Politecnico di Torino

Mathematics in Machine Learning Tesina

Vincenzo De Marco, s290373

A.A. 2021/2022

Dataset Overview

The dataset used is the Dry Bean Dataset. It is composed by:

- 7 categorical classes, corresponding to the beans' species
- 16 numeric features obtained from pictures of the grains (12 dimensions and 4 shape forms)
- 13 611 total samples

Data Exploration and Pre-processing

Class Encoding

Categorical values (the class in our case) must be encoded.

Simple integer encoding:

Seker,		0,
Barbunya,		1,
Bombay,		2,
Cali,	become	3,
Dermosan,		4,
Horoz,		5,
Sira		6

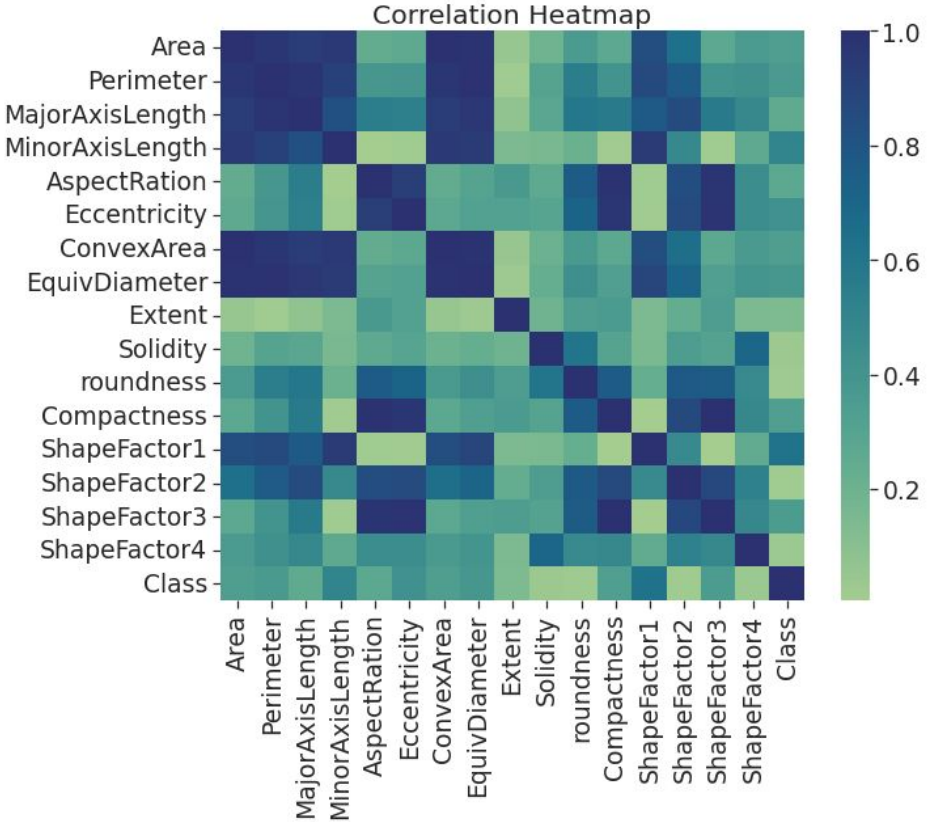
Duplicates and Missing Values

Being redundant information, duplicate records have been removed.

13 611 samples ⟶ 13 543 unique samples

We then check for missing values, that may harm the training, but the dataset is complete and so no action is taken.

Attributes' Correlation

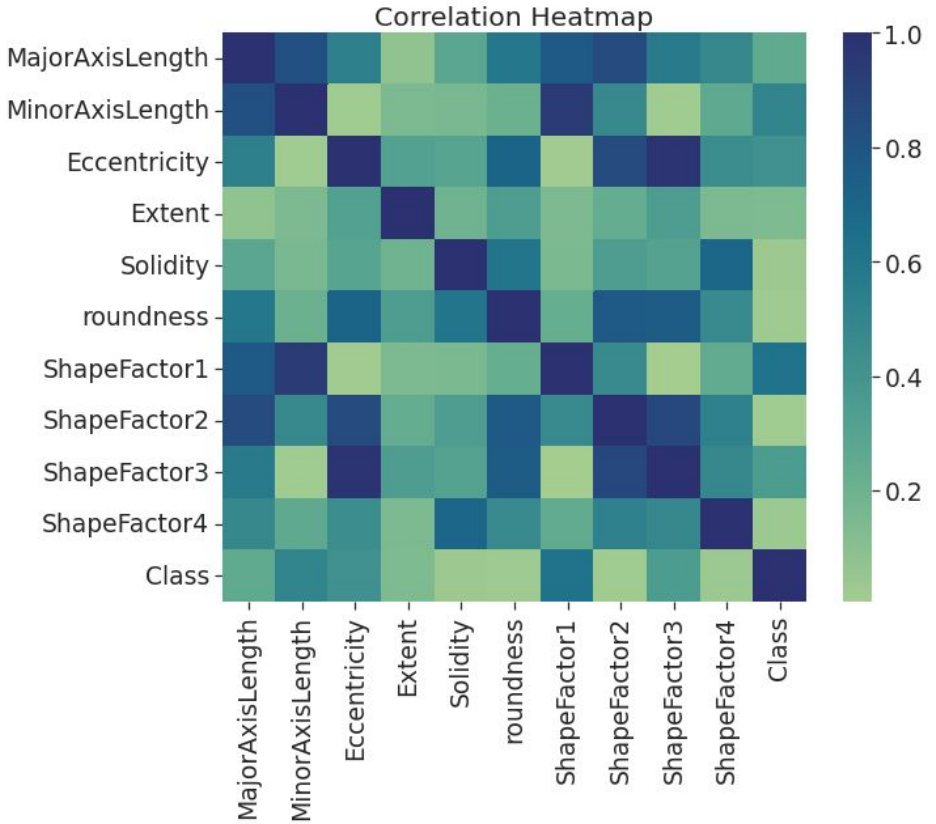


With an high ***Collinearity*** it can be difficult to separate the individual effects of collinear variables on the dataset's classes.
Hence we decide to drop:

- Area
- Perimeter
- Aspect Ration
- Convex Area
- Equiv. Diameter
- Compactness

Attributes' Correlation

Resulting dataset correlation Heatmap:



Dataset splits

To avoid applying pre-processing also on test data, we split the dataset in *Train* and *Test* sets. Every next preprocessing step is applied on the *Train* set only. These are the resulting splits:

Train set**:**

- 85% of the dataset
- 11 511 records
- Will be pre-processed and used to train the models

Test set**:**

- 15% of the dataset
- 2 032 records
- Left untouched to approximate real-world samples, to obtain a fair model evaluation.

Outliers Removal

We remove a data point X with the i-th attribute value X_i if:

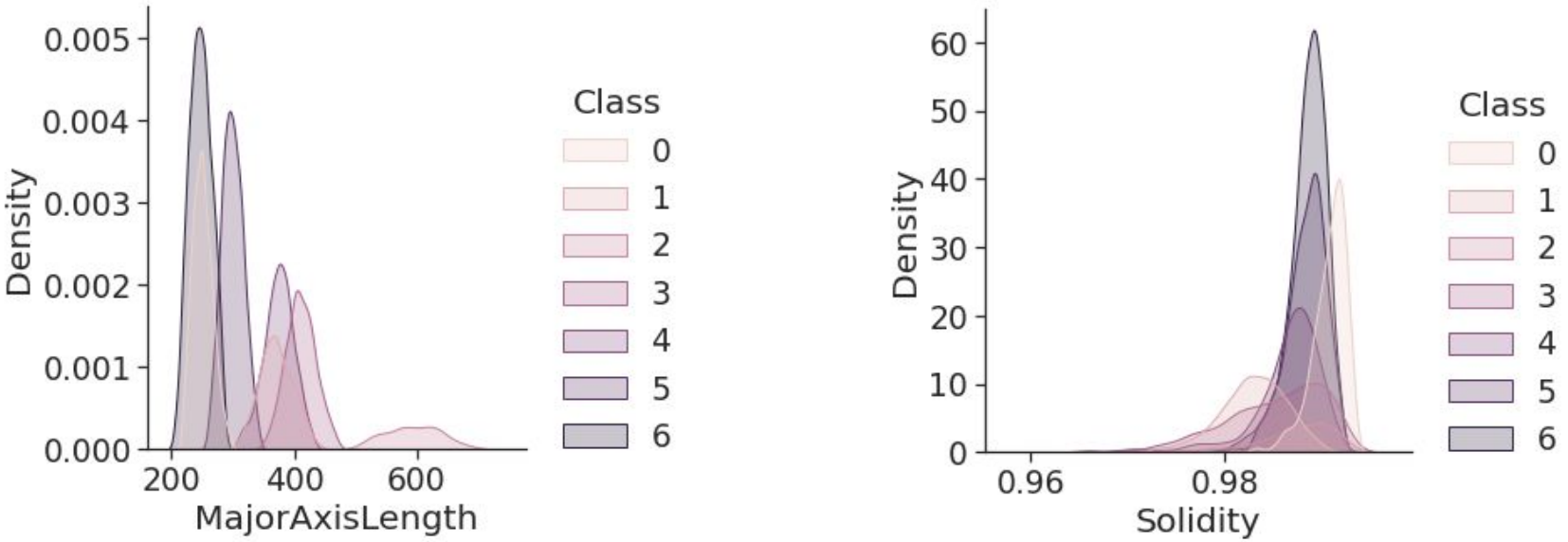
$$X_i < q_{0.01} \quad \text{OR} \quad X_i > q_{0.99}$$

Where q_z is the z-quantile of the attribute's values distribution (i.e. z% of values are < q_z)

We repeat this process for each attribute, resulting in:

11 511 samples ⟶ 9 343 samples

Features' Density Distribution



Each attribute distribution can be approximated with a **Normal Distribution**

Attribute Scaling

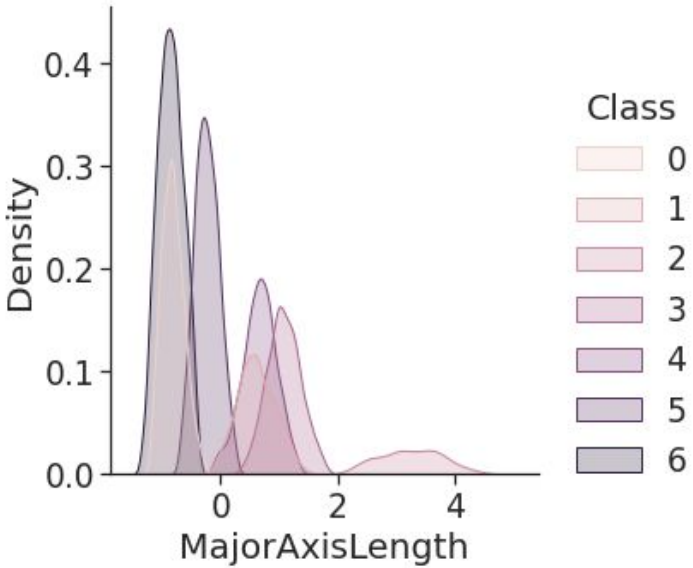
Since we're dealing with pseudo-normal distributions, we scale the data by removing the mean and scaling to unit variance:

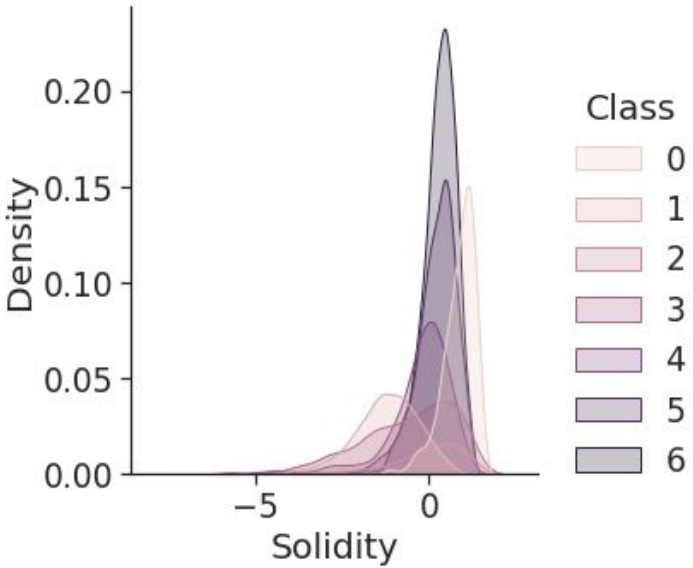
$$z = (x - \mu)/\sigma$$

where z is the normalized value, x is the original value, and μ and σ are the mean and standard deviation of the values' distributions, respectively.

We use the mean and standard deviation from the training distributions to scale also the *Test* set, so that the values' ranges are the same.

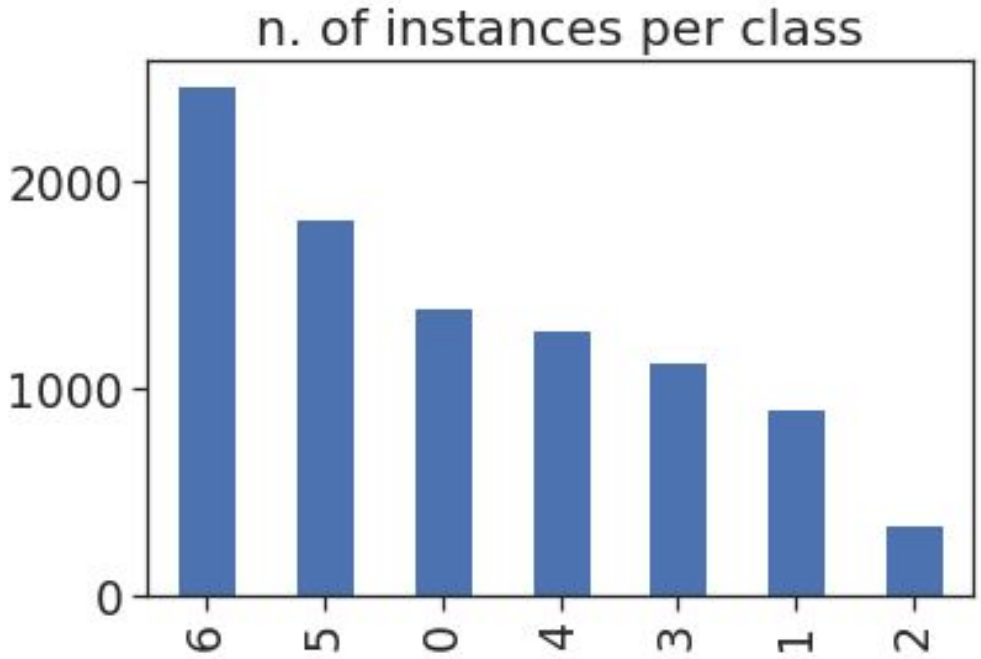
Features' Density Distribution (after scaling)





Values are **Normalized** but the distributions stay the same

Classes' Balance



Since the dataset is imbalanced, we decide to:

- *Oversample* classes 1, 2, 3
- *Undersample* classes 5, 6

Re-Sampling Strategy

Random Over Sampler:

Randomly x choose samples among the n available with replacement. Those samples will be duplicated, to have n+x final samples.

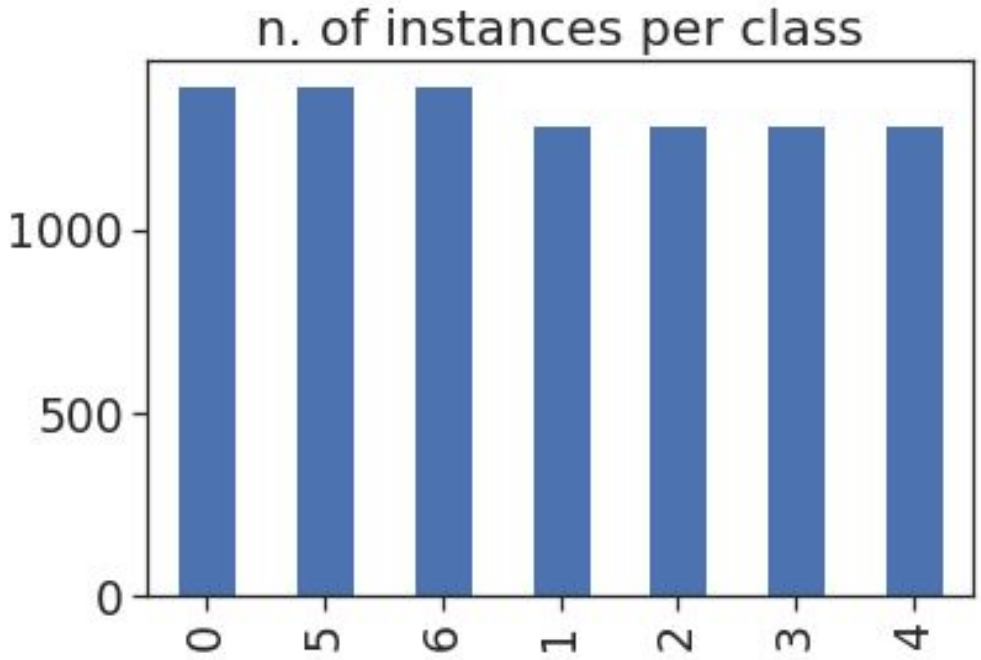
We Over-sample classes 1, 2, 3 to have the same number of samples of class 4, i.e. 1284

Random Under Sampler:

Randomly choose x samples among the n available. Those samples will be deleted, to have n-x final samples.

We Under-sample classes 5, 6 to have the same number of samples of class 0, i.e. 1392

Classes' Balance (after re-sampling)



Now the dataset is balanced:

- 1284 records for classes 1, 2, 3, 4
- 1392 records for classes 0, 5, 6
- Total number of records: 9312

Training and Evaluation Settings

Model Validation

To tune hyperparameters and validate our model we will use:

- Hyperparameters *Grid Search*: Scenario where, given a dictionary with the desired subset of hyperparameters and their possible values, every possible configuration of the model will be created and tested using *k-fold cross-validation*. The best performing model will be saved.

- *K-folds Cross Validation*: Technique based on splitting the training dataset in K sub-datasets. In each of the K iteration, a different one is used as *validation set* and the others for training the model. The K final scores are then averaged.

We use *K = 5* and *Accuracy* as validation metric.

5-folds Cross-Validation example

Validation and Evaluation Metrics

- **Accuracy:** Used both as validation and evaluation metric, it's calculated as:

$$accuracy = 1 - \frac{\sum_{i=1}^{n} I(y_i \neq Y_i)}{n}$$

Where n is the number of predictions, y is the predicted class and Y is the true class, and the indicator variable I() returns 1 if y=Y and 0 otherwise.

Validation and Evaluation Metrics

- **F1 score:** Used as an additional evaluation metric, it's calculated as:

$$f_1 = 2\frac{precision \cdot recall}{precision + recall} = \frac{tp}{tp + ((fp + fn)/2)}$$

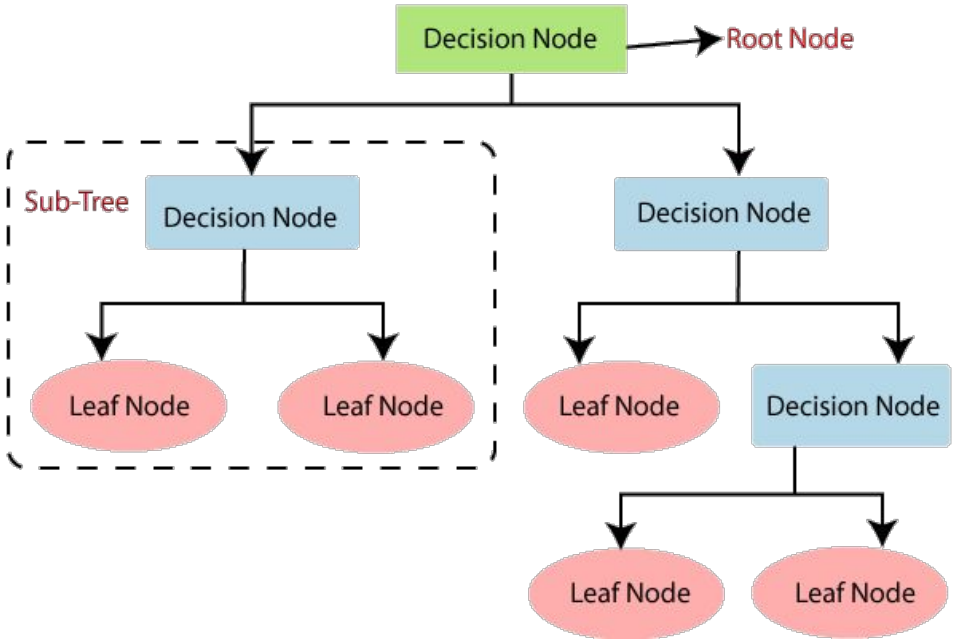
Where tp is the number of True Positives and fp and fn are the number of False Positives and False Negatives, respectively.

The f1 score can be calculated only in a binary classification scenario: in our multi-class setting, it is calculated as the average of the F1 score of each class with weighting depending on the number of instances for that class.

Selected Models

Decision Trees

- Tree-based method that stratify the predictor space in non-overlapping regions.
- Each decision node splits an attribute's feature space in two parts by applying a condition.
- *Recursive binary splitting*: choose the best local split; top-down, greedy approach.
- The majority class in the arrival leaf node is the predicted class.



↓ High variance and risk of over-fitting

Random Forest Classifier

- Meta classifier composed by aggregating many (usually independent) decision trees.

- Each tree is sequentially trained (***Boosting***) or independently trained on a different subset of the dataset (***Bagging***).

- Each tree outputs its prediction, then the ensemble decides by *majority voting*.

- *Bagging* or *Boosting* ensure low variance and good generalization capabilities.

Random Forest Classifier: Hyperparameters

- n_estimators (int): Number of decision trees in the forest.
- max_features (float): Fraction of features to consider when looking for the best split.
- max_depth (int): Maximum depth of the tree.
- min_samples_split (int): Minimum number of samples required to split a node.
- bootstrap (bool): Whether bootstrap samples are used when building trees.
- criterion: Function to measure the quality of a split (*Gini index* or *cross-entropy*).

Gini index:

$$G = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$$

Cross-entropy:

$$D = -\sum_{k=1}^{K} p_{mk} \cdot log(p_{mk})$$

Maximal Margin Classifier

- Supervised model with the goal of finding a separating hyperplane that maximize the distance between classes in the feature space:

$$y_i(\beta_0 + \beta_1 X_1 + ... + \beta_p X_p) > 0$$

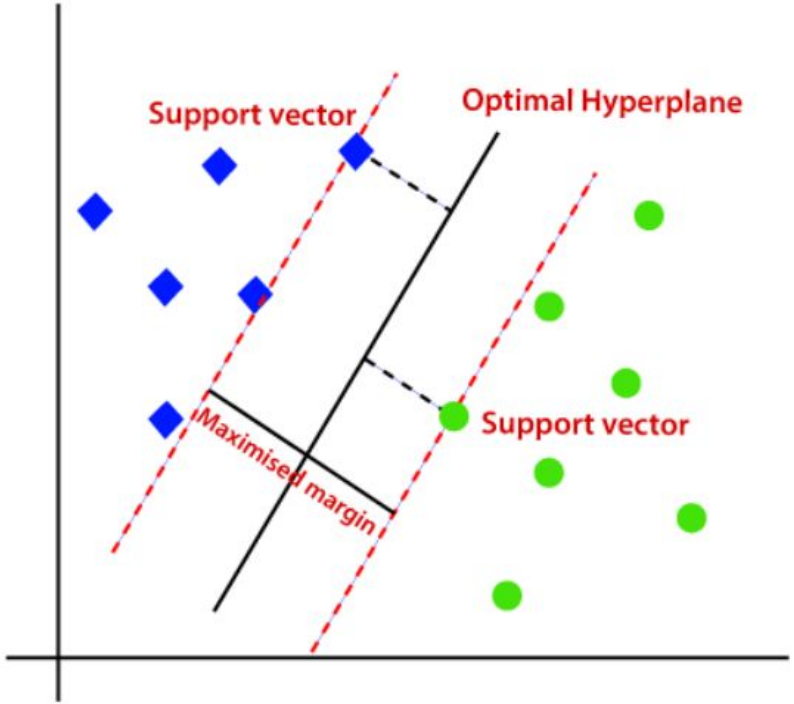
- A ***Margin*** is defined as the minimum distance between the hyperplane and the closest data point(s).

The maximal margin hyperplane is the solution to the optimization problem:

$$\begin{cases} \text{find } \beta_0, \beta_1, ..., \beta_p \text{ to maximize } M \\ \text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1 \\ y_i(\beta_0 + \beta_1 X_{i1} + ... + \beta_p X_{ip}) \geq M \end{cases}$$

Maximal Margin Classifier

The coefficients β_i will be equal to 0 for every data point except the ***support vectors***, i.e. the ones with the minimum distance from the hyperplane.



Support Vector Classifier

Two main differences w.r.t. Maximal Margin Classifier:

- Uses a ***soft margin***, that allows some observation to be in the wrong side of the margin or the hyperplane.

The optimization problem becomes:

$$\begin{cases} \text{find } \beta_0, \beta_1, ..., \beta_p \text{ to maximize } M \\ \text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1 \\ y_i(\beta_0 + \beta_1 X_{i1} + ... + \beta_p X_{ip}) \geq M(1 - \epsilon_i) \\ \epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C \end{cases}$$

(C is a tuning parameter)

Support Vector Classifier

- Uses ***kernels*** to enlarge the feature space.

  Kernels are generalizations of the inner product between two observations:

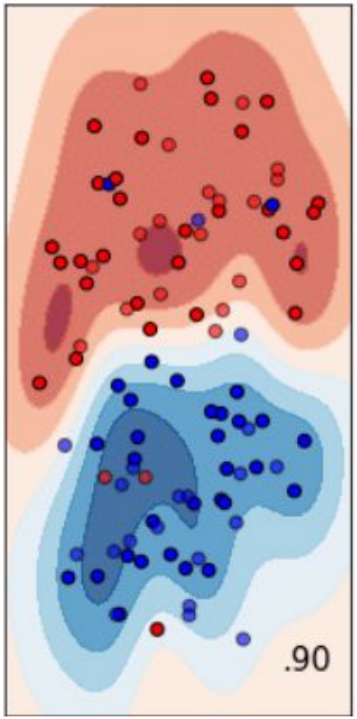
Linear kernel:

$$K(x_i, x_{i'}) = \sum_{j=1}^{p} x_{ij} \cdot x_{i'j}$$

Radial Basis Function kernel:

$$K(x_i, x_{i'}) = exp(-\gamma ||x_i - x_{i'}||^2)$$

Example of a rbf kernel:



SVC for Multi-class problems

The support vector classifier works by separating the (enlarged) feature space in 2 parts, so it can be applies only on binary classification problems.

To extend it for multi-class problems (with K classes), there are two approaches:

- One-Versus-One: Construct and train one SVC for each pair of classes in the dataset, comparing them. Classification is performed by assigning to the sample the class to which it was most frequently assigned by every pair-wise SVC.

- One-Versus-All: K SVCs, are created and fitted. Each SVC compares one of the classes to the remaining (K-1). Classification is achieved by assigning to the sample the class of the SVC with the biggest confidence in assigning to the sample its class, compared to the remaining ones.

Support Vector Classifier: Hyperparameters

- C (float): Regularization parameter for the soft margin.
- kernel: Specifies the kernel type to be used in the algorithm.
- gamma (float): Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’ kernels.
- decision_function_shape: Whether to return a one-vs-rest decision function or the original one-vs-one decision function.
  - Internally, only the one-vs-one multi-class strategy is implemented, so the one-vs-rest matrix is constructed from the one-vs-one matrix.

Results Obtained

Random Forest Classifier

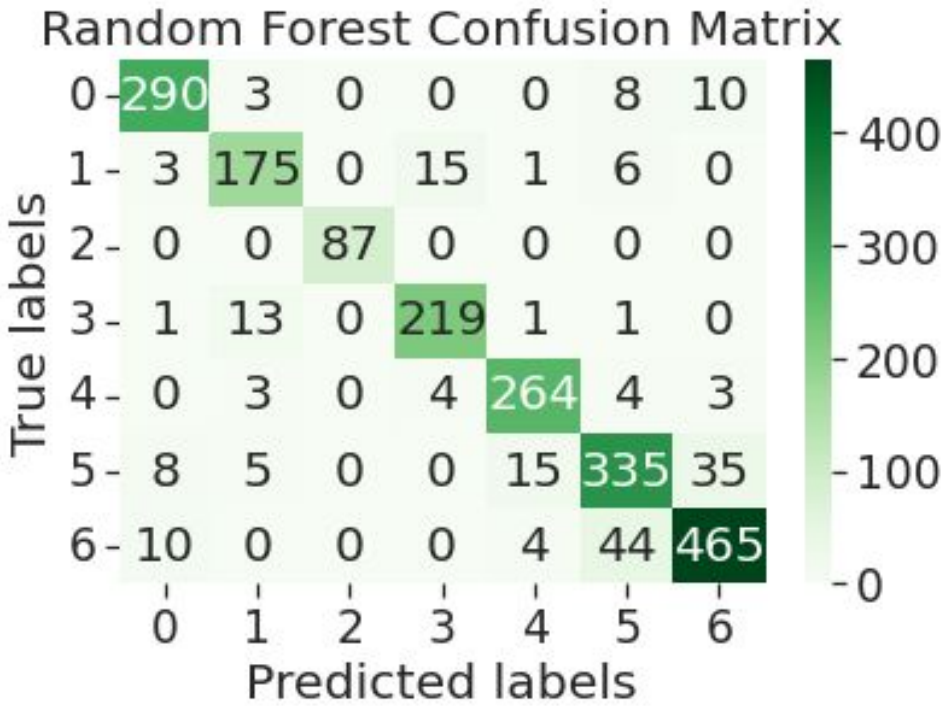
Hyperparameters values (**selected values**) and scores obtained:

Parameters	Values				
n_estimators	100	200	**300**		
max_features	**'log2'**	'sqrt'	0.33	0.6	0.8
max_depth	5	10	15	**20**	
min_samples_split	2	**4**	6	8	
bootstrap	**'True'**	'False'			
criterion	'gini'	**'entropy'**			

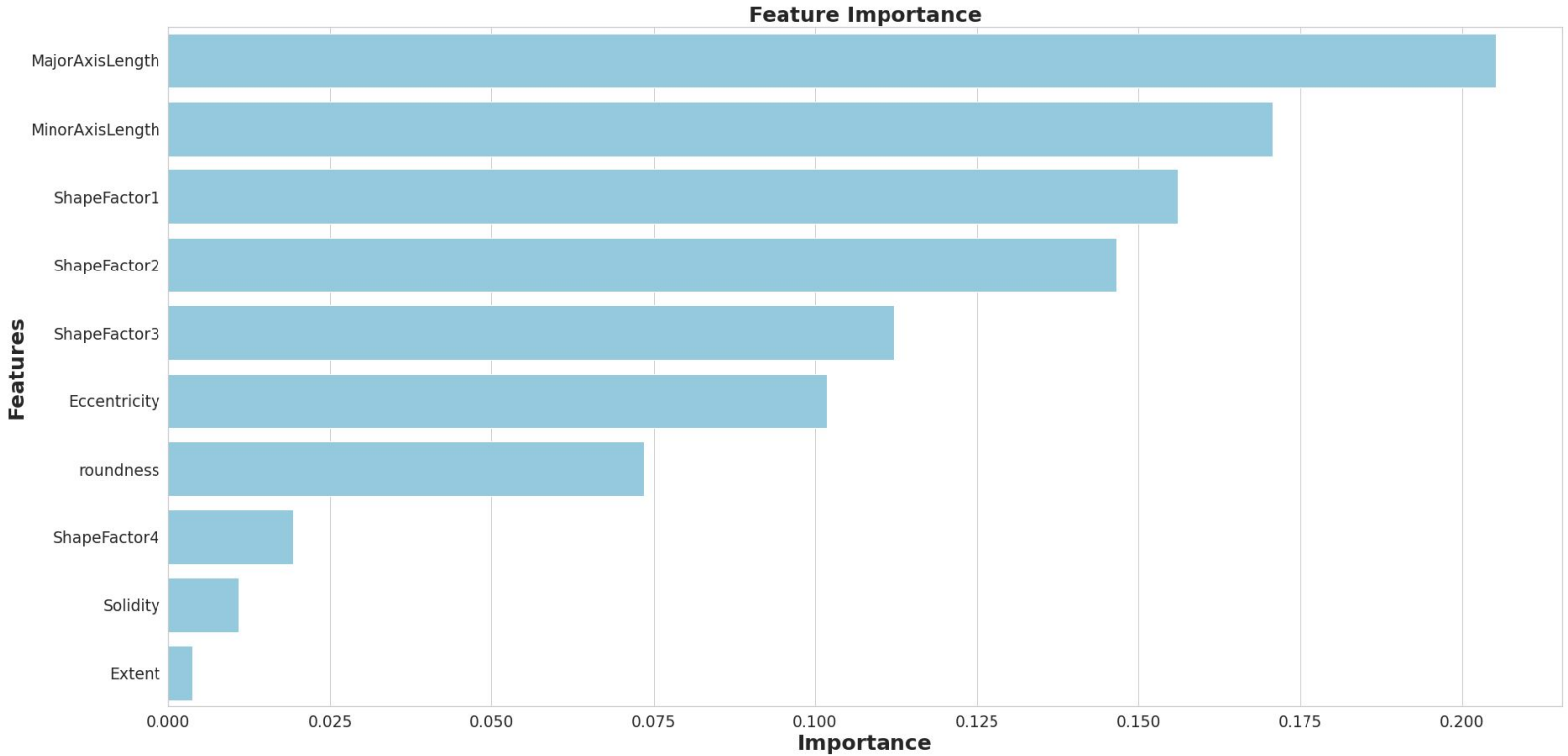
Random Forest	
Val accuracy	0.9794
Test accuracy	0.9040
Test F1 score	0.9038

Random Forest Classifier

Confusion Matrix:



Random Forest Classifier



Support Vector Classifier

Hyperparameters values (**selected values**) and scores obtained:

Parameters	Values				
C	0.1	0.5	1	10	**100**
gamma	1	0.75	0.25	**0.1**	0.01
kernel	'linear'	**'rbf'**	'poly'	'sigmoid'	
decision_function_shape	'ovo'	**'ovr'**			

SVC	
Val accuracy	0.9775
Test accuracy	0.9203
Test F1 score	0.9201

Support Vector Classifier

Confusion Matrix: