Statistical Methods in Data Mining

Instituto Superior Técnico

1st Semester - 2021/22

Project 1

Group 6:

Catarina Alexandra Saleiro Rodrigues	92434
Cristiano José Monte Mendonça	102355
Filipa Barros Costa	92626
Matheus Monteiro Casagrandi	101763
Natalija Stanojlovic	101644

Professor:
Conceição Amado

November 17, 2021

Contents

1 Introduction

The aim of this study is to predict the varieties of dry beans using binary classification. The problem consists of seven different types of dry beans used in a research, taking into account the variables such as form, shape, type, and structure by the market situation. For the classification model, images of 13,611 grains of 7 different registered dry beans were taken with a high-resolution camera. Bean images obtained by computer vision system were subjected to segmentation and feature extraction stages, and a total of 16 variables; 12 contain information regarding different dimension measures and the remaining about shape forms, were obtained from the grains. The original data had observations referring to 7 different bean varieties, in this study's case, only two varieties were considered. Therefore, the objective of this work is to apply models that correctly determine the variety. The variables associated to the data are:

- **Area (A)**- The area of a bean zone and the number of pixels within its boundaries.
- **Perimeter (P)**- Bean circumference is defined as the length of its border.
- **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.
- **Aspect ratio (K)**- Defines the relationship between L and l.
- **Eccentricity (Ec)**- Eccentricity of the ellipse having the same moments as the region.
- **Convex area (C)**- Number of pixels in the smallest convex polygon that can contain the area of a bean seed.
- **Equivalent diameter (Ed)**- The diameter of a circle having the same area as a bean seed area.
- **Extent (Ex)**- The ratio of the pixels in the bounding box to the bean area.
- **Solidity (S)**- Also known as convexity. The ratio of the pixels in the convex shell to those found in beans.
- **Roundness (R)**- Calculated with the following formula: $\frac{(4A)}{(P^2)}$.
- **Compactness (CO)**- Measures the roundness of an object: Ed/L.
- **ShapeFactor1 (SF1)**.
- **ShapeFactor2 (SF2)**.
- **ShapeFactor3 (SF3)**.
- **ShapeFactor4 (SF4)**.
- **Response variable:** Beans can be one of two varieties, BOMBAY or DERMASON.

This report consists of 6 sections and is organized as follows: Section 2 describes the dataset and some preliminary analysis, Section 3 describes the methodology, Section 4 describes the classification methods, Section 5 presents the experimental findings and Section 6 concludes.

2 Exploratory Data Analysis

The dataset for this specific analysis contains 4068 observations, with 16 variables. The response variable of dry beans varieties can take two possible outcomes - DERMOSAN and BOMBAY. The remaining 15 variables (called explanatory variables) are of a real nature. Doing an exploratory analysis of the dataset will give us an understanding of the data.

Starting the analysis by checking if there are missing values (NaN) in the dataset or if there are duplicate observations, which do not exist. Thereafter, one continued by investigating the variation between the variables and the co-variation. This is can be observed in Figure 1. From the plot bellow it is noticeably that the data is clearly distributed in two clusters for most of the predictors.

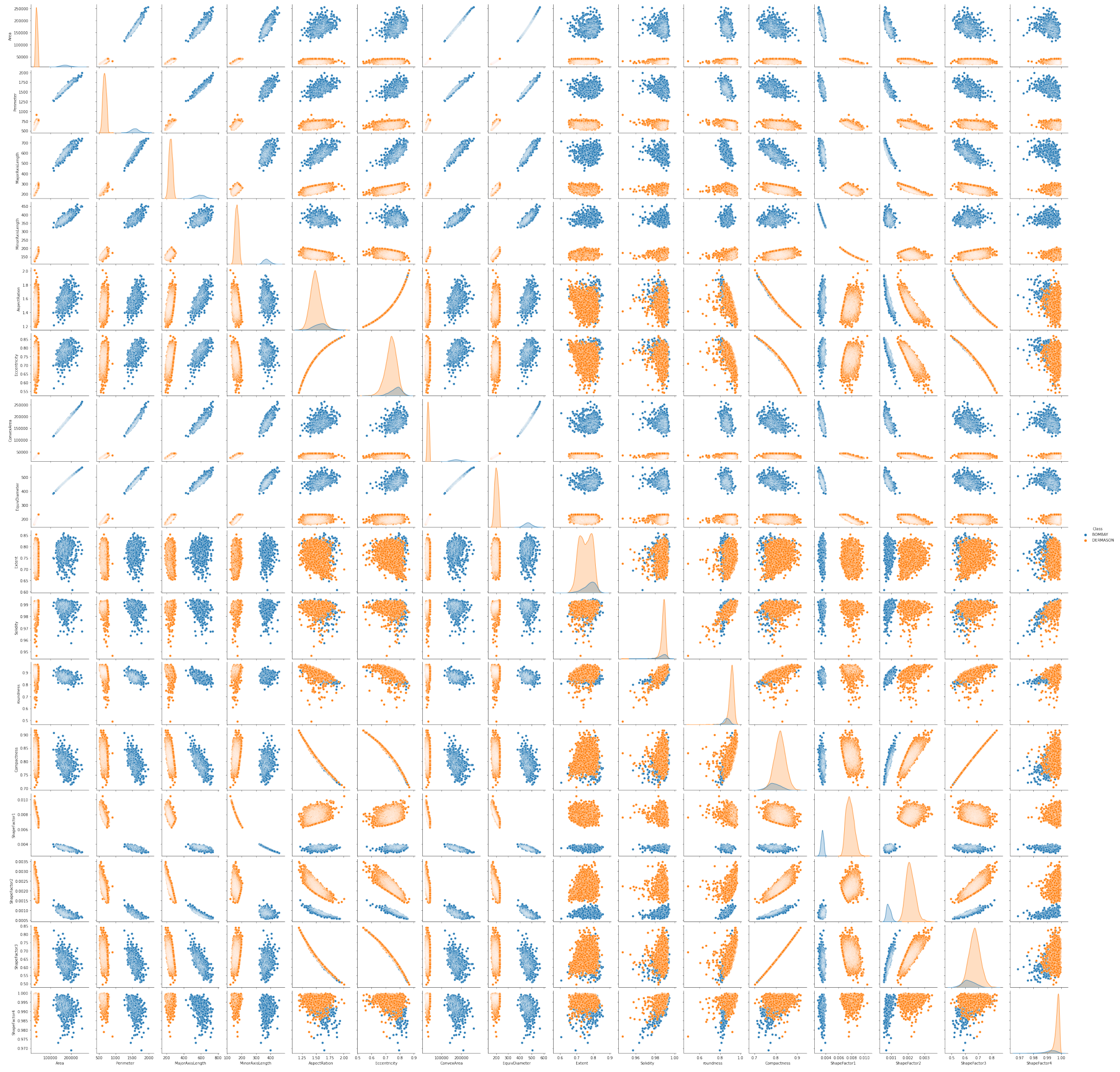


Figure 1: Scatter plot of the relationship between the predictor variables (in pairs) and the distribution of classes within a exploratory variable.

Figure 1 could be used to investigate in values and distribution within the variables as this is showcased in the diagonal axis of the scatter plot. One chose to add a separate histogram to present the result in a distinct way, as all the exploratory variables are continuous this is an accurate graphic representation of their behaviour.

Thus, in the Figure 2, it is possible to see that all the predictors that were well separated are grouped into two clusters in the histogram, namely the Area, Perimeter, MajorAxisLength, MinorAxisLength, ConvexArea, EquivDiameter, ShapeFactor1 and ShapeFactor2. In these resources, one can derive a conclusion from these materials that one might predict.

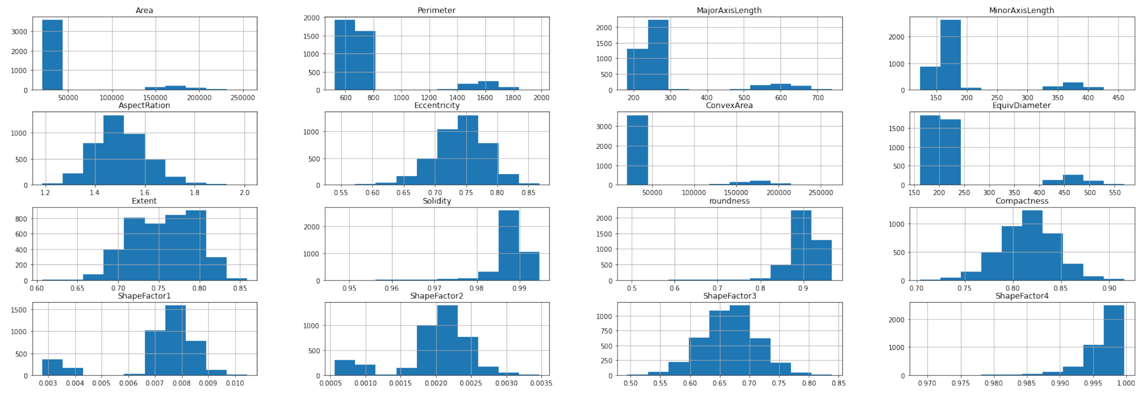


Figure 2: Histogram of variance in the exploratory variables.

The analysis is continued by checking the presence of atypical values using a boxplot representation (Figure 3). At this stage one chose to normalize the data to get a nice representation of the dataset, a deeper discussion on normalizing will be conducted in the next subsection (2.1.).

Analysing Figure 3 it is clear that there are a lot of values outside of the upper and lower whisker. The reason for this is because a binary classification is being done and sometimes there are well separated groups. The boxplot will then show one of the class as outliers for the well separated groups mentioned above. Therefore, one chose not to act on these and the outliers, neither dropping them or replacing the values. Regarding the other variables, there are not many clear outliers either. The points that differ the most from the data are the observations with value zero for Extent, Solidity and Roundness. This will be taken in account when performing the classification.

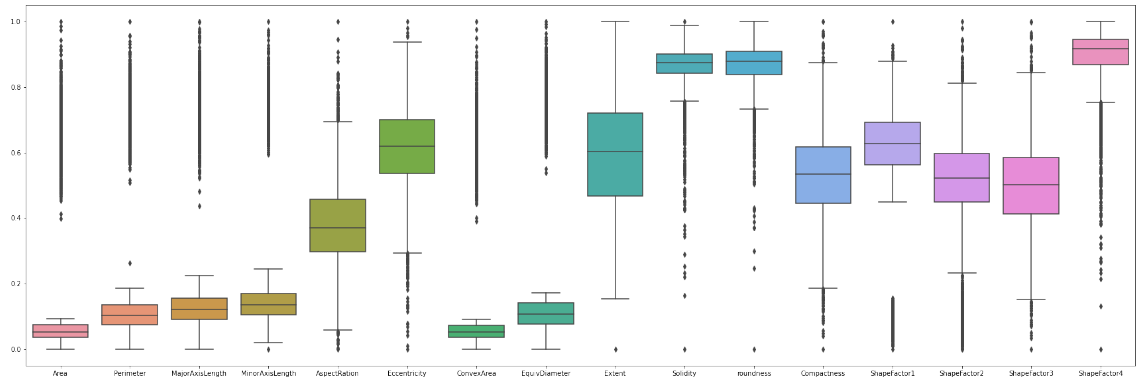


Figure 3: Values are displayed using a boxplot to help understand values in the quantiles and outliers.

To filter the proposed dataset, two different major approaches were used. This branching of options will be the starting point to obtain several possible datasets that will be tested in section 5. Considering different possibilities for the final data is of utmost importance, as there is no "ideal" dataset.

The first approach was done by exploring the correlation between the exploratory variables and response variable. This is done with Sckit Learn library for Random Forest using the build in algorithm for variables importance, represented in Figure 4. The computation is done using Gini impurity measure (or Mean Decrease in Impurity (MDI)). For the decision tree, Gini impurity measure is used for determining how to split the data in smaller groups and is basically a measurement of the probability of a new observation being incorrect classified in the decision tree. The Gini index used for random forests measures the times a feature is used to split a node, weighted by the number of samples it splits.

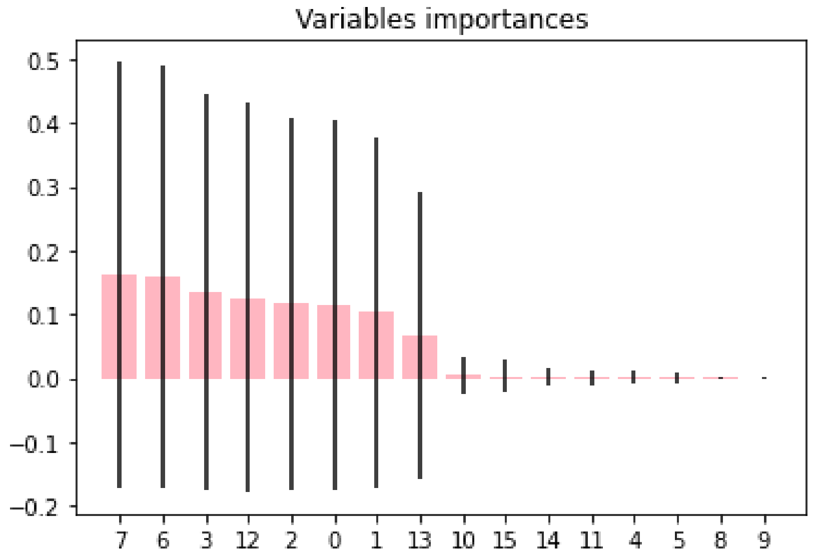
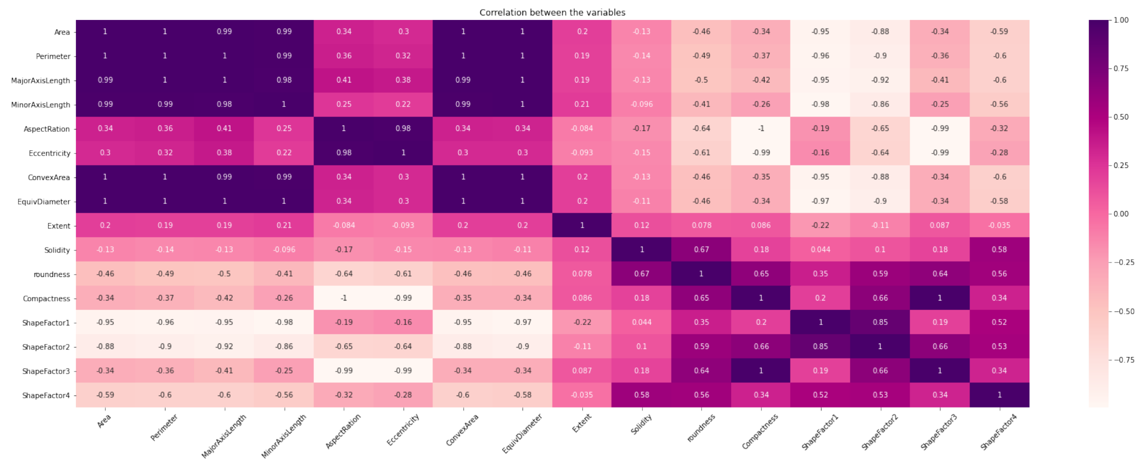


Figure 4: Variables importances.

With this, a new set of predictors with the variables: Area, Perimeter, MajorAxisLength, MinorAxisLength, ConvexArea, EquivDiameter, ShapeFactor1and ShapeFactor2 was formed.

The second approach was done by analysing the correlation matrix available in Figure 5.



	Area	Perimeter	MajorAxisLength	MinorAxisLength	AspectRation	Eccentricity	ConvexArea	EquivDiameter	Extent	Solidity	roundness	Compactness	ShapeFactor1	ShapeFactor2	ShapeFactor3	ShapeFactor4
Area	1	1	0.99	0.99	0.34	0.3	1	1	0.2	-0.13	-0.46	-0.34	-0.95	-0.88	-0.34	-0.59
Perimeter	1	1	1	0.99	0.36	0.32	1	1	0.19	-0.14	-0.49	-0.37	-0.96	-0.9	-0.36	-0.6
MajorAxisLength	0.99	1	1	0.98	0.41	0.38	0.99	1	0.19	-0.13	-0.5	-0.42	-0.95	-0.92	-0.41	-0.6
MinorAxisLength	0.99	0.99	0.98	1	0.25	0.22	0.99	1	0.21	-0.096	-0.41	-0.26	-0.98	-0.86	-0.25	-0.56
AspectRation	0.34	0.36	0.41	0.25	1	0.98	0.34	0.34	-0.084	-0.17	-0.64	-1	-0.19	-0.65	-0.99	-0.32
Eccentricity	0.3	0.32	0.38	0.22	0.98	1	0.3	0.3	-0.093	-0.15	-0.61	-0.99	-0.16	-0.64	-0.99	-0.28
ConvexArea	1	1	0.99	0.99	0.34	0.3	1	1	0.2	-0.13	-0.46	-0.35	-0.95	-0.88	-0.34	-0.6
EquivDiameter	1	1	1	1	0.34	0.3	1	1	0.2	-0.11	-0.46	-0.34	-0.97	-0.9	-0.34	-0.58
Extent	0.2	0.19	0.19	0.21	-0.084	-0.093	0.2	0.2	1	0.12	0.078	0.086	-0.22	-0.11	0.087	-0.035
Solidity	-0.13	-0.14	-0.13	-0.096	-0.17	-0.15	-0.13	-0.11	0.12	1	0.67	0.18	0.044	0.1	0.18	0.58
roundness	-0.46	-0.49	-0.5	-0.41	-0.64	-0.61	-0.46	-0.46	0.078	0.67	1	0.65	0.35	0.59	0.64	0.56
Compactness	-0.34	-0.37	-0.42	-0.26	-1	-0.99	-0.35	-0.34	0.086	0.18	0.65	1	0.2	0.66	1	0.34
ShapeFactor1	-0.95	-0.96	-0.95	-0.98	-0.19	-0.16	-0.95	-0.97	-0.22	0.044	0.35	0.2	1	0.85	0.19	0.52
ShapeFactor2	-0.88	-0.9	-0.92	-0.86	-0.65	-0.64	-0.88	-0.9	-0.11	0.1	0.59	0.66	0.85	1	0.66	0.53
ShapeFactor3	-0.34	-0.36	-0.41	-0.25	-0.99	-0.99	-0.34	-0.34	0.087	0.18	0.64	1	0.19	0.66	1	0.34
ShapeFactor4	-0.59	-0.6	-0.6	-0.56	-0.32	-0.28	-0.6	-0.58	-0.035	0.58	0.56	0.34	0.52	0.53	0.34	1

Figure 5: Correlation matrix between the exploratory variables.

To decide which variables should be removed, the applied reasoning was as follows:

- The correlation matrix was analysed to find out which explanatory variables had a correlation of more than 80%, since highly correlated variables explain the same information, then in principle it is not necessary to have both in the model and there may be interference between them.
- Among correlated variables, the one with the least correlation with the variable Class was eliminated through the correlation ratio, since between numeric-categorical variables the correlation cannot be measured directly. The correlation ratio (η) is as following:
$$\eta^2 = \frac{\sum_x n_x(\overline{y_x} - \overline{y})^2}{\sum_{x,i}(y_{xi} - \overline{y})^2} \tag{1}$$
where n_x is the number of observations in category x, y_{xi} is the observation corresponding to category x and i is the label of the particular observation. $\overline{y}_x$ is the mean of the category x and $\overline{y}$ is the mean of the whole sample. When x only has two different possibilities, the correlation ratio gives the same result as the square of Pearson's correlation coefficient.

This approach leds us to keep the variables: EquivDiameter, Extent, Solidity, Roundness, Compactness and ShapeFactor4.

Another problem with this dataset is the imbalance between observations in both classes of the response variable, as it is visible in the following Figure 6.

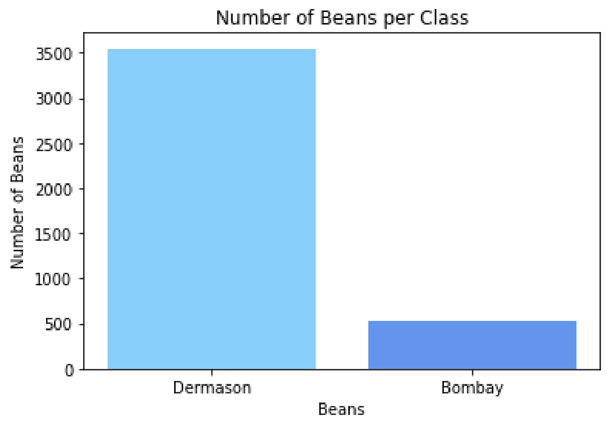


Figure 6: Number of Beans per Class

The dataset consists of 552 observations of BOMBAY and 3546 observations of DERMASON (which corresponds to 87% of the data), thus the dataset is unbalanced in the number of data points for each class to train on for the classification. Due to this reason, one will make use of an oversampling technique (SMOTE) to try to increase the predictive power of dry bean species, since it is the most important goal. This technique will be explained in subsection 5.2.

2.1 Test and Train Data

In order to avoid situations like overfitting phenomena, the dataset was firstly splitted into training and test sets. The training set is composed by 80% of the original data and the test set is composed of the remaining 20%. The splitting of the data was handled carefully, since the same proportion of BOMBAY and DERMASON observations of the response variable in both sets needed to be guaranteed. For this reason, 2839 observations of the DERMASON class appear in the training set, however only 707 of them are tested.

In the explanatory variables, it is necessary to proceed to their normalisation, since they present different scales among themselves. Thus, the command `min_max_scaler` from the pyhton library Scikit Learn was used in order to transform the values of the variables to values in the scale [0,1]. To do this, the scaler was fitted using the training data, and then the training and test data were normalised with that same scaler. This is done because the test data is handled as new, unseen data.

3 Performance Measures and Confusion Matrix

Having in hand a binary classification problem (0 (DERMASON) or 1 (BOMBAY)), here are the measures used to evaluate the quality of the resulting models of the classifiers. All the performance measures used, can take values in the interval [0, 1]. Before defining the performance measures, it is necessary to introduce the following 4 concepts: In order to define the infra measures, it is necessary to define the following 4 values:

- **True Positive (TP)** - number of observations correctly classified into BOMBAY variety;
- **True Negative (TN)** - number of observations correctly classified into DERMASON variety;
- **False Positive (FP)** - number of misclassified observations into BOMBAY variety;
- **False Negative (FN)** - number of misclassified observations into DERMASON variety.

3.1 Confusion Matrix

The confusion matrix describes the performance of the classifier. If $i = j$, the input a_{ij} corresponds to the number of correct classifications of an observation of a class i; if $i! = j$, then it corresponds to the number of times class i (j) is incorrectly identified as j (i). From the confusion matrix, it is possible to determine other performance measures, described below.

3.2 Sensitivity/Recall

Sensitivity is the rate of true positives among all positive ones, which means that is the relative frequency of true positive observations and all the positive observations:

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

3.3 Specificity

Specificity (or true negative rate) reflects the same as sensitivity, but for true negatives: transmits the proportion of false negatives in relation to all observations that are really negative:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

3.4 Accuracy

As the name required, we are interest in knowing the exactness of the model, which means the relative frequency of correctly classified observations in the sample. The expression of the accuracy is:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

3.5 Balanced Accuracy

Due to the possibility that there are unbalances in the classification and this affects the results, the balanced accuracy can be obtained through the average of the two measure mentioned above:

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}$$

3.6 Precision

Precision is the rate of true positives among all the observativons that are classified as positive:

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

3.7 F-Measure

The F-measure, or F1-score, is built on the basis of accuracy and sensitivity, and this can be interpreted as a harmonic mean of these measurements:

$$\text{F-Measure} = 2 \times \frac{\text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Sensitivity}} = \frac{2TP}{2TP + FP + FN}$$

Note: F1-score and Balanced Accuracy, should ideally be close to 1. Both evaluators should be used especially when there exists unbalanced classes (i.e., when there are significant differences between the number of observations of the various classes - unbalanced dataset).

4 Brief description of methods used for classification

4.1 K Nearest Neighbor (KNN)

K-nearest neighbors (KNN) [4] is a type of supervised learning algorithm used for both regression and classification. This method tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. The used distance can be the Euclidean distance (norm 2 - simple case for evaluation of spatial distance) or another distance measure can be considered. For instance, for numerical explanatory variables, the Manhattan distance must be used, which is this problem's case. After that, one should select the k number of points which is closet to the test data. Knowing that a certain item (the observation one wants to categorize) has been observed, the probability of belonging to a certain class is determined for each class as a classification criterion.

How does KNN work? The first step is to select the number k of neighbors. The value of k indicates the number of the nearest neighbors that should be taken into account. There are no pre-defined statistical methods to find the most favorable value of k. One should, first, initialize a random k value and start computing (it is important that k is not too high, as this increases the risk of making the classification boundaries less distinct). On the other hand, there is the risk that the defined neighborhood may include objects from other classes - due to an increase in the bias, also characterized by a low variance, in general. Choosing a small value of k leads to unstable decision boundaries and creates a situation of possible greater dispersion - variance and small bias. Yet, if the value of k is too small it can lead to overfitting issues. This phenomenon occurs in complex models, in which the number of variables is immeasurably higher than the number of observations). The substantial k value is better for classification as it leads to smoothening the decision boundaries.

It is better if k is an odd number, as this avoids statistical ties, which are more probable with even numbers. The value of k will be the hyperparameter to be determined with the k-fold CV.

4.2 Naïve Bayes

Naïve Bayes Classifier [7] is known as a probabilistic algorithm that is typically used for classification problems. In this algorithm, the distribution of samples in each class is modeled using a probabilistic model which assumes that all the variables (given the class) are independent from each other. Based on this assumption there is not a single NB classifier, but a family of them. All models have, as main goal, to find a posteriori probability of a class, based on the distribution of the variables of each sample. Now, before moving to the formula for Naïve Bayes, it is important to know about Bayes' theorem:

Bayes' Theorem: This theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Now, with regards to this problem's dataset, one can apply this theorem in the following way: Given a set of training examples $T = t_1, t_2, ..., t_n$ with length n, the respective labels $L = l_1, l_2, ..., l_n$, the set of variables $X = x_1, x_2, ..., x_m$ with m variables and k possible outcomes $o = o_1, o_2, ..., o_k$ corresponding to all the different label classes, the model assigns probabilities

$$P(o_j|x_1, x_2, ..., x_m)$$

with $j \in 1, ..., k$ for each class o_j.

Now the conditional probability above can be decomposed as:

$$P(o_j|t_i) = \frac{P(t_i|o_j)P(o_j)}{P(t_i)} \tag{2}$$

The numerator of equation 1 is equivalent to the joint probability model $P(o_j, x_1, ..., x_m) = P(o_j)\prod_{i=1}^{m} P(x_i|o_j)$. Now, as the denominator remains constant for a given input, since $P(t_i) = \sum_k P(o_j)P(t_i|o_j)$, one can remove that term and obtain $P(o_j|x_1, ..., x_m) \propto P(o_j)\prod_{i=1}^{m} P(x_i|o_j)$

The Naive Bayes classifier combines Naive Bayes model with the maximum a posteriori decision rule, which instructs the model to choose the most probable hypothesis in the end. Now, a classifier model needs to be created. For this, one finds the probability of a given set of inputs for all possible values of the possible outcomes and pick up the output with maximum probability. This can be expressed mathematically as:

$$\hat{y} = argmax_j P(o_j)\prod_{i=1}^{m} P(x_i|o_j) \tag{3}$$

In this model there is a hyperparameter called alpha that is going to be chosen (by k-fold CV) to control the smoothing.

4.3 Binary logistic regression

Logistic Regression is based on multiple linear regression. It is a powerful tool when dealing with discrete classification problems as it allows independent variables to be either categorical or continuous. The method does not require a great deal of exposure as it was taught in class, but it is important to mention the most important aspects. Note that, in this case, one is interested in only two classes (binary case) The Logistic Regression translates into a linear model that best fits this problem's independent variables and the use of the sigmoid function allows the obtained values to belong to the range from 0 to 1. When these values are entered as probabilities (P), the result obtained is the probability of the dependent variable is equal to one of the possible cases. In this project's case where the possible classes are 0 or 1, the following decision rule was considered:

$$Class = \begin{cases} 1 \text{ , if } P > 0.5 \\ 0 \text{ , otherwise} \end{cases} \tag{4}$$

4.4 Decision Tree and Random forests

Decision Trees [5] are a non-parametric supervised learning method used for classification. They learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model. Also, this method can handle both categorical and numerical data. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

Construction of decision trees A decision tree can be “learned” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. It is important to refer that decision trees can handle high dimensional data.

Decision trees Representation Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the subtree rooted at the new node.

However, the classifier that will be used are the random forests that consists of multiple decision trees just as a forest has many trees. On top of that, it uses randomness to enhance its accuracy and combat overfitting, which can be a huge issue for such a sophisticated algorithm. These algorithms make decision trees based on a random selection of data samples and get predictions from every tree. After that, they select the best viable solution through votes. The basis of the construction of these forests is the Bagging method (an ensemble method that can be used to improve the perfomance of classifiers):

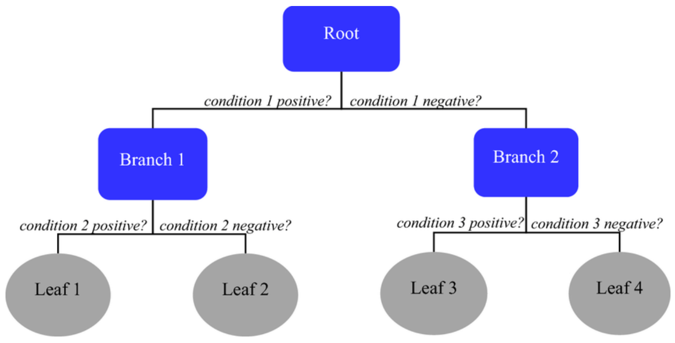


Figure 7: Decision Tree structure

Recall that given n independent observations $Z_1, ..., Z_n$ each with variance σ^2, the variance of the mean $\overline{Z}$ of the observations is given by $\frac{\sigma^2}{n^2}$. The algorithm that produces this classifier is the following:

At the beginning, there is a training set $T = (x^0, y^0), (x^1, y^1), ..., (x^n, y^n)$, where the outcomes $y^{(i)} \in \{0, 1, ..., k-1\}$.

- Generate B training sets $T^{(i)}$ from T by bootstrap (sampling with a replacement);
- We then average the predictions: Train a decision tree from each $T^{(i)}$ with the following idea in mind: randomly select a subset of variables (features) at each node and only those variables are candidates for splitting features (the name given to this process is random subspace). Also, note that the percentage of features candidates at each node is a paremeter of the algorithm;
- These trees are grown deep and are not pruned.
- Each tree has a high variance with low bias. Averaging the $T^{(i)}$ trees brings down the variance.
- After this, compute the a posteriori distributions: $[P^{(i)}(y = 0|x), ..., P^{(i)}(y = k-1|x)]$, for all trees obtained.
- Finally, aggregate all a posterior distributions:

$$\hat{P}(y = c|x) := \frac{1}{B}\sum_{i=1}^{B} P^{(i)}(y = c|x)$$

One of the greatest advantages of this classifier is that it reduces overfitting problem in decision trees and also reduces the variance as well as errors due to bias and therefore improves the accuracy. Also, two hyperparameters will be considered when using this classifier: the number of estimators (the number of trees in the forest) and the maximum possible depth of each tree in the forest.

4.5 Support Vector Machine

A support vector machine (SVM) [3] is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text. They have two main advantages: higher speed and better performance with a limited number of samples (in the thousands).

How does SVM works?

SVM are a form of linear classifiers which compare each input vector with a hyperplane decision boundary:

$$x.w + b > 0 \Rightarrow \hat{y} = +1 \tag{5}$$

$$x.w + b < 0 \Rightarrow \hat{y} = -1 \tag{6}$$

Therefore,

$$\hat{y} = sign(x.w + b) \tag{7}$$

This algorithm aims to find a hyperplane in an N-dimensional space that separates the data test, or at least maximizes the distinction of each class. Since that can be defined infinites hyperplanes, the idea is to find the plane that has the maximum margin, i.e. the maximum distance between observations of different classes. Although hyperplanes are the simplest geometric shape one can imagine, sometimes the data is not linearly separated, which compels us to use other functions in the kernel, namely the radial base kernel. In this case, a hyperplane is determined in such a way that the least number of vectors possible is located in the wrong side (misclassified).

The regularization parameter, also known as C, in the SVM classifier determines how many data samples are allowed to be placed in different classes. If the value of C is set to a low value, the probability of the outliers is increased, and the general decision boundary is found. If the value of C is set high, the decision boundary is found more carefully. If the points are linearly separable then only our hyperplane is able to distinguish between them and if any outlier is introduced, the hyperplane is not able to separate the classes. In order to fix this, one could decrease C value which causes the classifier to leave linear separability and gain more stability. This allows the classifier to skip few outliers. In this case, we are in presence of soft margin.

There is no rule of thumb to choose a C value, it totally depends on our training data. The only option is trying bunch of different values and choose the value which gives you lowest misclassification rate on testing data and for that one could use `GridsearchCV`, in which one can directly give a list of different values parameter and it will tell us which value is best. One of the advantages of SVM is that non linear SVM can be tested and trained using low dimension data, by replacing the inner products of the feature vectors by the kernel. This is called the kernel trick and this is what makes SVM suitable for datasets with high dimensions (a large number of features).

4.6 XGBoost

XGBoost [2] classifier is a Machine learning algorithm that is applied for structured and tabular data and is an implementation of gradient boosted decision trees designed for speed and performance. It's attractivness is due to a very simple base model, known to most people, decision trees. Given its recent performance in machine learning competitions, it is fair to say that this is one of the best, most adaptable algorithms one can use, given reasonable optimization of its hyperparameters. XGBoost is an extreme gradient boost algorithm and works with large complicated datasets. Boosting refers to a class of learning algorithm that fit models by combining a number of simpler models, referred to as base models. These simpler models tend to be quite limited in their predicted ability, but when selected through a boosting algorithm, they form a more accurate model. This model is often referred to as an ensemble models.

Furthermore, Gradient boosting is a powerful machine learning technique, capable of fitting nonparametric predictive models and was motivated as being a gradient descent method in a function space that is the reason it is called a "Gradient Boosting Algorithm".

XGBoost operates on decision trees, models that construct a graph that examines the input under various "if" statements (vertices in the graph). Whether the "if" condition is satisfied influences the next "if" condition and eventual prediction. XGBoost progressively adds more and more "if" conditions to the decision tree to build a stronger model.

There are 6 key XGBoost optimisations that make it unique:

- Approximate Greedy Algorithm By default, XGBoost uses a greedy algorithm for split finding which makes the split finding process very quick. However, with relatively large training datasets, the greedy algorithm becomes very slow. To overcome this, the splits in the trees are approximated. These approximations are based on quantiles. That is, the first quantile is the first threshold, the second quantile is the second threshold, and so on and so forth. By default, approximate greedy algorithm builds approximately 33 quantiles;

- Parallel learning and Weighted Quantile Sketch The quantiles are built using parallel learning and weighted quantile sketch. That is, the dataset is split into multiple small subsets and distributed across multiple cores. Each subset calculates the quantiles in parallel and the statistic is pulled together to form an approximate histogram of quantiles. In XGBoost, the quantiles are weighted, such that, the sum of the weights within each quantile are approximately the same. For regression, the weights associated with each quantile is 1. However, for classification, the weights are calculated based on the probabilities.

- Sparsity-Aware Split finding Sparsity aware split finding helps to handle missing information in data and provides a basis on how to deal with new missing data. In this optimisation, the data is split into two groups. One group has data with no missing feature values, and the second group has data with all the missing features rows with its associated response variable. The data from group one is sorted in an ascending fashion. Then, the split finding process calculates two sets of gain values: First, it would calculate gain by adding the missing data from the second group to the left of the tree and secondly it would calculate gain by adding the missing data from the second group to the right of the tree. This is done for each of the quantiles. The largest gain overall is picked as the default, when there is missing data.

- Cache-Aware Access Cache memory in the CPU is the fastest to access. Hence, this is used to store the first and second order derivatives within XGBoost (gradients and hessians) to rapidly calculate the scores for each node and leaf in the tree.

- Compressed Sparse Column (CSC) data format XGBoost divides the dataset into multiple blocks in a Compressed Sparse Column format (CSC) to distribute the blocks to multiple cores for parallel learning.

XGBoost is a faster algorithm when compared to other algorithms because of its parallel and distributed computing. XGBoost is developed with both deep considerations in terms of systems optimization and principles in machine learning. The goal of this algorithm is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate method. Although this method is computationally faster, the other models are faster since in our case the data is simple.

5 Applying the classifiers to the data

Before proceeding, it is necessary to explain the datasets that were chosen to be analysed. Firstly, the classifiers were applied to the original datatrain, that is, with all the variables. As the Naïve Bayes method was the only one that failed to correctly classify the types of beans, and due to the classes being extremely unbalanced as it is visible in the Figure 6 the SMOTE method was applied to the data, in order to verify if, by increasing the observations of the underrepresented class, the performance measures of this classifier would present better results.

Finally, this dataset was exploited in the [6] study. In this scenario, the response variable had seven outcomes (bean species) in the original problem. The best-performing model, according to the article reporting the results, contains the following variables: Perimeter, MajorAxisLength, MinorAxisLength, Compactness and ShapeFactor1. To understand whether in fact this choice of predictors produces good results in this case, where the response variable is restricted to two levels, the study under analysis was replicated.

Thus, the datasets used were:

Datasets

5.1. Original dataset - All Variables
5.2. Original dataset (with SMOTE)
5.3. 8 variables - Area, Perimeter, MajorAxisLength, MinorAxisLength, ConvexArea, EquivDiameter, ShapeFactor1 and ShapeFactor 2
5.4. 6 variables - EquivDiameter, Extent, Solidity, Roundness, Compactness and ShapeFactor4
5.5. 5 variables - Perimeter, MajorAxisLength, MinorAxisLength, Compactness and ShapeFactor1

5.1 With the original data

5.1.1 K Nearest Neighbor (KNN)

In the case of the KNN classifier, one first did a selection of the hyperparameters: k values from 1 to 49 were tested, with odd k, along with the Euclidian and Manhattan metrics. Thereafter, one chose the best combination of these hyperparameters, using the averages of the `accuracy`, `f1_micro`,`recall`,`precision` and `roc_auc` and k-fold CV used was 10. One chose $k = 1$ and `metric: Manhattan`.

The value $k = 1$ was chosen sin, according to the theory, small and large k cause overfitting issues. Since in this case the scores for every k are all the same, one chose a smaller k which makes the model simpler. Given that the metric that leads to the best performing classification is the Manhattan distance one again sees the different values of k for this distance and one used accuracy as a score because all the scores had 100%, so it was indifferent to use another score. Again, the best k value was 1. The plot is shown below:

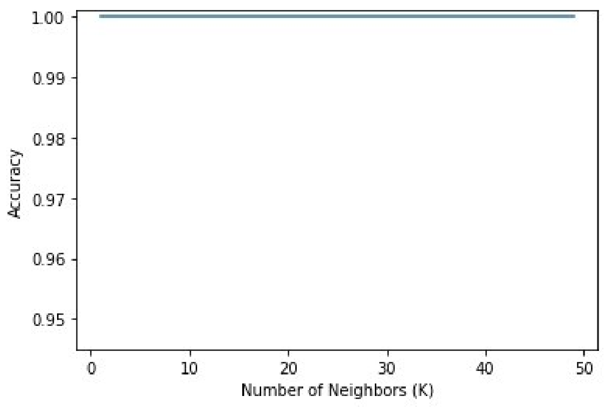


Figure 8: KNN Classifier scores for different k values

After all of this, one applied KNN with the Manhattan distance and $k = 1$. With this values one obtained the following results:

Performance Measures for KNN					
Label	Precision	Recall	Specificity	F1-Score	Support
BOMBAY (0)	1.00	1.00	1.00	1.00	107
DERMASON (1)	1.00	1.00	1.00	1.00	707

Macro average	1.00	1.00	1.00	1.00	
Weighted average	1.00	1.00		1.00	

Accuracy	1.00
Balanced Accuracy	1.00
Running Time	0.0025073 s

Table 1: Performance Measures for KNN with $k = 1$ and `metric:  Manhattan`.

$$\begin{bmatrix} 107 & 0 \\ 0 & 707 \end{bmatrix}$$

Figure 9: Confusion Matrix for KNN

In Table 1 , one can see that, for both of the classes (DERMASON and BOMBAY) the classifier application resulted in a precision of 100% such as the recall and the F1-score, which means that the rate of true positives among all positive ones and the harmonic mean between precision and sensitivity is always 1.0 and that leads to conclude that all observations were correctly classified.

Besides that one can also see that the accuracy is 1.0 such as the macro average and the weighted average. Finally, by observing the confusion matrix one can see that there is no class that has not been misclassified, so the KNN algorithm has predicted 100% of the test set observations (as it can be seen in Figure 9), due to the excellent results obtained.

5.1.2 Naïve Bayes

In the case of the Naive Bayes classifier, the parameteres that were studied in order to find the optimal model were the alpha value (smoothing parameter to avoid the zero probability problem) and as one did in the KNN, the following hyperparameters were tuned: α values from 1 to 20 were tested, using the averages of the `accuracy`, `f1_micro`, `recall`, `precision` and `roc_auc` and k-fold CV used was 10. The hyperparameter selection decision is supported by several performance measures. However, most of these (exactly one) remained constant in this process. Thus, the decision is supported by the measure `roc_aux` and it determined that $\alpha = 6$. Below, it can be seen in the plot that for $\alpha = 6$ the highest `roc_auc` value is obtained:

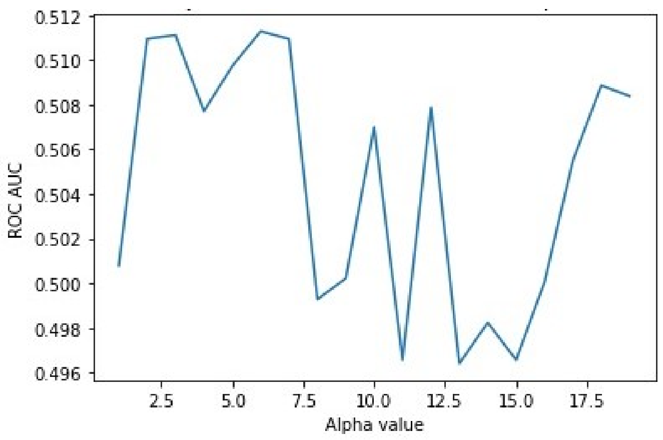


Figure 10: Naïve Bayes Classifier scores for different alpha values

After all of this, the Naïve Bayes method with $\alpha = 6$ was applied. With this values following results were obtained:

Performance Measures for Naïve Bayes					
Label	Precision	Recall	Specificity	F1-Score	Support
BOMBAY (0)	0.0	0.0	0.997	0.0	107
DERMASON (1)	0.87	1.00	0.0	0.93	707

Macro average	0.43	0.50	0.50	0.46	
Weighted average	0.75	0.87		0.81	

Accuracy	0.87
Balanced Accuracy	0.4986
Running Time	0.0070860 s

Table 2: Performance Measures for Naïve Bayes with $\alpha = 6$.

$$\begin{bmatrix} 0 & 107 \\ 2 & 705 \end{bmatrix}$$

Figure 11: Confusion Matrix for Naïve Bayes

By observing the table 2 above, starting with accuracy 0.87, approximately 0.9, one could think that the model is actually very good, however, looking at the balanced accuracy (0.4986) the performance is not so good. Also, even though both accuracy and balanced accuracy are used as classification metrics, as the data set (original data) is unbalanced it can be predicted a significant difference between these two (balanced accuracy with worse results and yet more realistic). Moreover, observing the other classification metrics (that are evaluated per label), it is straightforward that the main reason for this outcome is the fact that one of the classes (BOMBAY) is not classified at all, as it can be seen by the values of Precision, Recall and F1-score (all equal to zero). This could be related to the fact that this class is poorly represented in the data set (only 107 observations over 814).

Looking at Figure 11, it is also interesting to notice that the class DERMASON is almost 100% correctly classified as it can be seen by the values of Precision (0.87), Recall (1.0) and F1-score (0.93). Although the values for the weighted average are reasonable (0.75 for the precision, 0.87 for recall and 0.81 for F1-Score) the value of the macro average for all the scores is not that good (0.43 for the precision, 0.50 for recall and 0.46 for F1-Score). Looking at the Confusion Matrix all of the 107 BOMBAY observations were classified as DERMASON, so the observations from BOMBAY class have been misclassified. This way, the Naïve Bayes algorithm has a bad prediction of the test set observations. There is an overfitting phenomenom with this method, which occurs when a model is overly suited to the data it was trained with and so does not generalize well to other data sets. That is, it performs poorly when used in different sets. Concluding, as this model has a particularity of making assumptions, it is no longer able to bring credible results. In this case, if the strong assumption of the Naïve Bayes is not satisfied, the results of this are neither good nor respectful. This model is simple and computationally cheap, so it is more accurate for more optimized datasets, to behave better, which is not this problem's case. This classifier is not optimal and assumes a demanding premise (the independence of each conditioned variable given the class) that is not verified in many cases [8]. The previously analyzed model, KNN, does not make assumptions in the data so the results obtained are more reliable.

5.1.3 Binary logistic regression

In the case of the Logistic Regression classifier, first one did a tuning of the hyperparameters: the regularization parameter C got values tested from -4 to 4 and the `class_weight` parameter if it's balanced or not. The different combinations of hyperparameters were scored, using the averages of the `accuracy`, `f1_micro`, `recall`, `precision` and `roc_auc` and k-fold CV used was 7. After that, one chose the `class_weight` parameter to be balanced because this parameter sets weights to each class (balanced mode uses the values of the response variable to automatically adjust weights inversely proportional to class frequencies in the input data), since the dataset is unbalanced and also by computing, one obtained always 100% when the `class_weight` was balanced.

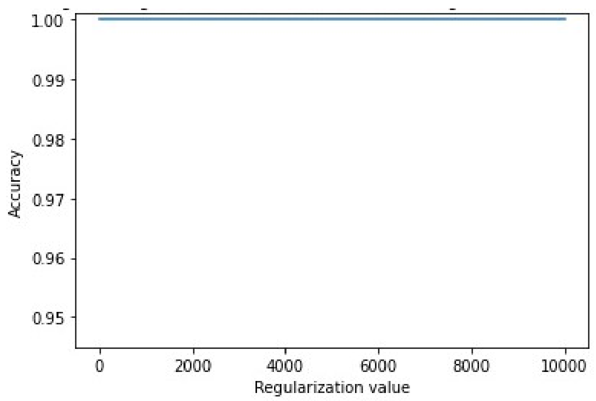


Figure 12: Logistic regression Classifier for different regularization values

After obtaining that the best `class_weight` was balanced, one tried to find the best C value for a balanced `class_weight` parameter and the best one found was when $C = 0.0001$, which defines how much influence a single training example has. As it can be seen in the plot above, all the values were good because they have an accuracy of 1.0. However, if the C parameter is large, one is in presence of hard margin which is quite sensitive to outliers and the hyperplane is not able to separate the classes so it was decided to choose a small value of C which causes the classifier to leave linear separability and gain more stability. This allows the classifier to skip few outliers.

Performance Measures for Binary Logistic regression					
Label	Precision	Recall	Specificity	F1-Score	Support
BOMBAY (0)	1.00	1.00	1.00	1.00	107
DERMASON (1)	1.00	1.00	1.00	1.00	707

Macro average	1.00	1.00	1.00	1.00	
Weighted average	1.00	1.00		1.00	

Accuracy	1.00
Balanced Accuracy	1.00
Running Time	0.0309327 s

Table 3: Performance Measures for Binary Logistic regression with $C = 0.0001$.

$$\begin{bmatrix} 107 & 0 \\ 0 & 707 \end{bmatrix}$$

Figure 13: Confusion Matrix for Binary Logistic regression

In the table 3 , one can see that, for both of the classes (DERMASON and BOMBAY) the precision was 100% such as the recall and the F1-score, which means that the rate of true positives among all positive ones and the harmonic mean between precision and sensitivity is always 1.0 and that allows us to conclude that all observations were correctly classified.

Besides that one can also see that the relative frequency of correctly classified observations in the sample (accuracy) is 1.0 such as the macro average (averaging the unweighted mean per label) and the weighted average (averaging the support-weighted mean per label). Finally, by observing the Figure 13 one can see that there is no class that has not been misclassified, so the Binary Logistic regression algorithm has predicted 100% of the test set observations, because of the excellent results obtained.

5.1.4 Random Forest

With regard to the Random Forest classifier, it was analysed which would be the optimal trio based on:

- Number of Estimators - Number of trees being considered;
- Maximum Depth - The longest path between the root node and the leaf node of a tree. As the `max_depth` value increases, the performance over the test set, after a certain point, decreases, since the tree starts to overfit the training set and is therefore unable to generalise about the invisible points in the test set;
- Criterion - Used to find the optimum split.

For this, it was developed a grid where the number of estimators could vary between (1, 2, 4, 10, 30, 100, 300), the maximum depth could take values from 1 to 10 and the criterion was allowed to be gini impurity or entropy. Thus, the parameters that obtained the best score were `max_depth = 1.0`, `n_estimators = 1` and `criterion: gini`.

In the following table, the performance measures referring to the Random Forest method are presented.

Performance Measures for Random Forest					
Label	Precision	Recall	Specificity	F1-Score	Support
BOMBAY (0)	1.00	1.00	1.00	1.00	107
DERMASON (1)	1.00	1.00	1.00	1.00	707

Macro average	1.00	1.00	1.00	1.00	
Weighted average	1.00	1.00		1.00	

Accuracy	1.00
Balanced Accuracy	1.00
Running Time	0.012516 s

Table 4: Performance Measures for Random Forest with `max_depth = 1.0`, `n_estimators = 1` and `criterion: gini`.

$$\begin{bmatrix} 107 & 0 \\ 0 & 707 \end{bmatrix}$$

Figure 14: Confusion Matrix for Random Forest

One can notice that the accuracy, as well as balanced accuracy has remained at 100% with this algorithm, which means that it has no sensitivity to unbalanced classes.

Moreover, for both BOMBAY and DERMASON class the recall and specificity measures take the value 1.0, which concludes that all observations were well classified.

In turn, in the Figure 14 we can verify that there is no class that has not been misclassified, so the Random Forest Classifier has predicted 100% the observations of the test set.

Finally, precision and the F1-Score also have a value of 1.0. The former indicates that all observations were true positive, and the latter emerges as a consequence since it aims at balancing both the concerns of precision and recall in one number. Regarding the running time, it is expected to be low since the optimal number of trees is 1.

5.1.5 Support Vector Machine

To apply the Support Vector Machine classifier, it was applied the function *GridSearchCV* to find the optimal hyper-parameters of the model. In this line, the following were studied:

- C - The regularization parameter. This parameter tells the SVM optimization how much it wants to avoid misclassifying each training example. Here, it was tested C $\in$ `np.logspace(-4, 4, 20)`;
- Type of Kernel - The function of kernel is to take data as input and transform it into the required form. Four different types were tested, namely linear, poly, rbf and sigmoid;
- Gamma - The Gamma value decides what curvature the decision boundary will have. The Gamma values tuned were: (0.0001, 0.001, 0.01, 0.1, 1, 10).

The results that took us into the best score were $C = 0.0001$, `Gamma: 1` and `Kernel: polinomial`.

Again, as one can see in the Table 5 bellow, the performance measures obtained the same results of the past methods, as exception the Naive Bayes. Accordingly, recall and specificity remain equal to 1, meaning that a true positive and negative was obtained in all classifications. The accuracy and balanced accuracy also remain 1.0, as it can be illustrated in the table below.

Performance Measures for SVM					
Label	Precision	Recall	Specificity	F1-Score	Support
BOMBAY (0)	1.00	1.00	1.00	1.00	107
DERMASON (1)	1.00	1.00	1.00	1.00	707

Macro average	1.00	1.00	1.00	1.00	
Weighted average	1.00	1.00		1.00	

Accuracy	1.00
Balanced Accuracy	1.00
Running Time	0.1312984 s

Table 5: Performance Measures for SVM with $C = 0.0001$, `Gamma: 1` and `Kernel`.

$$\begin{bmatrix} 107 & 0 \\ 0 & 707 \end{bmatrix}$$

Figure 15: Confusion Matrix for SVM

These results are supported by the Figure 15, where the parcels outside the main diagonal are equal to 0, and BOMBAY (107) and DERMASON (707) classes were all correctly classified, which proves that there was no misclassification.

It is also notable that this method presents a running time 10 times superior (0.1313 s) to that resulting from the Random Forest algorithm.

5.1.6 XGBoost

The XGBoost algorithm provides a wrapper class to allow models to be treated like classifiers or regressors. The XGBoost model for classification is called XGBClassifier. As this case study faced a binary problem, it was decided to use the binary:logistic objective function.

The Booster Parameters tuned were:

- min_child_weight - Defines the minimum sum of weights of all observations required in a child. It is also used to control over-fitting, and too high values can lead to under-fitting.
- `Gamma` - Gamma specifies the minimum loss reduction required to make a split.
- `SubSample` - Denotes the fraction of observations to be randomly samples for each tree. Low values prevents overfitting but too small values might lead to under-fitting.
- `colsample_bytree` - Denotes the fraction of columns to be randomly sampled for each tree.
- `max_depth` - The maximum depth of a tree. It's used to control over-fitting, as higher depth will allow model to learn relations very specific to a particular sample.

After tunning the hyperparameters, the optimal results are: `colsample_bytree = 0.6`,`Gamma = 0.5`,`max_depth = 3`,`min_child_weight = 1`, `subsample: 0.6`.
Follows the algorithm performance measures, when applied joint the tests.

Performance Measures for XGBoost					
Label	Precision	Recall	Specificity	F1-Score	Support
BOMBAY (0)	1.00	1.00	1.00	1.00	107
DERMASON (1)	1.00	1.00	1.00	1.00	707

Macro average	1.00	1.00	1.00	1.00	
Weighted average	1.00	1.00		1.00	

Accuracy	1.00
Balanced Accuracy	1.00
Running Time	4.316148 s

Table 6: Performance Measures for XGBoost with `colsample_bytree = 0.6`, `Gamma = 0.5`, `max_depth = 3`,`min_child_weight = 1`, `subsample = 0.6`.

$$\begin{bmatrix} 107 & 0 \\ 0 & 707 \end{bmatrix}$$

Figure 16: Confusion Matrix XGBoost

By observing the Table 6, on a similar way to the past methods, the resultant accuracy was 100% as the precision, recall and F1-Score, which is expected, given the capabilities of the model and the modest complexity of the problem.
Moreover, there were not misclassifications (Figure 16) and all the observations were well classified, on their respective class.
Lastly, the algorithm shown to have a superior running time (4.316148 s) than the other methods result.

5.1.7 General Analysis

In the Table 7 all the results are summarized, comparing all the classifiers in terms of each performance measure. In the table, shaded, are the best results.

	KNN	Naïve Bayes	Random Forest	SVM	LR
Balanced Accuracy	1.0	0.4986	1.0	1.0	1.0
Precision	1.0	0.75	1.0	1.0	1.0
Recall	1.0	0.87	1.0	1.0	1.0
F1-score	1.0	0.81	1.0	1.0	1.0

Table 7: Weighted average for each performance measure and for each classifier

There are some inferences that can be made (some of them already pointed out when interpreting the results of each classifier):

- All the classifiers, except Naïve Bayes have the same results;

- Naïve Bayes is the classifier with the worst results;
- All classifiers have 100% Balanced Accuracy, Precision, Recall and F1-score results, except the Naïve Bayes model;

Therefore, looking at these inferences we could think that there is a tie between all the classifiers (except the Naïve Bayes). However, it should be reminded that the main goal is to correctly classify all the classes and thus the best classifier is the one that correctly (100%) classifies the highest number of labels, in our case, all of them, except Naïve Bayes, have the same and the best results. That being said, the best classifiers with the original data can be the KNN, Binary Logistic regression, Random Forest, SVM or XGBoost since all of them have the same results. However we can choose, from all of these classifiers, the one that has less time computing which is KNN with 0.0025s as time running.

5.2 With the SMOTE technique

Looking at the distribution over classes, it is heavily unbalanced and this can skew the model to predicting the majority class, which in this case is DERMASON. From the analysis of the previous subsection, the Naive Bayes couldn't classify a bean as a BOMBAY bean. In an attempt to resolve this situation, SMOTE (Synthetic Minority Oversampling Technique)[1] was used.

When dealing with unbalanced datasets, the resulting model may try to fit the majority class and consequently provide a false prediction. With SMOTE, either the minority class can be increased or the majority one decreased. In order to better understand SMOTE and how it creates new synthetic observations of new datapoints, it's better to visualize it:

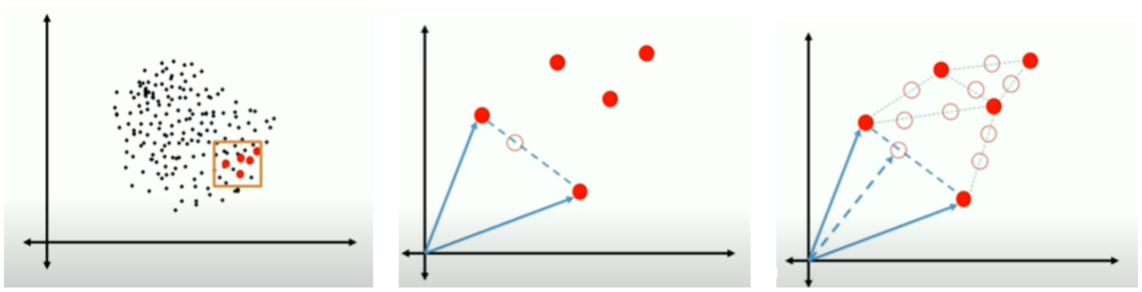


Figure 17: SMOTE technique explained

As it is illustrated in Figure 17, at the first step, all the observations are plotted in 2 dimensions. To create more observations of the minority class (in red), the distance between two observations (dashed/imaginary line) is calculated, and multiplied by a random number between 0 and 1. Then, a new datapoint (in light red) will be created along this imaginary line, and the feature vector for this new point is our synthetic datapoint. We keep doing this for all the points and can keep on adding synthetic observations on these lines depending upon how many datapoints our SMOTE wants to create.

So, one recommendation would be to balance out the classes, typically by adding more instances of the minority classes to be equal to the majority class.

Therefore, 2 different approaches were made:

- **Approach No. 1**: 2839 BOMBAY and DERMASON observations (balanced).
- **Approach No. 2**: 2000 (it was also performed 1000 and 15000) BOMBAY observations and 2839 DERMASON observations.

However, the results obtained by both approaches were equal to those obtained using the original data set. Therefore, the BOMBAY bean was still not properly classified by the Naive Bayes classifier. Since the results do not add any relevant information, the performance measures and confusion matrix have been omitted from the report. However, they can be found in the .ipynb file.

5.3 With the filtered variables - 8 variables

As mentioned in the initial exploratory analysis, it was decided to analyse the 8 variables that have the greatest ability to separate classes. Thus, the classifiers were trained on the new dataset, consisting only of the variables: Area, Perimeter, MajorAxisLength, MinorAxisLength, ConvexArea, EquivDiameter, ShapeFactor1 and ShapeFactor2.

In this case, the same performance measures and confusion matrices for all the models were obtained with the exception of the Naïve Bayes.
Moreover, the optimal parameters for the SVM also suffered alteration, more specifically, the gamma hyperparameter, which took the value 10. This is due to, as the points that are closer to the decision boundary have a greater weight, the decision boundary presents greater curvature, and consequently shapes better to our dataset.

Finally, the only thing that changed with the models that kept the same results is the running time, which is expect because variables were removed. The running time is the following:

Running times (All Variables vs 8 Variables)		
Classifier	**All Variables**	**8 Variables**
KNN	0.0036912 s	0.019857 s
Naïve Bayes	0.007087 s	0.006262 s
Logistic Regression	0.030933 s	0.027047 s
Random Forest	0.012516 s	0.011768 s
SVM	0.131298 s	0.007432 s
XGBoost	4.316148 s	3.281450 s

5.3.1 Naïve Bayes

Regarding Naive Bayes classifier, with the original data, one selected the following hyperparameters: tested for α values from 1 to 20 what was the best combination of hyperparameters, using the averages of the `accuracy`, `f1_micro`, `recall`, `precision` and `roc_auc` and k-fold CV used was 10. As explained before the decision is supported by the measure `roc_aux` and it determined that $\alpha = 14$. Down below, one can see in the plot that for $\alpha = 14$ the highest `roc_auc` value is obtained:

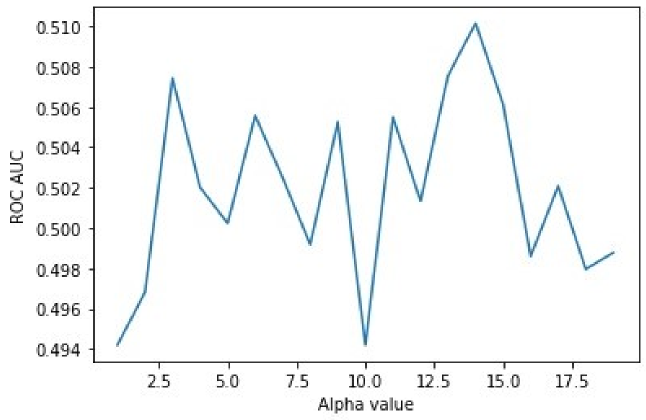


Figure 18: Naïve Bayes Classifier scores for different alpha values

After all of this, applying the Naïve Bayes with $\alpha = 14$, the results were the following:

Performance Measures for Naïve Bayes with the filtered variables					
Label	Precision	Recall	Specificity	F1-Score	Support
BOMBAY (0)	0.0	0.0	1.0	0.0	107
DERMASON (1)	0.87	1.00	0.0	0.93	707

Macro average	0.43	0.50	0.50	0.46	
Weighted average	0.75	0.87		0.81	

Accuracy	0.87
Balanced Accuracy	0.5
Running Time	0.0062626 s

Table 8: Performance Measures for Naïve Bayes with $\alpha = 14$.

$$\begin{bmatrix} 0 & 107 \\ 0 & 707 \end{bmatrix}$$

Figure 19: Confusion Matrix for Naïve Bayes

Looking at the table 8, starting with accuracy 0.87, one could think that the model is actually very good, however, looking at the balanced accuracy (0.5000) the performance is not so good, similarly to what happens with the original data. Moreover, observing the other classification metrics (that are evaluated per label), it is straightforward that the main reason for this outcome is the fact that one of the classes (BOMBAY) is not classified at all, as it can be seen by the values of Precision, Recall and F1-score (all equal to zero). This could be related to the fact that this class is poorly represented in the data set (only 107 observations over 814). It also happens that the class DERMASON is all well classified in this case, which did not happen with the original date, as two of the observations were missclassified, so the class DERMASON is 100% correctly classified as it can be seen by the values of Precision (0.87), Recall (1.0) and F1-score (0.93).

Although the values for the weighted average are reasonable (0.75 for the precision, 0.87 for recall and 0.81 for F1-Score) the value of the macro average for all the scores are not that good (0.43 for the precision, 0.50 for recall and 0.46 for F1-Score). Looking at the Figure 19 all of the 107 BOMBAY observations were classified as DERMASON, so we have a class that has been misclassified, therefore the Naïve Bayes algorithm has poor prediction power of the test set observations. Naïve Bayes is redundant because it classifies everything in the same class.

As it happened with the original data, one obtained almost the same results. The only difference is the fact that the class DERMASON is 100% well classified, concluding that this model is no longer able to bring credible results.

5.4 With the filtered variables - 6 variables

Once again, in order to try to improve the Naive Bayes classifier results, new variables were considered. As mentioned in the initial exploratory analysis, the variables were filtered according to their correlation with each other and with the response variable, resulting in a variables selection. When applying the classifiers to this new set of variables, the results were the same as those obtained in the previous subsection with the 8 variables, except for the Binary Logistic Regression algorithm that resulted in the following confusion matrix:

$$\begin{bmatrix} 106 & 1 \\ 0 & 707 \end{bmatrix}$$

Figure 20: Confusion Matrix for Binary Logistic Regression

In this manner, there was an observation of the BOMBAY class that was classified as DERMASON, which originated an impact on the performance measures: Recall is down to 99% and accuracy to 99.88%. When analyzing the balanced accuracy (99.53%), the difference is not that significant. The specificity of the DERMASON class also decreased (99%), which makes sense as there was one observation classified as false negative. However, the BOMBAY bean was still not properly classified by the Naïve Bayes classifier.

The running times were the following:

Running times	
Classifier	**Running Time**
KNN	0.021954 s
Naïve Bayes	0.005509 s
Logistic Regression	0.032968 s
Random Forest	0.019540 s
SVM	0.018400 s
XGBoost	4.314140 s

5.5 With the original paper variables - 5 variables

In the case of the variables used in the original study, the classifiers results also did not add anything relevant. The performance measures kept the same, namely, both of the classes were well classified for all of the classifiers, as exception of the Naive Bayes, which resulted in the confusion matrix shown bellow:

$$\begin{bmatrix} 0 & 107 \\ 1 & 706 \end{bmatrix}$$

Figure 21: Confusion Matrix for Naive Bayes

This classifier is clearly very sensitive to overfitting by the training set, and therefore cannot classify well on the test set. Therefore, Naïve Bayes algorithm has a bad prediction performance and the conclusions from the subsection 5.3. remain.

As for the running time, the results were the following:

Running times	
Classifier	**Running Time**
KNN	0.014843 s
Naïve Bayes	0.006321 s
Logistic Regression	0.032794 s
Random Forest	0.011176 s
SVM	0.015833 s
XGBoost	4.092819 s

6 Discussion and conclusion

To begin with, early data analysis revealed certain peculiarities of the data, such as the variables being on various scales (which could influence the results) and the dataset being unbalanced in number of datapoints for each class, both of which influenced the study's decisions. In reality, in order to address the issue of unbalanced data, it was decided to use oversampling techniques, specifically the SMOTE approach.

However, before using the SMOTE technique, six classifiers were used to categorize the raw data, and it was discovered that one of them (Naive Bayes) misclassified an entire class (BOMBAY).

The first thought was that, despite the fact that all of the classifiers utilized, with the exception of the Naive Bayes, produced equal results, it was chosen to choose the best model based on the running time. With that stated, the top classifiers (using the original data) were the Random Forest, Logistic Regression, KNN, and XGBoost, while the fastest model computing the results was KNN, with a time running of 0.0025s. The Naïhve Bayes, on the other hand, produced the worst outcomes.

Despite this, the SMOTE technique was used, first reproducing the number of observations of the minority subspecie and then balancing the observations of both dry bean species, however the results provided by both procedures (detailed in section 5.2) were similar, much like the original data. As a result, the Naive Bayes classifier was unable to correctly categorize the BOMBAY class.

The classifiers were then applied to the filtered data, which had eight variables. With the exception of the Naive Bayes, the dataset utilized in this case comprised attributes that exhibited more separability between classes, implying a higher separability power and as previously, the same performance metrics and confusion matrices were generated for all the models. The only difference between this scenario and the original data was the reduced execution time, which is to be expected given the absence of variables.

It was also decided to apply the classifiers to the filtered data with 6 variables in order to improve the Naive Bayes classifier performance. This variables were filtered based on their correlation with each other and with the response variable, but the same results were obtained as with the other eight variables, with the exception of the Binary Logistic Regression algorithm, which produced a different confusion matrix because there was an observation of the BOMBAY class that was classified as DERMASON, which had an impact on recall and accuracy, but the difference was not significant.

The final attempt to improve the Naive Bayes classifier was to include the variables from the original study, but the classifier's outputs did not offer anything useful. The performance measures remained the same, meaning that all of the classifiers, with the exception of the Naive Bayes, correctly categorized both groups.

With that in mind, and knowing that these strategies did not produce worst results when compared to the original data, it can be concluded that the classifiers should be used on the 5-variable model because it requires less computing work. Be aware that this is a trade off between overfitting (trade off bias variance)

In addition to this result, what supports the choice of KNN as a classifier for this problem is Occam's razor, to choose a simpler model over a more complex one as it generalize better. Suggesting a more complex model will overfit the data and thereafter perform badly on new data.

We can deduce from all of the methodologies evaluated that the Naive Bayes algorithm performs poorly in terms of prediction, and the original data findings remain. This is due to the strong assumption of predictor independence in this strategy. The explanatory factors' lack of independence (and relevance) is critical. Also, because the other methods are far more robust to overfitting than a Naive Bayes classifier, it is not unexpected that they performed significantly better.

Concerning possible future work, even though these results were good, more data is increasing the stability of the model and in this case it would be preferable to have more observations for BOMBAY class.

References

[1] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[2] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. (2016), 785–794.

[3] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[4] Evelyn Fix and Joseph Lawson Hodges. 1989. Discriminatory analysis. Nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique* 57, 3 (1989), 238–247.

[5] Tin Kam Ho. 1995. Random decision forests. 1 (1995), 278–282.

[6] Murat Koklu and Ilker Ali Ozkan. 2020. Multiclass classification of dry beans using computer vision and machine learning techniques. *Computers and Electronics in Agriculture* 174 (2020), 105507.

[7] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge.

[8] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. 2008. *Introduction to information retrieval*. Vol. 39. Cambridge University Press Cambridge.