Politecnico di Torino

Corso di laurea magistrale in Data Science and Engineering

Mathematics in Machine Learning

Anuran Calls (MFCCs) dataset analysis

Professors:

Prof: Francesco Vaccarino

Prof: Mauro Gasparini

Candidate:

Stefano Galvagno s290191

ANNO ACCADEMICO 2021/2022

Contents

Chapter 1

Introduction

This dataset contains acoustic features from **syllables of anuran** (frogs and toads) **calls**. These features were extracted from 60 audio recordings, containing information about 4 different families, 8 genus and 10 species. Each audio is associated with a single specimen, and the record ID is supplied as an extra column. All audio files has been segmented at syllable level by means of Matlab, finally obtaining 7195 instances for training. From every extracted syllable 22 **MFCCs** were calculated by using 44 triangular filters. Mel-frequency cepstral coefficients (MFCCs) are coefficients that collectively make up a mel-frequency cepstrum (MFC). Particularly, they are derived from a type of cepstral representation (i.e. the result of computing the inverse Fourier transform (IFT) of the logarithm of the estimated signal spectrum) of the audio clip. All these numeric features have been normalized by the authors of the data set.

The task is a *Multiclass* classification problem, namely we aim at classifying instances into one of three or more classes. More precisely, the goal is to **classify the family** of a given anuran, given its audio recording. Alternatively, one can consider to classify the species or the genus.

Chapter 2

Data Exploration

2.1 Class distribution

It is quite common to deal with datasets characterized by imbalanced classes. This fact often results in a **prediction bias** in favor of most frequent classes, leading to inaccurate results. As we can see in 2.1, 2.2 and 2.3, more than half of the observations belong to the Leptodactylidae family, Adenomera genus and AdenomeraHylaedactylus species. Therefore, during the data preparation step we will adopt an appropriate solution to the problem.

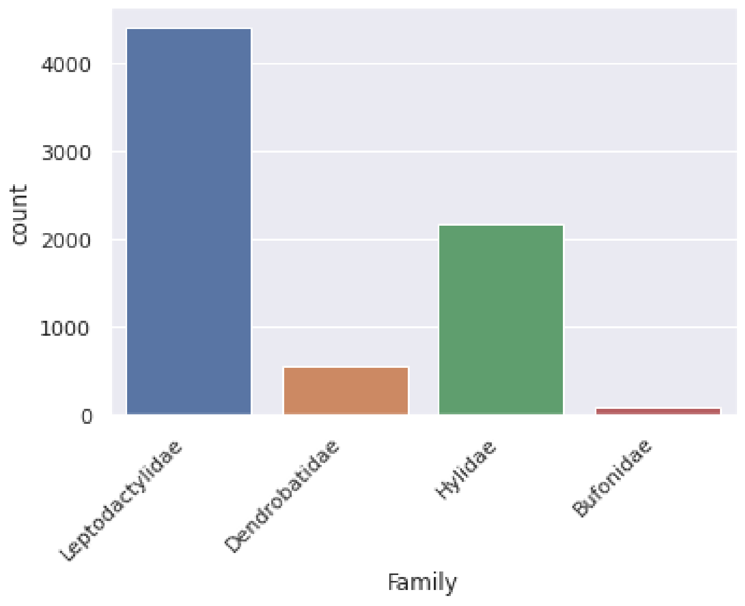

Figure 2.1: Family distribution

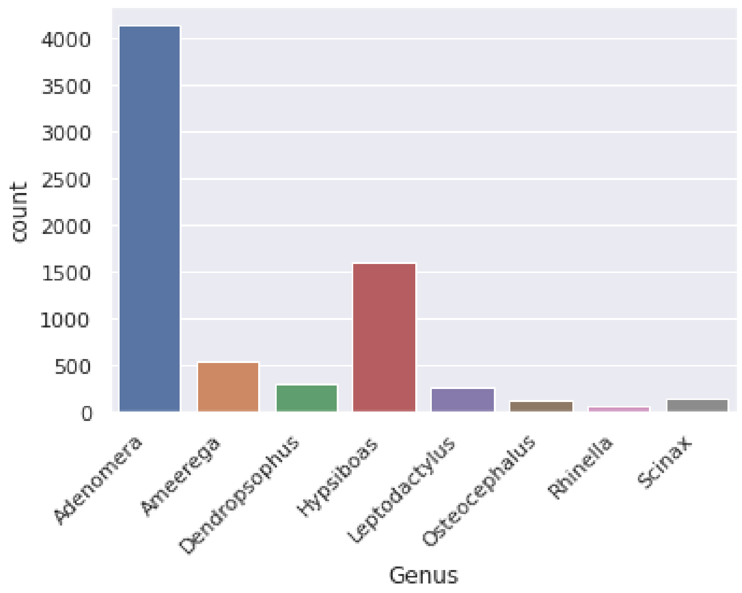

Figure 2.2: Genus distribution

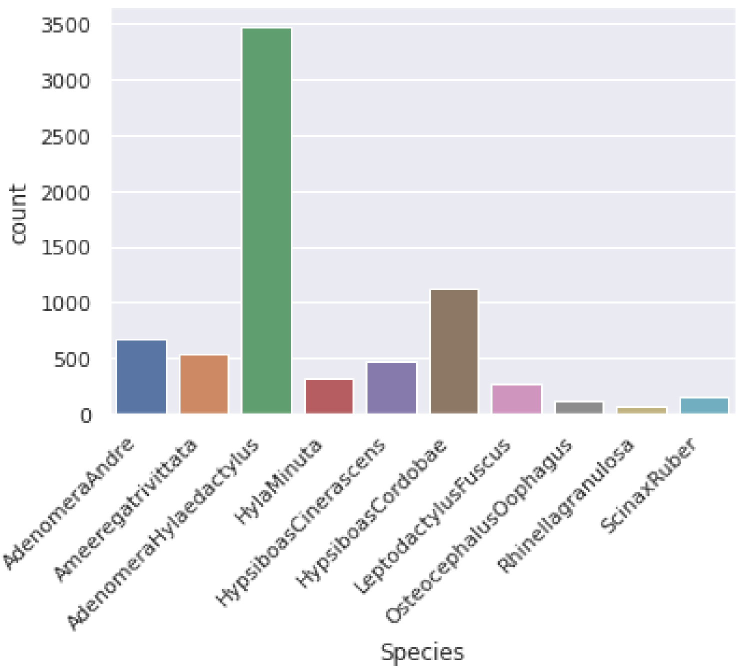

Figure 2.3: Species distribution

2.2 Missing values

The dataset does not contain any missing value, therefore imputation is not necessary.

2.3 Features analysis

As mentioned in the introduction, the dataset consists of 22 numerical (float) features, normalized between -1 and +1, and three categorical attributes (possible labels). As we can see in fig. 2.4, in general, most of attributes are roughly normally distributed, except for MFCCs1 that assumes the value 1 in almost all the cases. Consequently, this attribute will be discarded, since it does not carry any useful information.
One can observe the same features from another perspective, namely considering the distributions at family level (species and genus will be disregarded).
As we can see in fig. 2.5, in general, the mean tends to fall around the same values across different families. Particularly, Bufonidae in all cases are the most concentrated class (lowest variance), whereas all the others have different dispersion, depending on the considered attribute.

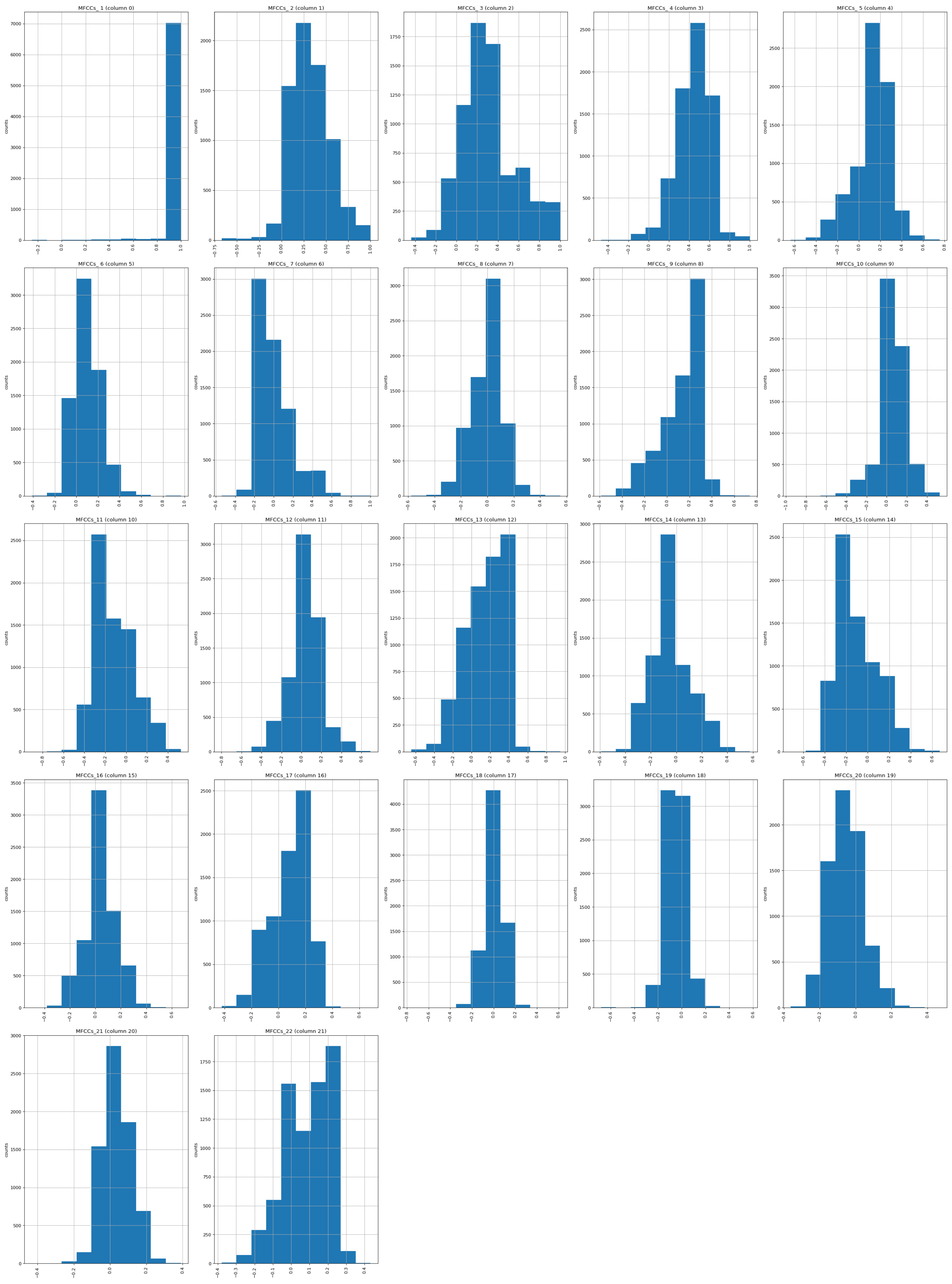

Figure 2.4: Features distribution

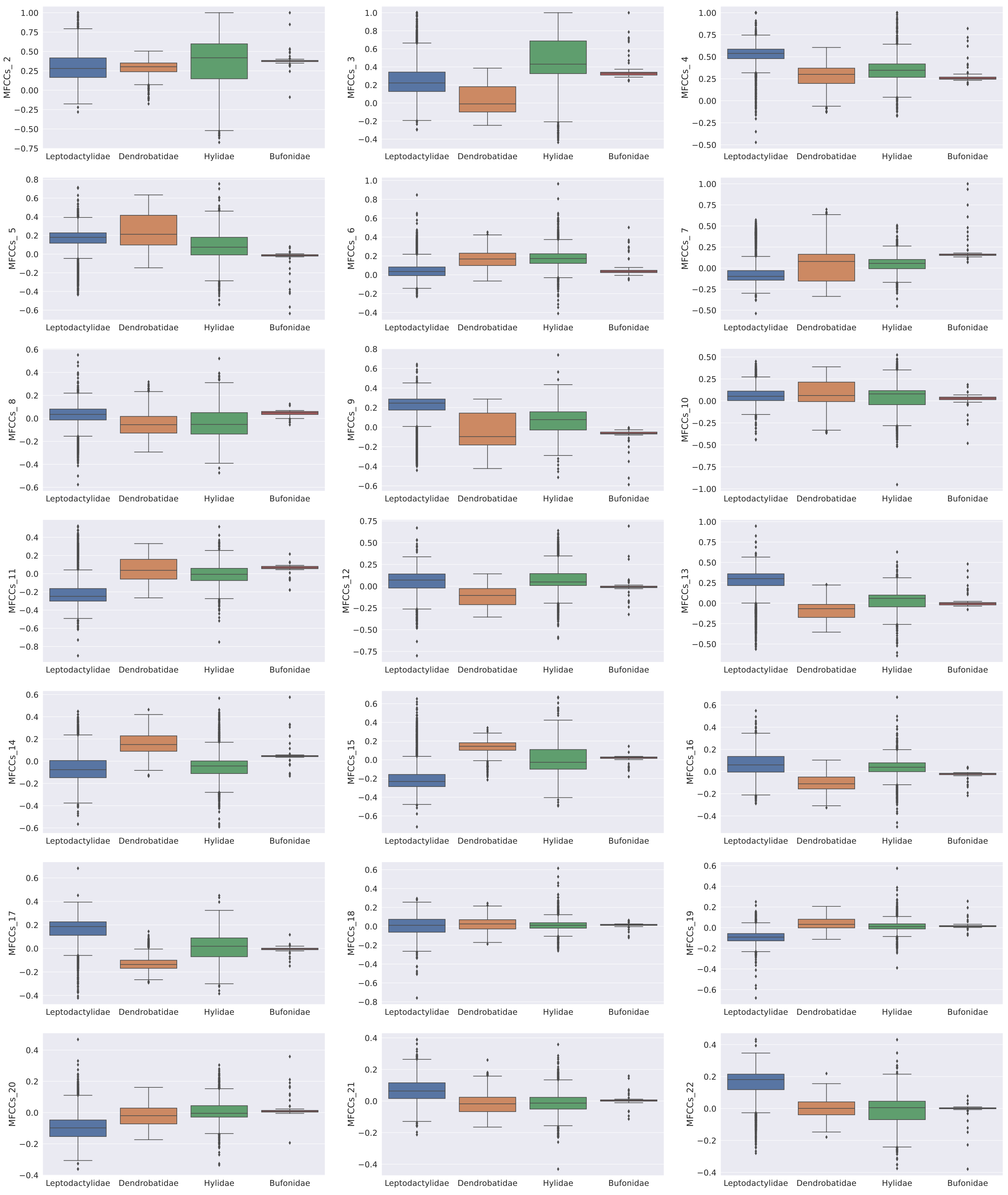

Figure 2.5: Features distribution by family

2.4 Features correlation

An interesting tool for identifying patterns is the **Pearson correlation** matrix. The Pearson correlation is a measure of **linear** correlation between two sets of data. Specifically, it is computed as follows:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X,Y)}{\sigma_x \sigma_y}$$

namely, dividing the covariance of the two variables by the product of their standard deviations. The drawback of this index is the fact that it is not able to detect neither non-linear relationships nor, in case of linear correlation, the slope. Therefore we cannot conclude anything about pairs of features that present Pearson's values close to 0. In our case, inspecting the features correlation matrix, we can observe that some features at index i are negatively correlated with attributes at index i+2 and positively correlated with the ones at index i+4.

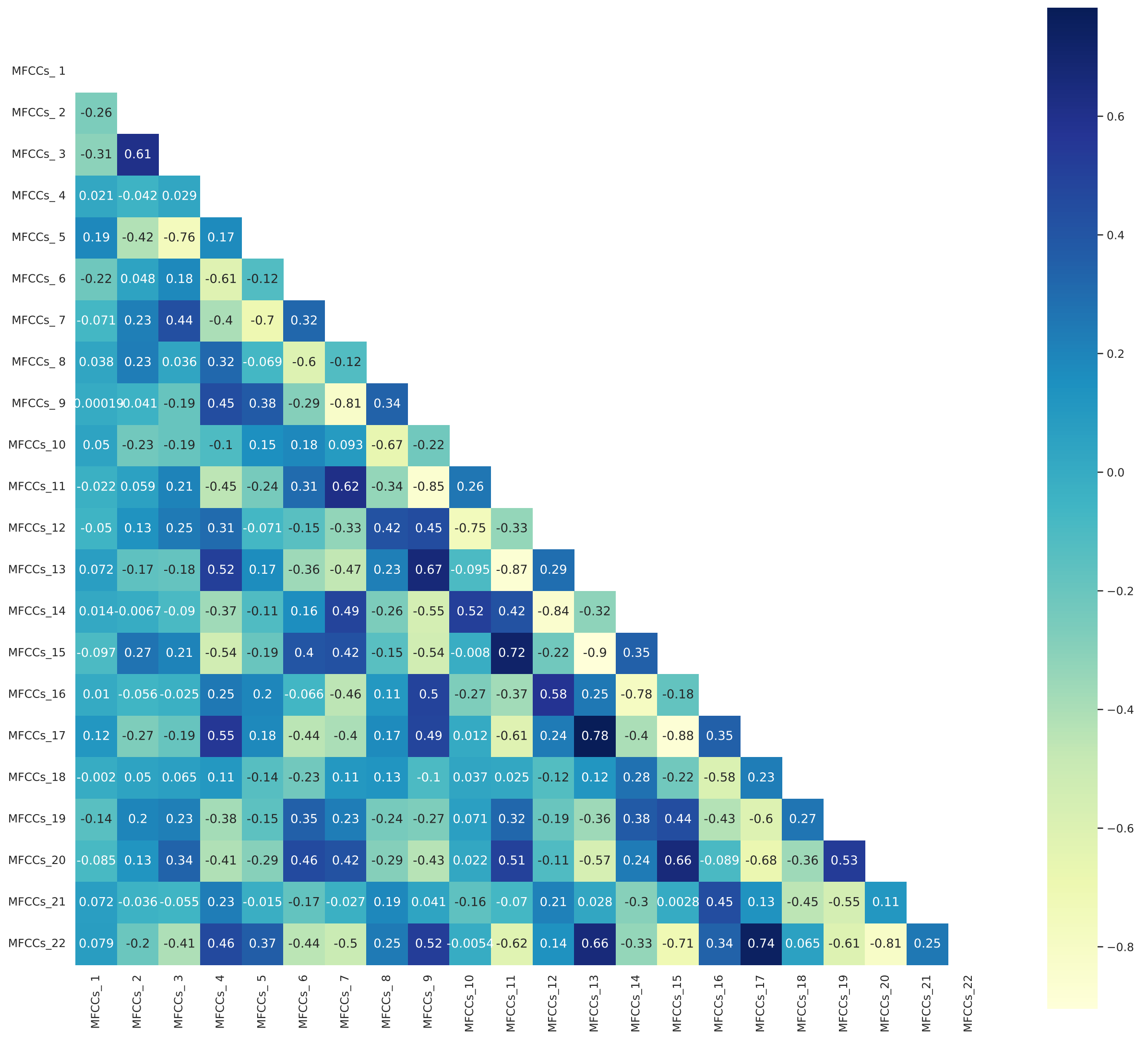

Figure 2.6: Features correlation matrix

Chapter 3

Data Preparation

3.1 Outliers management

As we can see in fig. 2.5, there are some outliers. Since all the features were already normalized by the creator of the dataset, we can assume that these outliers are just rare observations and not errors occurred during anuran's calls recordings.
A way to handle them is the **Interquartile range method**. It consists in computing the difference between the third and the first quartile, obtaining the IQR:

$$IQR = Q3 - Q1$$

Then, this quantity is multiplied by a constant (1.5) to discern outliers. More precisely, all the observations that fall outside the range

$$(Q1 - 1.5 * IQR,\ Q3 + 1.5 * IQR)$$

are considered outliers and, therefore, discarded.

3.2 Dimensionality reduction

3.2.1 Principal Component Analysis

Principal components analysis (PCA) is a statistical procedure that allows to **summarize** a dataset, described by a certain number of attributes, with a smaller number of features that explain most of the **variability** in the original set. This procedure outputs a set of **uncorrelated** (orthogonal) **features**. Specifically, considering a dataset with size $n \times m$:
we first compute the average value by column

$$\bar{x_j} = \frac{1}{n}\sum_{i=1}^{n} X_{ij}$$

the mean matrix is

$$\bar{X} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \bar{x}$$

then we obtain the mean-subtracted data B:

$$B = X - \bar{X}$$

not changing how data points are positioned relative to each other, but centering the mean in the origin.
Then, the scatter matrix is obtained in the following way:

$$S = B^T B$$

The next step consists in computing the eigenvectors corresponding to the largest k eigenvalues of S.
The final result is obtained projecting the points on the new set of coordinates identified by the eigenvectors. In other terms, the first component is the line in the m-dimensional variable space (each attribute is a dimension) that maximizes the sum of squared distances from the projected points, on the line itself, to the origin. In order to get a coordinate value along the PC-line, each observation (yellow dot in 3.1) has to be projected onto this line.
The unit-1 vector along the PC-line is the eigenvector, while the sum of squared

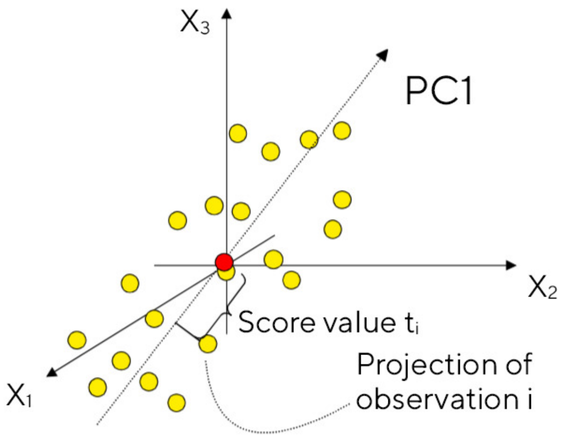

Figure 3.1: First 2 components obtained by PCA

distances is the eigenvalue for PC1. The second principal component is also represented by a line in the m-dimensional variable space, which is orthogonal to the first PC. This line also passes through the average point, and improves the approximation of the X-data as much as possible.
We can convert the eigenvalues into variations dividing them by n-1. For the sake of the example, imagine that the dataset is described by two attributes only. Let

us assume that the variation for PC1 and PC2 are 15 and 3, respectively. It means that the total variation around these principal components is 18. It means that PC1 accounts for $15/18 = 83\%$ of the total variation around PCs. This information is used to set the number of components to be considered. Usually we want certain number of components to retain 90% of the total variance. In our case, we need 7 components to reach such a threshold, as shown in 3.2.

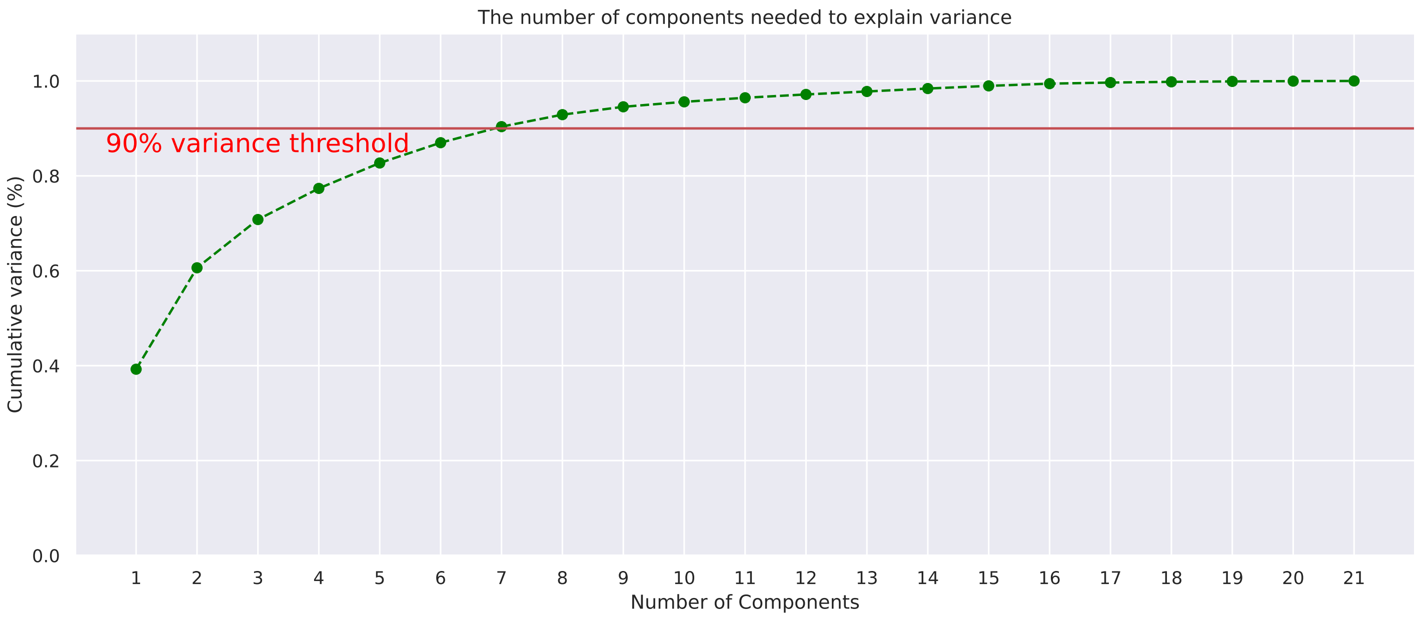

Figure 3.2: Cumulative variance explained

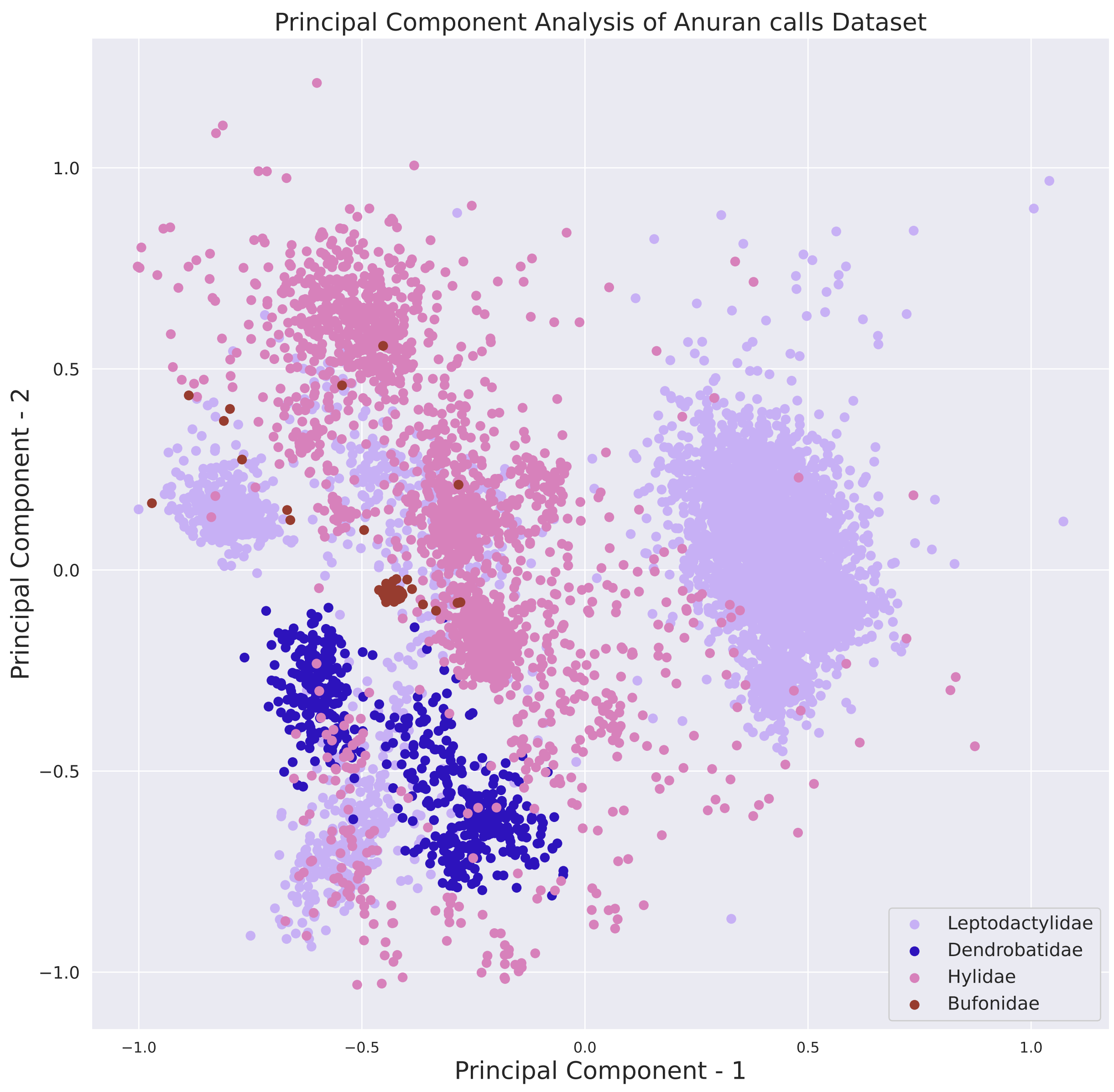

Figure 3.3: First 2 components obtained by PCA

3.3 Sampling

In order to obtain a balanced distribution over the species, we have to perform some sampling technique. We can either opt for oversampling or undersampling. Oversampling techniques are preferred over undersampling techniques in most circumstances. The reason for this is that when we undersample data, we tend to exclude occurrences that may contain crucial information. As we observed with 2.1, oversampling seems to be the most appropriate solution for not losing too much information.

SMOTE

SMOTE is a data augmentation algorithm that creates **synthetic data points** depending on the original data points. It can be thought as a more advanced variant of a standard oversampling. SMOTE has the advantage of not creating duplicate data points, but synthetic data points that are somewhat different from the original data points.
The algorithm is divided in the following steps:

- Draw a random sample from the minority class.
- For the observations in this sample, identify the k nearest neighbors.
- Take one of those neighbors and identify the difference between the current data point and the selected neighbor.
- Multiply the vector by a random number between 0 and 1.
- To obtain the synthetic data point, add this to the current data point.

The overall **result** is a **shifted copy** in the direction of the neighbor. Fig. 3.4 visually explains the procedure.

Moreover, it is fundamental to oversample the training data only.
Applying the procedure on the validation set would result in a misleading evaluation process.

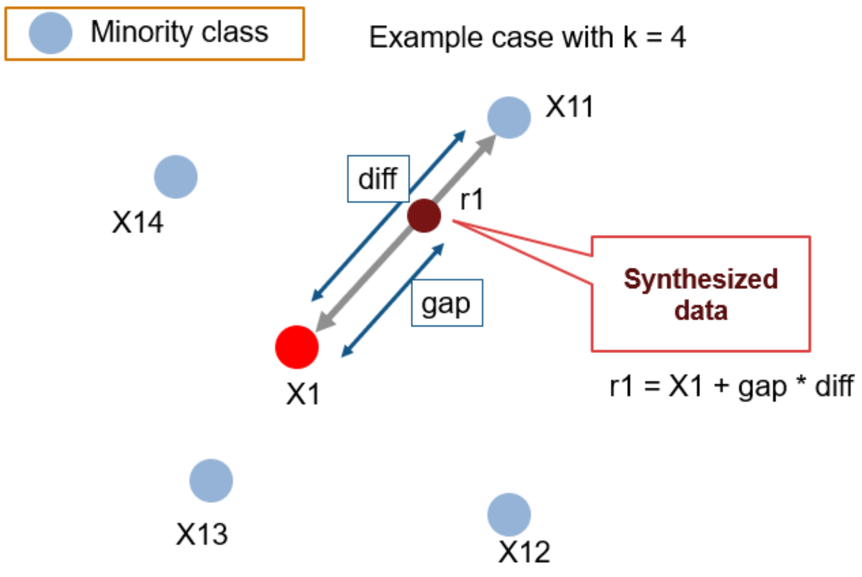

Figure 3.4: Generation of a synthetic sample by SMOTE

Chapter 4

Training and testing

4.1 Multi-class classification

Multi-class classification is not supported by all classification prediction models. For instance, Logistic Regression and Support Vector Machines algorithms were intended for binary classification and do not support classification tasks with more than two classes by default.

Splitting the multi-class classification data set into numerous binary classification data sets and fitting a binary classification model on each, is one method for applying binary classification methods for multi-classification problems. The One-vs-Rest and One-vs-One tactics are two instances of this strategy. In our case we opted for One-vs-Rest.

4.1.1 One-vs-Rest

One-vs-Rest consists in splitting the multi-class data set into **multiple binary classification problems**. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident. For example, let us assume that a classification problem is to classify various fruits into three types of fruits: lemon, orange or apple. Since there are three distinct classes, there will be three classification problems:

- Problem 1: Lemon vs [Orange, Apple]
- Problem 2: Orange vs [Lemon, Apple]
- Problem 3: Apple vs [Orange, Lemon]

4.1.2 One-vs-One

One-vs-one separates a multi-class classification data set into binary classification tasks, similar to one-vs-rest. Unlike the one-vs-rest method, which divides the data

set into one binary data set for each class, the one-vs-one method divides it into one data set for each class vs every other single class.
If we consider the previous example, the resulting number of problems will be:

- Problem 1: Lemon vs Orange
- Problem 2: Lemon vs Apple
- Problem 3: Orange vs Apple

4.2 K-fold cross validation

Cross-validation is a resampling technique for evaluating machine learning models on a small sample of data. The process includes only one parameter, k, which specifies how many groups a given data sample should be divided into. As a result, the process is frequently referred to as k-fold cross-validation. It is divided in the following steps:

- Shuffle the dataset randomly.
- Split the dataset into k groups.
- For each unique group:
  - Take the group as a hold out or test data set
  - Take the remaining groups as a training data set
  - Fit a model on the training set and evaluate it on the test set
  - Retain the evaluation score and discard the model
- Compute the k-fold CV estimate by averaging each iteration's model scores

In particular, each observation in the data sample is assigned to a distinct group and remains there throughout the method. This means that each sample has the chance to be utilized in the hold out set once and to train the model k times. We chose a k-fold cross-validation with k=5 for a decent trade-off between runtime and score f1, so the classifiers are trained on 80 percent of the train data in each iteration. Furthermore, the stratified version has been used, to ensure that each fold of data set has the same proportion of observations with a given family. The whole procedure in summarized in 4.1.

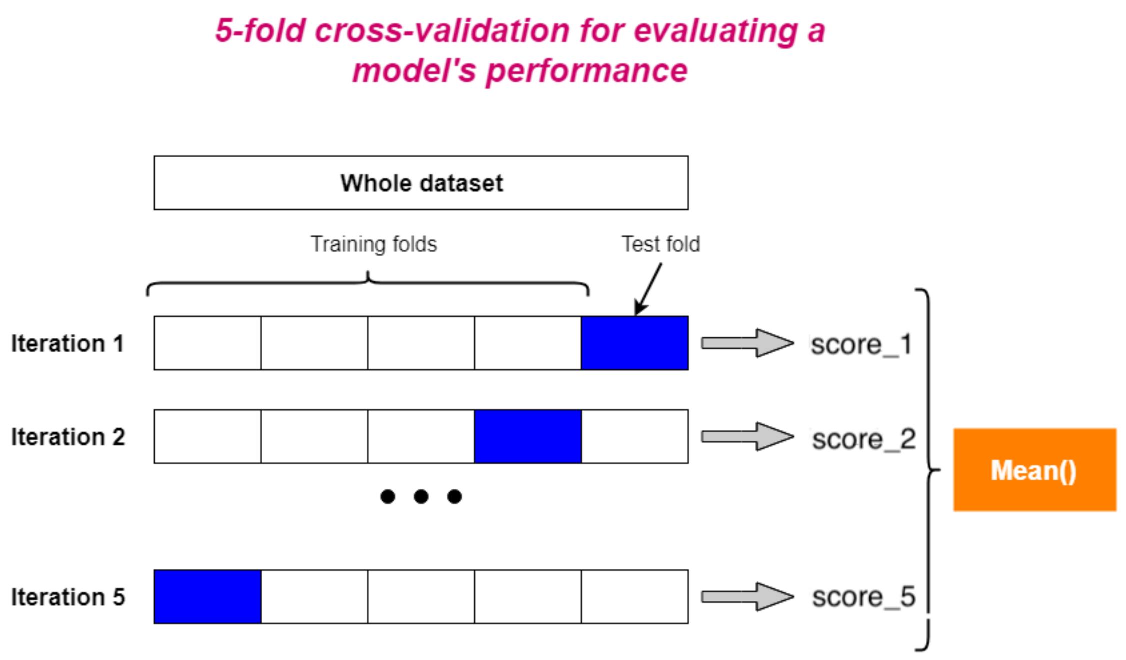

Figure 4.1: 5-fold cross validation procedure

4.3 Metrics

The performance of machine learning algorithms is usually evaluated through the **accuracy**. Considering binary problems, accuracy is the number of correctly predicted data points out of all the data points. It's calculated by dividing the number of true positives and true negatives by the total number of true positives, true negatives, false positives, and false negatives:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

However, accuracy could be misleading when you're working with a class-imbalanced data set, like this one, where there is a significant disparity between the number of Leptodactylidae and all the other families.
Therefore one can consider other metrics.
Firstly, **precision**, calculated by dividing the true positives by anything that was predicted as a positive:

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

namely, the number of instances that are relevant, out of the total instances the model retrieved. Then, **recall** (or True Positive Rate), is calculated by dividing the true positives by anything that should have been predicted as positive.

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

is the fraction of relevant instances that were retrieved.

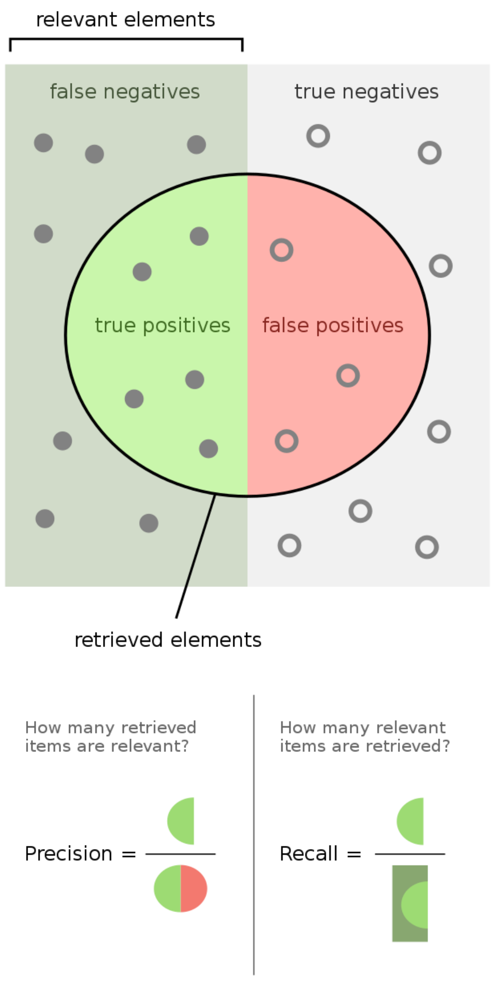

Figure 4.2: Precision and Recall

An alternative is represented by the **F1 score**, that takes into account both precision and recall:

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

We will consider this last metric for scoring.

Chapter 5

Classification models

5.1 Decision trees

Decision trees are machine learning models characterized by nodes. The very top of the tree is the root node and the intermediate nodes are named internal nodes or branches. Lastly, nodes that have no outgoing edges are called leaf nodes. One of the features is examined at each node in order to partition the observations during the training phase or to make a specific data point follow a specific path while producing a prediction. When constructing decision trees, they are developed by recursively considering different features and using the one that best separates the data at each node. Usually, the criteria used for evaluating the optimal feature are **Gini index** and the **Cross Entropy**, both **measuring** the **node impurity**. The best feature to split with is the one that presents the lowest impurity.
When dealing with attributes that can assume just discrete values, the Gini index is defined in the following way:

$$GINI = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

denotes the Gini Index for a given node t. $\hat{p}_{mk}$ is the proportion of training observations in the m-th region that are from the k-th class. It is a measure of total variance across K classes.
When dealing with continuous attributes, the mean of each two consecutive values (ordered from lowest to highest) of the training data is used as possible threshold. In other words, all the records below the threshold are treated like having the same attribute in the case of a discrete feature.
The second measure, the Entropy, is defined in the following way:

$$Entropy = -\sum_{k=1}^{K} \hat{p}_{mk} log_2 \hat{p}_{mk}$$

Trees divide the predictor space in non-overlapping regions that are high dimensional rectangles (fig. 5.2). Usually leaf nodes are still impure to some extent, therefore the output label is determined by majority voting at region level.

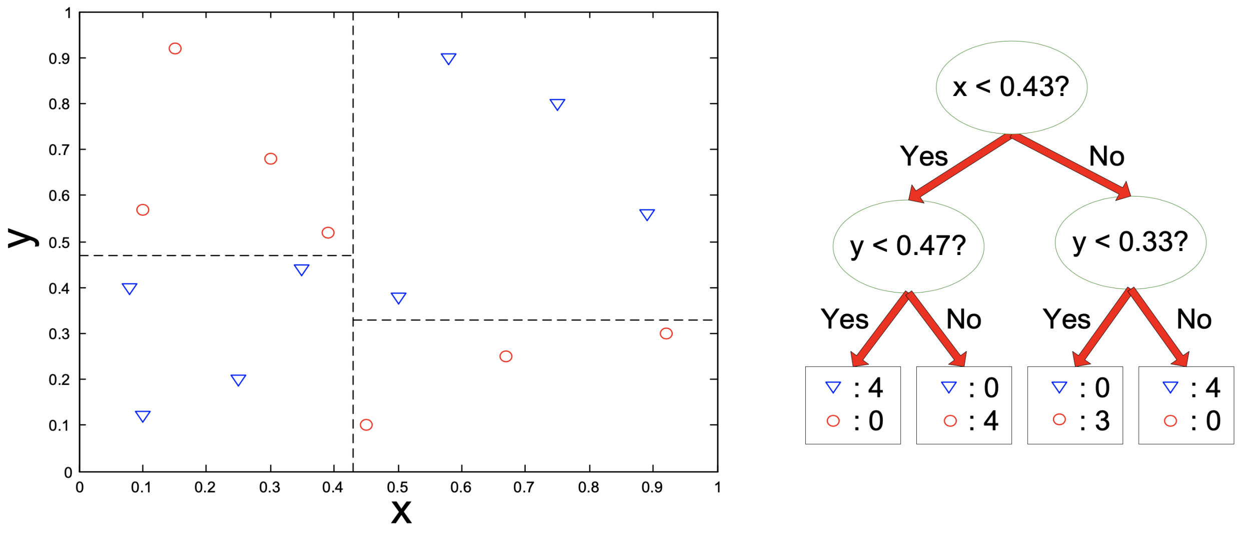

Figure 5.1: Decision boundaries with 2 dimensions

The main **drawback** of this model is **overfitting**. There are two main ways to deal with this problem: Pre-Pruning and Post-Pruning.
Pre-Pruning (Early Stopping Rule) consists in stopping the algorithm before it becomes a fully-grown tree. Typical stopping conditions for a node are:

- Stop if all instances belong to the same class
- Stop if all the attribute values are the same

By contrast, Post-pruning consists in growing decision tree to its entirety, trimming the nodes of the decision tree in a bottom- up fashion and, if generalization error improves after trimming, replace sub-tree by a leaf node. Class label of leaf node is determined from majority class of instances in the sub-tree.

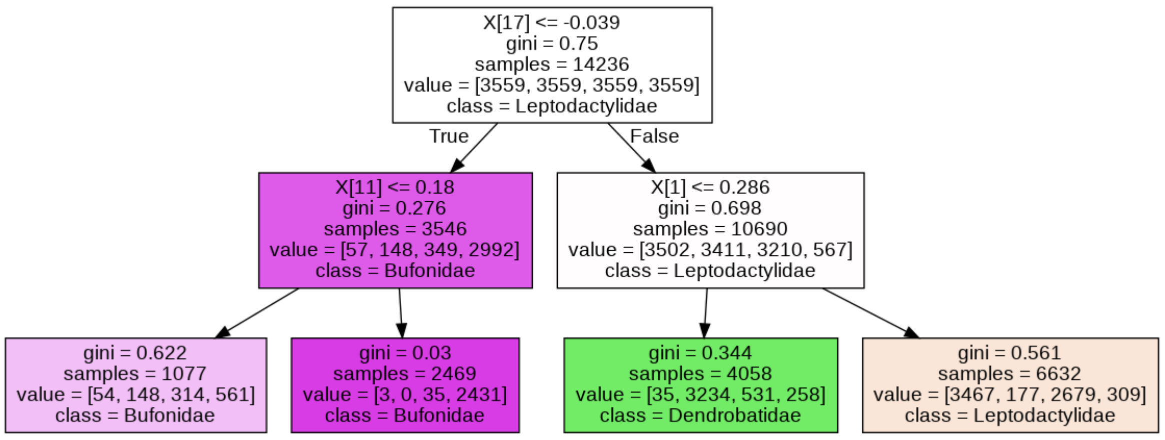

Figure 5.2: Example of decision tree built with our dataset (max depth: 2)

5.2 Random forest

Random forest is a supervised algorithm that builds decision trees on different samples and takes their majority vote for classification. The fundamental concept behind random forest is a simple but powerful one: the wisdom of crowds.
Particularly, the overall model exploits several decorrelated decision trees, that operate as an ensemble, outperforming any of the individual constituent models.
In order to ensure the aforementioned decorrelation, the model exploits two methods: Bagging and Feature Randomness.
Bagging (Bootstrap Aggregation) consists in allowing each individual tree to randomly sample from the data set resulting in different trees. More specifically, we give each tree a **training set** of size N (like the initial set), **randomly sampled** with **replacement** instead of the original training data. For instance if our training data was [1, 2, 3, 4, 5, 6] then we might give one of our trees the following list [1, 2, 2, 3, 6, 6].
Secondly, when a split in a tree is examined, a **random selection** of **m predictors** from the whole set of p predictors is picked as split candidates. Only one of the m predictors can be used in the split. This creates even more variance among the trees in the model, resulting in decreased correlation and increased diversification.
Fig. 5.3 explains this procedure.

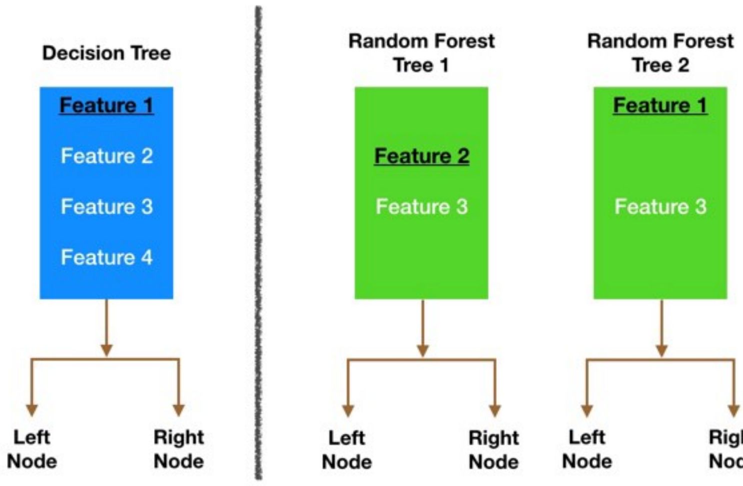

Figure 5.3: Decision tree vs Random forest at a given split

The decision tree has four features. The best attribute for splitting is Feature 1, therefore it picks it at the first step. By contrast, the Random forest tree 1 initially considers Features 2 and 3 (selected randomly) for its node splitting decision. Consequently, despite Feature 1 being the best overall choice, it cannot be used, therefore it selects Feature 2.
As a result, we end up having trees in our random forest that are not only trained on distinct sets of data (due to bagging), but also make decisions based on different

features.
The final prediction is determined with majority voting, in the classification case.

5.3 K Nearest Neighbors

KNN is an unsupervised technique that aims at predicting the labels given the **nearest data points**. Particularly, it is divided in the following steps:

- Select the number K of the neighbors
- Compute the distance (e.g. Euclidean distance) of K number of neighbors
- Take the K nearest neighbors
- Among these k neighbors, count the number of the data points in each category
- Assign the new data points to that category for which the number of the neighbors is maximum

For the sake of the example, suppose the configuration illustrated in Fig. 5.4. Given k = 5, the new data point is labeled with class A, since there are 3 observations out of 5 that belong to that class, between its 5 nearest neighbors.

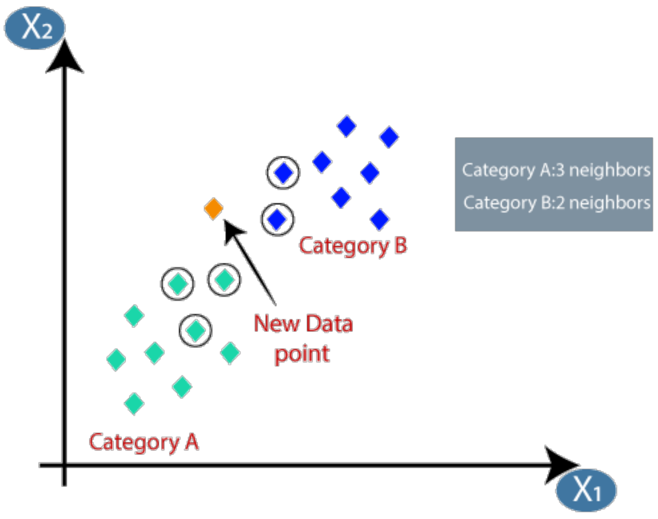

Figure 5.4: KNN with k = 5

The choice of the hyperparameter k does not follow a general rule, but usually low values imply more sensitivity to outliers and with high values the neighborhood may include points from other classes. In our case, the F1 decreases with the number of neighbors, as shown in 5.5.

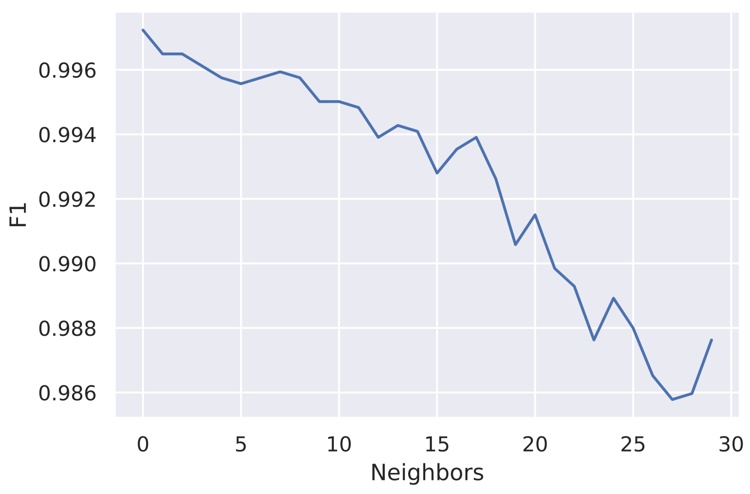

Figure 5.5: F1 score with respect to the number of neighbors

5.4 Logistic regression

Logistic regression is a supervised model for binary classification problems based on probability. In contrast to linear regression, which aims at predicting continuous values, instead of fitting a line to the data, this model fits an "S"-shaped logistic function:

$$f(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(w_1x_1+...+w_kx_k+b)}}$$

where $[x_1, ..., x_k]$ is the vector of features that represent a single observation. The curve is bounded between 0 and 1, therefore it represents a probability. Particularly,

$$\mathbb{P}(y=1) = \frac{1}{1+e^{-z}}$$

$$\mathbb{P}(y=0) = 1 - \mathbb{P}(y=1) = \frac{e^{-z}}{1+e^{-z}}$$

Finally, the classification is made applying a threshold to the obtained probabilities, for example assigning the label 1 if $\mathbb{P}(y=1) > 0.5$. The task consists in finding the set of weights and the bias $w_1, ..., w_k,\ b$ that appear in the equation. The weights represent the importance of a feature to determine the outcome of the classification task. These terms are estimated using the maximum likelihood method.

5.5 Support Vector Machines

SVM stands for Support Vector Machines and it is a supervised machine learning technique that can be used for classification.
The goal is to create a **maximum margin classifier**, which means that we want

to choose a decision boundary such that the space around it, named margin, is as big as possible. The decision boundary is defined by the equation

$$< w, x_i > -b = 0$$

that is the compact form for

$$w_1 x_1 + ... + w_p x_p - b = 0$$

describing an hyperplane. Particularly, w are the coefficients, b is the intercept and x is a p-dimensional vector (p predictors).

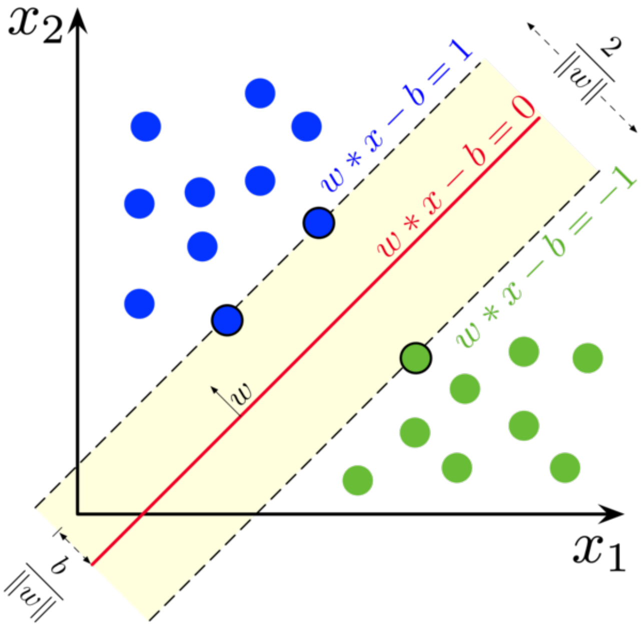

Figure 5.6:

For the sake of simplicity, as we can see in fig. 5.6, let us consider the case of two predictors, therefore the decision boundary is a line.
We can observe two further lines, described respectively by the equations:

$$< w, x_i > -b = \pm 1$$

The goal is to obtain proper values for w and b in order to maximize the margin, namely the space between these two lines. In particular,

$$margin\ size = \frac{2}{||w||}$$

Therefore we want to maximize the aforementioned quantity, that is equivalent to minimize the denominator.

$$\min_{w,b} \frac{||w||^2}{2}$$

In addition we require a constraint to be satisfied: everything on one side must be classified as +1 and everything on the other side as -1. Mathematically:

$$\exists w \in R^d, b \in R : y_i(< w, x_i > -b\ ) > 1\ \ \forall i$$

in other words, we require that the product of the predicted label and the actual label (that can assume the values +1 or -1) is > 1.
The closest points to the hyperplane are named **support vectors** since, with their elimination, the position of the aforementioned hyperplane would be modified. These points are the ones that perfectly satisfy the condition:

$$y_i(< w, x_i > -b\ ) = 1$$

The reasoning made so far applies in the ideal case of linearly separable classes, but in most of the cases we face the situation represented in fig. 5.7. The approach in this scenario is to accept a certain level of errors, assigning a penalty to mistakes based on how big they are. The evaluation method in this case is the Hinge loss:

$$Hinge\ loss = max(0,\ 1 - y_i(< w, x_i > -b))$$

where $< w, x_i > -b$ is the score, that assumes values greater than 1 in terms of module outside the margin and values between -1 and 1 inside it. Consequently, we can distinguish 3 scenarios, as shown in fig. 5.7:

1 Correctly classified: the observation falls in the correct region with respect to the margin. The value for the loss is 0.

2 Serious error: the observation falls in the wrong region with respect to the decision boundary, therefore it is misclassified. The loss assigns to it a relatively high value (>1).

3 Correctly classified with uncertainty: the observation is correctly classified but it falls inside the margin, therefore it gets a small penalty (between 0 and 1).

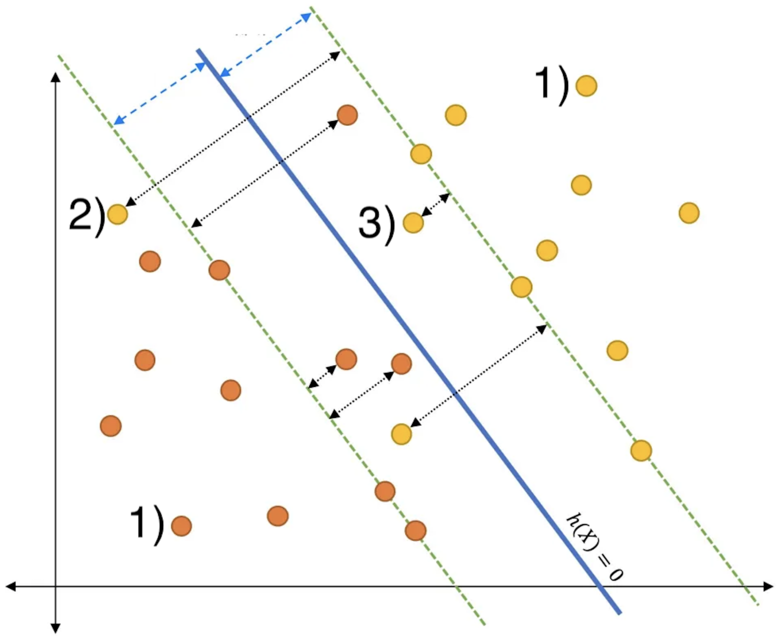

Figure 5.7: 3 scenarios in soft margin case

The overall optimization problem is formulated as:

$$\min_{w,b} C\frac{1}{N}\sum_{i=1}^{N}(max(0,\ 1-y_i(<w,x_i>-b)))+\frac{1}{2}||w||^2$$

namely, we minimize the sum of two terms: the average Hinge loss and magnitude of the vector w. The parameter C tunes how much we care about each of these two terms relatively.

Evidently, support vector classifiers will perform poorly with a non linear class boundaries. The solution for this issue consists in mapping the original feature space in a higher-dimensional one. The main drawback is the increase of computational complexity. The Kernel trick solves this problem: the idea is that the algorithm behind SVM does not need to know what each point is mapped to under a non-linear transformation. Rather, it only requires to know how each point compares to each other data point after we apply the non-linear transformation

$$f(x)\ vs\ f(x')$$

that mathematically corresponds to

$$f(x)^T f(x') := k(x,x')$$

the so called Kernel function.

Some of the most common are listed below:

Linear kernel:

$$\begin{aligned} f(x) &= x \\ k(x,x') &= x^T x' \end{aligned}$$

Polynomial kernel:

$$\begin{aligned} f(x) &= (x_1, x_2, x_1 x_2, x_1^2 x_2^2) \\ k(x,x') &= (1 + x^T x')^2 \end{aligned}$$

Radial basis function kernel (RBF):

$$\begin{aligned} f(x) &= \ (infinite\ dimensional) \\ k(x,x') &= e^{-\gamma ||x-x'||^2} \end{aligned}$$

The last case clearly shows the importance of the Kernel trick since $f(x)$, being infinite dimensional, cannot be used in a computer program, but the kernel expression is incredibly simple.

Everything we've discussed up to this point has only applies to binary classification. One-versus-one and one-versus-all are the two most widely used K-class algorithms for SVM.
The one-versus-all strategy fits K SVMs and builds an index of confidence that the test observation belongs to the k-th class. Finally, it assigns the label according to the maximum confidence index. Particularly, the latter is defined as a measure of how far a point is from the negative region.

Chapter 6

Results

As we can see in the first table below the best results have been obtained with the data set without outliers, for all the models. Furthermore, KNN outperforms all the other classifiers regardless of the considered data set.

F1	**Original dataset**	**No outliers dataset**	**PCA dataset**
Decision tree	0.959	0.984	0.970
Random forest	0.988	0.992	0.987
KNN	0.998	0.998	0.995
Logistic regression	0.923	0.957	0.840
SVM	0.940	0.961	0.881

In addition, the last table shows the best parameter configuration for each model and each data set. Since the hyperparameter tuning process is not the main scope of this study, in order not to spend too much computational time, just the most relevant options have been considered.

Best configuration	**Original dataset**	**No outliers dataset**	**PCA dataset**
Decision tree	criterion: 'entropy', max_depth: 10	criterion: 'gini', max_depth: 10	criterion: 'entropy', max_depth: 10
Random forest	bootstrap: 'False', criterion: 'entropy', n_estimators: 200	bootstrap: 'False', criterion: 'entropy', n_estimators: 50	bootstrap: 'False', criterion: 'entropy', n_estimators: 500
KNN	n_neighbors: 1	n_neighbors: 1	n_neighbors: 3
Logistic regression	C: 1000, penalty: 'l2'	C: 1000, penalty: 'l2'	C: 10, penalty: 'l2'
SVM	C: 100, gamma: 0.01, kernel: 'rbf'	C: 100, gamma: 0.0001, kernel: 'linear'	C: 100, gamma: 0.01, kernel: 'rbf'