Mathematics in Machine Learning

Machine Learning algorithms for Breast Cancer Dataset Wisconsin (Diagnosis)

Di Eugenio Valerio s278972

1 INTRODUCTION

The aim of this project is to exploit some supervised machine learning algorithms to diagnose some breast cells to be effectively symptoms of cancer or not , underlining the mathematical aspects behind the methods used. In particular, Decision Trees, Random Forest, Support Vector Machine, K_Nearest Neighbors and Linear Discriminant analysis algorithms have been applied and carefully explained in order to build an automated model for breast cancer diagnosis.

Breast cancer is the most common cancer among women and one of the major causes of death among women worldwide. To give a concrete idea, currently in the United States the average risk of a woman developing it sometimes in her life is about 13% (according to data collected in 2020). It consists on a malignant tumor that develops from cells in the breast, which grow out of control, but it is possible to analyze the shape, the size and the growth in order to recongnize benign from malignant cells. For this purpose, Dr. William H. Wolberg, physician at the university Of Wisconsin Hospital at Madison, in 2015 used a graphical computer program curve-fitting based on several breast cells images to extract some numerical characteristhics about cell nuclei to create the dataset which is now used for this project.

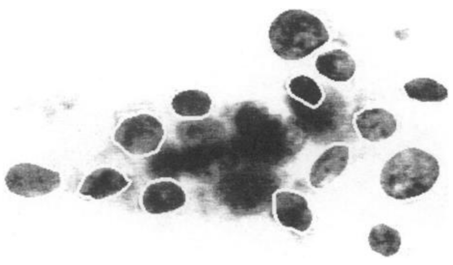


FIGURA 1: IMAGE OF A BREAST CELL NUCLEI OUTLINED BY THE CURVE-FITTING PROGRAM

1.1 ENVIRONMENT

As far as the code part is concerned, it has been fully conducted with Python language, exploiting the following frameworks:

- Scikit learn
- Numpy
- Pandas
- Imblearn

While the visualization part has been handled by using:

- Seaborn
- Matplotlib

2 DATA ANALYSIS

2.1 DATA DESCRIPTION

The dataset used in this case study is “Breast cancer Wisconsin (Diagnosis)” from the UCI machine learning repository. It contains the analysis of **569 images** cell nuclei, and for each one the following attributes are extracted:

Number	Feature	Description
1	ID number	Unique ID
2	Diagnosis	(M = malignant, B = benign) : the target classes, listed as 0 (M) and 1 (B)
3	Radius	(mean of distances from center to points on the perimeter)
4	Texture	Standard deviation of gray-scale values
5	Perimeter	Perimeter of the nucleus
6	Area	Area of the nucleus
7	Smoothness	Local variation in radius length
8	Compactness	(perimeter^2 / area - 1.0)
9	Concavity	(severity of concave portions of the contour)
	Concave points	Number of concave portions of the contour)
10	Symmetry	
11	Fractal dimension	("coastline approximation" - 1)

The first attribute is not considered in the analysis, it is the identification number of the cell’s nuclei the features refers to, while the second one, ‘Diagnosis’, is the column which are going to be predicted, the target label, which says if the cancer is M (malignant), or B (benign). When it is equal to 1, it means the cancer is malignant and when it is 0, it means it is benign.

The mean, standard error and "worst" or largest (mean of the three largest values) of features 3-11 were computed for each image, resulting in **30 features** effectively taken into account in the analysis.

In the following tables the minimum and the maximum values are reported for each attribute in order to have a general overview of the range of the values and the order of magnitude each one can assume:

Mean	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	Concave points	Symmetry	Fractal dimension
Max	28.11	39.28	188.5	2501	0.163	0.345	0.427	0.201	0.304	0.97
Min	6.98	9.71	43.79	143.5	0.053	0.019	0.0	0.0	0.106	0.05

SE	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	Concave points	Symmetry	Fractal dimension
Max	2.873	4.885	21.98	542.2	0.031	0.135	0.396	0.053	0.079	0.03
Min	0.112	0.36	0.757	6.802	0.002	0.002	0.0	0.0	0.008	0.001

Worst	Radius	Texture	Perimeter	Area	Smoothness	Compactness	Concavity	Concave points	Symmetry	Fractal dimension
Max	36.04	49.54	251.2	4254	0.223	1.058	1.252	0.291	0.664	0.208
Min	7.93	12.02	50.41	185.2	0.071	0.027	0.0	0.0	0.156	0.055

It is possible to note how values are different in magnitude, so a standardization phase before applying models is needed (it will be explained later in details)

2.1 DATA EXPLORATION

In this phase data are explored in order to extract as more information as possible before applying models.

The first important thing to check is the distribution of the labels, which is shown in the following charts:

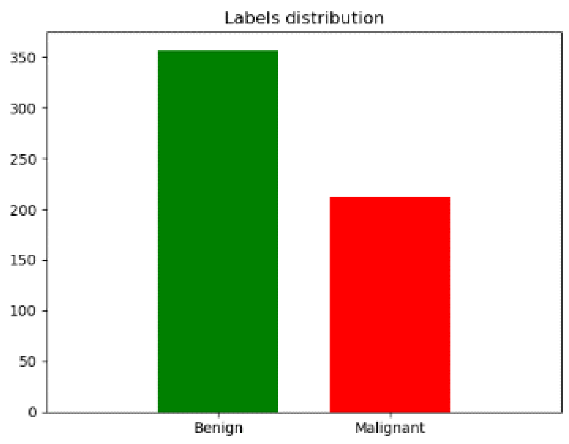


FIGURA 2: LABELS DISTRIBUTION

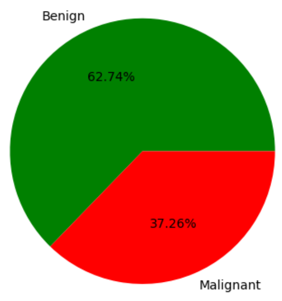


FIGURA 3: PERCENTAGE OF DISTRIBUTIONS

The charts show a classes imbalance: 357 observations, which account for 62.74% of the total , indicate the absence of cancer cells, while 212, 37.26% of the total, shows their presence. So, a binary classification problem on a that unbalanced dataset has to be handled.

Before computing some analysis on the features, it is relevant to check whether some null or empty fields are present:

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
```

in this case, the values of each attribute are valid, so no records have to be dropped and no values have to be filled.

One of the main goals of visualizing the data is to observe which features are most helpful in predicting malignant or benign cancer. The other is to see general trends that may aid us in model selection.

To know more about how the features affect the target (i.e. to understand which features have larger predictive value and which does not bring considerable predictive value for class assigning), it can be useful to visualise the distribution of data via histograms:

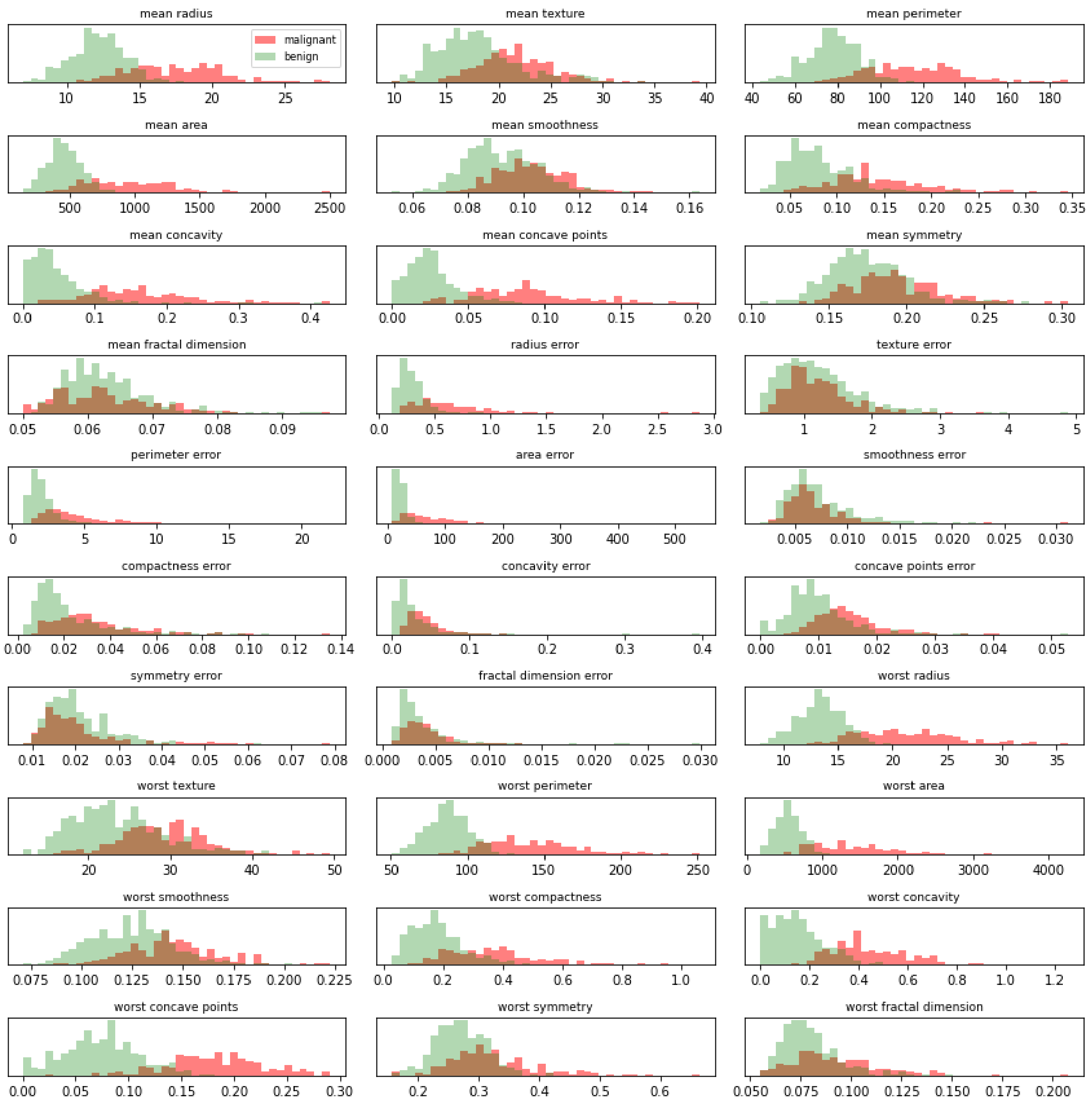


FIGURE 4: FEATURES DISTRIBUTION ACCORDING TO THE CLASSES

These representations are useful to grasp the degree of overlap between the two classes for each feature. The more the values overlap, the more that feature is not relevant in descerning the classes. According to this, 'symmetry error' and 'texture error' are only two examples of features that could be less meaningful for this task, while for example 'worst concave points' and 'worst perimeter' could be pretty much discriminative.

Another relevant point which could affect the classification performances is the correlation among features: the presence of strongly correlated features may lead to a decline in the performances of some classification algorithms which assume that the predictors are all independent. Another benefit from spotting correlation among features is that the same information may be encoded with less attributes, and this could lead to simpler final models: in fact, some methods suffer from high dimensional datasets (especially distance based ones with a few number of samples), so reducing the dimensions of the feature vectors can make it more trustable and stable, if the discarded dimensions don't affect significantly the total original information.

For the purpose, Pearson's coefficient is used to investigate the correlation between pairs of variables: in particular, given a random vector $(\boldsymbol{X}, \boldsymbol{Y})$:

$$\rho_{x,y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} \in [-1,1]$$

where $Cov(X, Y)$ is the covariance and it compares two variables in terms of deviations from their mean (or expected) value: in other words, it describes how similarly two random variables deviate from their mean, meaning that on average for each deviation of x (with respect to the mean of X), the same deviation can be observed for y (with respect to the mean of Y).

Correlation is a normalization of the covariance by σ_X and σ_Y**,** the standard deviations of $\boldsymbol{X}$ and $\boldsymbol{Y}$**.** The value of the coefficient $\rho_{x,y}$ is a number between -1 and +1 and it's a measure of the correlation strength between two variables. The higher the absolute value of the coefficient, the stronger the linear correlation between the two features.

It is important to stress how Pearson coefficient spots **only linear correlation**, hence a value around zero may indicate a strong non-linear trend between two variables, and not necessarily a total correlation absence.

The following representation shows the correlation matrix between features: since the way the matrix is constructed make it symmetric and because of the large number of features, only the heatmap of the lower diagonal matrix (the diagonal itself is excluded) is reported for a better visualization. In addition, the statistic is computed by inferring the covariance and the standard deviations from the dataset itself for each pair of attributes, since the real probability distribution is not known but only a sample of realizations of the random vector (X, Y) is available.

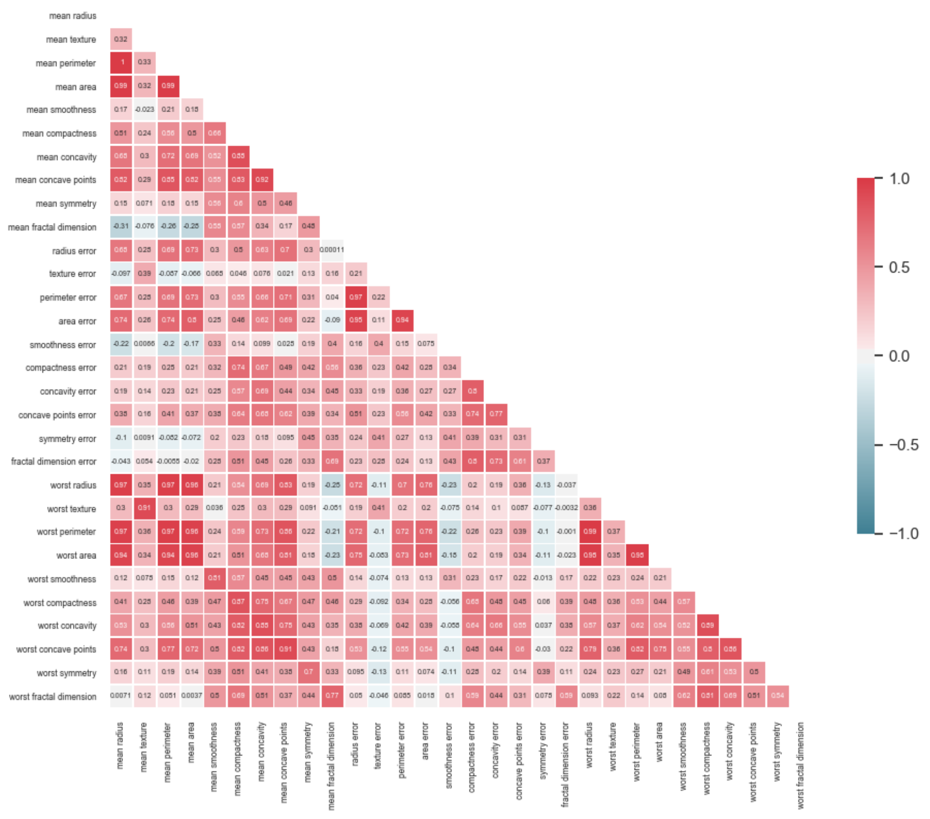


FIGURE 5: CORRELATION MATRIX WITH PEARSON'C COEFFICIENT

Figure 5 shows how many features are strongly correlated to each other, and some examples are reported below:

- Perimeter error and radius error are strongly correlated, with $\rho_{perimeter\_error,radius\_error}$ = 0.97

- Mean perimeter and mean radius are completely correlated, with $\rho_{mean\_perimeter,mean\_radius}$ = 1
- Worst radius and worst perimeter are strongly correlated, with pworst_perimeter,worst_radius $\rho_{worst\_perimeter,worst\_radius}$= 0.99

To give a further proof of the linear dipendence of these variables, their interactions plots are shown:

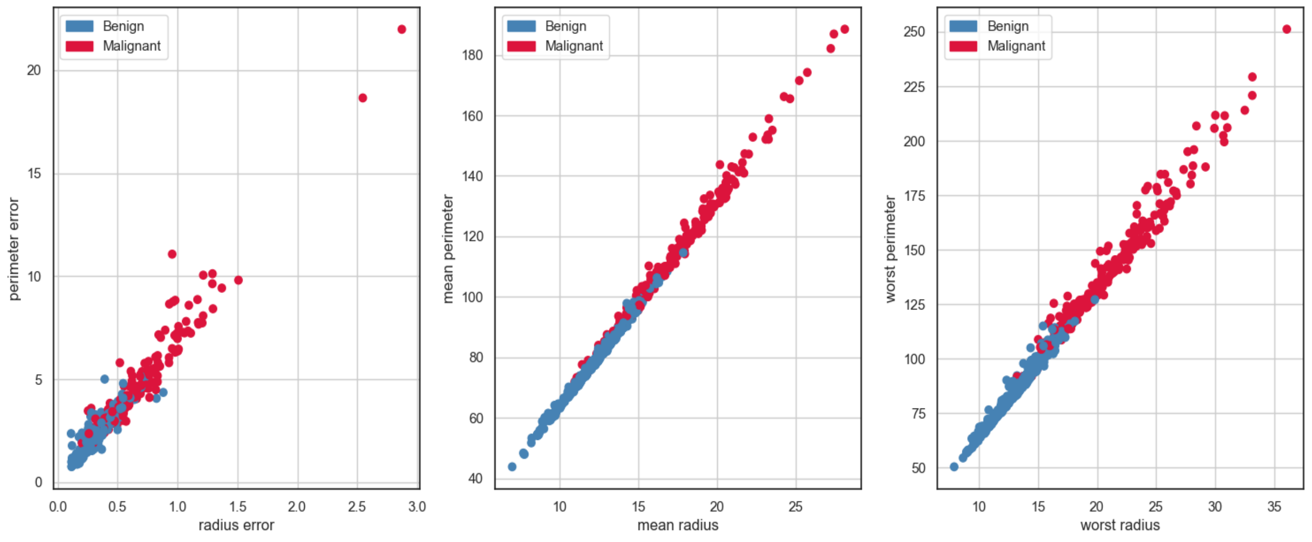


FIGURE 6: INTERACTIONS PLOTS BETWEEN STRONGLY CORRELATED FEATURES

The charts confirm what expected, the features in the same graph shows a linear trend as the Pearson coefficient suggested, indicating they encode pretty similar information, just on different scale.

For completness, also charts of non-strongly correlated features are reported. In particular:

- Smoothness error and mean texture with $\rho_{smoothness\_error,mean\_texture}$ = 0.098
- Worst area and fractal dimension error with $\rho_{worst\_area,fractal\_dimension\_error}$= 0.023
- Symmetry error and mean concave point with $\rho_{symmetry\_error,mean\_concave\_point}$=0.085

with the following charts:

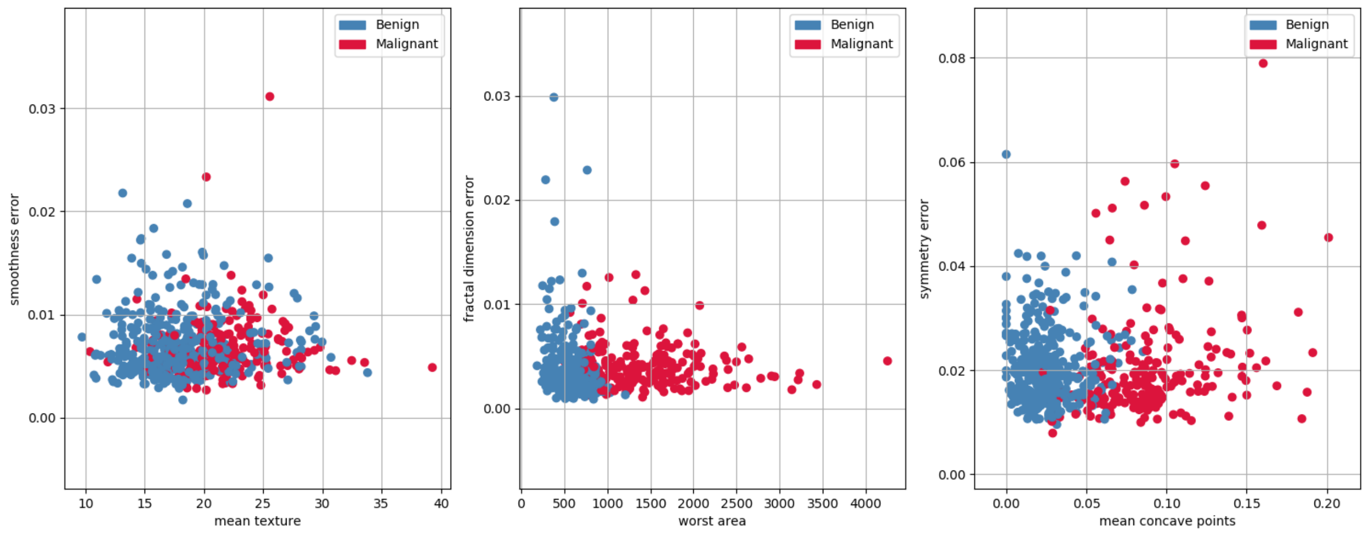


FIGURE 7: INTERACTIONS PLOTS BETWEEN WEAKLY LINEAR CORRELATED FEATURES

It is possible to notice how very low Pearson's coefficient values denotes no linear correlation.

Finally, the boxplots for each of the 30 features are plotted in order to have a further look at the distribution and to eventually spot outliers (i.e. bad computed data or really far in values from the expected range). It is important to stress that for the purpose a standardization was applied on all the attributes to make them have the same scale, otherwise it would have been impossible to compare them.

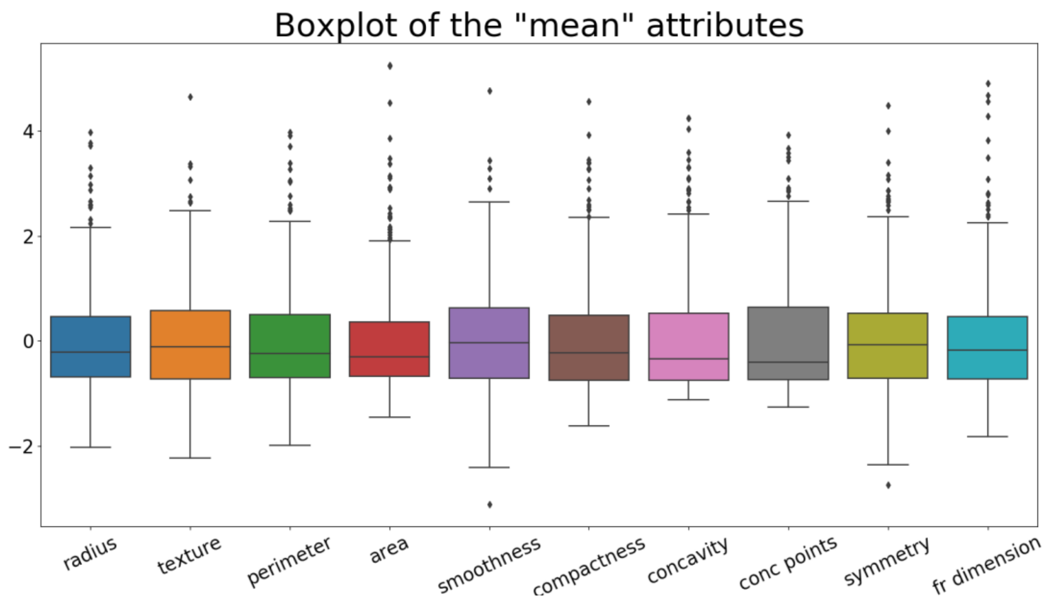


FIGURE 8: BOXPLOT OF THE 'MEAN' ATTRIBUTE

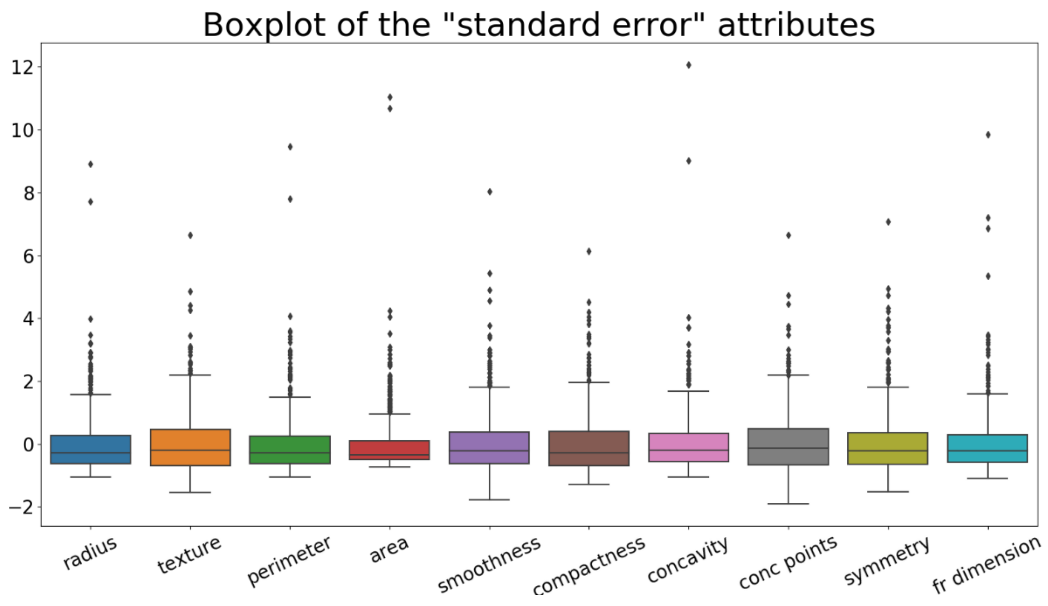


FIGURE 9:BOXPLOT OF THE 'STANDARD ERROR' ATTRIBUTES

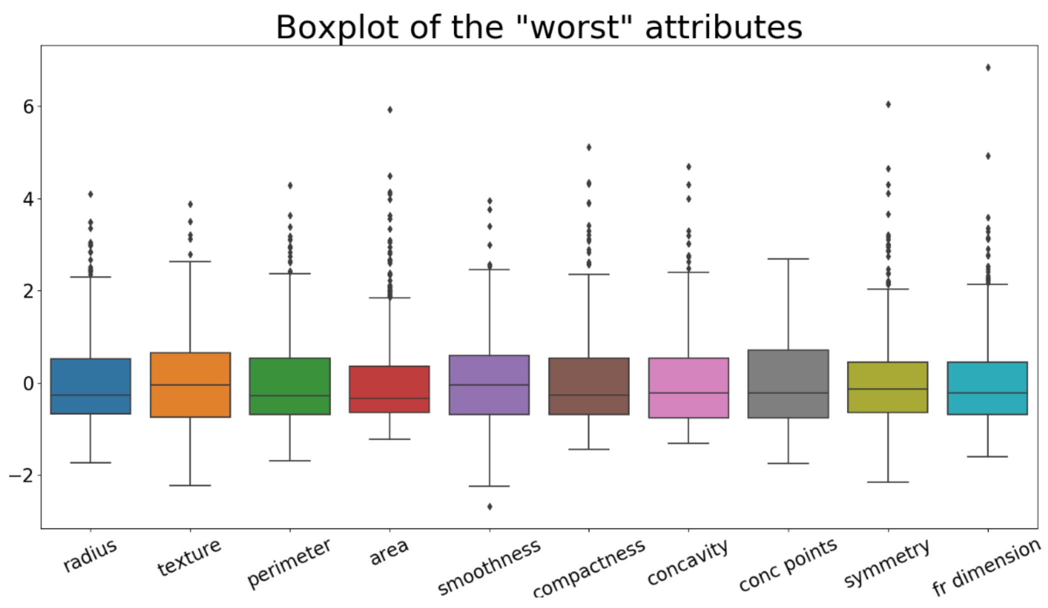


FIGURA 10: BOXPLOT OF THE 'WORST' ATTRIBUTES

Boxplots show the 1st, 2nd and 3rd quartiles, and the maximum and the minimum estimated bound after standardization. Furthermore, through them it is possible to detect outliers presence (i.e. the single grey points): defining the Interquartile Range as $IQR = (Q_3 - Q_1)$, every sample placing beyond $Q_1 - 1.5\ IQR$ and $Q_3 + 1.5\ IQR$ is considered an outlier. This method is the one exploited here, and according to it many outliers are detected. It is possible to change the multiplicative coefficient to decide the treshold to overcome to be considered an outlier, but, unlike this, for this task, no samples are discarded because there is no prior domain knowledges which safely allow to do it. In addition, the dataset is not too numerous in samples, and this has contributed to the decision to maintain all the observations.

To provide a general overview on all the data, all the previous boxplots are collected and plotted in a unique chart:

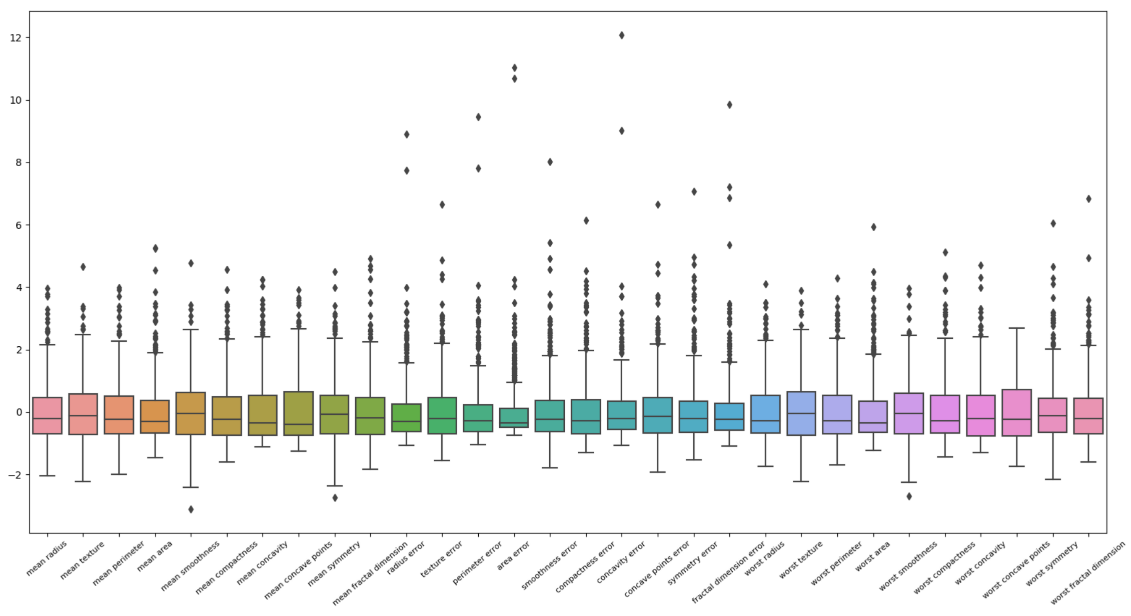


FIGURA 11: BOXPLOT OF ALL THE FEATURES

3 DATA PREPROCESSING

3.1 TRAIN-TEST SPLIT

The dataset is now divided in training set and test set, with the proportion 3:1. Given the imbalance fashion in the initial dataset, a stratified approach is adopted in order to have in both the new sets approximately the same percentage of benign and malignant samples of the initial set. The two final sets obtained are as follows:

	Total	Benign	Malignant	Benign %	Malignant %
Training set	381	239	142	63%	37%
Test set	188	118	70	63%	37%

It has to be considered that the split is performed randomly, but to make evaluations statistically meaningful and to make results consistent across the different classifiers, it is always kept the same one.

3.2 FEATURE SCALING

As the data exploration has shown, features have values in different magnitudes and this could be a problem for the majority of the algorithms: in fact, features with variance greater in magnitude than the others may control the objective function and make the classifiers not able to correctly learn properties from other attributes as expected. In addition, many of them assume data to be zero centered and with a variance equal to one, which make learning algorithms less sensitive to eventual data perturbations. For example a K-Nearest Neighbors algorithm is not scale invariant and the computation of the distance strongly depends on the variance of the attributes (high variance can cause it to model random noises in the training data rather than the intended output).

The scaling technique used is **standardization** and it works as follows, independently for each column:

$$x' = \frac{x - \mu_x}{\sigma_x}$$

where μ_x is the sample mean and σ_x the sample standard deviation of feature *X.* Basically, each value ***x*** of the feature ***X*** is transformed according to this, independently for each column. The standardization leads each attribute to have a new mean equal to 0 and variance 1 (1^2 precisely).

The mean and the standard deviation are computed only taking into account the training set, while succesively both the training and the test set are transformed. It has been processed in this way because in the initial phase test set is assumed to be unknown, so it would be formally incorrect to use some test set information during the training process.

3.3 DIMENSIONALITY

As quickly mentioned above, generally speaking it is not always a good idea to propagate in the algorithms all the samples attributes. High dimensional data may negatively affect the algorithms perfomances, slowing down the training time and worsen the interpretability. Moreover different machine learning algorithms suffer from the curse of dimensionality: if there are more features than observations the risk of massively overfitting the model becomes really high. Then, too many dimensions causes every observation in the dataset to appear equidistant from all the other, and this is effectively a problem when used distance-based algorithms (like K-Nearest Neighbors for example), because if the distances are all approximately equal, then the observation appear equally alike (as well as equally different), making the algorithm perfomances meaningless.

3.3.1 FEATURE SELECTION

As previously seen in Fig.5, some features in this dataset are strongly linearly correlated, so it is not meaningful to keep all of them because they may contain redundant information. In this way, features with a Pearson coefficient > 0.94 with other predictors are discarded. In particular 7 feature are dropped (mean perimeter, mean area, perimeter error, area error, worst radius, worst perimeter, worst area), leading the dataset to have 23 total attributes.

3.3.2 DIMENSIONALITY REDUCTION WITH PCA

Another tool for carrying the dimensionality reduction is the Principal Component Analysis (PCA): it is a method to reduce data dimensions of large datasets, by transforming a large set of variables into a smaller one that still contains most of the information of the largest dataset (in other words, trying to lose less information as possible). In order to adopt this technique, a previous data standardization is needed because it works with standardized data. In practice, it works as follow:

1) Given a dataset centered in zero, it implicity computes correlation matrix Σ to spot correlation among features, so the aim is to understand how the variables of the input dataset are varying from the mean with respect to each other. It is a $n * n$ symmetric matrix (with n equals to original number of dimensions) having covariances associated with all possible pairs of variables as entries).

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)**

EXAMPLE OF COVARIANCE MATRIX FOR 3 DIMENSIONAL DATA

Note that $Cov(a, a) = Var(a)$, so in the main diagonal there are the variances of each initial variables.

2) It computes the eigenvectors and eigenvalues of the covariance matrix to identify the principal components: principal components are new variables constructed as linear combinations of the initial variables, and such combinations are done in the way that these principal components are uncorrelated and the most of the information are compressed in the first principal components (basically, it tries to put maximum possible information in the first component, then the maximum remaining information in the second and so on).
More in depth, eigenvectors of the covariance matrix are the directions of the axes where there is the most variance (e.g. the principal components), and eigenvalues are the coefficients attached to eigenvectors, which give the amount of variance carried in each Principal Component. By ranking the eigenvectors in descending order of their eigenvalues, it is possible to get principal components in order of significance (information): in fact, principal components represent the directions that explain the maximal amount of variance (i.e. the lines that capture most information of the data). The relationship between data and information is that the larger the dispersion along a line, the larger the variance, the more the information it has.
This approach is basically the eigendecomposition of Σ: it is a symmetric positive semidefinite matrix, so it is possible to diagonalize it by means of an orthogonal matrix P composed by the eigenvectors of Σ and obtain a similar matrix A.

$$A = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_d \end{pmatrix}, \ \lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_d \geqslant 0, \ \lambda_i \in \mathbb{R}^+$$

Where λi are the eigenvalues (variance of new features). It's important to note that A and Σ are similar, so the initial total variance among the features of the dataset is always the same but redistributed on new axes. A basically is the the covariance matrix in the new basis found, and since it's diagonal all new features are uncorrelated.

3) The data from the original dataset are projected from the original axes to the ones represented by the principal components.

A mathematical approach could also considered and it leads to the equivalent result: PCA is performed by solving the following minimization problem in order to find the direction of maximum variance (first principal component):

$$X \in R^{n*d}, \mathit{dataset\ centered\ in\ zero}$$

$$\Sigma,\ \text{sample covariance matrix}$$

$$\mathit{find}\ \vec{z}_1 = a_1\vec{e}_1 + \cdots + a_d\vec{e}_d\ \ \text{subject to}$$

$$\vec{z}_1 = \mathit{argmax}_{\vec{z}_1}\ {\vec{z}_1}^T \Sigma\, \vec{z}_1\ s.t.:\ ||\vec{z}_1|| = 1$$

The other directions are calculated in the same way, with the additional constraint to be orthogonal to the dimensions already found.

One important parameter to choose before applying PCA is the number of principal components on which data have to be projected, looking for a good balance between the number of components and the covered variance. Here PCA is applied to both the datasets to reduce the number of dimensions. The following chart shows how variance is dristibuted among principal components:

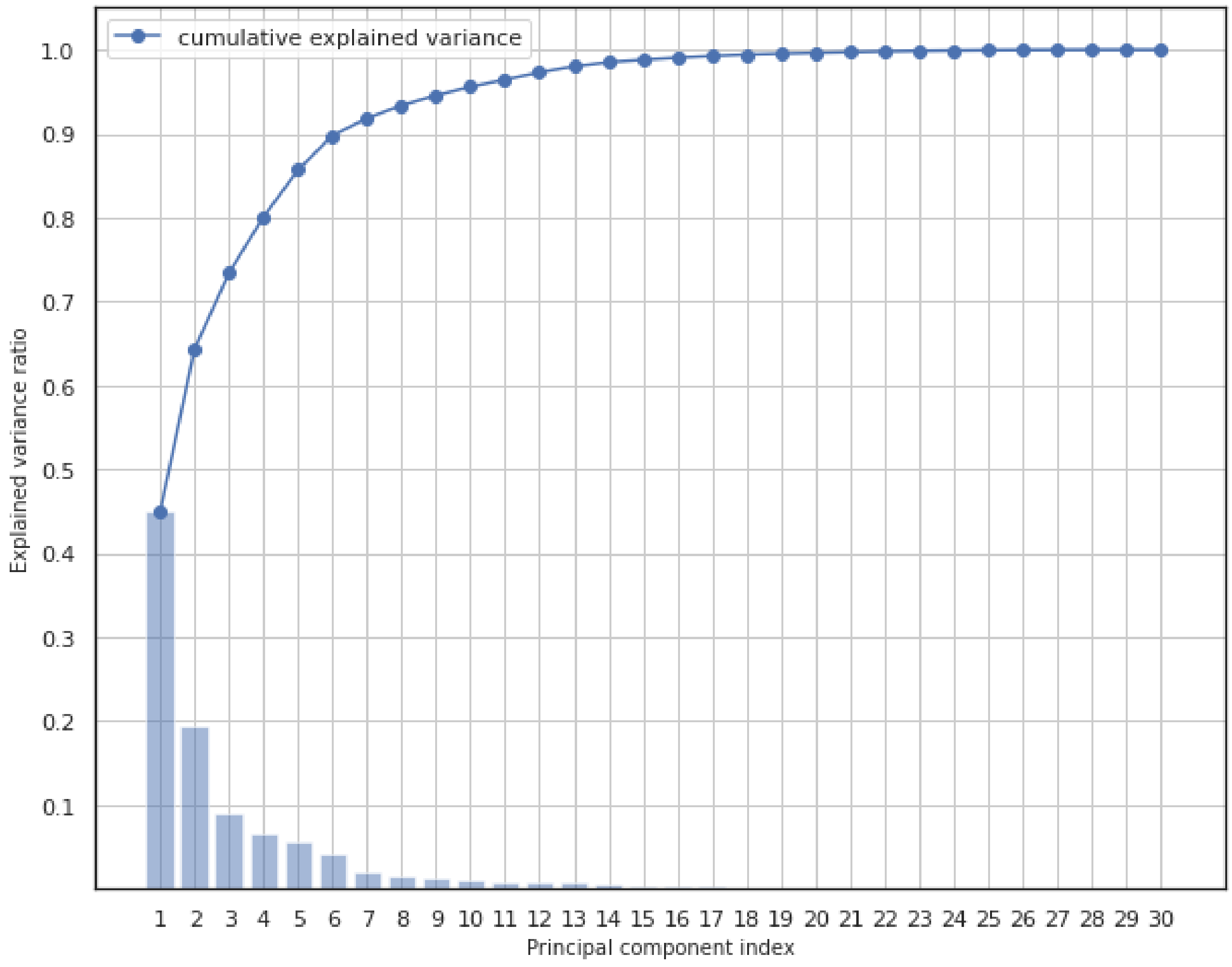


FIGURE 2: VARIANCE DISTRIBUTED ACROSS 30 PRINCIPAL COMPONENTS

The first 7 principal components are eventually kept to explain a variance of about 91%.

3.4 OVERSAMPLING

As shown in point 1, the dataset is that unbalanced, and this could become an issue for some machine learning algorithms, making them not able to properly learn all the classes (they could be biased towards the class most occurring). Even though the ideal scenario would be collecting more real data, there exist some techniques that could come in handy to adjust the class distribution of the dataset: oversampling and undersampling. The former involves duplicating examples in the minority class until the samples of both the classes become numerically equals, while the latter consists on deleting examples from the majority class for the same purpose as before.

Usually oversampling is better because let us keep all the information in the training set, while by performing undersampling there is the risk to drop relevant information. According to this, and given the size of the dataset used for the project, it is reasonable to go with oversampling technique. Two usual approaches are:

- **Random oversampling**: the minority class samples to duplicate are selected randomly.
- **Syinthethic Minority Over-sampling Technique** (SMOTE): synthetic samples from the minority class are generated to obtain a class-balanced dataset. SMOTE first randomly select a minority class instance point and finds its k nearest minority class neighbors, then it creates new artificial samples along the lines between the selected point and one of its neighbors.

It is important to state that resampling techniques should be used only in the training phase and never applied on the test set, because the artificial samples technically never existed in real dataset, they are not part of the real target distribution which is instead represented in the test data.

4 CLASSIFICATION ALGORITHMS

4.1 METRICS OF EVALUATION

Models are evaluated according to the following measures:

- Accuracy
- F1 score
- Precision
- Recall

In order to better undestand these metrics, it’s important to get some fundamentals:

	Predicted (0)	**Predicted (1)**
Actual (0)	**TN**	**FP**
Actual (1)	**FN**	**TP**

- **TP:** samples for which the prediction is positive and the true class is positive
- **FP:** samples for which the prediction is positive but the true class is negative
- **TN:** samples for which the prediction is negative and the true class is negative

- **FN:** samples for which the prediction is negative but the true class is positive

According to this, it possible to define the above metrics:

- **Accuracy** is a ratio of predicted labels which strictly match with the true labels to total number of samples. The closer to 1, the better it is.

$$\text{Accuracy}(y, \hat{y}) = \sum_{i=0}^{nsamples-1}(y_i - \hat{y}_i) = \frac{TP+TN}{TP+FP+FP*FN}$$

  It is not a trustable measure when dealing with unbalanced datasets.

- **Precision:** measures the fraction of correct classified instances among the ones classified as positive. Precision is an appropriate measure to use when the aim is to minimize **false positives.**

$$\text{precision} = \frac{TP}{TP+FP}$$

- **Recall**: it measures how many of the actual positives a model capture through labelling it as True Positive.It is an appropriate score when the aim is to minimize **false negatives**:

$$\text{recall} = \frac{TP}{TP+FN}$$

- **F1 score**: it depends on Precision and Recall, useful when dealing with unbalanced datasets.

$$\text{F1 score} = 2\ \frac{precision*recall}{precision+recall}$$

4.2 MODEL SELECTION

In this section a description of the algorithms used is presented. For each of them, different hyperparameters are tuned in order to find the ones performing best, and, once found, the model trained with them is used to predict class labels on test data combining different data preprocessing techniques presented above in order to spot some performances differences.

From section 3.1 it is possible to note how the entire dataset has been divided into training and test set, without any validation one. It has been decided to procede in this way in order to perform a **stratified K-fold cross validation technique** to look for the best hyperparameters for each model. This approach consists on dividing the training set of observation into K groups of approximately equal size. In turn, each partition is treated as validation set, while the remaining K – 1 ones are used to fit the model. The K groups are created ensuring that each partition contains approximately the same proportions of the original training set class labels. The value of K must be chosen carefully, a poorly chosen of this value may lead the model to have high variance or bias in score. A good trade-off is empirically given by a K values of 10 ot 5: in this project it is used K = 5 to have more samples in the validation set, given the size of the dataset.

The best configuration is selected by comparing the different metrics, but a greater weight is given to precision score, that, as explaind before, it is useful when minizing false positive: in this specific case, a false positive is a sample wrongly classified as ‘Benign’, and in a real case this is the worst possible scenario.

4.2.1 DECISION TREE

It is one of the most intuitive and interpretable supervised learning algorithms. It creates a model that predicts the class of the target variable by learning simple decision rules inferred from data. Decision Tree are composed of :

- Nodes, which represent a feature
- Branch, representing a rule (or decision)
- Leaf, representing an outcome.

They use a layered splitting process, where at each layer they try to split the data into two or more groups so that data that fall into the same group are most similar to each other, while groups themselves are as different as possible to each other. Each node acts as a 'test case' for some condition, and each branch descending from that node corresponds to one of the possible answers to that test case. At each step, it selects the feature that best divides the space into different labels exploiting some impurity measures. In this project the GINI index is used:

$$Gini = 1 - \sum_{i=1}^{C} pt(k)^2$$

Where $pt(k)$ is the frequency of class K appearing in node t, with C equals to the total number of classes. It measure the degree or probability of a particular variable being wrongly classified when it is randomly chosen. The degree of GINI index is etween 0 and 1, where 0 denotes that all the elements are correctly classified (ideal situation, the node is 'pure'). According to this, the lower the measure, the less impure the node is.

A decision tree is constructed with the training data and then it will be used at classification time to predict target variable. The more the levels of the tree, the more the depth (it doesn't include the root node) and the complexity. It is always preferable to have less complex models and to avoid overfitting, so, given this, often decision trees are pruned before reaching their full growth: it means that its size got reduced by removing some sections of the Tree that provide little predictive power, so predictions are made according to a majority vote on the sample of each leaf.

Hyperparemers tuning has been applied with stratified K-fold cross validation technique on the training set, and the best configurations for different combinations of data preprocessing techniques are reported in the table below (note that standardization and removal of high correlated features are always performed):

	Max Depth	**Min impurity decrease**	**Min samples split**	**Accuracy**	**F1 score**	**Precision**	**Recall**
None	5	0	2	0.931	0.926	0.948	1.0
OS	10	0	2	0.919	0.914	0.964	0.912
PCA	5	0	2	0.931	0.926	0.948	1.0
OS+PCA	10	0	2	0.918	0.914	0.964	0.912

The best configuration found is: Max_Depth=10, Min_impurity_decrease=0 , Min_samples_split=2, with data preprocessing: OS / OS + PCA

Generally speaking, Decision Trees don't suffer from the feature scaling since each attribute is treated independently, and it is very easy to interpret. On the other hand, they are eventually capable to deal with missing value, but they are also considered weak learners, and often ensambles techniques (bagging or boosting) are applied on them in order to achieve best performance.

4.2.3 RANDOM FOREST

Random forest consists of a large number of individual decision trees that operate as an ensamble. Each individual tree in the random forest makes a class prediction and the class with the most votes is assigned to the sample taken into account. The intuition behind this method is that a large number of relatively **uncorrelated** models (trees) operating as a committee will outperform any of the individual constituent model. Uncorrelation is the key of Random Forests better performances, and this is ensured through two methods:

1) **Bagging** (Bootstrap aggregation): decision trees are very sensitive to the data they are trained on (small changes to the training data can result in significantly different tree structures), so to promote variance between each internal model, each individual tree is trained on a sample of the same size of the training dataset randomly chosen with replacement.

2) **Feature randomness**: each decision tree of the model, at each split, can pick features only from a random subset of features (usually $\sqrt{p}$, where *p* is the number of total attributes) and not among all the features as usual in a standard decision tree algorithm. This helps in providing more diversification.

In conclusion with random forest we have trees that are not only trained on different sets of data but also use different features to make decision. In this way, even if decision trees are unstable (high variance in prediction), the overall prediction is instead more robust (low variance). This because given a set of n independent observations **Z1, Z2, ...Zn** each with variance $\boldsymbol{\sigma^2}$ the variance of the mean of the observations Z is $\frac{\sigma^2}{n}$ so averaging a set of observations the resultin variance is lower.

In the following table, the results of the Random Forest hyperparameters tuning, still performed in the way described above:

	Max Depth	**Min impurity decrease**	**Min samples split**	**Num estimators**	**Accuracy**	**F1 score**	**Precision**	**Recall**
None	15	0	2	50	0.958	0.954	0.971	1.0
OS	10	0	2	50	0.955	0.952	0.9635	0.970
PCA	10	0	4	50	0.958	0.957	0.971	1.0
OS+PCA	10	0	4	50	0.955	0.952	0.9635	0.970

The best configuration found is: Max_Depth = 10, Min_impurity_decrease=0 , Min_samples_split = 2, N_estimators = 50 with data preprocessing: OS/OS+PCA.

4.2.4 SUPPORT VECTOR MACHINE

Support vector machine (SVM) is a classifier which aims to find a hyperplane in the data space such that this hyperplane separates the classes as well as possible. Once found, the prediction rule simply takes into account the side of the plane the new data will lay.

If more than one hyperplanes fit the purpose, the one with the higher margin (distance) between support vector (the closest points to the hyperplane, one for each class) is selected.

1) Hard margin

Mathematically, the equation of the hyperplane is $\mathbf{w}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{b}=\mathbf{0}$ where $\vec{\boldsymbol{w}},\boldsymbol{b}$ are defined in order to maximize its distance to the nearest points from either group, and this is called 'maximum-margin hyperplane'. Instead planes defined by support vectors have respectively the equation $\mathbf{w}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{b}=\mathbf{1}$ and $\mathbf{w}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{b}=-\mathbf{1}$ and the distance in between is given by $\frac{2}{\|\mathbf{w}\|}$; the aim is to maximize this distance, but this is equal to minimize ½ $\|\mathbf{w}\|$. Moreover, in order to prevent data from falling into the margin, we impose that all points must satisfy $|\mathbf{w}^{\mathrm{T}}\boldsymbol{x}+\boldsymbol{b}|\geq\mathbf{1}$ relatively to their label.

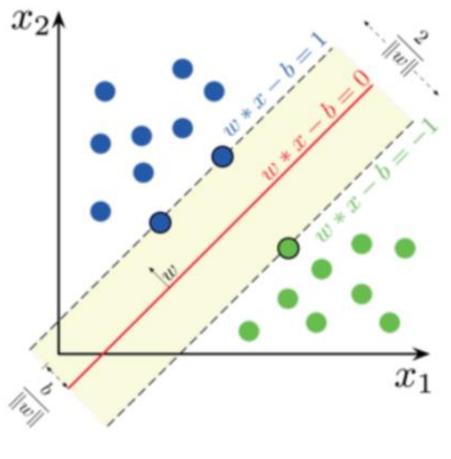


FIGURE 13: 2D REPRESENTATION OF SVM

So, the **optimization problem** (primal) is:

$$\boldsymbol{min}\|\vec{\mathrm{w}}\|^2 \quad \textbf{s.t.} \;\; \boldsymbol{y_i(w^T x_i + b) \geq 1, \forall_i}$$

In a binary classification case the labels are **{-1, 1}** and the decision function is applied as

$$\boldsymbol{x} \mapsto \boldsymbol{sign}(\mathbf{w}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b})$$

This model is referred to as **Hard margin**, and it assumes data are linearly separable, otherwise no feasible solution would be found.

2) Soft margin

To extend SVM to cases in which the data are not linearly separable, the Hinge Loss is introduced to penalize points falling into margins:

$$L_{hinge} = \max(0,1 - y_i(w^t + b))$$

.

According to this, the optimization problem to solve becomes:

$$min\frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \max(0,1 - y_i(w^t + b))$$

Which is supposed to be equal to:

$$min_{w,b,\xi} + \frac{1}{2}||w||^2 + C\sum_{i=1}^{n} \xi$$

Subject to:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \forall_{\boldsymbol{i}} \ ,$$

$$\xi_i \geq 0$$

ξ is a slack variable which is introduced to in order to soften the misclassification constraint, allowing the model to make a certain number of mistakes and letting the margin to remain as wide as possible. The amount of misclassification allowed is controlled by the hyperparameter C which regulates the strength of the Hinge Loss: low values of C give the priority to a simpler model, with an higher margin, with more misclassified points but that likely generalizes well on unseen data, whereas larger C leads the focus on avoiding misclassification at the expense of keeping the margin small (less robustness), by imposing a stronger minimization of the loss.

3) Dual problem (soft margin)

Soft margin optimization problem can be seen under another point of view, analyzing its dual representation:

$$max_\alpha \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (X_i^T X_j)$$

s.t.

$$\sum_{i=1}^{n} \alpha_i \, y_i = 0 \ \ and \ 0 \leq \ a_i \leq C, \forall i$$

This is obtained by solving the previous optimization problem through the Lagrangian multipliers methods, and from it it is obtained that $\boldsymbol{w} = \sum_i^n \boldsymbol{\alpha_i y_i x_i}$, meaning that weight vector is a linear combination of the training data. In particular, the dual variable α_i is obtained, and it is equals to 0 for all but the support vectors, so they are the only points taken into account in the analysis, and it means that the weight vector *W is the linear combination of support vectors.*

By exploiting these information, to classify a point *x* we calculate:

$$sign\left(\sum_i^n \alpha_i y_i (x_i^T x)\right) + b$$

It is important to note that both the training and the predictions on new data depend on the data points only through their inner product. This allows to best tackle non linearly datasets, where a linear hyperplane doesn't manage to fit the data. It may happen that such data are still linearly separable if projected onto higher dimensional space, so this is done through a non-linear function φ which transforms the training data as:

$$x_i \in R^d \rightarrow \varphi(xi) \in R^{d'}, d' > d$$

It is possible to make another step: by taking into account that in the new decision function just defined in the dual optimization problem only the inner product is meaningful, it is possible to apply any transformation (don't matter what it is) on the training set as long as it is possible to calculate the inner product. So it's important to find a function able to compute it in the new space.

Kernel trick comes handy in this way. A Kernel function is introduced and defined as below:

$K(xi, xj) = \varphi(xi) * \varphi(xj)$,

so it's important only to know K, which defines inner product in the transformed space, and not the mapping function itself.

So, the decision function can be written as:

$$sign \sum \alpha_i y_i K(x_i, x_j) + b$$

The most common kernel functions used are:

- Linear Kernel: $K(xi, xj) = xi^T xj$
- Polynomial Kernel: $K(xi, xj) = (x^T xj + c)^d$
- Radial Basis Function Kernel: $K(xi, xj) = \mathrm{e}^{-\gamma(\mathrm{xi}-\mathrm{xj})^2}$

Apart these, the condition which determines which functions con be considered as Kernel is given by the Mercer's theorem:

1) Symmetric function
2) Function whose Kernel matrix is positive semi-definite in a finite input space.

In particular, RBF Kernel computes a similarity measures between the test data and the support vectors, labelling the point taken into account according to the closeness to support vectors themselves. In this way, the new hyperparameter gamma defines how far the influence of each support vector reaches (the lower the value, the more the support vectors affecting the decision function).

In this case, both linear and RBF Kernel SVM are tested, still performed in the way described above. To give a slightly more accurate idea about how these work it's useful looking at their decision boundaries, so the data distribution and the hyperplane separating the two classes are plotted in the figure below: to make the visualization more understandable and clear, the data in the charts are temporarely projected on the first two principal components:

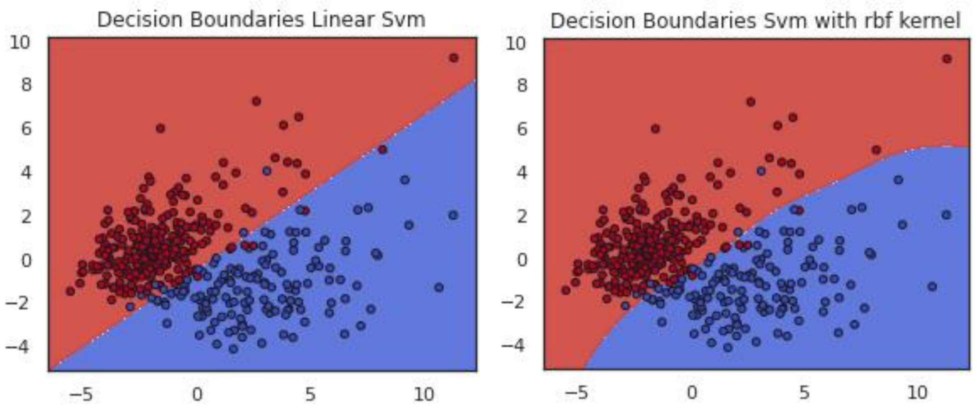


FIGURE 14: SVMS DECISION BOUNDARIES

From the figures it is possible to note that in two dimensions the boundary found by the Linear SVM is a line, while for the SVM with RBF kernel the situation is a bit more complex: being the mapping-data function unknown, it is not possible to catch the shape of the hyperplane found in higher space, but in the original space it results in a curve line.

The results achieved are the following:

Linear SVM	C	Accuracy	F1 score	Precision	Recall
None	0.01	0.971	0.977	0.972	0.979
OS	0.1	0.968	0.975	0.972	0.979
PCA	0.01	0.971	0.977	0.963	0.983
OS+PCA	0.1	0.968	0.975	0.975	0.979

The best configuration found is: C = 0.1 with data preprocessing: OS+PCA

SVM RBF	C	Gamma	Accuracy	F1 score	Precision	Recall
None	1	auto	0.960	0.969	0.959	1.0
OS	1	auto	0.960	0.968	0.965	1.0
PCA	1	auto	0.960	0.969	0.959	1.0
OS+PCA	1	auto	0.960	0.965	0.972	1.0

The best configuration found is: C = 1, gamma= 'auto', with data preprocessing: OS+PCA.

The gamma value 'auto' means that gamma is equal to $\sqrt{n\_features}$

4.2.5 LINEAR DISCRIMINANT ANALYSIS

Linear Discriminant analysis is an unsupervised technique used both for classification and dimensionality reduction and it is a generalization of Fisher's disciminant analysis. In a classification problem set up the objective is to ensure maximum separability or discrimination of classes, and this is achieved by finding a projection to a line such that projections of samples belonging to different classes on that line result well separated.

In particular, given d-dimensional points belonging to two different classes and let V be the unit vector giving the direction, $V^T x_i$ represents the projection of sample x_i onto a line in direction V.

Defining:

- n_1 , n_2 : the sets of samples belonging respectively to class 1 and class 2 ;
- μ_1 and μ_2 : the means of the two classes ;
- $\tilde{\mu}_1$ and $\tilde{\mu}_2$: the means of the projections of the two classes,

good way to measure separation of the projections could be $|\tilde{\mu}_1 - \tilde{\mu}_2|$. An example of how to exploit this to choose the direction is represented in the figure below:

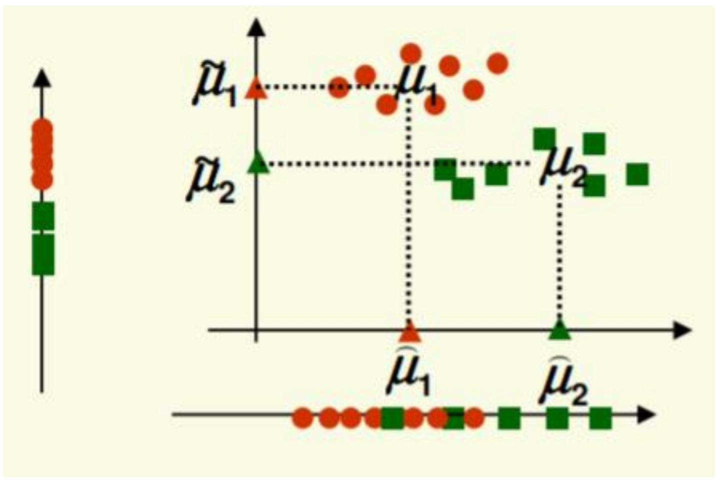


FIGURA 15: DATA PROJECTION ALONG THW TWO DIMENSIONS

According to the fig 15 , the direction which better divides the projections of the two classes is the vertical one, but it is not the right choice: in fact, $|\hat{\mu}_1 - \hat{\mu}_2| > |\tilde{\mu}_1 - \tilde{\mu}_2|$, so the horizontal one should be taken into account for projection. This because the measure described above doesn't take into account the variance of classes, so it has to be normalized by a factor proportional to the classes variance.

Given samples $Z_1, \dots, Z_n$ and their mean as $\mu_z = \frac{1}{n} \sum_{i=1}^{n} Z_i$, their **scatter** is defined as:

$$S = \sum_{i=1}^{n} (Z_i - \mu_z)^2$$

And represents the variance multiplied by n. Scatter measures the spread of data around the mean (as the variance, but on different scale).

Remembering that $y_i = V^T x_i$ is the projection of each sample, it possible to define the scatter for both the classes as:

$$\widetilde{S_1^2} = \sum_{y_i \in class1}^{n} (y_i - \mu_1)^2$$

$$\widetilde{S_2^2} = \sum_{y_i \in class2}^{n} (y_i - \mu_2)^2$$

In this way a measure which takes into account the variance is obtained, and it can be used as the normalization factor discussed above. A new measure is thus defined:

$$J(V) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)}{\widetilde{S_1^2} + \widetilde{S_2^2}}$$

The aim is to project data in the direction V which maximizes $J(V)$: projected means are wanted to be as far to each other as possible, while the scatter of the two classes as small as possible (so that samples are not so sparse from the mean of their belonging class).

At this point, it is important to express J as a function of V and maximize it. After some computations, it is possible to obtain that:

$$\widetilde{S_1^2} + \widetilde{S_2^2} = V^T S_w V$$

where $S_w = S_1 + S_2$ and represents the **within** class scatter matrix (S_1 and S_2 are respectively class scatter matrices for class1 and class2), and:

$$(\tilde{\mu}_1 - \tilde{\mu}_2) = V^T S_b V$$

where $S_b = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T$ and measures separation between the means of the two classes.

Thus the objective function can be written:

$$J(V) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)}{\widetilde{S_1^2} + \widetilde{S_2^2}} = \frac{V^T S_b V}{V^T S_w V}$$

So, if V making $J(V)$ larger is found, a well separation between classes is guaranteed. Linear Discriminant Analysis has no particular hyperparameters to be tuned, so it is evaluated with its default ones directly on the test data.

4.2.6 K-NEIGHBORS CLASSIFIER

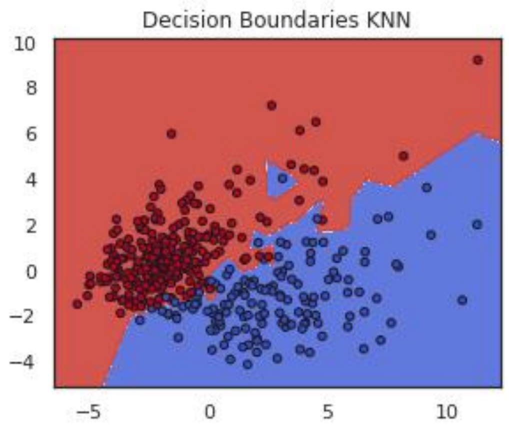
K-Nearest Neighbors is a classification algorithm which stores all available data during the training phase and classifies new test points with a similarity approach: in particular, given a sample to classify, it assigns it a label according to the classes of its K closest points by majority voting principle. The closeness can be measured through different methods, and some of them are reported below:

- **Euclidean** : $\sqrt{\sum_{i=1}^{k}(x_i - x_j)^2}$
- **Manhattan** : $\sum_{i=1}^{k}|x_i - x_j|$
- **Minkowski** : $(\sum_{i=1}^{k}|x_i - x_j|^p)^{\frac{1}{q}}$

KNN is a lazy learner, it does not learn anything from the training data, it simply stores and use them for classification. Generally, this could be computationally expensive and can lead to a slow process if dealing with a large amount of data.

Performances of the algorithm strongly depend on the quality and the structure of the data: if the dataset composition is not homogeneous (features are on different scale), it may be difficult to sub-divide each group correctly, so for better results it is highly suggested to perform data standardization. Moreover, if the dataset is characterized by a certain number of outliers, the performances, especially for low values of K, can fall. The size of the neighborhood has a meaningful impact on the goodness of the classification: when K is too small, the probability of misclassification is high (higher if there are also outliers, as stated above), if too large instead it may include unwanted data belonging to other classes. Furthemore, when dealing with high dimensional data, being a distance based algorithm it may struggle in predicting the correct class, it may suffer from the already mentioned curse of dimensionality , so dimensionality reduction technique is strongly recommended if dealing with high dimensions features. Finally, also high unbalanced dataset can make some issues come up, due to the nature of its classification process. On the other hand, it is that simple algorithm to interpret, and it requires no assumptions about data (as linearity).

The following chart shows the decision boundaries of KNN with K = 1. Even in this case, data are temporarily projected in two dimensions by using PCA.



The only parameter to be tuned is K, which generally is chosen such that K > number of classes, in order to avoid situations in which it is impossible to assign the label.

The K parameter hypertuning, still performed in the way described above, leads to the following results:

	Number of neighbors	**Accuracy**	**F1 score**	**Precision**	**Recall**
None	10	0.951	0.961	0.945	0.979
OS	5	0.942	0.953	0.958	0.954
PCA	10	0.950	0.961	0.945	0.979
OS+PCA	5	0.942	0.953	0.958	0.954

Best hyperparameter found: K = 5, with data preprocessing: OS / OS+PCA

5 RESULTS

In the following table, the best configuration and the results of all the classifiers tested are collected:

Classifier	**Configuration**	**Data Preprocessing**	**Accuracy**	**F1 score**	**Precision**	**Recall**
Decision Tree	MD=10 MID=0 MSS=2	OS OS +PCA	0.918	0.914	0.964	0.912
Random Forest	MD=10 MID=0 MSS=2 NE=50	OS OS +PCA	0.955	0.952	0.9635	0.970
Linear SVM	C=0.01	OS+PCA	0.968	0.975	0.975	0.979
RBF kerne SVM	C=1, Gamma='auto	OS+PCA	0.960	0.965	0.972	1.0
KNN classifier	K=5	OS OS +PCA	0.942	0.953	0.958	0.954

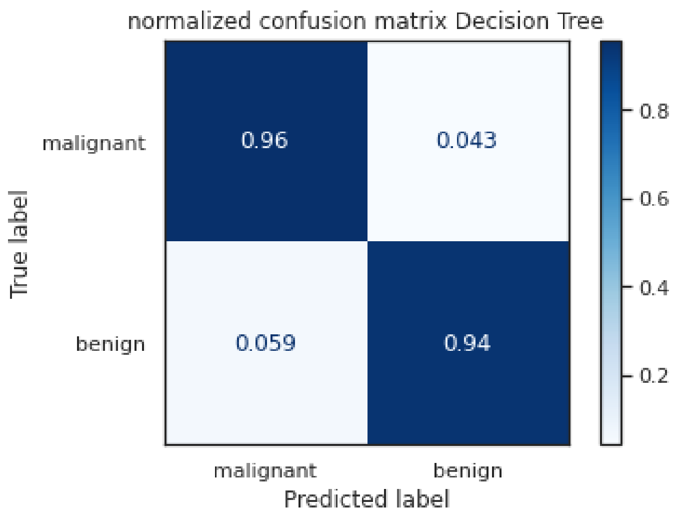
MD= max_depth, MID=min_impurity_decrease, MSS=min_samples_split, NE=n_estimators, OS=smote oversampling

LDA classifier is directly evaluated on the test set.

Finally, each of the model is rebuilt with the best hyperparameters found and evaluated on the test set. The correspondent performances and confusion matrices are:

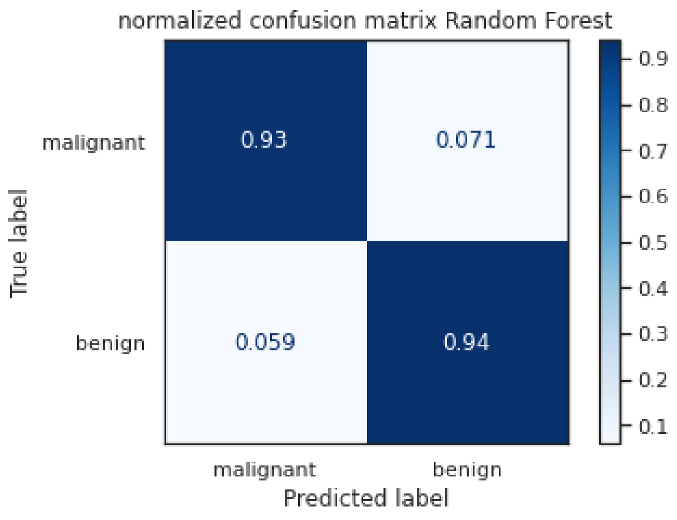
DECISION TREE

Accuracy	0.946
F1 score	0.956
Precision	0.973
Recall	0.941



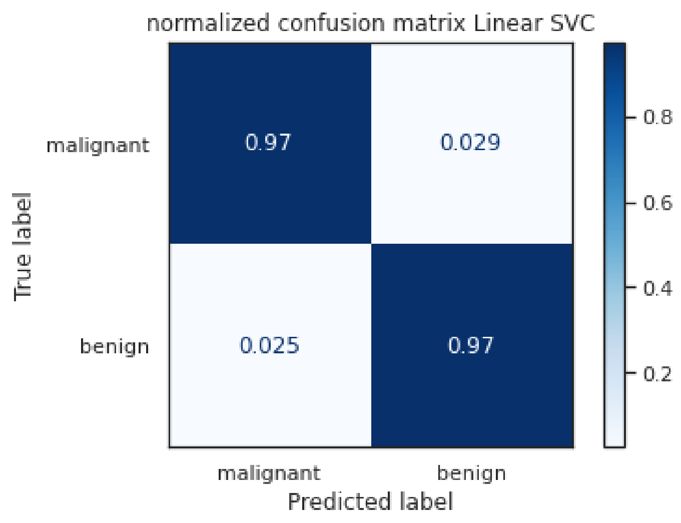
RANDOM FOREST

Accuracy	0.936
F1 score	0.948
Precision	0.956
Recall	0.940



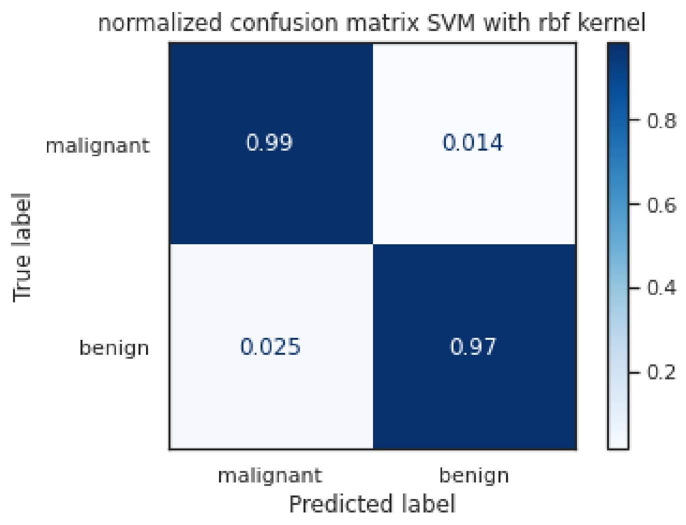
LINEAR SVC

Accuracy	0.968
F1 score	0.974
Precision	0.982
Recall	0.966



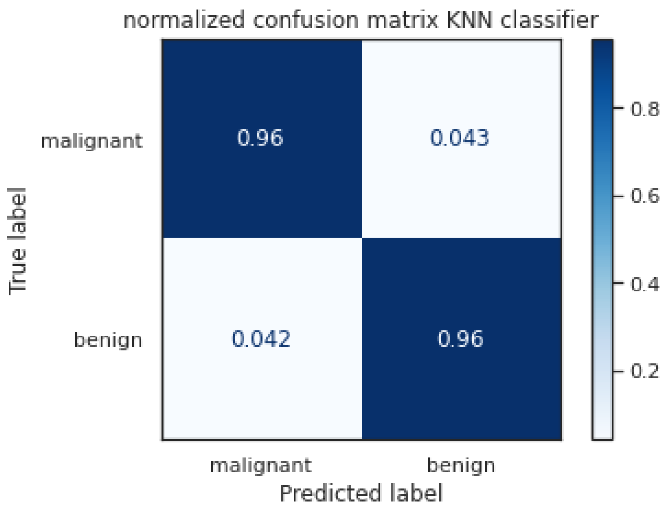
KERNEL RBF SVM

Accuracy	0.974
F1 score	0.978
Precision	0.982
Recall	0.974



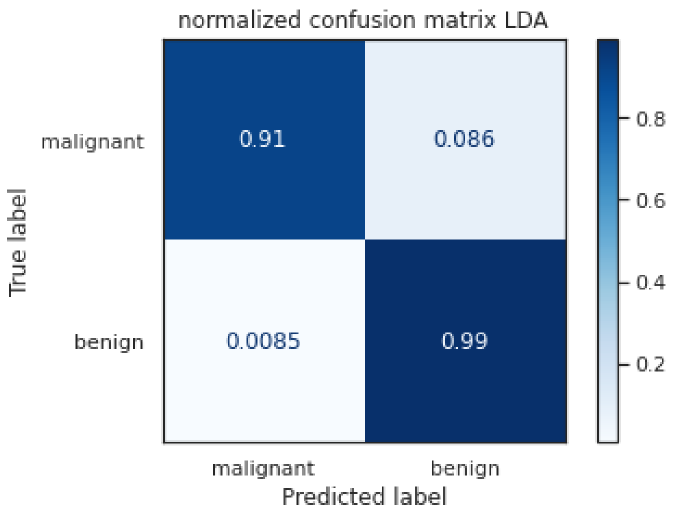
K-NEAREST NEIGHBORS

Accuracy	0.957
F1 score	0.965
Precision	0.974
Recall	0.957



LINEAR DISCRIMINANT ANALYSIS

Accuracy	0.962
F1 score	0.970
Precision	0.951
Recall	0.991



6 CONCLUSIONS

Data preprocessing makes algorithms perform slightly better than when trained with default data, but not in an evident way: in particular, PCA doesn't increase the model scores, probably due to the fact that the features dimension of the original data, cleaned from correlated features, was not enough to significantly perturb performances. Oversampling, instead, slightly improve them, given the dataset imbalance. However, all the models achieve really good results, but this is also due to the dataset which has been constructed for the same purpose of the analysis.