Universidad de Córdoba

Escuela Politécnica Superior de Córdoba

Grado en Ingeniería Informática

Especialidad en Computación

Bachelor Thesis

Aplicación de técnicas de deep learning para la extracción de reglas de interés de una base de datos de pacientes con cáncer colorectal.

Interesting rule mining from a colorectal database enhanced with deep learning techniques

---

- Technical, User and Code Documentation -

Author

Daniel Ranchal Parrado

Supervisor

Dr. Carlos García Martínez

Córdoba, June 9, 2020

Contents

List of Figures

List of Tables

Listings

Part I

Technical documentation

Chapter 1

Introduction

In the last few years, Colorectal Cancer [1] has affected young people in Europe more and more. Around 37000 cases have been diagnosed last year in Spain and more than 15000 had died from it [2]. We have been provided with a dataset which contains information about patients that have suffered from this cancer by professionals from Reina Sofia University Hospital and it provides more than 1500 patients that have been described with 125 attributes. However, it contains a great quantity of missing values due to the same information was not collected for each patient.

Due to deep neural networks' success and the wide range of papers which try to extract interpretable information from them using association rules generators [3], the objective of this project is to obtain interpretable descriptions of patients that has suffered complications from Colorectal Cancer from inferred deep neural networks models. Due to this fact, this project is oriented to this type of cancer and may help to detect faster this disease and that is why applying computing methodologies to medical problems is

interesting.

The development of computing has contributed to have more powerful and smaller devices in our pockets but also it has improved our society in many fields. One of the fields that has used the advantage of computing is medicine. From robots controlled by doctors [4] to decision support systems, they have changed the state of art in medicine but also people's health as well as life expectancy [5].

In the computing field, machine learning has been gaining more popularity in the last years despite the fact that most of the algorithms used in ML had been discovered more than 50 years ago [6]. One of the reasons of its popularity is that computer scientist did not have enough powerful CPUs to run them neither enough data. But nowadays the world generates around 2.5 quintillion bytes of data each day from social networks [7], internet of things, etc. This big amount of data is generally called big data and it is re-defining our lives due to we are applying ML algorithms to extract knowledge from it.

One of the most known ML algorithms are neural networks. The first time we heard about them was in 1943 when Warren McCulloch and Walter Pitts created the first computational model for neural networks [8]. Artificial neural networks try to simulate how our neurons retain certain type of information and how they sent that information to others neurons by its connections. Artificial neural networks consists of an input layer with a defined number of neurons which represent each attribute or feature of a pattern. Then they have a certain number of hidden layers with an undefined number of neurons. The neurons in the hidden layers have different activations or operations. Finally we have the output layer, which produces the prediction

for that pattern that we initially introduced. All neurons from one layer to another are connected and each connection has a certain weight that is randomly generated at the beginning [9].

One type of neural network is a deep neural network, which is trained with deep learning algorithms [10]. A deep neural network is defined by its big amount of hidden layers. We use them when we face problems which have lots of patterns with lots of dimensions. This means that we will need a complex model to represent that big amount of data. Deep neural networks are generally used in computer vision problems [11] (convolutional neural networks) and bio-medicine. Applying ML algorithms in bio-medicine problems are really needed by professionals. Making models from data that have been collected from doctors for a long time can save lives. These models can predict whether a patient has a illness or not in less time than in usual situations. But they are also used to fight viruses in order to get a cure, as it is happening with SARS-CoV-2 virus [12], in which the Barcelona Supercomputing Center is using it to face it [13].

This document is organised in different parts which are divided in different sections. The first part of this project is the technical documentation, in which the system is explained to an expert. The second part will be the user documentation, where it will be explained how to execute the software in order to recover the missing values and generate rules. The last part is the code documentation, in which the code that has been develop to make possible this project is listed. The technical documentation will be divided in 11 sections, but they can be divided in three main parts. In the first one, the problem will be explained and its constraints as well as the main objectives to tackle in this project, the background of the project and the chronology

of the project. The second main part consists of defining the specification of the requirements of the problem and the system provided in order to solved as also its resources to approach it. Finally, the last part will expose the results obtained using this system and the conclusions of this project.

Chapter 2

Abstract/Resumen

This project is going to work with a data-set provided by professionals from Reina Sofia University Hospital that contains information about more than 1500 patients, describing each one with 126 attributes, that have suffered from Colorectal Cancer, that nowadays is affecting a big portion of the population, specially young people.

In this work deep neural networks are going to be applied, and in order to obtain tractable and interpretable information from the inferred neural networks models, association rules are going to be generated.

Este proyecto va a trabajar con un conjunto de datos que ha sido realizado por los profesionales del Hospital Universitario Reina Sofía de Córdoba. Este conjunto de datos contiene información sobre más de 1500 pacientes, los cuáles están descritos por 126 atributos, que han experimentado cáncer de colon, que actualmente afecta a una gran proporción de la sociedad, entre los que destacan las personas jovenes.

En este trabajo, se aplicarán redes neuronales profundas sobre este conjunto de datos, y para obtener conocimiento interpretable, se generarán reglas de asociación a partir de los modelos inferidos.

Chapter 3

Definition of the problem

The problem that we are addressing is predicting complications related with Colorectal Cancer. In order to do that, a dataset have been provided in which each patient is described in medical terminology related to this disease.

Nowadays, the most used techniques in machine learning and deep learning will be enough to extract important information from the data source and predict whether a patient have suffered from the disease or not. Although this type of algorithms gives the user relevant knowledge, it would be impractical for doctors because they are black box models. The definition of black box implies that the model receives an input, in this case, the data that defines a patient and gives the user an output, in this case, if the patient had complications or not.

The reason why doctors will see that this inferred knowledge is inefficacious is because of a black box model does not show the reason of classifying a patient as a person that has suffered from the disease or not. In case that an user wants to know the reason of classifying a patient in one class or an-

other, there is a type of model called white box models. A white box model receives a set of inputs, which are processed, and finally, it gives an output. The difference with a black box model is that a white box model shows the decisions it has taken to generate that output. Therefore, this type of model will represent a cause and effect diagram, which are understood easily by doctors. The dataset provided for this project, as it was explained briefly in the introduction, has information of 1500 patients which are described with several attributes. Most of the attributes have missing attributes, as it can be seen in the figure 3.1. However, some white box models are not prepared to work with missing data while others can ignore them while the algorithm is running.

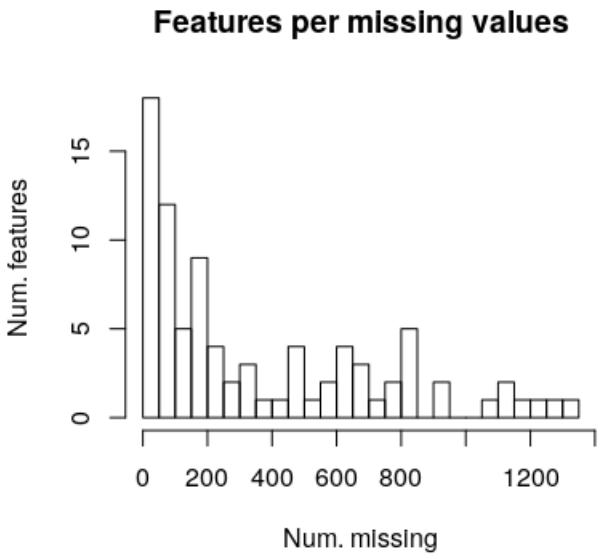

Figure 3.1: Histogram of features with missing values

Although ignoring missing values of a dataset could be an easy and fast method, using other types of methods, as recovering missing data, could improve the white box model and obtain better results. In this case, the

problem that is being approached in this project is going to use black box models in order to recover missing data from the dataset given, and once those values are properly recovered with several methods (Artificial neural networks, etc) which will be explained later, a rule-based system will be fed with this data to obtain valuable rules to predict complications in this type of disease.

Chapter 4

Objectives

The main objective of this project is to obtain interpretable descriptions, from a dataset, of patients that has suffered complications from Colorectal Cancer, and to analyse the possibilities that deep neural networks may offer in this endeavour. In order to make possible the main objective, the following sub-objectives must be fulfilled:

- Delete those attributes which are not relevant or really biased to the class attribute.
- Encode attributes with an one hot codification
- Normalization of the dataset in the scale [0,1]
- Define the strategies on how to recover missing values from the dataset
- Define the decision to see which attributes with missing values will be recovered

- Define a default method to recover data for those attributes that do not fulfill the decision defined in the previous objective
- Define methods to evaluate the quality of the different strategies of recovering data
- Generation of association rules from datasets with imputed values by means of neural networks.
- Discover the possibilities that neural networks provide us to obtain interpretable descriptions of patients.

Chapter 5

Background

The objective of this project is to get interpretable association rules with the aid of deep learning models from a Colorectal cancer database. In the state of the art of handling this type of problem, there are several approaches to solve it. In the following sections of this chapter, we are going to explain the background of the several approaches connected to this problem, the access to databases containing patients data, the approaches to recover missing values, the machine learning and deep learning models used in this project and finally, the association rules.

5.1 Papers related with Colorectal Cancer and ML

Y. Xu et al. in this publication [14], try to predict the recurrence of stage iv of the Colorectal Cancer after a tumor resection. In order to do that, they use four different machine learning algorithms which are a logistic regression,

a decision tree, Gradient boosting [15] and lightGBM [16] and compare their performance using the area under the ROC curve metric. The best ones are Gradient boosting and lightGBM. Using this type of algorithms, the authors are not interested in getting interpretable information but knowing the prediction of an stage of Colorectal Cancer.

Another approach applied to Colorectal Cancer data is the one published by M. Hornbrook et al. [17]. The authors of this publication use two different machine learning models with several datasets with anonymized data provided by institutions related to health. Although they are different datasets, they have the same attributes, which are gender, age and blood counts (number of red and white blood cells, haemoglobin, etc). The application of this computational models to this type of data helps to identify Colorectal Cancer in its early stages.

These techniques have not been applied in this project because we are interested in working with the dataset we have been provided which has certain peculiarities like the high number of missing values.

As it was explained in the several approaches to handle Colorectal Cancer with machine learning, the most important thing is to have access to databases with information of patients which have suffered from this disease, in other words, the data. Although it is the most obvious requirement, this type of datasets have so many restrictions in order to maintain the patients' privacy as well as the several difficulties to create and maintain a database like that. These restrictions cause an environment of inaccessible databases that force investigators to ask for help to institutions like hospitals or research centers as well as to comply with confidentiality agreements in order to not disclose private information.

5.2 Missing values

Another topic that is important when talking about machine learning and deep learning is missing values. Most of the databases available have unknown values [18] that need to be handled when using some computational models. There are many approaches, from easy ones to difficult ones, which are used.

In one hand, we have the group of the easiest ones, missing data is recovered with the mean and mode of the attribute, with a constant or if an attribute or a record have many missing values, those ones are deleted from the database. But in some cases, using this techniques can be harmful due to the database could have a few number of complete records and deleting them it is not a good idea, or using means, modes and constants in databases which could have a huge number of missing values, that could lead to bad results to the subsequent ML algorithm.

In the other hand, we have the group of the difficult ones, from using deep neural networks to K-nearest neighbours [19, 20]. The difficulty of using these techniques is that the investigator must dedicate a big amount of time to prepare the model, its parameters and prepare the dataset in order to feed the model with those ones which are not missing. Furthermore, the investigator must train the model for each attribute with missing values (dealing with error of the model), and the prediction of these missing values depends on the value of the rest of the attributes. Due to those reasons, most of the software related to manipulation and preprocessing of data contains easy ways to recover data.

In this project, we have used several models in order to recover the missing

values of the dataset with information of patients which have suffered from Colorectal Cancer. Those models are briefly explained in subsection 5.3.1 and 5.3.2.

5.3 ML and DL techniques related to the project

5.3.1 Deep neural networks

One of them is deep neural networks. They have been gaining more popularity in the last years thanks to the improvements made in hardware as well as problems as Image detection and reconstruction. They consists of many layers, and each layer has a huge number of neurons. Each neuron has an activation type. Although they are a huge success nowadays, this is the second birth of neural networks [21]. They became important in the 90's, but the lack of investigation on this model, the progress that investigators were doing in machine learning and the limitations in the existing software caused its fall. Another type of deep neural networks that we have used are Generative Adversarial Networks (GAN) [22] and it consists of two neural networks, and its main functionality is to generate synthetic instances from real data. They imitate the behaviour of the turing test [23], in which there is machine in one room and a real human in another room that asks several questions to the machine in order to know if the machine is a human or not. In this case, there is a network which generated fake instances (the robot) and the discriminator (the human), that evaluates if the instance is real or not.

5.3.2 Multiple Imputation by Chained Equations

Another algorithm that have been used in this project is Multiple Imputation by Chained Equations (MICE). We can not denominate it a model because it does not learn data to predict it but provides a set of steps using machine learning or deep learning models to recover the missing data from the database. This algorithm consists of iterating an undetermined number of times (also called cycles). The steps that MICE follows are explained by Azur, M.J et al. in their publication [24] as it follows:

1. *A simple imputation, such as imputing the mean, is performed for every missing value in the dataset. These mean imputations can be thought of as "place holders".*

2. *The "place holder" mean imputations for one variable ("var") are set back to missing.*

3. *The observed values from the variable "var" in Step 2 are regressed on the other variables in the imputation model, which may or may not consist of all of the variables in the dataset. In other words, "var" is the dependent variable in a regression model and all the other variables are independent variables in the regression model. These regression models operate under the same assumptions that one would make when performing linear, logistic, or Poison regression models outside of the context of imputing missing data.*

4. *The missing values for "var" are then replaced with predictions (imputations) from the regression model. When "var" is subsequently used*

as an independent variable in the regression models for other variables, both the observed and these imputed values will be used.

5. *Steps 2–4 are then repeated for each variable that has missing data. The cycling through each of the variables constitutes one iteration or "cycle". At the end of one cycle all of the missing values have been replaced with predictions from regressions that reflect the relationships observed in the data.*

6. *Steps 2–4 are repeated for a number of cycles, with the imputations being updated at each cycle.*

In step 3, although it is said to use linear, logistic or Poison model, machine learning models like K-nearest neighbours or deep neural networks can also be used.

5.3.3 Association Rules

The association rules are widely used in our society. It is used in the banking industry, where the financial information is processed to grant loans as well as predicting fraud cases among their clients, but they are also used by companies as a decision-based system in order to prevent failures in their business processes. Another technical field where they are used is in the health sector. This is because there are maintained databases in the sanity system and this data can be used to predict diseases, but also, doctors can see with them relations between symptoms and the disease that are not unknown. The previous examples have one thing in common. They provide interpretable knowledge that can be used for improving business processes or saving human lives [25].

There are several ways to generate association rules from a database, like the JRIP algorithm [26], CART algorithm [27] or Apriori Algorithm [28]. However, for this project we have used the Delgado-19 approach [29] described in their publication as it follows:

1. *First, the CART Tree is generated from the database.*
2. *Second, rules are produced for those interesting leaves with a maximum depth provided by the expert.*
3. *The attribute used in the root node is dropped.*
4. *Repeat from step 1 until no leaf within the maximum depth is obtained.*
5. *We may revise the final tree and decide repeating the process, from step 1, after having dropped other features for some reason*
6. *Finally, produce the rules for the interesting leaves.*

Chapter 6

Constraints of the problem

In this chapter we explain the constraints that this project has. The constraints that affect this project can be divided in two. The first type of constraints are related to the environment of the project while the second type of constraints are related to the decisions we have made to develop the project.

6.1 Constraints related to the nature of the problem

This type of constraints, as it was explained in the previous paragraph, are related to the environment (or nature) of the project. This definition means that those constraints can not be changed in the development of the project because they are inherited from the project. The constraints of this type for this problem are:

- The dataset we have been provided must be treated securely in order to avoid leaks of the private data of the patients.
- The dataset represent a binary classification problem.
- The last attribute of the dataset contains the classification label of the patient, which can take the values “Si” or “No”.
- The dataset contains missing data that need to be treated to work with it.
- The data of the patients are represented in a ARFF file.

6.2 Constraints related to the development of the project

As it says the name of this section, this type of constraints are related to the development of the project. This definition means that those constraints appear or are created when some decisions about taking one language or a specific tool are made.

Those constraints for this project are the following ones:

- The dataset must be normalized in the scale [0, 1] and one hot encoded to be read by the recovery module.
- The recovery of missing values module takes as input an ARFF file which contains the data of the patients.

- The recovery of missing values module must output a file with ARFF format with the recovered data to be understood by the generation of associative rules module.

- The use of computational models to recover the missing data from the dataset requires the definition of some metrics to measure the quality of the recovered data.

Chapter 7

Chronology

In this chapter we explain the several phases that this project had in order make possible the objectives listed in chapter 4. The list of phases are ordered by time and they are: study of Colorectal Cancer publication, requirements specification and system design, implementation, experimentation and documentation. Those phases are explained in the following sections.

7.1 Study of Colorectal Cancer publication

In this phase, in order to understand in which information is initially based our project, we need to study and collect information from the publication "Obtaining Tractable and Interpretable Descriptions for Cases with Complications from a Colorectal Cancer Database" [29]. This publication describes the dataset that we are using in this project to generate rules as well as the method to generate those rules, that we are also using. Once we know the limitations this dataset has, we can start to define the system regarding that

we are going to apply deep learning techniques.

7.2 Requirements specification and system design

In this phase, once we have collected the necessary information from the publication, we start to list the user and system requirements taking in account the information we got in the previous phase and the ideas that the authors have to apply deep learning in this process. Furthermore, we also define the modules that the system is going to have.

7.3 Implementation

In this phase, when we have defined the specifications of the system, we start to develop the several functionalities to recover the missing values of the dataset as well as the generation of rules, that are described in system requirements. We have used the tools which are described in section 8.2.

7.4 Experimentation

In this phase, after the implementation has been completed, we start to train several architectures of neural networks and comparing their metrics, to choose the best in order to recover missing values with a deep neural network. This process is explained in chapter 11.

7.5 Documentation

Finally, when all the previous phases have been finished, it is time to make the documentation that collects all the steps taken to make possible this project. The documentation has been divided in three parts:

- **Technical documentation**: It contains the description of the problem as well as the specification and the system we have develop to solve it. This part is oriented to users with software engineering skills.
- **User documentation**: It contains information on how to use the final product and it is oriented to the end user.
- **Code documentation**: It contains the code we have develop to build the system. It is commented to be easily understood.

Chapter 8

Resources

We have considered three types of resources. Data resources will show the information we have been provided to start this project. Software resources will show all the tools we have used in order to develop this project. And finally, hardware resources will show the specifications we have used to use the software resources.

8.1 Data resources

For this project, we have been provided with a dataset that contains information from patients which have suffered from Colorectal Cancer. The institution that had generated this database is Reina Sofia Hospital from Cordoba. The target attribute of this dataset is called "COMPLICACIONES" and it defines whether a patient has suffered complications or not. The original dataset that was given had 125 attributes that define a patient. After checking that some of the attributes belonged to stages previous the operation (not

adding knowledge to the database) or after the operation (adding biased information to the target attribute), they were deleted from the database, as it is required in the objectives of this project in section 4. This deletion changed the number of attributes from 125 to 87 attributes.

8.2 Software resources

In order to make possible this project, we have used a set of software tools that includes data visualization programs as also as programming languages and its packages. They are listed below:

- **Operating system**: Ubuntu 18.04 and Debian 8
- **Programming Languages**: Python 3.7.7, R 3.6.3
- **Main Python Packages**: Tensorflow [30], Keras [31], NumPy [32], SciPy [33], scikit-learn and [34]
- **Data visualization and preprocessing**: Weka [35]
- **IDEs**: Visual Studio Code, RStudio
- **Documentation**: LaTeX

8.3 Hardware resources

We have run all the experiments in the servers from KDIS research group, in the supervisor's computer and author's computer. Below, we described their specifications:

- KDIS server
  - **Operating system**: Debian 8
  - **CPU**: Intel(R) Core(TM) i7 CPU @ 2.67GHz
  - **Cores**: 4 (8 in all)
  - **RAM**: 12 GB
- Supervisor's computer
  - **Operating system**: Ubuntu 18.04.4
  - **CPU**: Intel(R) Core(TM) i7 CPU @ 2.80GHz
  - **Cores**: 8
  - **RAM**: 21 GB
- Author's computer
  - **Operating system**: Ubuntu 18.04.4
  - **CPU**: Intel(R) Core(TM) i7 CPU @ 2.50GHz
  - **Cores**: 2
  - **RAM**: 12 GB

Chapter 9

Requirements specification

In this chapter, we are going to list the different requirements that this project must satisfy once it is finished as well as defining the uses cases of the project. We divide the requirements in two big groups: the user requirements and system requirements.

9.1 User Requirements

This type of requirements are related with the actions that the user will do with this project. In other words, these requirements, for this project, are the functionalities that the user wants for this work. The user requirements are the following ones:

UR-1 As an user I can generate associative rules with Delgado-19 method [29] from the dataset of patients.

UR-2 As an user I can recover the missing data of the dataset using popular

techniques as well as novel approaches.

UR-3 As an user I can select which attributes with missing values will be imputed with advanced techniques and which ones will be recovered with simple techniques.

UR-4 As an user I can define the output file in which the dataset with the recovered data will be saved.

UR-5 As an user I can generate a CSV file with the list of rules generated and their metrics.

9.2 System Requirements

This type of requirements are related to the functionalities that the system must do to satisfy the user requirements which have been explained in section 9.1. System requirements are divided in three groups: functional requirements, non functional requirements and information requirements.

9.2.1 Functional Requirements

Functional requirements establish how the functionalities should work in the system in order to satisfy the user requirements as well as the main objectives of the project. The functional requirements of this system are the following ones:

FR-1 The system must be able to recover the missing data.

FR-2 The system must warn the user when something on the execution of the software goes wrong.

FR-3 The system must differentiate between not important and relevant attributes.

FR-4 The system must differentiate between numerical and categorical attributes when recovering their missing values.

FR-5 The system must provide different computational models for recovering missing values from numerical and categorical attributes.

FR-6 The system must provide information about which attribute is being recovered while it is executing.

FR-7 When the process of recovering the missing values is finished, the system must provide two different output files, one for checking the quality of the recover method, and the other one to be the input for that part of the system which generates rules.

FR-8 The system must provide a method to measure the quality and relevancy of the recovered values in the dataset.

FR-9 The system must generate a list of rules from a dataset.

FR-10 The system must output the metrics of the rules generated in order to check their quality and relevancy.

FR-11 The system must be able to modify the length of the output which are generated.

FR-12 The system must save the history of rules that have been generated each time it is executed.

FR-13 The system must be able to discard rules that have been generated previously.

9.2.2 Non Functional Requirements

Non functional requirements are not related with how functionalities are done in the system but they influence the system as constraints or restrictions which are related to reliability, usability, security, performance, etc. The non functional requirements of this system are the following ones:

NFR-1 The system should be executed in all operating systems.

NFR-2 The system must be terminated correctly in case of error.

NFR-3 The system is only prepared for those datasets which have missing values.

NFR-4 The part of the system which recovers missing values must accept datasets which have been previously normalized and encoded with the one hot method.

NFR-5 If the methodology selected for recovering missing values is not implemented, the system must output an error and be terminated.

9.2.3 Information Requirements

Information requirements establish how the system should proceed when it is working with any type of data, including management of user data (that is not relevant in this case) and the generation of information from different

sources like datasets. The information requirements of this system are the following ones:

IR-1 The system must accept as input files an ARFF file format, which represent a list of records with their attributes and their classification labels.

IR-2 The part of the system that preprocess the data must accept as input file an ARFF file, and it must output another ARFF file with preprocessed data.

IR-3 The part of the system that recovers the missing data must accept as input an ARFF file with its data preprocessed, and it must output two ARFF files, which have different goals.

IR-4 The part of the system which generates the associative rules must accept as input several ARFF files, and it must output a CSV file which contains the generated rules.

IR-5 The generated rules also must be saved in a TXT file, which acts as a buffer of list.

IR-6 The CSV file with the generated rules must show the different rules as well as its metrics.

An example of an ARFF file is displayed in listing 9.1.

```
1 @RELATION my-dataset
2
3 @ATTRIBUTE attribute-1  NUMERIC
4 @ATTRIBUTE attribute-2  NUMERIC
5 @ATTRIBUTE attribute-3  {Yes,No}
```

```
@ATTRIBUTE class {class−1,class−2}

@DATA
8.0,2.0, Yes, class −1
8.1,2.1, Yes, class −1
8.2,2.2, Yes, class −1
8.3,2.3, Yes, class −1
8.4,2.4, Yes, class −1
8.5,2.5, No,class−2
8.6,2.6, No,class−2
8.7,2.7, No,class−2
8.8,2.8, No,class−2
8.9,2.9, No,class−2
```

Listing 9.1: ARFF format example file

9.3 Use cases

9.3.1 UC-0: Context Diagram

UC-0:	**Context Diagram**
Description:	Description of the behaviour of the system
Primary Actor:	User
Precondition:	It belongs to the system
Annotation:	The context of the system gives an idea of which functionalities have this project

Table 9.1: UC-0: Context Diagram

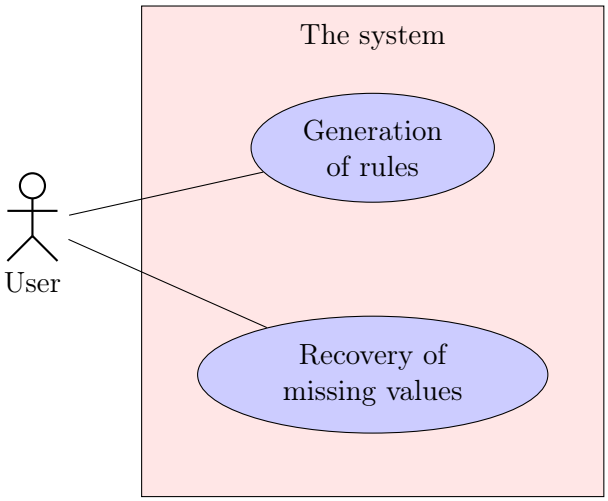

Figure 9.1: UC-0 Diagram

9.3.2 UC-1: Generation of associative rules

UC-1:	**Generation of associative rules**
Description:	It generates a list of rules from a dataset
Primary Actor:	User
Precondition:	The dataset with recovered values and prepared for the generation of rules must be generated
Postcondition:	The generated rules are stored in a CSV file

Main Success Scenario:

1. Select the dataset to generate the rules
2. Select the length of the rules
3. Select output file for the generated rules
4. Generate rules

Table 9.2: UC-1: Generation of associative rules

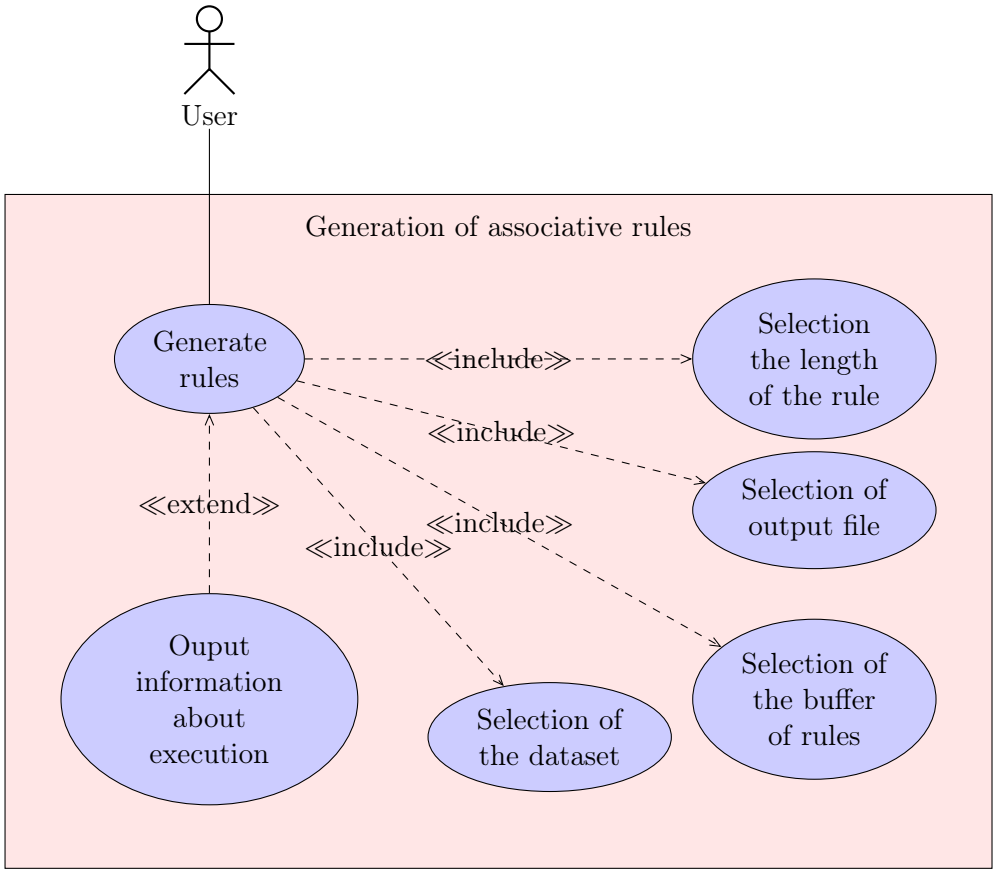

Figure 9.2: UC-1 diagram

9.3.3 UC-1.1: Selection of the dataset

UC-1.1:	**Selection of the dataset**
Description:	It is in charge of getting the file with the information to generate rules
Primary Actor:	User
Preconditions:	• The file must have ARFF format • The file must exist
Postcondition:	The file is read
Main Success Scenario: 1. The file is selected 2. The file is read	
Alternative Scenario 2.b An error is thrown due to the nonexistence of the file	

Table 9.3: UC-1.1: Selection of the dataset

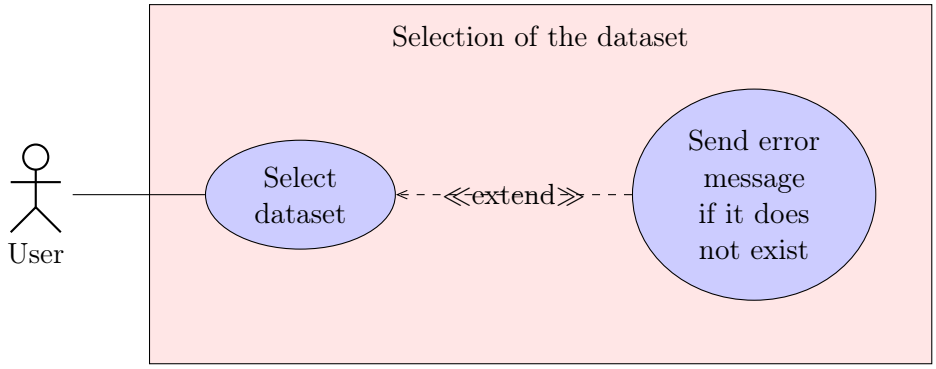

Figure 9.3: UC-1.1 diagram

9.3.4 UC-1.2: Selection of the output file

UC-1.2:	**Selection of the output file**
Description:	It sets the output file for the list of rules
Primary Actor:	User
Preconditions:	The file must have CSV format
Postcondition:	The file is opened or created

Main Success Scenario:

1. The file is selected
2. The file is opened or created

Alternative Scenario

2.b An error is thrown due to the lack of permissions of the file

Table 9.4: UC-1.2: Selection of the output file

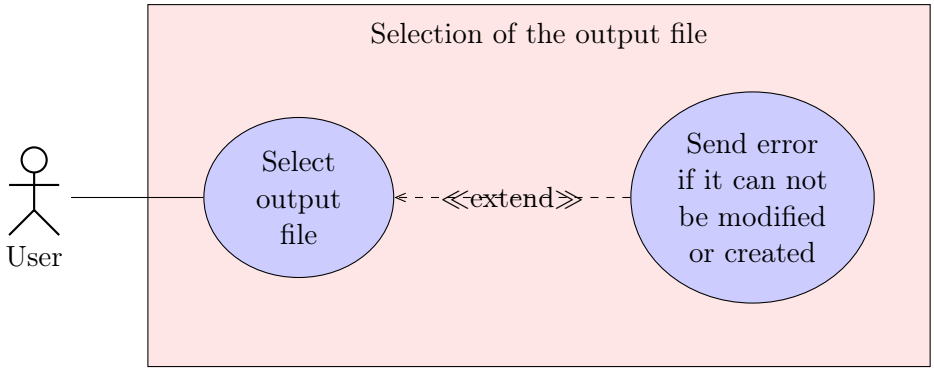

Figure 9.4: UC-1.2 diagram

9.3.5 UC-1.3: Selection of the buffer of rules

UC-1.3:	**Selection of the buffer of rules**
Description:	It sets the buffer file in which the generated rules are saved
Primary Actor:	User
Preconditions:	The file must have TXT format
Postcondition:	The file is opened or created

Main Success Scenario:

1. The file is selected
2. The file is opened or created

Alternative Scenario

2.b An error is thrown due to the lack of permissions of the file

Table 9.5: UC-1.3: Selection of the output file

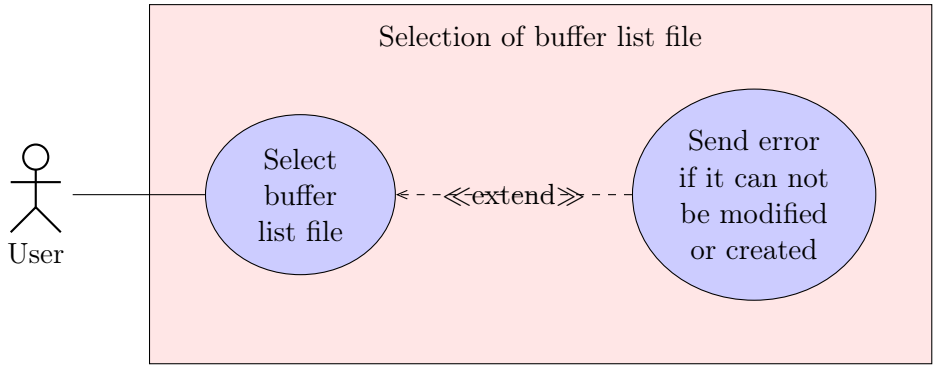

Figure 9.5: UC-1.3 diagram

9.3.6 UC-1.4: Selection of length of the rules

UC-1.4:	**Selection of length of the rules**
Description:	It sets the buffer file in which the generated rules are saved
Primary Actor:	User
Preconditions:	None
Postcondition:	The rule's length is a number and greater than 1
Main Success Scenario:	1. The user inputs a number greater than 1
Alternative Scenario	1.b An error is thrown if it is not a number or it is equal or lower than 1

Table 9.6: UC-1.4: Selection of length of the rules

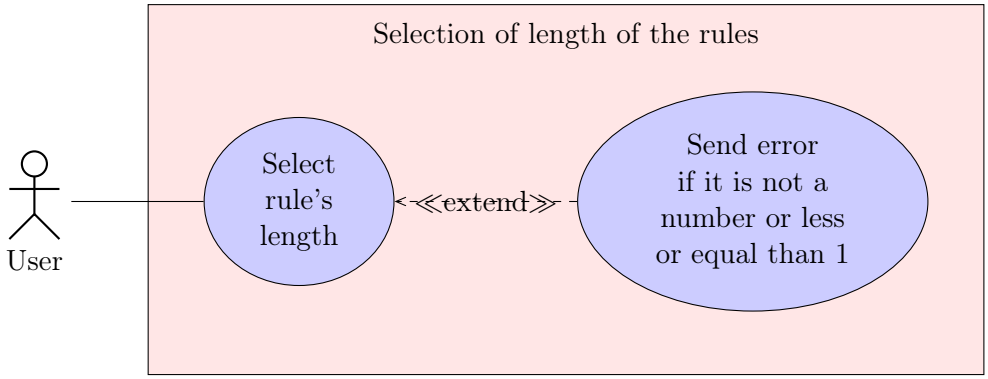

Figure 9.6: UC-1.4 diagram

9.3.7 UC-2: Recovery of missing values

UC-2:	**Recovery of missing values**
Description:	It recovers the missing values of the dataset
Primary Actor:	User
Precondition:	The dataset must be normalized as well as one hot encoded
Postconditions:	• The dataset has not got missing values • It generated two ARFF files. One prepared for checking the quality of the imputation and the other for generating rules.

Main Success Scenario:

1. Selection of the input file
2. Selection of easy and advanced methodologies for imputing data.
3. Selection of attributes that will be recover with easy and advances methodologies
4. Select output file for the recovered data
5. Recover missing values

Table 9.7: UC-2: Recovery of missing values

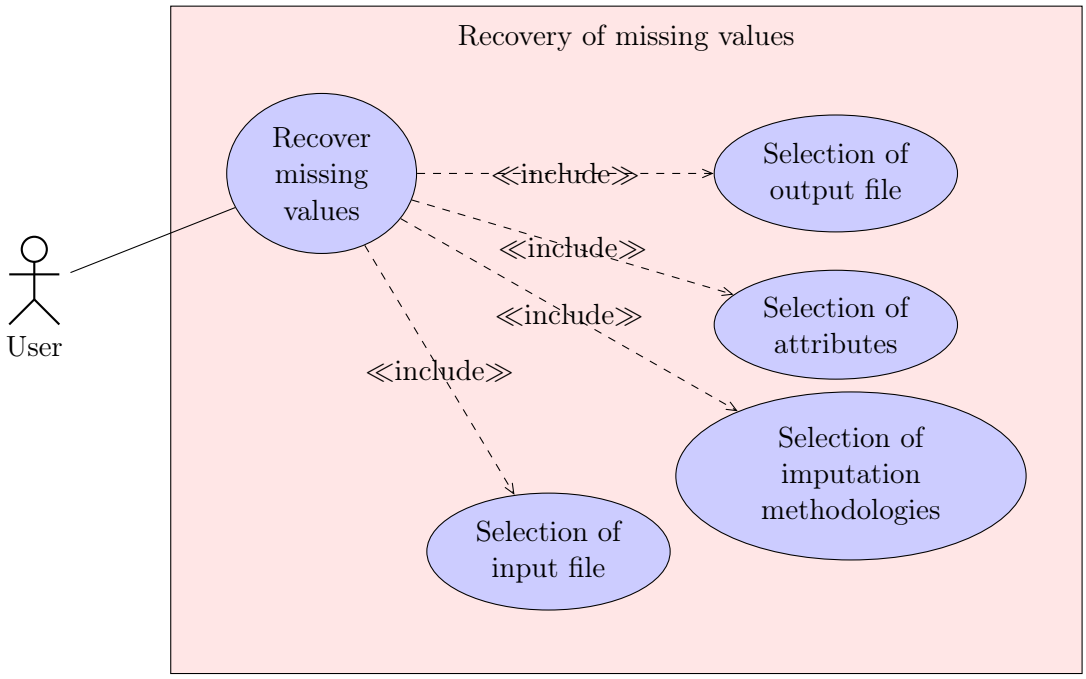

Figure 9.7: UC-2 diagram

9.3.8 UC-2.1: Selection of input file

UC-2.1:	**Selection of input file**

Description:	It reads the file with missing values
Primary Actor:	User
Preconditions:	• The file must have ARFF format • The file must exist
Postcondition:	The file is read
Main Success Scenario:	1. The file is selected 2. The file is read
Alternative Scenario	2.b An error is thrown due to the nonexistence of the file

Table 9.8: UC-2.1: Recovery of missing values

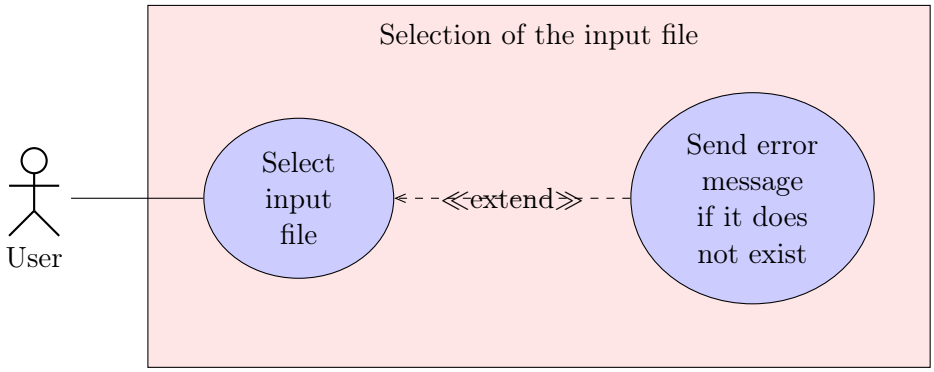

Figure 9.8: UC-2.1 diagram

9.3.9 UC-2.2: Selection of the output file

UC-2.2:	**Selection of the output file**
Description:	It sets the output file which does not contain missing values
Primary Actor:	User
Preconditions:	The file must have ARFF format
Postcondition:	The file is opened or created

Main Success Scenario:

1. The file is selected
2. The file is opened or created

Alternative Scenario

2.b An error is thrown due to the lack of permissions of the file

Table 9.9: UC-2.2: Selection of the output file

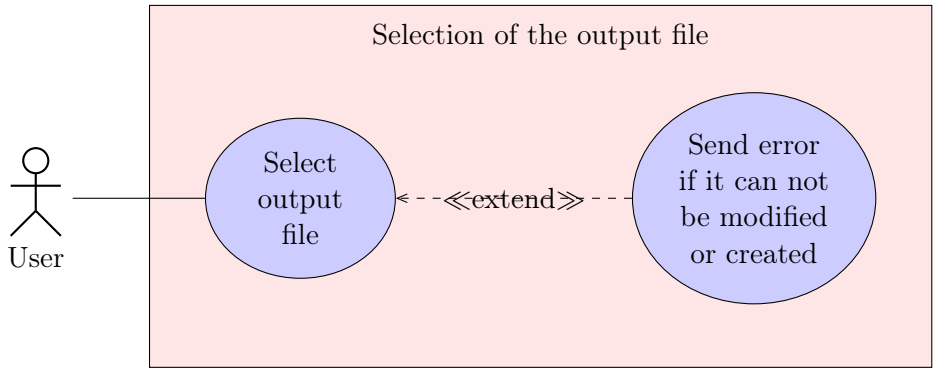

Figure 9.9: UC-2.2 diagram

9.3.10 UC-2.3: Selection of imputation methodologies

UC-2.3:	**Selection of imputation methodologies**
Description:	It chooses the methodologies used to impute the missing values
Primary Actor:	User
Preconditions:	None
Postcondition:	Implemented methodologies have been chosen

Main Success Scenario:

1. Select methodology for imputing interesting attributes
2. Select methodology for imputing not interesting attributes

Alternative Scenario

1.b An error is raised if the methodology is not implemented

2.b An error is raised if the methodology is not implemented

Table 9.10: UC-2.3: Selection of imputation methodologies

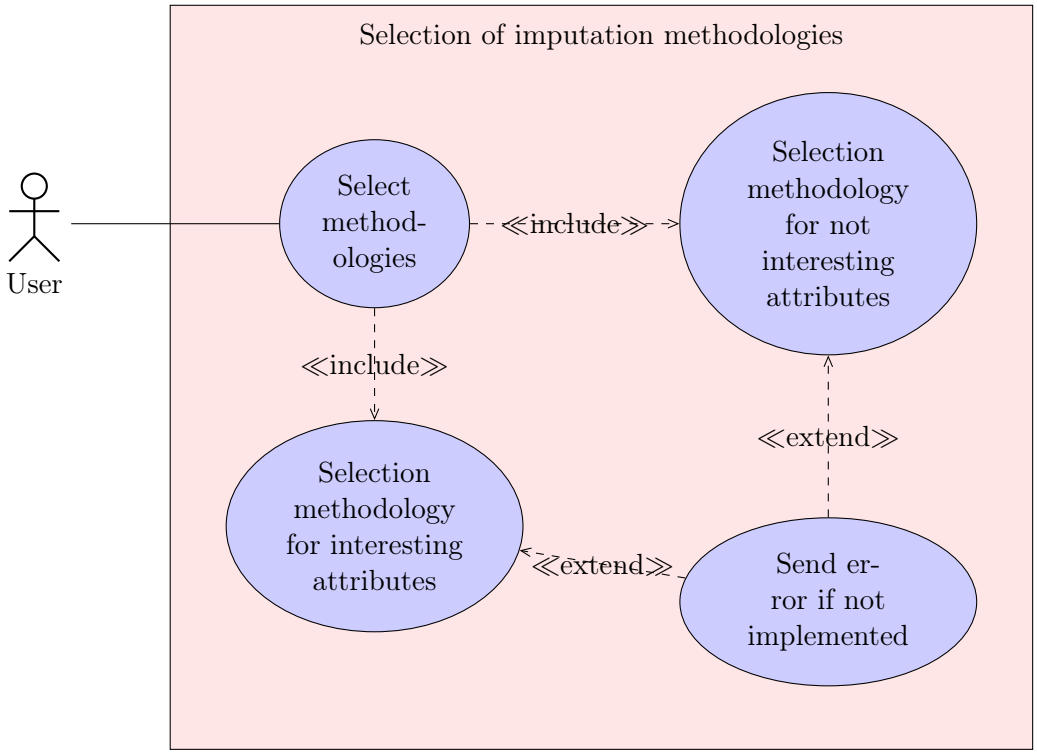

Figure 9.10: UC-2.3 diagram

9.3.11 UC-2.4: Selection of attributes

UC-2.4:	**Selection of attributes**
Description:	It chooses which attributes with a quantity of missing values in a determined range by the user are recover with advanced methodologies
Primary Actor:	User
Preconditions:	Methodologies have been chosen

Postcondition:	The attributes are classified to be recovered with easy and advanced methodologies

Main Success Scenario:

1. Select a numeric range for missing values

Alternative Scenario

1.b An error is raised if it is not a numerical range

Table 9.11: UC-2.4: Selection of attributes

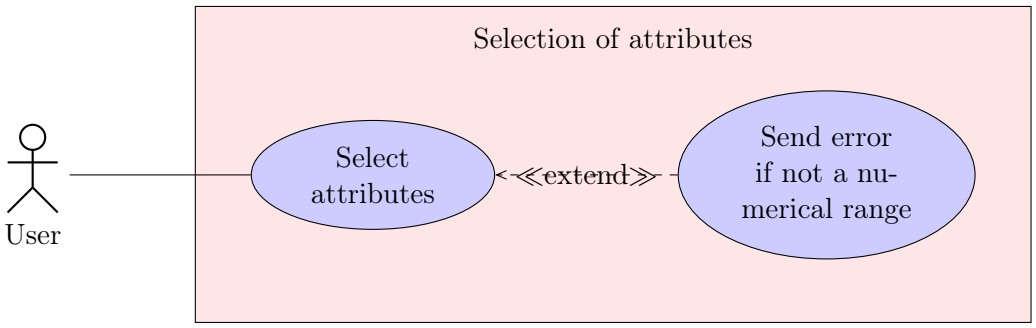

Figure 9.11: UC-2.4 diagram

Chapter 10

System design

In this chapter, we intend to explain how the system works explaining all the functionalities it has. We have divided this chapter in different sections, which contains the class diagram of the system (in section 10.1), the module diagram of the system (in section 10.3) as well as the description of each module. We define three main modules in the system: preprocessing (in section 10.4), recovery of missing values (in section 10.5) and generation of associative rules (in section 10.6).

10.1 Class Diagram

In figure 10.1, we can see the class diagram of the system. It consist of eleven classes, where one of them acts as an interface. As the diagram shows, the Recover uses Patients, DecodeService and baseModel as dependencies. The definition of dependency means that if any of those three classes are modified, the Recover class may change its implementation to support the changes made in any of those classes.

The diagram also shows relationships between the baseModel interface class and regressionGanImputationModel, classificationGanImputationModel, regressionModelNN, classificationModelNN, regressionModelkNN, classificationModelkNN and bestModelAttribute classes. This type of relationship between these classes is called implementation. That

means those classes are implementing the virtual methods that the interface is providing.

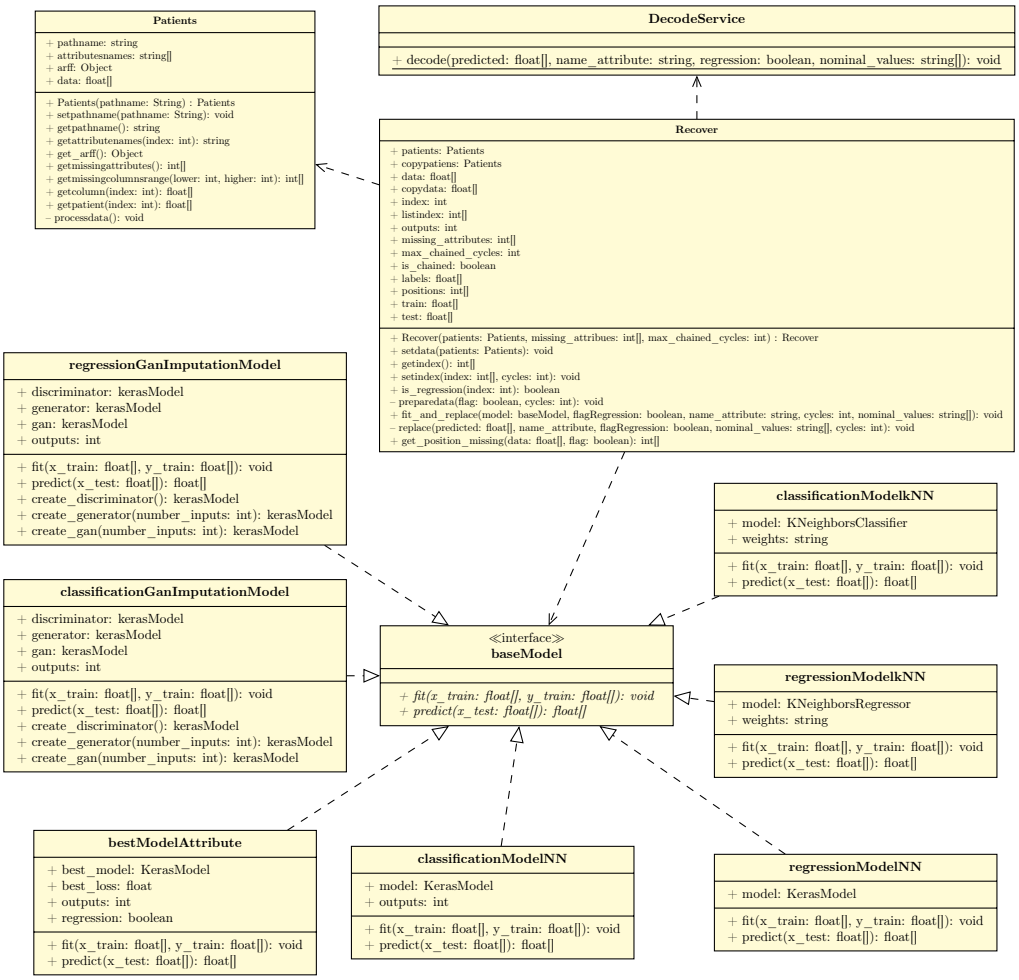

Figure 10.1: Class diagram

10.2 Class specification

In this section, we specify all the classes that appear in the figure 10.1, in section 10.1.

10.2.1 Recovery Class

Class: Patients
It is in charge of the recovery of missing values.
Attributes: *patients*: Patients → It is an instance of the class Patients. *copypatiens*: Patients → It is a copy of the instance patients. *data*: float[] → It stores the numerical data of the patients. *copydata*: float[] → It stores a copy of data. *index*: int → It stores the index of the attribute it is recovering. *listindex*: int[] → It stores the list of attributes it is recovering. *outputs*: int → It stores the number of values that the categorical attribute can take. *missing_ attributes*: int[] → It stores the indexes of the attributes with missing values. *max_ chained_ cycles*: int → It stores the number of cycles that MICE will execute. *is_ chained*: boolean → It says if the class is executing MICE. *labels*: float[] → It stores the values of the attributes that it is recovering. *positions*: int[] → It stores the positions of the missing values. *train*: float[] → It stores the training data for the computational model. *test*: float[] → It stores the test data for the computational model.
Methods: *Recover*: It is the constructor of the class. *setdata*: It updated the data of patients. *getindex*: It returns the indices of the attributes it is recovering. *setindex*: It sets the index of attributes to recover. *is_ regression*: It tells if the attribute is numerical. *preparedata*: It prepares the data to be divided in train and test depending on the location of the missing values. *fit_ and_ replace*: It is in charge of training the model and predicting the labels for the instances with missing values. *replace*: It replaces the missing values with the prediction of the model. *get_ position_ missing*: It returns the location of the missing values in the attributes it is recovering.

Table 10.1: Recovery class specification

10.2.2 DecodeService class

Class: DecodeService
It is in charge decoding the data to the scales of the original dataset.
Methods: *decode*: It is in charge of decoding the numerical and categorical data as well as modifying the dataset prepared for the generation of rules module.

Table 10.2: DecodeService class specification

10.2.3 Patients Class

Class: Patients
It represents a list of patients.
Attributes: *pathname*: string → It represents the name of the file. *attributesnames*: string[] → It stores the names of the attributes. *arff*: Object → liac-arff arff object. *data*: float[] → It stores the data of patients.
Methods: *Patients*: It is the constructor of the class. *setpathname*: It sets the filename of the dataset. *getpathname*: It returns the filename of the dataset. *getattributenames*: It returns the name of an attribute. *get_ arff*: It returns the object arff. *getmissingattributes*: Returns the list of attributes with *missing values*. *getmissingcolumnsrange*: Returns list of attributes with missing values in a range. *getcolumn*: Returns an attribute of the dataset. *getpatient*: Return a patient of the dataset. *processdata*: Reads the file and prepares the data.

Table 10.3: Patients class specification

10.2.4 baseModel class

Class: baseModel
It defines an interface class to be implemented by the rest of classes that represent computational models.
Methods: *fit*: Virtual class that is in charge of training the computational model. *predict*: Virtual class that is in charge of predicting labels for a group of instances.

Table 10.4: baseModel class specification

10.2.5 regressionModelkNN class

Class: regressionModelkNN
It represents a model to recover numerical attributes with the kNN model.
Attributes: *model*: KNeighboursRegressor → It is a kNN model. *weights*: string → It determines which weights is going to use the kNN model.
Methods: *fit*: It trains the kNN model. *predict*: It predicts the label of the instances with the kNN model.

Table 10.5: regressionModelkNN class specification

10.2.6 classificationModelkNN class

Class: classificationModelkNN
It represents a model to recover categorical attributes with the kNN model.
Attributes: *model*: KNeighboursClassifier → It is a kNN model. *weights*: string → It determines which weights is going to use the kNN model.
Methods: *fit*: It trains the kNN model. *predict*: It predicts the label of the instances with the kNN model.

Table 10.6: classificationModelkNN class specification

10.2.7 regressionModelNN class

Class: regressionModelNN
It represents a model to recover numerical attributes with a deep neural network.
Attributes: *model*: KerasModel → It is a neural network model.
Methods: *fit*: It trains the deep neural network. *predict*: It predicts the label of the instances with the deep neural network.

Table 10.7: regressionModelNN class specification

10.2.8 classificationModelNN class

Class: classificationModelNN
It represents a model to recover categorical attributes with a deep neural network
Attributes: *model*: KerasModel → It is a neural network model. *output*: int → It stores the number of values that the categorical attribute can take.
Methods: *fit*: It trains the deep neural network. *predict*: It predicts the label of the instances with the deep neural network.

Table 10.8: classificationModelNN class specification

10.2.9 regressionGanImputationNN class

Class: regressionGanImputationNN
It represents a model to recover numerical attributes with a deep neural network
Attributes: *discriminator*: KerasModel → It is a neural network model which acts as classifier. *generator*: KerasModel → It is a neural network model which generates fake data. *gan*: KerasModel → It is a neural network model that combines the discriminator and generator neural network model. *output*: int → It stores the number of attributes it recovers.
Methods: *fit*: It trains the Generative Adversarial Networks. *predict*: It predicts the label of the instances with the Generative Adversarial Networks. *create_ discriminator*: It builds the discriminator neural network. *create_ generator*: It builds the generator neural networl. *create_ gan*: It builds the GAN, which combines the discriminator and generator neural network.

Table 10.9: regressionGanImputationNN class specification

10.2.10 bestModelAttribute class

Class: bestModelAttribute
It searches the best neural network architecture for each attribute and recovers missing values from numerical and categorical attributes.
Attributes: *best_ model*: KerasModel → It stores the deep neural network model with the best architecture. *best_ loss*: float → It stores the loss metric of the best model. *outputs*: int → It stores the number of attributes it is recovering. *regression*: boolean → It stores whether it is recovering a numerical attribute or not.
Methods: *fit*: It trains the deep neural network model. *predict*: It predicts the labels of instances with the deep neural network model.

Table 10.10: bestModelAttribute class specification

10.2.11 classificationGanImputationNN class

Class: classificationGanImputationNN
It represents a model to recover categorical attributes with a deep neural network
Attributes: *discriminator*: KerasModel → It is a neural network model which acts as classifier. *generator*: KerasModel → It is a neural network model which generates fake data. *gan*: KerasModel → It is a neural network model that combines the discriminator and generator neural network model. *output*: int → It stores the number of attributes it recovers.
Methods: *fit*: It trains the Generative Adversarial Networks. *predict*: It predicts the label of the instances with the Generative Adversarial Networks. *create_ discriminator*: It builds the discriminator neural network. *create_ generator*: It builds the generator neural networl. *create_ gan*: It builds the GAN, which combines the discriminator and generator neural network.

Table 10.11: classificationGanImputationNN class specification

10.3 Module diagram

In figure 10.2, we show the architecture diagram of the system. The system has three main modules which are preprocessing, recovery of missing values and generation of associative

rules. These modules have interactions with the user, but also with other modules and those relationships are visible in the figure.

As it shows, the user of the system uses the three modules. When the user uses the preprocessing module, it is executed and then, it sends the preprocessed data to the recovery module, which is also used by the user. Once the recovery module receives the preprocessed dataset, it is executed and it is generate a dataset which is sent to generation of associative rules modules, which is used by the user of the system. Finally, when this module receives the dataset, it generates a list of rules and it sends it to the user.

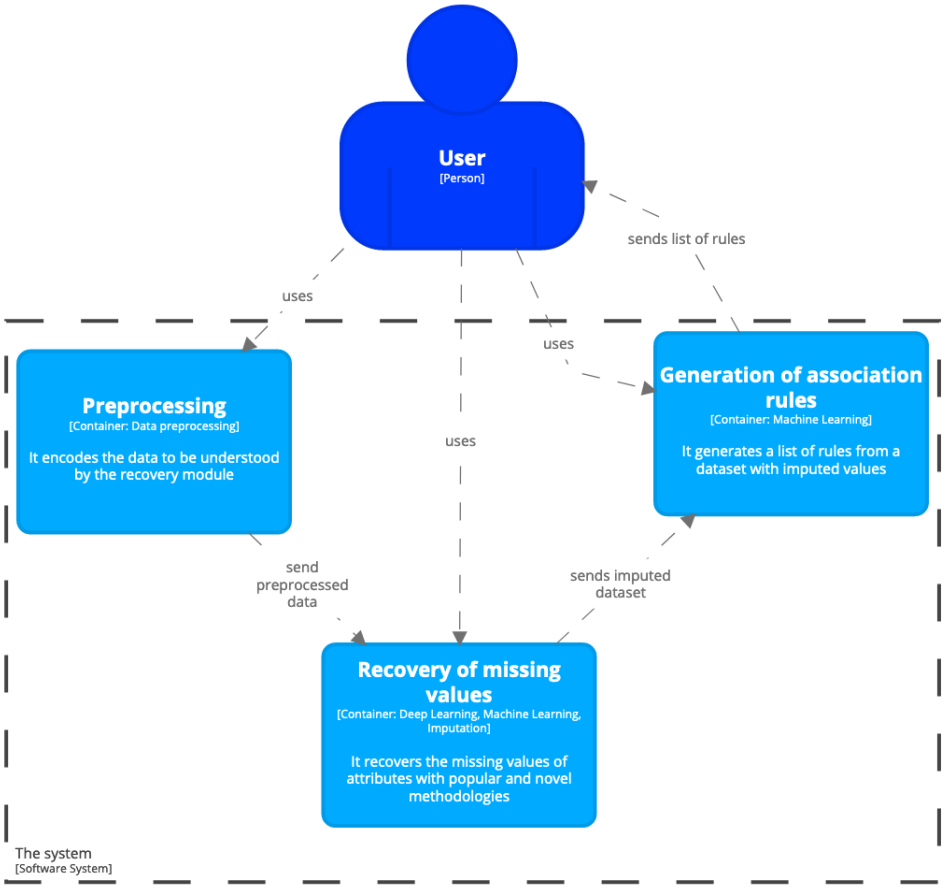

Figure 10.2: Software diagram

10.4 Preprocessing module

This part of the system is in charge of the preprocessing of the dataset provided by the user as an input. Although this part of the system is the easiest one, it has certain relevance.

We have been reviewing which alternative was the best to preprocess the data. One of them was using the preprocessing module of scikit-learn but the way in which one hot encoding works in their package is not the expected one by the authors of this project, so it was rejected. Another alternative we reviewed was Weka (which is the one we selected for this project). Weka provides a graphical method to preprocess but also via python using a wrapper. The preferred way by the authors was using the wrapper in order to have preprocessing and recovery of missing values in the same language, in other words, to be automatic.

The idea of using the wrapper was rejected due to the repository was not maintained in the moment that this project was being developed as well as it was not worth to fix that wrapper because of matter of time. Finally, it was decided to use the graphical method of Weka to preprocess the dataset.

We are applying two filters, which are really important to be applied in order to get better results with machine learning and deep learning algorithms.

10.4.1 One Hot Encoding

The first filter applied to the dataset is One Hot Encoding [36], and it affects categorical attributes whose values are not numerical but label data as a string. The main objective of this filter is to convert that label data (strings) into numerical data to be understood by computational algorithms. If this filter is not applied, the algorithms are going to fail.

To understand better one-hot encoding, an example of what happens inside Weka when the filter is executing is showed in table 10.12.

Record	an_attribute
1	value1
2	value2
3	value3

Table 10.12: Categorical attribute

When Weka checks that an_attribute is categorical, it creates as many new attributes as values can take an_attribute. In this case, the name of the new attributes will be "an_attribute=value1", "an_attribute=value2" and "an_attribute=value3". Also, Weka will put a 1 in the new attributes if they were the value of the record in table 10.12. The new attributes and their values are displayed in table 10.13. The convention of the naming of the new attributes must not be changed in order to be read by the part of the system which recovers the missing values.

Record	an_attribute=value1	an_attribute=value2	an_attribute=value3
1	1	0	0
2	0	1	0
3	0	0	1

Table 10.13: One hot applied to the categorical attribute

10.4.2 Normalization

The other filter that is applied is Normalization, which scales the numerical attributes between 0 and 1. Having the same scale for all the attributes of the dataset improves the accuracy and the execution of the computational algorithm. An example of normalization of a numerical attribute is displayed in table

Record	numerical-attribute	normalized-attribute
1	3	1
2	2	0.5
3	1	0

Table 10.14: Numerical attribute and normalized numerical attribute

After this operations are executed, the output of this part of the system is an ARFF

file which have been one hot encoded and normalized. This new dataset will be used in the part of the system that is described in section 10.5.

10.5 Recovery of missing values

This part of the system is in charge of recovering the unknown values from the dataset provided by the user as an input and has been preprocessed previously. We have developed several computational models, including deep learning models (the objective of this project) and some machine learning models to compare all of them. Those models are explained in the following subsections.

10.5.1 Deep learning model

In this subsection, we explain how we had develop artificial neural networks in the system. In order to simplify the development of artificial neural networks, we have used the framework Keras [31], which provides an easy API to build neural networks, and it is built on top of Tensorflow [30].

As it is explained in the requirements of the system, we need to build different models for imputing missing values from numerical and categorical attributes. We have used the model *Sequential* to initialize the neural network, and each layer of the artificial neural network has an activation. We have set the ReLu [37] activation for those layers that are not the output one. If we are recovering a categorical attribute, we use a softmax activation in the last layer but if we impute a numerical attribute, we only set "normal" as kernel_initializer. For the hidden layers of the neural network to recover numerical attributes, we also use ReLU activations but we also set "normal" as activation.

For both types of artificial neural networks, we set Adam [38] as the optimizer, however, the loss metric for the categorical neural network is categorical_crossentropy and mean_squared_error for the numerical neural network.

For training and predicting, Keras models provide us the function *fit* to train the model, which receives as arguments the instances and their labels. Once the model is

trained, in order to predict the labels of uncategorized data, Keras provides the function *predict* that receives as argument the instances without label and return the predictions for each instance.

10.5.2 K-Nearest Neighbours model

In this subsection, we explain how we had implemented kNN models in order to recover missing values.

To simplify the code, we have used the module neighbours from the scikit-learn package, which provides several models to apply this algorithm. As it is explained in the requirements section, the system must be able to handle the recovery of numerical and categorical attributes. In this case, when we recover numerical attributes, we use the model *KNeighborsRegressor* but when it is handling categorical attributes, we use the model *KNeighborsClassifier*. We have maintained the default parameters, which are the number of neighbours (set to 5) and the weight function used in the prediction (in this case "distance"). Those models provide a set of functions to train the model, which are used by this system and those are *fit* and *predict*.

10.5.3 Multiple Imputation by Chained Equations

Multiple Imputation by Chained Equations (MICE) is not a computational model but an algorithm with a list of steps. It consists of recovering missing values based on the attributes which had not got missing ones (as in sections 10.5.1 and 10.5.2) but also with the attributes which have been recovered. This methodology is executed a fixed number of cycles. A cycle is defined as recovering the attributes in which we are interested.

The recovery of this missing values is based on the execution of the models that have been explained before. The first time it is executed, it has the same behaviour as when we run it with only a deep learning model or only a K-nearest neighbours model. In successive cycles, the procedure is the following one:

1. The system gets the attribute or attributes (if it was a categorical attribute) which

have been recovered in previous cycles.

2. Then, the system searches for the positions of those values which were recovered in previous cycles.

3. Once it has the positions, the system fills them with *np.nan*, which is the value to represent unknown values in numpy arrays.

In the last cycle of MICE, in order to avoid useless files, the output files which are prepared for experimentation and for the generation of rules are generated.

10.5.4 Generative Adversarial Networks

Generative Adversarial Networks (GAN) are composed from two networks which have different functionalities. The first network is the *discriminator*, which decides the classification of a specific instance. The second network is the *generator*, which is in charge of generating new synthetic instances that are similar to the real data. The objective of the *generator* network is to generate a good enough fake instance in order to fool the *discriminator* network and be classified as a "real" instance.

We have implemented this methodology in the system for imputing missing values in the following way:

- When the system has ordered the data to put as label the attribute we want to recover, we take some instances of real data randomly.

- Then, the system generates several instances with random noise which are passed to the generator, and it generates fake data.

- After that, real and fake data are concatenated and the discriminator trains its model with that data.

- Once it has been trained, the system fix the weight of the *discriminator* to not be trained and then, the GAN is trained. This last action triggers the learning of the generator network by changing the weights of its connections.

- After a number of epochs or when the training error does not improve, the system generates as many new instances as missing values has the attribute that the system wants to recover.
- For each instance that has a missing value, the system calculates the distances or similarity between the instance with the missing value and every instance created in the previous step.
- Once we have the distances, the one more similar to the instance with the missing value will be passed to the discriminator network, and the system will predict the result of that generated instance. The output of the predict function will be the value that replaces the missing value.
- The previous step will be repeated for every missing value of the attribute that the system is handling.

Finally, this procedure will be repeated with all the attributes that the system needs to impute.

10.5.5 Implementation of models and preparation of data

In order to simplify the code in this part of the system and maintain a code that is almost generic for most the models that are implemented, an interface for this models have been defined. This interface defines two public functions, the first one is *fit*, that trains the computational model while the function *predict* predict the label of the instance that is passed by argument.

The other functionality that this system handle is that it can structure any dataset if it has been previously encoded as it is explained in the subsection 10.4.1 of this chapter. The system groups the categorical attributes which are from the same original attribute, and then, it prepares the data to get the position of missing values in the attribute. Knowing their positions help to organize which instances will be for training data and which ones will be for predicting (instances with missing value).

Another important functionality of this part of the system is the module that converts the numerical data into the standards of the original dataset. By standards, we

mean to convert the numerical attributes to their scale, using the class *MinMaxScaler*, from the preprocessing module of scikit-learn. To do so, the system use the function *inverse_ transform* to undo the process of normalization. For categorical attributes, in this case, the system use the class *OneHotEncoder*. Although it was rejected for converting label data to numerical values, it works perfectly for undoing the one hot encoding. For undoing this encoding, the system also uses the function *inverse_ transform*, which will return a list of numbers from 0 to $values - 1$. Luckily, it follows the same guidelines as Weka, so after getting this value, we replace it by their label value.

10.6 Generation of associative rules module

To generate the set of associative rules, we have used Delgado-19 [29] approach. The process for the generation of these rules appears in the figure 10.3.

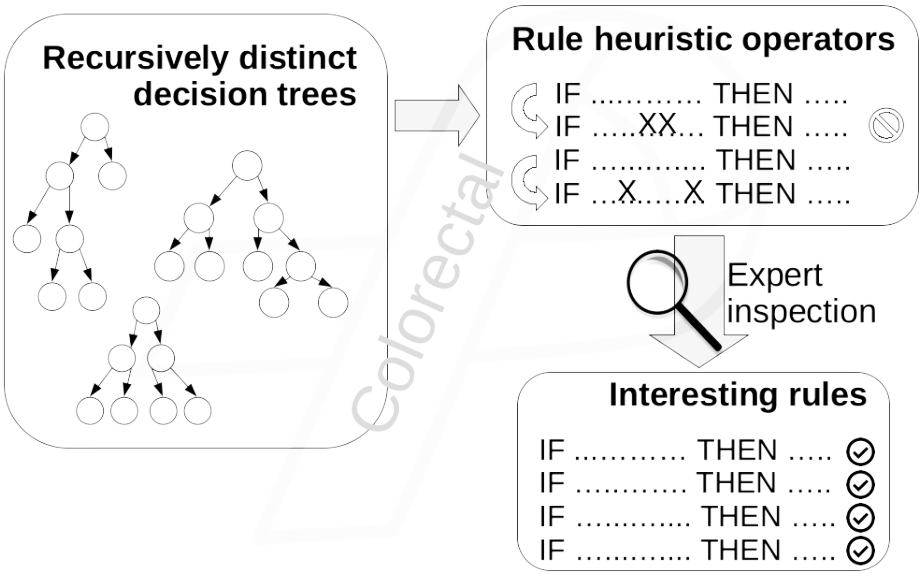

Figure 10.3: Process to generate rules

As it shows figure 10.3, a number of distinct decision trees are recursively generated in which the attribute in the root node is dropped everytime in order to generate new different trees. Once the trees have been generated and the rules obtained, we apply a list

of operators to the set rules.

The objective of applying these operators is to generate more rules apart from the ones we got running CART algorithm recursively. This methods are run after each other and it follows the order of the itemization. The list of operators are the following ones:

- **Branch simplification method**: This method consists of removing conditions from the rule and checks if the modification of the rule has made any improvement comparing the metrics of the original rule. If an improvement has been made, the new rule is added to the set of rules. In case of dealing with categorical attributes in the rule which takes two values or more, this operator also tries to take out one of the values in every step also to check if there are improvements.
- **OneCatValue method**: This methods is based on modifying the categorical conditions of the rule (if it has one), setting in every step one value of the categorical condition. If the metrics of the new rule (confidence and number of positive cases) is above a certain threshold (in this case the minimum confidence is 0.8 and the minimum of positive cases is 7), the rule is simplified with the previous operator. We only use this method for those rules that have been generated by the branch simplification method.
- **Optimization method**: This method is based on dropping one condition of the rule at each step. If the new rule has a confidence that is above a defined threshold (in this case 0.7), the rule is simplified using the branch simplification method. We only used this method for those rules that have been generated by branch simplification and OneCatValue method.

Once that the rules have been generated, they should be validated by an expert to identify which rules can be classified as interesting to the expert.

Chapter 11

Experiments

In this chapter, we present the empirical results of the described methodology on our dataset. In all the experiments, the dataset was previously normalized to make the numeric features be in the range $[0, 1]$. Section 11.1 presents the results for looking the appropriate architectures. In section 11.2, we compare the different methodologies to recover missing values using a cross validation with 10 folds. Finally, in section 11.3, we discuss the new rules that the system has generate with the dataset with recovered which have been recovered with the best methodology explained in 11.2.

11.1 Network selection

In this section, we describe how we proceeded to obtain a deep learning based model for imputing missing values in our dataset. The process consisted of the following steps:

1. *Basic preimputation*: Artificial neuronal networks require input values for all the features of the patterns, i.e., they can not work with missing values. Therefore, an initial imputation is needed to make them work. We have considered three alternatives

    - Feature removal: those features with missing values are removed from the

dataset and the neural network is trained with the remaining ones.

- Means and modes: missing values for numeric features are estimated according to the mean of the present values (which were previously normalized into [0,1]), and those for categorical features according to the mode of the values.
- Null values: missing values are replaced by null ones. In particular, since all numeric features take positive values, we set missing ones equal to minus one; and those for categorical features are marked as *missing*.

2. *Network architecture*: Then, we aim at finding a deep learning network architecture, not only capable enough of learning the distribution of the patterns in the dataset, for correctly predicting the goal feature, but also as simple as possible to be efficient and less susceptible to overfitting. Therefore, we test several architectures with different degrees of complexity (number of layers and neuron units), to finally select the simplest one that provides a sufficiently high re-substitution accuracy on the predictions on the whole dataset.

   These network models differ in the number of units in the first layer (all of them ReLU dense layers [37]), which is indicated in the corresponding experiment. The rest of the layers are either a *dropout* layer, with a 5% rate, between any two dense ones (to try to prevent overfitting), *dense* ones with half the number of units in the previous dense layer (until having less than 10 units), or a *sigmoid* one at the end (no dropout is considered between the two last layers, dense and sigmoid). All these models are trained, with Adam optimizer, with 0.001 as the learning rate and a batch size of 64, until producing a 100% re-substitution accuracy or a maximal number of 100 epochs.

3. *Network selection*: At this stage, we have as many network models as initial imputation techniques, and we are interested in those with good generalization capabilities. Therefore, we compare their performances according to a 10-fold cross-validation and the best one is selected.

4. *Network based missing values imputation*: The network model is trained and used for imputing the missing values of those features with 200 to 350 of such values, which correspond to ratios of 13% to 23%. This way, 9 out of the 77 features with missing values undergo imputation. Features with a ratio outside that interval undergo the previous basic imputation. Our hypothesis is that features with too many

missing values have not got enough information to train the network model, which would subsequently be used to estimated the value of the vast amount of missing values. On the other hand, initial experiments showed that imputing features with few missing values, by means of our model, provided minimal improvements in the final classification and required more computational resources than using the basic techniques. The last sigmoid layer is replaced by a *softmax* one, in case of estimating categorical features, or a linear activation for numerical features.

5. *Empirical comparison*: Finally, several neural networks, with the same previous architecture, are trained on either network-based imputed dataset or others with classic missing values imputation techniques. These networks are compared according to a 10-fold cross-validation to test whether our network-based imputation approach provides sufficiently good results.

First of all, we need to generate three new databases, as it is explained in step 1. In the first new database, missing values from numerical attributes have been replaced with -1 whereas in categorical attributes, they have been replaced with 0. In the second new database, missing values of numerical attributes have been replaced with the mean of the values of the attribute and the unknown values from categorical attributes have been replaced with the mode of values of the attribute. In the third database, those attributes with missing values have been erased.

Once we have the new databases, we need to find the simplest neural networks structure that has the ability of learning all the data (reaching a CCR of 100% at training) for each one of the databases. The output of this process will give three neural networks, the simplest one for the database which missing data is replaced with 0s and -1s, the simplest for the database which missing values are replaced with the values of mean and mode and the simplest one for the database in which attributes with missing values have been deleted. The neural network with the best performance will be chosen as the one used to replace the missing values of the original database.

The method used to generate neural network structure consists of defining the number of neurons of the first hidden layer of this network and then divide the number of neurons of the previous hidden layer by two in order to get the next hidden layer. Furthermore, each two hidden layers, a Dropout [39] layer is inserted to prevent the overfitting of the

neural network. The last layer of the neural network will always have 1 neuron due to this is 2-class problem. The activations of the neurons of the hidden layers are ReLU [37] while the neuron from the last layer uses a sigmoid activation, which has a good performance with 2-class problems [40].

Table 11.1 shows the resubstitution Correct Classification Rate (CCR) and consumed epochs of the network architectures described in step 2. Previously, the corresponding basic preimputation and the one-hot encoding were applied on the dataset.

	Basic preimputation					
	Null values		Means & modes		Feature removal	
First layer	CCR	Epochs	CCR	Epochs	CCR	Epochs
600	100%	21	100%	30	64.31%	100
500	100%	29	95.84%	100	64.18%	100
400	100%	45	95.84%	100	64.31%	100
300	96%	100	96.9%	100	63.72	100
150	97.43%	100	89.51%	100	62.47%	100
100	100%	17	100%	32	64.18%	100
50	100%	27	100%	42	64.78%	100
24	**100%**	**46**	100%	55	63.32%	100
14	100%	66	**99.87%**	**100**	63.39%	100
12	89.91%	100	95%	100	62.86%	100

Table 11.1: Resubstitution CCR and epochs per network architecture

We observe that all the networks attain resubstitution results close to 100%, with the exception of those with feature removal preimputation and perhaps three of the others with accuracies below 95%. This means that the network architectures with null values or means & modes preimputation, even the simplest ones, are able to adapt to the distribution of the patterns. For the following network selection, we chose the architectures with values in boldface. The reason is that they are the simplest ones almost reaching the 100% CCR. In the case of using null values, we opted for the network with 24 neurons in the first layer because it reached the maximal accuracy in less epochs, even though these results probably have some dependencies with the stochastic nature of the learning algorithm. Therefore, the network architectures selected for the following stages have:

- For the case with null values preimputation, 24 and 12 neurons in the dense layers, a dropout layer between them, and 1 neuron in the final sigmoid layer.

- For the case with means and modes preimputation, 14 and 7 neurons in the dense layers, a dropout layer between them, and 1 neuron in the final sigmoid layer.

- No network with feature removal was selected due to their poor performances.

Once we have found the simplest architectures that can attain the distribution of the patterns, we compare them according to their generalization capability. Thus, we train and compare these architectures according a 10-fold cross-validation. In this case, the networks were trained a maximum number of epochs (80), but 10% of the training examples were put aside to measure the validation evolution. As soon as 20 consecutive epochs did not improve the validation error, the training was stopped and the network with the best validation error was returned. In addition, we have considered another dropout rate to check its influence in the generalization capability of the networks.

Neural Network	Dropout	Mean
$24 \rightarrow 12 \rightarrow 1$	0.01	61.80%
$\mathbf{24 \rightarrow 12 \rightarrow 1}$	**0.05**	**62.66%**
$14 \rightarrow 7 \rightarrow 1$	0.01	62.06%
$14 \rightarrow 7 \rightarrow 1$	0.05	62.26%

Table 11.2: Cross-validation CCR results of the best previous architectures

Table 11.2 shows the results. Notice that these cross-validation CCR values, much worse than the previous re-substitution ones, evidence the complexity of our problem where a limited number of patterns were given and there were many missing values. This means that, even though previous architectures were able to learn the distribution of the patterns, there are many cases with subtle differences that, once put aside from the training set, belong to a distribution that is not well represented by those kept in the training set. And this fact can not be solved with more complex architectures, because their capacity to represent the patterns distribution was not better than those of the selected architectures.

To compare the performance of our methodology for imputing missing values, with other widely used in the literature, we have chosen the network with 12 units in the first layer and a dropout rate of 5%. Although its results were very similar to those of the other networks, it turned out to be the best ones according to Table 11.2.

11.2 Comparison with other imputation techniques

In this section we analyze the capacity of our methodology for imputing missing values, in a profitable way, in our dataset. First, we used the model selected in the previous stage to impute the missing values of those features with 200 to 350 of such values. As commented, this requires changing the last sigmoid layer for a softmax or a linear one according to the feature being addressed. Then, that model is trained on the set of the other features with basic preimputation, and used to predict the value of those values missing. Subsequently, the same model from the previous stage is trained on the imputed dataset and its cross-validation CCR is compared with that of also the same model with other imputation techniques from the literature. In all the cases, numeric features of the dataset are normalized into $[0, 1]$. The considered imputation techniques are:

- **Basic preimputation**: Null values and means & modes replacement of missing values.
- **kNN imputation**: In this case, the missing value is estimated according to that of the k closest patterns. We have used k equal to 5. Besides, kNN needs to be combined with a preimputation technique in order to properly compute distances, so three versions of kNN are considered: kNN with feature removal, with null values preimputation and with means & modes preimputation.
- **Multiple Imputation by Chained Equations**: With this technique, we combine it with kNN and network base imputation model. We have set 10 cycles. This technique generates four new imputation techniques that are displayed in table 11.3.
- **Generative Adversarial Networks**: Generation of missing values according to the value we want to recover from the fake instance that is most similar to the one which has the unknown value.
- **Network based imputation**: For the sake of coverage, we also include some of the network models that had been being discarded in previous stages, namely, that with feature removal, referred to as NN-feature removal; and that with null

values preimputation, referred to as NN-null values. Notice that the network model with means & modes preimputation, which is referred to as NN-means & modes, is precisely the selected in our methodology.

The procedure that has been used is cross validation with 10 folds, where each fold has around 152 records from the database. The neural network has been configured with 80 epochs and a batch of 64 records. In order to stop the neural network at its best epoch, an early stopping has been set in which the patience, number of epochs we wait for the neural network to improve its validation results, and a threshold to determine what type of variation of results is an improvement. Setting a Early Stopping entails taking some of the records for the validation set, in this case, a 10%. The previous constraints have been configured in the deep neural network using scikit-learn [34], a machine learning library for python as also as Keras [31], a deep learning library, which has been used for running all deep neural networks implemented for this paper.

Imputation technique	Mean (Std)
Feature removal	58.04% (0.0446)
Null values	62.20% (0.0277)
Means & modes	62.66% (0.0329)
kNN-feature removal	61.55% (0.0270)
kNN-null values	62.00% (0.0214)
kNN-means & modes	62.60% (0.0261)
NN-feature removal	59.86% (0.0248)
NN-null values	63.13% (0.0273)
NN-means & modes	**63.78% (0.0331)**
MICE-NN-null values	63.38% (2.4312)
MICE-NN-means & modes	**64.31% (2.6057)**
MICE-kNN-null values	62.26% (3.6061)
MICE-kNN-means & modes	63.51% (3.3949)
GAN-null values	62.06% (0.02721)
GAN-means & modes	63.18% (0.02360)

Table 11.3: Cross-validation CCR of the $24 \rightarrow 12 \rightarrow 1$ network trained on different imputed datasets

Table 11.3 shows the results. As we observe, the dataset imputed by means of the model produced by our methodology allows the neural network to attain the highest cross-

validation CCR, more than one point above those produced by non-network models and more than half point above that produced by the network with null values preimputation. On the other hand, we carried out the Wilcoxon statistical analysis [41] between NN-means & modes and each of the other alternatives, but only obtained p-values below 0.05 with the techniques of feature removal and kNN-feature removal. Therefore, we can conclude that our methodology may stand as a potentially interesting missing value imputation technique for other cases, though great statistical differences were not often found.

Another and curious observation is that kNN techniques did not improve the results of the basic imputation methods, which the network-based ones did.

11.3 Generated Rules

Here we present the rules we have generated with the best methodology pointed in table 11.3, which is our neural network imputing uninteresting attributes with means and modes used in Multiple Imputation by Chained Equations. We have set 5 as the maximum length of the rules we are generating and we are also adding the generated rules to a buffer of rules in order to discard old rules.

First we have run the rules generation module with the original dataset just to save the rules in the buffer. Doing this actions avoids the new dataset that we have created with the recovery data module to output those same rules, and be saved in the CSV output file. The rules which have not been generated previously by the original dataset but with the imputed dataset are the following ones:

RuleID	Antecedent	Confidence	Coverage
1	tamano_pieza $>$ 27.25 and p_urea $\geq$ 35.5 and m_circunferencia $<$ 5.5	72%	0.087
2	exeresis_mesorrecto = total and rescate = false and imc $\geq$ 28.23	72.22%	0.084
3	exeresis_mesorrecto = total and anastomosis = no and distancia_ano $\geq$ 6.5	80.76%	0.068
4	tamano_pieza $>$ 27.25 and distancia_ano $<$ 6.5 and colonoscopia = incompleta and margen_distal $>$ 1.45	72.58%	0.073
5	exeresis_mesorrecto = total and p_leucocitos $<$ 9885 and imc $\geq$ 28.23 and margen_distal $>$ 1.45	70.68%	-

Table 11.4: Generated rules table 1

RuleID	Antecedent	Confidence	Coverage
6	exeresis_mesorrecto = total and neoadyuvancia = false	65.71%	-
7	anastomosis = no and presentado_scccr = si and tamano_tumor ≥ 3.75	72.72%	0.064
8	exeresis_mesorrecto = total and imc ≥ 28.08	67.08%	-
9	exeresis_mesorrecto = total and imc ≥ 28.23	66.67%	-
10	distancia_ano < 13.5 and imc ≥ 28.68 and tamano_pieza ≥ 14.5	67.04%	-
11	tamano_pieza ≥ 27.25 and p_ecg = otros and m_circunferen ≥ 1.645 and	72.54%	0.06
12	exeresis_mesorrecto = total and p_leucocitos < 9885 and imc > 28.08	72.34%	0.055
13	tamano_pieza ≥ 27.25 and ap_tipo_histologico = Adenocarcinoma and p_ecg = otros	70%	0.056
14	tamano_pieza ≥ 27.25 and ap_tipo_histologico = Adenocarcinoma and p_ecg = otros and m_circunferencia ≥ 1.645	82.85%	0.047
15	exeresis_mesorrecto = total and p_leucocitos < 9885 and rescate = false and imc ≥ 28.23	77.35%	-
16	p_ecg = otros and ayudante = 12	70.27%	0.042
17	sexo = hombre and p_ecg = otros and ayudante = 12	81.48%	0.035
18	exeresis_mesorrecto = total and distancia_ano > 1.5 and imc ≥ 28.35	69.44%	-
19	sexo = hombre and grado_diferenciado = bien and reseccion_cilindric = no	80%	0.032
20	laparoscopia = no and eco_endorrectal = no_realizada and p_urea ≥ 36.5	84%	-
21	exeresis_mesorrecto = total and p_fc ≥ 66.5 and margen_distal ≥ 1.8 and p_edad < 78	67.30%	-
22	sexo = hombre and grado_diferenciado = bien and reseccion_cilindric = no and estado_donuts = no_aplicable	87.5%	0.022
23	estado_t = pT4b and demora_quirurgica ≥ 37	76.19%	0.025
24	laparescopia = no and ileostomia = no_aplicable and eco_endorrectal = no_realizada and ap_tipo_histologico = adenocarcinoma	92.3%	0.019
25	estado_donuts = no_aplicable and eco_endorrectal = no_realizada and invasion_serosa = si	73.68%	0.022
26	laparoscopia = no and eco_endorrectal ∈ {no_realizada, uT4N0} and p_urea ≥ 36.5	85.71%	0.038
27	tumor_sincronico = si and p_leucocitos < 6415	80%	0.019
28	tamano_pieza ≥ 27.25 and estado_t = pT2 and demora_quirurgica ≥ 39.5	72.22%	0.0211
29	exeresis_mesorrecto = total and perforacion_io = Si	72.22%	0.0211
30	tamano_pieza ≥ 27.25 and eco_endorrectal = no_aplicable and localizacion = angulo_esplenico and p_edad < 79.5	84.61%	0.01785
31	eco_endorrectal ∈ {no_realizada, uT4N0} and p_urea ≥ 36.5	68.75%	-
32	tamano_pieza ≥ 27.25 and localizacion = angulo_esplenico and p_edad < 76.96	70.58%	0.01948

Table 11.5: Generated rules table 2

RuleID	Antecedent	Confidence	Coverage
33	tamano_pieza ≥ 27.25 and estado_t ∈ {pT1, pT2, pT3b, pT4a, pT4b} and demora_quirurgica ≥ 37	77.14%	0.087
34	tumor_sincronico = Si and p_ecg = Otros and p_leucocitos < 6415	90%	0.01461
35	tamano_pieza ≥ 27.25 and distancia_ano < 82.5 and estado_t = pT1	90%	0.01461
36	sexo = hombre and ayudante = 12 and p_perdida_hematica = 501-1000_ml	90%	0.01461
37	tumor_sincronico = No and eco_endorrectal ∈ {no_realizada, uT1N0, uT4N0} and p_fc ≥ 73.5 and cea ≥ 0.5	68.62%	-
38	ayudante = 12 and ap_tipo_histologico = adenocarcinoma and p_disnea = Leve	81.81%	0.01461
39	anastomosis = No and cirujano = 29 and ganglios_aislados < 11.5	81.81%	0.01461
40	tamano_pieza ≥ 27.25 and p_urea ≥ 36.5 and ganglios_aislados < 3.5	68.91%	-
41	p_tipo_iq = Mayor and estado_donuts = no_aplicable and eco_endorrectal = no_realizada and invasion_serosa = Si	88.89%	0.01298
42	tamano_pieza ≥ 27.25 and p_urea ≥ 36.5 and grado_diferenciado = Bien	88.89%	0.01298
43	tamano_pieza ≥ 27.25 and estado_t = pT4b and demora_quirurgica ≥ 37	88.89%	0.01298
44	tamano_pieza ≥ 27.25 and estado_t ∈ {no_aplicable, pT1, pT2, pT3b, pT4a, pT4b} and demora_quirurgica ≥ 39.5	88.89%	0.01298
45	eco_endorrectal = no_realizada and p_urea ≥ 36.5	69.76%	-
46	p_ecg = Normal and estado_donuts = no_aplicable and eco_endorrectal = no_realizada and invasion_serosa = Si	100%	0.01136
47	tamano_pieza ≥ 27.25 and distancia_ano < 82.5 and colonoscopia = Incompleta and estado_n ∈ {pN0, pN2}	72.72%	-
48	exeresis_mesorrecto = Total and ap_tipo_histologico = adenocarcinoma_mucinoso and tipo_histologico = adenocarcinoma and p_edad < 78.5	75%	0.01461
49	tamano_pieza ≥ 27.25 and estado_t ∈ {pT1, pT3b, pT4, pT4a, pT4b} and imc ≥ 27.05	75.86%	0.07142

Table 11.6: Generated rules table 3

Chapter 12

Conclusions and future work

In this chapter, we explain the knowledge that we obtained in the development of this project as well as the future work concerning this project.

12.1 Conclusions

In this thesis we have addressed the application of deep learning techniques to a Colorectal Cancer database. The main objective, explained in chapter 4, was to obtain interpretable descriptions from this database enhanced with deep neural networks by recovering missing values from this dataset.

By implementing a system which consists of a preprocessing module, a recovery of data module and a generation of rules module, we have fulfilled the main objective of this project as well as its sub-objectives.

Furthermore, by generating associative rules with the dataset containing imputed values, we have generated new sources of knowledge which are useful for doctors and investigators. But, in a way, we have also been evaluating the performance and the accuracy of the deep neural network that we are using. This last thing is relevant because deep neural networks are a black box model, which complicates the process of interpretation of

their results.

12.2 Future work

While working on this project and after finishing it, we have thought in several investigation items although they were not in the baseline of this project.

The first item in which we have thought is in applying the techniques used in this project with other datasets. Due to every dataset is completely a new problem, we do not know if our techniques could be useful for different datasets. In addition to that, the architecture of the neural network that we have used would not probably be the adequate.

Another item we have discussed is the existence of another imputation methodologies to recover missing values and their appliance to this dataset. We have used deep neural networks as well as Generative Adversarial Networks, Multiple Imputation by Chained Equations and K-nearest neighbours imputation. It will be interesting to study another methods recently published to measure their quality when applying them to our dataset. One example of novel methodologies can be this publication [42], which imputes data depending on the correlation of the instances which have not got any missing value, orthis publication [43], in which a iterative imputation methodology based in a random forest is explained.

Finally, the last item we have discussed that will be interesting to research is how to interpret the results of neural networks in order to measure their performance. As it was said before, they are black box algorithms, which means that we can not supervise how they make the results.

Part II

User documentation

Chapter 13

Introduction

The objective of this part is to explain to the final user how to use the system in order to enhance the database for getting new association rules. It is divided in several chapters, in which we expose the hardware and software requirements, the way to install the dependencies used by the software and its usage.

In chapter 14, we define the hardware requirements to run the software. These requirements will be the minimum required in order to run it.

In chapter 15, we define the software requirements in order to execute this software. Due to the constant change of the packages that have been used, we will recommend the versions we have used to run the experiments.

Finally, in chapter 16 and 17, we explain how to set up the software to run it using a tool to manage the installation of the dependencies and execution of the software. In case that the user wants to install and execute it in a specific system, we have explained how to do it in Linux based systems (we have chosen Ubuntu because it is the most used) and in macOS operating system.

Chapter 14

Hardware Requirements

Although most of the computers are recently new and they have not got problems installing standard software, we intend to give a list of recommendations about the hardware to run these experiments.

The recommended configuration of the hardware is the following:

- **CPU**: It is recommended to have an AMD or Intel CPU due most of the tools use specific CPU instructions related to previous ones. Furthermore, the x86_64 is recommended due to the same reason and it is recommended to have at least 4 cores.
- **GPU**: The GPU is never used to run the experiments but it is heavily recommended when running the computational model. If used, it is mandatory to use a NVIDIA GPU card with CUDA Compute Capability 3.5 or higher.
- **RAM memory**: It is recommended to have at least 8 GigaBytes of memory because the software used takes at least 1 GigaByte. It is preferred to have DDR4 RAM type to reduce the time of the execution.
- **Storage**: It is recommend to have at least 10 GigaBytes free in the hard drive in order to install all the software and its dependencies.

Chapter 15

Software Requirements

Another important thing to run this project is the software requirements. As it was explained in the introduction of this part, we are going to display the software versions that we have used to run the several experiments which involve this project.

The software requirements of this project are the following:

- **Operating System**:
  - Ubuntu 18.04 or higher
  - macOS 10.15 or higher
- **R Programming Language**: It is recommended to install version 3.6 or higher.
- **RStudio**: It is recommended to installed version 1.2 or higher.
- **R packages**:
  - **Arules**: Version 1.6-6
- **Weka**: Version 3.6 or higher
- **Python**: It is mandatory to use the version 3.7
- **Python dependency manager**: Pipenv version 2018.11.26
- **Python packages**:

  - **autopep8**: Version 1.5.2
  - **liac-arff**: Version 2.4.0
  - **numpy**: Version 1.18.4
  - **scipy**: Version 1.4.1
  - **scikit-learn**: Version 0.23.0
  - **tensorflow**: Version 2.2.0
  - **Keras**: Version 2.3.1

In case of running the computational models in the GPU, the following software is required:

- **NVIDIA GPU Drivers**: Version 418.x or higher
- **CUDA Toolkit**: Version 10.1
- **cuDNN SDK**: Version 7.6 or higher
- **TensorRT**: Version 6.0

Chapter 16

Installation

In this chapter, we highlight the steps to install the software used to make possible this project. It will explain how to install Python and its virtual environment and R. About the installation of Rstudio (its IDE) and Weka, the steps for their installation are explained in their user's documentation, but we have explained how we proceeded to install them in order to develop our project as an example.

16.1 Weka

16.1.1 Linux

In order to install weka in a Linux based system, the program must be downloaded from the following url. Once downloaded, it should be unzipped and to run it, the following commands need to be executed:

```
user@computer$ chmod u+x weka.sh
user@computer$ ./weka.sh
```

16.1.2 macOS

In order to install weka in a macOS, the program must be downloaded from this url. Once downloaded, it must be executed and then, follow the instructions showed in the prompt.

16.2 R and RStudio

16.2.1 Linux

To install R in Linux base systems, we need to run the following commands:

```
user@computer$ sudo apt-key adv --keyserver keyserver.ubuntu.com --
    recv-keys E298A3A825C0D65DFD57CBB651716619E084DAB9
user@computer$ sudo add-apt-repository 'deb https://cloud.r-project.org/
    bin/linux/ubuntu bionic-cran40/'
user@computer$ sudo apt update
user@computer$ sudo apt install r-base
```

Once we have R installed, we must install RStudio executing the following commands:

```
user@computer$ curl https://download1.rstudio.org/desktop/bionic/amd64/
    rstudio-1.2.5042-amd64.deb --output rstudio.deb
user@computer$ sudo dpkg -i rstudio.deb
```

16.2.2 macOS

In order to install R in macOS, the following file must be downloaded and executed it. Once executed, a prompt will appear to install it in a graphical way.

After installing R, now we can install RStudio from the following link. One executed, a prompt will appear to execute it.

16.3 Python and Virtual Environment

16.3.1 Linux

For installing Python in linux, the following command must be executed in the terminal:

```
user@computer$ sudo apt install python3
```

Once we have python installed, we must install pipenv, which is a tool that is in charge of the dependencies of python. To install it, the following command must be executed:

```
user@computer$ sudo apt install pipenv
```

After installing, now we can install the dependencies of the python project that is in charge of recovering the missing values of the dataset. To do so, the following command must be executed in the root of the project:

```
user@computer$ pipenv install --dev
```

16.3.2 macOS

For installing Python in macOS, the following command must be executed in the terminal:

```
user@computer$ brew install python3
```

Once we have python installed, we must install pipenv, which is a tool that is in charge of the dependencies of python. To install it, the following command must be executed:

```
user@computer$ brew install pipenv
```

After installing, now we can install the dependencies of the python project that is in charge of recovering the missing values of the dataset. To do so, the following command must be executed in the root of the project:

```
user@computer$ pipenv install --dev
```

Chapter 17

Usage

In this chapter, we pretend to give a list of steps that the user should do in case of running the software of this project. In order to reproduce the experiments, the user can pre-process the data using Weka before the script for recovering data is executed. After getting this data, it will processed by the R script to generate associative rules. This steps will be described carefully in the following sections.

17.1 Preprocessing the dataset

For preprocessing the dataset, we have used the tool called Weka. Weka is a software that is used for preprocessing data, running classification algorithms as well as a visualization tool for data. All those features are accesible by an user interface.

17.1.1 Loading files in Weka

When opening Weka, the user have the possibility of using several applications. In this case, we will use Explore, that gives the user the opportunity of preprocessing the dataset. Once in the application, the user can see that the actual view of the window is preprocess. In this window, the user will be able to open their files containing the data. In order to

do so, the user must click in ***Open file***. After this action, a prompt window will appear to select the file that the user want to open, as in figure .

Weka admits several types of files for representing data. It provides the user two formats of their own to represent data: Attribute-Relation File Format (ARFF) and XML attribute Relation File Format (XRFF). Although this type of formats are not really known, weka provides a set of converters that lets the user load files with C4.5 and CSV format files. However, it is heavily recommended to use their own formats to avoid problems when loading data from another type of file formats.

17.1.2 Applying filters to the dataset in Weka

Once the user have load the information of the patients have suffered from Colorectal Cancer, it is time to preprocess the data in order to prepare it for the python script, which is in charge of running the computational models. The first filter that the user must apply is to convert the categorical attributes into a one hot encoding. To do so, the user must click in Choose and select the filter called “NominalToBinary", which is placed inside the attribute folder, which is stored in the unsupervised folder.

After this, the user must configure the filter that is really important for the python script. To do that, the user must click in the text box that contains the name of the filter. Once clicked, the following window will appear to configure the filter.

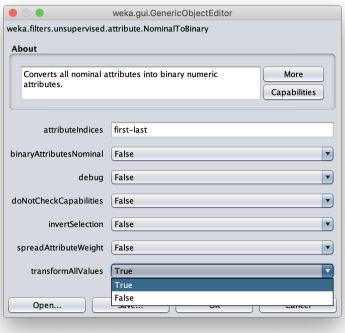

Figure 17.1: Setting up the “NominalToBinary" filter

As figure 17.1 shows, the user must change the value of the attribute “TransformAll-Values" to ***True***, which leads to make a real one hot encoding of the categorical attributes. In case it took the value ***False***, if exists categorical attributes which only can take two values, it will be transformed in only one attribute instead of two. Once the user have configured it, they should click ok to save the configuration, and then click in ***Apply*** to apply the filter to the dataset.

After applying this filter, the user must normalize the data. In order to do so, the user must search for the “Normalize" filter, which is inside the attribute folder, which is stored in the unsupervised folder. Once selected, the filter is applied when the user clicks in ***Apply***.

17.1.3 Saving the new dataset in Weka

Finally, after preprocessing the dataset must save the data clicking in ***Save***, which will open a window to give a name to the file with the new data. The format type of the output file will be ARFF.

17.2 Recovery of missing values

After preprocessing the dataset, now the user can run the python script to recover the missing values of the dataset. For the execution of the script, the user must run the following commands from the root of the repository:

```
user@computer$ cd src/
user@computer$ pipenv run python main.py <zeros/mean> <knn/nn/each/
    gan> output.arff <chained_cycles>
```

As it shows the console output, the script receive a list of arguments, which definition are the following:

- <**zeros**/**mean**>: This argument selects the default methodology to recover the missing values of those columns which are not interested (those who have less than 200 missing values or more than 350 missing values). If zeros methodology is selected, the missing values from numerical attributes are replaced by -1 whereas those attributes from categorical attributes are replaced by 0. If mean methodology is used, the missing values from numerical attributes will replaced by the mean of the column and the unknown values fro categorical attributes will replaced with mode of the values of the column. If any of those methodologies are chosen, the execution of the script is going to be halted. This argument is mandatory.

- <**knn**/**nn**/**each**/**gan**>: This argument selects the default methodology to recover the missing values of those columns which are interested to be replaced(those who have between 200 missing values and 350 missing values). If kNN is selected, a K-nearest neighbours model is used to predict the missing values where $k = 5$ and the weight used is the euclidean distance between the instances of the dataset. If the nn methodology is selected, a *fixed* deep neural network model will be in charge of recovering the missing values of the dataset. In case that the each methodology is selected, a search of the best architecture is done, so the best neural network for each attribute will be used to replaced the unknown data of the column. Finally, if gan methodology is selected, the missing values will be imputed as it is explained in subsection 10.5.4. If any of them is selected, the execution of the script is going to be halted. This argument is mandatory.

- **output.arff**: This argument defines the output file of the dataset without missing values. This argument is mandatory.

- **chained_cycles**: This argument defines the number of cycles for Multiple Imputation by Chained Equations (MICE). If the number of cycles is one, it is just an standard imputation but if it higher than one, the MICE methodology is executed. This argument is optional. In case that it is not set, it will take a one as its default value.

When the execution of the script is finished, the user can find that it has generated two output files. The first one is the indicated at the argument line of the script, which includes the dataset preprocessed without missing values. The second one that is generated is the

same file as the dataset without the preprocessing of the data. The unique difference it has is that those columns which were interested to be replaced have their missing replaced. The numerical values replaced have been set to the original scale of their respective attributes whereas the categorical values replaced have been set to its text values (doing the inverse action of an one hot encoding). This second file will be used by the R script for generating the associative, which is going to be explained in section 17.3.

17.2.1 Examples

In this subsection we give a set of examples to execute the recovery of data module in order to help the user. If the final user wants to recover the missing values of the uninteresting attributes with null values and the interesting value with a deep neural network model, the following command must be executed:

```
user@computer$ pipenv run python main.py zeros nn youroutputfile.arff
```

But in case that the user wants to generate a dataset in which the unknown values from uninteresting values will be imputed with the means and modes, the user must execute this command:

```
user@computer$ pipenv run python main.py mean nn youroutputfile.arff
```

In case of recovering missing values of interesting attributes with another model, the user should change the keyword nn by knn, each or gan. Another alternative that the user can execute is Multiple Imputation by Chained Equations defining the last argument of the script as in the next command:

```
user@computer$ pipenv run python main.py mean nn youroutputfile.arff 7
```

If the user executes MICE with one cycle, as it shows the next command, it will be

executing the same behaviour as in the second command of this subsection.

```
user@computer$ pipenv run python main.py mean nn youroutputfile.arff 1
```

17.3 Generating associative rules

Once we have the file with the dataset prepared for the R script, we can run it. For executing it, it is recommended to open Rstudio and open the repository as a project, as it shows the figure 17.2.

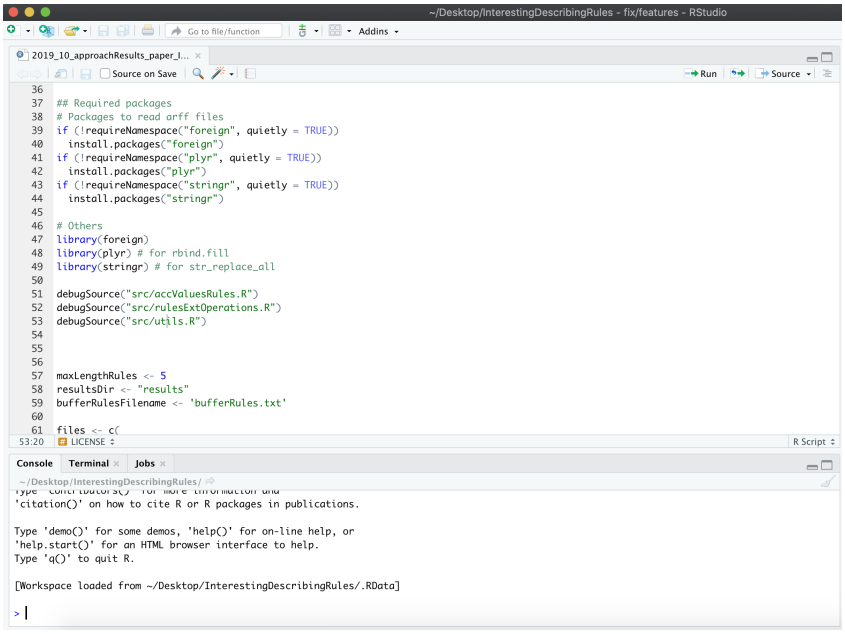

Figure 17.2: Project in Rstudio

To execute each line of the R Script, the user must press the combination of letters Ctrl+Enter in Linux and Command+Enter in macOS. This script can also be executed by terminal with the following command:

```
user@computer$ Rscript generateRules.R
```

But also, it can be executed as it follows:

```
user@computer$ R CMD BATCH generateRules.R
```

The most important lines that configure the behaviour are the following:

```
1 maxLengthRules <- 5
2 resultsDir  <- "results"
3 bufferRulesFilename <- "bufferRules.txt"
4
5  files  <- c(
6    "originalDataset.arff",
7    "datasetWithReplacedValues.arff"
8 )
```

Listing 17.1: Configuration of generation of rules

The first three lines defines the maximum length that the rules are going to have, the directory in which the rules that have been created will be saved (it is mandatory to create the directory before executing the script) and the name of the buffer that will contain the rules which have been produced and that will not be saved in the list of rules generated by the script.

Finally, in the last four lines, we have a vector which have the name of the files that will be used to generate the rules. In this case and taking in account the information given in the last paragraph, in the execution of the script, when it is generating rules for the file “datasetWithReplacedValues.arff", the rules generated by this one that has been generated by the previous file will be discarded from saving it in the list of rules generated by “datasetWithReplacedValues.arff".

Part III

Code documentation

Chapter 18

Introduction

In this part of the project, we list the code that has been produced in order to make possible this project. Each section of chapter 19 represents one file of the repository written in Python, and this code is in charge of retrieving the missing values of the database using several techniques as kNN, Deep neural networks, Multiple Imputation by Chained Equations and Generative Adversarial Networks.

The source code can be also found in the following github repository `https://github.com/danitico/Colorectal-deep-learning`.

The source code for generating rules can be found in `https://github.com/cgarcia-UCO/InterestingDescribingRules`.

Chapter 19

Code

As it was explained in the introduction of this section, each section of this chapter represent a file that belongs to the functionality of recovering missing values in the dataset. An explanation of each file will be displayed in this section.

19.1 main.py

This is the main file of the repository that must be executed by the user. It is the module that uses the rest of files and guides the execution of the recovery of data module. The main actions of this file is to handle the arguments that the user define to execute this program. In case that the number of the arguments are not the required, it executes the program. It is in charge of reading the ARFF file which contains the information about patients and defines the range of missing values to consider an attribute interesting. Furthermore, it differentiates between those columns which are recovered easily and those recovered by computational models. But the most important thing that it does is that it separates each column in order to prepare them for the computational models. Finally, it outputs the dataset into an ARFF file without missing values.

```
1  import re
2  import sys
3  import warnings
4
5  import arff
6  import numpy as np
7  from scipy.stats import mode
8
9  from models import (classificationModelkNN, classificationModelNN,
10                      regressionModelkNN, regressionModelNN, bestModelAttribute,
11                      classification_gan_imputation_model,
12                      regression_gan_imputation_model)
13 from PatientsData import Patients
14 from RecoverData import Recover
15
16 warnings.filterwarnings('ignore')
17
18 # Checking if the system has been called with the required number of arguments
19 # The last argument is optional while the others are mandatory
20 if len(sys.argv) < 3:
21     print(f'Bad arguments. python {sys.argv[0]} <zeros/mean> <knn/nn/each/gan> output.arff <
       chained_cycles>')
22     exit()
23
24 # Defining the range of missin values of an attribute to be recovered with advanced
25 # methodologies
26 MIN_LOST_VALUES = 200
27 MAX_LOST_VALUES = 350
28
29 # Reading the preprocessed dataset with missing values
30 data_patients = Patients('../datos/
       CA_COLORRECTAL_COMPLICACIONES_atributos_borrados_binary.arff')
31
32 # Getting the indices of those attributes which are going to be recovered
33 # with advanced methodologies
34 interesting_missing_columns = data_patients.getmissingcolumnsrange(MIN_LOST_VALUES,
       MAX_LOST_VALUES)
35
36 # Getting the indices of those attributes with missing values
37 missing_columns = data_patients.getmissingattributes()
38
```

```
39  # Getting the indices of the attributes to be recovered with easy methodologies
40  easy_replacement_values = np.setxor1d(interesting_missing_columns, missing_columns)
41
42
43  # Defining the max number of cycles of MICE
44  max_chained_cycles = 1
45
46
47  if len(sys.argv) == 5:
48      max_chained_cycles = int(sys.argv[4])
49
50  # Initializing the object which is in charge of recovering the missing values
51  recover_data = Recover(data_patients, interesting_missing_columns, max_chained_cycles)
52
53  # If the methodology selected for the uninteresting attributes
54  # is not implemented, the script is terminated
55  if sys.argv[1] not in ['zeros', 'mean']:
56      print('That technique to recover useless data is not implemented')
57      exit()
58
59  # Recovering the missing values of uninteresting attributes
60  for attribute in easy_replacement_values:
61      if sys.argv[1] == 'zeros':
62          # numerical attributes missing values imputed with -1
63          if recover_data.is_regression(attribute):
64              for i in range(data_patients.data.shape[0]):
65                  if np.isnan(data_patients.data[i, attribute]):
66                      data_patients.data[i, attribute] = -1
67          else:
68              # categorical attributes missing values recovered with 0
69              for j in range(data_patients.data.shape[0]):
70                  if np.isnan(data_patients.data[j, attribute]):
71                      data_patients.data[j, attribute] = 0
72
73      if sys.argv[1] == 'mean':
74          # numerical attributes missing values imputed with the mean of the attribute
75          if recover_data.is_regression(attribute):
76              column = data_patients.data[:, attribute]
77              column[np.isnan(column)] = np.mean(column[np.isnan(column) == False])
78
79          else:
```

```
80                  # categorical attributes missing values recovered with the
81                  # mode of the attribute
82                  column = data_patients.data[:, attribute]
83                  column[np.isnan(column)] = mode(
84                      column[np.isnan(column) == False])[0]
85
86              data_patients.data[:, attribute] = column
87
88      # updating the data of the recover object
89      recover_data.setData(data_patients)
90
91      # If the methodology selected for the interesting attributes
92      # is not implemented, the script is terminated
93      if sys.argv[2] not in ['nn', 'knn', 'each', 'gan']:
94          print(f'The technique {sys.argv[2]} is not implemented to recover data')
95          exit()
96
97
98      cycles = 0 # variable in charge of storing the actual cycle
99      # lenght of the array with the indices of interesting attributes
100     LENGTH_INTERESTING_COLUMNS = len(interesting_missing_columns)
101
102     while cycles < recover_data.max_chained_cycles:
103         i = 0 # counter for iterating interesting_missing_columns array
104
105         # If MICE is selected, in other words `max_chained_cycles > 1`
106         # the script informs about the actual cycle
107         if recover_data.is_chained:
108             print(f'Cycle {cycles+1}')
109
110         # iterating the interesting attributes
111         while i < LENGTH_INTERESTING_COLUMNS:
112             # getting the index of the attribute
113             index_attribute = interesting_missing_columns[i]
114             # getting the name of the attribute
115             name_attribute = data_patients.getattributenames(index_attribute)
116
117             # if the attribute is a numerical attribute
118             if recover_data.is_regression(index_attribute):
119                 # The system outputs the attribute it is recovering
120                 print(f'Recovering column {name_attribute}')
```

```
121
122             # set the index attribute for preparing the data for training
123             recover_data.setindex(index_attribute, cycles)
124
125             # selecting the model
126             if sys.argv[2] == 'knn':
127                 model = regressionModelkNN()
128
129             if sys.argv[2] == 'nn':
130                 model = regressionModelNN()
131
132             if sys.argv[2] == 'each':
133                 model = bestModelAttribute()
134
135             if sys.argv[2] == 'gan':
136                 model = regression_gan_imputation_model()
137
138             # Training model and imputing missing values
139             is_numerical = True
140             recover_data.fit_and_replace(model, is_numerical, name_attribute, cycles)
141
142         else:
143             # defining the list which have the different values
144             # that the categorical attribute can take
145             nominal_values = list()
146
147             # List that stores the index of the attributes
148             # which define the original categorical attribute
149             list_attributes = list()
150
151             # adding the index of the actual attribute to the list
152             list_attributes.append(index_attribute)
153
154             # taking one of the values that the categorical attribute can take
155             first_value_categorical = re.search(r'=(.*)', name_attribute).group(1)
156
157             # Taking the name of the categorical attribute
158             name_attribute = re.search(r'(.*)=', name_attribute).group(1)
159
160             # adding the value to the list of values
161             nominal_values.append(first_value_categorical)
```

```
162
163             # The system outputs the attribute it is recovering
164             print(f'Recovering columns {name_attribute}')
165
166             # Now we iterate from the next attribute just to group
167             # the attributes whichj conform the original categorical attribute
168             for j in range(i + 1, LENGTH_INTERESTING_COLUMNS):
169                 # getting the index of the next attribute
170                 index_next_attribute = interesting_missing_columns[j]
171
172                 # getting the name of the next attribute
173                 next_attribute = data_patients.getattributenames(index_next_attribute)
174
175                 # If the name of the attribute has not got
176                 # an =, it is a numerical attribute
177                 # and we break the loop of searching attributes
178                 # from the same original categorical attribute
179                 if re.match(".*=", next_attribute) is None:
180                     break
181
182                 # we take the name of its original categorical attribute
183                 name_next_attribute = re.search(r'(.*)=', next_attribute).group(1)
184
185                 # if the name of the categorical attribute is different to the
186                 # one we are working on, we break the loop
187                 if not recover_data.is_regression(index_next_attribute) \
188                         and name_attribute == name_next_attribute:
189
190                     # we add the index to the list of attributes
191                     list_attributes.append(index_next_attribute)
192
193                     # get the value that the original categorical attribute can take
194                     value_next_attribute = re.search(r'=(.*)', next_attribute).group(1)
195
196                     # adding that value to the list of values
197                     nominal_values.append(value_next_attribute)
198
199                     # incrementing the counter which iterates the
200                     # attributes to avoid repetition
201                     i += 1
202                 else:
```

```
203                         # break the loop if it is not the same categorical attribute
204                         break
205
206                 # setting the list of indexes of attributes to prepare the data
207                 recover_data.setindex(np.array(list_attributes), cycles)
208
209                 # selecting the model
210                 if sys.argv[2] == 'knn':
211                     model1 = classificationModelkNN()
212                 if sys.argv[2] == 'nn':
213                     model1 = classificationModelNN(outputs=len(list_attributes))
214                 if sys.argv[2] == 'each':
215                     model1 = bestModelAttribute(outputs=len(list_attributes))
216                 if sys.argv[2] == 'gan':
217                     model1 = classification_gan_imputation_model(
218                         outputs=len(list_attributes)
219                     )
220
221                 # Training model and imputing missing values
222                 is_numerical = False
223                 recover_data.fit_and_replace(model1, is_numerical, name_attribute, cycles,
        nominal_values)
224
225             # incrementing the counter to check the following attribute
226             i += 1
227
228         # incrementing the number of cycles
229         cycles += 1
230
231 # opening the file to write the imputed data
232 f = open(sys.argv[3], 'w')
233
234 if recover_data.is_chained:
235     data_patients.arff['data'] = recover_data.patients.data
236 else:
237     data_patients.arff['data'] = recover_data.copypatients.data
238
239 arff.dump(data_patients.arff, f)
240 f.close()
```

Listing 19.1: main.py file

19.2 models.py

This file implements the several models which are used in the rest of the files. In order to make generic code, an abstract class has been defined.

```
1  import warnings
2  from abc import ABC, abstractmethod
3
4  from keras import Sequential, Model
5  from keras.callbacks import EarlyStopping
6  from keras.layers import Dense, Dropout, Input
7  from numpy import argmin, array, concatenate, ones, zeros
8  from numpy.random import default_rng
9  from scipy.spatial import distance_matrix
10 from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
11
12 warnings.filterwarnings('ignore')
13
14 # defining an abstract class for all the models
15 class baseModel(ABC):
16     @abstractmethod
17     def fit(self, x_train, y_train):
18         pass
19
20     @abstractmethod
21     def predict(self, x_test):
22         pass
23
24 # model which search the best network architecture for each attribute
25 class bestModelAttribute(baseModel):
26     def __init__(self, outputs=1):
27         self.best_model = None
28         self.best_loss = None
29         self.outputs = outputs
30         self.regression = (outputs == 1)
31
32     def fit(self, x_train, y_train):
33         callback = list()
34
35         # defining the callbacks for the neural network
36         callback.append(EarlyStopping(
```

```
37                 patience=10,
38                 monitor='val_loss',
39                 mode='min',
40                 min_delta=0.01,
41                 restore_best_weights=True,
42                 verbose=0
43             ))
44
45
46             for first_layer in [700, 600, 500, 400, 300, 200, 100, 50, 20, 10]:
47                 model = Sequential()
48                 count = 1
49                 next_layer = int(first_layer / 2)
50
51                 if self.regression:
52                     model.add(Dense(first_layer, kernel_initializer='normal', activation='relu'))
53                 else:
54                     model.add(Dense(first_layer, activation='relu'))
55
56                 while next_layer > self.outputs:
57                     if self.regression:
58                         model.add(Dense(next_layer, kernel_initializer='normal', activation='relu'))
59                     else:
60                         model.add(Dense(next_layer, activation='relu'))
61
62                     count += 1
63                     if count == 2 and next_layer > 10:
64                         model.add(Dropout(0.05))
65                         count = 0
66
67                     next_layer = int(next_layer / 2)
68
69                 if self.regression:
70                     model.add(Dense(self.outputs, kernel_initializer='normal'))
71                     model.compile(optimizer='adam', loss='mean_squared_error')
72                 else:
73                     model.add(Dense(self.outputs, activation='softmax'))
74                     model.compile(
75                         optimizer='adam',
76                         loss='categorical_crossentropy',
77                         metrics=['accuracy']
```

```
78                      )
79
80                  # training the model
81                  model.fit(
82                      x_train,
83                      y_train,
84                      batch_size=128,
85                      epochs=50,
86                      validation_split=0.15,
87                      callbacks=callback,
88                      verbose=0
89                  )
90
91                  if self.regression:
92                      loss = model.evaluate(x_train, y_train, verbose=0)
93                  else:
94                      loss = model.evaluate(x_train, y_train, verbose=0)[0]
95
96                  if self.best_loss == None:
97                      self.best_loss = loss
98                      self.best_model = model
99                  elif loss < self.best_loss:
100                     self.best_loss = loss
101                     self.best_model = model
102
103         # predicting the values with the best model
104         def predict(self, x_test):
105             return self.best_model.predict(x_test)
106
107 # model which implements a neural network for a numerical attribute
108 class regressionModelNN(baseModel):
109         def __init__(self):
110             self.model = Sequential()
111
112             self.model.add(Dense(24, kernel_initializer='normal', activation='relu'))
113             self.model.add(Dropout(0.05))
114             self.model.add(Dense(12, kernel_initializer='normal', activation='relu'))
115             self.model.add(Dense(1, kernel_initializer='normal'))
116
117             self.model.compile(loss='mean_squared_error', optimizer='adam')
118
```

```
119         def fit(self, x_train, y_train):
120             callback = list()
121             callback.append(EarlyStopping(
122                 patience=10,
123                 monitor='val_loss',
124                 mode='min',
125                 min_delta=0.01,
126                 restore_best_weights=True,
127                 verbose=0
128             ))
129 
130             self.model.fit(
131                 x_train,
132                 y_train,
133                 batch_size=128,
134                 epochs=50,
135                 validation_split=0.15,
136                 callbacks=callback,
137                 verbose=0
138             )
139 
140         def predict(self, x_test):
141             return self.model.predict(x_test)
142 
143     # model which implements a neural network for a categorical attribute
144     class classificationModelNN(baseModel):
145         def __init__(self, outputs):
146             self.outputs = outputs
147             self.model = Sequential()
148             self.model.add(Dense(24, activation='relu'))
149             self.model.add(Dropout(0.05))
150             self.model.add(Dense(12, activation='relu'))
151             self.model.add(Dense(outputs, activation='softmax'))
152 
153             self.model.compile(
154                 optimizer='adam',
155                 loss='categorical_crossentropy',
156                 metrics=['accuracy']
157             )
158 
159         def fit(self, x_train, y_train):
```

```
160             callback = list()
161             callback.append(EarlyStopping(
162                 patience=10,
163                 monitor='val_loss',
164                 mode='min',
165                 min_delta=0.01,
166                 restore_best_weights=True,
167                 verbose=0
168             ))
169
170             self.model.fit(
171                 x_train,
172                 y_train,
173                 batch_size=128,
174                 epochs=50,
175                 validation_split=0.15,
176                 callbacks=callback,
177                 verbose=0
178             )
179
180         def predict(self, x_test):
181             return self.model.predict(x_test)
182
183     # model which implements a kNN model for a numerical attribute
184     class regressionModelkNN(baseModel):
185         def __init__(self, weights='distance'):
186             self.weights = weights
187             self.model = KNeighborsRegressor(weights=weights)
188
189         def fit(self, x_train, y_train):
190             self.model.fit(x_train, y_train)
191
192         def predict(self, x_test):
193             return self.model.predict(x_test)
194
195     # model which implements a kNN model for a categorical attribute
196     class classificationModelkNN(baseModel):
197         def __init__(self, weights='distance'):
198             self.weights = weights
199             self.model = KNeighborsClassifier(weights=weights)
200
```

```
201         def fit(self, x_train, y_train):
202             self.model.fit(x_train, y_train)
203 
204         def predict(self, x_test):
205             return self.model.predict(x_test)
206 
207 
208     class classification_gan_imputation_model(baseModel):
209         def __init__(self, outputs):
210             self.outputs = outputs
211 
212         def fit(self, x_train, y_train):
213             self.discriminator = self.create_discriminator()
214             self.generator = self.create_generator(x_train.shape[1])
215             self.gan = self.create_gan(x_train.shape[1])
216 
217             epochs = 5
218             batch_size = 128
219 
220             number_generator = default_rng()
221 
222             for i in range(epochs):
223                 for j in range(batch_size):
224                     # generate random noise as an input to initialize the generator
225                     noise = number_generator.random((batch_size, x_train.shape[1]))
226 
227                     # Generate fake instances from noised input
228                     fake_instances = self.generator.predict(noise)
229 
230                     # Get a random set of real instances
231                     random_samples = number_generator.integers(low=2, size=batch_size)
232                     x_real_instances = x_train[random_samples]
233                     y_real_instances = y_train[random_samples]
234 
235 
236                     # Construct different batches of real and fake instances
237                     X = concatenate([x_real_instances, fake_instances])
238 
239                     # Labels for generated and real data
240                     y_fake_instances = zeros((batch_size, self.outputs))
241                     Y = concatenate([y_real_instances, y_fake_instances])
```

```
242
243                 # Pre train discriminator on fake and real data  before  starting  the gan.
244                 self.discriminator.trainable = True
245                 self.discriminator.train_on_batch(X, Y)
246
247                 # Tricking the noised input of the Generator as real data
248                 noise = number_generator.random((batch_size, x_train.shape[1]))
249                 Y_generated = zeros((batch_size, self.outputs))
250
251                 for index_row in range(Y_generated.shape[0]):
252                     Y_generated[index_row][number_generator.integers(low=self.outputs)] = 1
253
254                 # During the training of gan,
255                 # the weights of discriminator should be fixed.
256                 # We can enforce that by setting the trainable flag
257                 self.discriminator.trainable = False
258
259                 # training  the GAN by alternating the training of the Discriminator
260                 # and training the chained GAN model with Discriminator's weights freezed.
261                 self.gan.train_on_batch(noise, Y_generated)
262
263     def predict(self, x_test):
264         number_missing_values = x_test.shape[0]
265         number_attributes = x_test.shape[1]
266         number_generator = default_rng()
267
268         noise = number_generator.random(
269             (number_missing_values, number_attributes)
270         )
271
272         fake_instances = self.generator.predict(noise)
273
274         distances = distance_matrix(x_test, fake_instances)
275
276         recovered_values = list()
277
278         for index in range(number_missing_values):
279             index_min = argmin(distances[index, :])
280
281             recovered_value = self.discriminator.predict(
282                 fake_instances[index_min].reshape(1, number_attributes)
```

```
283                 )
284
285                 recovered_values.append(recovered_value[0])
286
287             return array(recovered_values)
288
289
290         def create_discriminator(self):
291             model = Sequential()
292             model.add(Dense(24, activation='relu'))
293             model.add(Dropout(0.05))
294             model.add(Dense(12, activation='relu'))
295             model.add(Dense(self.outputs, activation='softmax'))
296
297             model.compile(
298                 optimizer='adam',
299                 loss='categorical_crossentropy'
300             )
301
302             return model
303
304         def create_generator(self, number_inputs):
305             model = Sequential()
306             model.add(Dense(12, activation='relu'))
307             model.add(Dropout(0.05))
308             model.add(Dense(24, activation='relu'))
309             model.add(Dropout(0.05))
310             model.add(Dense(number_inputs, activation='tanh'))
311
312             model.compile(loss='mean_squared_error', optimizer='adam')
313
314             return model
315
316         def create_gan(self, number_inputs):
317             self.discriminator.trainable = False
318             gan_input = Input(shape=(number_inputs,))
319             x = self.generator(gan_input)
320             gan_output = self.discriminator(x)
321
322             gan = Model(inputs=gan_input, outputs=gan_output)
323             gan.compile(loss='categorical_crossentropy', optimizer='adam')
```

```
324
325             return gan
326
327     class regression_gan_imputation_model(baseModel):
328         def __init__(self, outputs=1):
329             self.outputs = outputs
330
331         def fit(self, x_train, y_train):
332             self.discriminator = self.create_discriminator()
333             self.generator = self.create_generator(x_train.shape[1])
334             self.gan = self.create_gan(x_train.shape[1])
335
336             epochs = 5
337             batch_size = 128
338
339             number_generator = default_rng()
340
341             for i in range(epochs):
342                 for j in range(batch_size):
343                     # generate random noise as an input to initialize the generator
344                     noise = number_generator.random((batch_size, x_train.shape[1]))
345
346                     # Generate fake instances from noised input
347                     fake_instances = self.generator.predict(noise)
348
349                     # Get a random set of real instances
350                     random_samples = number_generator.integers(low=2, size=batch_size)
351                     x_real_instances = x_train[random_samples]
352                     y_real_instances = y_train[random_samples]
353
354
355                     # Construct different batches of real and fake instances
356                     X = concatenate([x_real_instances, fake_instances])
357
358                     # Labels for generated and real data
359                     y_fake_instances = ones((batch_size, self.outputs))*-1
360                     Y = concatenate([y_real_instances, y_fake_instances[:, 0]])
361
362                     # Pre train discriminator on fake and real data before starting the gan.
363                     self.discriminator.trainable = True
364                     self.discriminator.train_on_batch(X, Y)
```

```
365
366                 # Tricking the noised input of the Generator as real data
367                 noise = number_generator.random((batch_size, x_train.shape[1]))
368                 Y_generated = number_generator.random((batch_size, self.outputs))
369
370                 # During the training of gan,
371                 # the weights of discriminator should be fixed.
372                 # We can enforce that by setting the trainable flag
373                 self.discriminator.trainable = False
374
375                 # training the GAN by alternating the training of the Discriminator
376                 # and training the chained GAN model with Discriminator's weights freezed.
377                 self.gan.train_on_batch(noise, Y_generated)
378
379     def predict(self, x_test):
380         number_missing_values = x_test.shape[0]
381         number_attributes = x_test.shape[1]
382         number_generator = default_rng()
383
384         noise = number_generator.random(
385             (number_missing_values, number_attributes)
386         )
387
388         fake_instances = self.generator.predict(noise)
389
390         distances = distance_matrix(x_test, fake_instances)
391
392         recovered_values = list()
393
394         for index in range(number_missing_values):
395             index_min = argmin(distances[index, :])
396
397             recovered_value = self.discriminator.predict(
398                 fake_instances[index_min].reshape(1, number_attributes)
399             )
400
401             recovered_values.append(recovered_value[0])
402
403         return array(recovered_values)
404
405
```

```
406     def create_discriminator(self):
407         model = Sequential()
408         model.add(Dense(24, kernel_initializer='normal', activation='relu'))
409         model.add(Dropout(0.05))
410         model.add(Dense(12, kernel_initializer='normal', activation='relu'))
411         model.add(Dense(self.outputs, kernel_initializer ='normal'))
412 
413         model.compile(
414             optimizer='adam',
415             loss='mean_squared_error'
416         )
417 
418         return model
419 
420     def create_generator(self, number_inputs):
421         model = Sequential()
422         model.add(Dense(12, kernel_initializer='normal', activation='relu'))
423         model.add(Dropout(0.05))
424         model.add(Dense(24, kernel_initializer='normal', activation='relu'))
425         model.add(Dropout(0.05))
426         model.add(Dense(number_inputs, activation='tanh'))
427 
428         model.compile(loss='mean_squared_error', optimizer='adam')
429 
430         return model
431 
432     def create_gan(self, number_inputs):
433         self.discriminator.trainable = False
434         gan_input = Input(shape=(number_inputs,))
435         x = self.generator(gan_input)
436         gan_output = self.discriminator(x)
437 
438         gan = Model(inputs=gan_input, outputs=gan_output)
439         gan.compile(loss='mean_squared_error', optimizer='adam')
440 
441         return gan
```

Listing 19.2: models.py file

19.3 PatientsData.py

In this file, the Patient class is defined. It represents the data of the patients contained in the dataset. Basically it provides two important methods called *getmissingattributes* and *getmissingcolumnsrange*, which returns the indexes of the attributes with missing values and the indexes of those attributes which have a number of missing values defined in a range respectively. Furthemore, it has a private function which is in charge of reading the dataset, storing the names of the attributes in an attribute class as well as changing their classification labels by numerical values.

```
1  import re
2  import warnings
3
4  import arff
5  import numpy as np
6
7  warnings.filterwarnings('ignore')
8
9
10 class Patients:
11     # defining the constructor of Patients Class
12     def __init__(self, pathname):
13         # setting the pathname of the file
14         self.setpathname(pathname)
15
16         if self.pathname is None:
17             raise Exception(
18                 "You should introduce a arff file to process the data")
19         else:
20             # Call __processdata function
21             self.__processdata()
22
23     # it returns the pathname of the file
24     def getpathname(self):
25         return self.pathname
26
27     # it sets the pathname of the file
28     def setpathname(self, pathname):
29         self.pathname = pathname
30
```

```
31      # it returns the name of attribute with index 'index'
32      def getattributenames(self, index):
33          return str(self.attributesnames[index][0])
34
35      # it returns the arff object
36      def get_arff(self):
37          return self.arff
38
39      # it returns an array with the indices of attributes with missing data
40      def getmissingattributes(self):
41          indexes = list()
42          for i in range(0, self.data.shape[1]):
43              for j in range(0, self.data.shape[0]):
44                  if np.isnan(self.data[j][i]):
45                      indexes.append(i)
46                      break
47
48          return np.array(indexes)
49
50      # It returns an attribute with the index 'index'
51      def getcolumn(self, index):
52          if index in range(0, self.data.shape[1]):
53              return self.data[:, index]
54          else:
55              raise Exception("Index out of bounds")
56
57      # it returns an array with the indices of attributes which
58      # have a number of missing values between lower and upper
59      def getmissingcolumnsrange(self, lower, upper):
60          indexes = list()
61          count = 0
62
63          for i in range(0, self.data.shape[1]):
64              for j in range(0, self.data.shape[0]):
65                  if np.isnan(self.data[j][i]):
66                      count += 1
67
68              if count in range(lower, upper + 1) and re.search(r'AYUDANTE|CIRUJANO', self.
    getattributenames(i)) == None:
69                  indexes.append(i)
70
```

```
71                 count = 0
72
73             return np.array(indexes)
74
75         # It return the information of a patient
76         def getpatient(self, index):
77             if index in range(0, self.data.shape[0]):
78                 return self.data[index]
79             else:
80                 raise Exception("Index out of bounds")
81
82         # It reads the files and initializes the attributes
83         # of the class
84         def __processdata(self):
85             try:
86                 self.file = open(self.pathname, 'r')
87             except IOError:
88                 raise Exception('The file does not exist')
89             else:
90                 # loads the file
91                 self.arff = arff.load(self.file)
92
93                 # get the list of attribute names
94                 self.attributesnames = np.array(self.arff['attributes'])
95                 # gets the data of the dataset
96                 self.data = np.array(self.arff['data'])
97
98                 # change the label class to a numerical value
99                 for i in range(0, self.data.shape[0]):
100                    if self.data[i, -1] == 'Si':
101                        self.data[i, -1] = 1
102                    elif self.data[i, -1] == 'No':
103                        self.data[i, -1] = 0
104
105                # converts the data to a float type
106                self.data = self.data.astype(np.float64)
107                # update the information on the arff project
108                self.arff['data'] = self.data
```

Listing 19.3: PatientsData.py file

19.4 RecoverData.py

In this file, the Recover class is defined. It represents the recovery of missing values. It is in charge of re-estructuring the data in order to separate between columns and records which have missing values or not and it is the one who calls the computational models to be trained and to predict values. Another task it has is to prepare data for the *DecodeService* class, which decode the numerical and categorical attributes.

```
1  import warnings
2
3  import numpy as np
4
5  from models import baseModel
6  from services import DecodeService
7  from PatientsData import Patients
8
9  warnings.filterwarnings('ignore')
10
11
12 class Recover:
13     # defining the constructor of Recover Class
14     def __init__(self, patients: Patients, missing_attributes, max_chained_cycles):
15         # We save the data of patients and a copy
16         self.patients = patients
17         self.copypatients = patients
18
19         # initialise index and outputs attribute
20         self.index = -1
21         self.outputs = 1
22
23         # Saving list of interesting attributes
24         self.missing_attributes = missing_attributes
25
26         # Storing the value of the max number of cycles for MICE
27         self.max_chained_cycles = max_chained_cycles
28
29         # If max_chained_cycles is 1, mice is not applied
30         self.is_chained = max_chained_cycles != 1
31
32         # setting the seed for random numbers
```

```
33          np.random.seed(42)
34
35      # It sets the data of patients, and saves two copies
36      def setData(self, patients: Patients):
37          self.patients = patients
38          self.copypatients = patients
39          self.copydata = np.copy(patients.data)
40
41      # It returns the index or indexes of the attribute or attributes
42      # it is recovering
43      def getindex(self):
44          return self.index
45
46      # set the index or indexed if it is numerical or categorical
47      def setindex(self, index, cycles):
48          # if it is a numerical value
49          if self.is_regression(index) and not isinstance(index, np.ndarray):
50              self.index = index
51              # It only recovers one value
52              self.outputs = 1
53              is_numerical = True
54
55              # preparing the data for the models
56              self.__preparedata(is_numerical, cycles)
57          else:
58              # if it is a categorical attribute
59              self.listindex = index
60              # It recovers several values
61              self.outputs = len(self.listindex)
62              is_numerical = False
63
64              # preparing the data for the models
65              self.__preparedata(is_numerical, cycles)
66
67      # returns True if the attribute with index 'index ' is numerical
68      def is_regression(self, index):
69          column = self.patients.data[:, index]
70
71          result = np.unique(column[~np.isnan(column)])
72
73          if len(result) > 2:
```

```
74                  return True
75              else:
76                  return False
77
78      def __preparedata(self, flag, cycles):
79          # we get a copy of the actual data
80          self.data = np.copy(self.patients.data)
81
82          # It searchs for those elements which are None. This function
83          # returns the position of those elements.
84          # We are interested in getting what is the new index of the attribute that
85          # we want to recover
86          # But we are also interested in getting the position of those patients with that missing data
87          # This function return that
88
89          # we get the values of the attribute or attributes
90          # we want to recover
91          if flag:
92              self.labels = self.data[:, self.index]
93          else:
94              self.labels = self.data[:, self.listindex]
95
96
97          if not self.is_chained:
98              # if we do not apply mice, we need to erase
99              # the attributes with missing values, including the
100             # one we want because we have stored it in self.labels
101             self.data = np.delete(self.data, self.missing_attributes, 1)
102         else:
103             # if we are applying mice
104             # if it is the first cycle we are running
105             if cycles == 0:
106                 # we fill the missing values of those attributes
107                 # different to the attribute we want to recover
108                 # with -1
109                 if flag:
110                     for a in self.missing_attributes:
111                         if a != self.index:
112                             for b in range(self.data.shape[0]):
113                                 if np.isnan(self.data[b, a]):
114                                     self.data[b, a] = -1
```

```
115                     else:
116                         for a in self.missing_attributes:
117                             if a not in self.listindex:
118                                 for b in range(self.data.shape[0]):
119                                     if np.isnan(self.data[b, a]):
120                                         self.data[b, a] = -1
121                 else:
122                     # in succesive cycles of MICE
123                     # we get the positions of the missing values
124                     # of the attribute or attributes(saved in self.labels)
125                     # and we fill it with np.nan (mark as missing)
126                     positions_missing = self.get_position_missing(self.copydata, flag)
127                     if flag:
128                         self.labels[positions_missing] = np.nan
129                     else:
130                         self.labels[positions_missing, :] = np.nan
131
132
133             # now we get the positions of the missing values
134             # this process is repeated when mice is applied and cycles
135             # is greater than 0. But this operation is important when
136             # cycles is zero or mice is not applied
137             self.positions = np.argwhere(np.isnan(self.labels))
138             self.positions = np.unique(self.positions[:, 0])
139
140
141             # in case of appyling mice, we delete the attributes
142             # we want to recover from self.data
143             if self.is_chained:
144                 if flag:
145                     self.data = np.delete(self.data, self.index, 1)
146                 else:
147                     self.data = np.delete(self.data, self.listindex, 1)
148
149             self.train = []
150             self.test = []
151
152             # now we divide the data depending on the number
153             # of missing values have each record
154             # if it has not got any missing values, it goes
155             # to training, test in the other case
```

```
156             if flag:
157                 for i in range(0, self.labels.shape[0]):
158                     if np.isnan(self.labels[i]):
159                         self.test.append(self.data[i])
160                     else:
161                         self.train.append(self.data[i])
162             else:
163                 for i in range(0, self.labels.shape[0]):
164                     if np.argwhere(np.isnan(self.labels[i])).size > 0:
165                         self.test.append(self.data[i])
166                     else:
167                         self.train.append(self.data[i])
168 
169             self.train = np.array(self.train)
170             self.test = np.array(self.test)
171 
172         # It fits the model, get the predicted values and calls __replace function
173         def fit_and_replace(self, model: baseModel, flagRegression, name_attribute, cycles,
             nominal_values=None):
174             x_train = self.train
175             x_test = self.test
176 
177             y_train = []
178 
179             # now we get the labels of the instances which go
180             # to training
181             for z in range(self.labels.shape[0]):
182                 if z not in self.positions:
183                     y_train.append(self.labels[z])
184 
185             y_train = np.array(y_train)
186 
187             # if it is categorical, we convert that data
188             # to int
189             if flagRegression is False:
190                 y_train = y_train.astype(int)
191 
192             # fitting the model with training data
193             model.fit(x_train, y_train)
194 
195             # predicting the labels for x_text
```

```
196             predicted = model.predict(x_test)
197
198             # replaces the values in the file prepared for R and for experiments
199             self.__replace(predicted, name_attribute, flagRegression, nominal_values, cycles)
200
201         def __replace(self, predicted, name_attribute, flagRegression, nominal_values, cycles):
202             # if we are replacing a numerical value
203             if nominal_values == None:
204                 k = 0
205                 if self.is_chained:
206                     # we update the data of patients
207                     for i in self.positions:
208                         self.patients.data[i, self.index] = predicted[k]
209                         k += 1
210
211                     # if it is the last cycle, we save the information
212                     # for the file prepared for R
213                     if self.max_chained_cycles - cycles - 1 == 0:
214                         DecodeService.decode(
215                             self.patients.data[:, self.index],
216                             name_attribute,
217                             flagRegression
218                         )
219                 else:
220                     # we update the data of copypatients
221                     for i in self.positions:
222                         self.copypatients.data[i, self.index] = predicted[k]
223                         k += 1
224
225                     # saves the information for R file
226                     DecodeService.decode(
227                         self.copypatients.data[:, self.index],
228                         name_attribute,
229                         flagRegression
230                     )
231
232             else:
233                 # if it is a categorical attribute
234                 k = 0
235                 if self.is_chained:
236                     for i in self.positions:
```

```
237                     # we put to 1 the max value of the predicted
238                     # a zero for the rest
239                     c = predicted[k]
240                     c[np.argmax(c)] = 1
241                     c[c != 1] = 0
242 
243                     # we update the data of patients
244                     self.patients.data[i, np.min(self.listindex):(np.max(self.listindex) + 1)] = c
245 
246                     k += 1
247 
248                 # if it is the last cycle, we save the information
249                 # for the file prepared for R
250                 if self.max_chained_cycles - cycles - 1 == 0:
251                     DecodeService.decode(
252                         self.patients.data[:, np.min(self.listindex):(np.max(self.listindex) + 1)],
253                         name_attribute,
254                         flagRegression,
255                         nominal_values
256                     )
257 
258             else:
259                 for i in self.positions:
260                     # we put to 1 the max value of the predicted
261                     # a zero for the rest
262                     c = predicted[k]
263                     c[np.argmax(c)] = 1
264                     c[c != 1] = 0
265 
266                     # we update the data of copypatients
267                     self.copypatients.data[i, np.min(self.listindex):(
268                         np.max(self.listindex) + 1)] = c
269 
270                     k += 1
271 
272                 # saves the information for R file
273                 DecodeService.decode(
274                     self.copypatients.data[:, np.min(self.listindex):(np.max(self.listindex) + 1)],
275                     name_attribute,
276                     flagRegression,
277                     nominal_values
```

```
278                     )
279
280             # if mice is not applied
281             # we udpate the arff object
282             if not self.is_chained:
283                 self.copypatients.arff['data'] = self.copypatients.data
284
285
286         # get the positions of the missing vaklues in an attribute or attributes
287         def get_position_missing(self, data, flag):
288             if flag:
289                 labels = data[:, self.index]
290             else:
291                 labels = data[:, self.listindex]
292
293             positions = np.argwhere(np.isnan(labels))
294             return np.unique(positions[:, 0])
```

Listing 19.4: RecoverData.py file

19.5 services.py

This file is in charge of converting one hot encoding representations to the real values of the categorical attributes and scaling numerical values to their original scale.

```
1   from os.path import exists
2
3   import arff
4   import numpy as np
5   from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
6
7
8   class DecodeService:
9       @staticmethod
10      def decode(predicted, name_attribute, regression, nominal_values=None):
11          dictionary = None
12
13          # If the output file exists, we read it to update it
14          if exists('outputForR.arff'):
```

```
15                dictionary = arff.load(open('outputForR.arff', 'r'))
16            else:
17                # if it doesn't, we create a copy and we update it
18                dictionary = arff.load(open('../datos/
          CA_COLORRECTAL_COMPLICACIONES_atributos_borrados.arff', 'r'))
19
20            # getting the data of the file
21            datos = np.array(dictionary['data'])
22
23            # getting the attributes names of the datasets
24            names = np.array(dictionary['attributes'])
25            names = names[:, 0]
26
27            # getting the index of the attribute with the name 'name_attribute'
28            index_attribute = np.where(names == name_attribute)[0][0]
29
30            # getting the data from that attribute
31            column = datos[:, index_attribute]
32
33            # erasing the missing values of the column
34            column = column[column != None]
35
36            # if is is categorical
37            if not regression:
38                # store the length of list of values
39                nominal_values_length = len(nominal_values)
40
41                # represent labels as 0 1 2 3 ...
42                labels = range(nominal_values_length)
43
44                lb = OneHotEncoder(handle_unknown='ignore')
45
46                # training one hot encoder
47                lb.fit(np.array(labels).reshape(-1, 1))
48
49                # decode the predicted data with 0 1 2 3 ...
50                inverse = lb.inverse_transform(predicted)
51                inverse = inverse.astype(object)
52
53                # setting the original values of the categorical attribute
54                for j in range(nominal_values_length):
```

```
55                  inverse[inverse == labels[j]] = nominal_values[j]
56
57              else:
58                  # if it is numerical attribute
59                  column = column.reshape(-1, 1)
60                  scaler = MinMaxScaler()
61
62                  # fit a scaler object with the data of the column
63                  scaler.fit(column)
64
65                  predicted = predicted.reshape(-1, 1)
66                  # doing the inverse operation to re-scale the data to its original scale
67                  inverse = scaler.inverse_transform(predicted)
68
69              # saving the decoded data in thr dataset
70              datos[:, index_attribute] = inverse.reshape(datos[:, index_attribute].shape[0])
71
72              # updating the data into the file
73              dictionary['data'] = datos
74              f = open('outputForR.arff', 'w')
75              arff.dump(dictionary, f)
```

Listing 19.5: services.py file

19.6 crossValidation.py

This file runs the experiments to measure the quality of the recovery of missing values.

```
1   import sys
2
3   import arff
4   import numpy as np
5   from keras.callbacks import EarlyStopping
6   from keras.layers import Dense, Dropout
7   from keras.models import Sequential
8   from keras.wrappers.scikit_learn import KerasClassifier
9   from sklearn.model_selection import StratifiedKFold, cross_val_score
10
11  # creating the model
```

```
12  def create_model():
13      model = Sequential()
14      model.add(Dense(24, activation='relu'))
15      model.add(Dropout(0.05))
16      model.add(Dense(12, activation='relu'))
17      model.add(Dense(1, activation='sigmoid'))
18      model.compile(optimizer='adam', loss='binary_crossentropy',
19                    metrics=['accuracy'])
20      return model
21
22  # checking if the program has the right amount of arguments
23  if len(sys.argv) != 2:
24      print(f'Bad arguments: python {sys.argv[0]} input.arff')
25      exit(-1)
26
27  # setting the seed of random numbers
28  np.random.seed(42)
29
30  # loading the dataset from the file
31  data = arff.load(open(sys.argv[1]))
32
33  # setting the callbacks for the neural network model
34  callbacks = list()
35  callbacks.append(EarlyStopping(patience=20, verbose=0,
36                                 monitor='val_loss', mode='min',
37                                 min_delta=0.0001, restore_best_weights=True))
38
39  # converting the data to an array
40  array = np.array(data['data'])
41
42  # structuring the data for dividing instance and label
43  x_train_label = np.array(array[:, -1]).reshape(-1, 1)
44  x_train = np.array(array[:, 0:-1])
45
46  X = x_train.astype(np.float32)
47  Y = x_train_label.astype(int)
48
49  # creating a keras Classifier wrapper for scikit learn functions
50  model = KerasClassifier(
51      build_fn=create_model,
52      epochs=80,
```

```
53          batch_size=64,
54          verbose=0,
55          validation_split=0.1
56      )
57
58      # creating a kfold object
59      kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
60
61      # running a cross validation with 10 folds
62      results = cross_val_score(model, X, Y, cv=kfold, fit_params={
63                                  'callbacks': callbacks}, n_jobs=-1)
64
65      # printing the results
66      print(results)
67      print(results.mean())
68      print(results.std())
```

Listing 19.6: crossValidation.py file

Glossary

ARFF Attribute-Relation File Format. 24, 25, 39, 43, 47, 49, 50, 64, 96, 97, 107

artificial neural network Computation algorithms inspired by the biological neural networks. 4, 11

big data Big amount of information which is very complex to be processed by data processing applications. 4

CCR Correct Classification Rate. x, 73–76, 78

CPU Central Processing Unit. 4

CSV Comma-separated Values file. 36, 39, 41, 44, 78, 96

decision tree decision support tool that uses a tree-like model of decisions and their possible consequences. 16

deep neural network Artificial neural network with many layers. 3, 5, 13, 17, 18, 20, 77, 105

GAN Generative Adversarial Networks. 18, 60, 61, 67, 76, 82, 105

gradient boosting machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models. 16

IDE Integrated Development Environment. 32, 91

kNN K-nearest neighbours. 17, 20, 66, 76, 78, 82, 98, 105

lightGBM gradient boosting framework that uses tree based learning algorithms. 16

logistic regression classification algorithm used to assign observations to a discrete set of classes. 15

MICE Multiple Imputation by Chained Equations. 19, 57, 66, 67, 76, 78, 82, 98, 99, 105

ML Machine Learning. 4, 5, 17

python An interpreted, high-level, general-purpose programming language. 32, 63, 77, 89, 91, 93, 94, 96, 97, 105

quintillion Equal to 10^{18}. 4

R Programming language for statistical computing and graphics. 32, 89, 91, 92, 95, 99, 100

ROC Receiver operating characteristic. 16

SARS-CoV-2 Severe Acute Respiratory Syndrome Coronavirus 2. 5

TXT Text file. 39, 45

weka A workbench for machine learning. 32, 63, 64, 69, 89, 91, 92, 95, 96

wrapper Software that calls another program. 63

XRFF XML attribute Relation File Format. 96

Bibliography

[1] "Pdq colon cancer treatment." [Online]. Available: https://www.cancer.gov/types/colorectal/patient/colon-treatment-pdq

[2] R. I., "El cáncer de colon aumenta en los jóvenes europeos," *Diario ABC*, 2019. [Online]. Available: https://www.abc.es/salud/enfermedades/abci-cancer-colon-aumenta-jovenes-europeos-201905170030_noticia.html

[3] A. Garg, "Complete guide to association rules," 2018. [Online]. Available: https://towardsdatascience.com/association-rules-2-aa9a77241654

[4] B. J. Feder, "Prepping robots to perform surgery," *The New York Times*, 2008. [Online]. Available: https://www.nytimes.com/2008/05/04/business/04moll.html

[5] "Life expectancy around the world has increased steadily for nearly 200 years." [Online]. Available: https://www.nature.com/scitable/content/life-expectancy-around-the-world-has-increased-19786/

[6] "A history of machine learning." [Online]. Available: https://cloud.withgoogle.com/build/data-analytics/explore-history-machine-learning/

[7] B. Marr, "How much data do we create every day? the mind-blowing stats everyone should read," *Forbes*, 2018.

[8] A. C. Lagandula, "Mcculloch-pitts neuron — mankind's first mathematical model of a biological neuron," 2018. [Online]. Available: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

[9] Wikipedia contributors, "Artificial neural network — Wikipedia, the free encyclopedia," 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=916535747

[10] ——, "Deep learning — Wikipedia, the free encyclopedia," 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=917568476

[11] P. L. Junfeng Gao, Yong Yang and D. S. Park, "Computer vision in healthcare applications," *Journal of Healthcare Engineering*, vol. 2018, 2018. [Online]. Available: https://doi.org/10.1155/2018/5157020

[12] "Coronavirus disease (covid-19) pandemic," [Accessed 2-April-2020]. [Online]. Available: https://www.who.int/emergencies/diseases/novel-coronavirus-2019

[13] "Bsc uses bioinformatics, artificial intelligence and the computing power of the marenostrum supercomputer in the fight against the coronavirus," [Accessed 2-April-2020]. [Online]. Available: https://bit.ly/39AWJz8

[14] Y. Xu, L. Ju, J. Tong, C.-M. Zhou, and J.-J. Yang, "Machine learning algorithms for predicting the recurrence of stage iv colorectal cancer after tumor resection," *Scientific Reports*, vol. 10, no. 1, 2020. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85079334138&doi=10.1038%2fs41598-020-59115-y&partnerID=40&md5=76539a81254dd19394dcc0ac06150a45

[15] J. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0035470889&partnerID=40&md5=ad9996316bf9ad0a202e905fee48c810

[16] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," vol. 2017-December, 2017, pp. 3147–3155. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85038215940&partnerID=40&md5=e7f6611aa423276ab95bd73aa45ad93e

[17] M. Hornbrook, R. Goshen, E. Choman, M. O'Keeffe-Rosetti, Y. Kinar, E. Liles, and K. Rust, "Early colorectal cancer detected by machine learning model using gender, age, and complete blood count data," *Digestive Diseases and Sciences*, vol. 62, no. 10, pp. 2719–2727, 2017. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85027979337&doi=10.1007%2fs10620-017-4722-8&partnerID=40&md5=19e516f5f4e23e7e44dc30aa309ab99d

[18] E. M. Beale and R. J. Little, "Missing values in multivariate analysis," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 37, no. 1, pp. 129–145, 1975.

[19] J. Chen and J. Shao, "Nearest neighbor imputation for survey data," *Journal of Official Statistics*, vol. 16, pp. 113–131, 01 2000.

[20] S. Zhang, "Nearest neighbor selection for iteratively knn imputation," *Journal of Systems and Software*, vol. 85, no. 11, pp. 2541 – 2552, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121212001586

[21] O. Temam, "The rebirth of neural networks," *SIGARCH Comput. Archit. News*, vol. 38, no. 3, p. 349, Jun. 2010. [Online]. Available: https://doi.org/10.1145/1816038.1816008

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," vol. 3, no. January, 2014, pp. 2672–2680. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84937849144&partnerID=40&md5=6441b2a288c5fdded2adbcc8b21e092c

[23] J. H. Moor, *Turing Test*. GBR: John Wiley and Sons Ltd., 2003, p. 1801–1802.

[24] M. Azur, E. Stuart, C. Frangakis, and P. Leaf, "Multiple imputation by chained equations: What is it and how does it work?" *International Journal of Methods in Psychiatric Research*, vol. 20, no. 1, pp. 40–49, 2011. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-79951982954&doi=10.1002%2fmpr.329&partnerID=40&md5=c8660bc5a3858208048d894a92e8fe4a

[25] B. Sierra, I. Inza, and P. Larrañaga, "On applying supervised classification techniques in medicine," in *Medical Data Analysis*, J. Crespo, V. Maojo, and F. Martin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 14–19.

[26] W. W. Cohen, "Fast effective rule induction," in *Twelfth International Conference on Machine Learning*. Morgan Kaufmann, 1995, pp. 115–123.

[27] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, California: Wadsworth International Group, 1984.

[28] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Morgan Kaufmann, 1994, pp. 487–499.

[29] J. Delgado-Osuna, C. García-Martínez, S. Ventura, and J. Gómez Barbadillo, "Obtaining tractable and interpretable descriptions for cases with complications from a colorectal cancer database," in *2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS)*, June 2019, pp. 459–464.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[31] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[32] T. E. Oliphant, *A guide to NumPy*. Trelgol Publishing USA, 2006, vol. 1.

[33] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. . . Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[34] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[35] I. Witten, E. Frank, M. Hall, and C. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 2016. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85009962818&partnerID=40&md5=9d48eb8ad0e28bd4ad13ef43f6c5d085

[36] "Why one-hot encode data in machine learning?" [Online]. Available: https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/

[37] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," *CoRR*, vol. abs/1611.01491, 2016. [Online]. Available: http://arxiv.org/abs/1611.01491

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[39] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[40] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *From Natural to Artificial Neural Computation*, J. Mira and F. Sandoval, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 195–201.

[41] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945. [Online]. Available: http://www.jstor.org/stable/3001968

[42] A. M. Sefidian and N. Daneshpour, "Estimating missing data using novel correlation maximization based methods," *Applied Soft Computing*, vol. 91, p. 106249, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494620301897

[43] D. J. Stekhoven and P. Bühlmann, "MissForest—non-parametric missing value imputation for mixed-type data," *Bioinformatics*, vol. 28, no. 1, pp. 112–118, 10 2011. [Online]. Available: https://doi.org/10.1093/bioinformatics/btr597