POLITECNICO DI TORINO

Mathematics in Machine Learning

Dry Bean Dataset Analysis

Academic year 2021/2022

Student: Alessandro Desole

Student ID: 278936

Professors: Francesco Vaccarino, Mauro Gasparini

Contents

Introduction

The main aim of this study is to exploit some supervised machine learning methods to detect the membership category of dry beans, taking into account the features such as form, shape, type, and structure by the market situation.

Dry beans (*Phaseolus vulgaris)* belong to the diverse *Fabaceae* family, sometimes referred to as *Leguminosae,* and they are the most important and the most produced pulse over the world.

Dry bean is originally from America, while there is a wide genetic diversity in the world since, in the 15-16th centuries, they were transported to Europe and Africa and quickly spreading to the rest of the globe.

Dry beans are known as an economical source of protein. In addition, they are low in fat and a rich source of fiber and other important nutrients. Also, they are associated with a myriad of environmental and human health benefits, such as improved soil fertility and reduced risk of chronic disease. Finally, beans are also the perfect food to improve and/or promote glycemic control.

Figure 1 - Different types of dry beans

1. Environment

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 other data visualization libraries (*matplotlib* and *seaborn*).

Project's code has been developed by using Jupyter Notebook App (a browser-based tool) and has been uploaded to GitHub at https://github.com/alessandrodesole/dry-bean-dataset-analysis

2. Exploratory data analysis

The dataset used in this study is the *Dry Bean Dataset* from the UCI machine learning repository, available at the following link. It was created by Murat Koklu and Ilker Ali Ozkan of the Selcuk University in 2020. They took images of **13611 grains** of 7 different registered dry beans with a high-resolution camera. Then a computer vision system was developed to perform segmentation and feature extraction stages on bean pictures, in order to obtain uniform seed classification. From the process described before, **16 features** were obtained from the grains.

It is possible to notice that no missing values are recorded for any of the samples.

The number of duplicate rows detected is equal to **68** (0.5% of the dataset). As duplicate rows we mean that rows containing identical values for each feature and belonging to the same category. This could be due to things like data entry error.

By going deeper the duplicate rows, we can observe that all of these come from a unique class: indeed the whole set of these entries describe ***Horoz*** grains. These rows will be dropped and not considered in this study.

These basic dataset statistics are summarized in the table below.

Number of observations (after duplicate removal)	Number of missing cells	Number of duplicate rows	Number of features	Number of classes
13543	0	68	16	7

Table 1 - Basic dataset statistics

2.1. Class distribution

As said before, 7 different types of dry beans are described in the dataset. Class details are collected and sorted according to descend frequency order in the table on left. A graphical representation of class distribution can be observed in Figure 2 on right.

Class name	Count	Frequency (%)
Dermason	3546	26.2%
Sira	2636	19.5%
Seker	2027	15.0%
Horoz	1860	13.7%
Cali	1630	12.0%
Barbunya	1322	9.8%
Bombay	522	3.9%

Table 2 - Class distribution

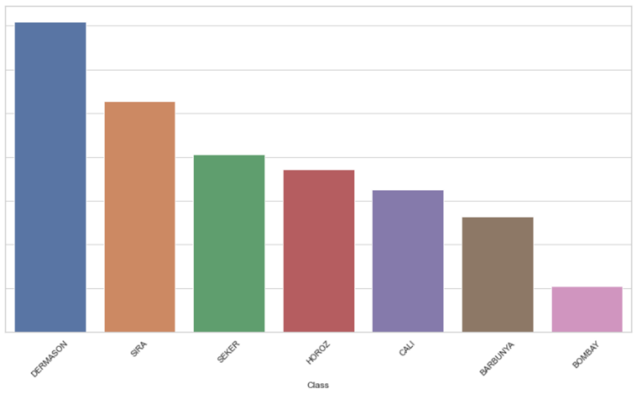


Figure 2 - Class distribution countplot

The main aim of data is to discriminate the membership category of each grain. Thus, it is a ***multiclass classification*** problem on a relatively unbalanced dataset.

2.2. Feature analysis

As previously mentioned, **16 features** (12 dimensions and 4 shape forms) were obtained from the grains. All the features are numeric, some of that are real, others integer and are summarized in the table below.

🟧 Dimensions 🟦 Shape forms

Number	Feature	Type	Description
1	Area (A)	Integer	The area of a bean zone and the number of pixels within its boundaries
2	Perimeter (P)	Real	Bean circumference is defined as the length of its border
3	Major axis length (L)	Real	The distance between the ends of the longest line that can be drawn from a bean
4	Minor axis length (l)	Real	The longest line that can be drawn from the bean while standing perpendicular to the main axis
5	Aspect ratio (K)	Real	Defines the relationship between L and l
6	Eccentricity (Ec)	Real	Eccentricity of the ellipse having the same moments as the region
7	Convex area (C)	Integer	Number of pixels in the smallest convex polygon that can contain the area of a bean seed
8	Equivalent diameter (Ed)	Real	The diameter of a circle having the same area as a bean seed area
9	Extent (Ex)	Real	The ratio of the pixels in the bounding box to the bean area
10	Solidity (S)	Real	Also known as convexity. The ratio of the pixels in the convex shell to those found in beans
11	Roundness (R)	Real	Calculated with the following formula: (4piA)/(P^2)
12	Compactness (CO)	Real	Measures the roundness of an object: Ed/L
13	ShapeFactor1 (SF1)	Real	Shape factor 1
14	ShapeFactor2 (SF2)	Real	Shape factor 2
15	ShapeFactor3 (SF3)	Real	Shape factor 3
16	ShapeFactor4 (SF4)	Real	Shape factor 4

Table 1 - Feature description

2.3. Statistics on dataset

In order to analyze all the features in detail, the following statistics are defined and computed on each feature. Then they are summarized in Table 3.

- ***Count***: number of occurrences for each feature
- ***Mean*** $(\bar{x})$: the average of the values of each variable in the dataset, which is the sum of those values divided by the number of values

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- ***Unbiased standard deviation*** $(\boldsymbol{s})$: the estimated value of the standard deviation of a population of values

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

- ***Minimum*** $(\boldsymbol{min})$: the smallest value
- ***First quartile***[1] $(\boldsymbol{25\%})$: the middle number that falls between the smallest value of the dataset and the median
- ***Second quartile*** $(\boldsymbol{50\%})$: also called median, is the value separating the higher half from the lower half of values
- ***Third quartile*** $(\boldsymbol{75\%})$: the central point that lies between the median and the highest number of the distribution
- ***Maximum*** $(\boldsymbol{max})$: the largest value

	count	mean	std	min	25%	50%	75%	max
Area	13543.0	53048.460385	29392.438324	20420.000000	36282.500000	44580.000000	61382.000000	254616.000000
Perimeter	13543.0	854.993406	214.722684	524.736000	703.230000	793.896000	977.146500	1985.370000
MajorAxisLength	13543.0	319.895602	85.809260	183.601165	253.086806	296.404589	376.312489	738.860153
MinorAxisLength	13543.0	202.365321	45.051632	122.512653	175.886357	192.491117	217.245403	460.198497
AspectRation	13543.0	1.581075	0.245245	1.024868	1.430662	1.549860	1.703916	2.430306
Eccentricity	13543.0	0.750315	0.091858	0.218951	0.715144	0.763997	0.809671	0.911423
ConvexArea	13543.0	53767.986709	29844.248525	20684.000000	36673.000000	45122.000000	62360.000000	263261.000000
EquivDiameter	13543.0	253.034094	59.307709	161.243764	214.933277	238.245711	279.560351	569.374358
Extent	13543.0	0.749829	0.048939	0.555315	0.718735	0.759903	0.786849	0.866195
Solidity	13543.0	0.987152	0.004650	0.919246	0.985678	0.988288	0.990019	0.994677
roundness	13543.0	0.873671	0.059393	0.489618	0.833410	0.883490	0.917031	0.990685
Compactness	13543.0	0.800352	0.061464	0.640577	0.763228	0.801514	0.834470	0.987303
ShapeFactor1	13543.0	0.006561	0.001130	0.002778	0.005893	0.006643	0.007270	0.010451
ShapeFactor2	13543.0	0.001719	0.000595	0.000564	0.001158	0.001700	0.002173	0.003665
ShapeFactor3	13543.0	0.644341	0.098653	0.410339	0.582517	0.642424	0.696341	0.974767
ShapeFactor4	13543.0	0.995078	0.004347	0.947687	0.993720	0.996393	0.997891	0.999733

Table 2 - Dataset statistics

[1] *Quartile*: statistical term that describes a division of observation into four defined intervals based on the values of the data

The computation of the previous statistics reveals as the 16 attributes describing dry beans present values very different from each other: in fact, values lie in very different ranges, making hardly impossible a comparison among them and suggesting that a feature scaling process is needed.

In general, if a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected. In particular, many machine learning algorithms suffer from features that are not centered in zero and/or do not have a similar variance among them. For example, K-Nearest Neighbor algorithm would compute distances that would be highly dependent on the variance of each feature, leading to some attributes getting more importance than others during the prediction.

2.4. Boxplots

In descriptive statistics, a **boxplot** is a method for graphically demonstrating the locality, spread and skewness groups of numerical data through their quartiles. So, a boxplot shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable.

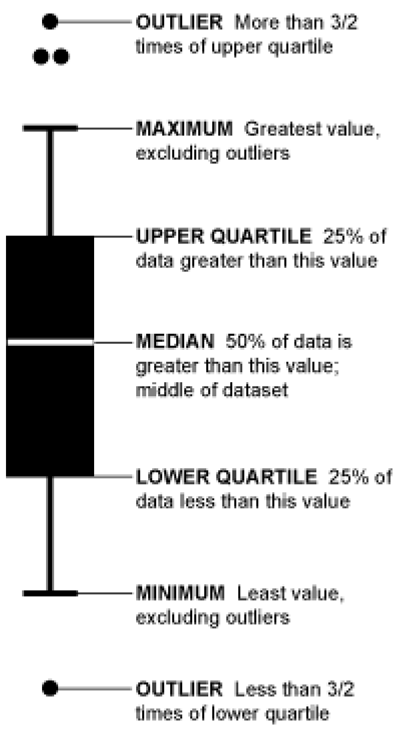

OUTLIER More than 3/2 times of upper quartile
MAXIMUM Greatest value, excluding outliers
UPPER QUARTILE 25% of data greater than this value
MEDIAN 50% of data is greater than this value; middle of dataset
LOWER QUARTILE 25% of data less than this value
MINIMUM Least value, excluding outliers
OUTLIER Less than 3/2 times of lower quartile

Figure 3 - Different parts of a boxplot

In detail, a **boxplot** is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell you about the outliers and what their values are. It can also show if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.

Now we define the **Interquartile range** (IQR) as is the distance between the upper (Q_3) and lower (Q_1) quartiles: every sample placed beyond $Q_1 - 1.5 \times IQR$ and $Q_3 + 1.5 \times IQR$ is considered an ***outlier*** and represented by a dot. The lines extending parallel from the boxes are known as the ***whiskers***, which are used to indicate variability outside the upper and lower quartiles.

Boxplots representing the feature distribution of Dry Bean Dataset can be observed in the next page.

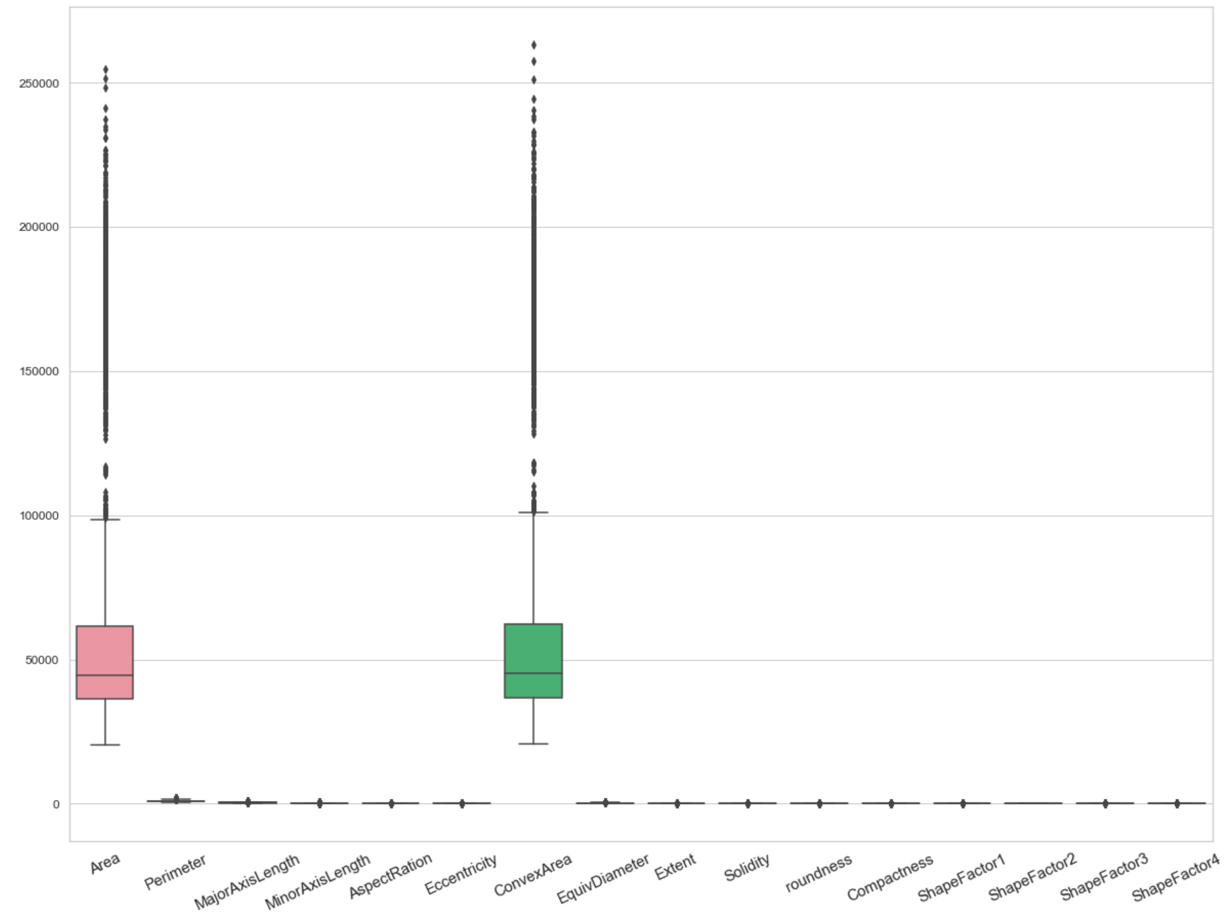


Figure 4 - Boxplot about the original feature distribution

As we can see, features have very different scales and are hardly impossible to compare out of the box without a scaling method. To solve this issue, the two following strategies will be applied to transform data:

- **Min-Max normalization**: for every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between this range. The transformation is given by the following formula:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

  where x_{min} is the minimum value and x_{max} is the maximum value on each column.

- **Standardization**: the values of each feature are centered around mean with a unit standard deviation. It means that mean and standard deviation of standard scores will be 0 and 1 respectively.

  For each column:

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

  where μ_x is the sample mean and σ_x is the sample standard deviation of feature x.

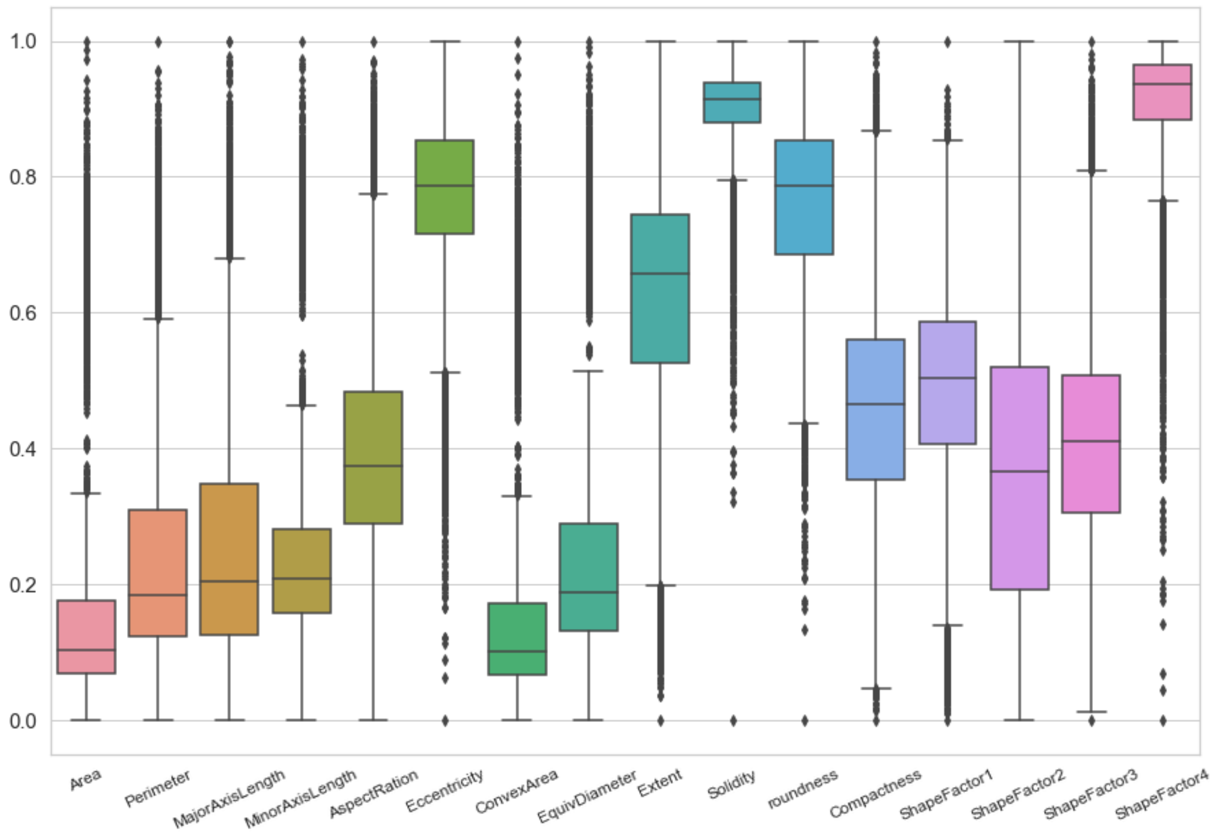


Figure 5 - Boxplots of features scaled according to Min-Max normalization

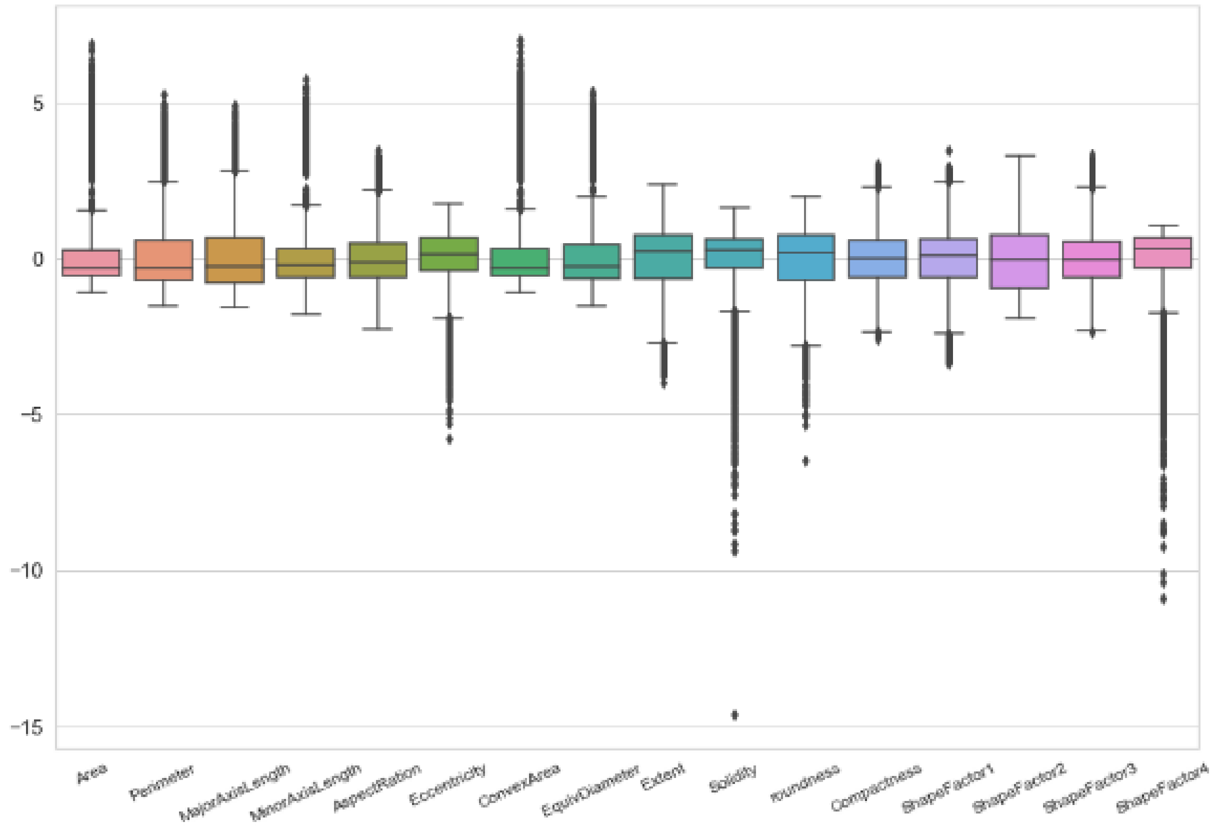


Figure 6 - Boxplots of features scaled according to standardization

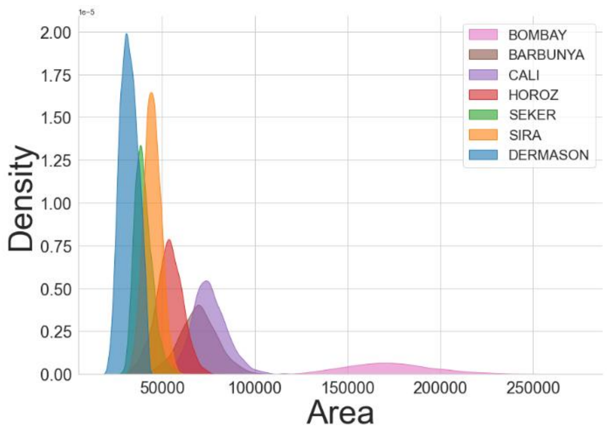
Min-max normalization has one fairly significant downside: it does not handle outliers very well. In fact, a point to be noted that unlike **normalization**, **standardization** doesn't have a bounding range. It's also not influenced by maximum and minimum values in the dataset so if data contains ***outliers***, **standardization** is good to go.

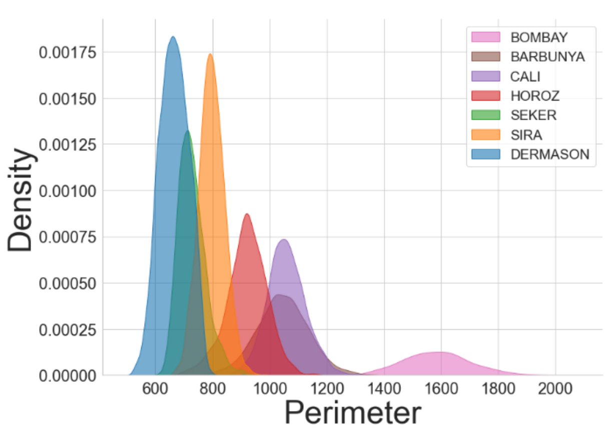
The boxplots in the previous page highlight the presence of many ***outliers***[2], so in this work **standardization** scaling method will be applied.

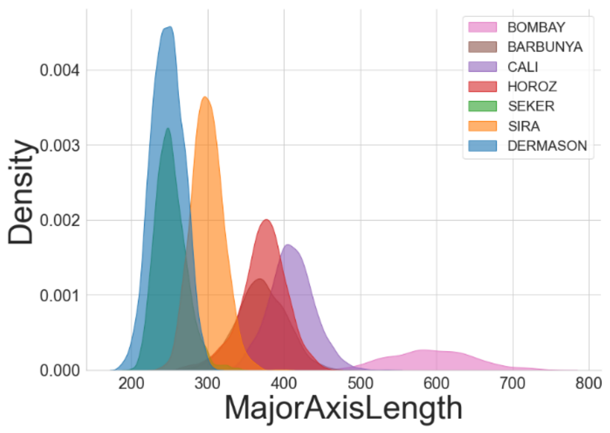
2.5 Feature distribution

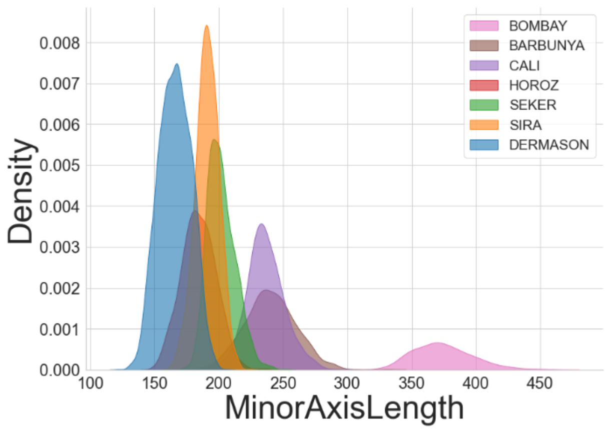
Feature distribution is very important, because machine learning models learn from data. So, it helps in understanding what kind of feature we are dealing with, and what values we can expect this feature to have.

Through the following plots, it is possible to observe how the distribution of features varies as the labels change. In this way we can analyze if there is a relationship between the trend of a certain feature and the change of class.

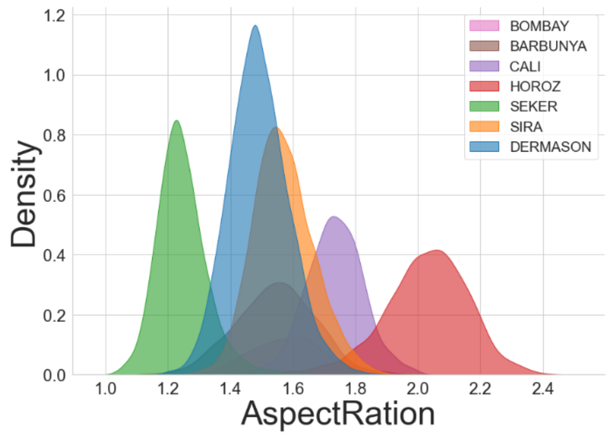


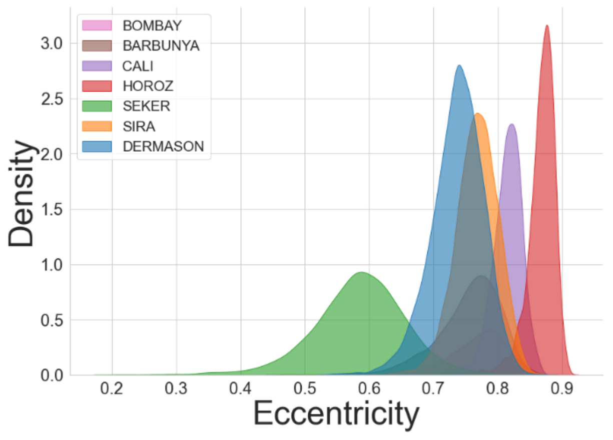


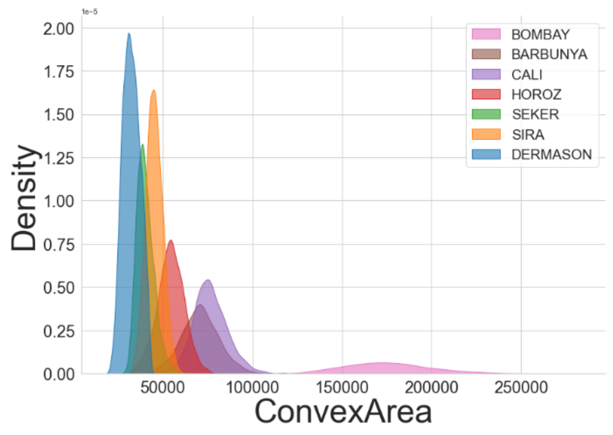


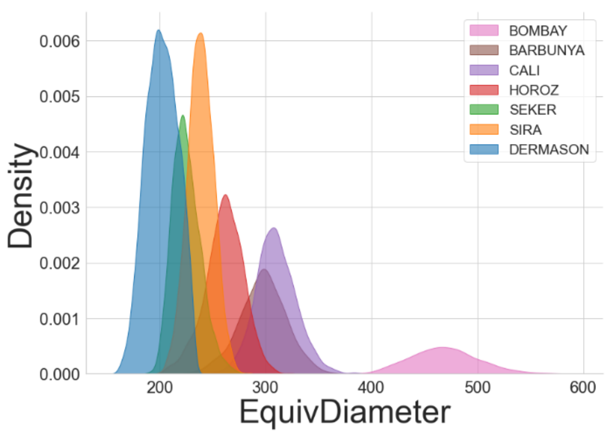


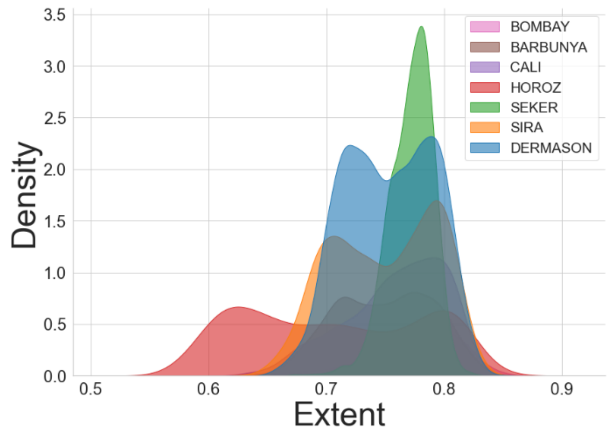
[2] ***Outlier***: a data point that differs significantly from other observations

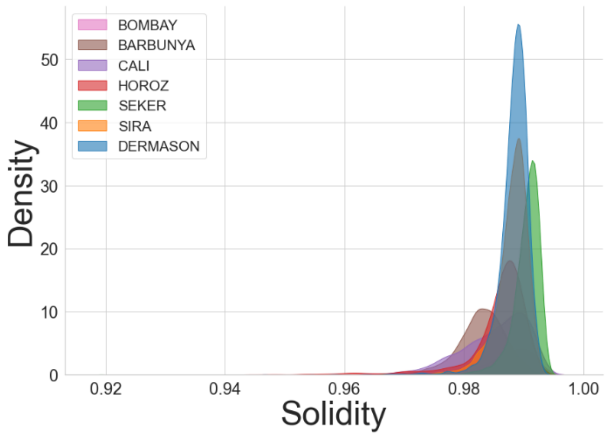

Density
AspectRation
BOMBAY
BARBUNYA
CALI
HOROZ
SEKER
SIRA
DERMASON

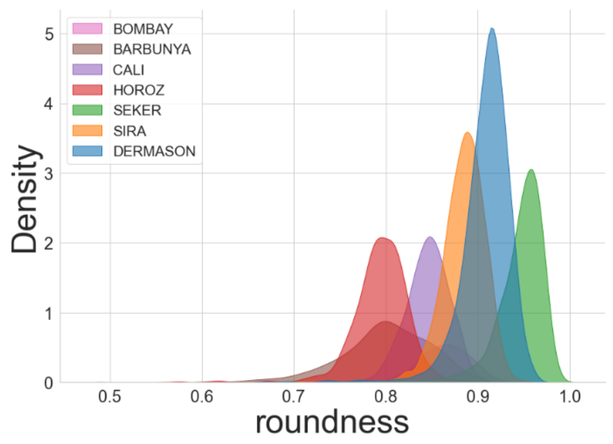

Density
Eccentricity
BOMBAY
BARBUNYA
CALI
HOROZ
SEKER
SIRA
DERMASON

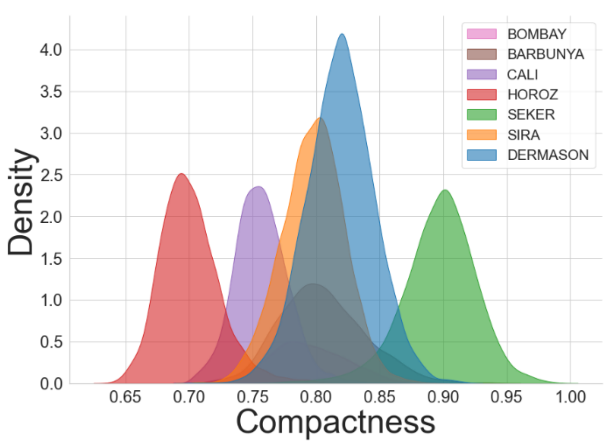

1e-5
Density
ConvexArea
BOMBAY
BARBUNYA
CALI
HOROZ
SEKER
SIRA
DERMASON


Density
EquivDiameter
BOMBAY
BARBUNYA
CALI
HOROZ
SEKER
SIRA
DERMASON


Density
Extent
BOMBAY
BARBUNYA
CALI
HOROZ
SEKER
SIRA
DERMASON


Density
Solidity
BOMBAY
BARBUNYA
CALI
HOROZ
SEKER
SIRA
DERMASON


Density
roundness
BOMBAY
BARBUNYA
CALI
HOROZ
SEKER
SIRA
DERMASON


Density
Compactness
BOMBAY
BARBUNYA
CALI
HOROZ
SEKER
SIRA
DERMASON

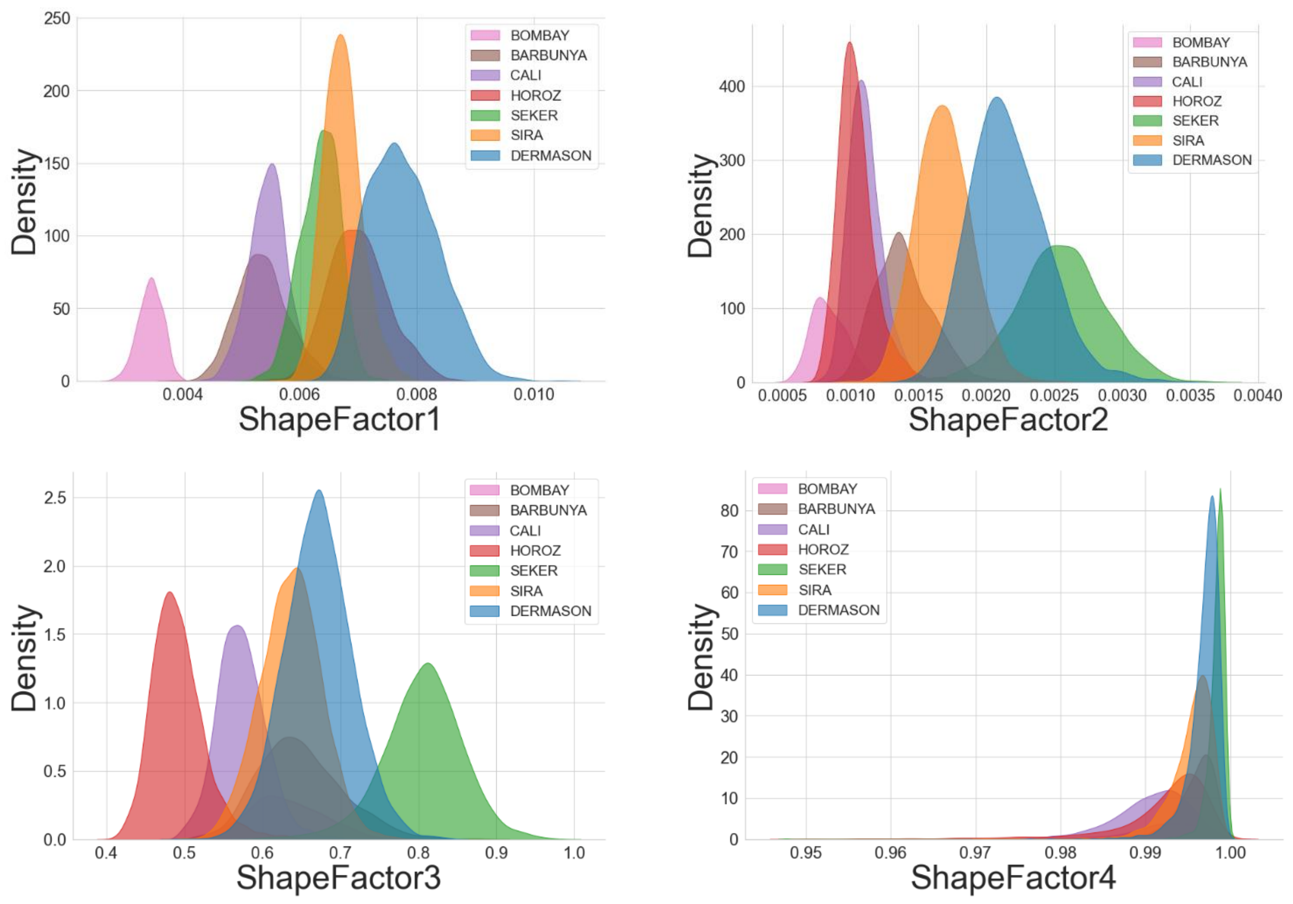


Figure 7 - Kernel density estimate plot about dry bean features

To represent the feature distribution, Kernel density estimation (***KDE***) approach has been used. It is a non-parametric way to estimate the probability density function of a random variable. ***KDE*** is a fundamental data smoothing problem where inferences about the population are made, based on a finite data sample.

From independent and identically distributed samples we are interested in estimating the shape of this function ***f***. So, its kernel density estimator is:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Where ***K*** is the kernel function (a non-negative function that in this case is represented by the Gaussian kernel $K(x; h) \propto \exp\left(-\frac{x^2}{2h^2}\right)$) and $h > 0$ is a smoothing parameter called the bandwidth. This parameter has the role of controlling the tradeoff between bias and variance in the result.

The aim of this graphical representation is to detect the features able to distinguish better each class. As instance, we can observe that Bombay type is easily identifiable from the others by its area, perimeter, major and minor axis length. At the same time, we can say that aspect ratio, roundness, compactness and some shape factors are the best features to classify samples correctly, due to the distribution curves are better recognizable.

Another density estimation method consists of using histograms, but while a histogram aims to approximate the underlying probability density function that generated the data by binning and counting observations, **KDE** presents a different solution to the same problem. Rather than using discrete bins, a **KDE** plot smooths the observations with a Gaussian kernel, producing a continuous density estimate.

Relative to a histogram, **KDE** can produce a plot that is less cluttered and more interpretable, especially when drawing multiple distributions (like in this case). But it has the potential to introduce distortions if the underlying distribution is bounded or not smooth. Like a histogram, the quality of the representation also depends on the selection of good smoothing parameters.

2.6 Correlation between features

It can be useful in this data analysis and modeling study to better understand the relationships between variables. The statistical relationship between two variables is referred to as their ***correlation***.

A correlation could be positive, meaning both variables move in the same direction, or negative, meaning that when one variable's value increases, the other variables' values decrease. Correlation can also be neutral or zero, meaning that the variables are unrelated.

The performance of some algorithms can deteriorate if two or more variables are tightly related, called multicollinearity. An example is linear regression, where one of the offending correlated variables should be removed in order to improve the skill of the model.

Another benefit from spotting correlation between features is that the same information may be encoded with less attributes, and this could lead to simpler final models. Indeed, some methods suffer from high dimensional datasets (especially distance-based ones with a few numbers 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.

The most familiar measure of dependence between two quantities is the Pearson's correlation coefficient ($\boldsymbol{\rho_{X,Y}}$), that is a measure of linear correlation between two variables. Its value lies between -1 and +1, -1 indicating total negative linear correlation, 0 indicating no linear correlation and 1 indicating total positive linear correlation. Furthermore, $\boldsymbol{\rho_{X,Y}}$ is invariant under separate changes in location and scale of the two variables, implying that for a linear function the angle to the x-axis does not affect $\boldsymbol{\rho_{X,Y}}$.

To calculate $\boldsymbol{\rho_{X,Y}}$ for two variables X and Y, one divides the covariance of X and Y by the product of their standard deviations. So,

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

where, $Cov(X,Y)$ is the Covariance and σ_X and σ_Y are respectively the standard deviations of X and Y.

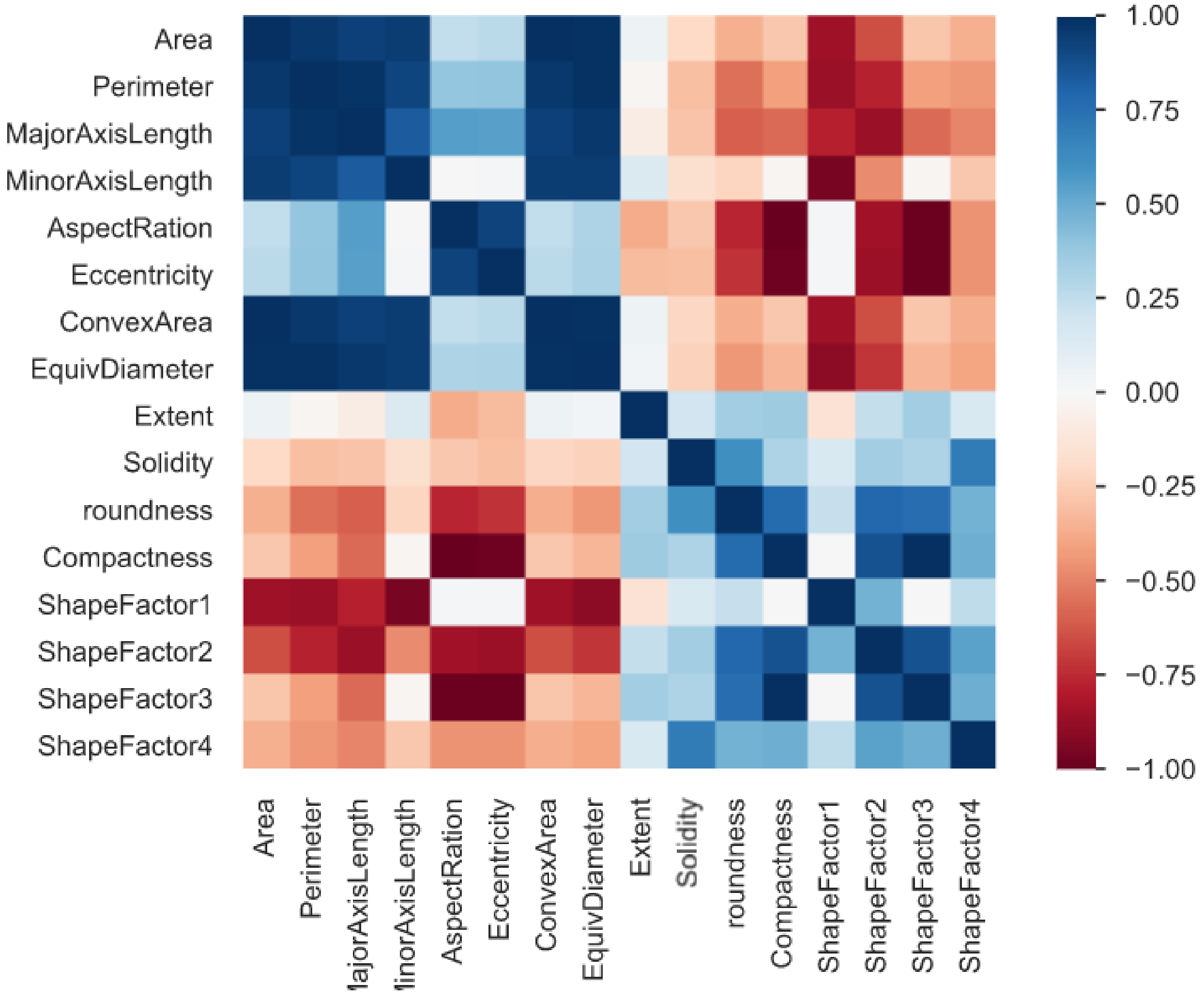


Figure 8 - Correlation matrix by means of the Pearson's coefficient for all feature pairs

The correlation matrix in Figure 8 above reveals as some features show high correlations with each other. The darker the shade of blue, the more positively two features are correlated (e.g., Area and Convex area or Area and Equivalent diameter), the darker the shade of red, the more negatively two variables are correlated (e.g., Compactness and Aspect Ratio or Shape factor 1 and Minor axis length).

For further evidence of the correlation between these variables we can also plot some of their scatter plot which shows us the distribution in space of these pairs.

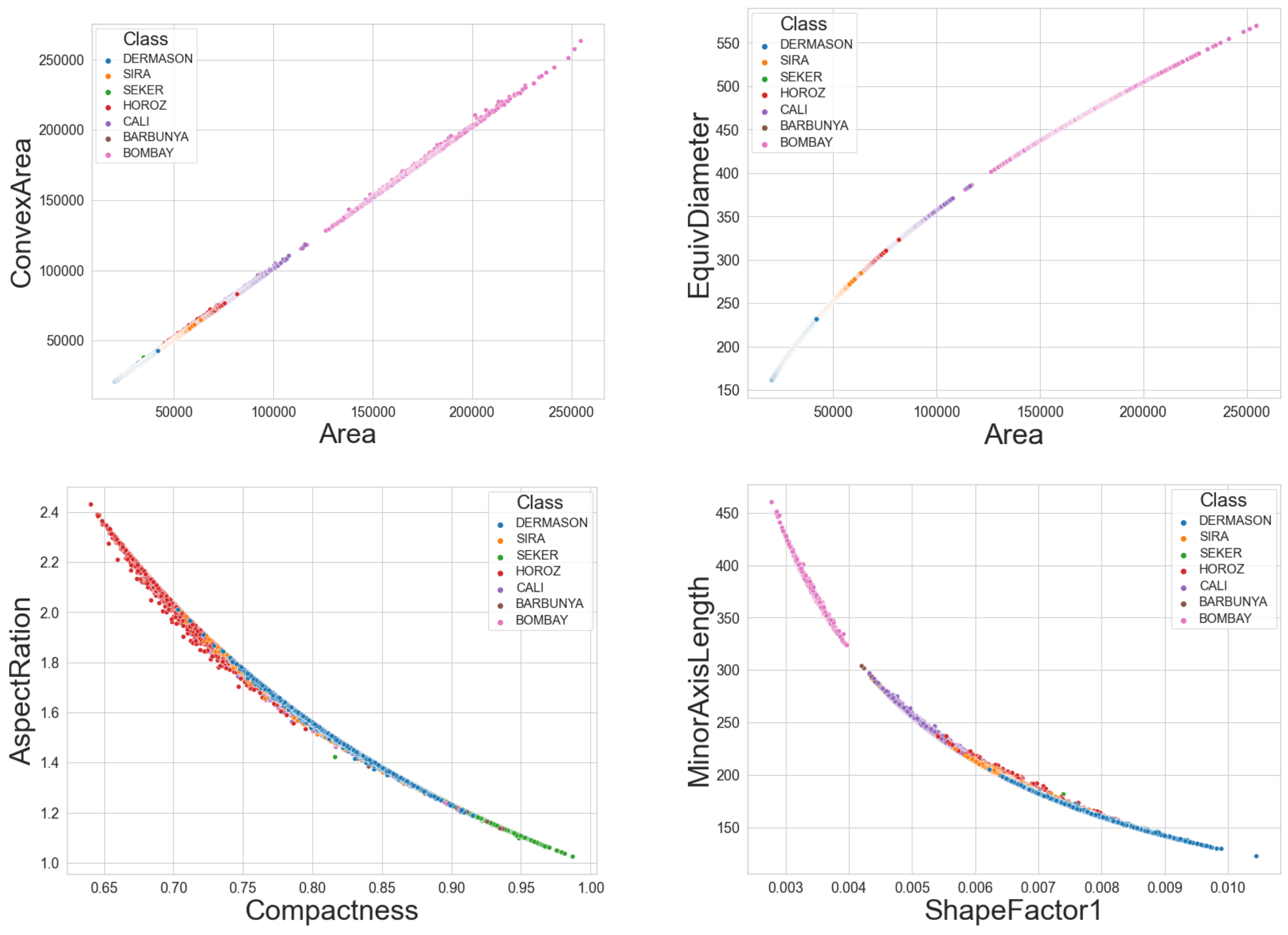


Figure 9 - Scatter plots of highly correlated features

Figure 8 also shows as some features are not linearly correlated each other and they are represented by the squares tending more toward white color. As instance, Equivalent Diameter and Extent are examples of non-linearly correlated features. In the same way Eccentricity and Minor axis length are.

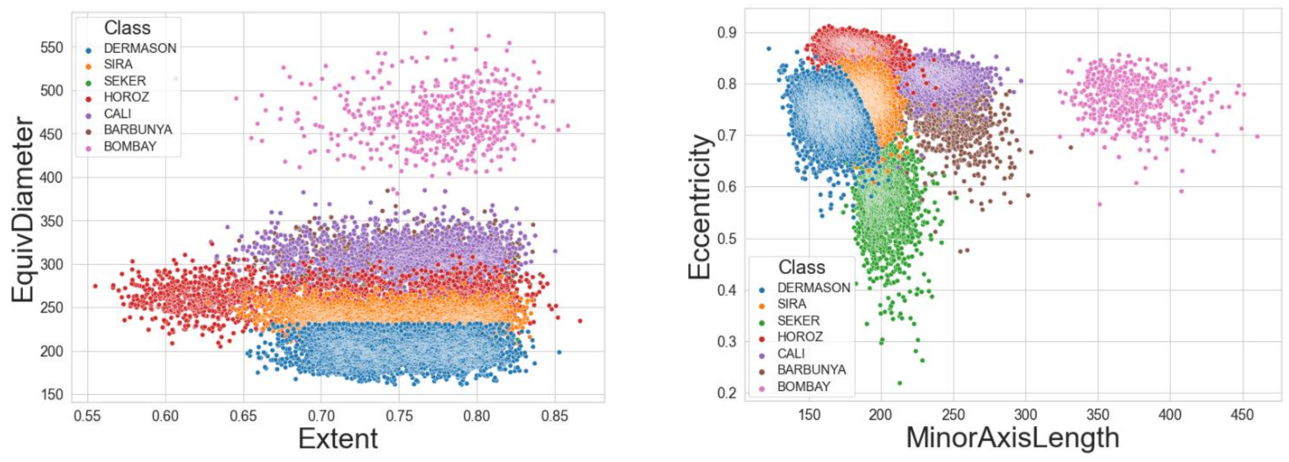


Figure 10 - Scatter plots of non-linearly correlated features

3. Data preprocessing

Data preprocessing describes any type of processing performed on raw data to prepare it for another data processing procedure. This operation transforms the data into a format that is more easily and effectively processed in machine learning.

3.1 Categorical data encoding

Encoding categorical data is a process of converting categorical data into integer format so that the data with converted categorical values can be provided to the different models.

So, an integer value is associated with each type of dry bean as shown in the following table:

Class name	Code
Dermason	0
Sira	1
Seker	2
Horoz	3
Cali	4
Barbunya	5
Bombay	6

Table 3 – Class encoding table

3.2 Train-test split and validation of dataset

The train-test split procedure is a fast and easy technique used to estimate the performance of machine learning algorithms. The procedure involves taking the dataset and dividing it into two subsets:

- **Train dataset**: used to fit the machine learning models.
- **Test dataset**: used to evaluate the fit machine learning models, so predictions are made on it and compared to the expected values.

In this case, the dry bean dataset is divided in training and test set with the proportion 3:1, that means that 75% of data (10157 samples) is used to train the models, while the remaining 25% (3386 samples) represents the test set.

Due to the large imbalance in the distribution of the target classes, to maintain the **IID assumption** about dataset instances and to ensure that relative class frequencies is approximately preserved, the **stratified sampling technique** has been used to split data into train and test datasets.

The following table shows the distribution of samples among classes after the splitting phase.

Class name	No. samples in train set	No. samples in test set
Dermason	2659	887
Sira	1977	659
Seker	1520	507
Horoz	1395	465
Cali	1222	408
Barbunya	992	330
Bombay	392	130

Table 4 - Class distribution after splitting process

To do more effectively the data splitting process, **stratified K-fold cross-validation** strategy has been applied. Machine Learning is all about generalization and this method is useful for building a more generalized model because it allows to evaluate a model's performance and perform hyperparameter tuning.

The general procedure is as follows:

1. Shuffle the dataset randomly
2. Split the dataset into K groups
3. For each unique group:
    a. Take the group as a hold out or test data set
    b. Take the remaining groups as a training data set
    c. Fit a model on the training set and evaluate it on the test set
    d. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once. Stratified means that each subset contains the same underlying distribution of target labels.

In this study, a value of K=5 has been considered.

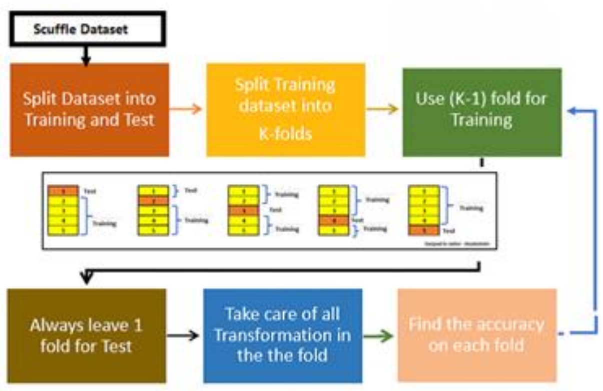

Scuffle Dataset
Split Dataset into Training and Test
Split Training dataset into K-folds
Use (K-1) fold for Training
Test
Training
Always leave 1 fold for Test
Take care of all Transformation in the the fold
Find the accuracy on each fold

Figure 11 - K-fold cross-validation scheme

It is important to highlight that as for the test set, also the validation sets involved in the cross-validation process are not affected by preprocessing steps in order to keep the verification phase as agnostic and independent as possible. This is very important as it avoids leaking of any information from the test set into the training process, which would be formally incorrect.

3.3 Feature scaling

As discussed in paragraph 2.3., the ultimate goal of performing **standardization** is to bring down all the features to a common scale without distorting the differences in the range of the values, so **standardization** technique is applied to data due to the fact that it is much less affected by outliers with respect to normalization.

Note that the statistics (mean and standard deviation) are computed on the training set only, then both datasets are 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 by definition.

3.4 Dimensionality reduction

Dimensionality reduction is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data. In practice, it is the process of reducing the number of random variables under consideration in the dataset, by obtaining a set of principal variables. In fact, more input features often make a predictive modeling task more challenging to model, more generally referred to as the **curse of dimensionality**.

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. 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 (PCA) have been exploited.

3.4.1 Feature selection

On top of feature transformation, the data preprocessing phase often includes a feature selection step, where the attributes from the given dataset are carefully analyzed and finally selected to feed the machine learning algorithms.

As shown in point 2.6., many of the 16 features available in the dataset are linearly correlated among them. Based on the Pearson's coefficients previously reported, a total of 7 features have been dropped in this step having a $\rho \geq 0.92$ with some other predictor, resulting in a smaller dataset of 9 features total.

In particular, the following features have been removed: *Perimeter*, *MajorAxisLength*, *MinorAxisLength*, *Eccentricity*, *EquivDiameter*, *ConvexArea*, *ShapeFactor3*.

3.4.2 Principal Component Analysis (PCA)

Another way to obtain a lower dimensional dataset when dealing with multicollinearity among features, is by applying a **Principal Component Analysis** (PCA) to our data.

PCA is an **unsupervised learning technique** that performs a linear transformation (by means of an orthogonal matrix) of the original space such that the new basis found has dimensions (features) that are sorted from largest to smallest possible variance. So, in addition to reducing the dimensionality, PCA increases interpretability but at the same time minimizes information loss.

The intuition behind this approach lies in the fact that if the direction of maximum variance in the original space is not directly captured by the features in the dataset, then that direction might be used to construct a new feature that has a larger variance, hence probably encoding more information. Therefore, the following minimization problem is performed for finding the direction of maximum variance, corresponding to the first principal component and calculated as follow:

$$X \in \mathbb{R}^{n \times d}, \text{ dataset centered in zero}$$

$$\Sigma := \frac{X^T X}{n-1}, \text{ sample covariance matrix}$$

$$\text{Find} \qquad \vec{z_1} := a_1 \vec{e_1} + \cdots + a_d \vec{e_d} \qquad \text{subjected to}$$

$$\vec{z_1} = argmax_{\vec{z_1}} \vec{z_1}^T \Sigma \vec{z_1} \quad \text{s.t.} : \|\vec{z_1}\| = 1$$

Recursively, all other dimensions are then computed in the same way, additionally forcing them to be orthogonal to the previous dimensions found. The new features z_i are denoted as **principal components** (PCs).

The above optimization problem can be easily computed through the eigenvectors of the covariance matrix Σ that spots correlation among features and it is a $n \times n$ symmetric matrix (where n is equal to the original number of dimensions having covariances associated with all possible pairs of variables as entries. For example, for a 3-dimensional dataset with 3 variables x, y, and z, the covariance matrix is a 3×3 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 5 - Example of covariance matrix for 3 features

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

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

The covariance of two variables (e.g., x and y) is computed as:

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

In the previous equations, n represents the number of samples in the dataset.

Due to the fact that Σ is a symmetric positive semidefinite matrix, it is always possible to diagonalize it by means of an orthogonal matrix $\boldsymbol{P}$ (made by eigenvectors of Σ, that are orthogonal and uncorrelated with each other), and obtain the resulting similar matrix Λ:

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

Where λ_i are the eigenvalues of Σ, or equivalently the variances of the new features found, and are computed by solving the following equation in λ with d degrees:

$$det|\Sigma - \lambda I| = 0$$

Where I is the identity matrix.

Then, it is possible to compute the eigenvectors v_i by using the following equation:

$$\Sigma \cdot v_i = \lambda_i \cdot v_i$$

Where $1 \leq i \leq d$.

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.

Finally, the data from the original dataset are projected from the original axes to the ones represented by the principal components.

In this study, PCA has been performed on the Dry Beans' dataset to deal with the multicollinearity problem and reduce the number of dimensions. Figure 12 shows how the variance has been redistributed on the new features extracted: the bars represent the proportion of variance explained by each principal component, while the brown line instead is the cumulative amount of proportion of variance explained.

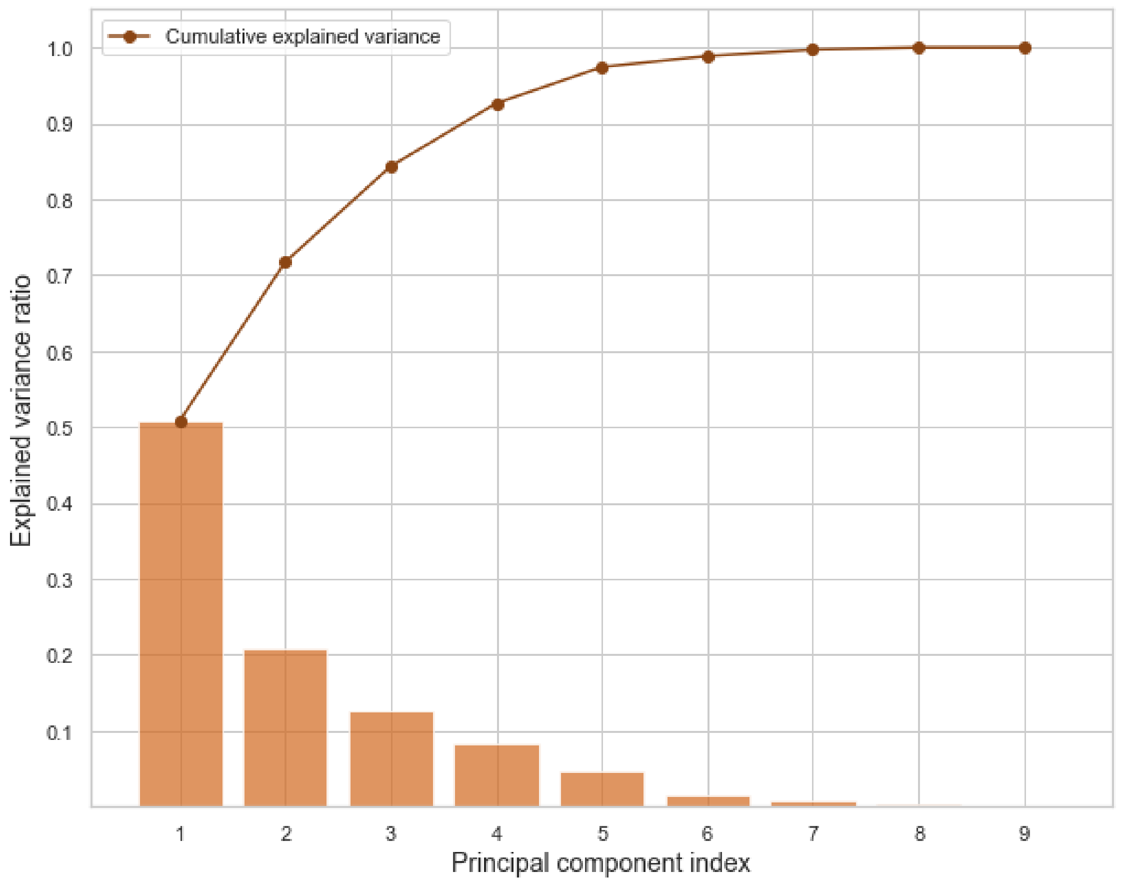


Figure 12 - Explained variance ratio of each principle component, together with the cumulative variance explained as more dimensions are considered

This graph is useful to take decision regarding the number of components to keep. The following table report some interesting values for the number of components.

Number of Principal Components	**Cumulative Explained Variance**
1	50.8%
2	71.7%
3	84.3%
4	92.6%
5	97.4%
6	98.8%
7	99.7%
8	99.9%
9	100%

The first 6 PCs have been kept, to explain a total variance of about 99%.

Note that PCA is applied based only on the training data in order to avoid any leaking the information of test data. Remind also that, due to how PCA is defined, scaling features to similar variances before applying the algorithm is essential.

3.5 Dataset balancing

Imbalanced datasets are those where there is a severe skew in the class distribution. This bias in the training dataset can influence many machine learning algorithms, leading some to ignore the minority classes entirely.

The two main approaches to randomly resampling an imbalanced dataset are to delete examples from the majority classes, called **undersampling**, and to duplicate examples from the minority classes, called **oversampling**.

In the specific case of dry beans dataset, the distribution of data among the 7 different categories reveals as we are dealing with an imbalanced dataset. Techniques to handle this issue are as follows:

- **Random oversampling**: it includes selecting random examples from the minority class with replacement and supplementing the training data with multiple copies of this instance, hence it is possible that a single instance may be selected multiple times.
- **Random undersampling**: This method seeks to randomly select and remove samples from the majority class, consequently reducing the number of examples in the majority class in the transformed data.
- **Synthetic Minority Oversampling Technique** (**SMOTE**): this is an oversampling technique that allows us to generate synthetic samples for the minority categories. The idea is based on the K-Nearest Neighbors algorithm. Specifically, a random example from the minority class is first chosen. Then *k* of the nearest neighbors for that example are found. A randomly selected neighbor is chosen, and a synthetic example is created at a randomly selected point between the two examples in feature space.

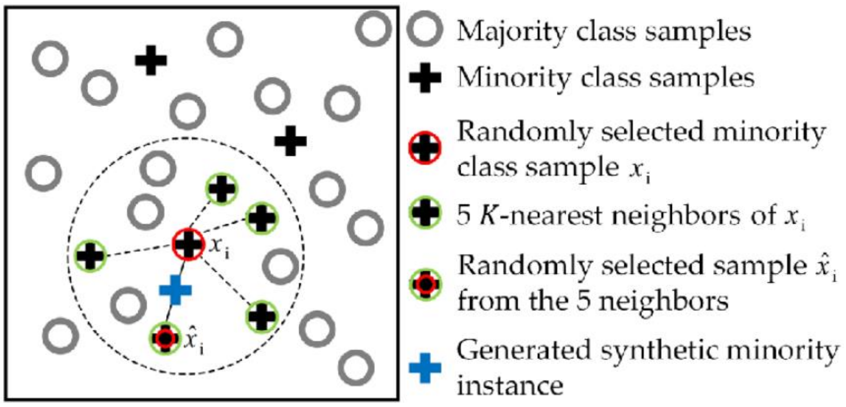

x_i
$\hat{x}_i$
Majority class samples
Minority class samples
Randomly selected minority class sample x_i
5 K-nearest neighbors of x_i
Randomly selected sample $\hat{x}_i$ from the 5 neighbors
Generated synthetic minority instance

Figure 13 - SMOTE graphical representation for a binary classification task

- **Cluster Centroids undersampling:** it is a method that under samples the majority class by replacing a cluster of majority samples by the cluster centroid of a KMeans algorithm. This algorithm keeps N majority samples by fitting the KMeans algorithm with N cluster to the majority class and using the coordinates of the N cluster centroids as the new majority samples.

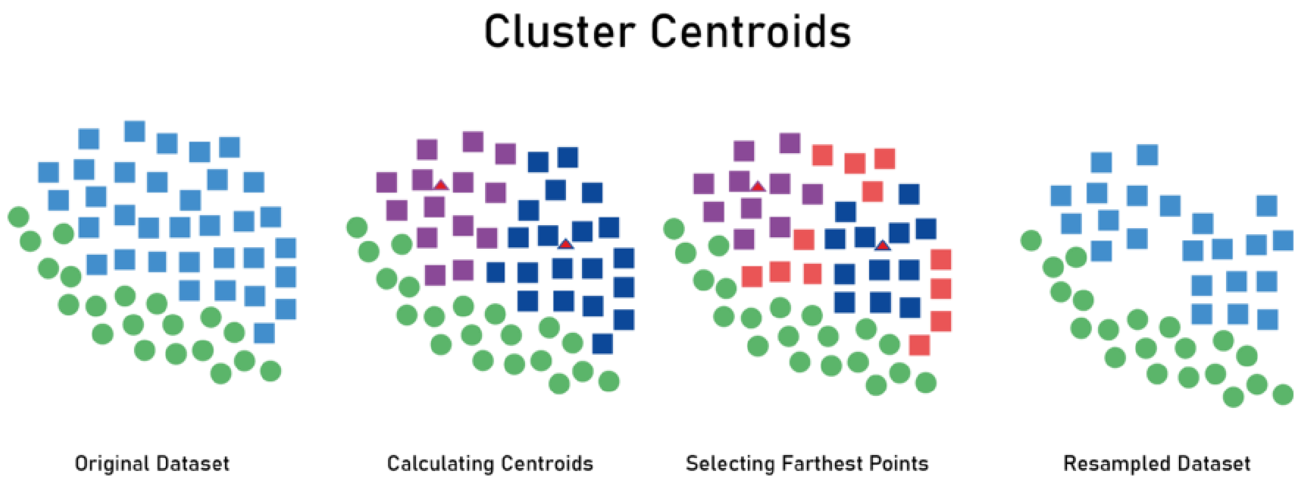


Figure 14 - Example of Cluster Centroids undersampling

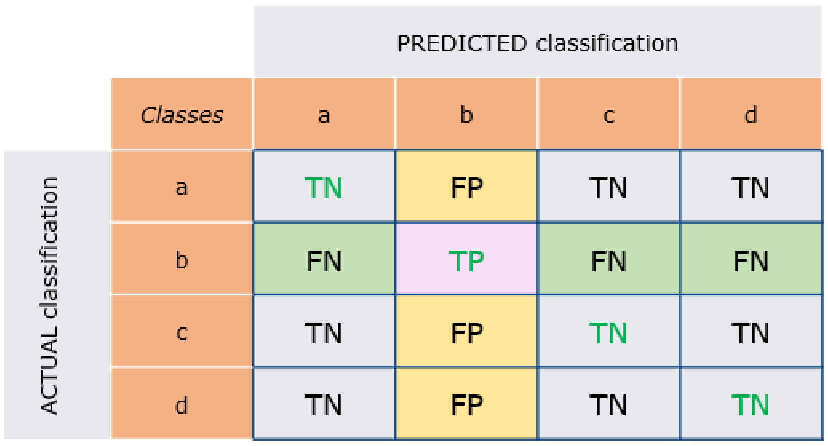
4. Evaluation metrics

An evaluation metric quantifies the performance of a predictive model. For classification problems, metrics involve comparing the expected class label to the predicted class label or interpreting the predicted probabilities for the class labels for the problem.

First of all, it is necessary to define some basic terms useful to understand metrics:

- *TP* = True positive, the only correctly classified units for our class
- *TN* = True negative, all the other tiles
- *FP* = False positive, the wrongly classified elements on the column of the class
- *FN* = False negative, the wrongly classified elements on the row of the class

To better understand these concepts, a confusion matrix (a tabular way of visualizing the performance of your prediction model) for an example of multiclass classification problem is shown.



		PREDICTED classification			
	Classes	a	b	c	d
ACTUAL classification	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN

Figure 15 - Example of confusion matrix for a multiclass classification problem. Class b is the reference in this case

In this study, the following metrics have been used:

- **Accuracy** $= \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} = \frac{TP+TN}{TP+TN+FP+FN}$, that represents the probability that the model prediction is correct. Accuracy is one of the most popular metrics in multi-class classification and it is directly computed from the confusion matrix, by summing the elements on the main diagonal.

- **Macro-Precision:** it is calculated by taking the average precision for each predicted class. It is computed as follows:

$$Precision_k = \frac{TP_k}{TP_k + FP_k}$$

  Where k is a generic class

$$MacroAveragePrecision = \frac{\sum_{k=1}^{K} Precision_k}{K}$$

  Where K is the total number of classes

- **Macro-Recall**: it is calculated by taking the average recall for each actual class. It is computed as follows:

$$Recall_k = \frac{TP_k}{TP_k + FN_k}$$

  Where k is a generic class

$$MacroAverageRecall = \frac{\sum_{k=1}^{K} Recall_k}{K}$$

  Where K is the total number of classes

- **Macro F1-score:** it is the harmonic mean of Macro-Precision and Macro-Recall

$$Macro\ F1\ score = 2 * \left(\frac{MacroAveragePrecision * MacroAverageRecall}{MacroAveragePrecision^{-1} + MacroAverageRecall^{-1}}\right)$$

Hence, the Macro approach considers all the classes as basic elements of the calculation: each class has the same weight in the average, so that there is no distinction between highly and poorly populated classes. High Macro-F1 values indicate that the algorithm has good performance on all the classes, whereas low Macro-F1 values refers to poorly predicted classes.

- **Micro-Precision**:

$$MicroAveragePrecision = \frac{\sum_{k=1}^{K} TP_k}{\sum_{k=1}^{K} Total\ Column_k} = \frac{\sum_{k=1}^{K} TP_k}{Grand\ Total}$$

- **Micro-Recall**:

$$MicroAverageRecall = \frac{\sum_{k=1}^{K} TP_k}{\sum_{k=1}^{K} Total\ Row_k} = \frac{\sum_{k=1}^{K} TP_k}{Grand\ Total}$$

As we can see, Micro-Precision and Micro-Recall are just the same values, therefore the Micro-Average F1-Score is just the same as well (the harmonic mean of two equal values is just the value).

- **Micro F1-score**:

$$Micro\ Average\ F1 = \frac{\sum_{k=1}^{K} TP_k}{Grand\ Total}$$

Taking a look to the formula, we may see that Micro-Average F1-Score is just equal to Accuracy. Therefore, the Accuracy gives different importance to different classes, based on their frequency in the dataset.

- **Weighted-Precision, Weighted-Recall and Weighted average F1 score**: the weighted average will be calculated by multiplying each score by the number of occurrences of each class and dividing by the total number of samples. So, with weighted averaging, the output average would have accounted for the contribution of each class as weighted by the number of examples of that given class.

5. Classification algorithms

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset and then classifies new observations into a number of classes.

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 performance differences. The best configuration is selected by comparing the different metrics, principally based on the *weighted average F1 score* because Macro and Micro approaches on unbalance datasets may return a high value even if the minority class is not correctly classified.

Each algorithm is trained on dataset with different preprocessing techniques combined after Standardization and feature selection based on Pearson's coefficient:

- No PCA and resampling (BASIC)
- Principal Component Analysis (PCA)
- Random oversampling (with and without PCA)
- SMOTE (with and without PCA)
- Random undersampling (with and without PCA)
- Cluster Centroids undersampling (with and without PCA)

The algorithms considered are:

- K-Nearest Neighbors (KNN)
- Decision Trees
- Random forest
- Multinomial Logistic Regression (One-vs-rest approach)
- Support Vector Machine (SVM) (One-vs-one approach)

One-vs-rest (**OvR**) and One-vs-one (**OvO**) are heuristic methods for using binary classification algorithms (such as SVM and logistic regression) for multi-class classification.

OvR strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency, one advantage of this approach is its interpretability.

OvO strategy consists in fitting one classifier per class pair. At prediction time, the class which received the most votes is selected. Since it requires to fit more classifiers, this method is usually slower than OvR, due to its greater complexity.

Each experiment performed is replicable thanks to fixed random states, that control the randomization of the algorithms.

5.1 K-Nearest Neighbors (KNN)

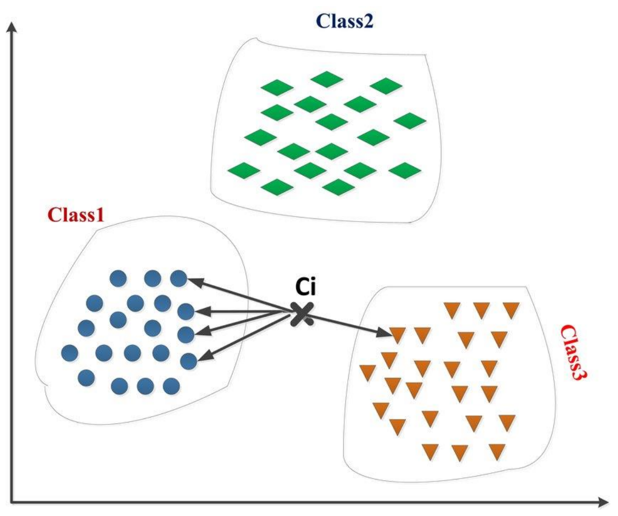


Figure 16 - Graphical representation of KNN

The K-Nearest Neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

It is a non-parametric and lazy learning method. Non-parametric means there is no assumption for underlying data distribution and so the model structure is determined from the dataset. Lazy algorithm means it does not need any training data points for model generation, because all training data are used in the testing phase. This makes training faster and testing phase slower and costlier.

So, *KNN* does not attempt to construct a general internal model, but simply stores instances of the training data and classification is based on the distance metric and is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned to data class which has the most representatives within the nearest neighbors of the point.

Between two points

$$X = (x_1, x_2, \dots, x_n) \text{ and } Y = (y_1, y_2, \dots, y_n) \in \mathbb{R}^n$$

The metric used in this study is the Minkowski distance defined as follows:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Where p is and integer chosen between 1 and 2. When $p = 1$, this is equivalent to use the Manhattan distance:

$$D(X,Y) = \sum_{i=1}^{n} |x_i - y_i|$$

When $p = 2$, this is equivalent to use the Euclidean distance:

$$D(X,Y) = \sqrt{\sum_{n=1}^{i} (x_i - y_i)^2}$$

Specifically, in this analysis, hyperparameters tuned are the following:

- The number of neighbors considered (**K**). If K is too small, the model becomes sensitive to outliers, while if K is too large, the model may become too much general and classification could not perform well. Values considered are 8, 10, 15, 20, 30, 50
- Parameter ***p***, chosen between 1 and 2
- The **weight** function used in prediction. In this study will be applied the *uniform* criterium, where all points in each neighborhood are weighted equally and *distance* criterium, where closer neighbors of a query point will have a greater influence than neighbors which are further away.

Pipeline	Accuracy	Macro-Precision	Macro-Recall	Macro-F1	Micro-Precision	Micro-Recall	Micro-F1	Weighted-Precision	Weighted-Recall	Weighted-F1	K	*p*	Weight
BASIC	0.9202	0.9351	0.9301	0.9323	0.9202	0.9202	0.9202	0.9213	0.9202	0.9205	15	1	distance
PCA	0.9250	0.9386	0.9340	0.9361	0.9250	0.9250	0.9250	0.9256	0.9250	0.9251	10	2	distance
RO	0.9200	0.9328	0.9322	0.9321	0.9200	0.9200	0.9200	0.9213	0.9200	0.9202	30	1	uniform
RO+PCA	0.9200	0.9312	0.9328	0.9317	0.9200	0.9200	0.9200	0.9215	0.9200	0.9205	30	1	distance
SMOTE	0.9202	0.9326	0.9338	0.9329	0.9202	0.9202	0.9202	0.9217	0.9202	0.9205	20	2	distance
SMOTE+PCA	0.9182	0.9310	0.9319	0.9311	0.9182	0.9182	0.9182	0.9198	0.9182	0.9185	15	2	uniform
RU	0.9120	0.9249	0.9236	0.9240	0.9120	0.9120	0.9120	0.9128	0.9120	0.9121	8	2	uniform
RU+PCA	0.9138	0.9273	0.9251	0.9258	0.9138	0.9138	0.9138	0.9149	0.9138	0.9140	10	1	uniform
CCU	0.9173	0.9302	0.9301	0.9296	0.9173	0.9173	0.9173	0.9191	0.9173	0.9175	8	2	distance
CCU+PCA	0.9185	0.9330	0.9330	0.9322	0.9185	0.9185	0.9185	0.9208	0.9185	0.9187	10	2	distance

Table 6 - Summary table for KNN

Best configuration found and scores are highlighted in green, worst ones in red.

Weighted-F1 Score trend among the different preprocessing techniques can be observed also in the following line plot.

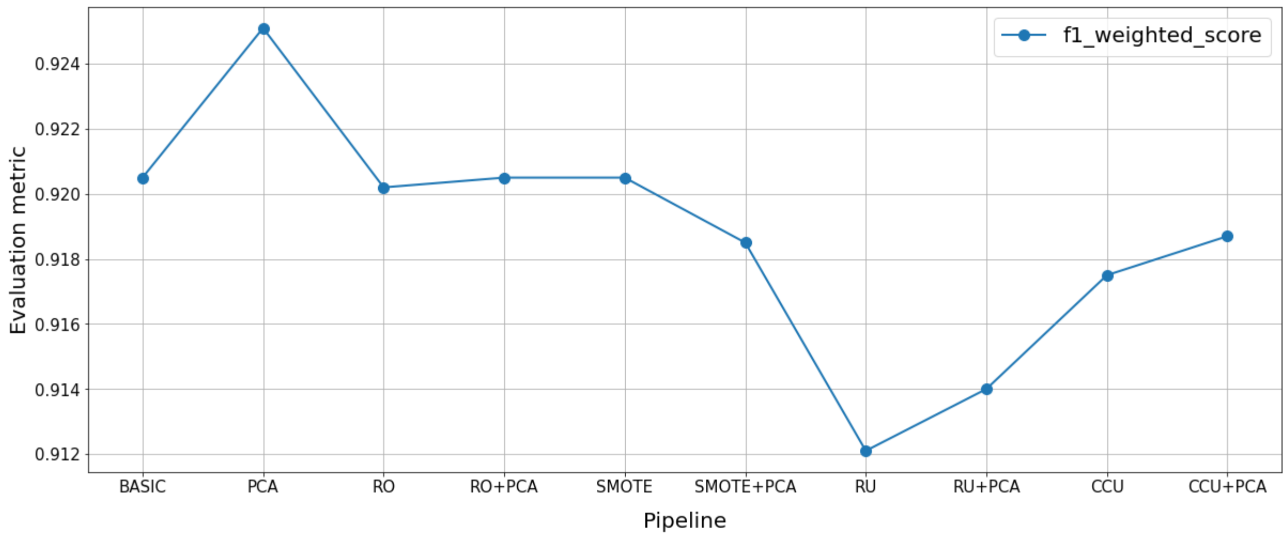
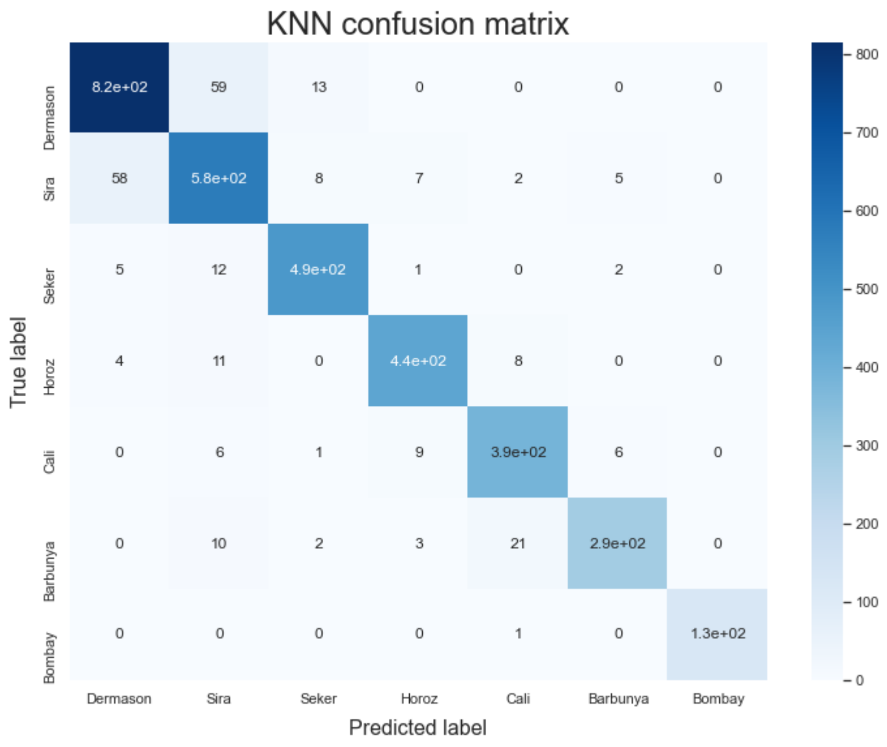


Figure 17 – KNN Weighted-F1-Score for each preprocessing method

As expected, best results are obtained by applying PCA. 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. On the other side, worst results are provided by using undersampling techniques. So, we can say that a smaller amount of data affects the prediction negatively.

Confusion matrix about the best configuration found is shown below.



True label \ Predicted label	Dermason	Sira	Seker	Horoz	Cali	Barbunya	Bombay
Dermason	8.2e+02	59	13	0	0	0	0
Sira	58	5.8e+02	8	7	2	5	0
Seker	5	12	4.9e+02	1	0	2	0
Horoz	4	11	0	4.4e+02	8	0	0
Cali	0	6	1	9	3.9e+02	6	0
Barbunya	0	10	2	3	21	2.9e+02	0
Bombay	0	0	0	0	1	0	1.3e+02

Figure 18 - KNN Confusion matrix

5.2 Decision Trees

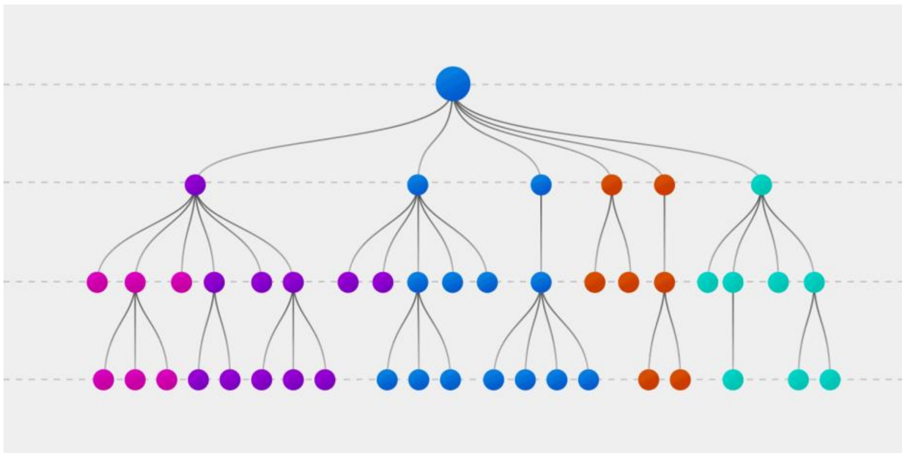


Figure 19 - Example of Decision Tree schema

Decision Trees are a non-parametric supervised learning method. They are the most intuitive and interpretable machine learning model, which predict the target label by learning simple decision rules inferred from the data features.

At each step, a Decision Tree picks the feature that best divides the space into different labels, by means of an impurity measure. In this study, the following criteria are applied:

GINI:

$$GINI(t) = \sum_{i=1}^{C} p_t(k)\big(1 - p_t(k)\big) = 1 - \sum_{i=1}^{C} p_t(k)^2$$

Entropy:

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

Where $p_t(k)$ is the frequency of class k appearing in node t, with C the total number of classes. The lower the measure, the less impure the node is.

A full tree is then constructed on the training data and will be used at classification time for predicting the target label. To avoid overfitting, 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. Note how trees cannot handle missing values, but do not suffer from feature scaling since each attribute is treated independently.

Results of the experiments are collected in the table below.

Pipeline	Accuracy	Macro-Precision	Macro-Recall	Macro-F1	Micro-Precision	Micro-Recall	Micro-F1	Weighted-Precision	Weighted-Recall	Weighted-F1	Criterion	Class weight
BASIC	0.9090	0.9245	0.9205	0.9224	0.9090	0.9090	0.9090	0.9093	0.9090	0.9090	Gini	None
PCA	0.8960	0.9112	0.9108	0.9108	0.8960	0.8960	0.8960	0.8956	0.8960	0.8957	Entropy	Balanced
RO	0.9084	0.9216	0.9218	0.9208	0.9084	0.9084	0.9084	0.9121	0.9084	0.9090	Gini	Balanced
RO+PCA	0.8996	0.9117	0.9127	0.9120	0.8996	0.8996	0.8996	0.9007	0.8996	0.8998	Gini	Balanced
SMOTE	0.9028	0.9180	0.9175	0.9174	0.9028	0.9028	0.9028	0.9047	0.9028	0.9033	Entropy	None
SMOTE+PCA	0.9046	0.9182	0.9186	0.9183	0.9046	0.9046	0.9046	0.9051	0.9046	0.9047	Gini	None
RU	0.8969	0.9111	0.9114	0.9109	0.8969	0.8969	0.8969	0.8993	0.8969	0.8975	Gini	Balanced
RU+PCA	0.8624	0.8782	0.8833	0.8801	0.8624	0.8624	0.8624	0.8652	0.8624	0.8630	Gini	None
CCU	0.8860	0.9016	0.8988	0.8998	0.8860	0.8860	0.8860	0.8871	0.8860	0.8817	Gini	None
CCU+PCA	0.8573	0.8765	0.8801	0.8773	0.8573	0.8573	0.8573	0.8609	0.8573	0.8577	Gini	Balanced

Table 7 - Summary table for Decision Trees

Best configuration found and scores are highlighted in green, worst ones in red.

Weighted-F1 Score trend among the different preprocessing techniques can be observed also in the following line plot.

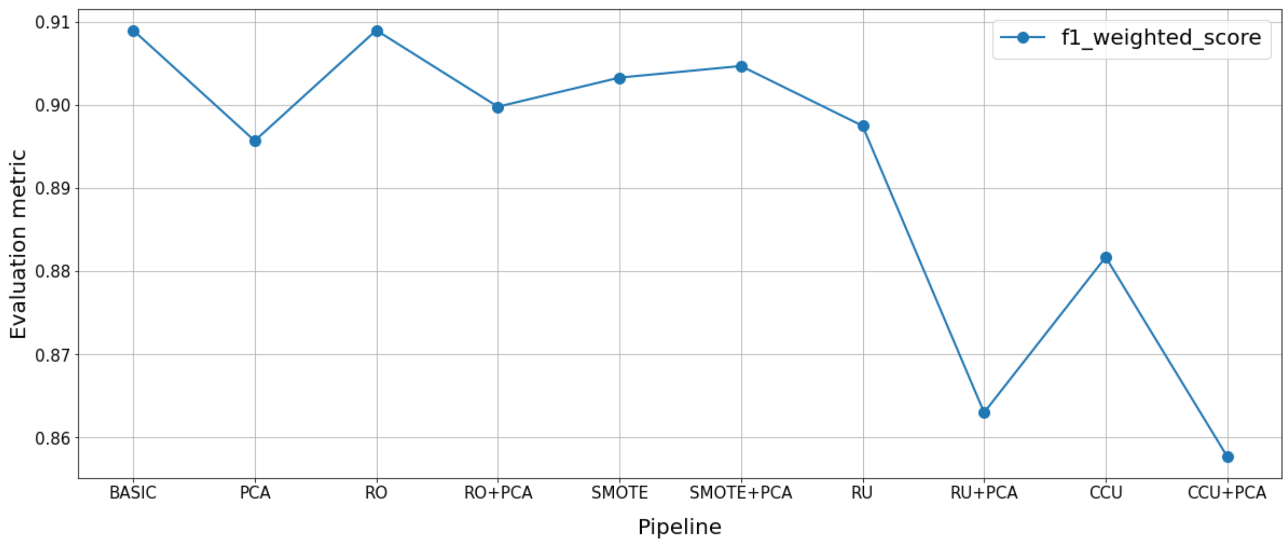
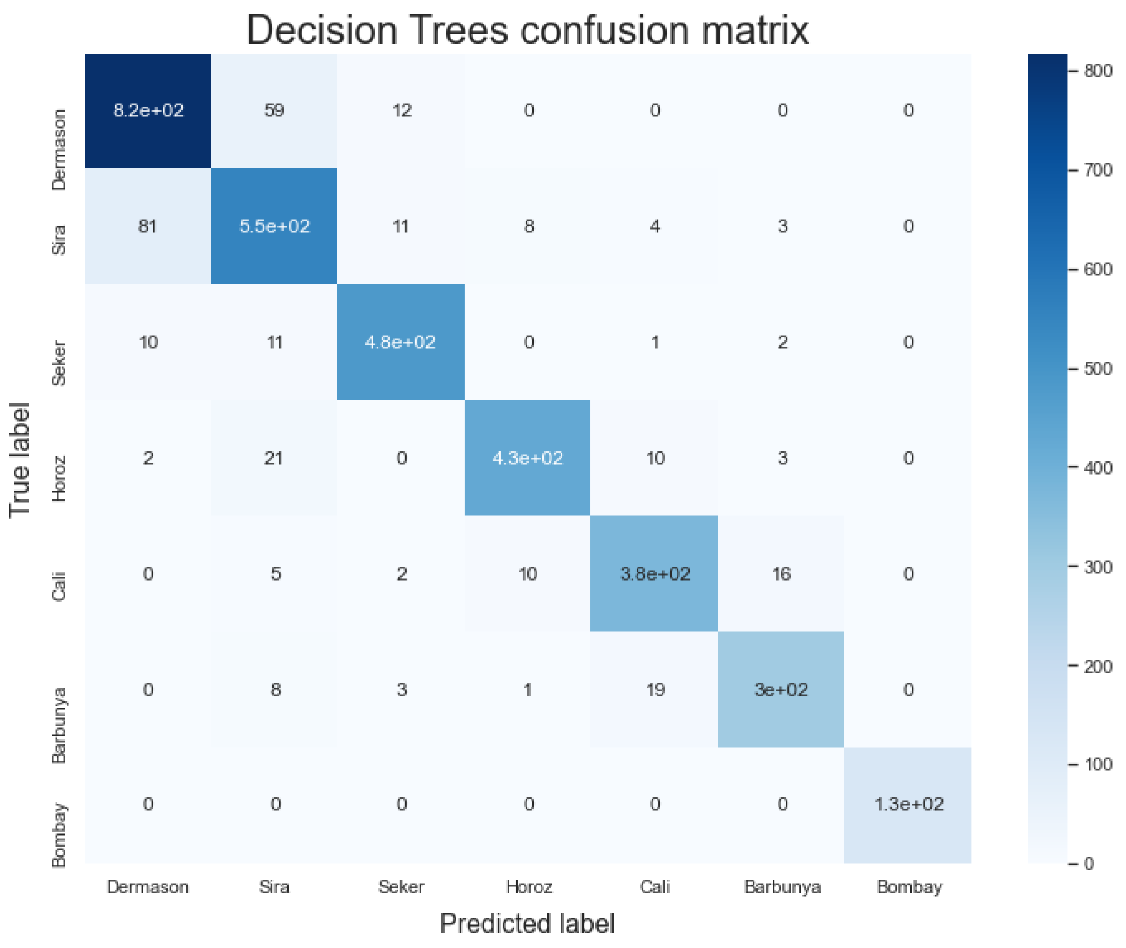


Figure 20 - Decision Trees Weighted-F1-Score for each preprocessing method

It is possible to notice how the worst results are obtained by using PCA. This is easily explained by the fact that PCA aims to reduce the feature space and this is counterproductive for classification.

Confusion matrix about the best configuration found is shown below.



True label \ Predicted label	Dermason	Sira	Seker	Horoz	Cali	Barbunya	Bombay
Dermason	8.2e+02	59	12	0	0	0	0
Sira	81	5.5e+02	11	8	4	3	0
Seker	10	11	4.8e+02	0	1	2	0
Horoz	2	21	0	4.3e+02	10	3	0
Cali	0	5	2	10	3.8e+02	16	0
Barbunya	0	8	3	1	19	3e+02	0
Bombay	0	0	0	0	0	0	1.3e+02

Figure 21 - Decision Trees confusion matrix

Decision trees are however considered weak learners in the majority of cases, and ensemble techniques such as *bagging* (Random forests) or *boosting* are in practice applied on them in order to reach higher performances and more robust solutions.

5.3 Random Forests

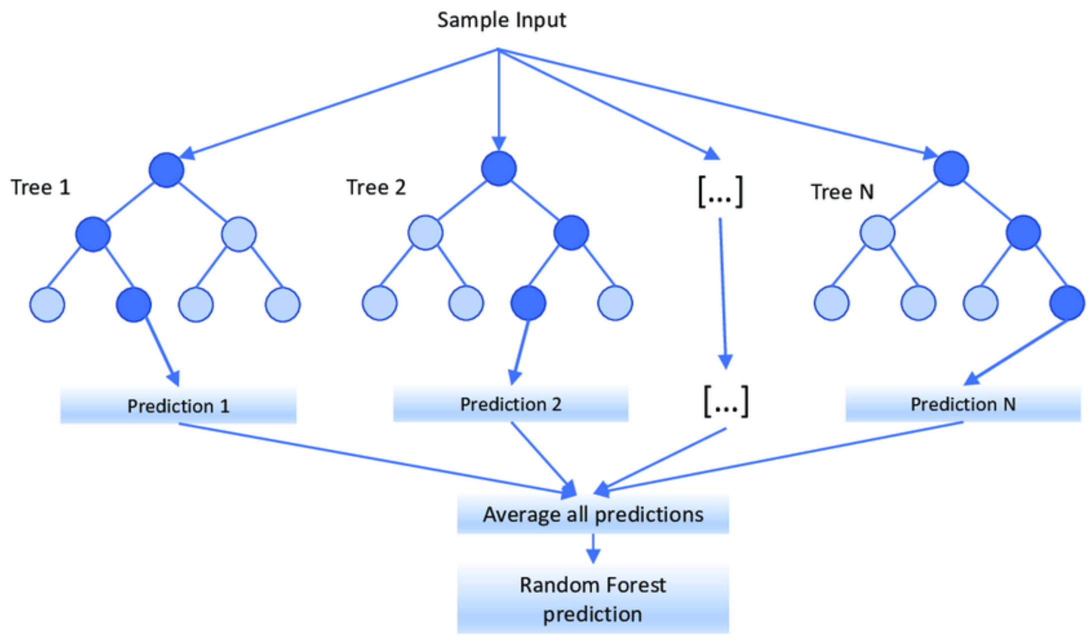


Figure 22 - Random Forest schema

Random forest is an ensemble learning method consisting of many decisions trees. It uses **bagging** and **feature randomness** when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

Let's go into detail about these two methods:

Bagging (Bootstrap Aggregation) is the ensemble learning method that is commonly used to reduce variance within a noisy dataset. A random sample of data in a training set is selected with replacement (meaning that the individual data points can be chosen more than once). After several data samples are generated, decision trees are then trained independently, and depending on the type of task the average or majority of those predictions yield a more accurate estimate.

Feature randomness: when it is time to split a node, instead of considering every possible feature (like in a normal decision tree), each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.

In this study, we will consider GINI and Entropy (explained in section 5.2.) impurity measure at each node of the trees and we will exploit the scikit-learn implementation of random forests that combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

Hyperparameter tuning on the number of trees has been performed.

Results of the experiments are collected in the table below.

Pipeline	Accuracy	Macro-Precision	Macro-Recall	Macro-F1	Micro-Precision	Micro-Recall	Micro-F1	Weighted-Precision	Weighted-Recall	Weighted-F1	Criterion	No. estimators
BASIC	0.9267	0.9398	0.9364	0.9380	0.9267	0.9267	0.9267	0.9272	0.9267	0.9269	Entropy	100
PCA	0.9208	0.9345	0.9306	0.9324	0.9208	0.9208	0.9208	0.9214	0.9208	0.9210	Entropy	50
RO	0.9259	0.9381	0.9365	0.9372	0.9259	0.9259	0.9259	0.9263	0.9259	0.9260	Gini	100
RO+PCA	0.9220	0.9337	0.9333	0.9333	0.9220	0.9220	0.9220	0.9225	0.9220	0.9221	Gini	100
SMOTE	0.9265	0.9381	0.9377	0.9378	0.9265	0.9265	0.9265	0.9267	0.9265	0.9265	Gini	100
SMOTE+PCA	0.9220	0.9338	0.9338	0.9337	0.9220	0.9220	0.9220	0.9227	0.9220	0.9222	Gini	100
RU	0.9279	0.9400	0.9395	0.9395	0.9279	0.9279	0.9279	0.9290	0.9279	0.9282	Entropy	120
RU+PCA	0.9087	0.9227	0.9236	0.9229	0.9087	0.9087	0.9087	0.9097	0.9087	0.9089	Entropy	80
CCU	0.9164	0.9300	0.9296	0.9295	0.9164	0.9164	0.9164	0.9173	0.9164	0.9166	Entropy	50
CCU+PCA	0.9108	0.9241	0.9256	0.9243	0.9108	0.9108	0.9108	0.9122	0.9108	0.9109	Gini	80

Table 8 - Summary table for Random Forest

Best configuration found and scores are highlighted in green, worst ones in red.

Weighted-F1 Score trend among the different preprocessing techniques can be observed also in the following line plot.

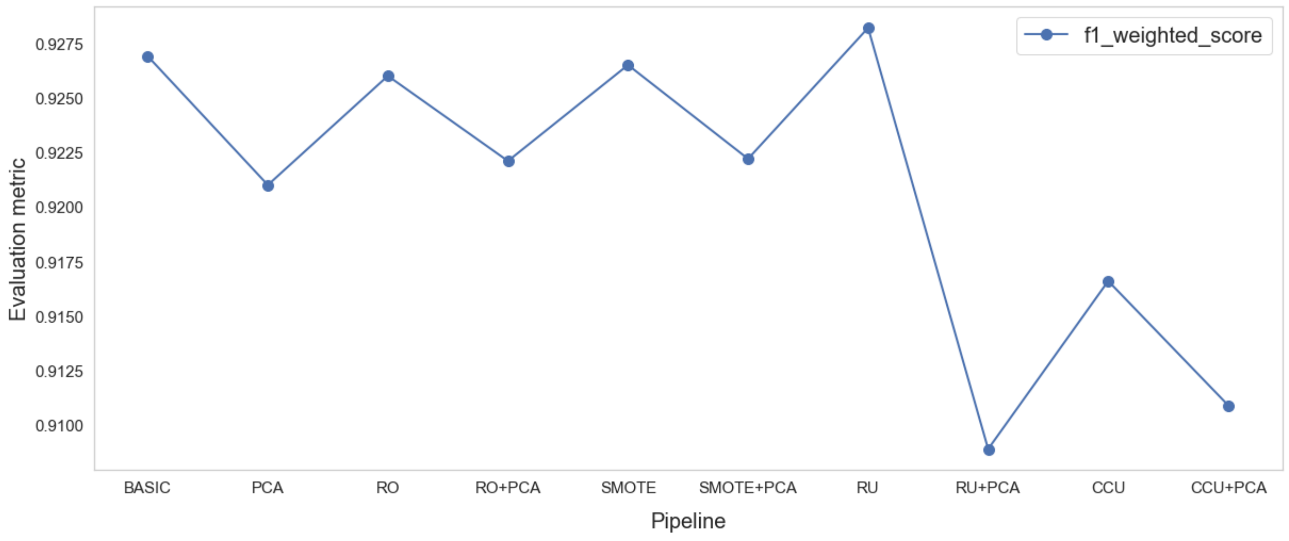


Figure 23 – Random Forest Weighted-F1-Score for each preprocessing method

As also noticed in section 5.2., 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.

Confusion matrix about the best configuration found is shown below.

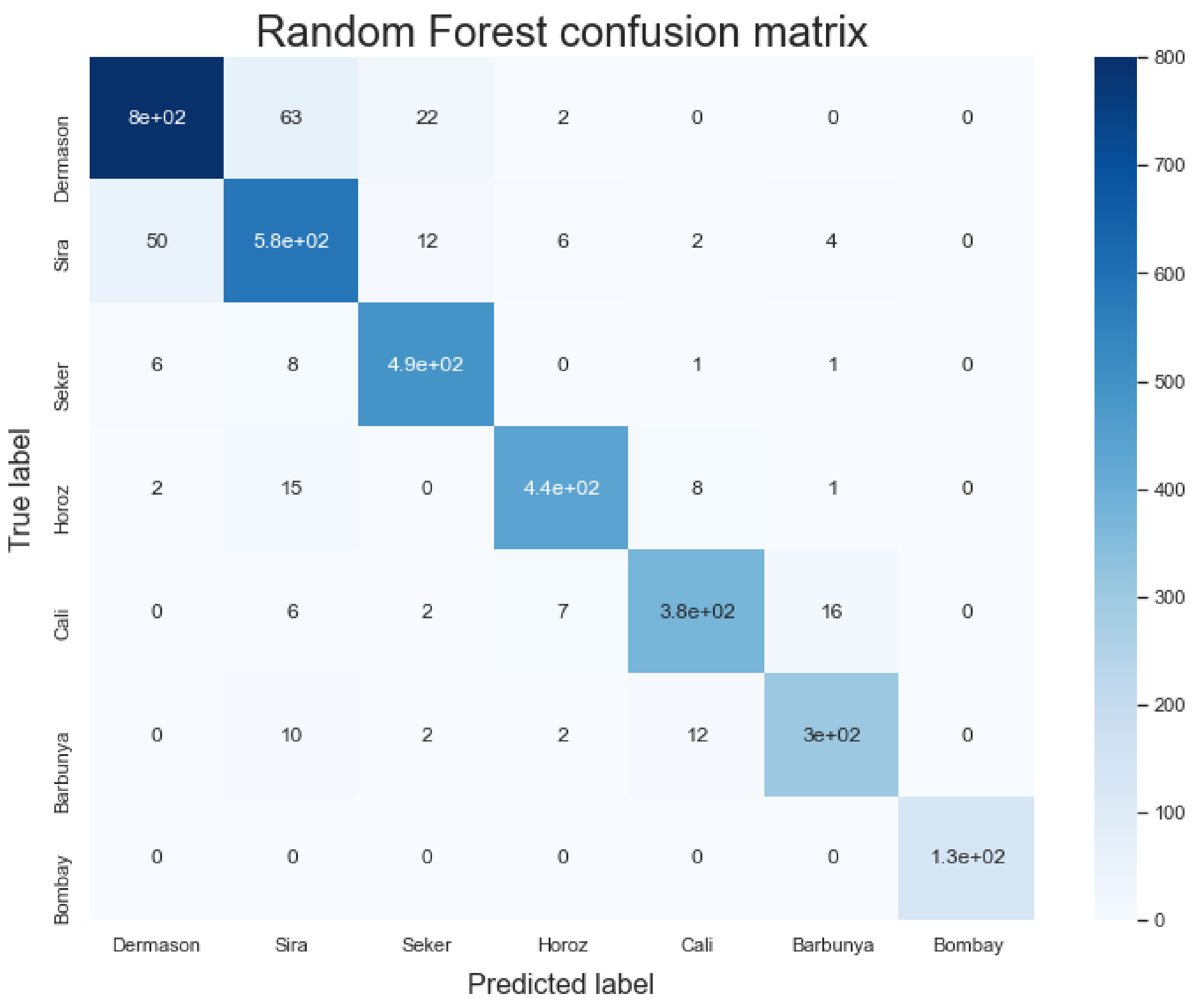


Figure 24 - Random Forest confusion matrix

5.4 Multinomial Logistic Regression

Multinomial Logistic Regression algorithm is an extension to the logistic regression model that involves changing the loss function to cross-entropy loss and predict probability distribution to a multinomial probability distribution to natively support multi-class classification problems.

Cross-Entropy Loss function is defined as follows:

$$L_{CE} = -\sum_{i=1}^{n} t_i \, log(p_i)$$

Where t_i is the truth label, p_i is the Softmax probability for the i^{th} class and n is the number of classes.

A benefit of multinomial logistic regression is that it can predict calibrated probabilities across all known class labels in the dataset.

An important hyperparameter for multinomial logistic regression is the penalty term. This term imposes pressure on the model to seek smaller model weights. This is achieved by adding a weighted sum of the model coefficients to the loss function, encouraging the model to reduce the size of the weights along with the error while fitting the model.

The type of penalty used in this study is the L2 penalty that adds the (weighted) sum of the squared coefficients to the loss function. A weighting of the coefficients can be used that reduces the strength of the penalty from full penalty to a very slight penalty.

The inverse of regularization strength C is tuned. If C is close to 1.0 indicate very little penalty, while if C close to zero indicate a strong penalty.

In the experiments performed, penalty values are tested on a log scale in order to quickly discover the scale of penalty that works well for a model. L2 penalty is explored with weighting values in the range from 0.0001 to 1.0.

Results are collected in the table shown in the next page.

Pipeline	Accuracy	Macro-Precision	Macro-Recall	Macro-F1	Micro-Precision	Micro-Recall	Micro-F1	Weighted-Precision	Weighted-Recall	Weighted-F1	C
BASIC	0.9217	0.9349	0.9323	0.9334	0.9217	0.9217	0.9217	0.9227	0.9217	0.9220	10
PCA	0.9217	0.9346	0.9325	0.9333	0.9217	0.9217	0.9217	0.9226	0.9217	0.9219	1
RO	0.9217	0.9333	0.9354	0.9340	0.9217	0.9217	0.9217	0.9230	0.9217	0.9219	1
RO+PCA	0.9214	0.9330	0.9350	0.9337	0.9214	0.9214	0.9214	0.9226	0.9214	0.9216	1
SMOTE	0.9238	0.9357	0.9374	0.9363	0.9238	0.9238	0.9238	0.9250	0.9238	0.9240	10
SMOTE+PCA	0.9220	0.9336	0.9356	0.9344	0.9220	0.9220	0.9220	0.9231	0.9220	0.9222	1
RU	0.9173	0.9300	0.9323	0.9307	0.9173	0.9173	0.9173	0.9190	0.9173	0.9176	10
RU+PCA	0.9185	0.9310	0.9336	0.9319	0.9185	0.9185	0.9185	0.9201	0.9185	0.9187	10
CCU	0.9129	0.9267	0.9293	0.9271	0.9129	0.9129	0.9129	0.9157	0.9129	0.9131	1
CCU+PCA	0.9114	0.9253	0.9278	0.9257	0.9114	0.9114	0.9114	0.9145	0.9114	0.9117	10

Table 9 - Summary table for Logistic Regression

Best configuration found and scores are highlighted in green, worst ones in red.

Weighted-F1 Score trend among the different preprocessing techniques can be observed also in the following line plot.

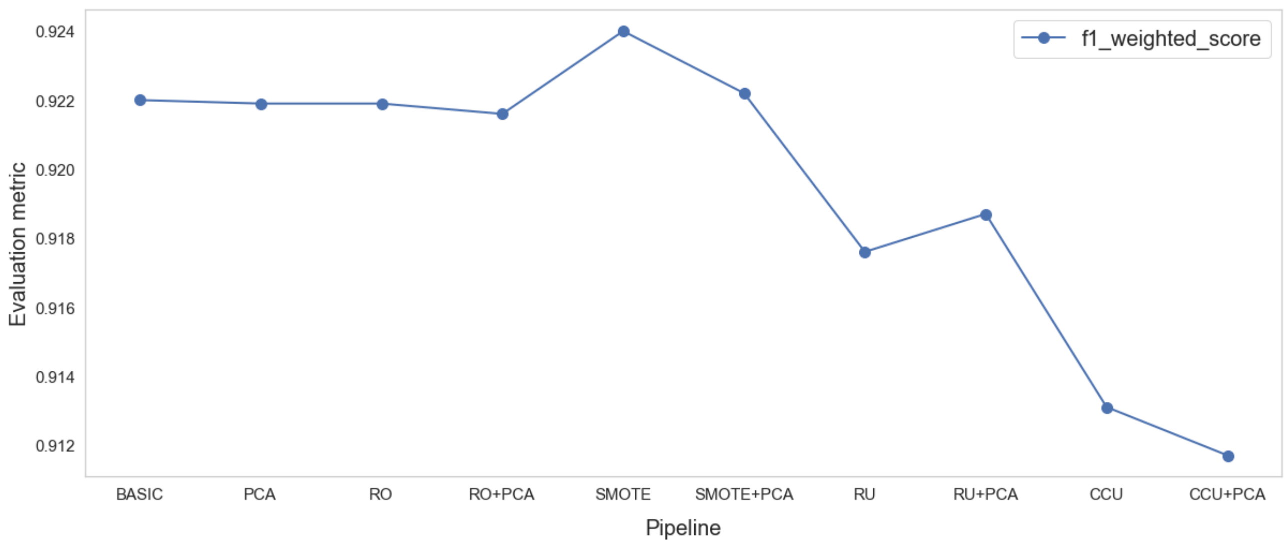


Figure 25 - Logistic Regression Weighted-F1-Score for each preprocessing method

As mentioned at the beginning of chapter 5., Logistic regression approach has been adapted to a multi-classification problem through one-vs-rest approach.

In one-vs-rest logistic regression a separate model is trained for each class predicted whether an observation is that class or not (thus making it a binary classification problem). It assumes that each classification problem is independent.

Confusion matrix about the best configuration found is shown below.

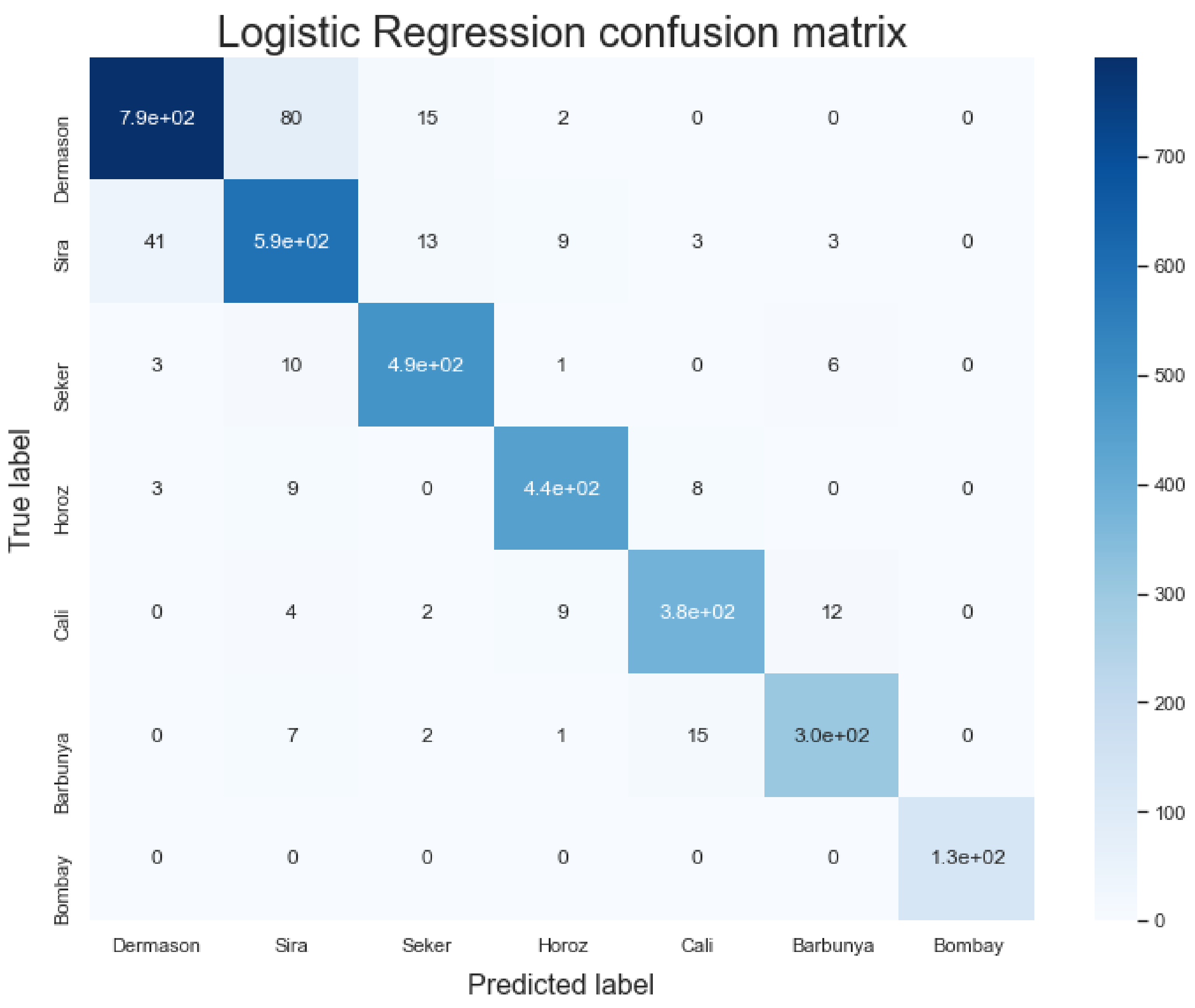


Figure 26 - Logistic Regression confusion matrix

5.5 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised machine learning algorithm that helps in classification problems. The objective is to find a hyperplane that maximizes the separation of the data points to their potential classes in an n-dimensional space (where n is equal to the number of classes). So, the hyperplane should be positioned with the maximum distance to the data points. The data points with the minimum distance to the hyperplane are called *Support Vectors*. Due to their close position, their influence on the exact position of the hyperplane is bigger than of other data points.

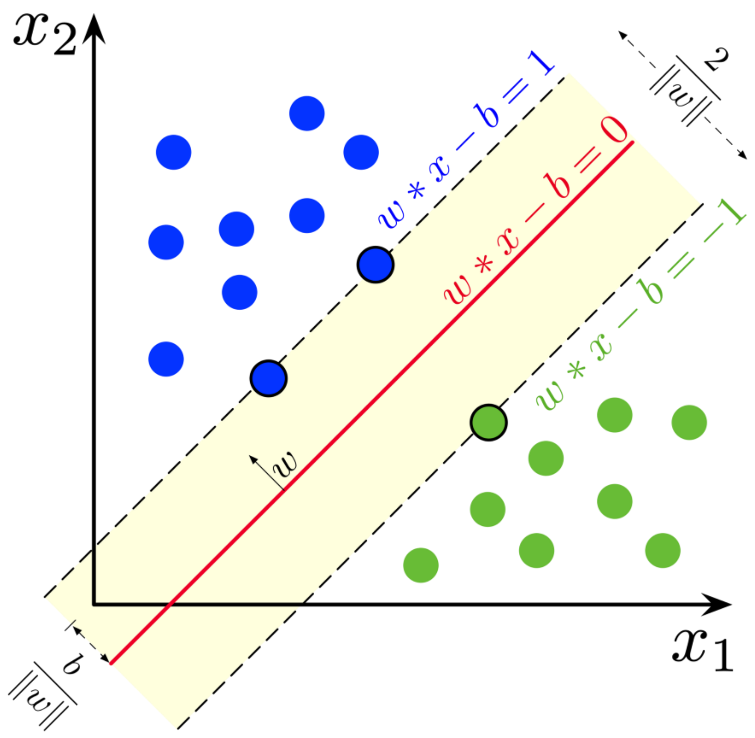


Figure 27 - Graphical representation of hard margin SVM for binary classification

More formally, SVM tries to define $\boldsymbol{w}$ and b, such that the hyperplane $\boldsymbol{w^T x} + b = 0$ is the “maximum-margin hyperplane”, i.e., such that the distance between the hyperplane and the nearest point $\vec{x_i}$ from either group is maximized. If we impose that all points must satisfy $|\boldsymbol{w^T} + b| \geq \boldsymbol{1}$ relatively to their label (in binary case), then the optimization problem becomes equivalent to finding the smallest $|\boldsymbol{w}|$.

5.5.1 Hard Margin

If our data is linearly separable, we go for a hard margin.

Let’s assume that the labels for our classes are {-1, +1} (binary problem). When classifying the data points, we want the points belonging to positives classes to be greater than +1, meaning $w^T x + b \geq 1$, and the points belonging to the negative classes to be less than -1, i.e. $w^T x + b \leq -1$.

We can combine these two constraints and express them as: $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1$. Therefore, our optimization problem would become:

$$\min_{\boldsymbol{w},b} \frac{1}{2} w^T \boldsymbol{w}$$

$$s.t. \quad y_i(\boldsymbol{w}^{\boldsymbol{T}} x_i + b) \geq 1$$

The $\boldsymbol{w}$ and b that solve this problem determine our classifier, $x \to\ sgn(w^T x - b)$ where $sgn(\cdot)$ is the sign function.

Sometimes, the data is linearly separable, but the margin is so small that the model becomes prone to overfitting or being too sensitive to outliers. Also, in this case, we can opt for a larger margin by using soft margin SVM in order to help the model generalize better.

5.5.3 Soft Margin

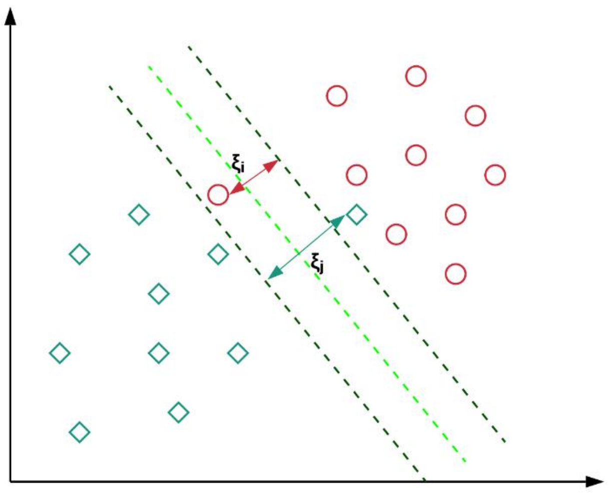


Figure 28 - Soft Margin SVM

The soft margin SVM follows a somewhat similar optimization procedure with a couple of differences. First, in this scenario, we allow misclassifications to happen. So, we'll need to minimize the misclassification error, which means that we'll have to deal with one more constraint. Second, to minimize the error, we should define a loss function. A common loss function used for soft margin is the hinge loss.

$$max\{0, 1 - y_i(\boldsymbol{w}^T \boldsymbol{x}_i + b)\}$$

The loss of a misclassified point is called a slack variable and is added to the primal problem that we had for hard margin SVM. So, the primal problem for the soft margin becomes:

$$min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \zeta_i$$

$$s.t. \quad y_i(w^T x_i + b) \geq 1 - \zeta_i \quad \forall i = 1, \dots, n, \zeta_i \geq 0$$

A new regularization parameter C controls the trade-off between maximizing the margin and minimizing the loss. As you can see, the difference between the primal problem and the one for the hard margin is the addition of slack variables. The new slack variables (ζ_i in the figure below) add flexibility for misclassifications of the model.

Finally, we can also compare the dual problems:

$$\max - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_{i=1}^{n} \alpha_i$$

$$s.t. \quad \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C$$

As you can see, in the dual form, the difference is only the upper bound applied to the Lagrange multipliers.

5.5.4 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 $x_i \in \mathbb{R}^N \to \varphi(x_i) \in \mathbb{R}^M$, allowing for a linear separation of the data in the new dimension.

In this way the algorithm can be kernelized and the scalar product inside the dual problem is replaced with a **kernel function** that represents the scalar product in another space. No matter what the higher-dimensional space is, all that it's needed is a function able to compute the scalar products in the new transformed space.

The kernel function k is a function defined as $k : X \times X \to \mathbb{R}$ that satisfies the conditions of the Mercer's theorem for which it must be symmetric, positive semidefinite. For the Mercer's theorem, there exist a Hilbert Space S and a transformation φ that maps our data into a space in which the kernel represents the scalar product on the dual formulation:

$$k(x, x') := \varphi(x)^T \varphi(x')$$

In this way we can work in our space, without mapping data in a new one, replacing the scalar product with the kernel function.

There are different kinds of Kernel. In this study following ones are used:

- **Linear Kernel**

$$K(x, x') = x^T x'$$

- **Gaussian RBF (Radial Basis Functions) Kernel**

$$K(x, x') = e^{-\gamma \|x - x'\|^2}, \gamma > 0$$

When training an SVM with the *Radial Basis Function* (RBF) kernel, two parameters must be considered: C and γ. The parameter C, common to all SVM kernels, trades off misclassification of training examples against simplicity of the decision surface. A low C makes the decision surface smooth, while a high C aims at classifying all training examples correctly. γ defines how much influence a single training example has. The γ parameter can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. So, the larger γ is, the closer other examples must be to be affected.

- **Polynomial Kernel**

$$K(x, x') = (x^T x' + c)^d$$

Where d is the polynomial degree and $c \geq 0$ is a free parameter trading off the influence of higher-order versus lower-order terms in the polynomial.

Simply put, these functions determine the smoothness and efficiency of class separation and playing around with their hyperparameters may lead to overfitting or underfitting.

Results of Grid Search are grouped in the table in the following page.

Pipeline	Accuracy	Macro-Precision	Macro-Recall	Macro-F1	Micro-Precision	Micro-Recall	Micro-F1	Weighted-Precision	Weighted-Recall	Weighted-F1	C	Kernel	γ
BASIC	0.9297	0.9417	0.9384	0.9399	0.9297	0.9297	0.9297	0.9301	0.9297	0.9298	100	rbf	0.01
PCA	0.9297	0.9418	0.9384	0.9400	0.9297	0.9297	0.9297	0.9301	0.9297	0.9298	100	rbf	0.01
RO	0.9291	0.9406	0.9404	0.9403	0.9291	0.9291	0.9291	0.9297	0.9291	0.9292	100	Rbf	0.01
RO+PCA	0.9294	0.9408	0.9404	0.9405	0.9294	0.9294	0.9294	0.9300	0.9294	0.9295	100	Rbf	0.01
SMOTE	0.9288	0.9395	0.9394	0.9393	0.9288	0.9288	0.9288	0.9292	0.9288	0.9289	100	rbf	0.01
SMOTE + PCA	0.9285	0.9394	0.9391	0.9391	0.9285	0.9285	0.9285	0.9291	0.9285	0.9286	100	Rbf	0.01
RU	0.9235	0.9354	0.9355	0.9353	0.9235	0.9235	0.9235	0.9241	0.9235	0.9236	100	Rbf	0.01
RU+PCA	0.9232	0.9352	0.9353	0.9351	0.9232	0.9232	0.9232	0.9238	0.9232	0.9233	100	Rbf	0.01
CCU	0.9232	0.9358	0.9362	0.9355	0.9232	0.9232	0.9232	0.9249	0.9232	0.9234	10	Rbf	0.01
CCU + PCA	0.9238	0.9361	0.9359	0.9356	0.9238	0.9238	0.9238	0.9246	0.9238	0.9238	100	Rbf	0.01

Table 10 - Summary table for SVM

Best configuration found and scores are highlighted in green, worst ones in red.

As we can see, best results for each preprocessing strategy are reached by applying RBF kernel. It allows us to say that the starting problem is not linearly separable and needs a non-linear function.

Weighted-F1 Score trend among the different preprocessing techniques can be observed also in the following line plot.

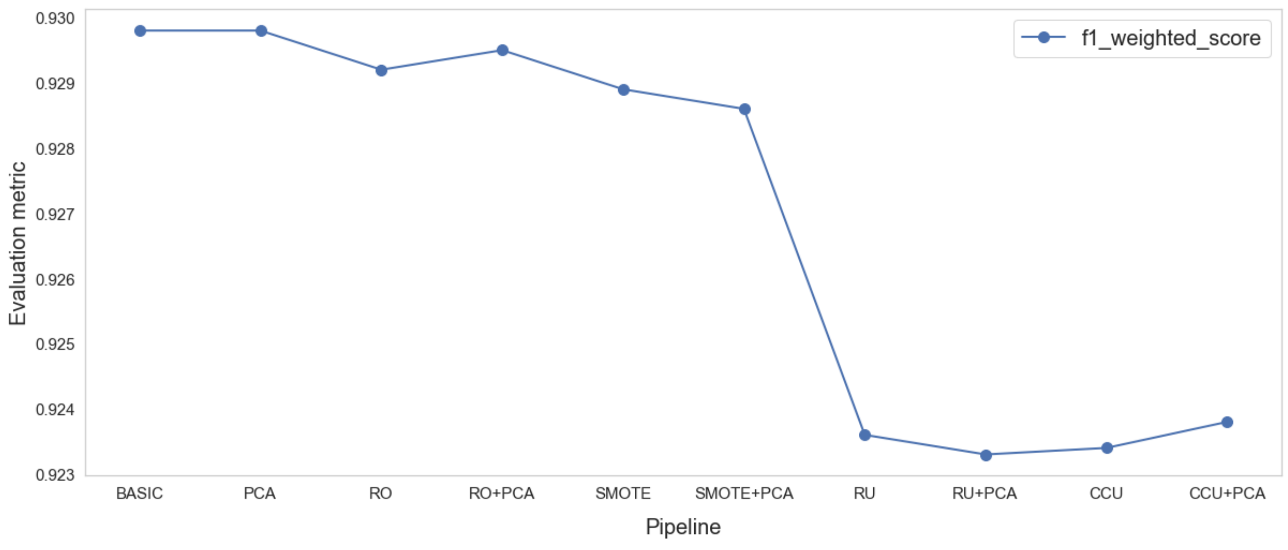


Figure 29 – SVM weighted-F1-Score for each preprocessing method

As we can see, PCA application does not always improve results. Indeed, the usage of PCA technique can lose some spatial information which is important for classification, so the classification accuracy can decrease.

The confusion matrix of the configuration with the highest F1-score is displayed:

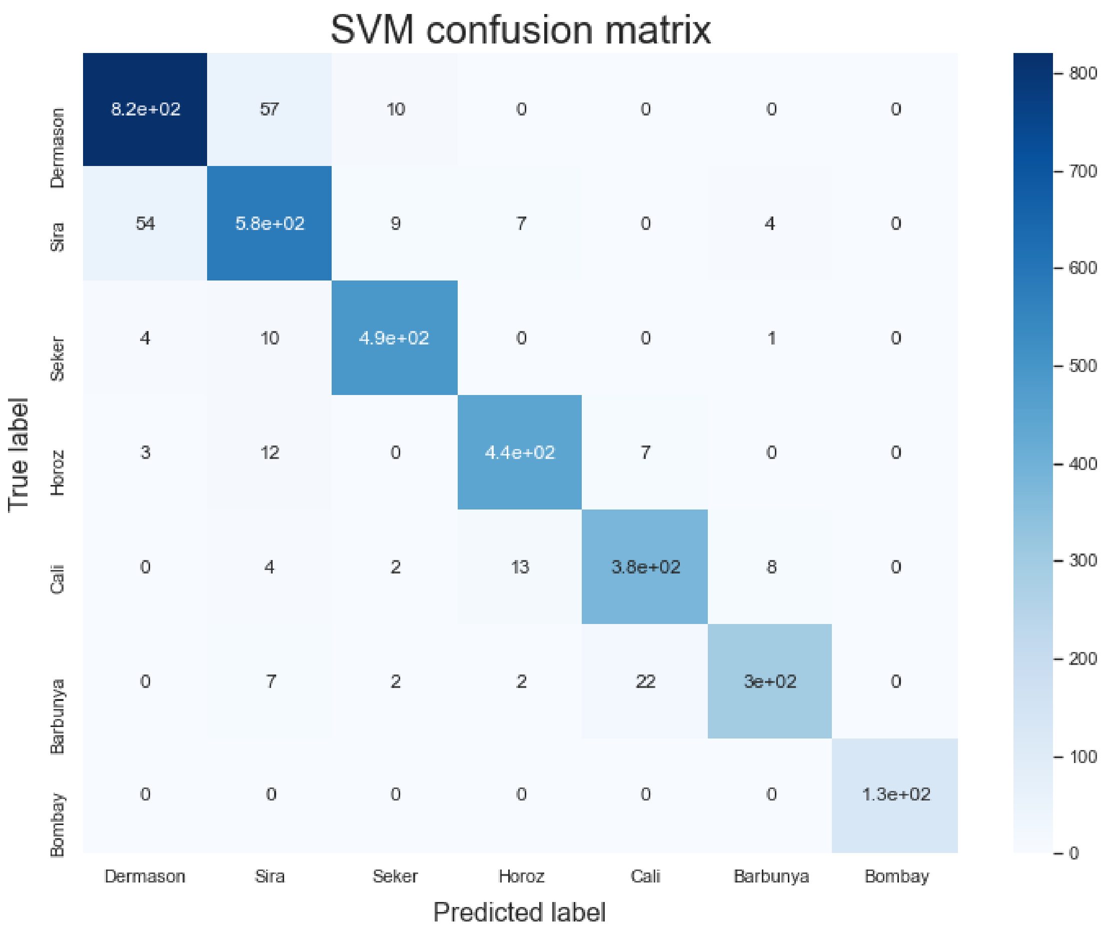


Figure 30 - SVM confusion matrix

6. Conclusions

In this study different supervised learning algorithms have been inspected and presented with their mathematical details, and finally used on the Dry Bean Dataset to build a machine learning pipeline. The Support Vector Machine with RBF Kernel was proved to be the best one on the task, by providing a F1-score equal to 0.9298. We can also notice that PCA algorithm is not always improving the classifiers, so feature selection and also the application of resampling strategies were not always beneficial, probably because of the distribution of features in the space and especially for algorithms that are more prone to overfitting, such as Random Forest.

7. References

1. https://machinelearningmastery.com/
2. https://towardsdatascience.com
3. https://fsi.colostate.edu

4. Murat Koklu, Ilker Ali Ozkan, Multiclass classification of dry beans using computer vision and machine learning techniques (2020)

5. Margherita Grandini, Enrico Bagli, Giorgio Visani, 'Metrics For Multi-Class Classification: An Overview' (2020)