Universidad Autónoma de Madrid

Escuela Politécnica Superior

23|24

Bachelor thesis

Comparative Analysis of Classifiers for Breast Cancer Detection with Visualizations

Iván Sotillo del Horno

Campus Internacional excelencia UAM CSIC

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Bachelor as Ingeniería Informática (modalidad bilingüe)

BACHELOR THESIS

Comparative Analysis of Classifiers for Breast Cancer Detection with Visualizations

Author: Iván Sotillo del Horno
Advisor: Alejandro Bellogín Kouki

marzo 2024

Francisco Tomás y Valiente, n^o 1
Madrid, 28049
Spain

Iván Sotillo del Horno
Comparative Analysis of Classifiers for Breast Cancer Detection with Visualizations

Iván Sotillo del Horno

PRINTED IN SPAIN

A mi madre y a mi abuela, cuya lucha contra el cáncer de mama me ha inspirado a realizar este trabajo.

Resumen

Esta tesis presenta un análisis comparativo de clasificadores para la detección del cáncer de mama y el uso de Inteligencia Artificial Explicable (XAI) para interpretar los resultados. En la fase inicial se realizará la construcción y optimización de los modelos de clasificación, estos clasificadores analizarán los resultados de las biopsias de aguja fina y clasificarán las muestras como benignas o malignas.

Posteriormente, se realiza una comparación de rendimiento comparando métricas como la puntuación F1 o la *recall*. El objetivo es identificar el mejor clasificador de acuerdo a nuestras métricas. Una vez encontrado el mejor modelo, nos adentramos más en él para entender cómo funciona. Para esto, utilizaremos SHAP (SHapley Additive exPlanations), un método de XAI que nos permite ver la importancia de cada característica y cómo contribuyen a la decisión final del modelo. Esto nos permitirá no solo clasificar las muestras, sino también entender por qué el modelo ha tomado esa decisión, lo que puede ser un avance en la comprensión de los modelos de IA para fines médicos.

Palabras clave

Detección de Cáncer de Mama, Clasificadores, Análisis Comparativo, Interpretabilidad, SHAP, IA Explicable, Visualización

Abstract

This thesis presents a comparative analysis of base and ensemble classifiers for breast cancer detection and the use of eXplainable AI (XAI) to interpret the results. The initial phase involves constructing and optimizing the classifier models, these classifiers will analyze the results from fine needle biopsy aspirations and classify the samples as benign or malignant.

Following this, a performance comparison is conducted comparing metrics such as the F1 score or the recall. The aim is to identify the best classifier regarding our metrics. Once the best classifier model is found, we dive deeper into it to understand how it works. For this, we will use SHAP (SHapley Additive exPlanations), a method of XAI (eXplainable AI) that allows us to see the importance of each feature, and how they contribute to the final decision of the model. This will allow us to not only classify the samples but also to understand why the model has made that decision which can be a step forward in understanding AI models for medical purposes.

Keywords

Breast Cancer Detection, Classifiers, Comparative Analysis, Interpretability, SHAP, eXplainable AI, Visualization

Table of Contents

Lists

List of algorithms

List of codes

List of equations

List of figures

List of tables

1

Introduction

1.1. Motivation

Breast cancer is the most common cancer type among women [**?**]; in 2020, there were more than 2.26 million women diagnosed with breast cancer [**?**], being the second leading cause of death among women in the United States [**?**]. Early detection is a crucial step for improving survival rates. With the current analysis techniques of FNAB (Fine Needle Aspiration Biopsy), we have a sensitivity (ability of a test to identify positive cases correctly) of 0.927 [**?**]. Therefore, there is a need for a more accurate interpretation of those tests.

Machine learning is a branch of artificial intelligence that focuses on developing algorithms that can learn from data and extract patterns from it to be then able to generalize it to unseen data. In this case, we care about classifiers, whose potential is in the ability to learn from a dataset and then on unseen data being able to classify it as one class or another; in this case, we will be able to classify as benign or malign the results of a fine needle aspiration.

The potential of classifiers in breast cancer detection is immense. However, the effectiveness of the different classifiers can vary; this is why it is crucial to understand how each classifier works, how to tweak it, and how to make them as precise and effective as possible, which is the goal of this thesis.

Finding the best possible classifier for this problem would impact cancer detection tasks, facilitating healthcare professionals in their diagnostic responsibilities and, ultimately, improving patient outcomes.

1.2. Objectives

1.3. Structure of the document

2

State of the art

2.1. Base and Ensemble Classifiers

Classification is the process of taking input and assigning it to a class, it is a fundamental task in machine learning. There are different types of classification, but for this work, we are going to focus on binary supervised classification. Binary means that the classification will only be carried out between two classes, in our case, benign and malignant; supervised means that the training process will be performed with labeled data, that is, we will train that classifier with Fine Needle Aspiration Biopsy (FNAB) samples that have been labeled as benign or malignant by a pathologist. Our classifiers will not use the image directly, but rather the features extracted from the images.

A classifier is an algorithm that performs the aforementioned task, it receives an input and assigns it to a class. To train a classifier we must first have a dataset, this dataset will be divided into two parts, the training set, and the test set. The training set will be used to train the classifier, which means that the classifier will learn from this data trying to find patterns that allow it to classify the samples correctly. The test set will be used to evaluate the classifier, this set will be used to see how well the classifier performs with data that it has not seen before, this is done to ensure that the classifier is not overfitting (memorizing the training data) and that it can generalize to new data.

Classifiers can be split into two main groups, base classifiers and ensemble classifiers. Their main difference is that base classifiers are single classifiers, while ensemble classifiers are composed of multiple base classifiers. In this section, we will give an overview of the most common base and ensemble classifiers.

2.1.1. Base Classifiers

Base classifiers as just mentioned are single classifiers, they are the simplest form of classifiers, and they are the building blocks of ensemble classifiers. There are many base classifiers, but for this work, we are going to focus on the most common ones.

Logistic Regression

Logistic Regression (LR) is a probabilistic model that uses a logistic function to predict the probability of a binary outcome. The central idea is to find the best-fitting model to describe the relationship between the binary dependent variable and one or more independent variables.

Multilayer Perceptron

A Multilayer Perceptron (MLP) is a neural network composed of perceptrons, a perceptron is a simple model of a biological neuron, it takes several binary inputs and produces a single binary output using a weighted sum of the inputs and an activation function.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is not a classifier itself, but an optimization algorithm used instead of the standard gradient descent algorithm, it is used to minimize the loss function of a classifier. It is in this list because in Scikit-learn it can be used as a classifier by indicating the linear model to be used [**?**].

Support Vector Machine

Support Vector Machine (SVM) is a statistical model that separates data into classes by finding a hyperplane that separates the classes, since there can be multiple hyperplanes that do so, the model tries to find the one that maximizes the margin between the classes.

K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a model that classifies the sample based on the majority of its neighbors, the number of neighbors, k, is a parameter that must be set by the user.

Linear and Quadratic Discriminant Analysis

Two different models, Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA). They are used to find a linear or quadratic decision boundary that separates the classes.

Decision Tree

Decision Tree (DT) is a model that separates the data into classes by asking a series of questions, each question is based on the value of a feature, and the answer to the question determines the next question. It is a tree-like model, where the leaves are the classes.

Gaussian Naive Bayes

Gaussian Naive Bayes (GNB) is a probabilistic model that uses Naive Bayes' theorem with the assumption that the features are independent, given the class. It is called Gaussian because it assumes that the features are normally distributed.

2.1.2. Ensemble Classifiers

Ensemble classifiers are a type of classifier that combines the predictions of multiple base classifiers to improve the accuracy of the predictions. Some of the ensemble classifiers allow the base classifiers to be chosen, which means that we can use the best base classifiers to build the ensemble classifier. In this section, we are going to give an overview of the ensemble classifiers that we are going to use in this project.

Gradient Boosting

Gradient Boosting (GB) is an ensemble machine learning technique that combines the predictions from several models to improve the overall predictive accuracy. It works by first building a weak model (a model that is slightly better than random guessing) and then building a second model that corrects the errors made by the first model and so on [**?**], we can see a graphical representation of the GB in the Figure 2.1.

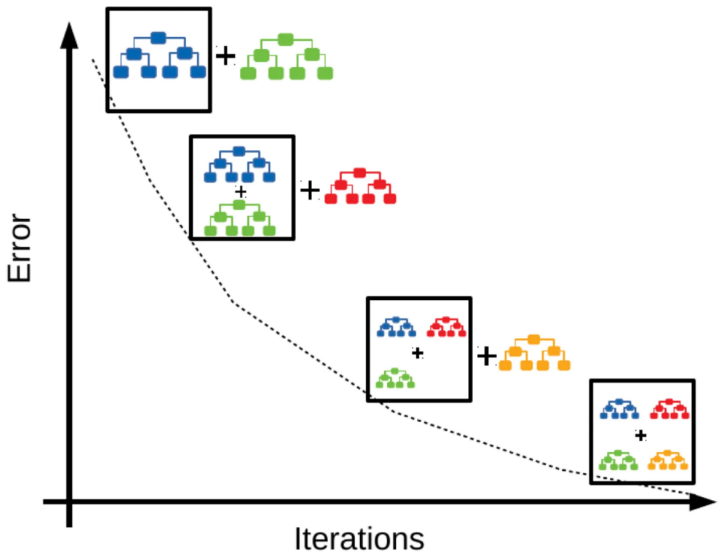

Figure 2.1: Graphical representation of Gradient Boosting [**?**]

AdaBoost

AdaBoost (AB) is similar to GB in the sense that it combines the predictions of multiple models but it does so by giving more weight to the misclassified data points in each iteration.

Random Forest

Random Forest (RF) is based on using DT as base classifiers. It builds multiple DT and combines their predictions, it is called random because it uses a random subset of the features to build the DT [**?**], this randomness is called bagging. We can see a graphical representation of the RF in the Figure 2.2.

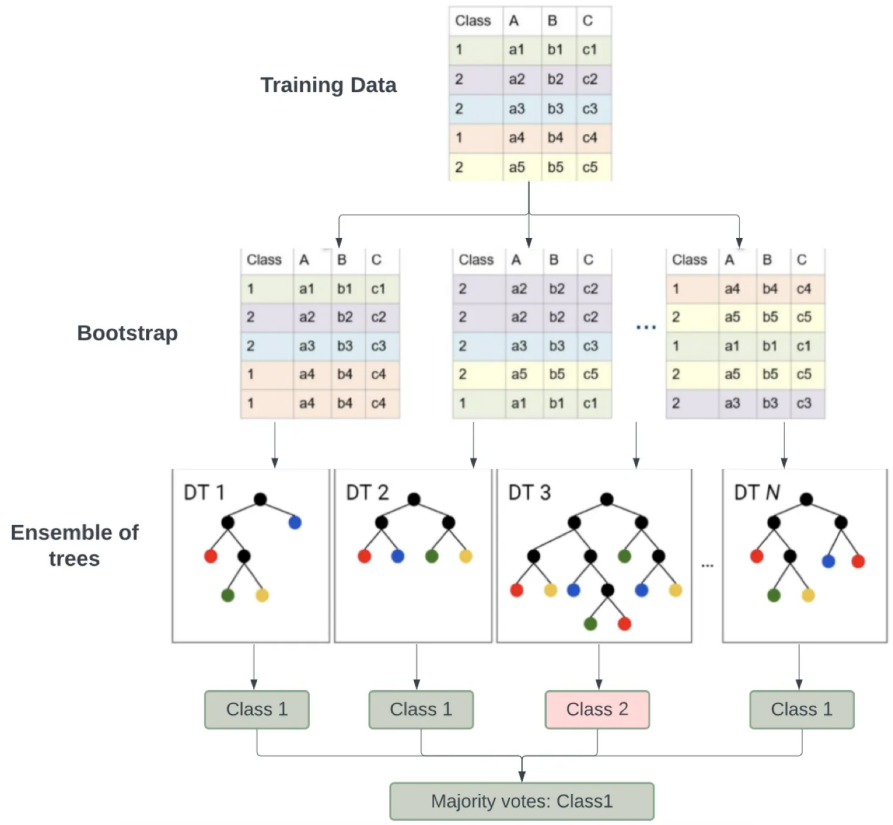

Figure 2.2: Graphical representation of Random Forest [**?**]

Voting Classifier

A Vote Classifier (VC) as its name suggests, is an ensemble classifier that combines the predictions of multiple classifiers by voting. It can be hard or soft voting, hard voting is the usual voting where the class that gets the most votes is the final prediction, and soft voting is when the average of the probabilities of the classes is used to make the final prediction.

Bagging Classifier

A Bagging Classifier (BC) is an ensemble classifier that builds multiple base classifiers and combines their predictions by averaging or voting, as it was mentioned before, in bagging the base classifiers are built using a random subset of the features [**?**], we can see a graphical representation of the BC in the Figure 2.3.

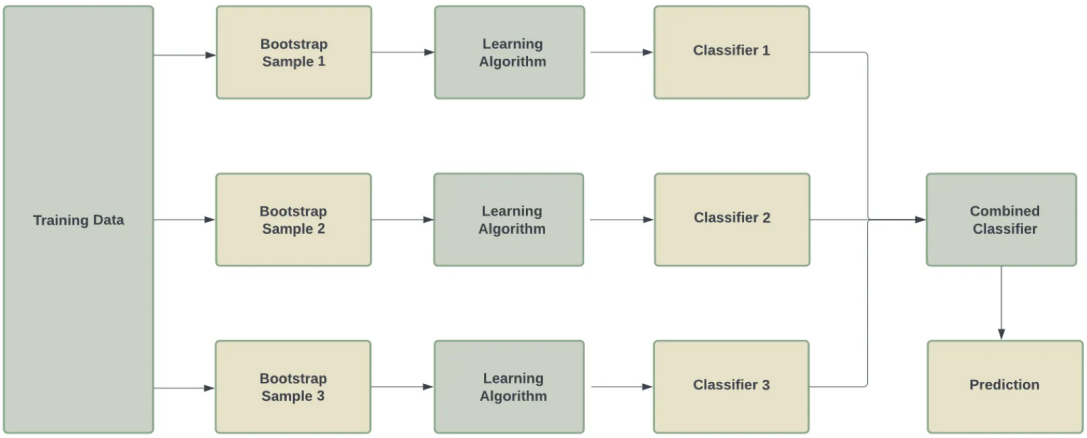

Figure 2.3: Graphical representation of Bagging [**?**]

Stacking Classifier

A Stacking Classifier (SC) is an ensemble that combines the base classifiers into a meta-classifier, that is, the sample to be classified goes through the base classifiers and then the predictions of the base classifiers are used as features to build the meta-classifier [**?**]. We can see a graphical representation of the SC in the Figure 2.4.

2.2. Evaluation of Classifiers

Now that we know what a classifier is, how it works, and the different types of classifiers, we need to know which classifier is the best for our problem, for this, we need to use some metric to evaluate the performance of the classifiers. Before we describe the metrics we need to understand the values that we are going to use to calculate these metrics, the values are:

- True Positive (TP): The number of instances that are positive and the classifier correctly classified as positive.
- True Negative (TN): The number of instances that are negative and the classifier correctly classified as negative.
- False Positive (FP): The number of instances that are negative and the classifier incorrectly classified as positive.
- False Negative (FN): The number of instances that are positive and the classifier incorrectly classified as negative.

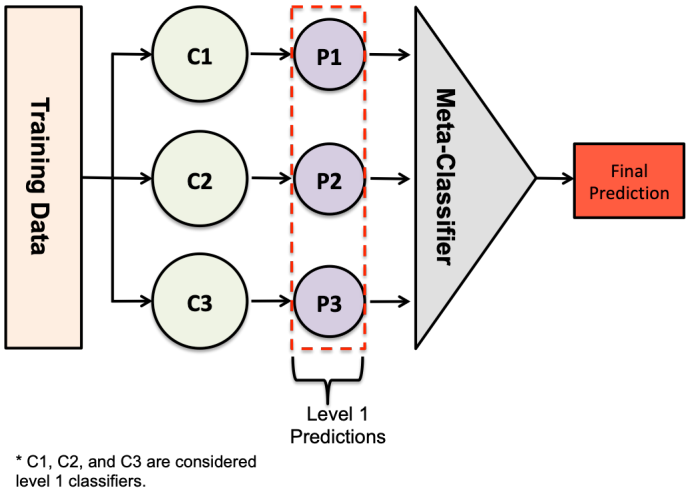

Figure 2.4: Graphical representation of Stacking [**?**]

Now that we have our building blocks for the metrics, we can describe the main four metric [**?**]:

- **Accuracy**: Proportion of correct predictions over the total number of instances evaluated. $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
- **Recall**: Proportion of TP results among the number of all positive instances. $Recall = \frac{TP}{TP+FN}$
- **Precision**: The proportion of TP results among the number of cases that were predicted to be positive. $Precision = \frac{TP}{TP+FP}$
- **Specificity**: The proportion of TN results among the number of cases that were actually negative. $Specificity = \frac{TN}{TN+FP}$

A confusion matrix is a table, that is used to see the performance of the classifier in a more detailed way, it shows the number of TP, TN, FP, and FN, this is shown in Table 2.1. We can use this table to calculate all the metrics we have seen before. The table is split into four quadrants, the top left quadrant is the TP, the top right quadrant is the FP, the bottom left quadrant is the FN, and the bottom right quadrant is the TN, what we want is to have as many TP and TN as possible since these are the ones that have been correctly classified.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive	False Negative
Actual Negative	False Positive	True Negative

Table 2.1: Confusion Matrix

The problem with the accuracy metric is that it is not a good metric when the classes are imbalanced, for example, if we have 95 % of the instances of class 1 and 5 % of the instances of class 2, a classifier that always predicts the class 1 will have a 95 % accuracy, but it is not a good classifier. For this reason, we need to use other metrics, but we can also fix the accuracy metric by using the *Balanced Accuracy*, which is the average of the recall of each class, this metric is better than the accuracy when the classes are imbalanced, the formula for the balanced accuracy is shown in Equation 2.1.

$$BalancedAccuracy = \frac{Recall + Specificity}{2} \tag{2.1}$$

With this new accuracy metric, we can see a better performance of the classifier when the classes are imbalanced. We can clearly see the difference between the accuracy and the balanced accuracy with a quick example, if we have a dataset with 100 instances, 97 of them are positive and 3 of them are negative, and we have a classifier that predicted 96 samples as positive and 4 as negative, this is expressed in the Table 2.2, lets calculate the accuracy and the balanced accuracy.

	Predicted Positive	**Predicted Negative**
Actual Positive	94	3
Actual Negative	2	1

Table 2.2: Confusion Matrix Example

- **Accuracy**: $\frac{TP+TN}{TP+TN+FP+FN} = \frac{94+1}{94+1+2+1} = \frac{95}{100} = 95{,}00\,\%$
- **Recall**: $\frac{TP}{TP+FN} = \frac{94}{94+3} = 96{,}91\,\%$
- **Specificity**: $\frac{TN}{TN+FP} = \frac{1}{1+3} = 25{,}00\,\%$
- **Balanced Accuracy**: $\frac{Recall+Specificity}{2} = \frac{0{,}9691+0{,}25}{2} = 60{,}95\,\%$

As we can see, we obtained a 95 % accuracy, but the balanced accuracy is 60.95 %, this is a big difference and represents better the performance of the classifier which misclassified a high percentage of the negative instances.

Apart from these basic metrics we have other metrics that are based on these, for example, we have the *F-Beta Score*, which is the weighted harmonic mean of the precision and recall, depending on the value of β we can give more importance to the precision or the recall, for example, if we want to give more importance to the recall we can use the *F2 Score*, which means that the β is 2, and if we want to give more importance to the precision we can use the *F0.5 Score*, which means that the β is 0.5. The general formula for the F-Beta Score is shown in Equation 2.2. This metric can be really useful when we want to give more importance to one of the metrics, for example, if we are predicting a disease, we want to give more importance to the recall since we want to detect as many cases as possible, even if we have some FP, but if we are predicting if a person is going to buy a product, we want to give more importance to the precision, since we want to avoid FP.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall} \tag{2.2}$$

We also have the *Matthews Correlation Coefficient*, this metric tries to summarize the confusion matrix into a single value, it doesn't take into account the class imbalance, so when the classes are imbalanced Fβ Score is better [**?**], but when the classes are balanced the Matthews Correlation Coefficient can be useful, it measures the correlation between the true and predicted values [**?**]. The formula for the Matthews Correlation Coefficient is shown in Equation 2.3.

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.3}$$

The last metric we are going to mention is the *ROC AUC*, this metric is used to measure the performance of the classifier, it is the area under the curve of the Receiver Operating Characteristic (ROC) curve, this curve is a plot of the true positive rate (Recall) against the False Positive rate ($FPR = \frac{FP}{FP+TN}$), the ROC curve is a good way to see the performance of the classifier, the closer the curve is to the top left corner, the better the performance of the classifier. The area under the curve is the ROC AUC, and it is a value between 0 and 1, the closer to 1 the better the performance of the classifier, if the area is 0.5 or the curve is close to the diagonal, the classifier is not better than random guessing. An example of an ROC curve is shown in Figure 2.5, we can see that the curve is somewhat close to the top left corner so it is a good classifier but there is still room for improvement. This metric is really useful when the classes are balanced, if the classes are imbalanced the ROC AUC can be misleading, and thus we will want to use the aforementioned metric Fβ Score.

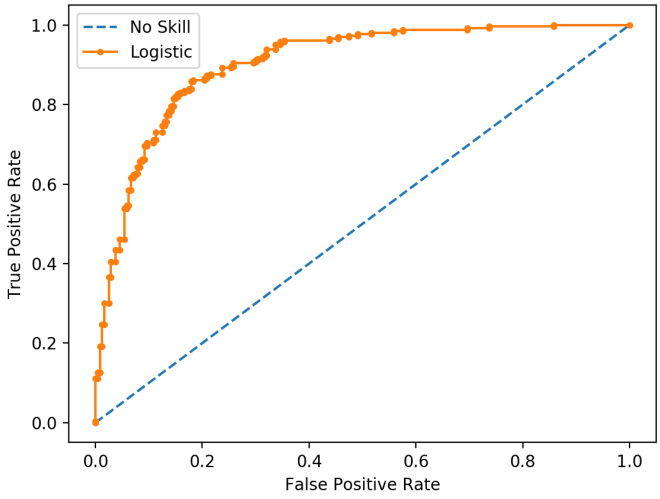

Figure 2.5: Example of a ROC Curve, the y-axis is the True Positive Rate or Recall, and the x-axis is the FP Rate

2.3. Classifier Optimization

We have understood the metrics that are used to evaluate the performance of a classifier, now we can move to the optimization of the classifier. The optimization of a classifier is the process of finding the best hyperparameters for it, the hyperparameters are the parameters that are not learned by the model, for example, the number of trees in a Random Forest, the number of neighbors in a K-Nearest Neighbors, etc. These parameters can heavily influence the performance of the classifier. The two main methods to optimize a classifier are Grid Search and Random Search.

- **Grid Search**: This method is a brute force method, it tries all the possible combinations of the hyperparameters, for example, if we have two hyperparameters, one with 3 possible values and the other with 4 possible values, the Grid Search will try 12 different combinations. This method is very useful when we have a small number of hyperparameters, but it is not efficient when we have a large number of hyperparameters since the number of combinations grows exponentially.
- **Random Search**: This method instead of trying all the possible combinations, tries a random subset of the combinations. The number of combinations to try is a parameter of the method. Since it doesn't try all the combinations, it is faster than the Grid Search, but it is not guaranteed to find the best combination of hyperparameters.

Both of these methods use Cross Validation (CV), a technique that is used to evaluate the performance of the classifier without using the test set. The CV technique splits the dataset into k folds, then leaves one fold out and trains the classifier with the remaining $k-1$ folds, then it evaluates the classifier with the left out fold and repeats this process changing the fold that is left out, this process is repeated k times, and the average of the results is the final result. This technique is used to avoid overfitting the classifier to the training set. An example of a CV with 5 folds is shown in Figure 2.6.

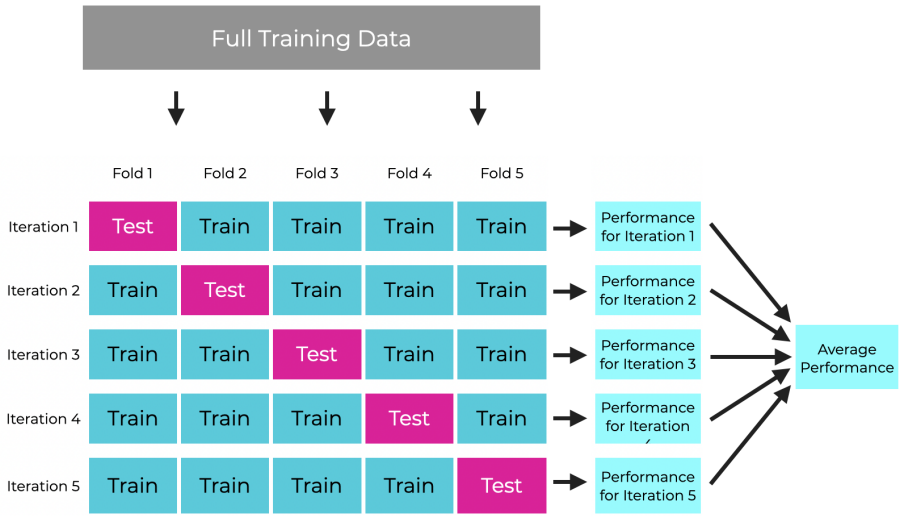

Figure 2.6: Graphical representation of a 5 fold CV [**?**]

As we saw, both methods have their advantages and disadvantages, the Grid Search is guaranteed to find the best combination of hyperparameters, but it is not efficient when we have a large number of hyperparameters since it can take hours and even days to finish, the Random Search is faster, but it is not guaranteed to find the best combination of hyperparameters. Researchers have found that Random Search is more efficient than Grid Search, if both methods have the same number of iterations, Random Search will find a better combination of hyperparameters [**?**]. Knowing this we can first use Random Search to find a boundary of the best hyperparameters and then use Grid Search to find the best combination of hyperparameters.

2.4. Explainable AI

Classifiers are a really helpful tool, but they are a black box, that is, they are fed with an input and provide an output, but we do not know why they made a certain decision. This is a problem in the medical field since we need to know why a certain decision was made, for example, if a patient has cancer, we need to know which features determine that the patient has cancer, and which ones determine that the patient does not have cancer. This is where Explainable Artificial Intelligence (XAI) comes into play, XAI is a branch of Artificial Intelligence (AI) that is focused on making Machine Learning (ML) models transparent, explaining why a certain decision was made and allowing doctors to understand the decision [**?**].

2.4.1. SHAP

SHapley Additive exPlanations (SHAP) [**?**] is a model-agnostic approach to XAI, that is, it can be used with any model since it only uses the input and output. In short, how SHAP works, is that it uses the Shapley values from cooperative game theory to explain the output of any machine learning model, a concept named after Lloyd Shapley who introduced the idea that got him the Nobel Prize in Economic Sciences in 2012 [**?**].

The Shapley value is a concept from cooperative game theory, it is a way to fairly distribute the loot obtained by a coalition of players. The difference is that in SHAP instead of players we have features and instead of loot, we have the predicted output as a probability. The Shapley value is the average contribution of a feature to the prediction, and it is calculated by averaging the marginal contributions of a feature to the prediction over all possible orderings of the features. The marginal contribution of a feature to the prediction is the difference between the prediction with the feature and the prediction without the feature. This is done for all possible subsets of features, and then the Shapley value is the average of all these marginal contributions. This is done for all the features, and then we have the Shapley values, which are the importance of the features for the prediction.

To better understand how these values are obtained let's give an example. To change topics instead of breast cancer, let's say we want to predict the price of a car, and for this, we will have three features, the year of the car, the horsepowers, and the brand. Now, we want to know how much each feature contributes to the price of the car. To do this first we have to obtain the **powerset** of the features, which is the set of all possible subsets of the features, going from $f = 0$ to $f = 3$, where f is the number of features in the subset, we can see the power set in Figure 2.7.

Now that we have all the possible subsets of the features, the next step is to train a model for each subset, the number of models is given by the cardinality of the power set, 2^f where f is the number of features, in this case $2^3 = 8$. All the models are equal, same architecture, same hyperparameters, the only difference is the number of features used to train the model. Now imagine that we have trained all

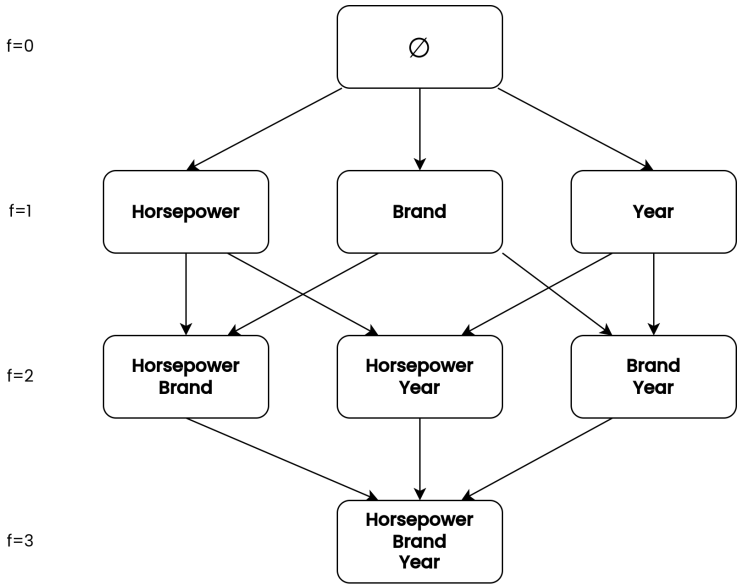

Figure 2.7: Powerset with the features of the car example

the models, and we have the predictions for all the models, we can see the predictions in Figure 2.8.

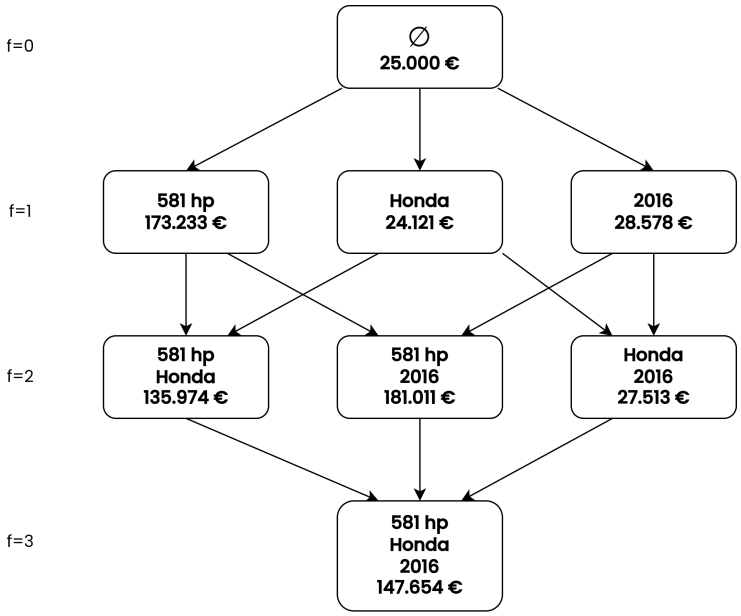

Figure 2.8: Example predictions for the powerset of the car features

All the nodes that are connected differ by just one feature, knowing this we can see that, for example, the base model, the one with no features, has a prediction of 25.000 euros, and the model with the year of the car has a prediction of 28.578 euros, this means that the year of the car contributes 3.578 euros to the price of the car, and for example, the brand, Honda, contributes -879 euros. This is called marginal contribution and the formula is shown in Equation 2.4.

$$MC_{Brand,\{Brand\}(x)} = Prediction_{\{Brand\}(x)} - Prediction_{\emptyset(x)} = 24{,}121 - 25{,}000 = -879 \quad (2.4)$$

Each edge also has a weight, the important thing to note is that the weights are the same for all the edges in a row and that the weights in a row add up to 1. The way to obtain this weight then, is by knowing the number of edges in a row and then dividing 1 by the number of edges to obtain the number of edges we can use basic combinatorics. First, the number of nodes in a row is given by the binomial coefficient, which is given by $\binom{F}{f}$, where F is the number of features, and f is the number of features in the subset just like in the power set. Then the number of edges is given by $f \cdot \binom{F}{f}$, that is, to get the weights of the nodes that connect the base model ($f = 0$) to the models with one feature ($f = 1$) we have to divide 1 by the number of edges, which is $1 \cdot \binom{3}{1} = 3$, and then we get the weights 1/3, 1/3, and 1/3. This is done for all the rows, and we obtain the weights shown in Table 2.3.

f	$\binom{F}{f}$	$f \cdot \binom{F}{f}$	Weights
0	1	0	0
1	3	3	1/3
2	3	6	1/6
3	1	3	1/3

Table 2.3: Weights for the example

Now that we have the weights, we can do the weighted sum of the marginal contributions, and we obtain the Shapley value, which is the importance of the feature for the prediction. So if we do the weighted sum for the brand, we obtain the result shown in Equation 2.5.

$$\begin{aligned} SV_{Brand}(x) = &\frac{1}{3} \cdot MC_{Brand,\{Brand\}(x)} + \\ &\frac{1}{6} \cdot MC_{Brand,\{Brand,Year\}(x)} + \\ &\frac{1}{6} \cdot MC_{Brand,\{Brand,Horsepowers\}(x)} + \\ &\frac{1}{3} \cdot MC_{Brand,\{Brand,Year,Horsepowers\}(x)} \\ = &\frac{1}{3} \cdot -879 + \frac{1}{6} \cdot -37259 + \frac{1}{6} \cdot -1075 + \frac{1}{3} \cdot -33357 \\ = &-17801 \end{aligned} \tag{2.5}$$

With this, we can see that the brand contributes -17801 euros to the price of the car, and we can do the same for the other features, we obtain the Shapley values for all the features and see how much each feature contributes to the price of the car, and we can see the results in Table 2.4. If we add all the Shapley values we obtain the difference between the base prediction with no features, a lot of times represented as the mean of the target $E(y)$, and the prediction with all the features, which is the prediction of the model, $f(x)$.

Then we have that the general formula for the Shapley value is given by Equation 2.6. Now we understand how the Shapley values are obtained and what they are, and we can see why SHAP is model-agnostic.

Feature	Shapley Value
Year	6947.6
Horsepowers	133505.6
Brand	-17801
Sum	**122652.2**

Table 2.4: Shapley values for the example

$$SV_{feature}(x) = \sum_{set: feature \in set} \frac{|set|}{\binom{F}{|set|}} \cdot (prediction_{set} - prediction_{set \setminus feature}) \tag{2.6}$$

2.5. Web Application Development

3

Design and Implementation

In this section the design of the project will be shown, showing the structure of the project, its requirements, and the implementation of the different parts of the project. Additionally, the implementation of each part will be explained.

3.1. Project Structure

The structure of the project is divided into four main modules corresponding to the main stages of the project: dataset, classifiers, XAI, and web application. The dataset module is responsible for loading the dataset and splitting it into a 70 % training set and a 30 % test set. Then in the classifiers module, we do the Exploratory Data Analysis (EDA), data preprocessing, and building and optimizing the classifiers. The XAI module is responsible for the implementation of the SHAP algorithm and the analysis of the results. Finally, the web application module is responsible for the development of the web application. The structure of the project is shown in Figure 3.1. Now we will describe each module in more detail.

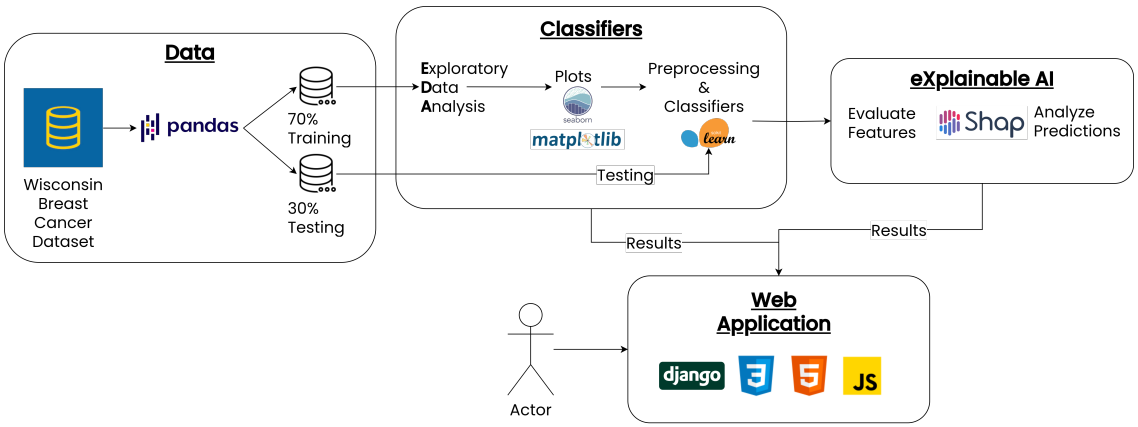

Figure 3.1: Structure of the project

1. **Dataset module**: This module is responsible for loading the dataset and splitting it into a 70 % training set and a 30 % test set. The dataset used in this project is the Breast Cancer Wisconsin dataset [**?**], which contains the parameters of the cell nuclei of the sample obtained by a FNAB of breast masses.

The dataset is loaded using the *pandas* library. The dataset is then split into a 70 % training set and a 30 % test set using *scikit-learn*. From now on, only the training will be used for the Exploratory Data Analysis, data preprocessing, and building and optimizing the classifiers. The test set will be used to evaluate the classifiers. This ensures that we avoid data leakage. An example of a FNAB is shown in Figure 3.2.

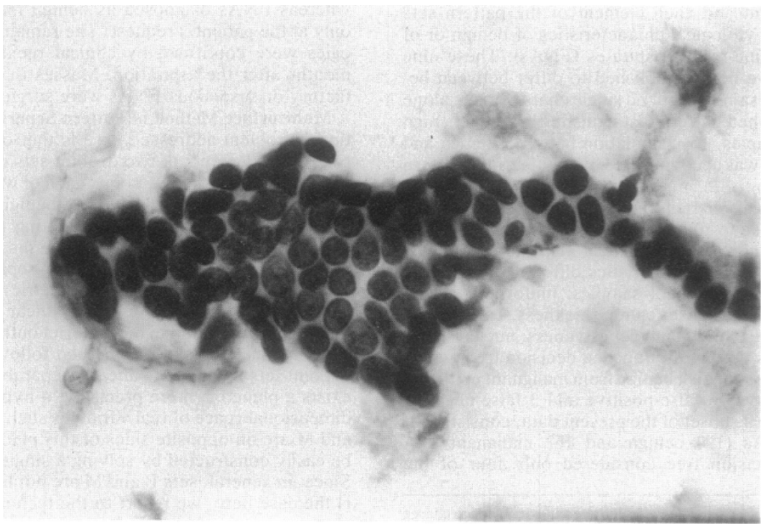

Figure 3.2: Fine Needle Aspiration Biopsy [**?**]

2. **Classifiers module**: This module is responsible for the EDA, data preprocessing, and building and optimizing the classifiers. The EDA is carried out using *matplotlib* and *seaborn* to obtain descriptive and visual statistics of the dataset. The data preprocessing is carried out using *scikit-learn* to scale and normalize the data and to apply Principal Component Analysis (PCA), this is made using pipelines that facilitate the process of building and optimizing the classifiers and their reproducibility. The building and optimization of the classifiers are carried out using *scikit-learn* using the training set to build and optimize the classifiers. Once the classifiers are built and optimized, they are evaluated using the test set.

3. **XAI module**: This module is responsible for the implementation of the SHAP algorithm and the analysis of the results. This is implemented using the *shap* library, which is a Python library that allows us to calculate the SHAP values of the classifiers, this values will be used to understand the importance of the features for each classifier and to understand how a prediction was made. This is what makes this project more interpretable than just using classifiers since this will allow a doctor to understand why a prediction was made.

4. **Web application module**: This module is responsible for the development of the web application. This website was made using *Django* as the backend, *HTML*, *CSS*, and *JavaScript* as the frontend. The website is mainly a way of visualizing the results of the project, it allows the user to see a list of all classifiers and their metrics, which features are the most important for that classifier, and how some predictions were made. It also allows the user to compare two classifiers and see the differences in

their metrics and the importance of the features.

3.2. Requirements

In this section, we will define the requirements of the project. We will divide them into functional (what the system should do) and non-functional (how the system should do it) requirements.

3.2.1. Functional Requirements

Dataset

- **FR-1.-** The application should allow the use of the Wisconsin breast cancer dataset or one with the same features.
- **FR-2.-** The dataset should be split into a training and testing dataset using a 70-30 ratio.

Classifiers

- **FR-3.-** The application should employ the features provided in the dataset for classifier comparison and XAI.
- **FR-4.-** The application should work with multiple classifiers for breast cancer detection.
- **FR-5.-** The application should preprocess the dataset before feeding it to the classifiers. This may include steps such as normalization, and feature selection, focusing on reproducibility and repeatability.
- **FR-6.-** The application should use a test dataset for classifier comparison.
- **FR-7.-** The application should use evaluation metrics to determine the effectiveness of the classifiers taking into account the nature of the problem.

Explainable Artificial Intelligence

- **FR-8.-** The application should show which features are the most important for a classifier.
- **FR-9.-** The application should show how sure the classifier was of its prediction.
- **FR-10.-** The application should show which features determined if the sample was benign or malignant.

Web Application

- **FR-11.-** The web application should have a home page with a slideshow with information.
- **FR-12.-** The web application should display a list with all the classifiers.
- **FR-13.-** The web application should show a details page about each classifier, showing its metrics, most important features, and examples of decisions made by the classifier.
- **FR-14.-** The web application should compare any two classifiers showing their metrics and most important features.

3.2.2. Non-Functional Requirements

Classifiers

- **NFR-1.-** The programming language should be Python.
- **NFR-2.-** The classifiers should be created and trained in a Jupiter Notebook.
- **NFR-3.-** The Pandas library should be used for data manipulation.
- **NFR-4.-** The Scikit-learn library should be used for the classifiers.
- **NFR-5.-** The Matplotlib and Seaborn libraries should be used for the plots.
- **NFR-6.-** The Shap library should be used to understand the feature importance of the models and how a sample decision was made.

Web Application

- **NFR-7.-** The web application should be designed for desktop usage.
- **NFR-8.-** The web application should have a pink color palette and a modern design.
- **NFR-9.-** The web application should be implemented in Django.
- **NFR-10.-** The web application should have an intuitive interface.

3.3. Exploratory Data Analysis

As it has been previously mentioned the dataset used in this project is the Breast Cancer Wisconsin dataset [**?**]. This dataset is composed of 569 samples of breast masses obtained by FNAB, then these samples were parametrized, meaning that now we will work with numbers instead of images. As a result of this process, we have 30 features that describe the cell nuclei of the sample, these features are the mean, standard error, and worst of the ten different parameters of the cell nuclei.

Before using these values in the classifier we first need to understand the dataset, this is done using EDA techniques. This involves, first obtaining a general understanding of the dataset, then visualizing the dataset, and finally understanding the relationships between the features. All of this is done only using the training set, this is to avoid data leakage.

3.3.1. Descriptive Statistics

To have a general understanding of the dataset we can use the *pandas* library to load the dataset and then use the *.describe()* method to see an overview of the features in the dataset, their mean, standard deviation, minimum, and maximum values; and then use the *.skew()* method to see the skewness of the features. With this, we would have the needed descriptive statistics of the dataset.

3.3.2. Data Visualization

After seeing a statistical overview of the dataset, we can now dive into visualizing the dataset. This is done using the *matplotlib* and *seaborn* libraries. With these libraries, we will be able to see more visually the distribution of the target variable, the distribution of the features, and the relationships between the features. To achieve this, we will use histograms, correlation matrices, pair, density, and violin plots among others. This will give us a better understanding of the dataset and the relationships between the features which will allow us to make more informed decisions when preprocessing the dataset and building the classifiers.

3.4. Data Preprocessing

Once we have done the EDA and we know how the dataset is structured, how the different features are distributed, and the relationships between the features, we can start preprocessing the dataset. To achieve this we will standardize the features and then reduce the dimensionality of the features by using PCA, this will result in a dataset that is easier to work with and that will allow us to build better classifiers [**?**]. To facilitate this process we will use pipelines, which will allow us to easily build and optimize the classifiers and to ensure reproducibility and prevent data leakage [**?**], this means that we won't directly have to modify the dataset. The Code Snippet 3.1 shows how the pipeline is built.

3.4.1. Scaling and Normalization

This is done using the *scikit-learn* library. The first step is to handle the missing values, this is done using the *SimpleImputer* class, which replaces the missing values with the mean of the feature. Then we scale the features using the *StandardScaler* class, which scales the features to have a mean of 0 and a standard deviation of 1, this is done like the Equation 3.1, where μ is the mean of all the samples of the feature, σ is the standard deviation, and x is the value we want to standardize. We can see in Figure 3.3 how the distribution of the feature stays the same but the mean is 0 and the standard deviation is 1 after standardization.

$$x' = \frac{x - \mu}{\sigma} \tag{3.1}$$

3.4.2. Principal Component Analysis

Lastly, we select the most important variables using the PCA, what this does is that it takes the features and transforms them into a new set of features that are linear combinations of the original features trying to maximize the variance of the new features while the correlation between the new

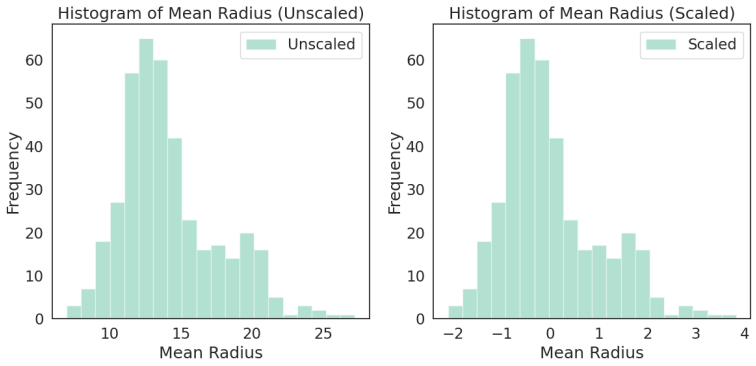

Figure 3.3: Standardization before and after of mean radius

features is 0, this is done using orthogonal transformations [**?**], in our case instead of specifying the number of components we want to keep, we specify the percentage of variance we want to keep, in this case, we want to keep 95 % of the variance, this is done so that we can reduce the number of features while keeping most of the information. In our case, when we plot the cumulative explained variance we can see that with 10 features we can keep 95 % of the variance, this is shown in the Figure 3.4. The Code Snippet 3.1 shows how the pipeline is built.

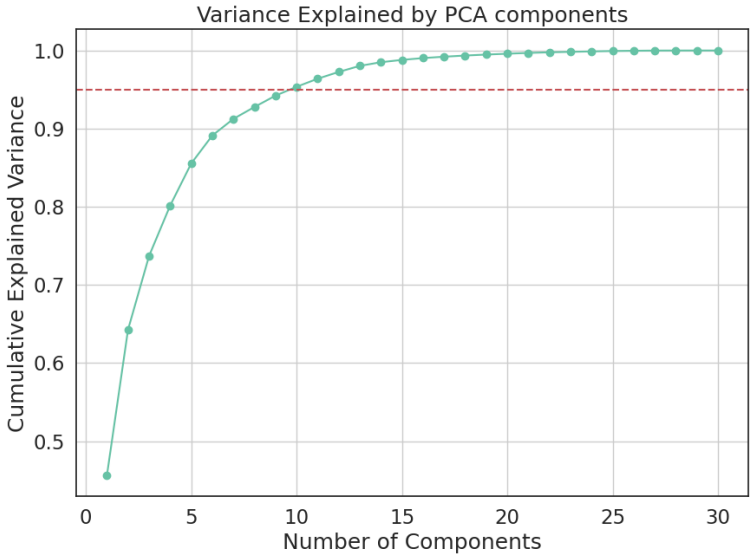

Figure 3.4: Cumulative explained variance when using PCA

3.5. Building and Optimizing Classifiers

To build the classifiers, *Scikit-learn* was used. This library was chosen because it is the most popular library for building machine learning models in *Python* and it has a wide variety of classifiers and ensemble methods. The classifiers were built in a *Jupyter Notebook* to allow reproducibility and to facilitate the process of building and optimizing the classifiers. The classifiers can be divided into two

Code 3.1: Building the pipeline with the StandardScaler and PCA

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),
    ('clf', Classifier())
])
```

types, base and ensemble, base classifiers are the simplest classifiers, and ensemble classifiers are a combination of base classifiers, these classifiers are explained in the Section 2.1.

The base classifiers used are:

- **Logistic Regression**
- **Multilayer Perceptron**
- **Support Vector Machine**
- **Decision Tree**
- **Stochastic Gradient Descent**
- **K-Nearest Neighbors**
- **Linear Discriminant Analysis**
- **Gaussian Naive Bayes**
- **Quadratic Discriminant Analysis**

Based on the best results of the base classifiers, the ensemble classifiers were built. Some of these ensemble classifiers allow one to choose the base classifier to use, for these cases, the best base classifiers were used. The ensemble classifiers used are:

- **Random Forest**
- **AdaBoost**
- **Gradient Boosting**
- **Bagging**
- **Voting**
- **Stacking**

3.5.1. Comparison of Classifiers

To decide if a classifier is better than another, we must use some kind of metric, in Section 2.2 we explained the metrics that are most commonly used to evaluate the performance of a classifier. Now we have to choose the appropriate metrics for this problem, the most important thing for this problem is to minimize the number of false negatives, since a false negative means that a patient with cancer was classified as healthy, and this could have fatal consequences. To minimize the number of false negatives, we will use the *Recall*, but the problem of only taking into account the recall is that to

maximize it, the model could classify all samples as positive, to avoid this we will use the *Precision* since this will penalize the model for classifying a sample as positive if it was not. To combine these two metrics we will use the *F1 Score*, which is the harmonic mean of the precision and recall. We will also use the *F2 Score*, which is the weighted harmonic mean of the precision and recall, this will give more importance to the recall since we want to minimize the number of false negatives [**?**]. So, the evaluation metrics sorted by the importance that we will use are:

- **F1 Score**:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- **Recall**:

$$Recall = \frac{TP}{TP + FN}$$

- **F2 Score**:

$$F2 = 5 \times \frac{Precision \times Recall}{\frac{4}{Precision} + \frac{1}{Recall}}$$

- **Precision**:

$$Precision = \frac{TP}{TP + FP}$$

3.5.2. Optimization of Classifiers

To maximize these metrics in the classifiers, we have to optimize the hyperparameters of the classifiers. The hyperparameters are the parameters that are not learned by the model, and they are set before the training process, these can, for example, be the number of neighbors in the KNN classifier, the number of neurons in the MLP classifier, or the maximum depth of the DT classifier. To optimize these hyperparameters, we used the *RandomizedSearchCV* followed by the *GridSearchCV*. The reason for using the *RandomizedSearchCV* first is that can achieve similar results to the *GridSearchCV* but with a lower computational cost [**?**], this will allow us to explore a wider range of hyperparameters and have a range of values to use in the *GridSearchCV*. Then we used the *GridSearchCV* to find the best hyperparameters in the range of values found by the *RandomizedSearchCV*, a better explanation of these methods is given in Section 2.3.

These searches had a 5-fold CV (explained in Section 2.3) using the *F1 Score* as the metric to optimize. All the *RandomizedSearchCV* and *GridSearchCV* were done using the same random seed, this means that the split of the dataset was the same for all the classifiers, this ends up in a more reliable result and also allows the reproducibility of the results. This process was done to avoid overfitting and to have a more reliable result, of course, this process was only done in the training set, to avoid data leakage.

3.6. SHAP Implementation

Building and optimizing the classifiers is a crucial step and can be helpful in a clinical situation, but it can only get us so far. We also need to know why the model made a certain decision, for this, there is a branch of AI called Explainable Artificial Intelligence, this branch of AI is focused on making ML models more transparent, allowing clinical practitioners to understand why a certain decision was made [**?**]. To achieve this we will use the SHapley Additive exPlanations library. The reason for using SHAP is that it has been proven to achieve great explanations for ML models in the medical field [**?**]. A good explanation of SHAP is given in Section 2.4.1.

We will use SHAP to have a global understanding of the importance of the features in the models and a local understanding of how a certain decision was made. This will allow us to understand which features are the most important for a classifier, how sure the classifier was of its prediction, and which features determined if the sample was benign or malignant. This will also be done in the *Jupiter Notebook* to ensure reproducibility.

To be able to calculate the SHAP values, we need to create an *Explainer* object, this object is created with the classifier as a parameter, but in our case, we are not using just the classifier but pipelines, this means, that we will have to find a way to obtain the predicted probability of the pipeline. To do this, we will use the *predict_proba* method of the pipeline, this method will return the predicted probability of the sample being malignant no matter which classifier is used, with this, we will be able to create the *Explainer* object and calculate the SHAP values, all this process is show in the Code Snippet 3.2.

After we have the SHAP values, we will be able to use them to understand the importance of the features for the classifiers by plotting a graph with the SHAP values of all the samples and also one for each single feature which will show the behavior of that feature, meaning that we will be able to see if the higher the value of the feature the higher the probability of the sample being malignant or benign. We will also be able to use them to understand how a certain prediction was made by plotting the SHAP values of a certain sample. This will allow us to understand why a certain prediction was made and which features were the most important for that prediction, to do this we will plot the prediction of three cases where the classifier was sure the sample was benign, three where it was sure it benign, the three closest to the decision boundary and the misclassified ones. In the Code Snippet 3.3 we can see how to code the global feature importance, the behavior of a single feature, and the local explanation of a prediction for a certain classifier, this will be done for all the classifiers, features, and the aforementioned samples

Code 3.2: Obtaining the predicted probability of the pipeline and calculating the SHAP values

```
def model_predict_proba(data_asarray):
    data_asframe = pd.DataFrame(data_asarray, columns=X_train.columns)
    proba = lr_pipeline.predict_proba(data_asframe)
    return proba[:, 1]


# explainer object
explainer = shap.Explainer(model_predict_proba, X_train)

shap_values = explainer.shap_values(X_test)

explainer_x_test = explainer(X_test)
```

Code 3.3: Plotting the SHAP global features, single feature, and local explanation

```
shap.initjs()

# global feature importance
shap.plots.bar(explainer_x_test, max_display=25)
shap.plots.beeswarm(explainer_x_test, max_display=30, alpha=0.75)

# single feature importance
shap.dependence_plot(feature, shap_values, X_test, alpha=0.65, show=False)
ax = plt.gca()
x = ax.collections[0].get_offsets()[:, 0]
y = ax.collections[0].get_offsets()[:, 1]
regression_model = LinearRegression().fit(x.reshape(-1, 1), y)
x_range = np.linspace(x.min(), x.max(), 100)
y_range = regression_model.predict(x_range.reshape(-1, 1))
plt.plot(x_range, y_range, color='red')

# prediction explanation
shap.plots.waterfall(explainers_x_test[prob][idx], show=False)
```

3.7. Web Application Development

To make the results more accessible and visual, we have built a web application using *Django*. This framework was used because *Python* has been the main programming language used in the project and *Django* was the most familiar framework. For the front-end, we used HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript (JS). The web application is hosted in *Render* with the database in *Neon* this makes it accessible using this link, since the free hosting of *Render* is not always available, we have also made a video showing the web application in action, this video can be found in this link or if not the project can be run locally cloning the repository. The home page of the web application is shown in Figure 3.5, an example of the comparison of two classifiers is shown in Figure 3.6.

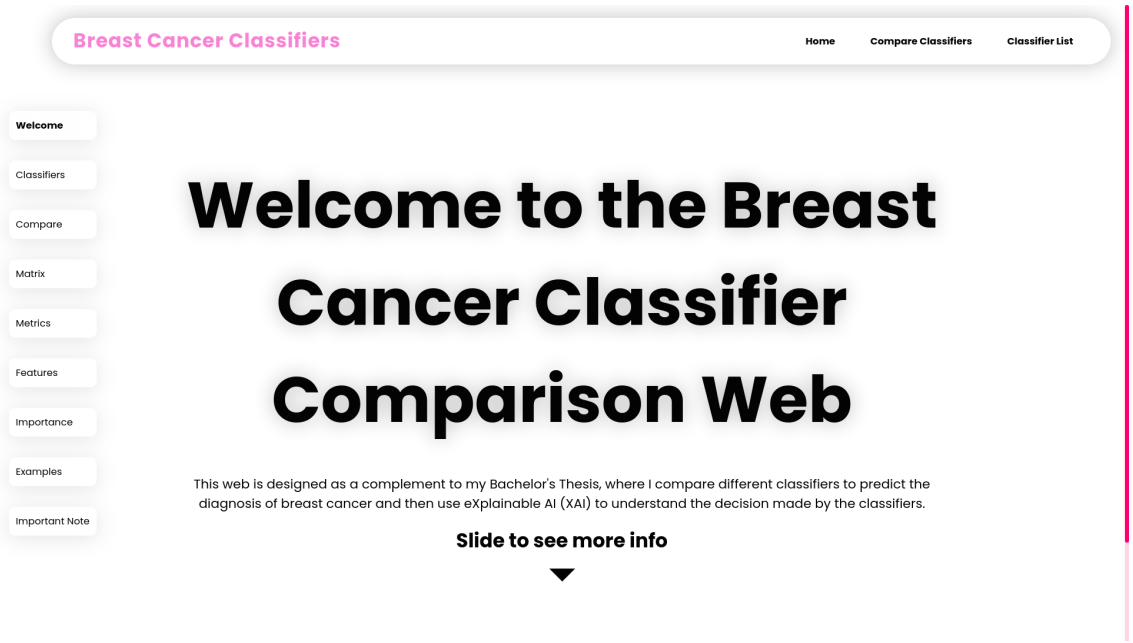

Figure 3.5: Home page of the web application

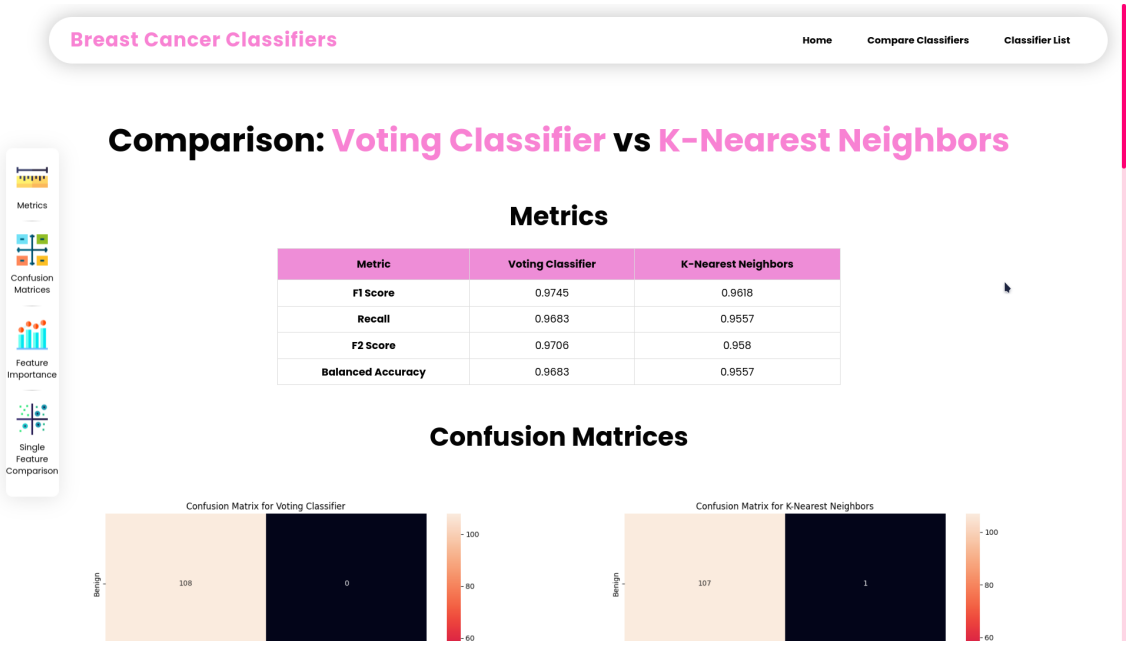

Figure 3.6: Comparison of two classifiers

4

Experiments and Results

4.1. Exploratory Data Analysis

4.2. Classifier Comparison

4.2.1. Base Classifiers Comparison

4.2.2. Ensemble Classifiers Comparison

4.2.3. Choosing the Best Classifier

4.3. SHAP Analysis

4.3.1. Global Interpretability

4.3.2. Local Interpretability

5

Conclusions and Future Work

5.1. Conclusions

5.2. Future Work

Appendices


UÁM
Universidad Autónoma
de Madrid