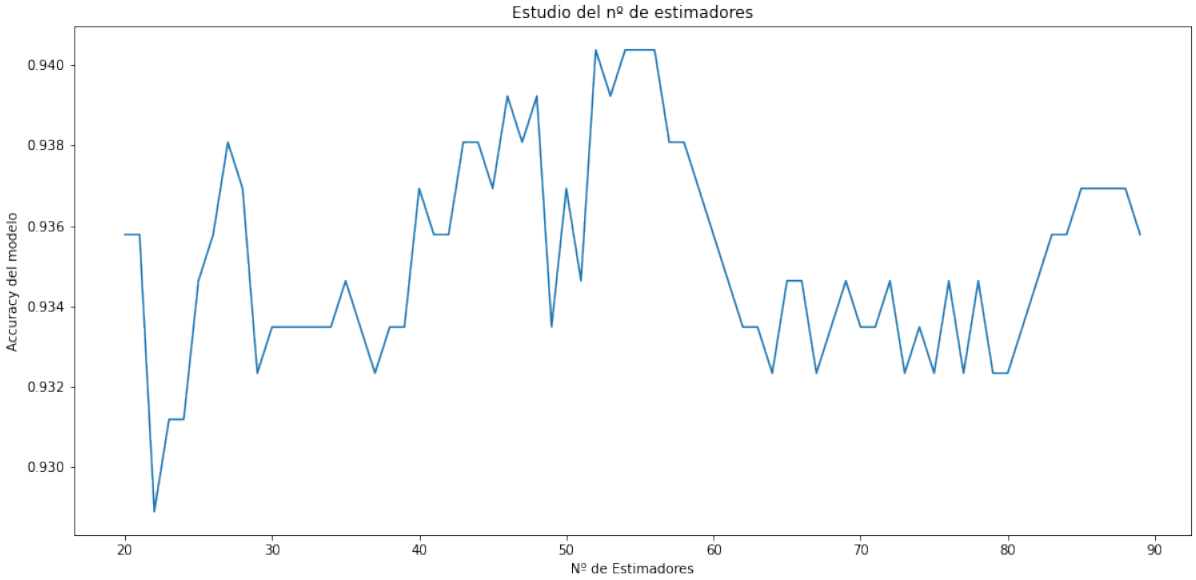


Figure 6.1: Study over the number of estimators

Once the returned values are known, we validate the model and the returned accuracy value in order to detect overfitting or underfitting problems. To do this, the predictions for a new dataset not used during the training phase and the accuracy of the model on this dataset are calculated. We use now the validation set.

```
#run model with optimal parameters
RF = RandomForestClassifier(54, random_state=0)
RF = rf_max
RF.fit(X_Train, Y_Train)

#test
predicciones = RF.predict(X_Test)
report = pd.DataFrame()
reporte_actual = classification_report(Y_Test, predicciones, output_dict=True)
report['1'] = reporte_actual['1'].values()
```

Model accuracy is: 0.935

In this case, slightly lower accuracy values are obtained than those previously collected, which indicates that the model does not present overfitting or underfitting problems.

6.2 Evaluation metrics

In order to fully understand the degree of precision of the model in question, there are various interpretability techniques for Machine Learning models in classification problems as there can be produced different outcomes. In this example, the predicted value can

either be **true** or **false**, meaning that a patient is diagnosed bacteremia or not. However that prediction might not be true for all the cases. This falls for hypothesis testing, where a property is to be evaluated whether is compatible with what is observed in a sample of that population.

In binary classification a **false positive** is an error in which a test result incorrectly indicates the presence of a condition such as a disease when the disease is not present, while a **false negative** is the opposite error where the test result incorrectly fails to indicate the absence of a condition when it is present. Also, the predictor can accurately label the income data, in this case the labels are **true positive** if the prediction indicates the presence of a condition when it is present, or **true negative** when the predicted value correctly indicates the absence of the condition. This concept is illustrated in Figure 6.2.

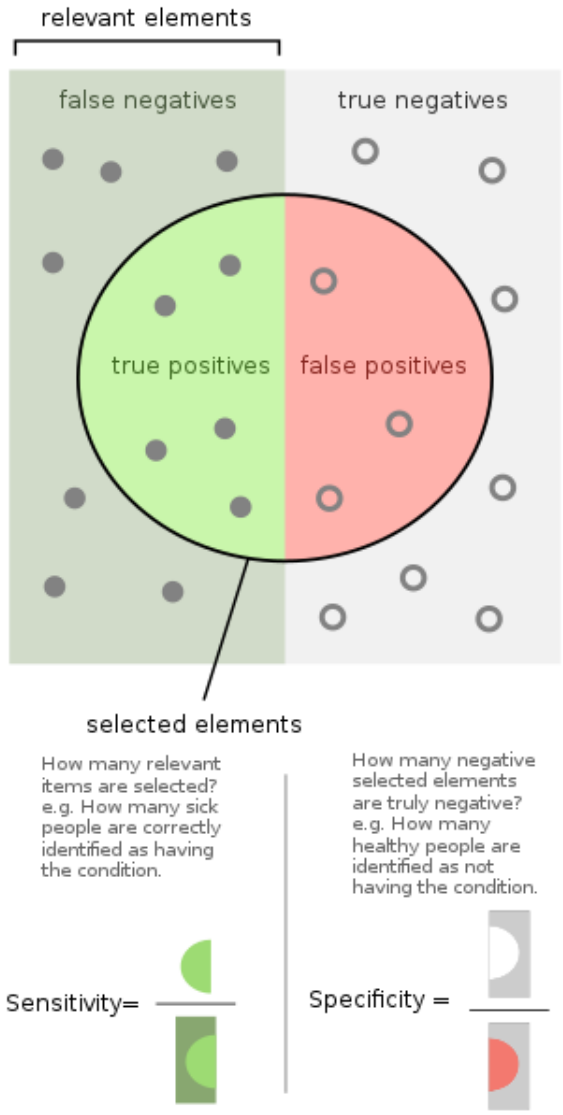


Figure 6.2: Hypothesis testing
[19]

Over the hypothesis testing attributes, we can measure the accuracy of the model by

some simple calculations. The following four metrics stand out:

- **PPV** (also called Positive Predictive Value or Precision) is the fraction of relevant instances among the retrieved instances.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{6.1}$$

- **Recall** (also known as Sensitivity) is the fraction of relevant instances that were retrieved. Both precision and recall are therefore based on relevance.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{6.2}$$

- **NPV** (also known as Negative Predictive Value) is the probability that, in this example, subjects with a negative predictive result truly do not have the disease.

$$\text{NPV} = \frac{\text{True Negative}}{\text{False Positives} + \text{True Negatives}} \tag{6.3}$$

- F-score or F-measure is a measure of a test's accuracy. The **F1-score** is the harmonic mean of the precision and recall.

$$\text{F1} - \text{score} = \frac{2 * \text{Recall} + \text{Precision}}{\text{Recall} + \text{Precision}} \tag{6.4}$$

- **Support** is the number of entries of each class within the data group intended to validate the model.

The model provides the next information for each of the predicted classes:

```
'1': {'precision': 0.9222462203023758, 'recall': 0.963882618510158,
'f1-score': 0.9426048565121412, 'support': 443}

'3': {'precision': 0.960880195599022, 'recall': 0.916083916083916,
'f1-score': 0.9379474940334128, 'support': 429}
```

6.2.1 Confusion matrix

The confusion matrix is a performance measurement graph for Machine Learning classification problem where output can be two or more classes [20]. It is a table with 4 different combinations of predicted and actual values. Those combinations refer to the hypothesis testing labels explained in Section 6.2. Figure 6.3 illustrates how the the number of patients detected with bacteremia correctly (upper left quadrant) is much higher than those detected with bacteremia incorrectly. Patients in whom bacteremia has not been detected correctly (lower right quadrant) are also superior to those who have not been detected incorrectly (lower left quadrant), but with a lower success rate.

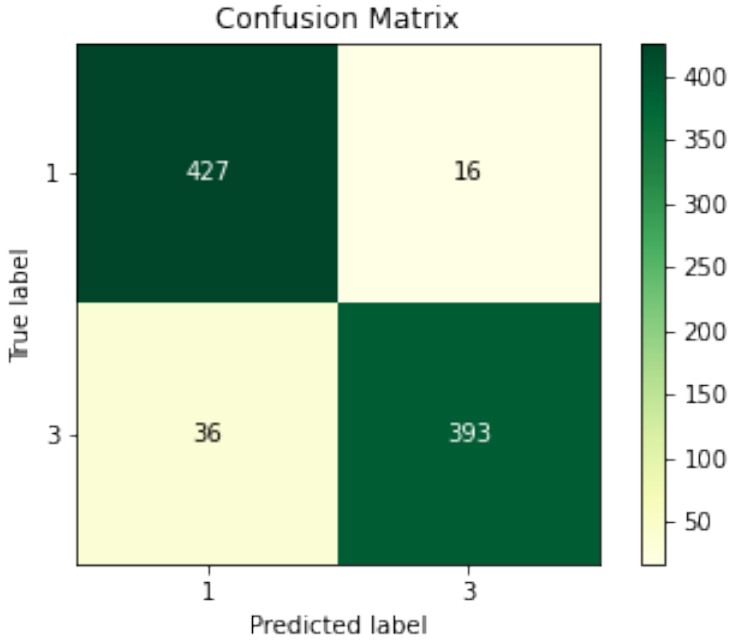


Figure 6.3: Confusion matrix

6.2.2 ROC Graph

When we need to check or visualize the performance of the multi-class classification problem, we use the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics).

AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease [21].

The performance of a model can, therfore, be speculated based on what the ROC curve shows. An excellent model has AUC near to the 1 which means it has a good measure of separability. A poor model has an AUC value close to 0.5, it means the model has no class separation capacity whatsoever. Let us evaluate some examples.

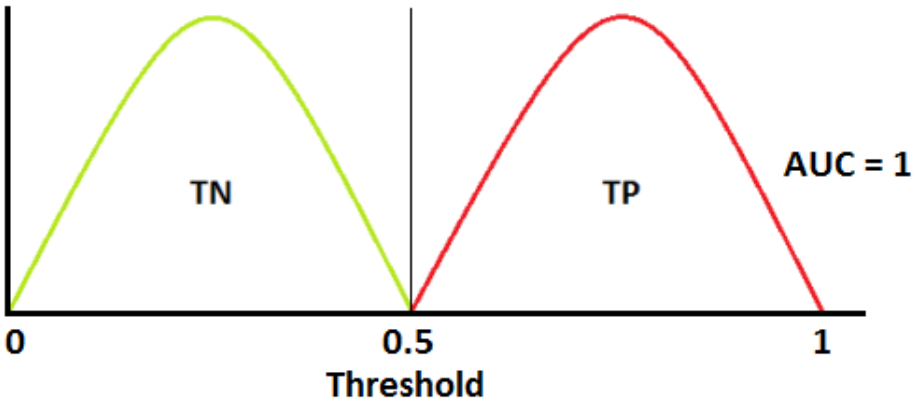


Figure 6.4: Ideal AUC scenario
[22]

Figure 6.4 shows a case scenario where the two distributions do not overlap at all. This means the model will simply be able to distinguish between each of the possible classes.

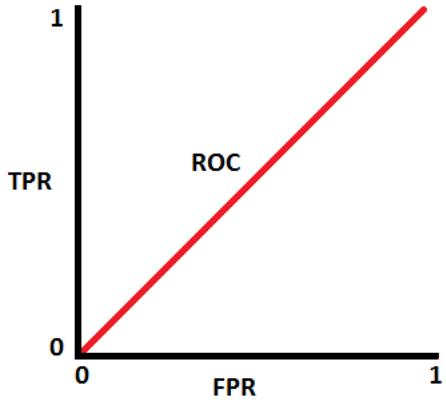


Figure 6.5: Worst ROC scenario
[23]

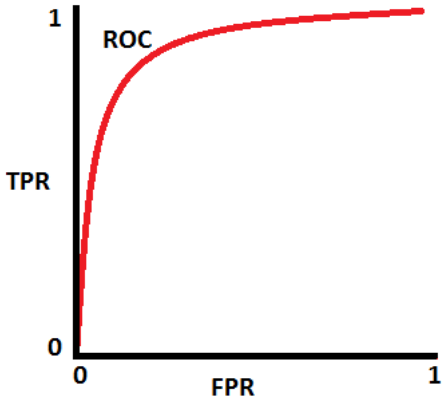


Figure 6.6: Ideal ROC scenario
[24]

Figure 6.5 shows the scenarios for the ROC curve where the AUC value is close to 0.5. This situation is far from ideal since it would be as if you toss a coin each time you want to classify a certain sample. On the other hand, Figure 6.6 depicts an scenario where the model is almost perfectly able to correctly classify all the incoming data. It is the case where the AUC value approaches 1.

Now if we look at the ROC curve of our model in Figure 6.7, we can speculate that the model do really identify and classify correctly most of the samples. The AUC value is 0.97 which allows us to know that the two distributions do barely overlap.

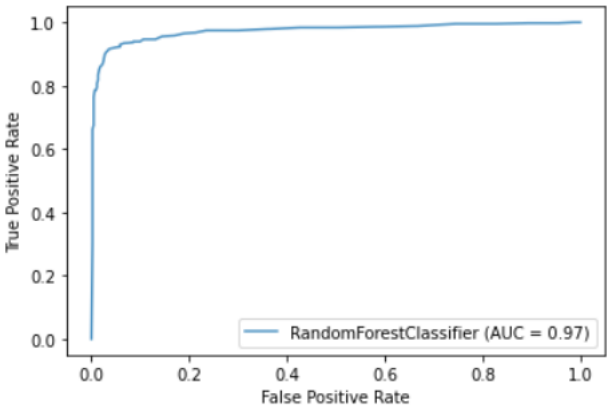


Figure 6.7: ROC curve from Random Forest

6.3 Weighing the features

Finally, it remains to evaluate which variables have the greatest relevance when making the decision to predict whether a patient suffers from bacteremia or not. In this section a couple of measurements will be shown to illustrate the most weighed attributes.

Weight	Feature
0.1601 ± 0.2411	x53
0.1333 ± 0.2479	x65
0.0498 ± 0.0907	x11
0.0330 ± 0.0402	x0
0.0302 ± 0.1294	x31
0.0237 ± 0.0805	x16
0.0216 ± 0.0955	x27
0.0213 ± 0.0298	x13
0.0206 ± 0.0338	x5
0.0193 ± 0.1032	x34
0.0190 ± 0.0379	x21
0.0175 ± 0.0445	x56
0.0167 ± 0.0941	x32
0.0167 ± 0.0368	x17
0.0165 ± 0.1062	x36
0.0163 ± 0.0603	x52
0.0161 ± 0.0115	x3
0.0158 ± 0.0369	x22
0.0155 ± 0.0936	x38
0.0152 ± 0.0900	x33
... 223 more ...	

Figure 6.8: Estimated weigh per feature

	importance
adquisic	0.160088
origensos	0.133346
medio	0.049807
periodo	0.032965
Diabetes	0.030235
Urea	0.023679
sedorina	0.021641
frasanae	0.021323
Prifrpos	0.020552
Neoplasia	0.019350
pmfn	0.019038
tad	0.017534
Cardiopatia	0.016715
creatin	0.016654
Hepatopatia	0.016503
Especialid	0.016323
edad	0.016125
plaqut	0.015769
Alcoholismo	0.015486

Figure 6.9: Most weighed features

Figures 6.8 and 6.9 show the features with the most relevance when making the predictions. If we evaluate them we can find *adquisic* which is the adquisition of the infection, *origensos* as the suspected origin of the bacteremia at the time of the blood culture drawn, *medio* growth medium of true positive, but also *Diabetes* or *plaqut* which measures the platelets in the bloodstream.

Neither in the top 20 features by relevance, which are shown in Figure 6.9, nor in the top 50 features are there any of the categories resulting from the process of category

recognition in Section Category recognition. Stands out that all most relevant features were also part of the previous study, yet the results have improved notably. This result does not go unnoticed and we will investigate it in the next section.

Another graph to understand how the features affect in the decission making processs is the SHAP graph. This value measures the positive and negative relationships of the predictors with the target variable. In Figure 6.10 the contribution of each of the variables on the decision making while labeling the variables is painted in red and blue color.

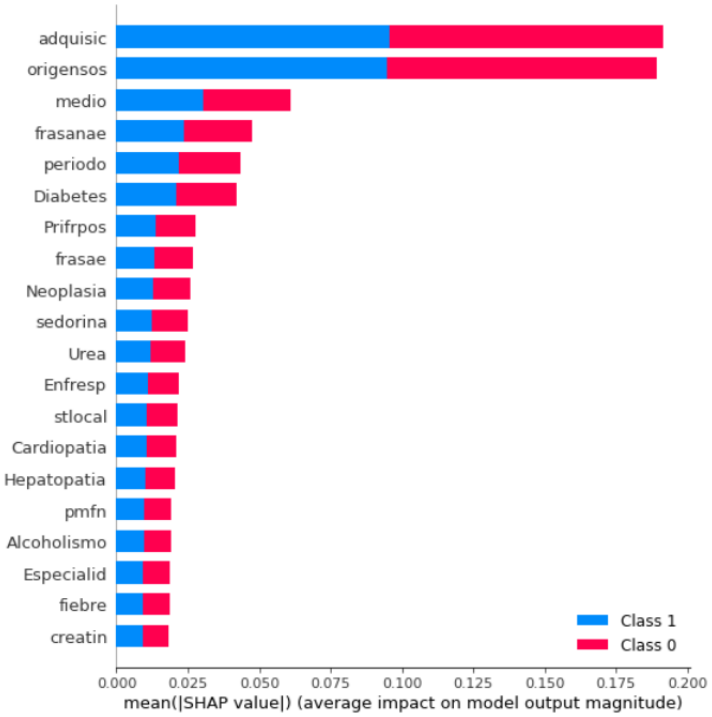


Figure 6.10: SHAP value (impact on model output)

To understand this graph we should know that each of the colors represent one class, as well as the value for that observation. Let us take, for example, the feature *Diabetes*, which is a binary variable. The graph shows that for the records where the patients have diabetes diagnosed, the majority of them were diagnosed bacteremia too. As it can be seen, this graph helps understand which features are relevant and how do they interfere in the classification for each of the classes.

6.4 Feature filtering

After noticing that no class from *Otrascomor* was part of the most relevant features, we found out in an article [7] that some features acted as predictors due to their nature. Those features are *origensos* that is the suspected origin of the bacteremia and *adquisic*, as the acquisition of the bacteremia, therefore a predictor.

We decided to remove those features from the dataset and, before running again the script and fitting the Random Forest we found out that there were another two features that belonged to the part of the process of bacteremia diagnosis where the blood cultures

had already been drawn. The features were *frasae* and *frasanae* and were also removed from the dataset.

In Figure 6.11 we can see the results of the training process of the Random Forest. The model accuracy value was **0.871** after testing the model with the validation set.

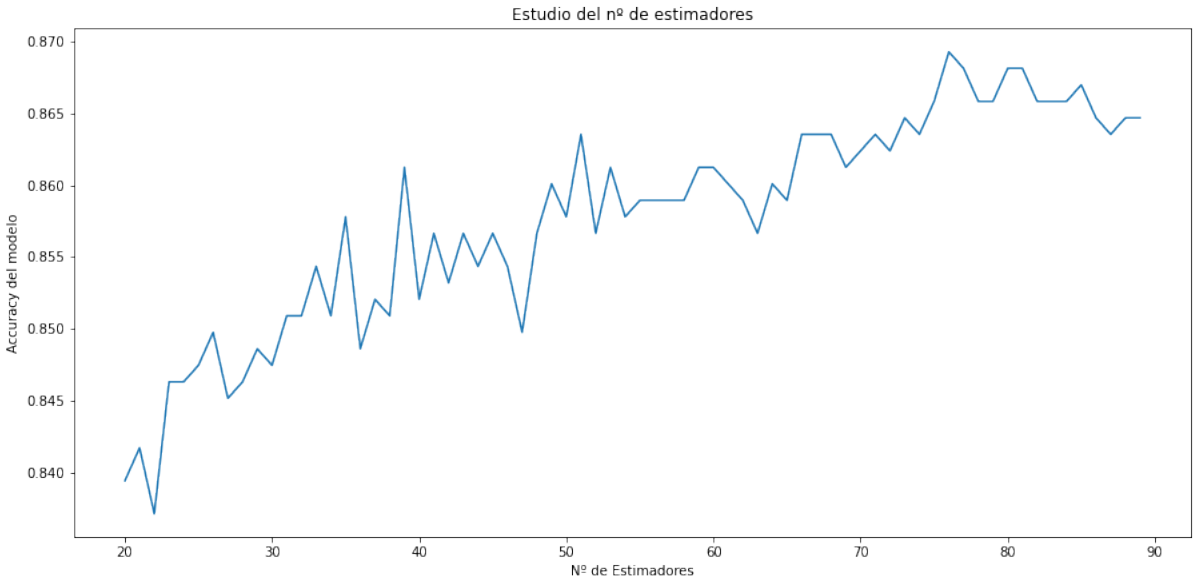


Figure 6.11: Study over the number of estimators with filtered features

This result is far more coherent taking into account the results from the previous study that scored an accuracy of **0.86** and the list of most relevant variables, now updated in Figure 6.12, where none of the new features represent a significant decision value and therefore our model should behave almost the same.

	importance
medio	0.088712
periodo	0.054371
Neoplasia	0.031847
pmfn	0.029728
Prifrpos	0.029626
plaqut	0.028439
glucosa	0.027744
primtemp	0.027269
creatin	0.027148
leuc	0.026055
edad	0.025871
hgb	0.023054
Urea	0.023002
sedorina	0.022935
Especialid	0.022674
dia	0.022216
stlocal	0.022045
Alcoholismo	0.021747
tad	0.020977
pcr	0.020593

Figure 6.12: Most relevant features in the second execution

Chapter 7

Conclusions

Working with such a heterogeneous dataset has been a real challenge for the entire dataset preparation process. The first conclusion is that the whole study and, in general, any process that is repeated similar to this, would greatly benefit from a more specific description of the text strings. Although it is a work that remains for the writing of doctors, we have seen how the substitution of, for example, stroke for cerebrovascular accident, allows the grouping and identification of categories in a much more reliable and exact way. As explained in the section on string similarity, misspellings were relevant when comparing short-length tokens, so another advantage of this situation would be the decrease in the relevance of misspellings in obtaining of the similarity between two strings.

Using the file provided by Óscar has allowed the convergence of all the cleaning and data preparation work into a useful result. This file, even so, does not collect all the variants of abbreviations or spelling mistakes that the set of variables contains. This leads to the conclusion that the tool alone did not provide sufficient support to carry out the study.

In short, is noticeable that natural language processing is a task that still has room for improvement, however the use of this tool has made it possible to reduce a set of more than 1000 categories to approximately one tenth.

Looking at the final dataset, the amount of attributes initially added by using a binary encoding to include the categories in the dataset seemed excessive, increasing the total number of attributes to 247, since it is a perfect example for a case that may suffer high bias and generalize the weights associated with the parameters in the learning process. However, since there were more than 4,300 examples, not only has it not produced this effect, but it has also contributed to an improvement in the success rate compared to the study carried out last year.

Regarding the model used, the conclusion drawn from the previous study showed a better success rate with the application of Random Forest on the dataset. The advantage of using this model over other Machine Learning models is that it is one of the models with more explanatory power, as it is directly formed by decision trees. The model resulting from this algorithm trained on the previously mentioned dataset has an accuracy of approximately 94%.

One of the reasons that the fitness of the first model can be considered successful is the

result of displaying the ROC curve (6.7) and the confusion matrix (6.3). The conclusion of the correct classification by the model is supported by the metrics that allow us to appreciate the clear differentiation between both distributions.

We can not ignore that, after all, the process of cleaning the data, computing the similarity between all the tokens and finally obtaining the different categories from the feature *Otrascomor*, none of them were in the top 50 of the most weighed categories. The conclusion is that, regarding the result of the second model, with the data provided and the Machine Learning type of model used, it is rather difficult to make a correct early diagnosis of bacteremia.

Chapter 8

Future improvements

This project began with the idea of working on all the nominal variables of the dataset provided by Hospital Universitario de Fuenlabrada. However, the organization and work that has led to the treatment of a single variable has taken up most of this study.

For this reason, one of the possible branches in which this study can lead is the one in which the rest of the variables are treated and Machine Learning methods are applied to the complemented set. Taking into account the dataset resulting from this study, it is possible that the inclusion of the other variables may overload the number of attributes for the set provided when coding them for correct treatment by Machine Learning models, yet it seems the most urgent step to take regarding the results of the last model.

Other possible variants are the application of other Machine Learning models such as Neural Networks or Support Vector Machines (SVM). These models have the great disadvantage of lacking explicability. Therefore, it would require an in-depth study of the variables and the weights associated with them. However, they are models that can detect complex relationships between attributes that help improve prediction efficiency. This is specially relevant in the context of this project, where it would also be interesting to find possible relationships between the categories obtained and the original variables from the dataset.

In addition, the study carried out last year and this one that complements it, provide conclusions and help to recognize bacteremia as a binary class, but in reality there are a diversity of types of bacteremia and not all share symptoms or treatment. Therefore, the conclusions drawn during these two years could be applied to the study of the detection of bacteremia as a multiclass classification problem. This particular study may be the most laborious, since it may require different datasets for each type of bacteraemia, however, if the results were successful, it would be a great boost in the prediction of bacteremia diagnoses.

List of Figures

1.1 Blood culture examples 2

3.1 Sample of the feature *Otrascomor*. 12
3.2 Sample of the elements in the feature *Otrascomor* refering to anemia as dirty categories. 13
3.3 Sample of the elements in the feature *Otrascomor* refering to alzheimer as dirty categories. 13
3.4 Sample of the elements in the bag of words. 18
3.5 Example of a heat map over dirty categories of different works. 20
3.6 Example 1 of a heat map over 25 random dirty categories from the dataset. 21
3.7 Example 2 of a heat map over 25 random dirty categories from the dataset. 22
3.8 Example of the classes grouped relative to the number of neighbors . . . 23
3.9 K-NN over the bag of words using k=4 neighbors 24

4.1 Example of graph with adjacency matrix 28

5.1 Bias and Variance 40
5.2 Examples of overfitting and underfitting 41
5.3 Examples of overfitting and underfitting 42

6.1 Study over the number of estimators 45
6.2 Hypothesis testing 46
6.3 Confusion matrix 48
6.4 Ideal AUC scenario 49
6.5 Worst ROC scenario 49
6.6 Ideal ROC scenario 49
6.7 ROC curve from Random Forest 50
6.8 Estimated weigh per feature 50
6.9 Most weighed features 50
6.10 SHAP value (impact on model output) 51
6.11 Study over the number of estimators with filtered features 52
6.12 Most relevant features in the second execution 53

List of Tables

3.1 Features from the dataset 8
3.2 Selected features in the study. 11

4.1 Remaining features 36

Bibliography

[1] "Blood cultures examples." https://lh3.googleusercontent.com/proxy/ahAkfK3-tOUDt-x2hAsWmf7s9i2sUeu6Gsr6dRLeHIpuegTu3mIUJIAQ3CyiTBRQe8ORCmbOpz

[2] "Outcome of patients with sepsis and septic shock after icu treatment." https://pubmed.ncbi.nlm.nih.gov/9627170/.

[3] "The complex pathogenesis of bacteremia - from antimicrobial clearance mechanisms to the genetic background of the host." https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3916384/.

[4] "Attributable mortality rate for carbapenem-resistant klebsiella pneumoniae bacteremia." https://www.cambridge.org/core/journals/infection-control-and-hospital-epidemiology/article/abs/attributable-mortality-rate-for-carbapenemresistant-klebsiella-pneumoniae-bacteremia/2AD70DA5BBF04419903E384B54081F79.

[5] "Time to positivity of blood cultures supports early re-evaluation of empiric broad-spectrum antimicrobial therapy." https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6314566/.

[6] "The incidence and prognosis of patients with bacteremia." https://pubmed.ncbi.nlm.nih.gov/26183054/.

[7] "Diagnosing hospital bacteraemia in the framework of predictive, preventive and personalised medicine using electronic health records and machine learning classifiers." https://link.springer.com/article/10.1007/s13167-021-00252-3.

[8] "Yufeng ding, jeffrey s. simonoff. an investigation of missing data methods for classification trees applied to binary response data." https://www.jmlr.org/papers/v11/ding10a.html.

[9] "Patricio cerda, gaël varoquaux, balázs kégl. encoding high-cardinality string categorical variables." https://arxiv.org/ct?url=https%3A%2F%2Fdx.doi.org%2F10.1109%2FTKDE.2020.2992529&v=aa8e6d50.

[10] "Patricio cerda, gaël varoquaux, balázs kégl. similarity encoding for learning with dirty categorical variables. machine learning, springer verlag, 2018,." https://dx.doi.org/10.1007/s10994-018-5724-2.

[11] "R. c. angell, g. e. freund, and p. willett, "automatic spelling cor- rection using a trigram similarity measure," information processing management, vol. 19, no. 4, pp. 255–261, 1983."

[12] "K-nn example." https://upload.wikimedia.org/wikipedia/commons/thumb/e/e7/KnnClassification.svg/220px-KnnClassification.svg.png.

[13] "Non directed graph." https://www.researchgate.net/profile/Santiago-Bianco/publication/309278789/figure/fig12/AS:750920426078221@1556044789668/Grafo-no-dirigido-con-su-representacion-como-matriz-de-adyacencia.ppm.

[14] "Bias and variance examples." https://3.bp.blogspot.com/-HiwFTSJ0fkI/XKR6_D2kCfI/AAAAAAAAPLc/cu-y-VsObZYZM2KixwHOl10P_GraLjxSwCEwYBhgL/s1600/1_v63L_h5WXGOb4o6oh_daAA.jpeg.

[15] "¿qué es overfitting y cómo evitarlo?." https://empresas.blogthinkbig.com/que-es-overfitting-y-como-evitarlo-html-2.

[16] "Model fitting behavior." https://miro.medium.com/max/1200/1*YQ5tjb1TqNHenYMFk2tPog.png.

[17] "Cross-validation: evaluating estimator performance." https://scikit-learn.org/stable/_images/grid_search_cross_validation.png.

[18] "Ensembles: voting, bagging, boosting, stacking." https://www.iartificial.net/ensembles-voting-bagging-boosting-stacking/#Bagging.

[19] "Sensitivity and specificity." https://upload.wikimedia.org/wikipedia/commons/thumb/5/5a/Sensitivity_and_specificity_1.01.svg/341px-Sensitivity_and_specificity_1.01.svg.png.

[20] "Understanding confusion matrix." https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[21] "Understanding auc - roc curve." https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[22] "Ideal auc." https://miro.medium.com/max/1056/1*Uu-t4pOotRQFoyrfqEvIEg.png.

[23] "Worst roc curve." https://miro.medium.com/max/860/1*iLW_BrJZRI0UZSflfMrmZQ.png.

[24] "Ideal roc curve." https://miro.medium.com/max/730/1*-tPXUvvNIZDbqXP0qqYNuQ.png.

[25] "Training and test dets: Splitting data." https://developers.google.com/machine-learning/crash-course/training-and-test-sets/splitting-data.

PASCAL
ENERO 2018
Ult. actualización September 21, 2021
LaTeX lic. LPPL & powered by TEFLON CC-ZERO