POLITECNICO DI TORINO

MSc in Data Science and Engineering

Mathematics in Machine Learning

Breast Cancer Dataset Analysis

Professors:
Francesco Vaccarino

Mauro Gasparini

Student:
Paola Privitera

Academic Year 2021-2022

Index

Introduction

The aim of this project is to exploit some supervised machine learning algorithms to discriminate benign from malignant lumps in breast cancer.
Breast cancer is the most common cancer worldwide and one of the major causes of death among women. It is a type of cancer that develops from breast tissue, whose cells grow out of control.
In general, the tumor can be classified as *benign*, when cells are close to normal in appearance, grow slowly and do not invade nearby tissues, and *malignant*, when cells can spread beyond the original tumor to other parts of the body, therefore being dangerous to health. Hence, it is crucial to analyze the shape, the size, and the texture of the cells to recognize benign cells from malignant ones.
Following this lead, in 1995, Dr. William H. Wolberg, W. Nick Street and Olvi L. Mangasarian of University of Wisconsin, used some interactive image processing techniques to contour the boundaries of the cell nuclei of several breast cells images, as shown in Figure 1, and then a computer vision system to extract some numerical characteristics to describe those boundaries. This resulted in the creation of the Breast Cancer Wisconsin (Diagnostic) Data Set that has been used for the development of the current project.

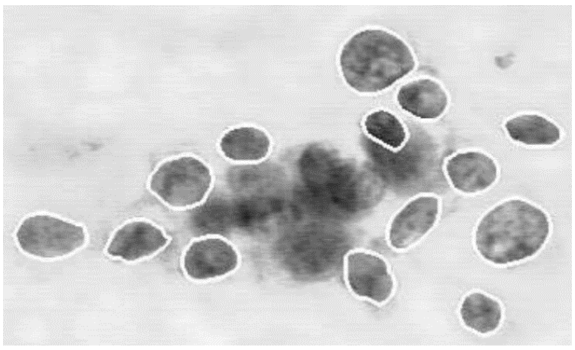


Figure 1: *Image of breast cell nuclei outlined by a curve-fitting program*

The analysis has been fully conducted with Python programming language, exploiting several machine learning and statistical frameworks available such as *numpy*, *pandas*, *scikit-learn*, *imblearn* together with some data visualization libraries, *matplotlib* and *seaborn*.
The project's code is available at the following link: https://github.com/paolaprivitera/Breast-Cancer-Dataset-Analysis/blob/main/Breast_Cancer_Dataset_Analysis_MML.ipynb

1 Exploratory Data Analysis

1.1 Data Description

The dataset is publicly available at the UCI machine learning repository [1]. It has been created basing on the analysis of 569 images of the cell nuclei, from which the following ten real-valued features have been extracted:

a) *radius*: mean of distances from center to points on the perimeter;
b) *texture*: standard deviation of gray-scale intensities in the component pixels;
c) *perimeter* (nuclear perimeter): sum of the distances between consecutive boundary points;
d) *area* (nuclear area): number of pixels on the interior of the boundary plus one-half of the pixels on the perimeter, to correct the error caused by digitalization;
e) *smoothness*: local variation in radius lengths;
f) *compactness*: $\frac{perimeter^2}{area - 1.0}$;
g) *concavity*: severity of concave portions of the contour;
h) *concave points*: number of contour concavities;
i) *symmetry*: difference in length between pairs of line segments perpendicular to the major axis of the contour of the cell nucleus;
j) *fractal dimension*: "coastline approximation" – 1. In other terms, it is approximated using the "coastline approximation" described by Mandelbrot [3].

The mean (M), standard error (SE) and worst (W) or largest (mean of the three worst/largest values) of each of the previously described features were computed, resulting in 30 continuous features describing 569 instances.

Moreover, each instance of the dataset is characterized by an *ID number*, which has not been considered for the purpose of the analysis, and by the *Diagnosis* column, that refers to the target label and assumes value "0" if the cancer is malignant or value "1" if it is benign, translating into a binary classification problem.

1.2 Data Exploration

The first phase consisted in exploring the data to acquire a thorough understanding of them and their distribution.

1.2.1 Duplicates and Missing Values

After a first analysis, it has been proved that the dataset does not contain **duplicates** or **missing values**, meaning that there are not instances to drop or values to fill.

1.2.2 Class Distribution

The class distribution is shown in the following figure.

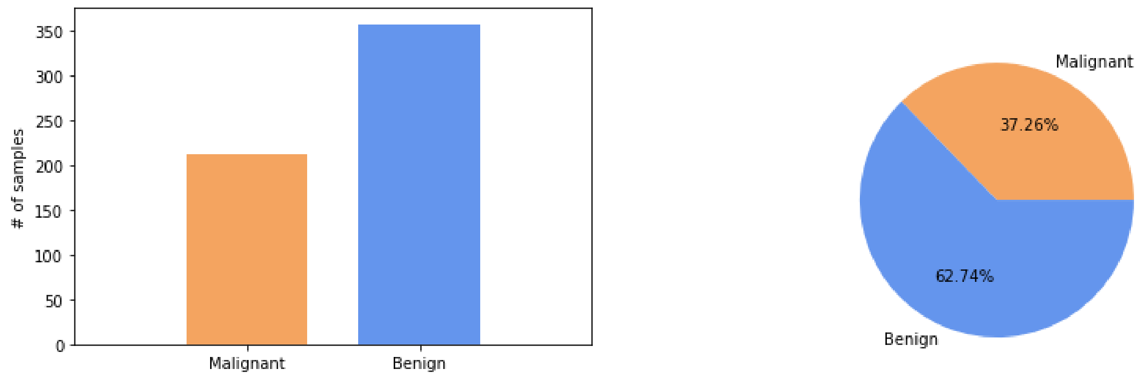


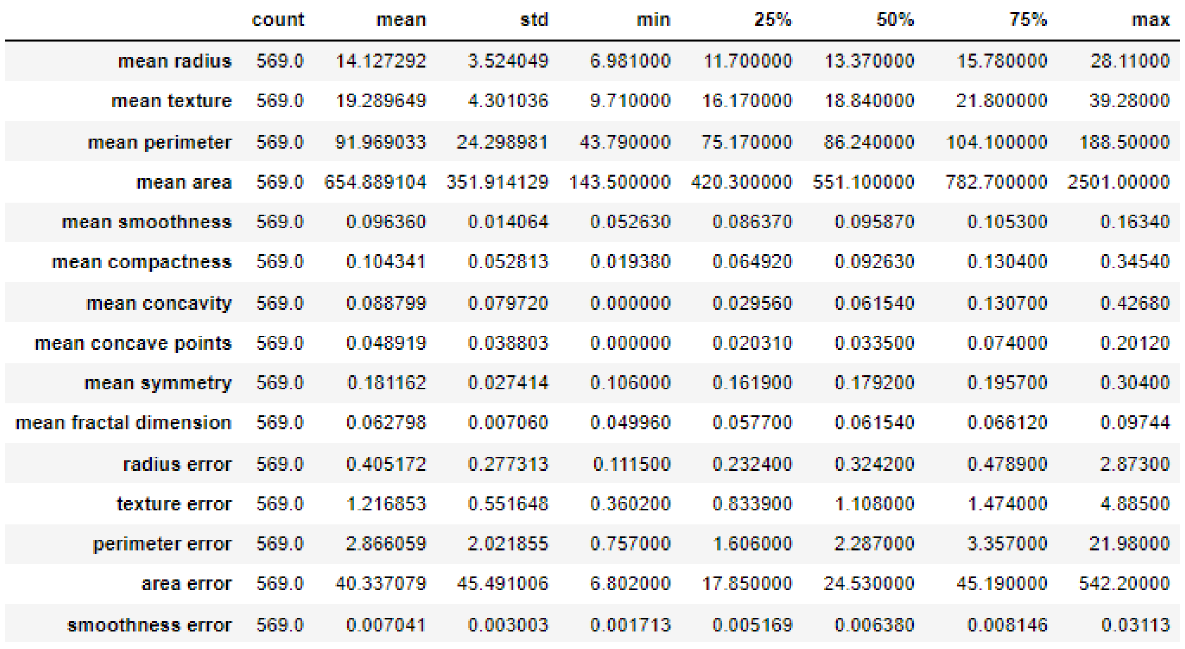
Figure 2: *Class distribution with histogram (left) and pie chart (right)*

As can be seen from the graphs, the dataset is relatively **unbalanced**, indeed 212 instances out of 569 refer to malignant cancer, accounting for 37.26% of the total, while the 62.74% are benign tumors.

1.2.3 Statistics

A further step of the analysis involved the calculation of some statistical data of the features in the dataset. In particular, the following **statistics** are reported in Table 1:

- *count*: number of samples for each feature;
- *mean*: average value for each feature;
- *std*: standard deviation from the mean value;
- *min*: minimum value;
- *25%*: lower quartile of the distribution;
- *50%*: median quartile;
- *75%*: upper quartile;
- *max*: maximum value.



	count	mean	std	min	25%	50%	75%	max
mean radius	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
mean texture	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
mean perimeter	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
mean area	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
mean smoothness	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
mean compactness	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
mean concavity	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
mean concave points	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120
mean symmetry	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30400
mean fractal dimension	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09744
radius error	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87300
texture error	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88500
perimeter error	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98000
area error	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20000
smoothness error	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03113

	count	mean	std	min	25%	50%	75%	max
compactness error	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13540
concavity error	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39600
concave points error	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05279
symmetry error	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.07895
fractal dimension error	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.02984
worst radius	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.04000
worst texture	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.54000
worst perimeter	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.20000
worst area	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.00000
worst smoothness	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.146000	0.22260
worst compactness	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.339100	1.05800
worst concavity	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.382900	1.25200
worst concave points	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.161400	0.29100
worst symmetry	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.317900	0.66380
worst fractal dimension	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.092080	0.20750

Table 1: *Statistics*

This summary table allowed to gain insight into the underlying distribution of the variables: in particular, it has been noticed that features' values lie in very different ranges, making hardly impossible a comparison among them and suggesting that a standard scaling is needed.

1.2.4 Boxplots

It has been possible to confirm the previous observation by plotting the boxplots, which are used to further visualize the distribution of the data. The following figures compare the boxplot of all features without standardization and the one of the standardized features.

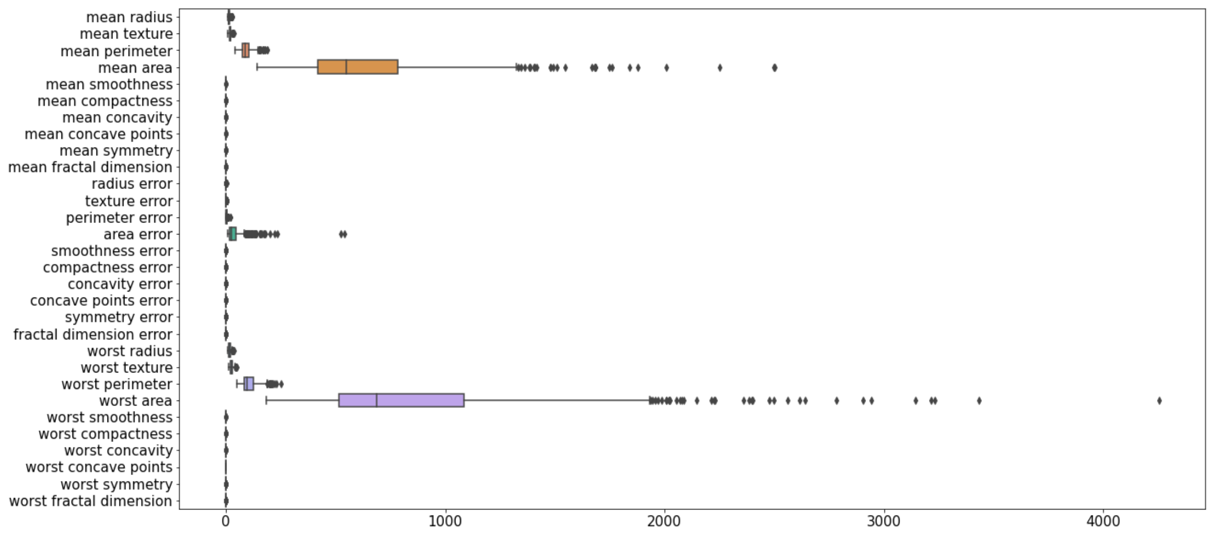


Figure 3: *Boxplot with initial data*

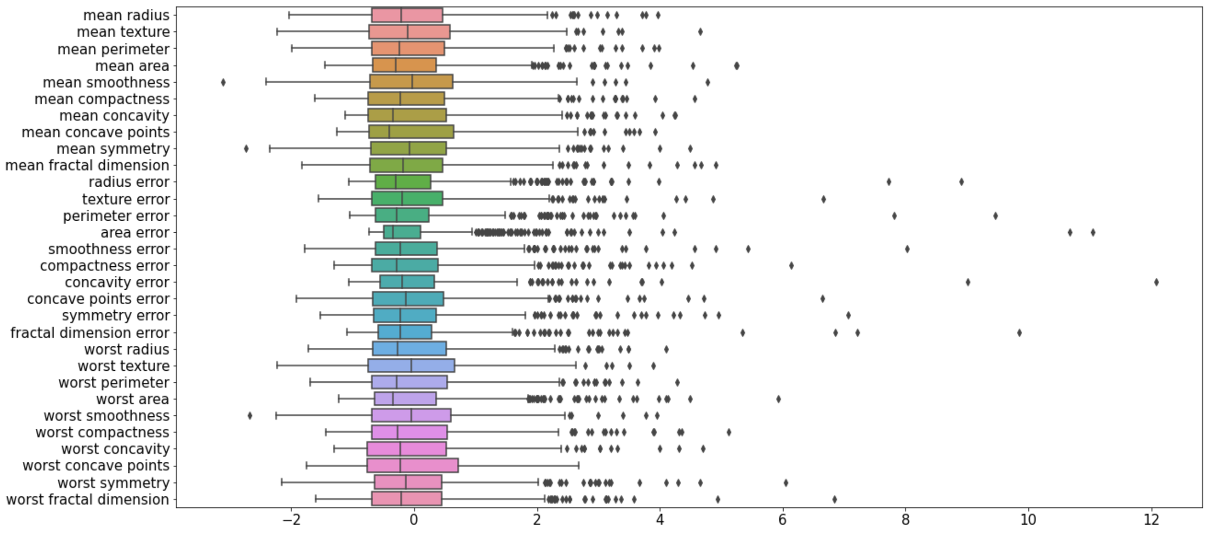


Figure 4: *Boxplot with standardized data*

A **boxplot** is a graphical representation of the five-number summary of the data consisting of the minimum, maximum, and the first, second and third quartiles. A quartile is a statistical term that describes a division of observation into four defined intervals based on the values of the data.

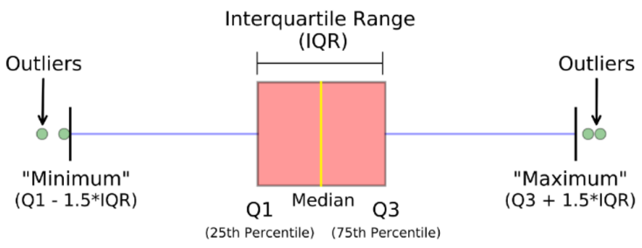


Figure 5: *Boxplot description*

The box is drawn from the first quartile (Q_1) to the third quartile (Q_3). The line inside the box indicates the location of the median, which is the point at which exactly half of the data lies below and above the central value.
Quartiles are used to calculate the interquartile range as $\mathrm{IQR} = Q_3 - Q_1$, which corresponds to the size of the box and is a measure of variability around the median. The left whisker extends to the largest of the minimum of the data ($Q_1 - 1.5\ \mathrm{IQR}$) and the right whisker extends to the smallest of the maximum of the data ($Q_3 + 1.5\ \mathrm{IQR}$). Any data point outside the whiskers is indicated by a small dot, showing a suspicious or deviant point (outlier), that sometimes could lead to errors and misclassification. For this reason, boxplots are useful tools to verify the presence of **outliers**.
As shown in Figure 4, many outliers have been detected but, due to the lack of domain knowledge and the limited number of observations, no samples have been discarded.

1.2.5 Histograms

A histogram is a common graphical representation of the distribution of a quantitative feature. To construct a histogram, the first step is to divide the range of values into a series of bins (on the x-axis) and then to draw bars to show the number of observations that fall in each bin (on the y-axis).

In the current project, a histogram has been plotted for each combination of feature and class. By doing so, it has been possible to observe how the distribution of the features varies as class changes, i.e. which features are most helpful in predicting malignant or benign cancer. In particular, the more the two distributions overlap, the more that specific feature is not relevant in differentiating between the classes.

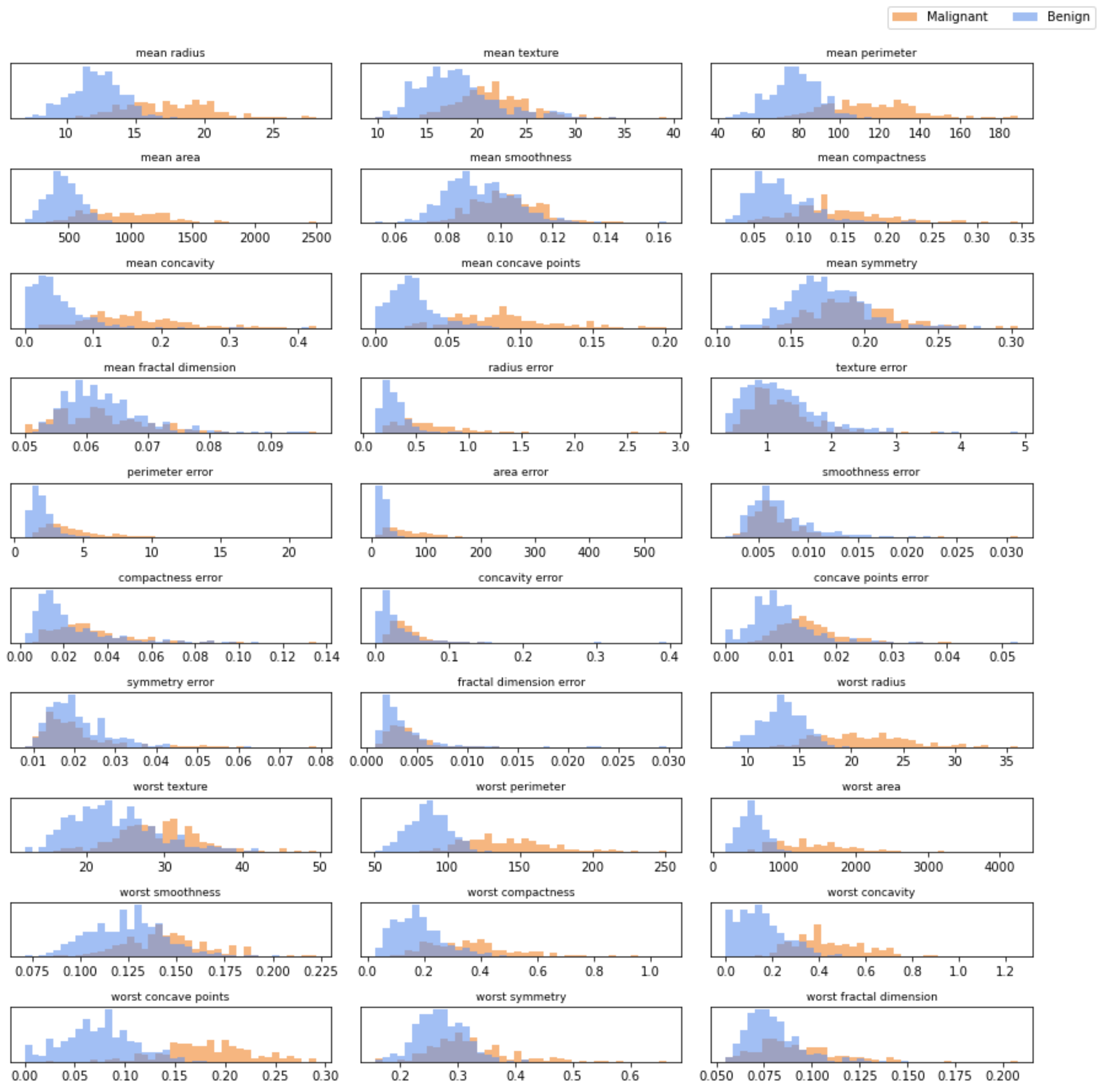


Figure 6: *Histograms for features' distribution according to the class*

According to the previous reasoning, features like “texture error” and “smoothness error” would be less helpful in predicting a diagnosis since their distribution is almost the same for the two classes. On the other hand, examples of features that could discriminate well between malignant and benign tumors could be “mean radius” and “worst concave points”. Indeed, by considering the former, it is noticeable that the mean radius of benign cells is characterized by smaller values, confirming that malignant cells are generally bigger than benign ones.

1.2.6 Correlation Matrix

To have a deeper insight in the understanding of the data and especially in the relationships between the features, the correlation matrix has been exploited.
A **correlation matrix** is a table that shows, in each cell, the correlation coefficients between pairs of attributes to measure how strong their relationship is.
Spotting correlation among features is crucial because it could lead to a decline in the performances of some classification algorithms which assume that the predictors are all independent. Furthermore, the same information may be encoded using less and not redundant attributes by exploiting some dimensionality reduction techniques, leading to simpler and less computationally expensive models.

A **heatmap** has been used to better visualize the values of the correlation matrix. Indeed, it is a very intuitive visualization tool because it is based on a color codification according to the correlation statistics: blue is for negative correlations, red is for positive relationships, while the intensity of the color is related to the strength of the correlation between variables.
Since the matrix is symmetrical and because of the large number of features, only the heatmap of the lower diagonal matrix has been reported. Moreover, the correlation coefficients in the diagonal are always equal to 1, meaning that each variable always perfectly correlates with itself.

By looking at the heatmap of Figure 7, it is possible to spot some pairs of strongly correlated variables. As one can easily expect, pairs such as “mean perimeter” and “mean radius”, “mean area” and “mean radius”, “mean area” and “mean perimeter” are characterized by high values of correlation coefficient. The same is true for “perimeter error” and “radius error”, “area error” and “radius error”, “area error” and “perimeter error”. Moreover, the mean value and the worst value of the same feature have a strong relationship. Hence, selecting only one of the two values could help in reducing data redundancy and dimensionality.

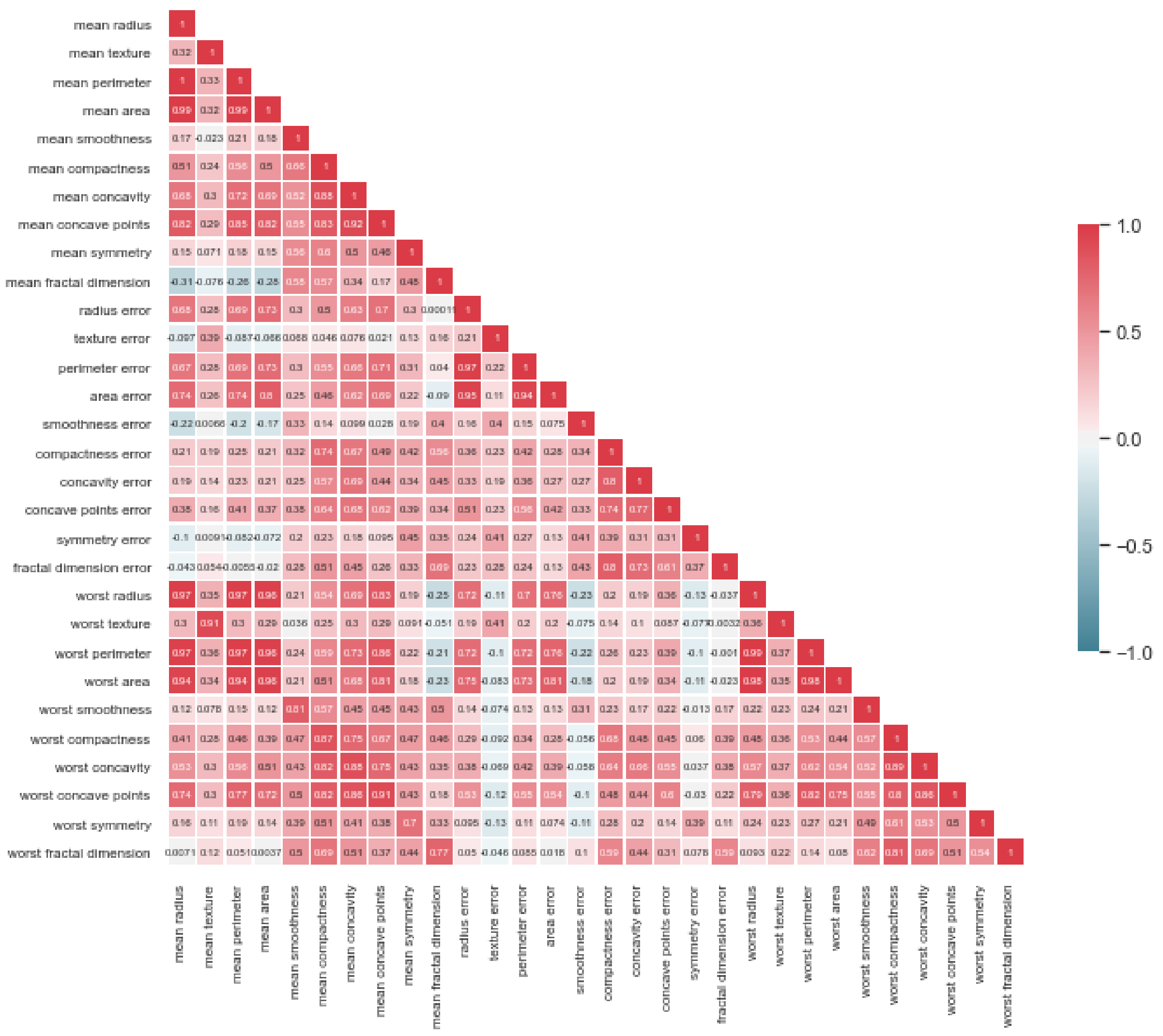


Figure 7: *Heatmap of the correlation matrix*

In the present work, the **Pearson's correlation coefficient** has been used to measure the linear correlation between the pairs of variables. Given two vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$, the Pearson product-moment correlation coefficient (PPMCC) is defined as:

$$\rho_{\boldsymbol{X},\boldsymbol{Y}} = \frac{Cov(\boldsymbol{X},\boldsymbol{Y})}{\sigma_{\boldsymbol{X}}\sigma_{\boldsymbol{Y}}} \in [-1,1]$$

where the numerator represents the covariance of the two variables, while the denominator corresponds to the product of the two standard deviations, each one indicating a measure of the dispersion of data from its average. On the other hand, the covariance is a measure of how the two variables change together but its magnitude is unbounded, so it is difficult to interpret. Hence, by normalizing it by the product of the two standard deviations, it is possible to describe how one variable moves in relation to the other. In this way, the correlation coefficient returns a value between -1 and 1:

- If the correlation coefficient is -1, it indicates a strong negative relationship between the two variables, meaning that for every positive increase in one variable, there is a negative decrease of a fixed proportion in the other.
- If the correlation coefficient is 0, it indicates no relationship. However, it is important to underline that the Pearson's correlation coefficient is only able to capture linear trends, hence it might lead to a value of 0 for strongly non-linearly correlated variables (e.g., quadratic trend).
  Figure 7 shows the scatter plot of two non-linearly correlated features ($\rho_{texture\_error,mean\_concave\_points} = 0.021$):

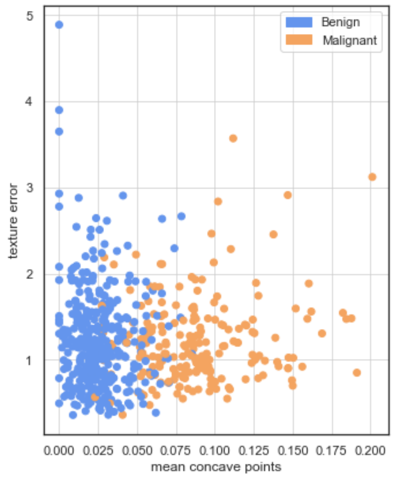


Figure 8: *Scatter plot of non-linearly correlated features*

- If the correlation coefficient is 1, it means that there is a strong positive relationship between the two variables. In other words, for every positive increase in one variable, there is a positive increase of a fixed proportion in the other.
  As an example, the scatter plot of two variables ("mean perimeter" and "mean radius") whose Pearson's correlation coefficient is equal to 1 has been reported below:

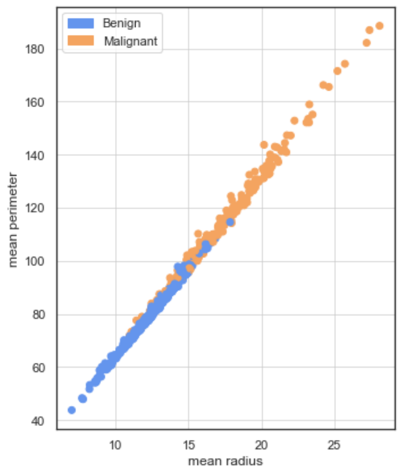


Figure 9: *Scatter plot of highly correlated features*

2 Data Preprocessing

Considering the observations collected in the exploratory data analysis, some data preprocessing techniques have been applied to prepare the data to feed the machine learning models in an appropriate way.

2.1 Partitioning the dataset

The general procedure to determine a model's goodness is to partition the dataset into specific subsets: training, validation, and test. The former is necessary to allow the model to understand and "learn" from the underlying data, derive their properties and be able to establish criteria for the classification. The validation set is necessary to test whether the training phase has been successful and determine which are the best model hyperparameters to achieve the maximum values for the evaluation metrics. Finally, the test set (composed by "fresh data") aims at estimating the actual goodness of the model.

Firstly, the dataset has been split into **train** and **test sets** adopting a random partitioning approach, keeping the 3:1 proportion. In particular, the split has been generated by setting a random seed that has been kept the same for all the methods, in order to provide consistent results and comparisons among the different classifiers. Moreover, given the unbalance of the dataset, through the stratification it has been possible to preserve the original labels' distribution, so that the proportion of benign and malignant samples in each of the new sets would be approximately the same as the original dataset.

Secondly, the **Stratified K-Fold Cross Validation** method has been adopted, useful in case of unbalanced data and small datasets. This approach involves randomly dividing the training set of observations into k non-overlapping groups, or folds, of approximately equal size, in a stratified manner. The first fold is treated as a **validation set**, and the method is fit on the remaining $k-1$ folds. The procedure is then repeated $k-1$ times, allowing each fold to be used as validation set exactly once. Finally, the results can be averaged to produce a single estimation. In the present work, a value of $k = 5$ has been chosen and the next picture shows a visual representation of the method:

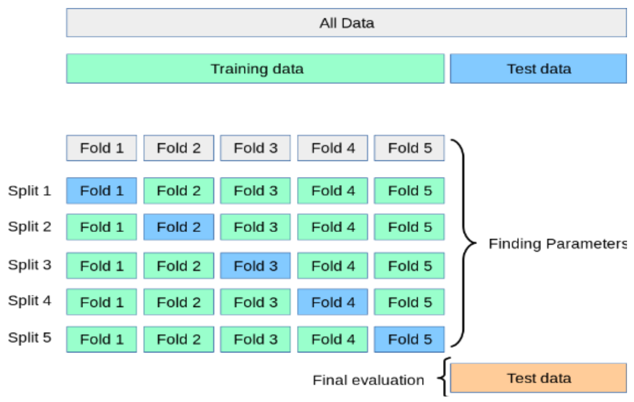


Figure 10: *Schema of dataset splitting*

It is important to mention that, in case of standardization and dimensionality reduction techniques (addressed in the following paragraphs), the statistics have been computed on the training set only, and then all sets have been transformed according to them. This is very important as it avoids leaking any information from the test set into the training process, which would be formally incorrect.
Also resampling techniques have been applied only on the training set, leaving untouched validation and test sets, in order to respect the real data distribution.

2.2 Standardization

Standardization has been used to bring down all the features to a common scale without distorting the differences in the range of the values. It is one of the most important data preprocessing steps in machine learning. Indeed, algorithms that compute distance between the features are biased towards numerically larger values if the data is not scaled. Also, feature scaling helps machine learning algorithms in training and converging faster. Moreover, some dimensionality reduction techniques such as PCA, which will be addressed in paragraph 2.3.2, rely on zero-centered data.

The process consists in rescaling the attributes so that they have a mean value of 0 and variance 1[2]. In details, standardization transforms each value x of a feature $\boldsymbol{X}$ as follows, independently for each column:

$$x' = \frac{x - \mu_{\boldsymbol{X}}}{\sigma_{\boldsymbol{X}}}$$

where $\mu_{\boldsymbol{X}}$ is the sample mean and $\sigma_{\boldsymbol{X}}$ is the sample standard deviation of feature $\boldsymbol{X}$.

2.3 Dimensionality Reduction

Dimensionality reduction refers to techniques that reduce the number of input variables in the dataset. The higher the number of features, the more difficult is to model them, inevitably increasing the computational complexity and impacting on the performance of machine learning algorithms. This problem is generally known as **curse of dimensionality**: it occurs when the considered space has a very high number of dimensions and its volume increases so fast that available data become sparse, making differences in distances less significant. Moreover, many input dimensions are often related to many parameters and complex machine learning models, which are likely to overfit the training dataset and therefore may not be able to generalize well on new data.
To tackle these issues, the feature selection technique and the principal component analysis have been exploited.

2.3.1 Feature selection

As it turned out from the analysis of the correlation matrix in paragraph 1.2.6, some features are strongly linearly correlated and therefore they may contain redundant information. For this reason, the pairs of variables characterized by a Pearson's correlation coefficient greater than or equal to 0.94, have been analyzed. For each pair, it has been decided to maintain only the variable whose relationship with the target variable was stronger (i.e. the one with the highest correlation index) and to drop the other. In case the feature selection step is applied, the following features are removed: "perimeter error", "mean perimeter", "mean radius", "worst area", "worst radius", "mean area", "area error".

2.3.2 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical procedure used to reduce the dimensionality of large datasets, by transforming a large set of variables into a smaller one, while preserving as much information as possible. The aim is to find the optimal values of the coefficient of each feature that maximizes the variance of the linear combination of all the features. To serve the purpose, it uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.
The steps involved in the PCA are the following:

1. Center the data around the origin by subtracting all the values of each variable by its corresponding mean and put them into a matrix D. This will help later in rotating the data.
   Note that, in the current work, data are already centered because they have been previously normalized.
2. Calculate the covariance matrix A, which is a dxd symmetric matrix (where d is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional dataset with 3 variables x, y, and z, the covariance matrix is a $3x3$ matrix of this form:

$Cov(x,x)$	$Cov(x,y)$	$Cov(x,z)$
$Cov(y,x)$	$Cov(y,y)$	$Cov(y,z)$
$Cov(z,x)$	$Cov(z,y)$	$Cov(z,z)$

Table 2: *Example of covariance matrix*

The main diagonal includes the covariance of a variable with itself which corresponds to its variance. The sample variance of a generic variable x is computed as follows:

$$var(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

where n is the number of samples, while the covariance of two variables x and y is calculated as:

$$cov(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

3. Compute the eigenvalues of the covariance matrix by solving the following equation in λ with d degrees:
$$det|A - \lambda I| = 0$$
where I is the identity matrix, which has the same number of rows and columns as the covariance matrix A.
4. Compute the eigenvector v_i that correspond to the eigenvalue λ_i by using as general equation:
$$A \cdot v_i = \lambda_i \cdot v_i$$
where $i \in [1,\ d]$.
To find the other eigenvectors, do the same calculations based on the other eigenvalues. Being the covariance matrix a symmetric positive semidefinite matrix, the eigenvectors will be orthogonal and therefore uncorrelated with each other (constituting an orthonormal basis).
The eigenvectors of the covariance matrix are actually the **Principal Components**, i.e. the directions of the axes where there is the most variance, whereas the eigenvalues are simply the coefficients attached to the eigenvectors, and they give the amount of variance carried in each Principal Component.
5. Order the eigenvectors based on their corresponding eigenvalues from the highest to the lowest, and put them into a matrix V, where the first column corresponds to the eigenvector with the largest eigenvalue (first eigenvector), the second column represents the second eigenvector, and so on. By doing so, the principal components are ranked in order of significance.
6. Recast the data along the principal components' axes by multiplying the matrix D (original centered data) by matrix V (eigenvectors sorted by eigenvalues). By doing so, a new matrix with the transformed data is obtained and it represents the original centered data in the principal component space.
As an example, by considering the first two principal components, this can be seen as rotating the data clockwise until the two eigenvectors point in the same direction as the x-axis and y-axis of the plot.

An important thing to realize is that the principal components lack **interpretability** since they are constructed as linear combination of the initial variables. Therefore, it can be useful to display a **biplot** to see if any of the original variables, which instead have a semantic meaning, is well explained by the principal components.
In the graph the axes correspond to the first and the second principal component that, by the way they have been constructed, are orthogonal to each other. It displays both the component scores and the component loadings, which are respectively the data points and the variables projected into the principal component space. In details, the loadings are represented with

arrows and correspond to the elements of the eigenvector, whereas the scores are the projected data points. The following properties hold:

- The cosine of the angle between any two arrows shows the correlation between those variables. Vectors that point in similar directions are highly correlated, while perpendicular ones are uncorrelated. A small angle implies positive correlation, while a large one means negative correlation.
- The cosine of the angle between a vector and an axis indicates the correlation of the corresponding variable with the principal component. For example, variables such as “worst concave points”, “worst concavity”, “mean radius”, “worst smoothness” have a strong positive relation with PC1, while “mean area”, “mean perimeter” and “smoothness error” are highly correlated with PC2.
- The lengths of arrows in the plot are directly proportional to the variability included in the two components. For example, a very short arrow would indicate that the first two components contain almost no information about that feature.
- Close points represent observations with similar values.

Moreover, by choosing the first two principal components, it is possible to make sure the direction of maximum variance is useful for classification purposes. In this case, the two classes from the input dataset are well separated.

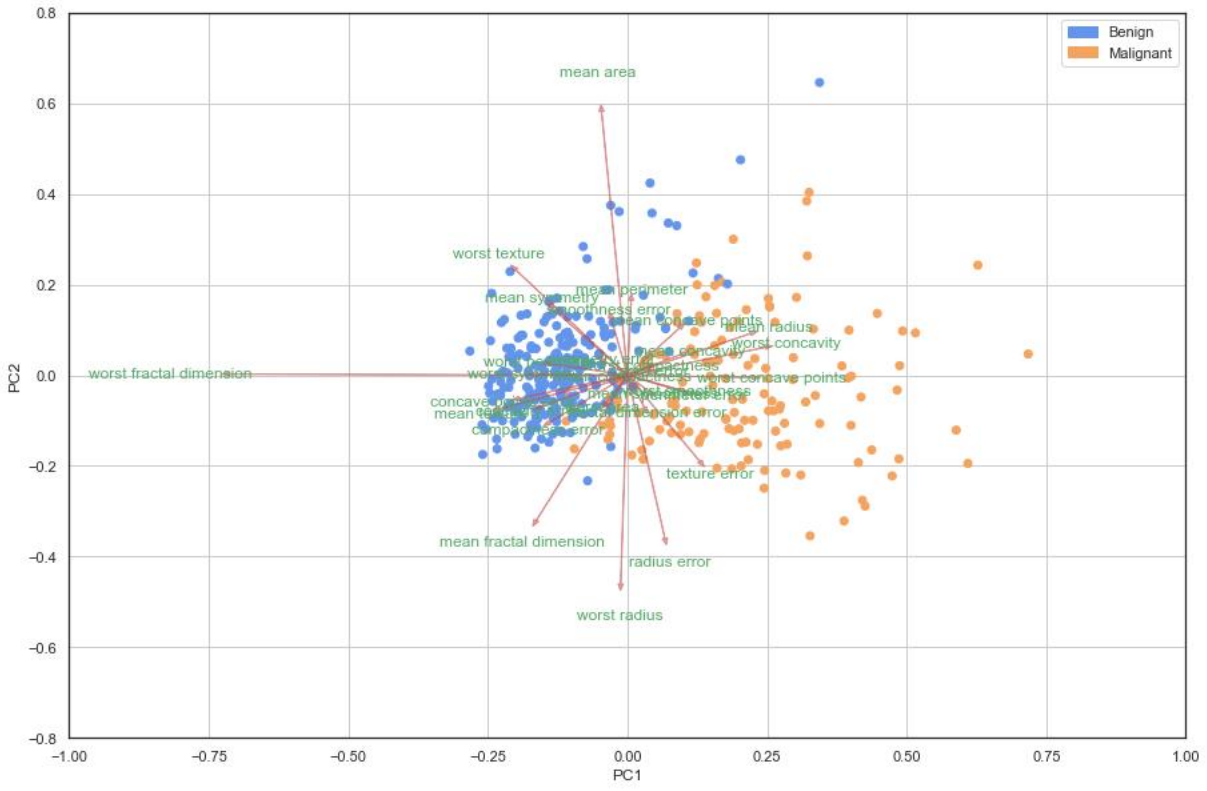


Figure 11: *Biplot*

To understand the strength of each component, it is important to know the **proportion of variance explained** (PVE) by each one. In particular, by looking at the variance explained by each principal component (i.e. scores) and comparing it with the total variance of the dataset, it is possible to select in an empirical way only a subset of PCs to reduce the dimensionality of the dataset, while preserving most of the information. Indeed, as can be seen from the graph, only the first principal component retains 45% of the total variance. For the analysis, the first 7 PCs have been kept, being able to explain about 91% of the total variance.

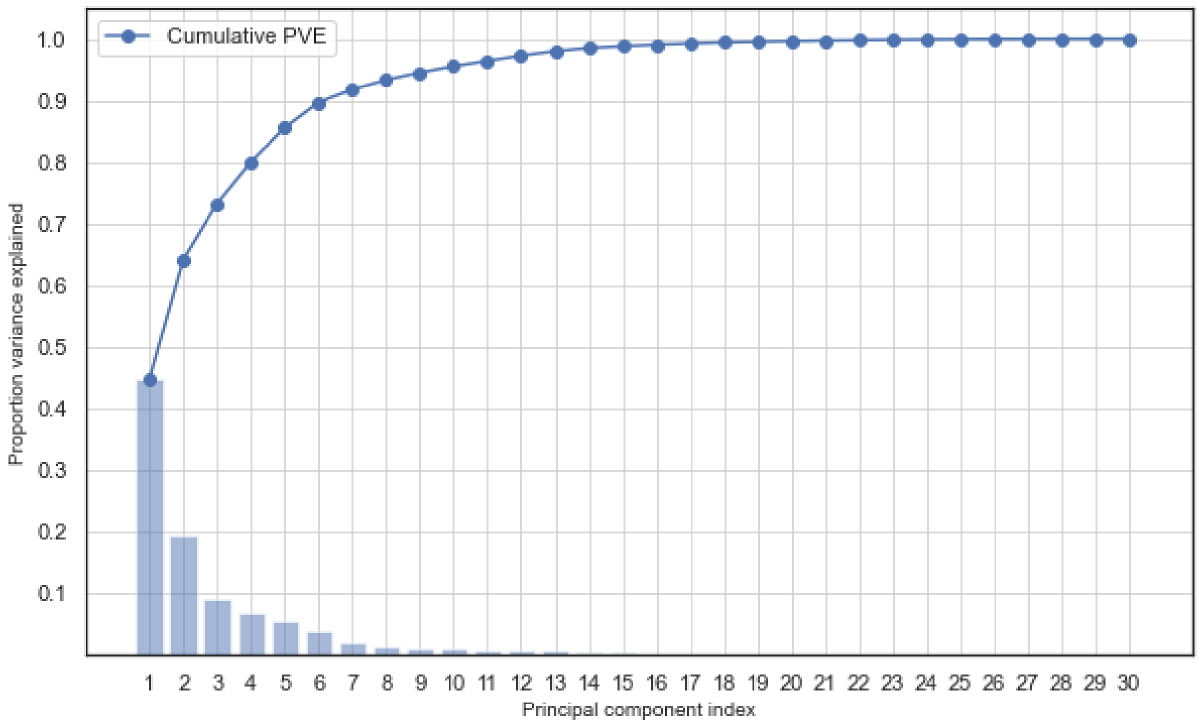


Figure 12: *Scree plot*

2.4 Dealing with unbalanced datasets

Unbalanced datasets constitute a challenge for predictive modelling as most of the machine learning algorithms used for classification are designed around the assumption of an equal number of samples for each class. The training process with an unbalanced dataset might be biased towards a certain class, meaning that models would have poor predictive performance, especially for the minority class that, in the present work, corresponds to malignant tumors (37.26% of the observations).

The most preferable solution to this problem would be collecting more real data to get a balanced dataset, but this is not always possible, especially in the medical field. Hence, there exist two main techniques to overcome it:

- **Undersampling**, consisting in removing samples from the majority class until the samples of the two classes become equal in number. However, by performing undersampling some relevant information would be lost, which is not desirable, especially in this case given the small size of the dataset.

- **Oversampling**, consisting in adding more samples from the minority class until the same proportion of classes is obtained. This can be done by duplicating some random samples or by generating synthetic data. Since the first approach is more prone to overfitting, it has been decided to proceed with the second one.

2.4.1 Synthetic Minority Oversampling Technique

Synthetic Minority Oversampling Technique (SMOTE) is an oversampling technique that creates synthetic minority class data points to balance the dataset. It works using a k-nearest neighbor algorithm to create new synthetic samples. The steps of the algorithm are:

1. Identify the minority class vector.
2. Decide the number of nearest neighbors to consider (in this case $k = 5$).
3. Compute a line between the minority data points and any of its neighbors and place a synthetic point (generated as a convex combination of the considered instances).
4. Repeat step 3 for all minority data points and their k neighbors, till the data is balanced.

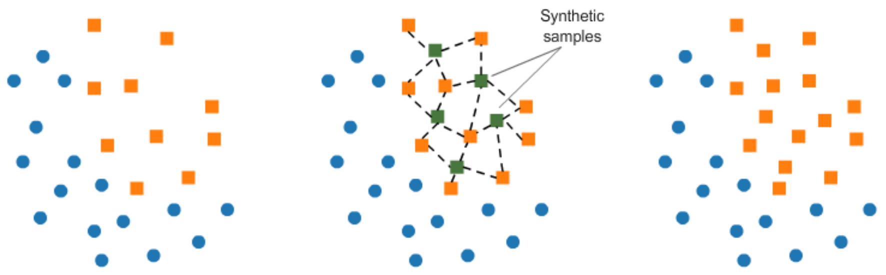


Figure 13: *Graphical representation of SMOTE*

It is important to remark that a synthetic sample should never be used as a test sample, since it has technically never appeared in the real world and hence not formally belonging to the target distribution.

The following picture shows a comparison of the training set before and after the application of SMOTE, by plotting the interaction of two not correlated features. It can be easily noticed that, after the application of the oversampling technique, there is an increase in the minority class samples.

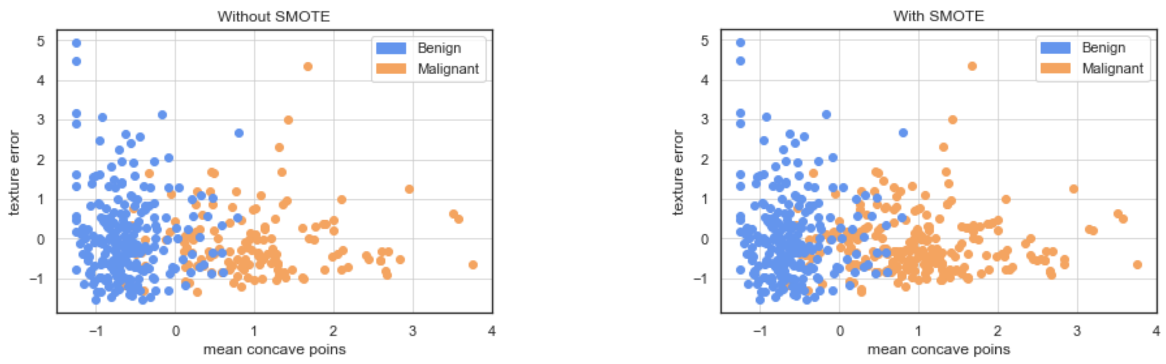
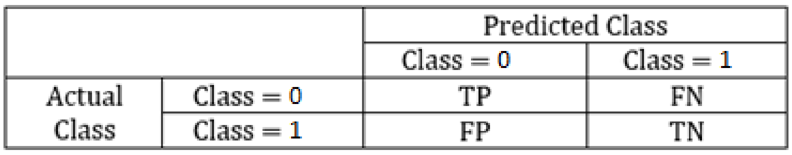


Figure 14: *Training set before and after SMOTE*

3 Evaluation metrics

Evaluation metrics play a crucial role in both guiding the classifier modeling and assessing the classification performance. For classification problems, metrics involve comparing the expected class label to the predicted one. To serve the purpose, the **confusion matrix** comes in handy because it summarizes the number of correctly and incorrectly predicted instances.



		Predicted Class	
		Class = 0	Class = 1
Actual Class	Class = 0	TP	FN
	Class = 1	FP	TN

Figure 15: *Confusion matrix*

- TP: samples for which the prediction is positive and the true label is positive;
- FP: samples for which the prediction is positive but the true label is negative;
- TN: samples for which the prediction is negative and the true label is negative;
- FN: samples for which the prediction is negative but the true label is positive.

In the current project, positive corresponds to *malignant* tumors and negative to *benign* ones. Starting from the previous values, it is possible to define the following metrics:

- **Accuracy**: it represents the fraction of predictions correctly classified by the model.

  $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\#\ of\ correctly\ classified\ samples}{Total\ number\ of\ samples}$$

  It is the most popular metric for model evaluation, but it is not reliable when dealing with significantly unbalanced datasets because it is not able to distinguish between the numbers of correctly classified samples of different classes. Indeed, predicting the majority class would always lead to a high value, even if the predictions of the minority class are all wrong.

- **Recall**: it indicates the percentage of correctly labelled positive instances out of all instances that are actually positive.

  $$Recall = \frac{TP}{TP + FN} = \frac{\#\ of\ samples\ correctly\ assigned\ to\ class\ C}{\#\ of\ samples\ actually\ belonging\ to\ C}$$

  In other words, out of everything that belongs to the *malignant* class, it tells how many of them the model was able to capture.

- **Precision**: it determines the percentage of correctly labelled positive instances out of all positive labelled instances.

  $$Precision = \frac{TP}{TP + FP} = \frac{\#\ of\ samples\ correctly\ assigned\ to\ class\ C}{\#\ of\ samples\ assigned\ to\ C}$$

Out of everything that the model labelled as *malignant*, it tells how many samples were actually belonging to that class.

- **F1-score**: it is the harmonic mean of precision and recall.

$$F1\_score = \frac{2 \cdot recall \cdot precision}{recall + precision}$$

As previously explained, accuracy is not the best choice in case of unbalanced datasets. On the other hand, the recall metric can be useful, especially in the medical field, because a high value would indicate a good ability in correctly identifying samples as malignant. However, it is advisable to pay attention also to the precision metric to do not mistakenly consider benign tumors as malignant ones. For this reason, F1-score has been used as the main reference metric, being able to incorporate both information.

4 Classification Algorithms

4.1 Random Forest

The first classification algorithm that has been exploited is the Random Forest, which is an ensemble of decision trees, whose goal is to build a model able to predict the class label by learning simple decision rules inferred from training data.
To fully understand the structure of a random forest, it is important to explain how a decision tree works.

The original idea behind a **decision tree** is to make a partition of the space into decision regions in a way that a certain error (e.g., misclassification error or classification error rate) is minimized. The number of splits is proportional to the number of distinct values of the predictors, meaning that there are too many possible solutions to the problem, making it computationally infeasible (NP-hard). For this reason, a **top-down**, **greedy** approach is used to make the best split locally at each step of the tree-building process, starting from the top of the tree (root node) and then successively splitting the predictor space further down on the tree.
In general, every node of the tree specifies a test involving an attribute, and every branch descending from a node matches one of the possible outcomes of the test, until reaching a leaf node that is associated to the class label.
At each step, the attributes that constitute the nodes are selected in a way that partition the learning set into subsets that are as "**pure**" as possible (i.e. homogeneous in terms of class label). The following measures have been used to calculate the degree of impurity of an attribute:

$$Gini(t) = \sum_{k=1}^{C} p_t(k)\big(1 - p_t(k)\big)$$

$$Entropy(t) = -\sum_{k=1}^{C} p_t(k)\ log_2(p_t(k))$$

where $p_t(k)$ is the frequency of class k at node t and C is the total number of classes. The lower the values of the indexes, the less impure the node is. Therefore, at each step, the feature with the lowest impurity value is chosen as node.

Decision trees are easy to understand and to implement but their main weakness is that they are unstable learners because they are extremely dependent on the characteristics of the training set (their output can change dramatically when the training is slightly changed). Random forest makes use of this limitation to promote diversity by combining the contribution of several decision trees to take a single decision.

The first step is the creation of *B* **bootstrapped datasets**, characterized by the same size of the training set and obtained by randomly sampling with replacement the original training set (therefore allowing for multiple copies of the same observation). On each bootstrapped dataset, a decision tree is built, for a total of *B* trees.

When building these decision trees, at each split of each tree, only a **random subset of features** is chosen as split candidates from the full set of predictors. The size of the subset is fixed, and it is typically equal to the square root of the total number of features. Introducing this source of randomness with a new subset of features at each split, allows in having decorrelated trees and therefore a more robust final prediction (low variance).

Finally, the classification is done by **majority voting** among the predictions of all the trees. In particular, for each test observation, the overall prediction is the most commonly occurring class among the *B* predictions.

The following picture shows a graphical schema of the algorithm.

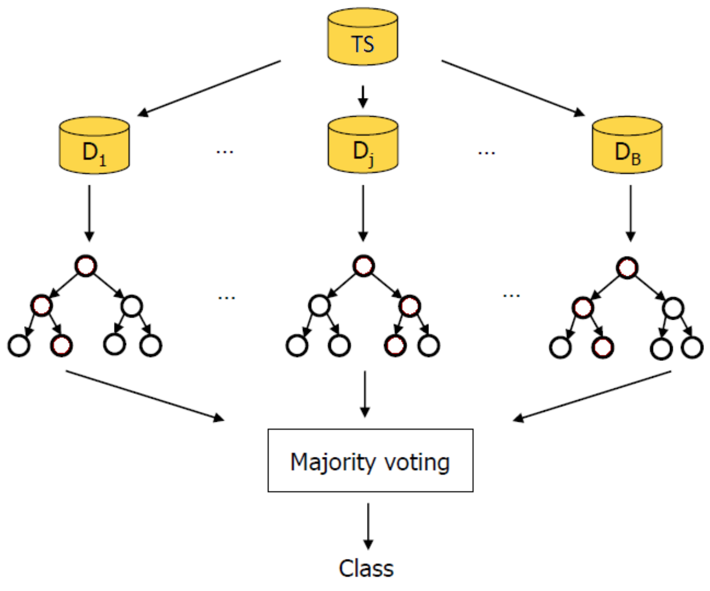


Figure 16: *Schema of random forest algorithm*

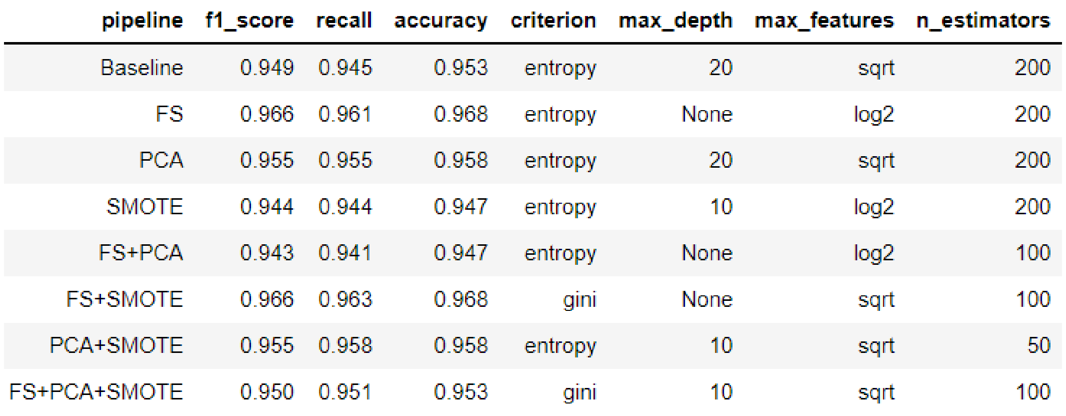
All the combinations of the following hyperparameters have been evaluated in hyperparameter tuning through a *Grid Search*:

- "criterion": ["gini", "entropy"]
- "max_features": ["sqrt", "log2"]
- "n_estimators": [10, 50, 100, 200]
- "max_depth": [None, 5, 10, 20, 25]

In particular, the last hyperparameter "max_depth" has been tuned to avoid overfitting. Indeed, decision trees are often pruned before reaching their full growth, and predictions are made according to a majority vote on the samples of each leaf.

The best performing configurations of hyperparameters on the training set have been chosen for rebuilding the model and evaluating it on the test set.

The following table shows the results of the hyperparameter tuning on each of the different transformations of the training set (pipeline), together with the main evaluation metrics:



pipeline	f1_score	recall	accuracy	criterion	max_depth	max_features	n_estimators
Baseline	0.949	0.945	0.953	entropy	20	sqrt	200
FS	0.966	0.961	0.968	entropy	None	log2	200
PCA	0.955	0.955	0.958	entropy	20	sqrt	200
SMOTE	0.944	0.944	0.947	entropy	10	log2	200
FS+PCA	0.943	0.941	0.947	entropy	None	log2	100
FS+SMOTE	0.966	0.963	0.968	gini	None	sqrt	100
PCA+SMOTE	0.955	0.958	0.958	entropy	10	sqrt	50
FS+PCA+SMOTE	0.950	0.951	0.953	gini	10	sqrt	100

Table 3: *Results of RF for each pipeline*

The performances for each pipeline have been compared in the following graph:

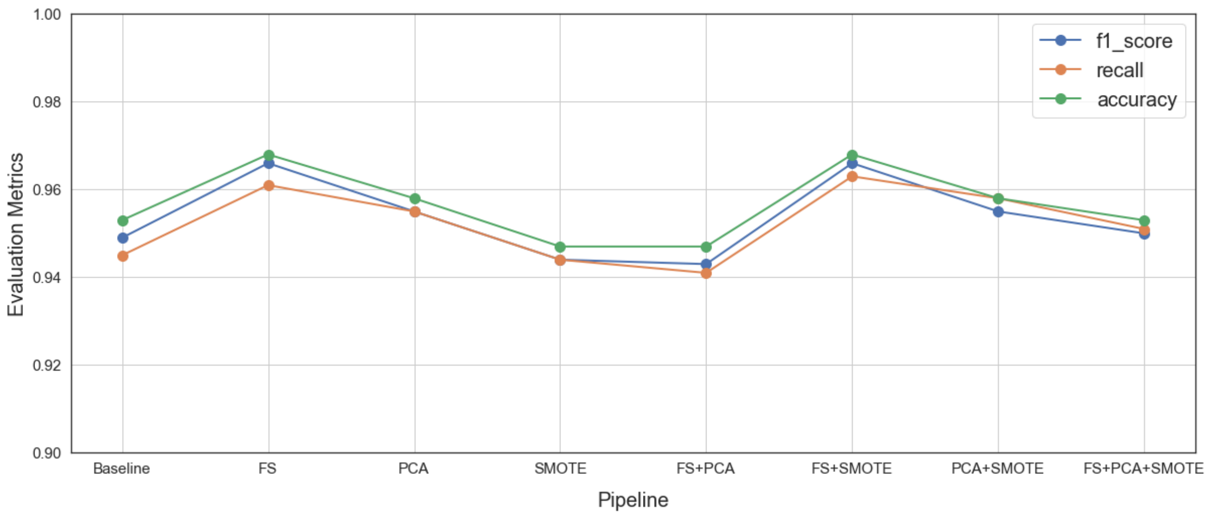
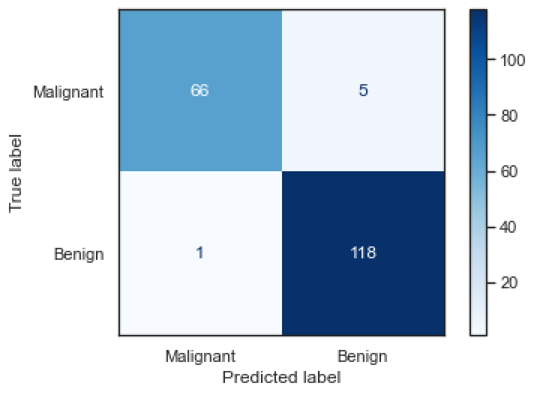


Figure 17: *Performances of RF pipelines*

As can be easily noticed, feature selection (FS) allowed to obtain good results, reaching the highest F1-score (96.6%) in random forest. On the other hand, PCA led to worse performances, probably due to the reduction of the feature space, making it more difficult for decision trees to achieve their goal of achieving the maximum class separation.

At the same F1-score (average value of the two classes), by comparing the confusion matrices of FS and FS+SMOTE, one can notice that, after having applied SMOTE, it has been possible to correctly classify a higher number of malignant samples. This may be relevant because, with the same F1-score, it would be more critical to consider a malignant tumor as benign rather than the reverse.



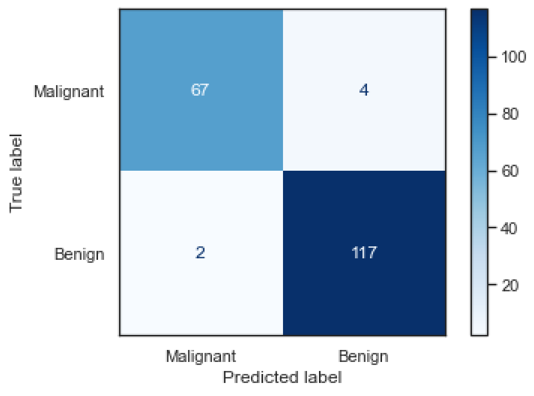


Figure 18: *Confusion matrices of RF with FS (left) and FS+SMOTE (right)*

4.2 Support Vector Machine

Support Vector Machine (SVM) is a binary linear classifier that aims at separating two classes through a **hyperplane**, with the goal of maximizing its distance (margin) from the closest points (**support vectors**) belonging to each class. Once the hyperplane has been found, the prediction rule is simply based on whether the test point lays on one side or the other one of it.

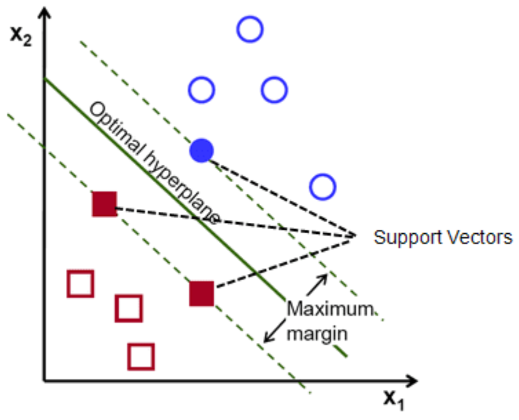


Figure 19: *Graphical representation of hard margin SVM*

More formally, it tries to define the hyperplane $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ such that its distance from the support vectors $\boldsymbol{x}_i$ is maximized. By supposing to have a binary label with values $\{-1, 1\}$, all points must satisfy $|\boldsymbol{w}^T\boldsymbol{x} + b| \geq 1$ relatively to their label. For support vectors, that lie on the boundary of the margin, it holds $|\boldsymbol{w}^T\boldsymbol{x} + b| = 1$.

Since the size of the margin is defined as $\frac{1}{||\boldsymbol{w}||}$, the optimization problem would be:

$$maximize_{w,b} \frac{1}{||\boldsymbol{w}||}$$

$$s.t.\ y_i[\boldsymbol{w}^T\boldsymbol{x}_i + b] \geq 1, \forall i$$

which means that $||w||$ should be as small as possible, and it is equivalent to:

$$minimize_{\boldsymbol{w},b}\frac{1}{2}||\boldsymbol{w}||^2$$

$$s.t.\ y_i[\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{i}}+b]\geq 1,\forall i$$

This is the primal optimization problem for **hard margin** SVM, where data is supposed to be perfectly linearly separable, otherwise no feasible solution would be found.
To extend the model on classes that are not perfectly linearly separable by means of a hyperplane, the addition of several **slack variables** ξ_i is necessary to allow the misclassification of difficult or noisy samples. The number of slack variables depends on a constant C which represents the **cost of misclassification**: the higher the value of the parameter, the lower is the margin of error of the classification.

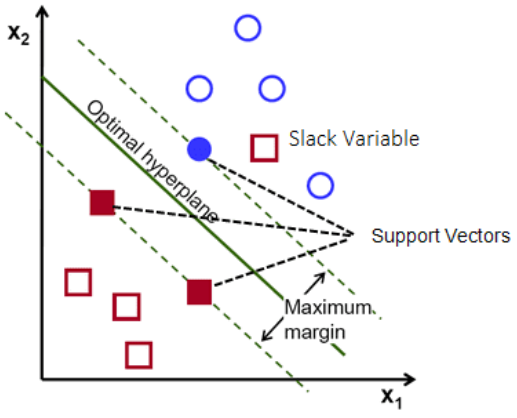


Figure 20: *Graphical representation of soft margin SVM*

The previous optimization problem is therefore generalized to the **soft margin** version:

$$minimize_{\boldsymbol{w},b}\frac{1}{2}||\boldsymbol{w}||^2+C\sum_i \xi_i$$

$$s.t.\ y_i[\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{i}}+b]\geq 1-\xi_i,\forall i$$

$$\xi_i\geq 0,\forall i$$

This optimization problem can be solved by using the Lagrange multiplier method, from which it is possible to obtain that $\boldsymbol{w}=\sum_i \alpha_i y_i x_i$, meaning that the weight vector $\boldsymbol{w}$ can be expressed as a linear combination of the support vectors. Indeed, only the points that are in between (or exactly on) the margin will have an $\alpha_i\neq 0$, i.e. only the support vectors will affect the decision function.
The **dual problem** is as follows:

$$max_\alpha\sum_i \alpha_i-\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j\,(\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{T}}\boldsymbol{x}_{\boldsymbol{j}})$$

$$s.t. \sum_i \alpha_i y_i = 0, \forall i$$

$$0 \leq \alpha_i \leq C, \forall i$$

4.2.1 Kernel Trick

A dataset that is not linearly separable in $\mathbb{R}^N$ may be linearly separable in a higher-dimensional space $\mathbb{R}^M$ (where $M > N$). In this context, the data should be mapped into a higher dimension by applying a non-linear function φ such that $\boldsymbol{x_i} \in \mathbb{R}^N \rightarrow \varphi(\boldsymbol{x_i}) \in \mathbb{R}^M$, allowing for a linear separation of the data in the new dimension. However, these transformations would lead to extremely high and impractical computational costs.

The **kernel trick** provides a solution to this problem. The "trick" is that kernel methods represent the data only through a set of pairwise similarity comparisons between the original data observations $\boldsymbol{x}$, instead of explicitly applying the transformations $\varphi(\boldsymbol{x})$.

In particular, if the kernel function $K: X \times X \rightarrow \mathbb{R}$ satisfies the conditions of the Mercer's theorem for which it must be symmetric, positive semidefinite, then it can be written as:

$$K(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T \varphi(\boldsymbol{x}')$$

Since the optimization problem only uses the training samples to compute pairwise scalar products $\boldsymbol{x_i^T x_j}$, by virtue of the kernel trick it is possible to efficiently learn non-linear decision boundaries for SVM by simply replacing all scalar products $\boldsymbol{x_i^T x_j}$ in the optimization problem with $K(\boldsymbol{x_i}, \boldsymbol{x_j})$, without explicitly transforming $\boldsymbol{x_i}$ and $\boldsymbol{x_j}$ to $\mathbb{R}^M$.

The most common kernel functions are:

- **Linear Kernel**: $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$
- **Gaussian RBF Kernel**: $K(\boldsymbol{x}, \boldsymbol{x}') = e^{-\gamma \|x - x'\|^2}$, with $\gamma > 0$ which is a hyperparameter that can be tuned to regulate how many support vectors are significantly affecting the decision function.
- **Polynomial Kernel**: $K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \boldsymbol{x}' + c)^d$ with $c \geq 0$ and $d \in \mathbb{N}$ which indicates the polynomial degree.

All the combinations of the following hyperparameters have been evaluated in hyperparameter tuning:

- "C": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]
- "kernel": ["linear", "rbf", "poly"]
- "gamma": [0.0001, 0.001, 0.01]

The following table shows the results of the hyperparameter tuning on each of the different transformations of the training set (pipeline), together with the main evaluation metrics:

pipeline	f1_score	recall	accuracy	C	kernel	gamma
Baseline	0.977	0.975	0.979	100.0	rbf	0.001
FS	0.983	0.982	0.984	100.0	rbf	0.001
PCA	0.960	0.956	0.963	100.0	rbf	0.001
SMOTE	0.972	0.970	0.974	1.0	rbf	0.01
FS+PCA	0.966	0.961	0.968	100.0	rbf	0.001
FS+SMOTE	0.983	0.982	0.984	0.1	linear	None
PCA+SMOTE	0.961	0.962	0.963	10.0	rbf	0.01
FS+PCA+SMOTE	0.972	0.970	0.974	0.1	linear	None

Table 4: *Results of SVM for each pipeline*

The performances for each pipeline have been compared in the following graph:

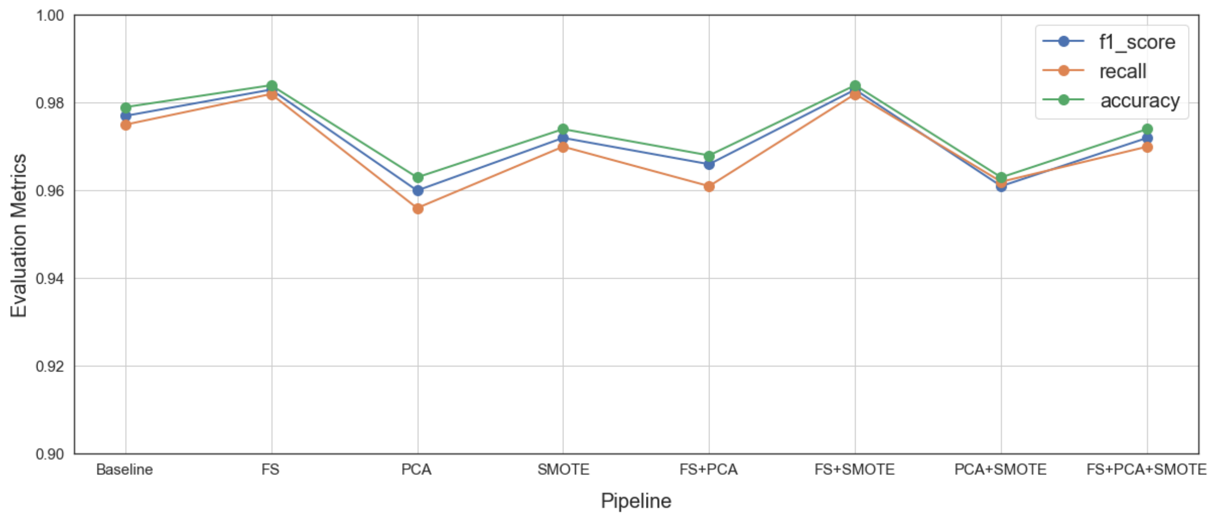


Figure 21: *Performances of SVM pipelines*

Similarly to the previous model, the highest performances have been achieved with FS, whereas PCA did not improve results with respect to the baseline (where no transformations to the training set are applied, except for standardization). This may be due to the fact that SVM favors higher dimensionality space, while PCA implies a dimensionality reduction. The confusion matrix of the configuration with the highest F1-score is displayed:

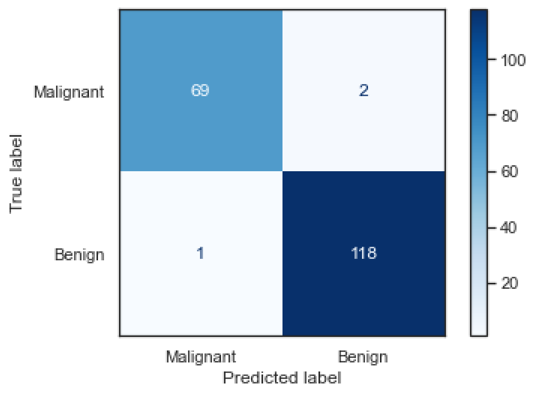


Figure 22: *Confusion matrix of SVM with FS*

4.3 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is one of the simplest supervised algorithms in machine learning and it is based on the assumptions that close samples in the feature space are similar. Specifically, a data point is classified according to the most frequent occurring class of its *k* closest samples, which are called neighbors.

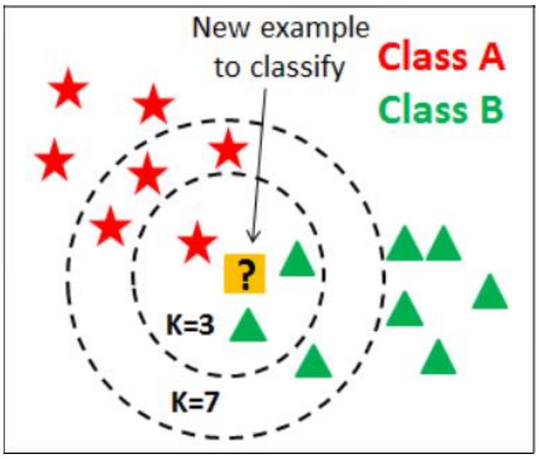


Figure 23: *Graphical representation of KNN*

The main steps of the algorithm are:

1. Choose an integer value for *k*. This selection can be non-trivial because if *k* is too small, the problem becomes sensitive to noisy points, while if *k* is too large, the neighborhood may include points from other classes.
2. For each sample in the test data, calculate the distance between the test data and each sample of the training set by using a distance measure. The ones used in the current work are:

   $$\text{Euclidean distance: } d(a,b) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

   $$\text{Manhattan distance: } d(a,b) = \sum_{i=1}^{n}|a_i - b_i|$$

   where a and b are the two samples and n is the number of different features.
3. Based on the distance value, sort them in ascending order and select the top *k* rows.
4. Assign the class label to the test point by taking the majority vote of class labels among the *k* neighbors.

Under some circumstances, it can be useful to add a weight in the final majority voting calculation, implementing a **Weighted KNN**, to give more importance to the points that are nearby and less importance to those which are far away. By doing so, the closer the neighbor is to the data point to classify, the more weight it has in determining the outcome of the vote.

All the combinations of the following hyperparameters have been evaluated in hyperparameter tuning:

- “n_neighbors”: [3, 5, 10, 20, 30, 50, 100]
- “weights”: [“uniform”, “distance”]
- “distance”: [“euclidean”, “ manhattan”]

The following table shows the results of the hyperparameter tuning on each of the different transformations of the training set (pipeline), together with the main evaluation metrics:

pipeline	f1_score	recall	accuracy	n_neighbors	weights	distance
Baseline	0.954	0.949	0.958	5	uniform	manhattan
FS	0.966	0.961	0.968	10	uniform	manhattan
PCA	0.966	0.963	0.968	3	uniform	euclidean
SMOTE	0.966	0.963	0.968	5	uniform	euclidean
FS+PCA	0.972	0.970	0.974	5	uniform	manhattan
FS+SMOTE	0.934	0.941	0.937	10	uniform	manhattan
PCA+SMOTE	0.955	0.952	0.958	100	uniform	manhattan
FS+PCA+SMOTE	0.938	0.940	0.942	30	uniform	euclidean

Table 5: *Results of KNN for each pipeline*

The performances for each pipeline have been compared in the following graph:

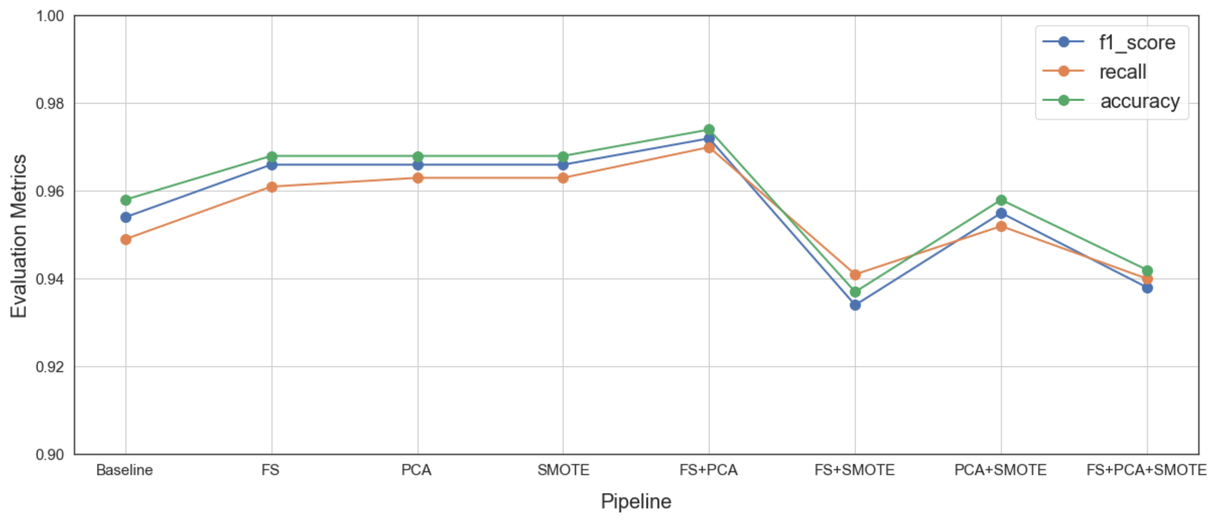


Figure 24: *Performances of KNN pipelines*

In the case of KNN, as one can easily expect, PCA improves the results. This is due to the fact that KNN, being distance-based, suffers from curse of dimensionality and therefore, reducing the feature space through PCA helps in reducing this effect on the algorithm.

Moreover, as can be seen from the chart, the combination of SMOTE with FS and/or PCA provides worse results. By analyzing the confusion matrices, it can be possible to state that this worsening is due to a misclassification of the benign class. Indeed, a reduction of the feature space and an increase in malignant samples could lead to an overlap of the samples of the two classes (i.e. some malignant samples could invade the space of the benign ones), therefore classifying some benign tumors as malignant.

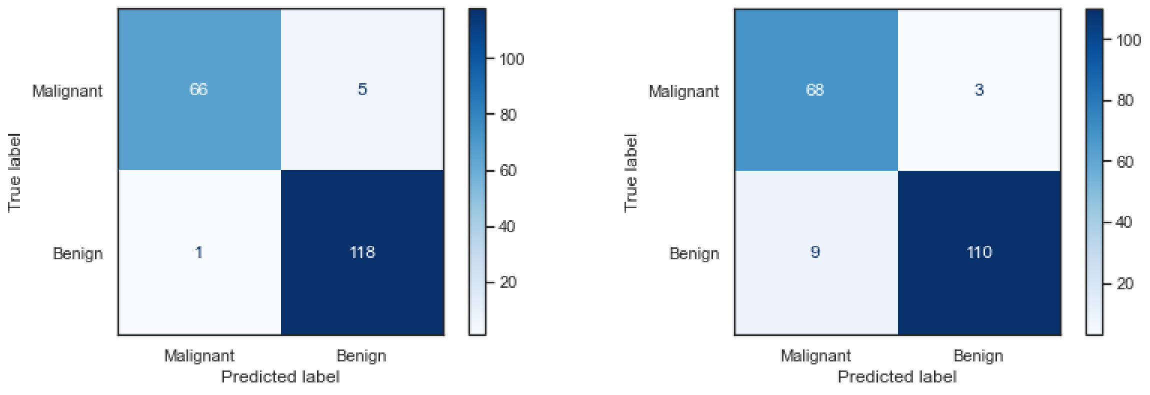


Figure 25: *Confusion matrices of KNN with FS (left) and FS+SMOTE (right)*

5 Conclusions

In this study, different supervised algorithms have been inspected and presented with their mathematical details, and finally used on the Breast Cancer Wisconsin (Diagnostic) Data Set to build an automated diagnosis system. The Support Vector Machine was proved to be the best one on the task, by providing a F1-score of 98.3%. However, all models exploited in the work can be considered adequate to the classification purpose because they were able to provide performances above the 93% for F1-score and accuracy, for every combination of data preprocessing techniques. Such reason for high performances is implicit in the dataset because it has been constructed for the same purpose of the analysis.
Moreover, given these high values of the metrics, by analyzing the confusion matrices, it is possible to notice how the variations are minimal and always involve a very limited number of samples, suggesting that those variations should not necessarily invalidate the choice of one pipeline over another.

Bibliography and Sitography

[1] https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)

[2] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.707&rep=rep1&type=pdf

[3] B. B. Mandelbrot. The Fractal Geometry of Nature, chapter 5. W. H. Freeman and Company, New York, NY, 1977

[4] Shai Shalev-Shwartz, Shai Ben-David, “Understanding Machine Learning: From Theory to Algorithms”, published by Cambridge University Press (2014)

[5] Dirk P. Kroese, Zdravko I. Botev, Thomas Taimre, Radislav Vaisman, “Data Science and Machine Learning, Mathematical and Statistical Methods”, 2020