Machine Learning & Data Mining

Dry Beans Classification

Davide Marinoni, Samuele Ponzin

2023

Abstract

In this work we analyse several machine learning techniques for multi-class classification of dry beans. We explore different ways of analysing the dataset and apply different pre-processing techniques for ensuring an high performance in terms of accuracy of the classification. We test different classifiers, training the model using k-fold cross validation. The results show high performance for every of the tested classification algorithm.

The dry beans classification is based on the data-set derived from the work of [KO20].

The analysis was carried on using Visual Studio Code in Python language. The code is available online @samuponz.drybeans

Contents

Project overview

This report is a summary of the project for the Machine Learning & Data Mining course, carried out by Davide Marinoni and Samuele Ponzin.

The aim of the project was to apply some of the Machine Learning techniques presented during the course and further explore other topics in the design of a machine learning pipeline, in order to solve a classification task starting from a given dataset.

We decided to work with the DryBeans dataset because it seemed to have a good instances to features ratio and because the way the data was collected was clearly explained in [KO20].

Our study can be easily divided in three main area that give the name to the three central chapters of this report: **Dataset Analysis**, **Pre-processing** and **Classification**.

The project repository, containing the IPython Jupyter Notebooks and other assets can be found at https://github.com/samuponz/drybeans.

Chapter 1

Dataset Analysis

1.1 Dataset overview

The data-set is composed of a total of 13611 instances. Each instance is represented by 17 attributes. 16 out of 17 attributes are numerical and represent *morphological features (regarding dimension and shape)* of the beans under analysis.

The left-out attribute is of categorical type and represents one of the 7 varieties of dry beans (bean *class*). In order to classify the beans, this last attribute is used as a label to build a supervised classification algorithm.

The dataset was built by taking 1 kg of every bean variety. Photos were taken of every bean from which the morphological features were computed.

1.1.1 Features

The morphological features of the beans were either directly collected by means of image processing techniques or later derived as deterministic functions of the directly measured features. Because of how they are defined, the "derived" features are expected to be correlated to the "raw" features. Values of all the features are to be intended in pixel count.

The nature of the features are explained in [KO20]. Here we report them for completeness, adding some useful comments for better interpretation.

- **Area (A)**: the area of a bean region is defined as the number of pixels within its boundaries.

  $$A = \sum_{r,c \in R} 1$$

  where (r, c) is the size of a Region R.

  The area of a bean is computed after having binary segmented its photo. The bean can be seen as an irregular polygon, thus with an irregular shape.

- **Perimeter (P)**: bean circumference is defined as the length of its border.

  The perimeter could have been computed after having performed edge detection or morphing on the binary image.

- **Major Axis Length (L)**: the distance between the ends of the longest line that can be drawn from a bean.

- **Minor Axis Length (l)**: the longest line that can be drawn from the bean while standing perpendicular to the main axis.

  L and l can be seen as "raw" features since they are computed using the actual region of the bean and not a more regular geometrical shape approximating the bean shape.

- **Aspect ratio (K)**: The ratio between L and l.

$$K = \frac{L}{l}$$

- **Eccentricity (Ec)**: eccentricity of the ellipse having the same moments as the region.

  This is the first feature that models the bean as a regular shape. This introduces an approximation error, different regions could be modeled by the same ellipse.

- **Convex Area(C)**: number of pixels in the smallest convex polygon that can contain the area of a bean seed.

  This feature basically tells how much the bean shape is regular, meaning the bean doesn't have any irregular shape along the plane of the image.

- **Equivalent diameter (Ed)**: the diameter of a circle having the same area as a bean seed area.

$$Ed = \sqrt{\frac{4A}{\pi}}$$

  This is the Feret Diameter: the diameter of a circle having the same area

  This is the first clear example of feature expansion. Ed is just a function of the Area. The two features are highly (non-linearly) correlated.

- **Extent (Ex)**: the ratio of the pixels in the bounding box to the bean area.

$$Ex = \frac{A}{A_B}$$

  where A_B = area of the bounding box rectangle.

- **Solidity (S)**: also known as **convexity**. The ratio of the pixels in the bean region to the convex shell.

$$A = \frac{A}{C}$$

- **Roundness (R)**: Calculated with the following formula:

$$R = \frac{4\pi A}{P^2}$$

- **Compactness (Co)**: Measures the roundness of an object:

$$Co = \frac{Ed}{L} = \frac{1}{L}\sqrt{\frac{4A}{\pi}}$$

  Compactness provides a measure of the object's roundness and is the ratio of the Equivalent (Feret) diameter to the object's length

  Roundness and compactness are highly correlated, they are also really similar conceptually.

- **Shape Factor 1 (SF1)**:
$$SF1 = \frac{L}{A}$$
- **Shape Factor 2 (SF2)**:
$$SF1 = \frac{l}{A}$$
- **Shape Factor 3 (SF3)**:
$$SF3 = \frac{A}{\pi \frac{L}{2} \frac{L}{2}} = \frac{4A}{\pi L^2} = \frac{Ed^2}{L^2} = Co^2$$
Conceptually SF3 is the ratio of the bean area (bean region) to the area of the equivalent circle of diameter L. As shown, SF3 is the squared root of the Compactness. They are thus highly (non-linearly) correlated.
- **Shape Factor 4 (SF4)**:
$$SF4 = \frac{A}{\pi \frac{L}{2} \frac{l}{2}}$$
Conceptually SF4 is the ratio of the bean area (bean region) to the area of the equivalent ellipse of axes L and l. This is really similar to Solidity, where the Area is divided by the Area of a Convex polygon.

It is clear that a lot of these features are highly correlated "by design".

1.1.2 Classes

The dataset is composed of dry beans of 7 different varieties that are different in size and shape. In Figure 1.1 it is possible to see examples of each different variety. Descriptions of the varieties are given in [KO20], we report them here for completeness.

- **Barbunya**: beige-colored background with red stripes or variegated, speckled color, its seeds are large, physical shape is oval close to the round.
- **Bombay**: it is white in color, its seeds are very big and its physical structure is oval and bulging.
- **Cali**: it is white in color, its seeds are slightly plump and slightly larger than dry beans and in shape of kidney.
- **Dermason**: this type of dry beans, which are fuller flat, is white in color and one end is round and the other ends are round.
- **Horoz**: dry beans of this type are long, cylindrical, white in color and generally medium in size.
- **Seker**: large seeds, white in color, physical shape is round.
- **Sira**: its seeds are small, white in color, physical structure is flat, one end is flat, and the other end is round.

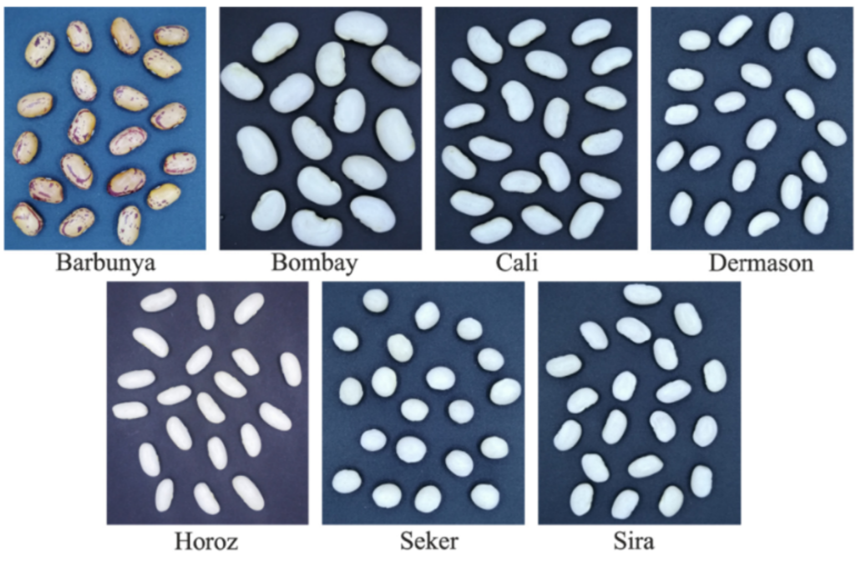


Figure 1.1: Examples of dry beans for each variety. Image taken from [KO20].

1.2 Dataset analysis

The dateset is a 13611×17 matrix and it weights 1.8 MB.

The first 16 columns are the morphological features of the dry beans: **Area** and **Convex Area** values are stored as `int64` while the rest are stored as `float64`.

All the features are numerical and are measured in pixel count. In [KO20] is mentioned that the acquisition system is set at a distance of 15 cm above a 12×10 cm^2 box inside which dry beans are placed.

Regarding the **Class** attribute, i.e. the dry beans varieties, this is of categorical nature and its values are stored as `object`.

1.2.1 Preparing the dataset

The first thing we did was to drop duplicate instances (68 instances). This led to a reduction of the instances from 13611 to 13543.

We casted all the numerical attributes values to `float64` and the categorical class attribute to `string`. We also renamed couple of attributes for the sake of clearness.

There were no missing values for any of the attributes (likely because all the attributes are returned by an image processing algorithm), thus there was no need of implementing any imputer strategy.

1.2.2 Visual inspection of features distributions

In Table 1.1 it is possible to observe the statistical distribution of the features of dry beans. It can be useful to understand which features have high variance and if there are a lot of outliers. In order to better understand these important aspects and the relationships between the different features, we reported the histograms of the features distributions in Figure 1.2.

Analysing the distributions of feature values it can be noticed that some features have a very similar distribution, this happens with features that are highly related to each other.

As seen in Section 1.1.1, there are some features that are highly correlated by design:

	count	mean	std	min	25%	50%	75%	max
Area	1.354E+04	5.305E+04	2.939E+04	2.042E+04	3.628E+04	4.458E+04	6.138E+04	2.546E+05
Perimeter	1.354E+04	8.550E+02	2.147E+02	5.247E+02	7.032E+02	7.939E+02	9.771E+02	1.985E+03
MajorAxisLength	1.354E+04	3.199E+02	8.581E+01	1.836E+02	2.531E+02	2.964E+02	3.763E+02	7.389E+02
MinorAxisLength	1.354E+04	2.024E+02	4.505E+01	1.225E+02	1.759E+02	1.925E+02	2.172E+02	4.602E+02
AspectRatio	1.354E+04	1.581E+00	2.452E-01	1.025E+00	1.431E+00	1.550E+00	1.704E+00	2.430E+00
Eccentricity	1.354E+04	7.503E-01	9.186E-02	2.190E-01	7.151E-01	7.640E-01	8.097E-01	9.114E-01
ConvexArea	1.354E+04	5.377E+04	2.984E+04	2.068E+04	3.667E+04	4.512E+04	6.236E+04	2.633E+05
EquivDiameter	1.354E+04	2.530E+02	5.931E+01	1.612E+02	2.149E+02	2.382E+02	2.796E+02	5.694E+02
Extent	1.354E+04	7.498E-01	4.894E-02	5.553E-01	7.187E-01	7.599E-01	7.868E-01	8.662E-01
Solidity	1.354E+04	9.872E-01	4.650E-03	9.192E-01	9.857E-01	9.883E-01	9.900E-01	9.947E-01
roundness	1.354E+04	8.737E-01	5.939E-02	4.896E-01	8.334E-01	8.835E-01	9.170E-01	9.907E-01
Compactness	1.354E+04	8.004E-01	6.146E-02	6.406E-01	7.632E-01	8.015E-01	8.345E-01	9.873E-01
ShapeFactor1	1.354E+04	6.561E-03	1.130E-03	2.778E-03	5.893E-03	6.643E-03	7.270E-03	1.045E-02
ShapeFactor2	1.354E+04	1.719E-03	5.955E-04	5.642E-04	1.158E-03	1.700E-03	2.173E-03	3.665E-03
ShapeFactor3	1.354E+04	6.443E-01	9.865E-02	4.103E-01	5.825E-01	6.424E-01	6.963E-01	9.748E-01
ShapeFactor4	1.354E+04	9.951E-01	4.347E-03	9.477E-01	9.937E-01	9.964E-01	9.979E-01	9.997E-01

Table 1.1: Statistical distribution of the features of dry bean varieties.

- **Shape Factor 3** is just **Compactness** squared, for this reason the distribution of values for these two features are really similar.
- **Solidity** and **Shape Factor 4**, which are conceptually really similar measures, have a very similar distribution.

Area, **Perimeter**, **Major Axis Length**, **Minor Axis Length**, **Convex Area** and **Equivalent Diameter** have a similar distribution. These seems to be naturally highly correlated, meaning that this correlation is due to the nature of the beans: if the area of a bean increases, also the perimeter increases, as well as the axes, the convex area and the equivalent diameter.

1.2.3 Univariate features distributions for different classes

After having studied the feature distributions we also looked at the distribution of feature values divided by class, this is necessary to understand which features are the most informative, i.e. capable of separating well the classes.

In Figure 1.3 are shown the features values distributions for every dry bean variety. Here it is possible to see how the highly correlated features **Area**, **Perimeter**, **Major Axis Length**, **Minor Axis Length**, **Convex Area** and **Equivalent Diameter** divide the classes in a similar way. In particular it can be seen that the **Bombay** variety is well separated from the others.

It seems that other features are not so informative: the mean value of **Solidity** as well as the standard deviation seem to be similar for all the classes. **Shape Factor 4**, which is highly correlated to Solidity seems to do a little bit better, showing that really similar features can give different results.

The best features, in terms of how well they can separate different classes (most relevant features), seem to be **Roundness**, **Compactness** and **the first 2 shape factors** (the 3rd one is also good but it is highly correlated with the 2nd one).

1.2.4 Multivariate features distributions

These univariate distributions can give an idea of the clusters of the different classes, but these are of course just marginals distributions of the 16D multivariate distribution placed in the feature space. For the sake of better visualizing the clusters of classes produced by our morphological features, we can plot the 2D and 3D distributions in the respective 2D and 3D subspaces of the feature space. Some examples of 2D and 3D distributions are shown respectively in Figure 1.4 and 1.5.

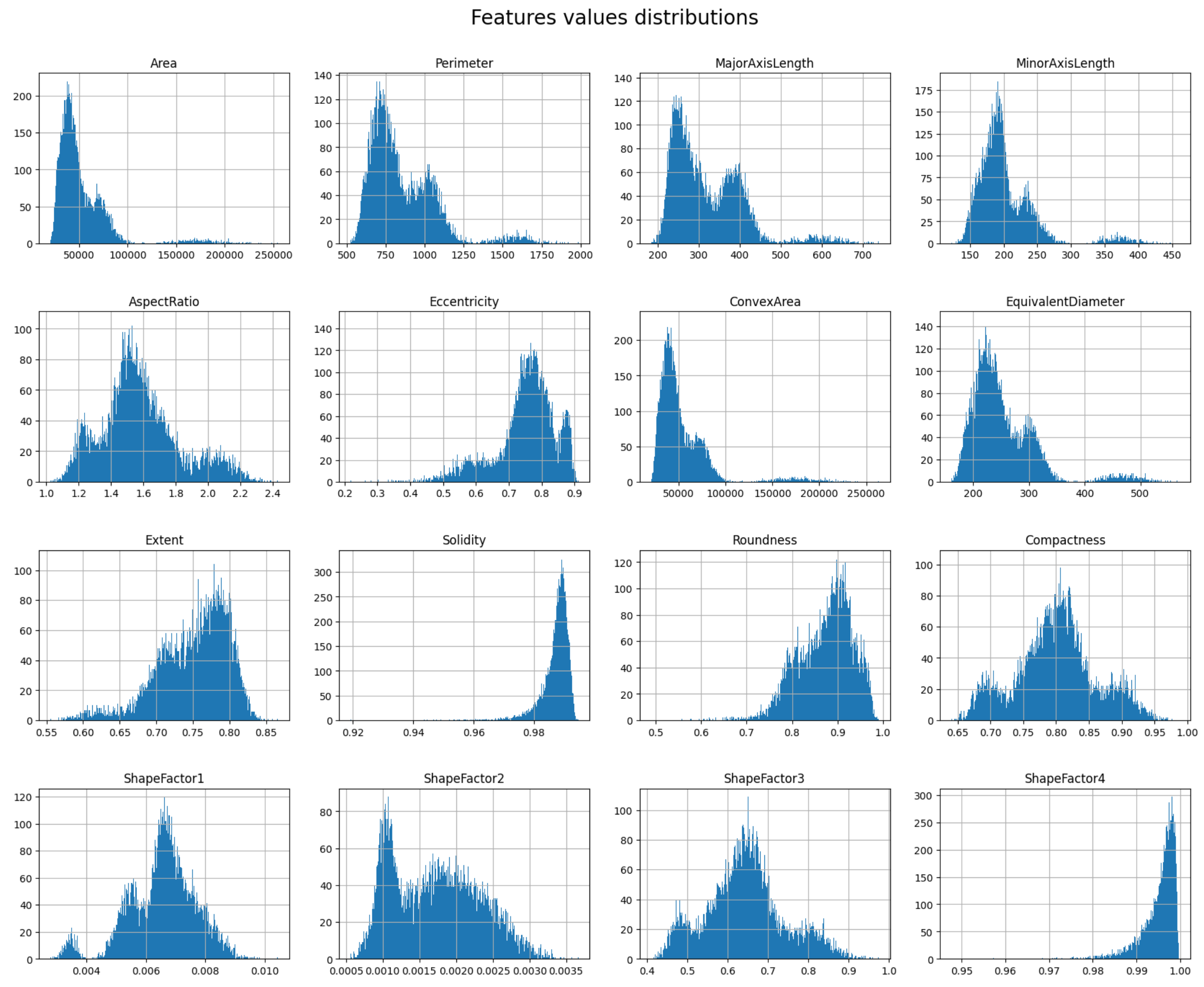


Figure 1.2: Distribution of the values for each of the 16 morphological features.

Analysis of two-dimensional multivariate feature distributions

Figure 1.4a and 1.4b show good clusters for different classes, this is due to the fact that the selected features are not highly correlated. If the features are correlated it is likely that there will not be a good partitioning of the classes, but it is not always true: in Figure 1.4c it can be seen that the features are perfectly correlated (correlation is exactly 1) but there seems to be a good clustering for a couple of classes: **Horoz** and **Seker** (Note that this division could just be a visual effect due to the plotting function).

In Figure 1.4d, instead, can be seen that the similar nature of the features **Solidity** and **Shape Factor 4** leads to a very poor partition of the classes, even if the correlation is not excessively high.

Analysis of three-dimensional multivariate feature distributions

In Figure 1.5a we try to visualize the joined distribution of values for what seem to be three of the best features in order to well partitioning the classes, because of the low correlation that there is between them. In Figure 1.5b, instead, we show the distribution for **Area**, **Perimeter** and **Major Axis Length**: again it is possible to see that even if these are highly correlated they manage to do some kind of separation for the **Bombay class**.

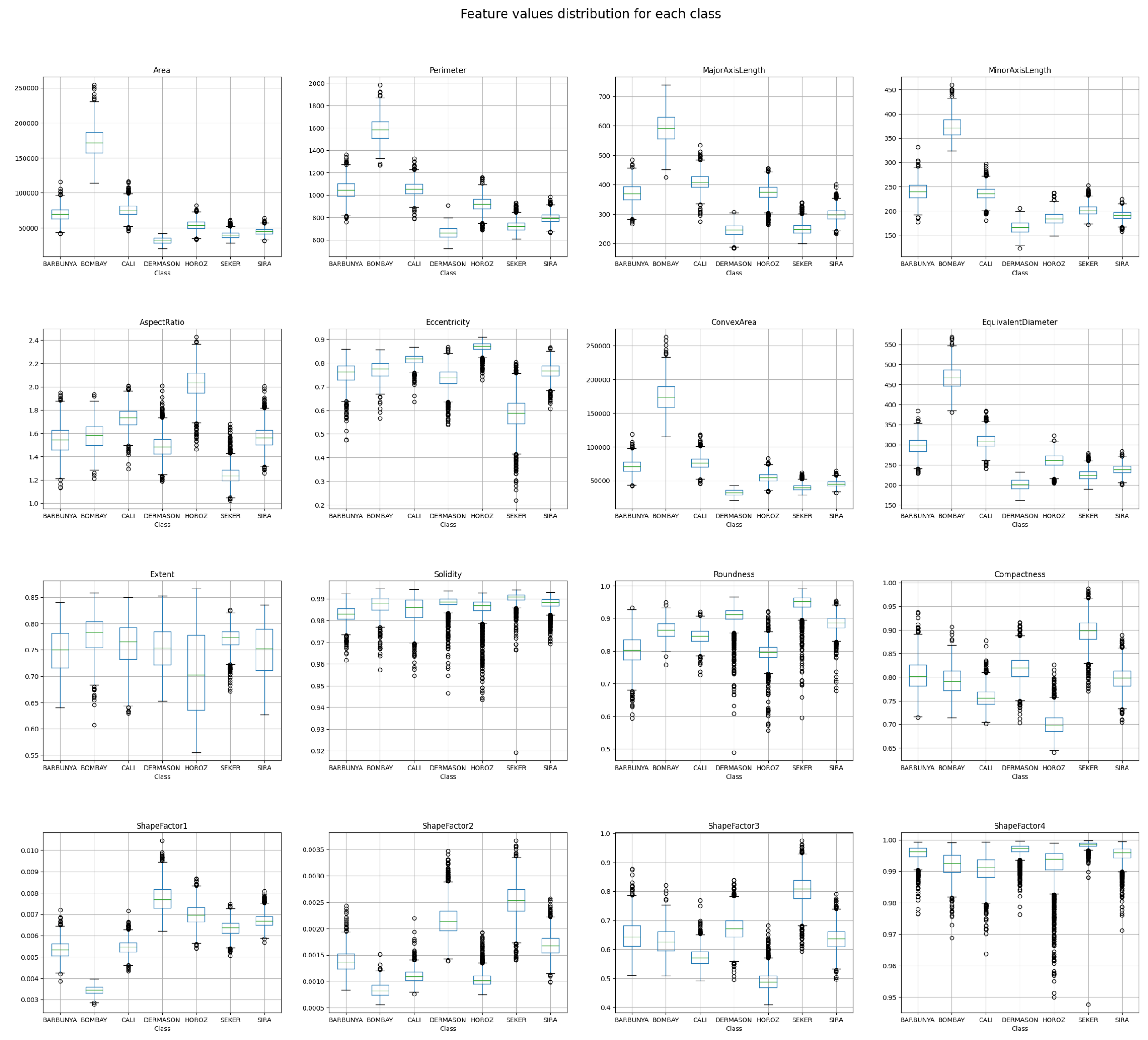


Figure 1.3: Distribution of the values for each of the 16 morphological features.

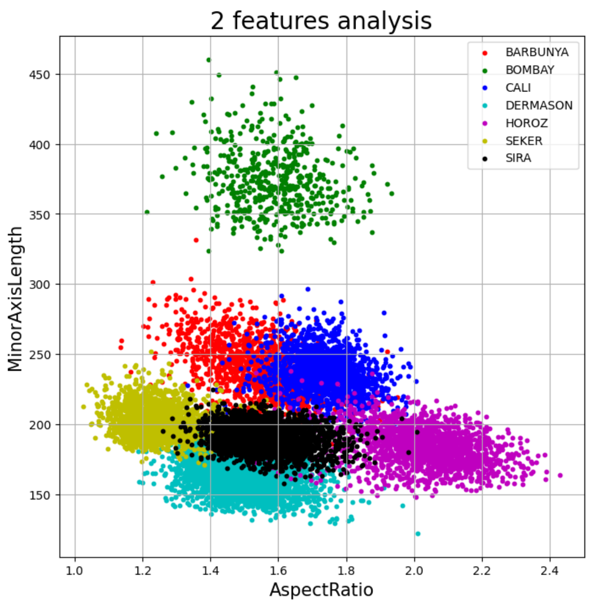
1.2.5 Features cross-correlation

After this qualitative visual study, it seems clear that correlation plays an important role in the selection of the best features for solving classification tasks. Here we finally quantitatively evaluate the correlation between each pair of features.

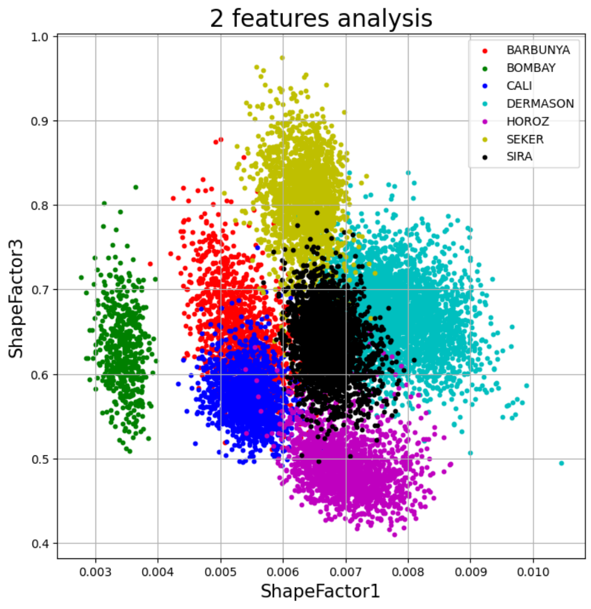
Correlation is a measure of similarity between vectors. In the context of data analysis, it is used to find out if there is some kind of relationship between different attributes.

There exists different correlation coefficients that can find different types of relationships between variables. Here we compute the **Pearson correlation coefficient**, that spans between -1 and 1 and quantifies the amount of **linear dependence** of two variables. Other coefficients, such as the **Spearman correlation coefficient**, are ranked-based (meaning that they assess how well the relationship between two variables can be described using a monotonic function) and thus are more general.

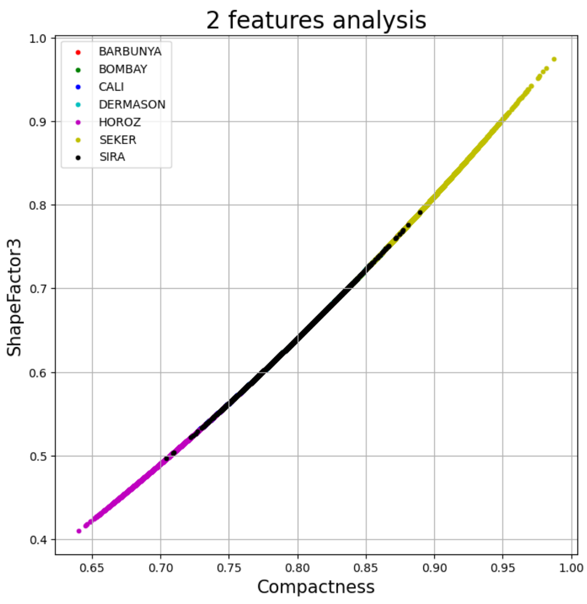
The Pearson correlation is computed for every pair of feature and the results is shown in the cross-correlation matrix of the Feature, reported in Figure 1.6. This matrix can



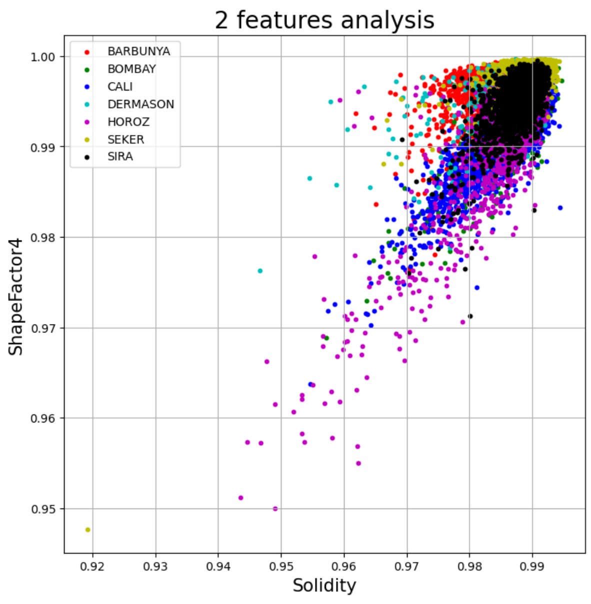
(a) Multivariate dist. of values (K, l)



(b) Multivariate dist. of values (SF1, SF3)

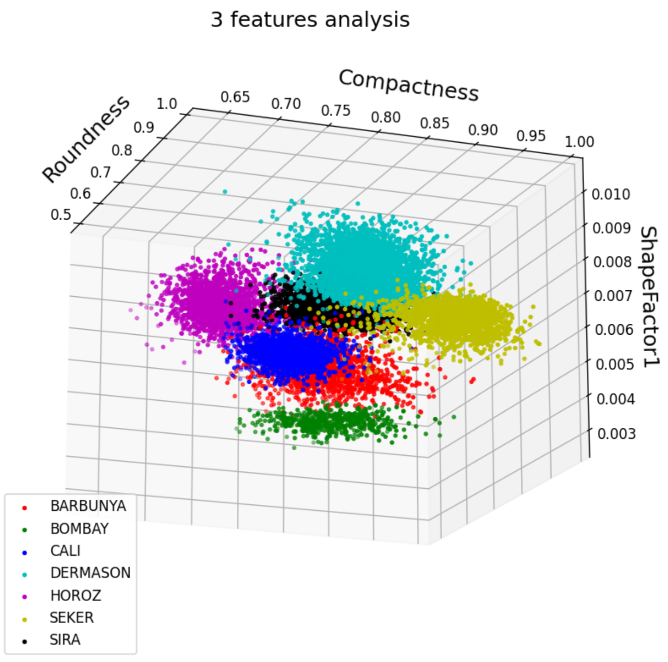


(c) Multivariate dist. of values (Co, SF3)

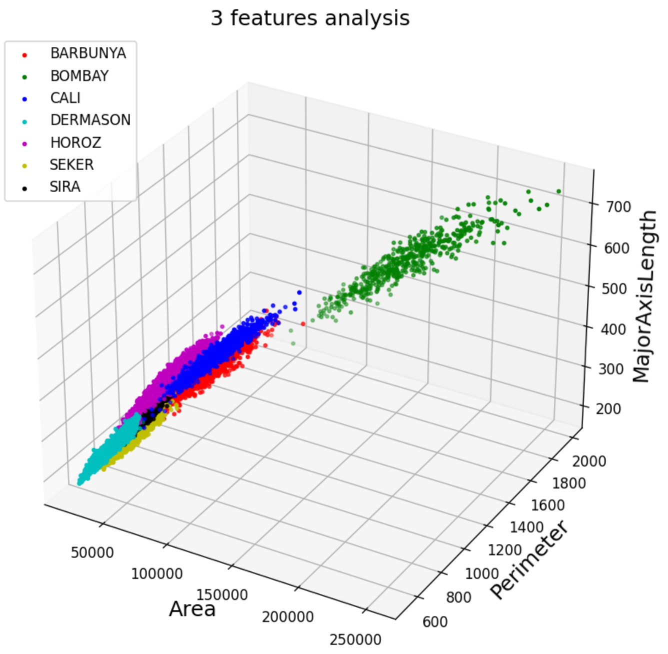


(d) Multivariate dist. of values (S, SF4)

Figure 1.4: Examples of 3D distributions for different features pairs.



(a) Multivariate distribution of values (Co, SF1, R)



(b) Multivariate distribution of values (A, L, P)

Figure 1.5: Examples of 3D distributions for different features triplets.

be obtained by computing the covariance matrix of the features and normalizing it by the product of the standard deviations of all the features.

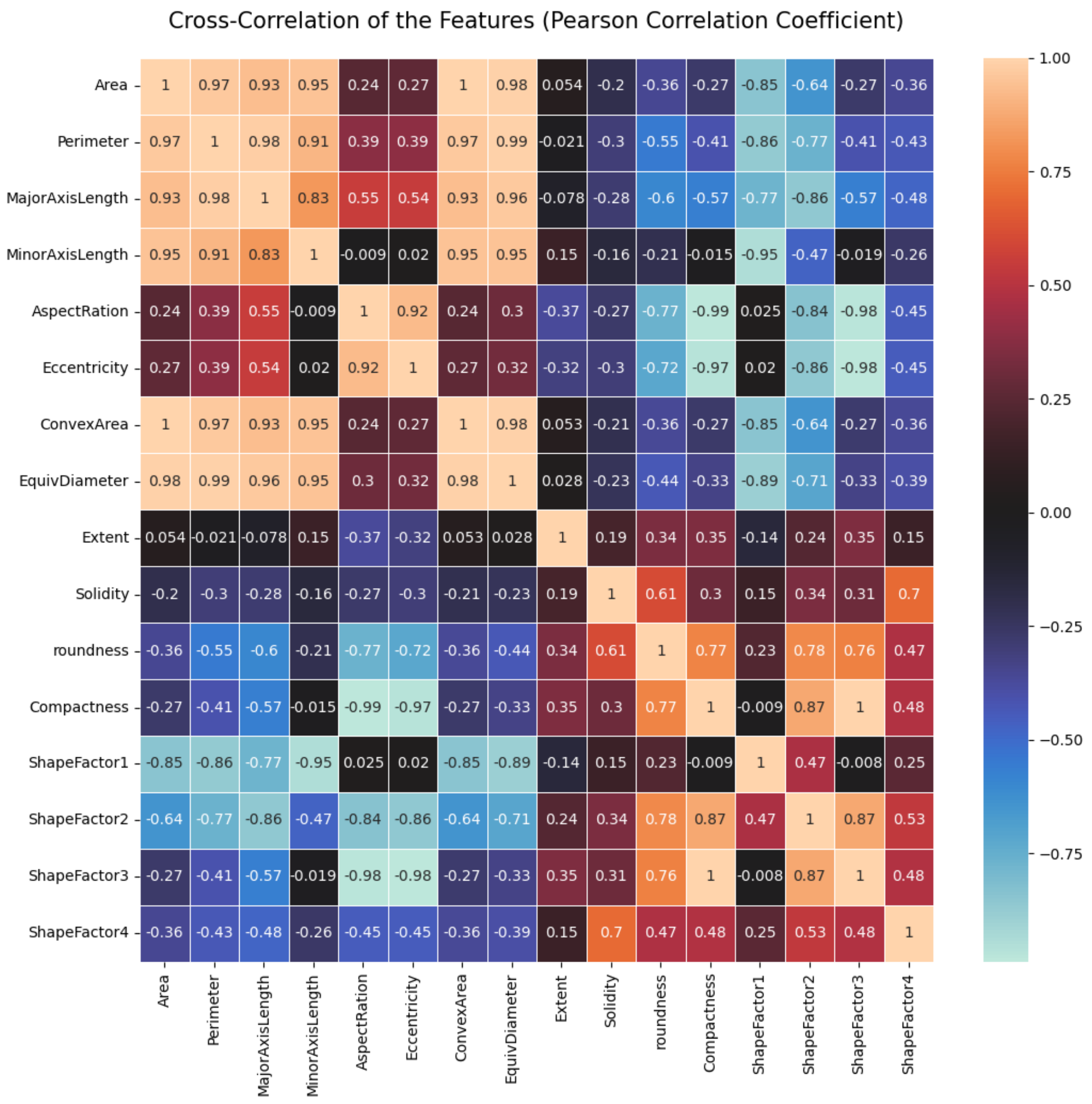


Figure 1.6: Cross-correlation of the 16 morphological features.

By looking at the cross-correlation matrix we can quantify the linear relationship between every pair of features. It is now possible to assess what we found by looking at the univariate and multivariate distributions of the features values:

- **Compactness** and **Shape Factor 3** are indeed linearly correlated, in fact their correlation is exactly 1;
- **Area**, **Perimeter**, **Major Axis Length**, **Minor Axis Length**, **Convex Area** and **Equivalent Diameter** are highly linearly correlated.

We will see in the next chapter how correlation can be exploited for improving the classification algorithm.

1.2.6 Target variable distribution

After having studied the distribution of feature values in multiple ways and having qualitatively observed how the different classes are partitioned in the feature space (at least in a restricted number of dimensions that we can physically understand), we also analysed the univariate distribution of the **Class** attribute.

In Figure 1.7 is reported the distribution of instances among the 7 different classes defined in the dataset. It is clear that there is non-negligible imbalance in the classes. This is due to the fact that the dataset was not built taking the same number of beans per class but taking the same amount of beans per class in terms of weight (1 kg). Due to the different average weight of the different varieties ([KO20]), the classes are imbalanced.

This is a problem we will deal with. Having unbalanced classes can lead to incorrect classification by the machine learning algorithm.

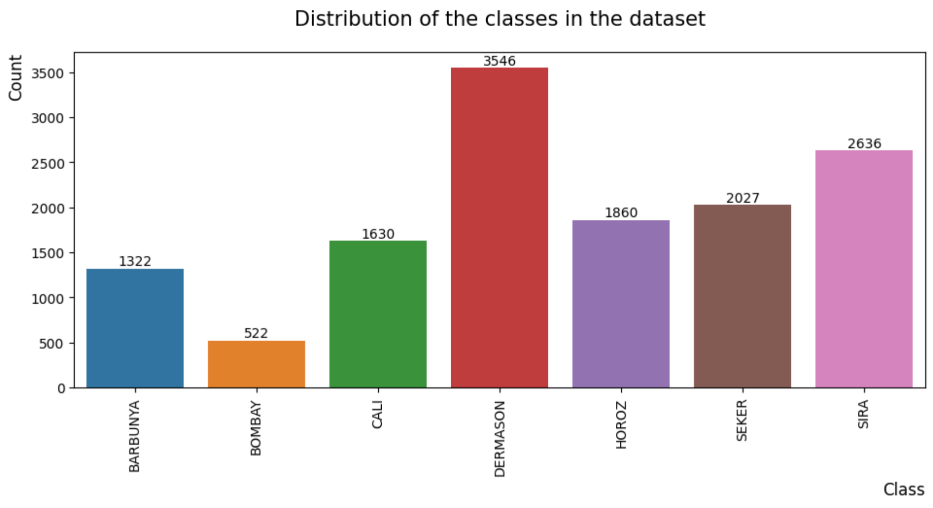


Figure 1.7: Distribution of values for the Class categorical attribute.

Chapter 2

Pre-processing

After having analysed the dataset, having understood the nature of the data and how it is distributed, we started to implement the actual machine learning algorithm. The task is to classify the 7 different dry beans varieties: do to this, a data-driven classification algorithm (a.k.a. a model) has to be implemented.

Before applying the classification algorithms we prepared the data in order to be sure to maximise their performance. Every operation performed before the classification can be included in the **pre-processing** stage. In this chapter we explore the pre-processing techniques that have been considered during the design of the machine learning algorithm pipeline.

In particular, we will present different techniques regarding feature scaling, feature selection and feature extraction. We will also present different ways to balance the number of instances of different classes in the dataset.

After having applied every feature selection/extraction technique, we created a new copy of every (pre-processed) dataset and store it in a `dictionary` of datasets. The original dataset was stored with the identifier `dataset_raw`.

2.1 Feature scaling techniques

Table 1.1 shows a summary of the statistical distributions of the different features. It can be easily observed that different features have values spanning different ranges of numbers, having ranges of different orders of magnitude. This can lead to problems working with different machine learning models, such as K-Nearest Neighbors (KNN), Support Vector Machines (SVMs) or Multilayer Perceptrons (MLPs). Tree-based methods such as Decision Trees (DTs), Random Forests (RFs) and Extra Trees (ETs) are instead scale-invariant.

The two common approaches to bring different features onto the same scale are **normalization** and **standardization**:

- *Normalization* refers to the re-scaling of the features to the range [0, 1], which is a special case of min-max scaling. Normalization is useful when the data is needed in the bounded intervals. To use this scaling technique, `sklearn.preprocessing.MinMaxScaler` was imported in the Python project.

- *Standardization* means to remove the features mean and set the standard deviation to 1 so that the feature columns have the same parameters as a standard normal distribution. To use this scaling technique, `sklearn.preprocessing.StandardScaler` was imported in the Python project.

2.1.1 MinMaxScaler vs StandardScaler

MinMaxScaler is useful when the data has a bounded range or when the distribution is not Gaussian. When dealing with non-Gaussian distributions, MinMaxScaler can be used to ensure that the range of values is scaled between 0 and 1.

StandardScaler is useful when the data has a Gaussian distribution or when the algorithm requires standardized features. Unlike normalization, standardization maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling, which scales the data to a limited range of values. When working with clustering algorithms such as KMeans, StandardScaler can be used to ensure that the features are standardized and contribute equally to the analysis.

2.2 Feature selection techniques

Feature selection is the process of reducing the number of features to input to the machine learning model. It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model.

Herein we propose a possible classification for the main feature selection techniques we have explored:

1. **Unsupervised**: do not use the target variable (remove *redundant* variables).
    - **Correlation**: study how different input variables (features) are related without considering the target variable.
2. **Supervised**: use the target variable (remove *irrelevant* variables).
    - **Filter**: select subsets of features based on their relationship with the target (e.g. univariate feature selection).
    - **Wrapper**: search for well-performing subsets of features (e.g. Recursive Feature Elimination - RFE).
    - **Intrinsic**: machine learning models that perform automatic feature selection during training (e.g. Decision Trees and Random Forests)

It is important to remove features that are not relevant for the classification problem, both in terms of redundancy (meaning that having two highly correlated features we can just keep one because the other is redundant) and irrelevancy (meaning that a feature, even with high variance, cannot contribute an effective partitioning of the data).

2.2.1 Correlation-based feature selection

During the analysis of the dataset we noticed that some subsets of features are highly correlated. Here we exploit the cross-correlation matrix to implement a feature selection algorithm. Using correlation is a (statistical) **unsupervised** feature selection technique, since we do not take into account the relationship between the different features and the target attribute (dry bean class).

Exploiting the cross-correlation matrix of the features

At first we used a manual approach of feature selection: by just looking at the cross-correlation matrix shown in Figure 1.6 we tried to remove the more correlated features. To emphasize the features with strong (linear) correlation we applied the **hyperbolic tangent function** to the obtained correlation measures. After this study we created a new copy of the dataset without some redundant features like **Perimeter**, **Major Axis Length**, **Minor Axis Length**, **Convex Area**, **Equivalent Diameter** and **Compactness**; we called the new dataset of 10 features `dataset_A`.

We later decided to implement an automatic and efficient pruning algorithm: scanning the cross-correlation matrix, each time an highly correlated pair is found (i.e. when the absolute value of the correlation higher than a given threshold) a metric is computed to decide which one of the two features needs to be dropped. The detailed procedure is reported in Algorithm 2.

We experimented with different values of the threshold and we finally set it to 0.8. The resulting cross-correlation matrix of the best features is shown in Figure 2.1. The resulting dataset of 6 features was stored in the dictionary of datasets and called `dataset_B`.

Algorithm 2

```
procedure PRUNING BY CORRELATION(dataset, $\theta = 0.8$).

    C = dataset.get_correlation().abs()            ▷ We work with the absolute values
    drop = []

    for i in C.rows() do
        for j in C.columns() do
            if C[i,j] $\geq \theta$ then
                feature i correlations sum = $\sum_{k=1}^{16} C[i,k]$
                feature j correlations sum = $\sum_{k=1}^{16} C[j,k]$
                if feature i correlations sum $\leq$ feature j correlations sum then
                    drop.append(j)
                else
                    drop.append(i)
    pruned_dataset = dataset.drop_features(drop)
```

Exploiting cross-correlation matrix of the features + target attribute (Class)

Correlation seems a good way to remove redundant features, but it doesn't take into account how features values are related to target values (as an unsupervised technique). In order to take into account also the **Class** target attribute in the cross-correlation matrix we thought about converting it from categorical to numerical. We implemented two different encoding techniques:

- **Integer encoding**: every class is encoded with a different integer from 0 to 6 according to the alphabetical order of the target labels. In this way we do not add any column to the dataset and the target attribute is **ordinal**.
- **One-hot encoding**: every class is encoded with a unique 7 bit binary sequence. Doing so, we do not impose any order on the classes and we increase the number of columns in the dataset.

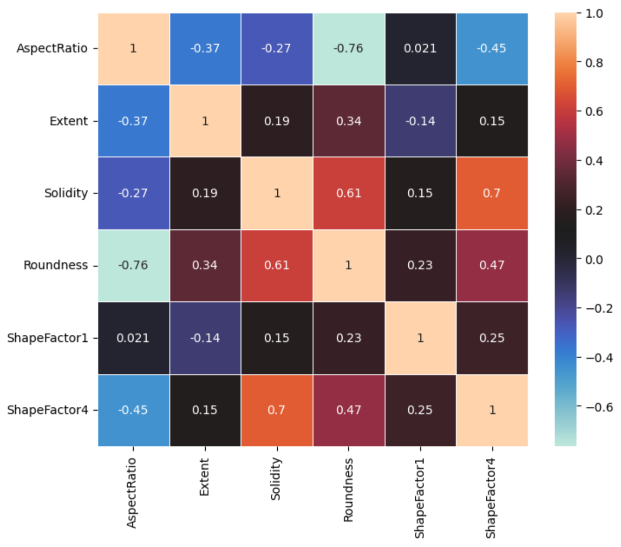


Figure 2.1: Cross-correlation matrix obtained after applying the pruning algorithm. With the threshold set at 0.8 only 6 features remain (10 features are dropped).

We tried to work in particular with the integer-encoded dataset: having all numerical attributes it is possible to compute the cross-correlation matrix taking into account the Class attribute along with the features. However, having imposed an order to the classes we did not know if the values of correlation were actually reliable. We explored the idea to compute the cross-correlation matrix using **Spearman correlation coefficient** (that has the perk of being able to work with ordinal attributes) but, again, we were not sure about the reliability of the results.

2.2.2 Feature importance with Random Forests

Random Forests are a popular ensemble learning method. While they are primarily used for predictive modeling, they can also be employed for feature selection.

The process of feature selection using Random Forests involves leveraging the inherent feature importance measures provided by the algorithm: during the construction of the Random Forest, the importance of each feature is measured; this importance metric is based on how much the feature contributes to reducing the impurity or error in the decision tree ensemble.

Once all the decision trees are built, the feature importance scores from each tree are averaged to obtain a final aggregated feature importance score for each feature. Features that are consistently used across many trees to make important splits will have higher importance scores. The features are ranked based on their importance scores and the top N features can be selected.

Random Forests are an example of intrinsic feature selection approach, since the selection happens naturally thanks to the way the ensemble method is designed. We made some tests using Random Forests for feature selection fixing N (number of components to keep) and storing the resulting pre-processed dataset of N features in the dictionary of dataset, calling it `dataset_C`.

2.2.3 Univariate feature selection

Univariate feature selection works by selecting the best features based on univariate statistical tests. Using statistics it is possible to select those input variables that have the strongest relationship with the target variable. **It is clear that these methods do not work with all the features collectively**.

This approach is an example of (statistical) supervised filter method since it ranks the features and keeps only the best ones according to the rank.

Univariate feature selection methods can be fast and effective, although the choice of statistical measures depends on the data type of both the input and output variables. In Figure 2.2 are shown which statistical metrics have to be used, according to the type of attributes we have in our dataset. According to this scheme, having numerical attributes and a categorical target attribute, we should work with the **ANOVA correlation coefficient**.

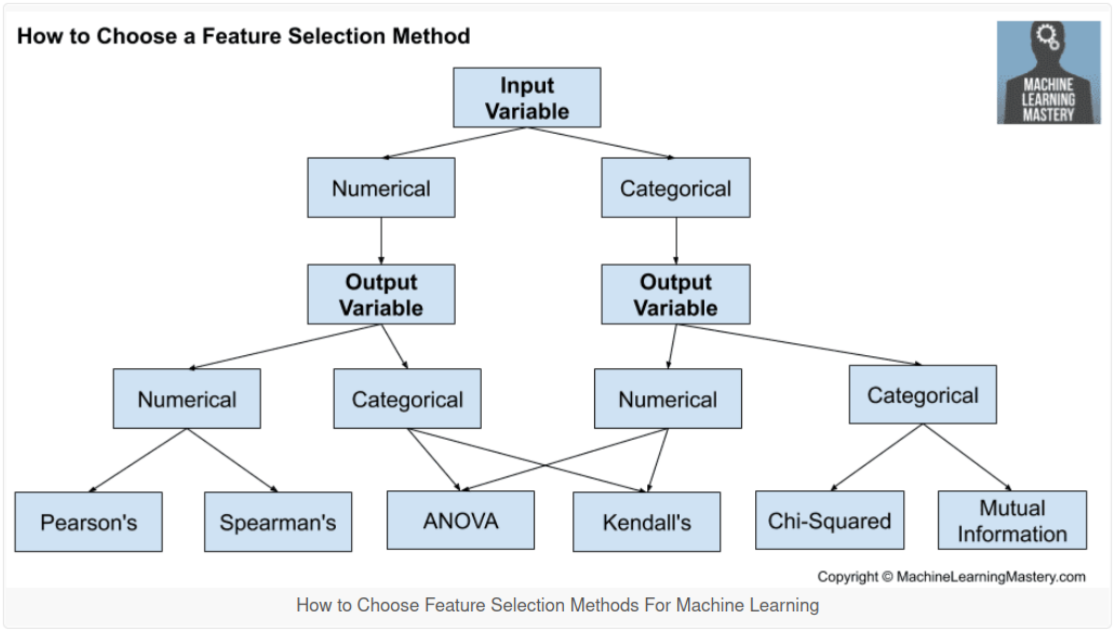


Figure 2.2: The statistical metrics used for different attributes types combinations. Courtesy of Machine Learning Mastery.

ANOVA F-test

In statistics, ANOVA is used to determine whether there is any statistically significant difference between the means of two or more groups. This is particularly useful in a classification problem where we want to know how well a continuous feature discriminates between multiple classes.

To implement the ANOVA F-test we imported `sklearn.feature_selection.SelectKBest`, and `sklearn.feature_selection.SelectPercentile`. The former returns the k-best features according to a pre-defined ranking metric. The latter returns the features with a score higher than a pre-defined percentage. The ANOVA F-test is performed by selecting the metric `sklearn.feature_selection.f_classif`.

We experimented with `SelectKBest`: executing the former with $k = 6$, the feature selection algorithm returns the following subset of features: **Area**, **Perimeter**, **Major Axis Length**, **Minor Axis Length**, **Convex Area** and **Equivalent Diameter**. It seems that since these 6 feature are highly correlated with each other, they are also all correlated with the **Class** distribution! This is the exact problem of univariate feature selection, i.e. each feature score is evaluated without considering the other features. For this reason we tried to execute again `SelectKBest` on `dataset_A` instead, where the redundant features were removed already.

The cross-correlation matrix of the best features according to `SelectKBest`, with $k = 6$, is shown in Figure 2.3. A new copy of `dataset_A` processed with univariate feature selection was stored in the dictionary of datasets and called `dataset_D`.

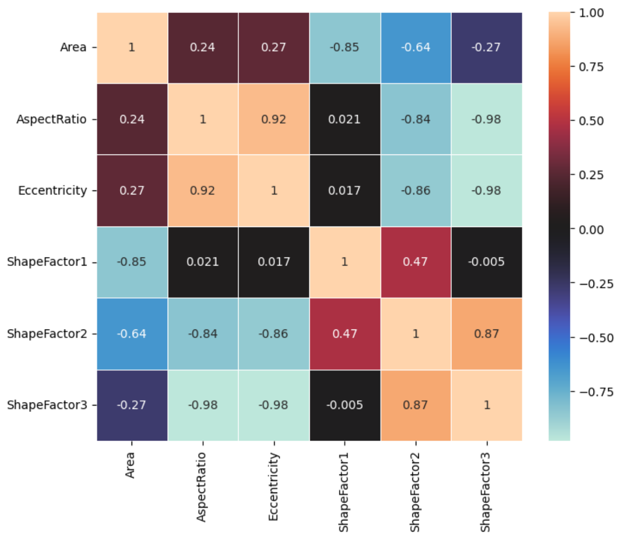


Figure 2.3: Cross-correlation matrix obtained after applying the ANOVA F-test on `dataset_A` and having kept the 6 most relevant features.

2.3 Feature extraction techniques

Feature extraction is a pre-processing technique that involves transforming the original features into a new set of features, typically of lower dimensionality. The objective is to capture essential information from the original features while discarding irrelevant or redundant information.

It seems similar to feature selection in the sense that both the approaches aim to dimensionalty reduction of the feature space. The main difference is that while feature selection retains the original features **and therefore their physical interpretation, if they have any**, feature extraction underlines the application of a transformation that returns new features without necessarily having a physical meaning.

2.3.1 Principal Component Analysis

Principal Component Analysis (PCA) aims to transform the original features into a new set of linearly uncorrelated features called principal components. These components are ordered in such a way that the first component captures the most significant variance in the data, the second component captures the second most significant variance, and so on. In Table 2.1 are reported the variances of the first 10 Principal Components after having applied PCA to the original features; it is possible to see how almost all the variance is contained in the first principal components.

PCA is a **linear transformation** based on Singular Value Decomposition (SVD) of rectangular matrices. Through PCA, the original feature space (with physical meaning) is subject to a change of basis by the application of a diagonalization on the features covariance matrix. The new orthogonal basis is the one with eigenvectors relative to the directions of maximum variance in the feature space. In Figure 2.4 it is possible to observe the cross-correlation matrix of the features before and after the application of the PCA on the dataset; the result is pretty obvious thinking that PCA is a SVD that diagonalizes the covariance matrix of the features.

PCA is of course a unsupervised feature extraction method as it does not take into account the distribution of the **Class** target attribute in the feature space; PCA just aims to transform the original features in a new set of features that are linear combinations of the original ones, in such a way that the new feature are uncorrelated. **In our case it seems that PCA does not mix up different classes when reducing the dimension of the feature space**; it is possible to observe this in the 2D and 3D feature spaces reported in Figure 2.5.

A version of the dataset, pre-processed with PCA, is stored in the dictionary of datasets and named `dataset_PCA`.

Number of PCA components	1	2	3	4	5	6	7	8	9	10
Single component variance	55.546%	26.360%	7.984%	5.108%	2.753%	1.152%	0.698%	0.325%	0.051%	0.009%
Cumulative variance	55.546%	81.907%	89.892%	95.000%	97.753%	98.905%	99.603%	99.929%	99.981%	99.990%

Table 2.1: Explained variance of the first 10 Principal Components.

2.4 Class balancing techniques

Class balancing refers to the process of addressing imbalances in the distribution of different values of a target attribute in the dataset. In our dataset, the **Class** target attribute is imbalanced because of the way the dataset was built:1 kg of beans of each variety was used, but the varieties have different average weight [KO20].

Class imbalance can lead to several challenges during the training and evaluation of machine learning models:

- **Bias**: models may be biased towards the majority class because they have more examples to learn from.
- **Poor generalization**: models may struggle to learn patterns from the minority class, resulting in lower performance on the underrepresented class.
- **Evaluation metrics**: traditional evaluation metrics like accuracy may not be suitable as they can be misleading in the presence of imbalanced data.

2.4.1 Resampling

To address for class imbalance we explored different **resampling methods**: this involves either oversampling the minority class (creating synthetic samples) or undersampling the majority class (removing real samples). Class balancing should be applied carefully, as oversampling or undersampling can introduce biases or noise.

All the resampling methods explained herein were imported in the project by means of the library `imblearn.combine`. The effects of the different resempling techniques are showed in Figure 2.6.

Cross-validation and performance evaluation on an independent test set are essential to ensure the effectiveness of class balancing techniques. These resampling methods should be therefore used only on the portion of data we use for training the classification models.

SMOTE (oversampling)

Synthetic Minority Over-sampling Technique (SMOTE) is an oversampling technique that creates synthetic samples for the minority class by interpolating between existing minority class samples.

SMOTE first identifies the minority classes and for every sample in the minority class it randomly selects one of their nearest neighbors from the same class. Synthetic samples are generated by linearly interpolating between each sample and the chosen nearest neighbor. By generating synthetic samples, SMOTE helps to balance the class distribution, ensuring that the minority class has a greater representation in the dataset.

The number of synthetic samples generated per minority sample is **fixed and user-defined**.

SMOTE has its limitations: if the data is too sparse or the number of neighbors is too large, SMOTE can generate noisy samples, which may impact model performance. There's a risk of overfitting on the synthetic data.

In Figure 2.6a is possible to see the result of SMOTE on the target distribution.

ADASYN (oversampling)

ADASYN (Adaptive Synthetic Sampling) is another oversampling technique.

like SMOTE, ADASYN creates synthetic samples by interpolating between minority class samples and their nearest neighbors. However, **the number of synthetic samples generated per minority sample is adaptive** and depends on the data distribution. ADASYN assigns higher weights to samples that are more challenging to classify, thus focusing on regions with higher class overlap. ADASYN's adaptive nature makes it particularly useful when dealing with severe class imbalances and in regions where class overlap is high.

In Figure 2.6b is possible to see the result of ADASYN on the target distribution.

SMOTE + TOMEK (hybridization)

Tomek links are pairs of instances from different classes that are close to each other but are of different classes. The distance between instances from different classes is calculated and if two instances are each other's nearest neighbors and belong to different classes, they are used to form a Tomek link. For each identified Tomek link, the instance from the majority class is removed, this helps in cleaning up noisy or borderline samples near the decision boundary.

By combining SMOTE and Tomek links, we can effectively oversample the minority class with synthetic instances while removing the noisy samples from both classes. The combined method is often referred to as "SMOTE-Tomek" and can be beneficial in improving the performance of classifiers on imbalanced datasets.

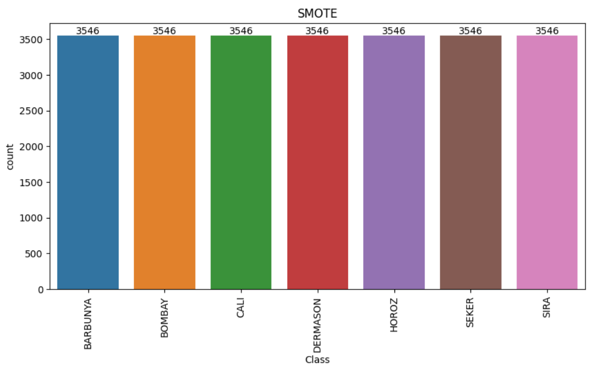
In Figure 2.6c is possible to see the result of SMOTE + TOMEK on the target distribution.

SMOTE + ENN (hybridization)

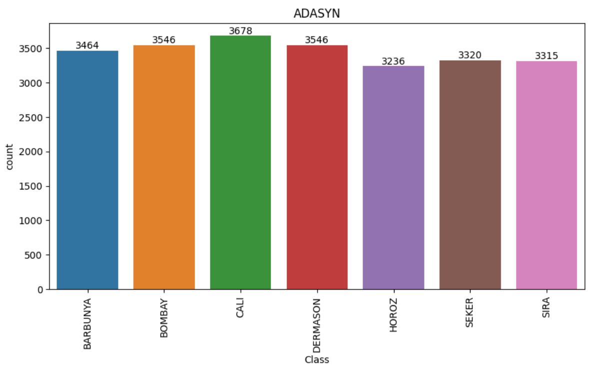
This resampling method is an hybridization since it combines SMOTE, to generate synthetic examples for minority classes, and ENN (Edited Nearest Neighbor), to delete some observations.

The ENN algorithm removes some of the samples from the majority class to balance the dataset. It looks at the instances of the majority class and determines which ones are misclassified by their k-nearest neighbors from the minority classes. It then removes those misclassified instances.

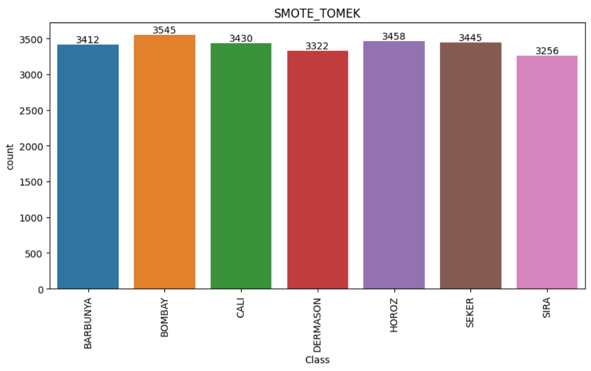
In Figure 2.6d is possible to see the result of SMOTE + ENN on the target distribution.



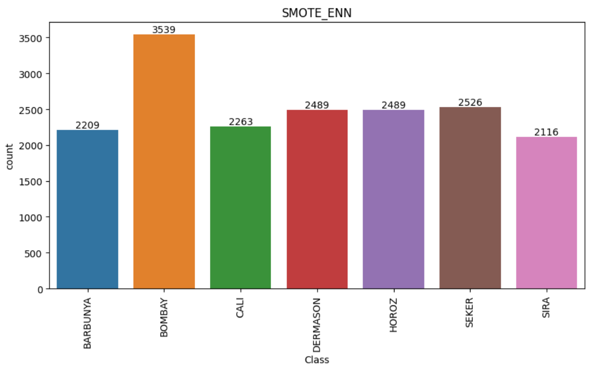
(a) SMOTE (oversampling).



(b) ADASYN (oversampling).



(c) SMOTE + TOMEK (hybridization).



(d) SMOTE + ENN (hybridization).

Figure 2.6: Training and testing accuracy for different classifiers with/without applying feature selection and class balancing.

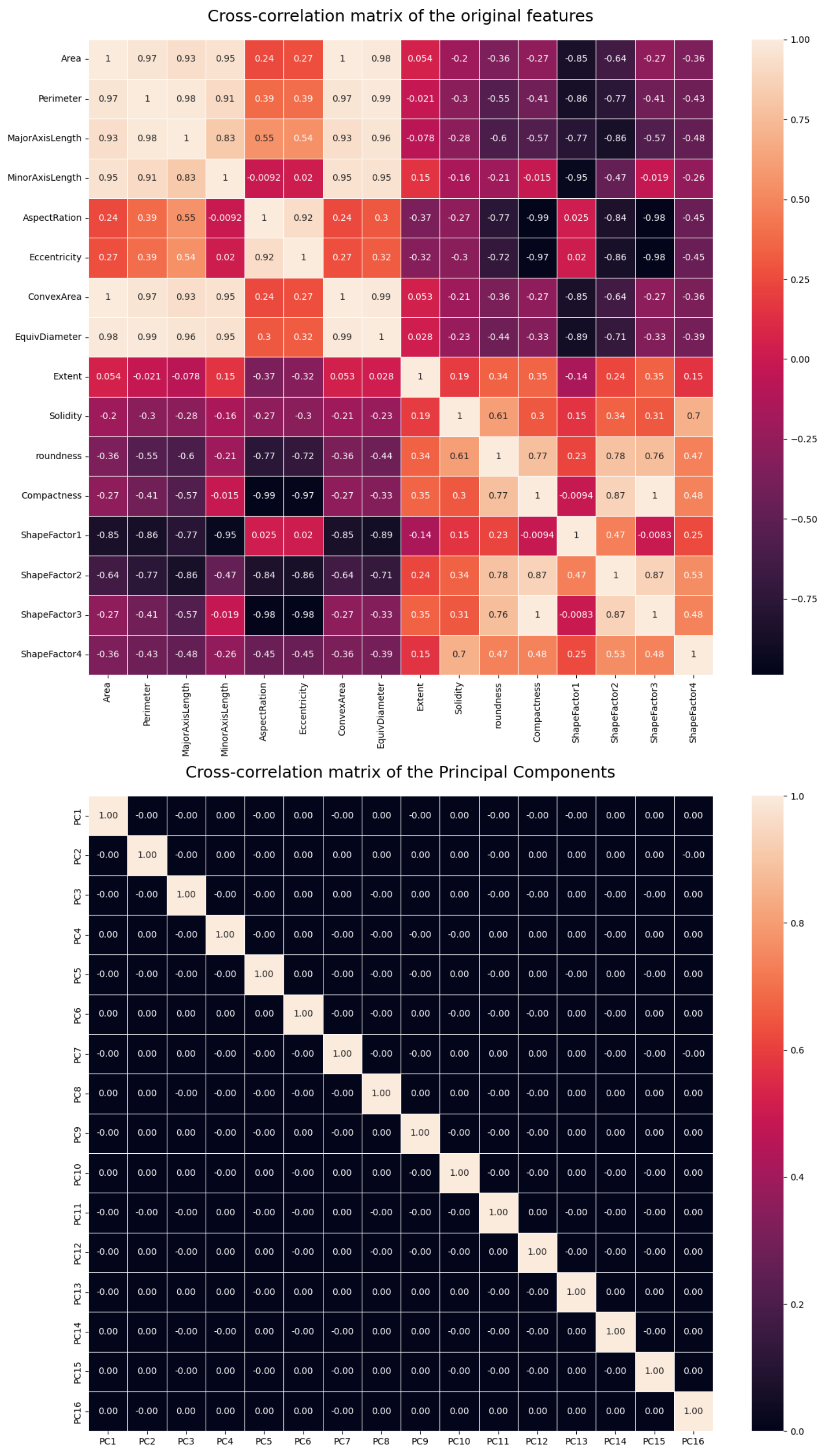
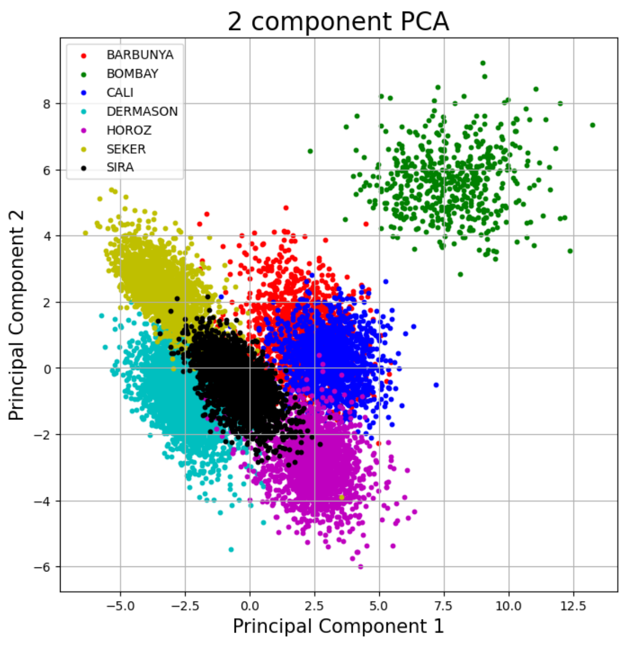
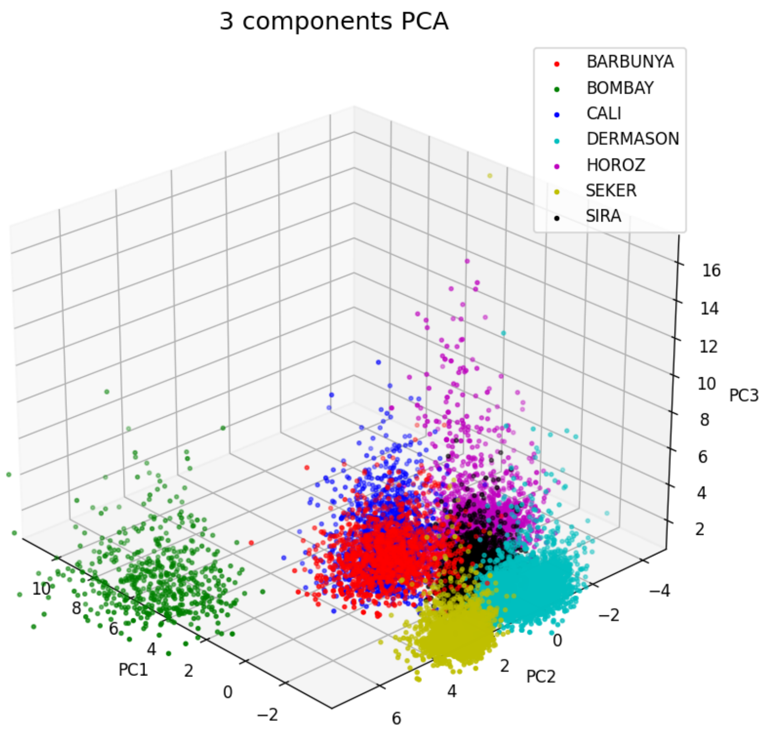


Figure 2.4: Cross-correlation matrix of the original features (above) and the same matrix on the Principal Components obtained applying PCA.



(a) Feature space of the first 2 principal components.



(b) Feature space of the first 3 principal components.

Figure 2.5: Data visualization with PCA.

Chapter 3

Classification

After having applied different pre-processing techniques and having obtained different versions of the original dataset, we can finally perform classification. As already mentioned, a dictionary of different datasets was created in order to easily select on which dataset perform the classification.

The different datasets consist of different feature selection or feature extraction techniques. The application of other pre-processing techniques, such as scaling and class balancing, is actually included in the classification since their application depends:

1. on the type of selected classification algorithm, e.g. SVMs are sensitive to scale while tree-based methods are scale-invariant;

2. on the train/test partitioning of the data: class balancing by resampling needs to be performed on the training dataset only, since with oversampling we create artificial data that cannot be used for the validation of the classification models in the testing phase.

In this chapter we present the results obtained applying different classification algorithms. We report the results of a first naive classification, without proper hyperparameters selection for the chosen models and the results after having applied a **grid search** for selecting the best hyperparameters for every model.

We decided to report only the results of the classification performed with `dataset_B`, obtained after having performed feature selection with Algorithm 2, because they seem to be the best ones among the explored feature selection/extraction methods.

3.1 Proposed classifiers and naive classification results

First we decided which of the classification algorithms studied during the course were more suitable for the dataset. We decided to work with:

- Decition Trees;

- Random Forests;

- Extra Trees;

- K-Nearest Neighbors;

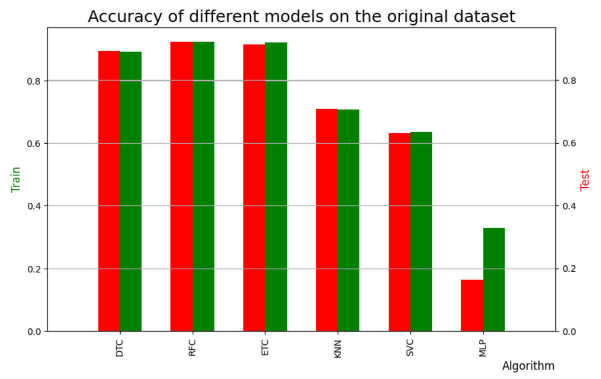
- Support Vector Machines;

- Multilayer Perceptrons.

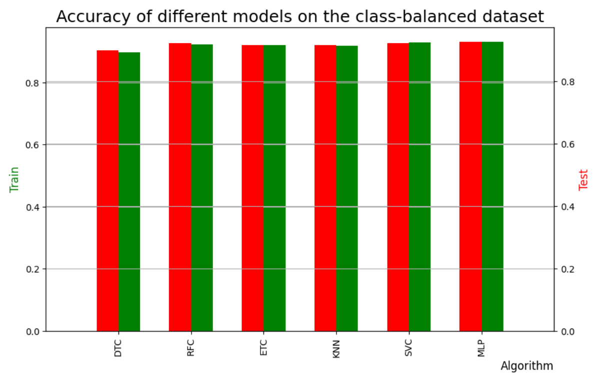
All these algorithms have different pros and cons. We have not found any specific reasons why to work with a specific classifier since all the features are numerical and we are able to scale the data before applying any of these algorithms,. Furthermore, explainability is not a specification of the problem. All the classification algorithms were imported from the `sklearn` library.

The listed classifiers were used to fit different versions of the dataset to assess the effect of pre-processing. We trained all the different classifiers using 70% of the data, trying to maximise for accuracy, with and without applying different pre-processing techniques, the results are showed in Figure 3.1.

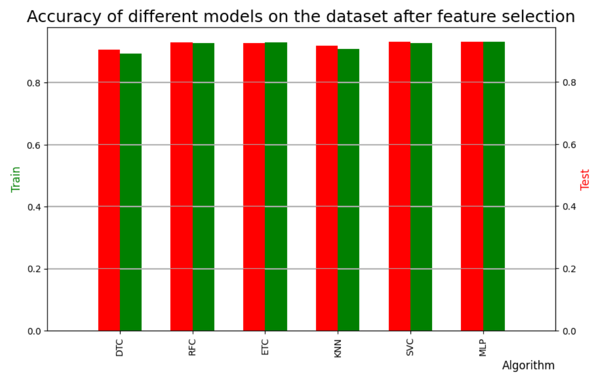
By looking at these approximate results it seems clear that the scale-variant models such as KNN, SVM and MLP are not performing well on the original dataset. After applying scaling and class balancing it is possible to see that all the models perform really well in terms of accuracy. Finally, applying either feature selection or feature extraction (thus having a dimensionality reduction) all the models still perform really well.



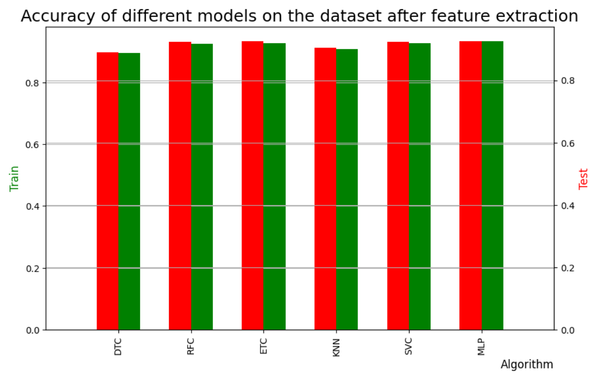
(a) Classifiers results on the original dataset. KNN, SVM and MLP obviously suffer from the absence of proper scaling.



(b) Classifiers results on the original dataset after standard scaling and resampling (SMOTE + TOMEK).



(c) Classifiers results on the dataset after standard scaling, resampling (SMOTE + TOMEK) and feature selection (Algorithm 2).



(d) Classifiers results on the dataset after after standard scaling, resampling (SMOTE + TOMEK) and feature extraction (8D PCA).

Figure 3.1: Training and testing accuracy for different classifiers with/without applying feature selection and class balancing.

3.2 Grid Search for hyperparameter selection

In order to further explore the different proposed classifiers and do training correctly we imported `sklearn.modelselection.GridSearchCV`. With this module we could implement hyperparameter selection routines, training the classifiers with multiple combinations of hyperparameters. Selection of the best parameters is done automatically by the `GridSeachCV` object, that performs training exploiting **k-fold cross validation**, and finally evaluates the models found with the best hyperparameters on the left-out test set.

In our project, we set the amount of training data to the 70% of the instances and performed k-fold cross validation with 5 folds. In order to apply class-balancing correctly, a `Pipeline` object was instantiated (imported from `imblearn.pipeline`): with pipelines it is possible to concatenate pre-processing operations to the classifiers in order to correctly execute them and include them in the grid search method. This last feature of pipelines makes possible to search for the best parameters for both the pre-processing techniques and the classifiers (similar to the concept of wrapper feature selection).

3.3 Classification results

Here we report the results of the classification for every classification algorithm. We obtained very similar results with different algorithms: every classifier was able to predict perfectly the **Bombay** class, that was clearly well separated from the other dry bean varieties. The worst predicted class was the **Sira** variety.

In particular, studying the confusion matrices it is even more clear that all the algorithms performed well but struggled with the same classes: from the matrices is possible to see that **Sira** beans are often misclassified as **Dermason** beans, an viceversa. Another pair of classes that are confused by the algorithms are **Barbunya** and **Cali**.

The classification results for each algorithm are reported in the following subsections as well as the best hyperparameters that led to these results.

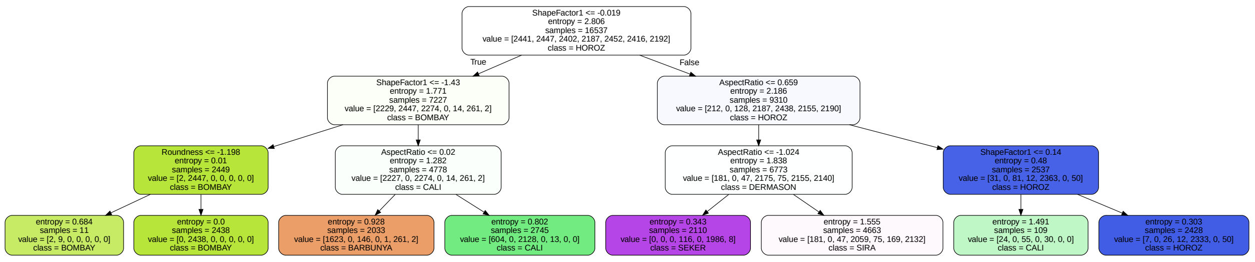


Figure 3.2: Example of Decision Tree with depth 3 learned from `dataset_B` (6 features). It can be seen how the model is explainable even though with this limited depth it has an accuracy of 63%.

3.3.1 Decision Trees

Decision Trees performed well with a **testing accuracy of 91%**. To get this result the **decision tree max depth** was set to **12** and the strategy used to split the nodes was to **choose the best splits** according to the obtained impurity reduction. The **criterion** for the impurity computation was the **GINI index**.

In Table 3.1 are reported the classification results for the Decision Tree algorithm and in Figure 3.3 is shown the confusion matrix computed on the test set.

To have an idea of way the decision tree is an explainable model, an example of short decision tree is shown in Figure 3.2.

Class	Precision	Recall	F-score	Support
BARBUNYA	0.89	0.92	0.9	396
BOMBAY	1.0	1.0	1.0	161
CALI	0.93	0.92	0.92	473
DERMASON	0.92	0.9	0.91	1065
HOROZ	0.93	0.95	0.94	553
SEKER	0.93	0.95	0.94	618
SIRA	0.86	0.85	0.86	797
accuracy			0.91	4063
macro avg	0.92	0.93	0.92	4063
avg	0.91	0.91	0.91	4063

Table 3.1: Classification Report of **Decision Trees**.

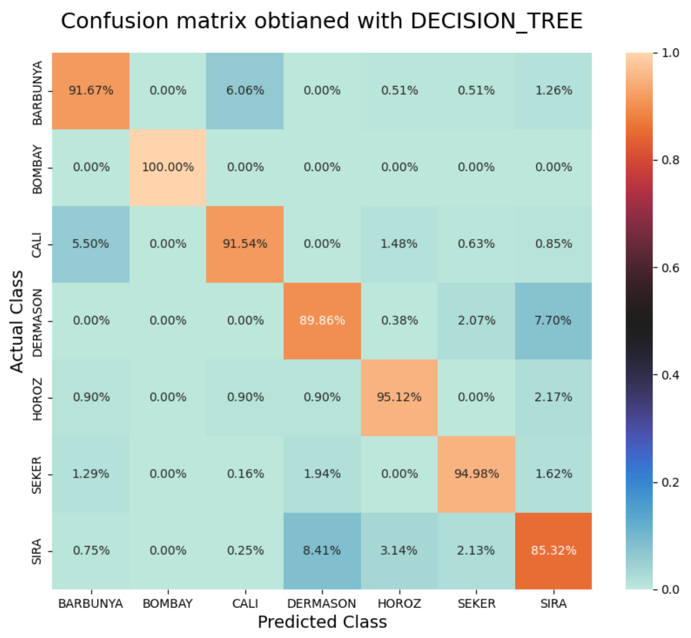


Figure 3.3: Confusion matrix resulting from **Decision Trees**.

3.3.2 Random Forests

Random Forests performed well with a **testing accuracy of 93%**. To get this result we set the forests to use **use bootstrap** and we used **128 estimators** (trees). The **criterion** for splitting was the **entropy** and the number of **max features** when looking for the best spilt was $\mathbf{log_2(features)}$.

In Table 3.2 are reported the classification results for the Random Forest algorithm and in Figure 3.4 is shown the confusion matrix computed on the test set.

Class	Precision	Recall	F-score	Support
BARBUNYA	0.93	0.94	0.93	396
BOMBAY	1.0	0.99	1.0	161
CALI	0.95	0.94	0.94	473
DERMASON	0.93	0.92	0.92	1065
HOROZ	0.96	0.96	0.96	553
SEKER	0.95	0.95	0.95	618
SIRA	0.87	0.88	0.87	797
accuracy			0.93	4063
macro avg	0.94	0.94	0.94	4063
avg	0.93	0.93	0.93	4063

Table 3.2: Classification Report of **Random Forests**.

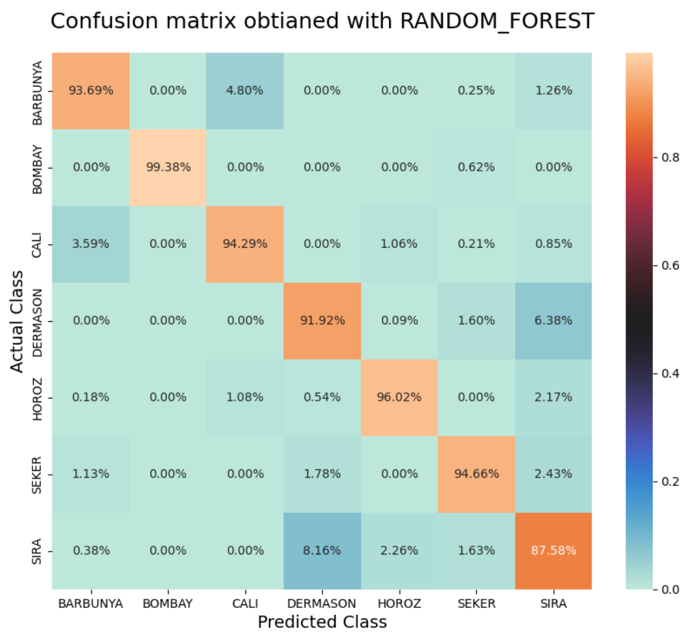


Figure 3.4: Confusion matrix resulting from **Random Forests**.

3.3.3 Extra Trees

Extra Trees performed well with a **testing accuracy of 93%**. To get this result we used **256 estimators** (trees). The **criterion** for measuring the quality of a split was the **log loss** and the number of **max features** when looking for the best spilt was $\sqrt{\textbf{features}}$.

In Table 3.3 are reported the classification results for the Extra Trees algorithm and in Figure 3.5 is shown the confusion matrix computed on the test set.

Class	Precision	Recall	F-score	Support
BARBUNYA	0.93	0.92	0.93	396
BOMBAY	1.0	1.0	1.0	161
CALI	0.94	0.95	0.94	473
DERMASON	0.92	0.92	0.92	1065
HOROZ	0.95	0.96	0.95	553
SEKER	0.94	0.94	0.94	618
SIRA	0.87	0.87	0.87	797
accuracy			0.93	4063
macro avg	0.94	0.94	0.94	4063
avg	0.93	0.93	0.93	4063

Table 3.3: Classification Report of **Extra Trees**.

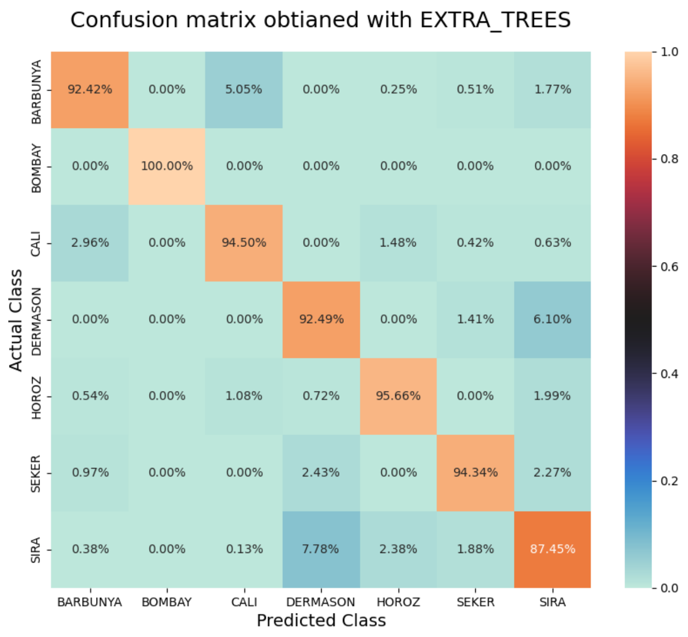


Figure 3.5: Confusion matrix resulting from **Extra Trees**.

3.3.4 K-Nearest Neighbors

K-Nearest Neighbors performed well with a **testing accuracy of 91%**. To get this result we set **K=15** and the **Euclidean norm** to compute the distances. We used the **"ball tree" algorithm** to compute the nearest neighbors. When finding the k-nearest neighbors we gave the neighboring samples a **weight that is inversely proportional to the distance from the current sample**.

In Table 3.4 are reported the classification results for the K-Nearest Neighbors algorithm and in Figure 3.6 is shown the confusion matrix computed on the test set.

Class	Precision	Recall	F-score	Support
BARBUNYA	0.93	0.92	0.92	396
BOMBAY	1.0	1.0	1.0	161
CALI	0.91	0.94	0.92	473
DERMASON	0.93	0.88	0.9	1065
HOROZ	0.95	0.95	0.95	553
SEKER	0.95	0.95	0.95	618
SIRA	0.82	0.88	0.85	797
accuracy			0.91	4063
macro avg	0.93	0.93	0.93	4063
avg	0.92	0.91	0.91	4063

Table 3.4: Classification Report of **K-Nearest Neighbors**.

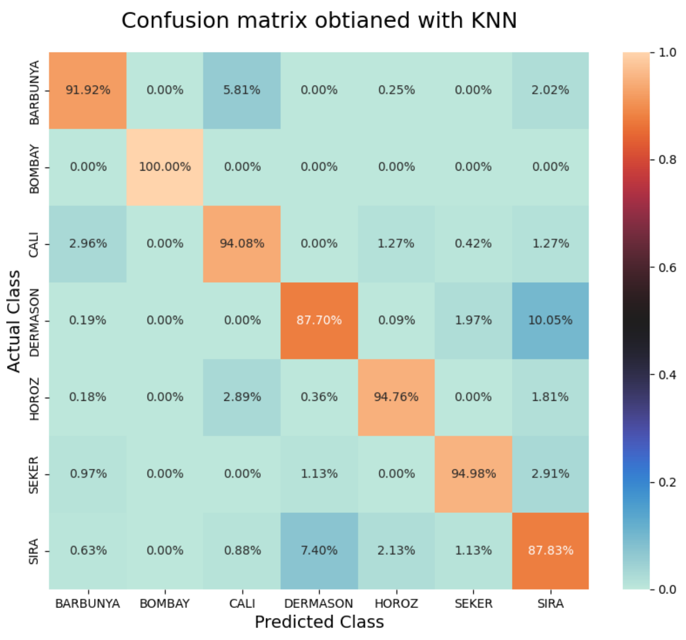


Figure 3.6: Confusion matrix resulting from **K-Nearest Neighbors**.

3.3.5 Support Vector Machines

Support Vector Machines performed well with a **testing accuracy of 93%**. To get this result we selected the **radial basis function kernel** with **$\gamma = 0.01$ (sample radius of influence)**, **regularization parameter $C = 100$** and a **multiclass classification strategy one-vs-one** where a binary classifier is created for every pair of classes. We wigthted the samples in a uniform way since class balancing was previously performed, but it is also possible to **weight every sample according to the relative frequency of the class** it belongs to. In Table 3.4 are reported the classification results for the Support Vector Machine algorithm and in Figure 3.6 is shown the confusion matrix computed on the test set.

Class	Precision	Recall	F-score	Support
BARBUNYA	0.93	0.93	0.93	396
BOMBAY	1.0	1.0	1.0	161
CALI	0.94	0.95	0.94	473
DERMASON	0.92	0.91	0.92	1065
HOROZ	0.96	0.95	0.96	553
SEKER	0.95	0.95	0.95	618
SIRA	0.86	0.88	0.87	797
accuracy			0.93	4063
macro avg	0.94	0.94	0.94	4063
avg	0.93	0.93	0.93	4063

Table 3.5: Classification Report of **Support Vector Machines**.

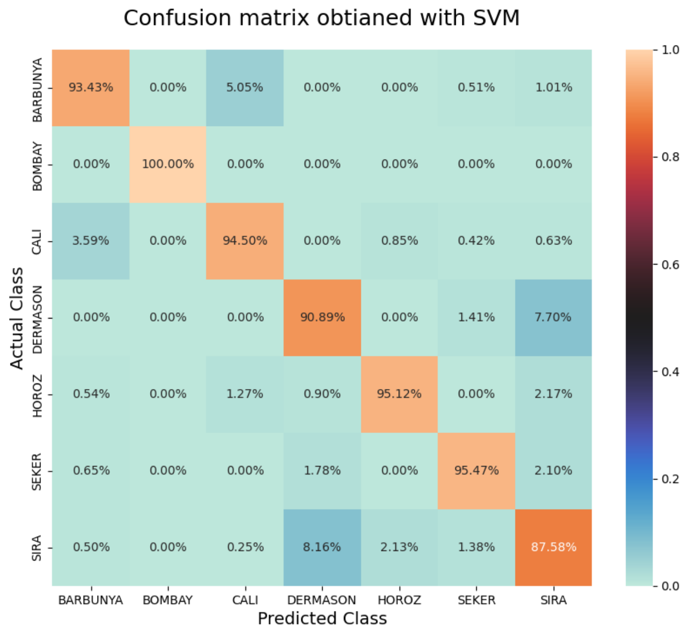


Figure 3.7: Confusion matrix resulting from **Support Vector Machines**.

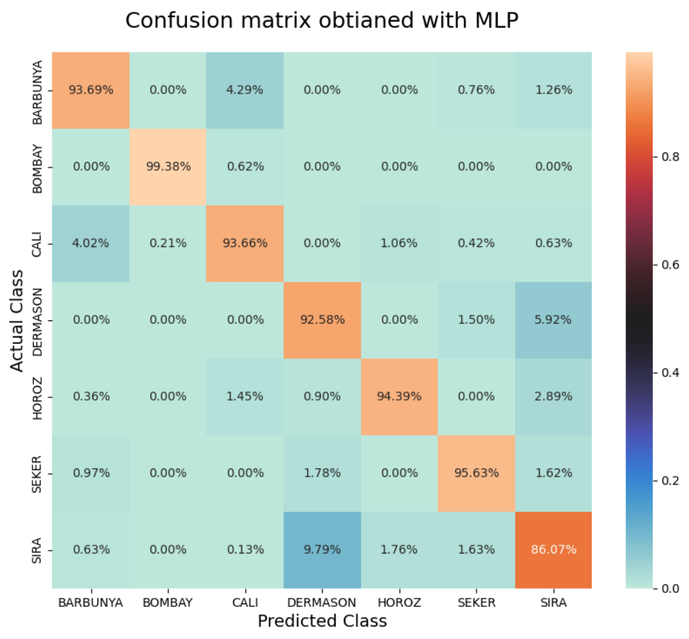
3.3.6 Multilayer Perceptrons

Multilayer Perceptrons performed well with a **testing accuracy of 93%**. To get this result selecting a (fully connected) architecture composed of **2 hidden layers with 16 perceptrons per layer**. We tried to increase both the number of perceptrons per layer (up to 256) and also to increase the depth of the network, but the algorithm did not performed better. We also tried to "code" the feature by decreasing the number of perceptrons per layer to 8 or 4, but we did not get better results. We used the **sigmoid (logistic) activation function** and a **constant learning rate** of **0.01**. The **momentum** was set to **0.9**.

In Table 3.6 are reported the classification results for the Multilayer Perceptrons algorithm and in Figure 3.8 is shown the confusion matrix computed on the test set.

Class	Precision	Recall	F-score	Support
BARBUNYA	0.92	0.94	0.93	396
BOMBAY	0.99	0.99	0.99	161
CALI	0.94	0.94	0.94	473
DERMASON	0.91	0.93	0.92	1065
HOROZ	0.96	0.94	0.95	553
SEKER	0.95	0.96	0.95	618
SIRA	0.88	0.86	0.87	797
accuracy			0.93	4063
macro avg	0.94	0.94	0.94	4063
avg	0.93	0.93	0.93	4063

Table 3.6: Classification Report of **Multilayer Perceptrons**.



	BARBUNYA	BOMBAY	CALI	DERMASON	HOROZ	SEKER	SIRA
BARBUNYA	93.69%	0.00%	4.29%	0.00%	0.00%	0.76%	1.26%
BOMBAY	0.00%	99.38%	0.62%	0.00%	0.00%	0.00%	0.00%
CALI	4.02%	0.21%	93.66%	0.00%	1.06%	0.42%	0.63%
DERMASON	0.00%	0.00%	0.00%	92.58%	0.00%	1.50%	5.92%
HOROZ	0.36%	0.00%	1.45%	0.90%	94.39%	0.00%	2.89%
SEKER	0.97%	0.00%	0.00%	1.78%	0.00%	95.63%	1.62%
SIRA	0.63%	0.00%	0.13%	9.79%	1.76%	1.63%	86.07%

Figure 3.8: Confusion matrix resulting from **Multilayer Perceptrons**.

Chapter 4

Conclusions

In the making of this project we explored different machine learning techniques to build a data-driven classification model for the DryBeans dataset. Given the morphological features of the instances we wanted to predict the type of bean variety (that is, our target attribute).

We analysed the dataset in depth, understanding the nature of the features and their multivariate and marginals distributions. This helped us get a proper understanding of the data we were working with and anticipate the results of the feature selection methods. Furthermore, we studied how the data instances were distributed in the different classes.

We applied several pre-processing techniques, including feature selection, feature extraction and class balancing. We created new versions of the dataset according to the different pre-processing applied, in order to understand how these affected the classification performance in terms of accuracy.

We explored different classifiers that we studied during the course. We performed grid search on each model for optimal hyperparameters selection using k-fold cross validation. The results of every model obtained with the best found hyperparameters were reported in tables containing accuracy and other metrics, in addition to their confusion matrix.

The project was overall an interesting a didactic experience that helped us familiarise with popular Python frameworks necessary to work in the data analysis field.

Bibliography

[KO20] Murat Koklu and Ilker Ali Ozkan. “Multiclass classification of dry beans using computer vision and machine learning techniques”. In: *Computers and Electronics in Agriculture* 174 (2020). ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2020.105507. URL: https://www.sciencedirect.com/science/article/pii/S0168169919311573.