STUDENT:

Irene García Medina

(Erasmus+)

REGISTRATION NUMBER:

838543

UNIVERSITY EMAIL:
i.garciamedina@studenti.uniba.it

REPOSITORY:
https://github.com/Irenegm22/ICON_progetto

UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

CASE STUDY

DETECTOR OF PATHOLOGIES GIVEN A BLOOD ANALYTICS

INDEX:

CHAPTER 0

Introduction 3

Pathologies 3

Project Structure 3

CHAPTER 1. Dataset Creation 4

CHAPTER 2. Prolog Rules. Knowledge Database 5

2.1. Query example 7

CHAPTER 3. Pre – Processing 7

3.1. Outliners 8

3.2. Utils 8

CHAPTER 4. Bayes 9

CHAPTER 5. Random Forest 10

5.1. Accuracy vs Hyperparameters 10

5.2. Confusion Matrix 12

CHAPTER 6. Supervised Training. Supervised Learning 13

6.1. Models Used 13

CHAPTER 7. Main 15

CHAPTER 8. Analysis of the Results 15

8.1. Analysis of Each Graph 16

8.1.1. Without SMOTE 16

8.1.2. With SMOTE 18

INTRODUCTION:

The following project has been carried out for the Knowledge Engineering course, taught by Fanizzi Nicola in the 24/25 academic year of the University of Bari.

The main objective of this project is to demonstrate all the knowledge acquired during the development of said course. In this case, it has been applied to the detection of possible pathologies through a blood test, such as various types of cancer, immunodeficiency or anemia.

PATHOLOGIES:

As indicated, the project focuses on 4 diverse diseases:

Leukemia and lymphoma: are due to a high or very high number of B lymphocytes and T lymphocytes, respectively.

Anemia: Produced by a low number of red blood cells

Immunodeficiency: Which occurs if there is a low number of white blood cells (lymphocytes of both types)

Finally, the option of the possibility of suffering from cancer has been added if there are slightly high values, without being dangerous. This factor is increased if there is a history of cancer in the patient's family.

PROJECT STRUCTURE:

The project is divided into the following parts:

Creation of the initial dataset: A python code which generates a '.csv' dataset of 2000 patients with the following data: Age, Gender, Lymphocytes T, Lymphocytes B, Red Cells, Background, Diagnosis, the latter "-" since The diagnosis has not yet been made.

Creation of prolog rules: These rules determine what type of pathology each patient has. This '.pl' code is generated by another python code.

Pre-processing: Its purpose is to increase the quality of the data.

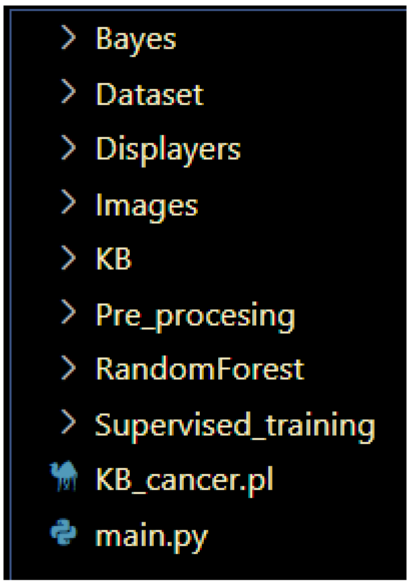
Bayes: This is a python code whose function is to generate a Bayesian network, which is saved as an image.

Random forest: Created in python, its objective is to generate two images: accuracy vs hyperparameters and confusion matrix.

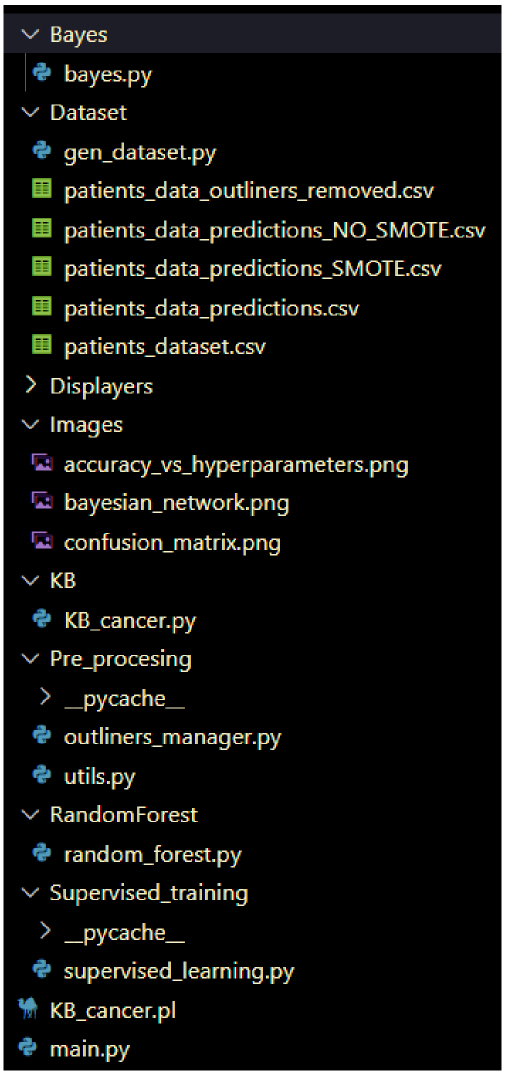
Supervised training: This is a python class that is responsible for initializing the various models, training them with cross-validation and evaluating them.

Finally, a main class has been created to carry out all these tests.

Below is an image of the organization of the structure:



```
> Bayes
> Dataset
> Displayers
> Images
> KB
> Pre_procesing
> RandomForest
> Supervised_training
KB_cancer.pl
main.py
```



```
v Bayes
    bayes.py
v Dataset
    gen_dataset.py
    patients_data_outliners_removed.csv
    patients_data_predictions_NO_SMOTE.csv
    patients_data_predictions_SMOTE.csv
    patients_data_predictions.csv
    patients_dataset.csv
> Displayers
v Images
    accuracy_vs_hyperparameters.png
    bayesian_network.png
    confusion_matrix.png
v KB
    KB_cancer.py
v Pre_procesing
  > __pycache__
    outliners_manager.py
    utils.py
v RandomForest
    random_forest.py
v Supervised_training
  > __pycache__
    supervised_learning.py
KB_cancer.pl
main.py
```

1. DATASET CREATION:

To generate the dataset, a python code called gen_dataset.py has been created, whose path is './Dataset/gen_dataset.py'. This code generates 2000 random patients, who have the following characteristics:

Age: Represents the age of the patient. In this case, the domain has been limited to patients whose age is between 18 and 90 years old.

Gender: Indicates the gender of the patient. It is a parameter whose data type is integer, but it only has two possible values; 0 in case of female and 1 in case of male.

Lymphocytes T: It shows the number of T-type lymphocytes that the patient has. The domain is limited to the interval [400, 2000].

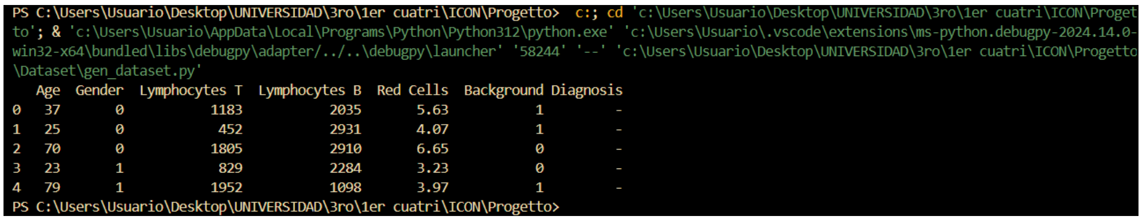
Lymphocytes B: It shows the number of B-type lymphocytes that the patient has. The domain is limited to the interval [480, 3300].

Red cells: Shows the number of red blood cells the patient has. Its domain is limited to [3.2, 8.0] million cells / mcL.

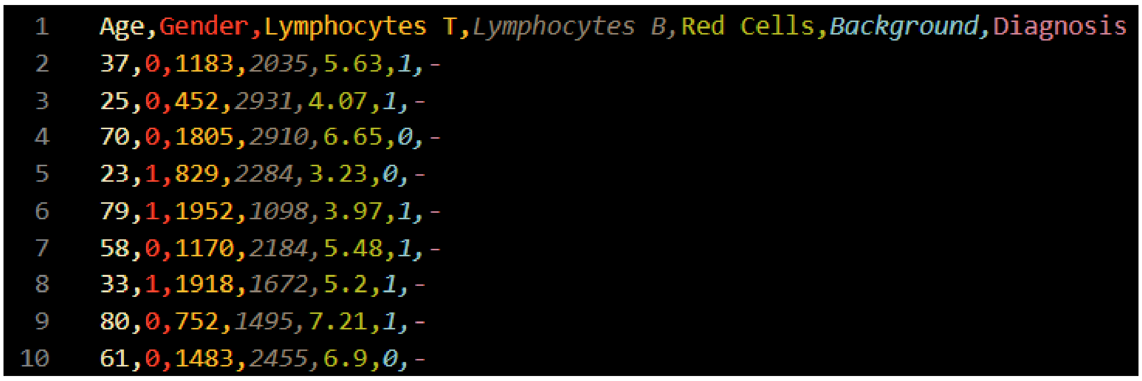
Background: Indicates whether the patient has a history of cancer in their family. If it has this, it is indicated with a 1, otherwise, the Background value is 0.

Diagnosis: When creating the default dataset, the diagnosis is initialized to '-' which indicates that the necessary tests have not yet been carried out to determine a possible diagnosis of the corresponding disease in each case.

Finally, this dataset is created in the same directory as the python code responsible for generating it.



```
PS C:\Users\Usuario\Desktop\UNIVERSIDAD\3ro\1er cuatri\ICON\Progetto>  c:; cd 'c:\Users\Usuario\Desktop\UNIVERSIDAD\3ro\1er cuatri\ICON\Proget
to'; & 'c:\Users\Usuario\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\Usuario\.vscode\extensions\ms-python.debugpy-2024.14.0-
win32-x64\bundled\libs\debugpy\adapter/../..\debugpy\launcher' '58244' '--' 'c:\Users\Usuario\Desktop\UNIVERSIDAD\3ro\1er cuatri\ICON\Progetto
\Dataset\gen_dataset.py'
   Age  Gender  Lymphocytes T  Lymphocytes B  Red Cells  Background Diagnosis
0   37       0           1183           2035       5.63           1         -
1   25       0            452           2931       4.07           1         -
2   70       0           1805           2910       6.65           0         -
3   23       1            829           2284       3.23           0         -
4   79       1           1952           1098       3.97           1         -
PS C:\Users\Usuario\Desktop\UNIVERSIDAD\3ro\1er cuatri\ICON\Progetto>
```

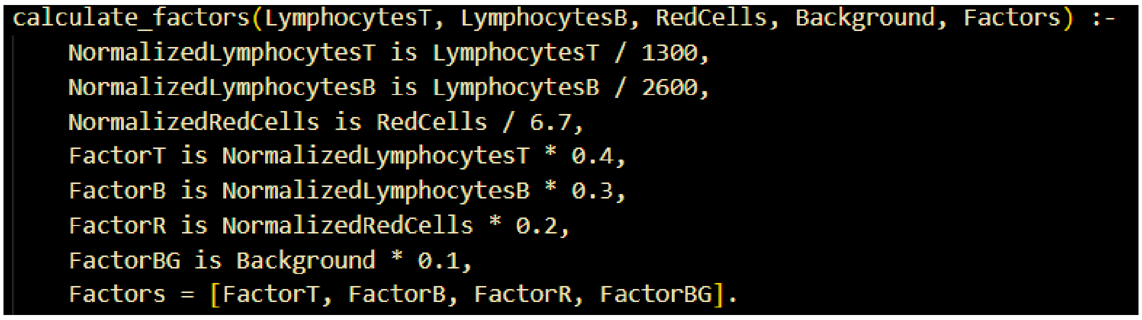


```
Age,Gender,Lymphocytes T,Lymphocytes B,Red Cells,Background,Diagnosis
37,0,1183,2035,5.63,1,-
25,0,452,2931,4.07,1,-
70,0,1805,2910,6.65,0,-
23,1,829,2284,3.23,0,-
79,1,1952,1098,3.97,1,-
58,0,1170,2184,5.48,1,-
33,1,1918,1672,5.2,1,-
80,0,752,1495,7.21,1,-
61,0,1483,2455,6.9,0,-
```

2. PROLOG RULES. CREATION OF A KNOWLEDGE BASE:

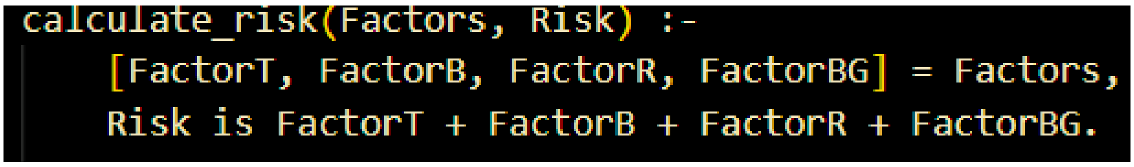
To create the prolog rules, a python class called KB_cancer.py has been created, whose path is the following: './KB/KB_cancer.py'. This code is responsible for generating a prolog file '.pl' in the current directory.

The prolog rules created are as follows:



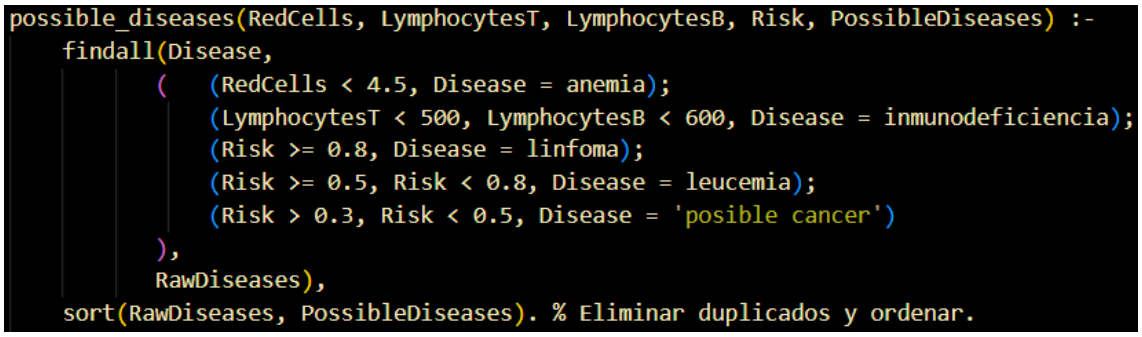
```
calculate_factors(LymphocytesT, LymphocytesB, RedCells, Background, Factors) :-
    NormalizedLymphocytesT is LymphocytesT / 1300,
    NormalizedLymphocytesB is LymphocytesB / 2600,
    NormalizedRedCells is RedCells / 6.7,
    FactorT is NormalizedLymphocytesT * 0.4,
    FactorB is NormalizedLymphocytesB * 0.3,
    FactorR is NormalizedRedCells * 0.2,
    FactorBG is Background * 0.1,
    Factors = [FactorT, FactorB, FactorR, FactorBG].
```

This function is responsible for normalizing the values of lymphocytes and red blood cells. Then, multiply this normalized value, in addition to the background history, by the percentage of risk importance it may have. These factors are stored in a vector.



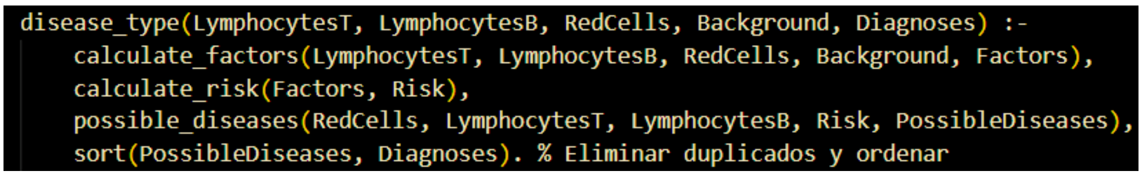
```
calculate_risk(Factors, Risk) :-
    [FactorT, FactorB, FactorR, FactorBG] = Factors,
    Risk is FactorT + FactorB + FactorR + FactorBG.
```

Function responsible for adding all the factors. Calculate the total risk of cancer.



```
possible_diseases(RedCells, LymphocytesT, LymphocytesB, Risk, PossibleDiseases) :-
    findall(Disease,
            (   (RedCells < 4.5, Disease = anemia);
                (LymphocytesT < 500, LymphocytesB < 600, Disease = inmunodeficiencia);
                (Risk >= 0.8, Disease = linfoma);
                (Risk >= 0.5, Risk < 0.8, Disease = leucemia);
                (Risk > 0.3, Risk < 0.5, Disease = 'posible cancer')
            ),
            RawDiseases),
    sort(RawDiseases, PossibleDiseases). % Eliminar duplicados y ordenar.
```

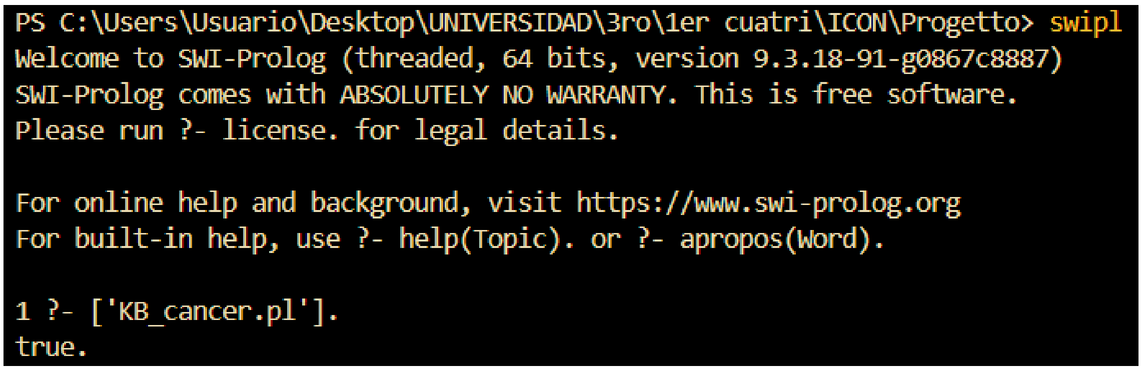
This function determines what type of disease/diseases you can suffer from according to the risk value calculated above. The last line of the function (sort) is responsible for ensuring that diagnoses are not duplicated for the same patient.



```
disease_type(LymphocytesT, LymphocytesB, RedCells, Background, Diagnoses) :-
    calculate_factors(LymphocytesT, LymphocytesB, RedCells, Background, Factors),
    calculate_risk(Factors, Risk),
    possible_diseases(RedCells, LymphocytesT, LymphocytesB, Risk, PossibleDiseases),
    sort(PossibleDiseases, Diagnoses). % Eliminar duplicados y ordenar
```

This is the query that is asked to prolog. It is then redirected to the 3 functions explained above.

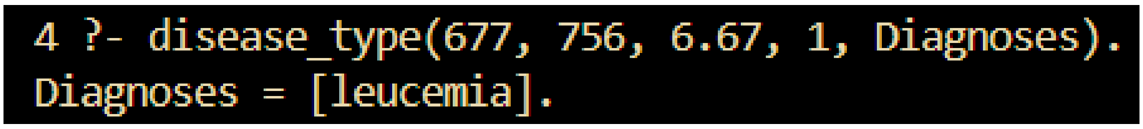
2.1. QUERY EXAMPLE:



```
PS C:\Users\Usuario\Desktop\UNIVERSIDAD\3ro\1er cuatri\ICON\Progetto> swipl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.3.18-91-g0867c8887)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

1 ?- ['KB_cancer.pl'].
true.
```



```
4 ?- disease_type(677, 756, 6.67, 1, Diagnoses).
Diagnoses = [leucemia].
```

In this way, we can consult the pathology of a specific patient. However, later we can obtain the total number of patients who have each type of disease (explained in the 'main.py' part of the code).

IMPORTANT!

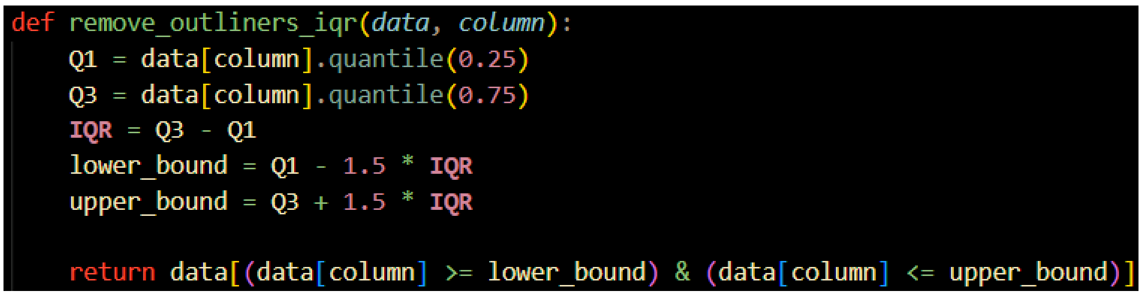
It is necessary to have the Prolog software previously installed and functional

3. PRE-PROCESSING:

The main purpose for which it is necessary to pre-process the data is to improve its quality. Through this process, data is cleaned in order to eliminate duplicate data, correct errors or complete missing data. In addition, the format or type of data of any of these can also be changed, so that we can facilitates its analysis. Finally, data can also be deleted, since it may not be relevant or, as mentioned above, it may be duplicated.

In the Pre-processing folder we find 2 different classes. They are the following:

3.1. OUTLINERS_MANAGER.PY:



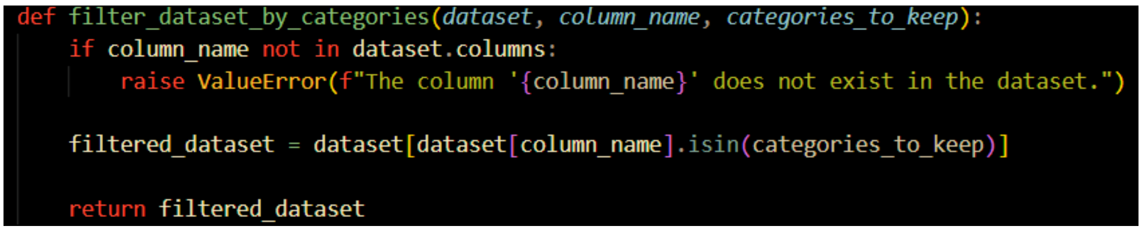
```python
def remove_outliners_iqr(data, column):
    Q1 = data[column].quantile(0.25)
    Q3 = data[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    return data[(data[column] >= lower_bound) & (data[column] <= upper_bound)]
```

This function is responsible for detecting and eliminating atypical values, if any occur. The method used for this is known as interqualitative range (IQR). Divide the data into four equal parts and calculate the range between the first and third quarters. Any value that is far above or far below this range is considered an outlier.

3.2. UTILS.PY:

It is divided into two functions, which are the following:

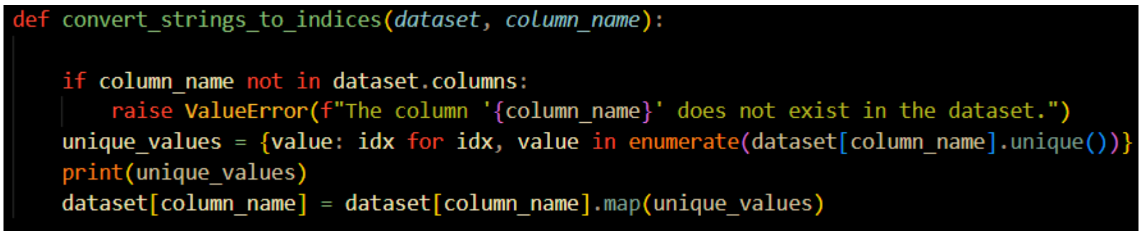


```python
def filter_dataset_by_categories(dataset, column_name, categories_to_keep):
    if column_name not in dataset.columns:
        raise ValueError(f"The column '{column_name}' does not exist in the dataset.")

    filtered_dataset = dataset[dataset[column_name].isin(categories_to_keep)]

    return filtered_dataset
```

As its name indicates, the function is responsible for filtering the dataset by categories. The objective is to create a filter that allows you to select specific information from a larger list.



```python
def convert_strings_to_indices(dataset, column_name):

    if column_name not in dataset.columns:
        raise ValueError(f"The column '{column_name}' does not exist in the dataset.")
    unique_values = {value: idx for idx, value in enumerate(dataset[column_name].unique())}
    print(unique_values)
    dataset[column_name] = dataset[column_name].map(unique_values)
```

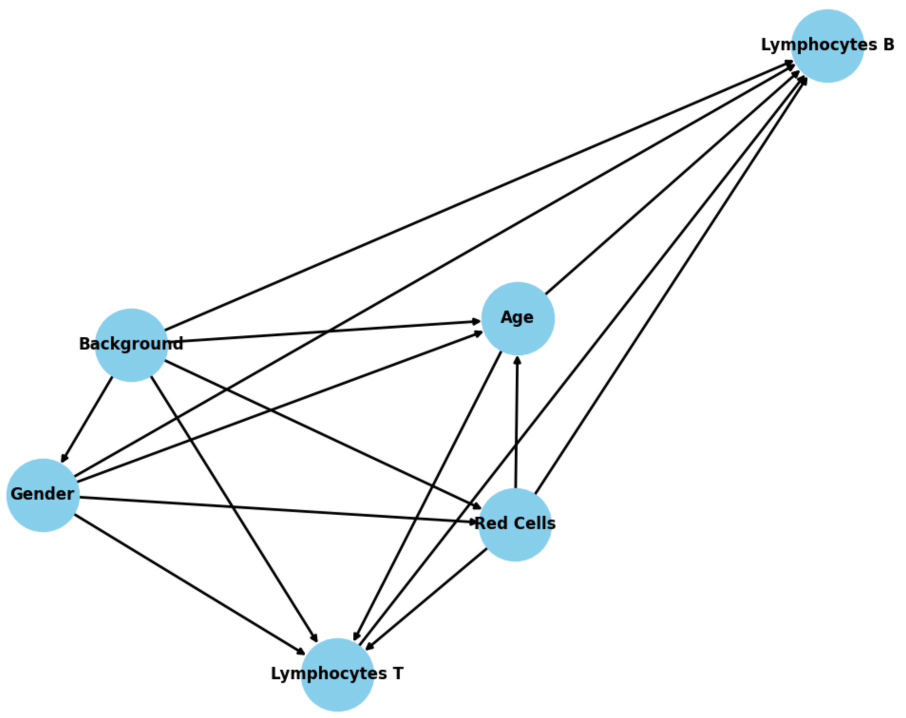
The objective of this function is to transform the categories from String type to integer type. Lists them uniquely and consecutively.

4. BAYES:

A Bayesian network is a system which makes it easier to understand how the various elements that make it up are related, in addition to how likely it is that one thing will happen if another has already happened. Its main objective is to help when making decisions, predicting what may happen or opening up the possibility of understanding more complex systems.

Therefore, it is useful when modeling uncertainty to know how sure we are about something; making decisions based on probabilities or solving complex systems.

In this case, a code called 'bayes.py' has been created which has the path: './Bayes/bayes.py' in order to generate the Bayesian network of the default dataset. The result obtained is stored in the 'Images' directory. Its name is 'bayesian_network.png' and it results:



The direction of the arrows indicates the influence of some parameters on others. For example, if an arrow points from A to B, it means that it influences the probability of B and, therefore, A is a cause that can affect B.

The direction of the arrow can also determine dependency. That is, if A goes to B and we know something about A, we can make a better prediction of B, if it is unknown.

5. RANDOM FOREST:

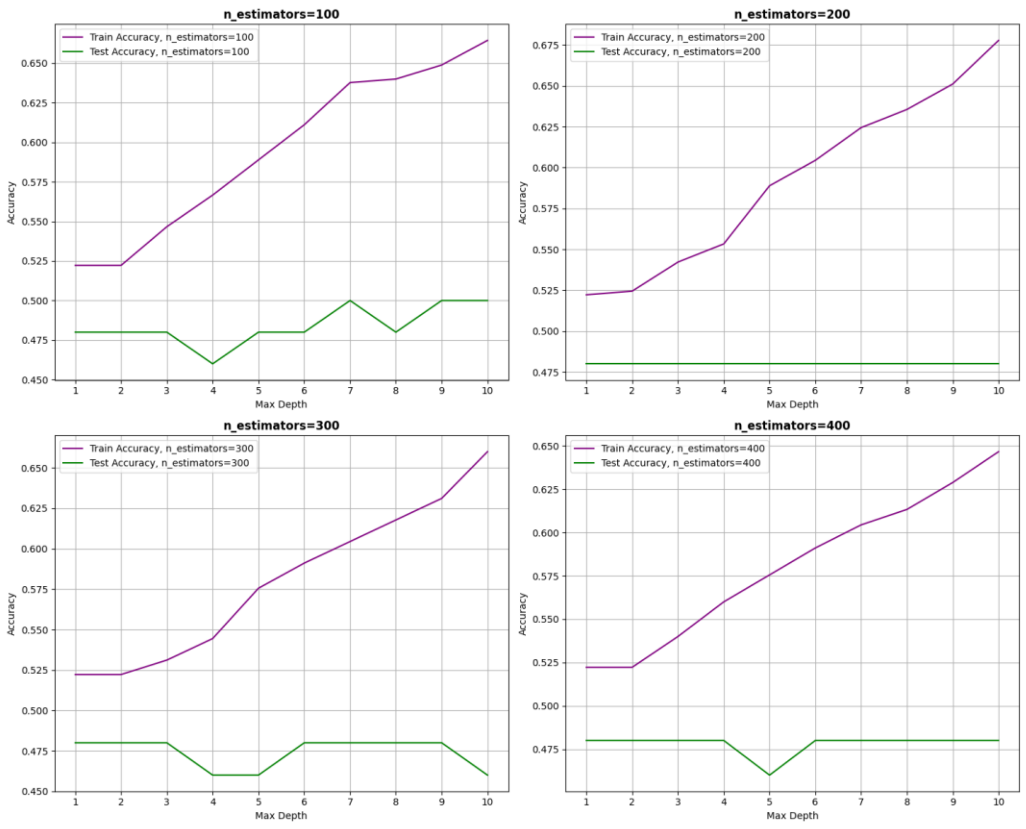
It is a method composed of many decision trees, where each one is constructed in a slightly different way from the previous one. Each one chooses a parameter and the one that has been chosen the most times is chosen. It is a widely used method that works well since diversity is taken into account and the average errors usually cancel each other.

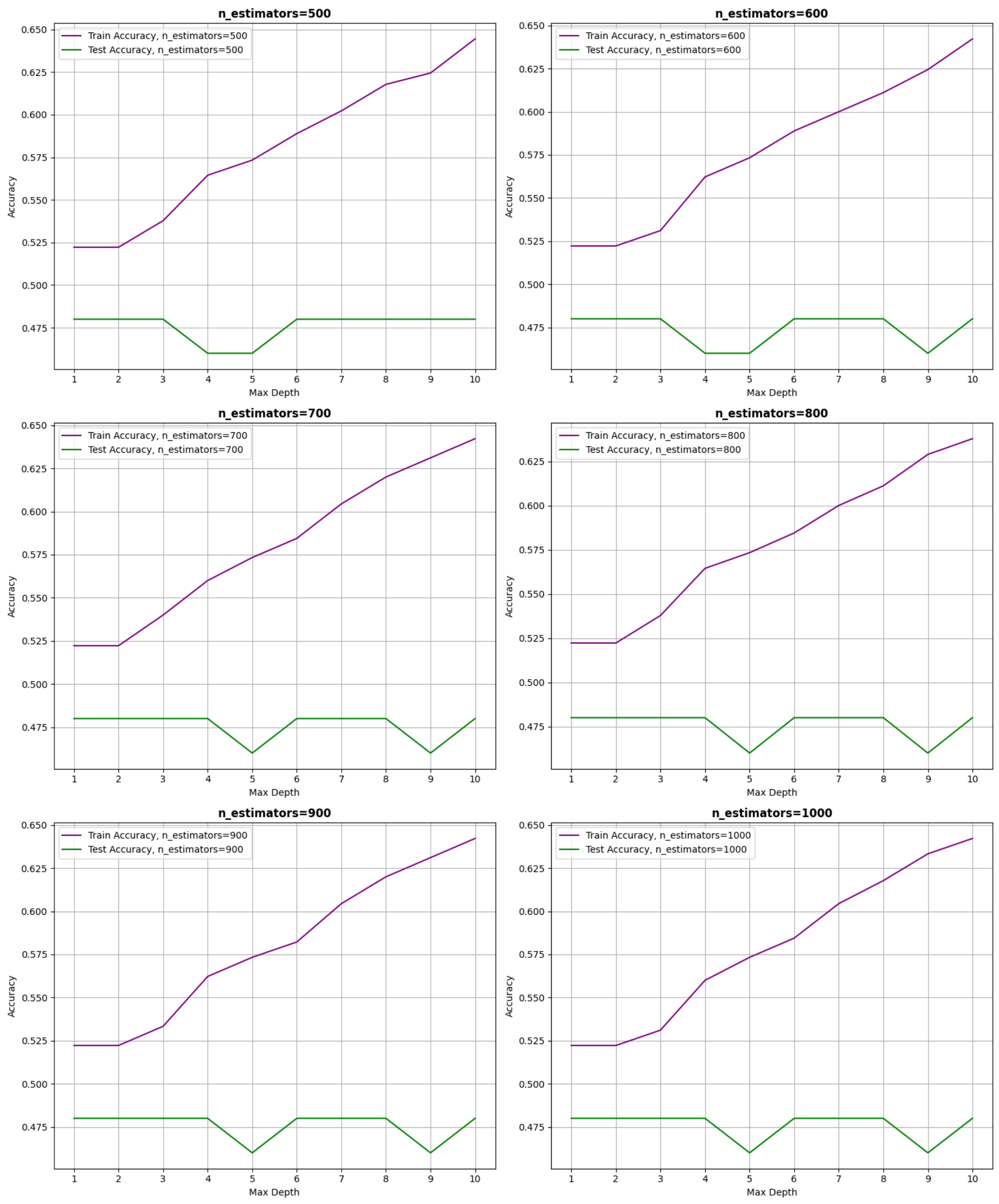
To do this, a file called 'random_forest.py' has been created and is located in the 'RandomForest' directory.

First, we load the default dataset, from which we will extract the data. If a missing value is found, it would be completed. Subsequently, a graph is created that represents the relationship between accuracy and hyperparameters. This graphic is stored in the 'Images' directory. Finally, a confusion matrix is generated, which is also stored in the previous directory.

The images created are the following:

5.1. ACCURACY VS HYPERPARAMETERS:





Accuracy vs hyperparameters graphs are tools that help understand how various settings affect the performance of a machine learning model.

Precision is defined as how well the model is able to predict. However, hyperparameters are defined as those factors that directly affect the system without being part of it.

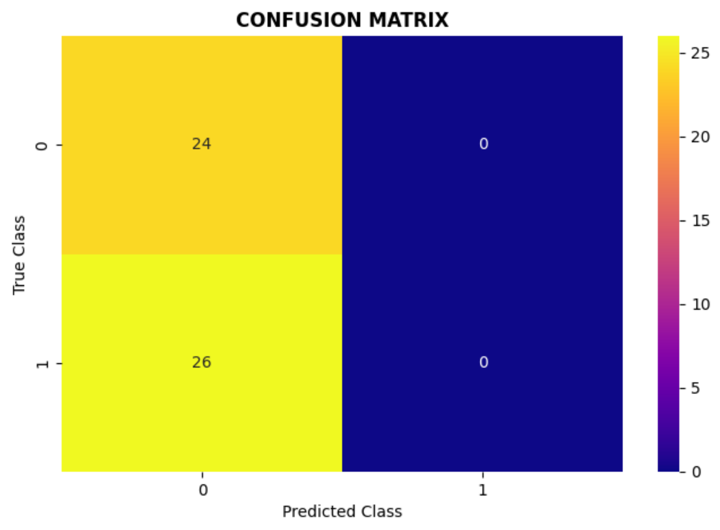
A graph of precision vs. Hyperparameters is a very useful tool to adjust the hyperparameters of a machine learning model and obtain the best possible performance.

All the graphs are very similar and the following stands out:

We see that in all cases the accuracy on the training set (purple line) increases as the maximum depth of the trees increases, while the accuracy on the test set (green line) reaches a maximum and then decreases. This is a clear sign of overfitting.

As we increase the number of estimators, the accuracy curve on the training set becomes smoother and the difference between the accuracy in training and testing becomes slightly smaller. This suggests that a larger number of estimators can help reduce overfitting.

5.2. CONFUSION MATRIX:



A confusion matrix is a table that shows us how a matching learning model is working.

The confusion matrix provides us with a detailed view of the performance of our model. It allows us to calculate metrics such as:

Precision: The proportion of positive predictions that were correct.

Recall: The proportion of positive cases that were correctly identified.

F1-score: A measure that combines precision and recall.

These metrics help us evaluate different aspects of the model's performance and identify areas where it can improve.

This matrix shows us how well a classification model has worked. In this case, it seems that the model is trying to classify elements into two categories: 0 and 1.

Main Diagon (values of 24 and 26): These numbers represent the correct predictions. That is, the model correctly predicted 24 elements as belonging to class 0 and 26 elements as belonging to class 1.

Off-diagonal (values of 0): These numbers represent incorrect predictions. The model did not classify any element of class 0 as 1, nor any element of class 1 as 0.

It indicates that the model has performed perfectly on this data set. You have classified all the elements correctly, without any errors. This is an excellent result and suggests that the model is a very good fit to the data.

6. SUPERVISED TRAINING. SUPERVISED LEARNING:

To carry out supervised training, a python class called 'supervised_learning.py' has been created, which is located in the 'Supervised_training' directory. This code is responsible for the following:

Data preparation: First, the data is prepared to be understood by the computer (organize it correctly and convert its format to be able to be processed).

Model creation: Next, the code generates several different models, which are responsible for guessing based on the provided characteristics.

Training the models: The code teaches the models to differences between the various diseases using the data provided.

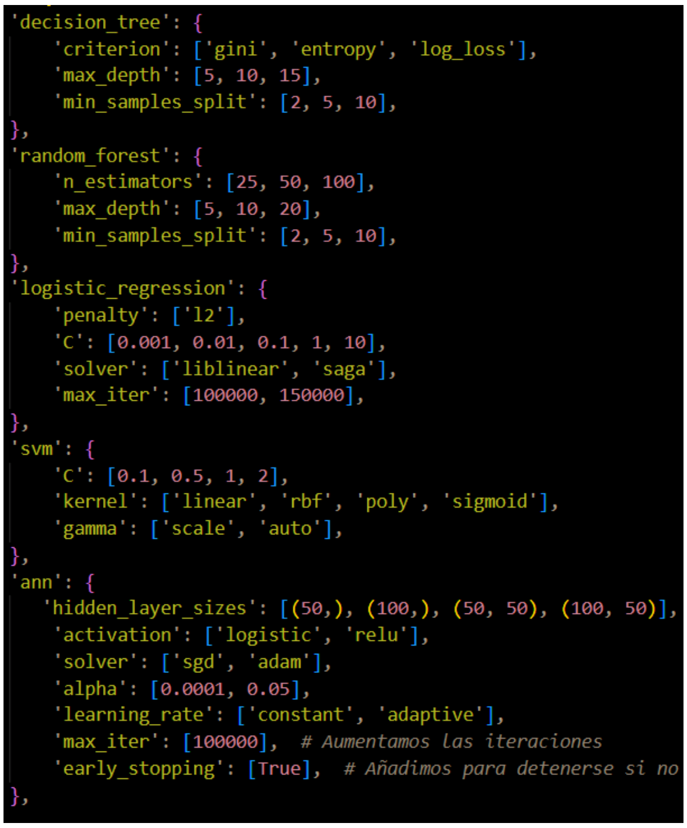
Model evaluation: Once the models are trained, the code tests them with new data that they have not seen before.

Comparison of the models: The code compares the different models to see which one is the best when it comes to distinguishing.

This type of code is widely used in areas related to medicine, since its use is very useful in diagnosing diseases, although it is also used in areas such as product recommendations or the detection of bank fraud.

6.1. MODELS USED:

The following models have been used:



```python
'decision_tree': {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
},
'random_forest': {
    'n_estimators': [25, 50, 100],
    'max_depth': [5, 10, 20],
    'min_samples_split': [2, 5, 10],
},
'logistic_regression': {
    'penalty': ['l2'],
    'C': [0.001, 0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'saga'],
    'max_iter': [100000, 150000],
},
'svm': {
    'C': [0.1, 0.5, 1, 2],
    'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
    'gamma': ['scale', 'auto'],
},
'ann': {
   'hidden_layer_sizes': [(50,), (100,), (50, 50), (100, 50)],
    'activation': ['logistic', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant', 'adaptive'],
    'max_iter': [100000],  # Aumentamos las iteraciones
    'early_stopping': [True],  # Añadimos para detenerse si no
},
```

<u>Diagram tree</u>: A decision tree is like a diagram that helps us organize information and make decisions logically, as well as to solve complex problems and predict future events (widely used in artificial intelligence).

<u>Random forest</u>: Random forest was explained previously.

<u>Logistic regression</u>: It is a mathematical tool that tries to guess whether something is going to happen or not, based on certain characteristics. It makes decisions based on data. It is used in the fields of medicine (predicting diseases), marketing (related to the purchase of products) or in finance (company bankruptcy).

<u>Support Vector Machine (SVM)</u>: An SVM seeks to find the best line (or plane, if you have more than two features) that separates the two classes of objects. This line is called a hyperplane. It try to find the hyperplane that is as far away as possible from the closest points of each class. These points are called support vectors. They are useful for classifying data.

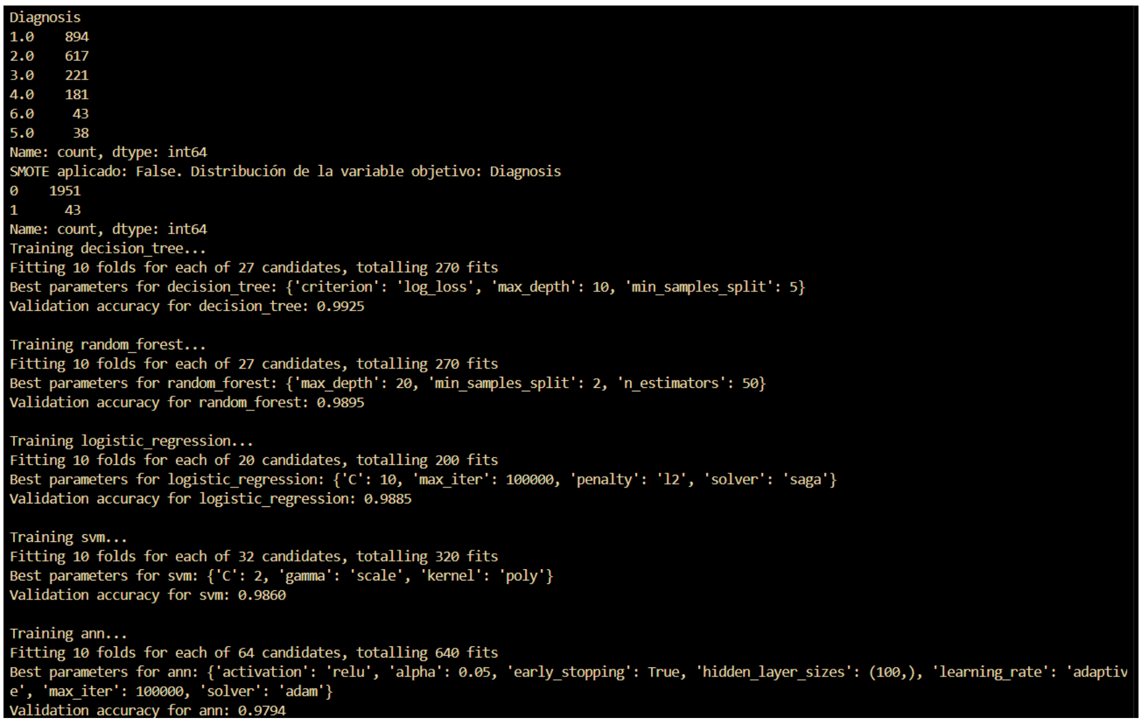
<u>Artificial Neural Network (ANN)</u>: Artificial neural networks are tools designed to learn and make decisions in a way similar to how humans do. An artificial neural network is made up of many interconnected processing units, called artificial neurons.

7. MAIN:

It is a python type class called 'main.py', which is located in the main directory. This class is responsible for connecting the dataset with the prolog rules and evaluating each model with the processed data.

8. ANALYSIS OF THE RESULTS:

As main is executed, the output that appears in the terminal is the following:



```
Diagnosis
1.0    894
2.0    617
3.0    221
4.0    181
6.0     43
5.0     38
Name: count, dtype: int64
SMOTE aplicado: False. Distribución de la variable objetivo: Diagnosis
0    1951
1      43
Name: count, dtype: int64
Training decision_tree...
Fitting 10 folds for each of 27 candidates, totalling 270 fits
Best parameters for decision_tree: {'criterion': 'log_loss', 'max_depth': 10, 'min_samples_split': 5}
Validation accuracy for decision_tree: 0.9925

Training random_forest...
Fitting 10 folds for each of 27 candidates, totalling 270 fits
Best parameters for random_forest: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 50}
Validation accuracy for random_forest: 0.9895

Training logistic_regression...
Fitting 10 folds for each of 20 candidates, totalling 200 fits
Best parameters for logistic_regression: {'C': 10, 'max_iter': 100000, 'penalty': 'l2', 'solver': 'saga'}
Validation accuracy for logistic_regression: 0.9885

Training svm...
Fitting 10 folds for each of 32 candidates, totalling 320 fits
Best parameters for svm: {'C': 2, 'gamma': 'scale', 'kernel': 'poly'}
Validation accuracy for svm: 0.9860

Training ann...
Fitting 10 folds for each of 64 candidates, totalling 640 fits
Best parameters for ann: {'activation': 'relu', 'alpha': 0.05, 'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptiv
e', 'max_iter': 100000, 'solver': 'adam'}
Validation accuracy for ann: 0.9794
```

In the main code we find the correspondence between the types of diseases and the corresponding integer value. Firstly, we find a distribution of the data according to what type of diseases the patients have, the first column being the number representative of each pathology; and the second column the number of patients who have that pathology.

Next, we find the statistics of sick and healthy people. In this case we see 1951 sick people and only 43 healthy. The data is very unbalanced.

For each model (decision tree, random forest, logistic regression, SVM and neural network), a search for the best hyperparameters is performed using cross validation.

Fitting: Indicates that the model is fitting to the training data.

Best parameters: Shows the best values found for the hyperparameters of each model. For example, for the decision tree, the best parameters are: "log_loss" split criterion, maximum depth of 10, and a minimum of 5 samples per split.

Validation accuracy: Shows the average accuracy of the model on the validation data. This value tells us how well the model generalizes to data it has not seen during training.

The models appear to perform quite well, with validation accuracy close to 99% in some cases. This suggests that the models are able to correctly classify the new data with a high probability.

8.1. ANALYSIS OF EACH GRAPH:

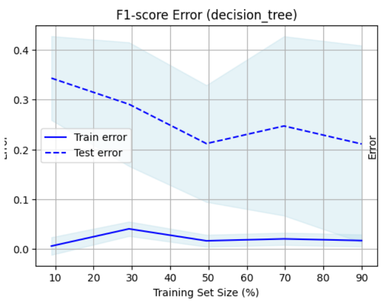
In machine learning, F1, precision, and recall are metrics used to evaluate the performance of classification models.

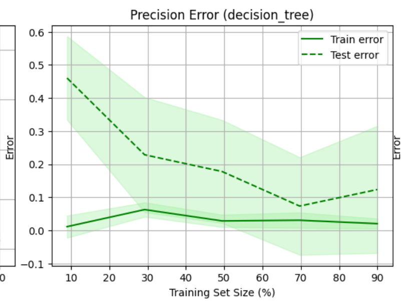
Precision refers to how close repeated measurements are to each other. It is a measure of the dispersion of a set of values.

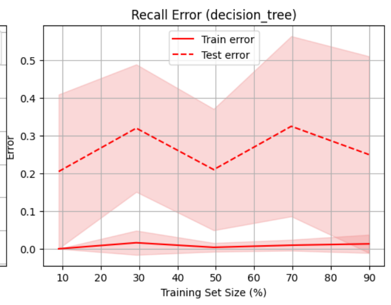
Recall is a metric that evaluates how well a model identifies positive cases within a data set. It measures the proportion of true positives that were correctly identified by the model.

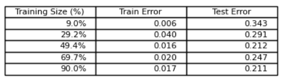
F1 is a metric that combines precision and recall into a single value. It is mainly used when looking for a balance between these two metrics, especially in problems with unbalanced classes.

8.1.1. WITHOUT SMOTE:

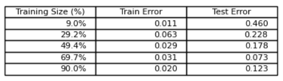




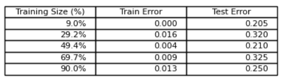




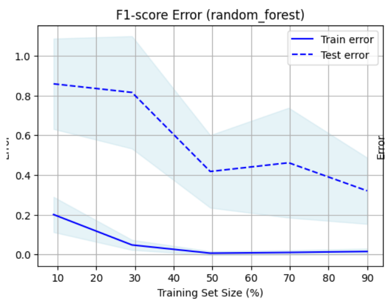
Training Size (%)	Train Error	Test Error
9.0%	0.006	0.343
29.2%	0.040	0.291
49.4%	0.016	0.212
69.7%	0.020	0.247
90.0%	0.017	0.211

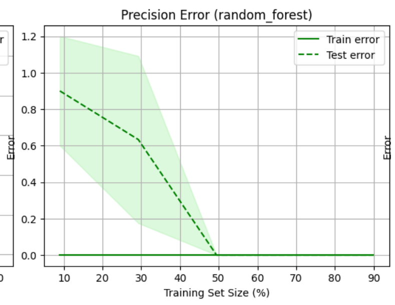


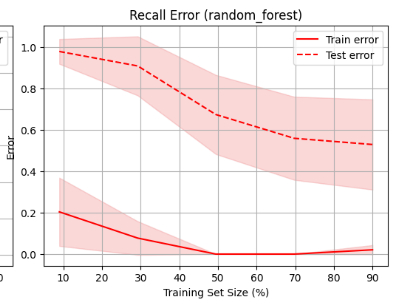
Training Size (%)	Train Error	Test Error
9.0%	0.011	0.460
29.2%	0.063	0.228
49.4%	0.029	0.178
69.7%	0.031	0.073
90.0%	0.020	0.123



Training Size (%)	Train Error	Test Error
9.0%	0.000	0.205
29.2%	0.016	0.320
49.4%	0.004	0.210
69.7%	0.009	0.325
90.0%	0.013	0.250



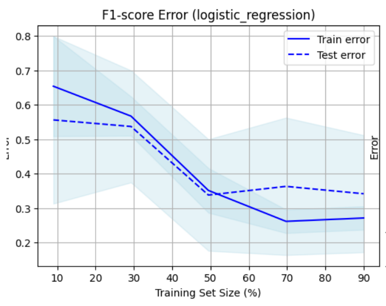


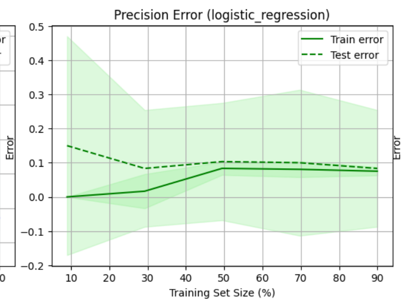


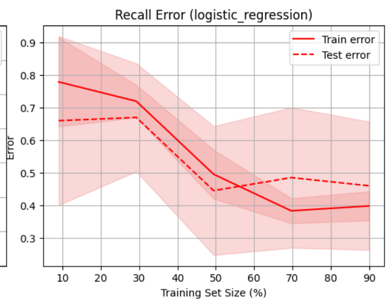
Training Size (%)	Train Error	Test Error
9.0%	0.201	0.860
29.2%	0.048	0.817
49.4%	0.006	0.418
69.7%	0.010	0.462
90.0%	0.015	0.321

Training Size (%)	Train Error	Test Error
9.0%	0.000	0.900
29.2%	0.000	0.633
49.4%	0.000	0.000
69.7%	0.000	0.000
90.0%	0.000	0.000

Training Size (%)	Train Error	Test Error
9.0%	0.204	0.980
29.2%	0.077	0.910
49.4%	0.000	0.675
69.7%	0.000	0.560
90.0%	0.021	0.530



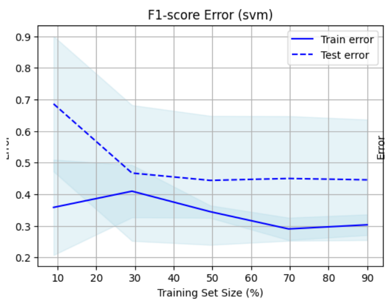


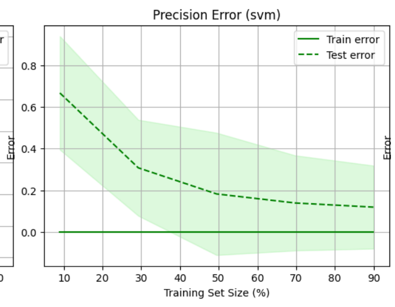


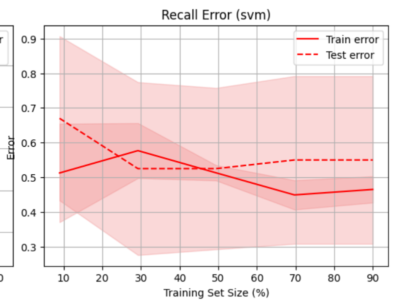
Training Size (%)	Train Error	Test Error
9.0%	0.655	0.557
29.2%	0.568	0.538
49.4%	0.352	0.338
69.7%	0.262	0.364
90.0%	0.272	0.342

Training Size (%)	Train Error	Test Error
9.0%	0.000	0.150
29.2%	0.017	0.083
49.4%	0.083	0.103
69.7%	0.081	0.100
90.0%	0.075	0.083

Training Size (%)	Train Error	Test Error
9.0%	0.779	0.660
29.2%	0.720	0.670
49.4%	0.495	0.445
69.7%	0.383	0.485
90.0%	0.398	0.460



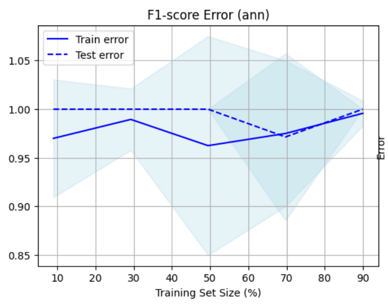


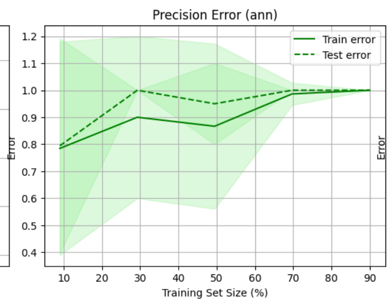


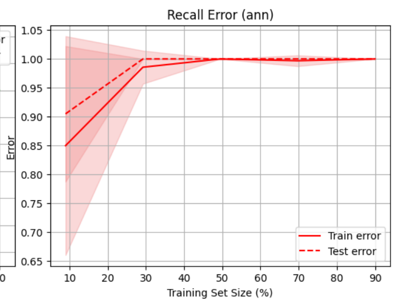
Training Size (%)	Train Error	Test Error
9.0%	0.359	0.686
29.2%	0.410	0.467
49.4%	0.345	0.444
69.7%	0.291	0.450
90.0%	0.304	0.446

Training Size (%)	Train Error	Test Error
9.0%	0.000	0.667
29.2%	0.000	0.308
49.4%	0.000	0.183
69.7%	0.000	0.140
90.0%	0.000	0.120

Training Size (%)	Train Error	Test Error
9.0%	0.512	0.670
29.2%	0.577	0.525
49.4%	0.513	0.525
69.7%	0.449	0.550
90.0%	0.465	0.550







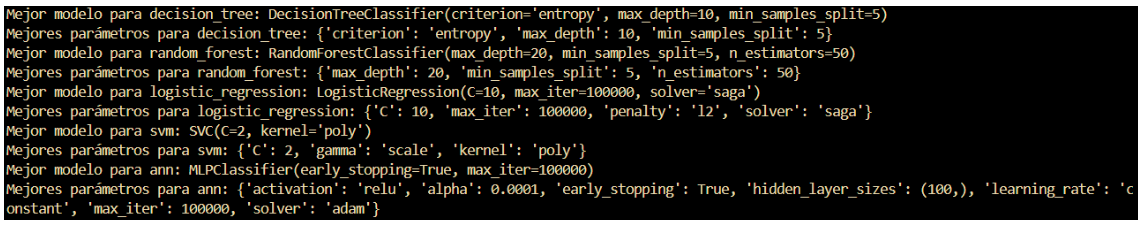
Training Size (%)	Train Error	Test Error
9.0%	0.970	1.000
29.2%	0.989	1.000
49.4%	0.963	1.000
69.7%	0.975	0.971
90.0%	0.996	1.000

Training Size (%)	Train Error	Test Error
9.0%	0.785	0.794
29.2%	0.900	1.000
49.4%	0.867	0.950
69.7%	0.986	1.000
90.0%	1.000	1.000

Training Size (%)	Train Error	Test Error
9.0%	0.850	0.905
29.2%	0.986	1.000
49.4%	1.000	1.000
69.7%	0.997	1.000
90.0%	1.000	1.000

These graphs show us how the performance of all models changes as we increase the size of the training set. They allow us to evaluate the generalization ability of the model, that is, how well it works with new data that it has not seen during training.

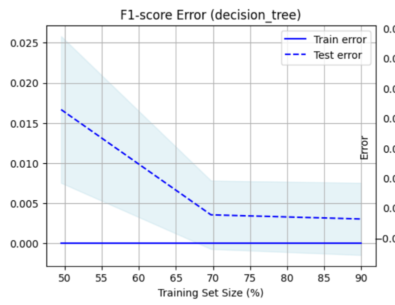
After generating the figures without SMOTE, the terminal shows which is the best model and the best parameters of each one:

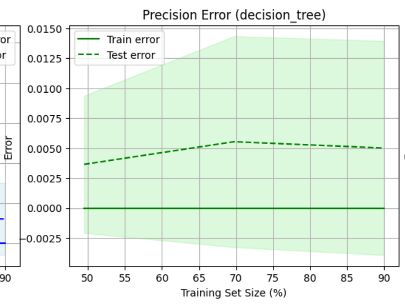


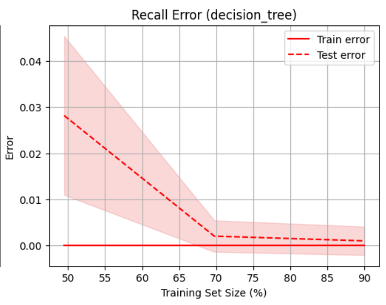
```
Mejor modelo para decision_tree: DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_split=5)
Mejores parámetros para decision_tree: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_split': 5}
Mejor modelo para random_forest: RandomForestClassifier(max_depth=20, min_samples_split=5, n_estimators=50)
Mejores parámetros para random_forest: {'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 50}
Mejor modelo para logistic_regression: LogisticRegression(C=10, max_iter=100000, solver='saga')
Mejores parámetros para logistic_regression: {'C': 10, 'max_iter': 100000, 'penalty': 'l2', 'solver': 'saga'}
Mejor modelo para svm: SVC(C=2, kernel='poly')
Mejores parámetros para svm: {'C': 2, 'gamma': 'scale', 'kernel': 'poly'}
Mejor modelo para ann: MLPClassifier(early_stopping=True, max_iter=100000)
Mejores parámetros para ann: {'activation': 'relu', 'alpha': 0.0001, 'early_stopping': True, 'hidden_layer_sizes': (100,), 'learning_rate': 'c
onstant', 'max_iter': 100000, 'solver': 'adam'}
```

8.1.2. WITH SMOTE:

SMOTE (synthetic Minority Over-sampling Technique) is a technique used in machine learning to solve a problem that occurs when we have unbalanced data sets. SMOTE helps balance the scale by creating synthetic examples of the minority class. Instead of simply duplicating existing examples, SMOTE generates new "invented" examples from existing ones.



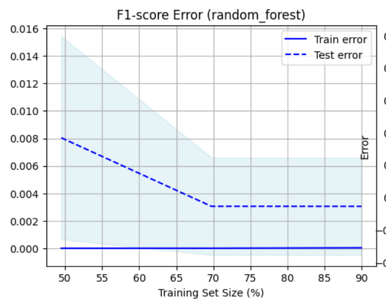


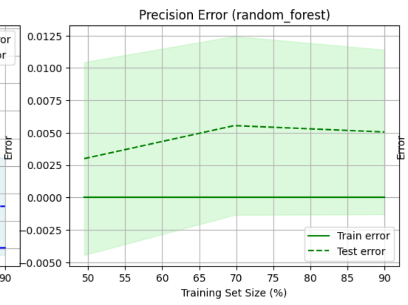


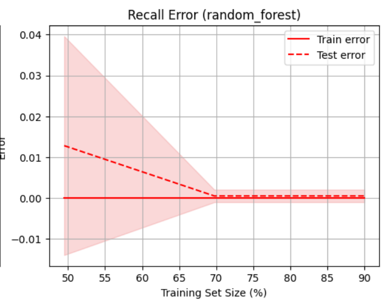
Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.000	0.017
69.7%	0.000	0.004
90.0%	0.000	0.003

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.000	0.004
69.7%	0.000	0.006
90.0%	0.000	0.005

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.000	0.028
69.7%	0.000	0.002
90.0%	0.000	0.001



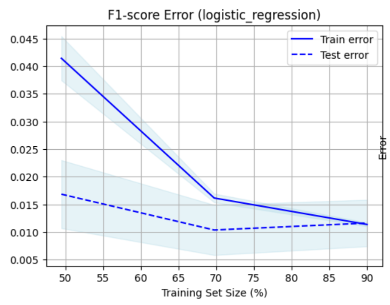


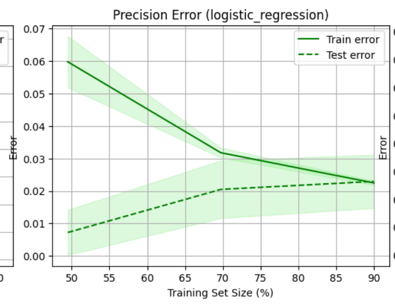


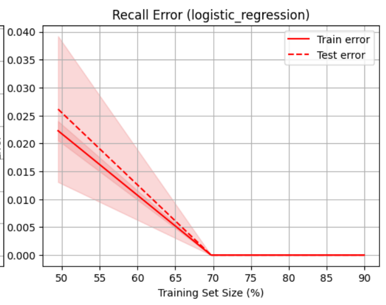
Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.000	0.008
69.7%	0.000	0.003
90.0%	0.000	0.003

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.000	0.003
69.7%	0.000	0.006
90.0%	0.000	0.005

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.000	0.013
69.7%	0.000	0.001
90.0%	0.000	0.001



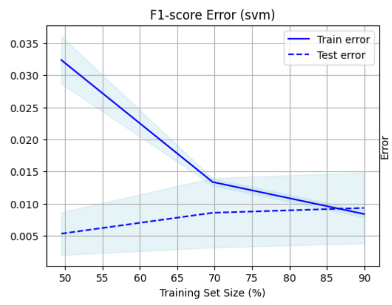


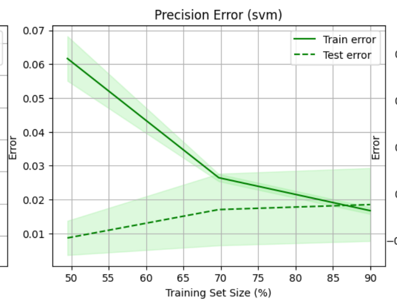


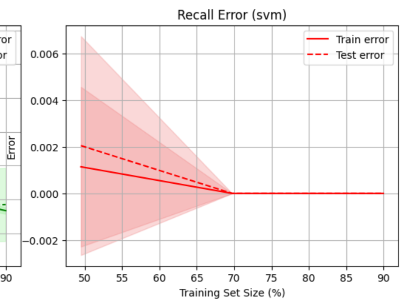
Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.041	0.017
69.7%	0.016	0.010
90.0%	0.011	0.012

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.060	0.007
69.7%	0.032	0.020
90.0%	0.023	0.023

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.022	0.026
69.7%	0.000	0.000
90.0%	0.000	0.000



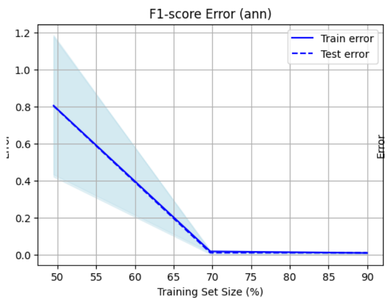


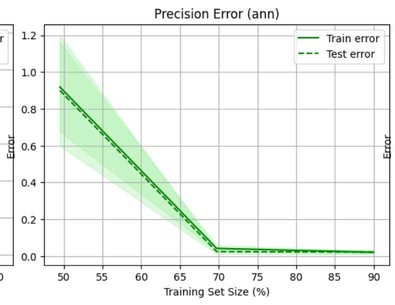


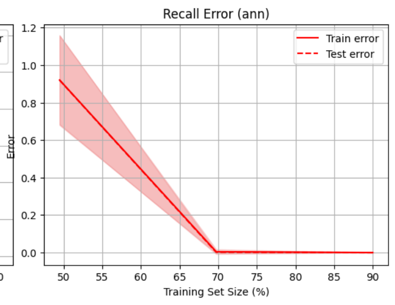
Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.032	0.005
69.7%	0.013	0.009
90.0%	0.008	0.009

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.062	0.009
69.7%	0.026	0.017
90.0%	0.017	0.018

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.001	0.002
69.7%	0.000	0.000
90.0%	0.000	0.000







Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.806	0.804
69.7%	0.017	0.010
90.0%	0.008	0.009

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.919	0.900
69.7%	0.042	0.024
90.0%	0.021	0.022

Training Size (%)	Train Error	Test Error
9.0%	nan	nan
29.2%	nan	nan
49.5%	0.921	0.920
69.7%	0.005	0.002
90.0%	0.000	0.000

In all cases we can see that the test error decreases as the training set size increases. This indicates that models are learning the data correctly. Furthermore, we can see that the model is not overfitted, that the models work correctly and that the unseen data are optimally generalized.