Analysis of the MAGIC Gamma Telescope Data Set

Mathematics for Machine Learning
Year 2021-2022

Politecnico
di Torino

Federico Lorenzo Pes s287822

Contents

Figure 1: Cherenkov gamma telescope

1 Introduction

The data present in this dataset are generated to simulate registration of high energy gamma particles in a ground-based atmosphere. Cherenkov gamma telescope (CTA) observes high energy gamma rays, taking advantage of the radiation emitted by charged particles produced inside the electro-magnetic showers initiated by the gammas, and developing them in the atmosphere. More precisely, when the gamma rays reach the earth's atmosphere they interact with it, producing cascades of subatomic particles. These cascades are also known as air or particle showers. Light travels 0.03 percent slower in air, thus these ultra-high energy particles can travel faster than light in air, creating a blue flash of "Cherenkov light". Although the light is spread over a large area (250 m in diameter), the cascade only lasts a few billionths of a second. CTA's large mirrors and high-speed cameras will detect the flash of light and image the cascade generated by the gamma rays for further study of their cosmic sources allowing reconstruction of the shower parameters. The available information consists of pulses left by the incoming Cherenkov photons on the photomultiplier tubes. Depending on the energy of the primary gamma, a total of few hundreds to some 10000 Cherenkov photons get collected, in patterns (called the shower image), allowing to discriminate statistically those caused by primary gammas (signal) from the images of hadronic showers initiated by cosmic rays in the upper atmosphere (background). The goal of the classification is to correctly classify the gamma signal (g) from hadron (h,background).

2 Data exploration

2.1 Attribute analysis

Each event is characterized by the following ten attributes:

1. **fLength**: continuous, major axis of ellipse [mm]
2. **fWidth**: continuous, minor axis of ellipse [mm]
3. **fSize**: continuous, 10-log of sum of content of all pixels [in phot]
4. **fConc**: continuous, ratio of sum of two highest pixels over fSize [ratio]
5. **fConc1** : continuous, ratio of highest pixel over fSize [ratio].
6. **fAsym**: continuous, distance from highest pixel to center, projected onto major axis [mm]
7. **fM3Long**: continuous, 3rd root of third moment along major axis [mm]
8. **fM3Trans**: continuous, 3rd root of third moment along minor axis [mm]
9. **fAlpha**: continuous, angle of major axis with vector to origin [deg]
10. **fDist**: continuous, distance from origin to center of ellipse [mm]

The last attribute is the **label** which is defined as **g** for *gammas (signal)* and **h** for *hadrons (background)*.

2.2 Dataset distribution

It is important to also consider how the dataset is divided among the two classes, in order to undertand if the dataset is balanced. In general we can consider a dataset **balanced** if the split of the classes are equal or almost. The dataset consists in **12332 (65%)** events which are gamma and **6888 (35%)** hadron events. Hence the dataset is unbalanced.

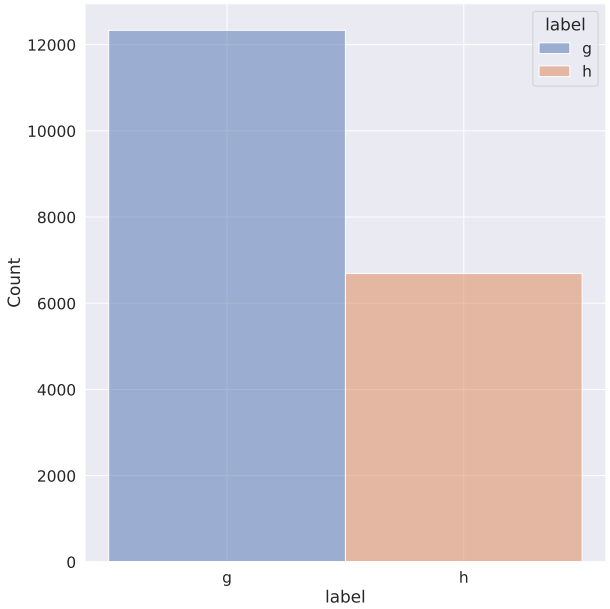


Figure 2: Points distribution over the classes gamma and hadron

2.3 Statistical overview

To have an idea about the distribution of the data we provide for each attribute: mean, standard deviation, minimum value, maximum value and the quartiles.

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist
count	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000
mean	53.250154	22.180966	2.825017	0.380327	0.214657	-4.331745	10.545545	0.249726	27.645707	193.818026
std	42.364855	18.346056	0.472599	0.182813	0.110511	59.206062	51.000118	20.827439	26.103621	74.731787
min	4.283500	0.000000	1.941300	0.013100	0.000300	-457.916100	-331.780000	-205.894700	0.000000	1.282600
25%	24.336000	11.863800	2.477100	0.235800	0.128475	-20.586550	-12.842775	-10.849375	5.547925	142.492250
50%	37.147700	17.139900	2.739600	0.354150	0.196500	4.013050	15.314100	0.666200	17.679500	191.851450
75%	70.122175	24.739475	3.101600	0.503700	0.285225	24.063700	35.837800	10.946425	45.883550	240.563825
max	334.177000	256.382000	5.323300	0.893000	0.675200	575.240700	238.321000	179.851000	90.000000	495.561000

Figure 3: Statistical features of the attributes

2.4 Data Visualization

The boxplots in the figure below show that each attribute has different range which sometimes may present high fluctuations. For this reason, during the preprocessing step, we are going to scale the data in order to reduce this diversity. Moreover from the figure we can see that there are some points distant from the others. These points might be caused by variability or also by mistake. Hence, we are also going to perform an outlier detection and removal step.

Interesting deduction can also been made from the pairplot in Fig. 5. On the diagonal are plotted

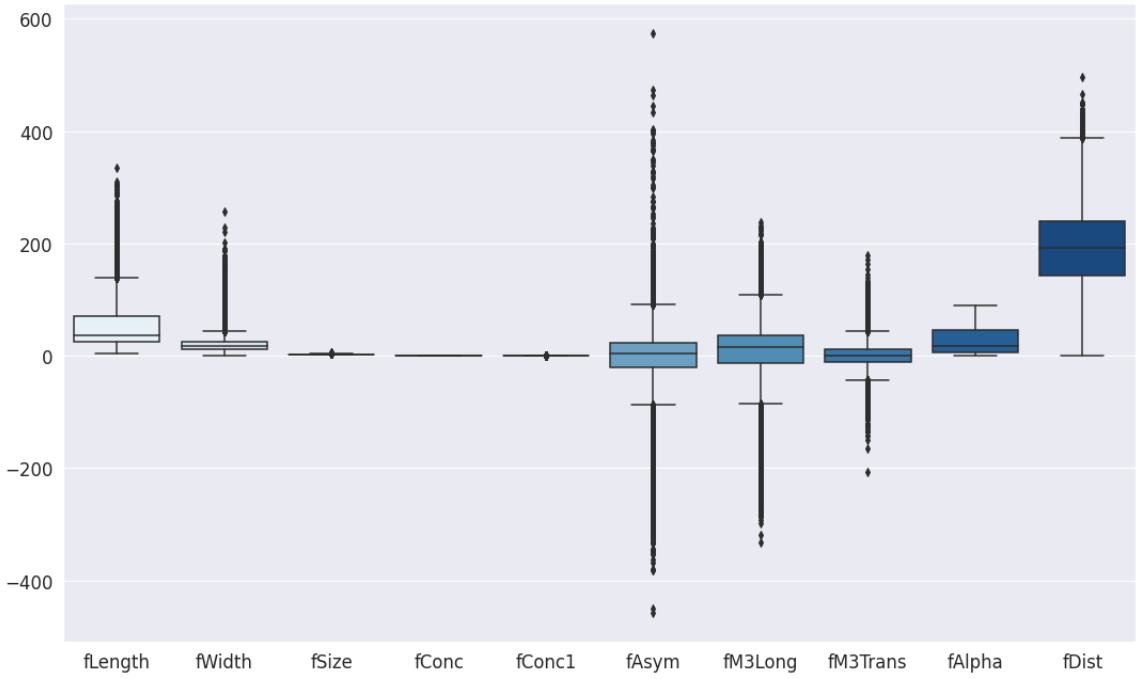


Figure 4: Boxplot of all the attributes

the kernel density estimate (KDE) figures. The KDE is a method for visualizing the distribution of observations in a dataset, analogous to a histogram but it represents the data using a continuous probability density curve in one or more dimensions. Almost all the features do not have a normal distributed KDE (fM3Long, fM3Trans, fAlpha, etc...). From the scatter plots, it is not possible to see any evident separation of classes. However, from the scatter plot *fConc-fConc1* it is possible to observe a certain degree of correlation between the two attributes, but we will discuss this matter further on.

2.5 Correlation

Correlation is a statistical relationship, whether causal or not, between two random variables. In statistics it normally refers to the degree to which a pair of variables are linearly related. The most common coefficient to measure correlation is the **Pearson product-moment correlation coefficient (PPMCC)**.

It is obtained by taking the ratio of the covariance of the two variables in question of our numerical dataset, normalized to the square root of their variances.

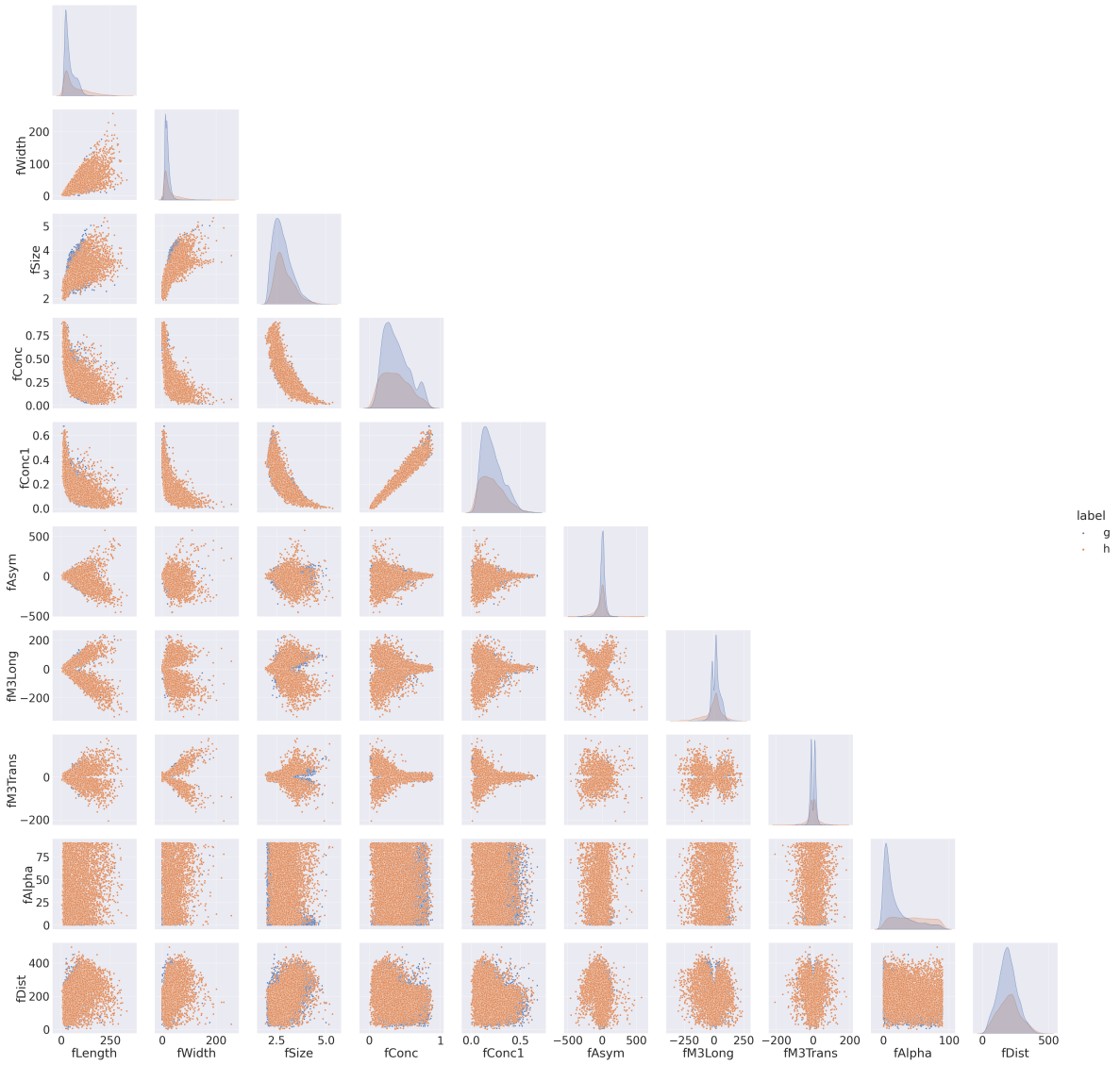


Figure 5: Pairplots for each couple of attributes where the points belonging to different classes are represented with different colours: gamma in blue and hadron in orange. The last diagonal also represent the distribution of the point with respect to that attribute.

$$\rho = \frac{cov(X,Y)}{\sigma_x \sigma_y} = \frac{E[(X-\mu_x)(Y-\mu_y)]}{\sigma_x \sigma_y} \tag{1}$$

A Pearson product-moment correlation coefficient attempts to establish a line of best fit through a dataset of two variables by essentially laying out the expected values and the resulting Pearson's correlation coefficient indicates how far away the actual dataset is from the expected values. Depending on the sign of our Pearson's correlation coefficient, we can end up with either a negative or positive correlation if there is any sort of relationship between the variables of our data set.

High correlation means that the two features are linearly correlated, that is, to be related in such a manner that their values form a straight line when plotted on a graph.In the figure is shown, as heatmap, the correlation between features. The most correlated features are *fConc* and *fConc1*. We

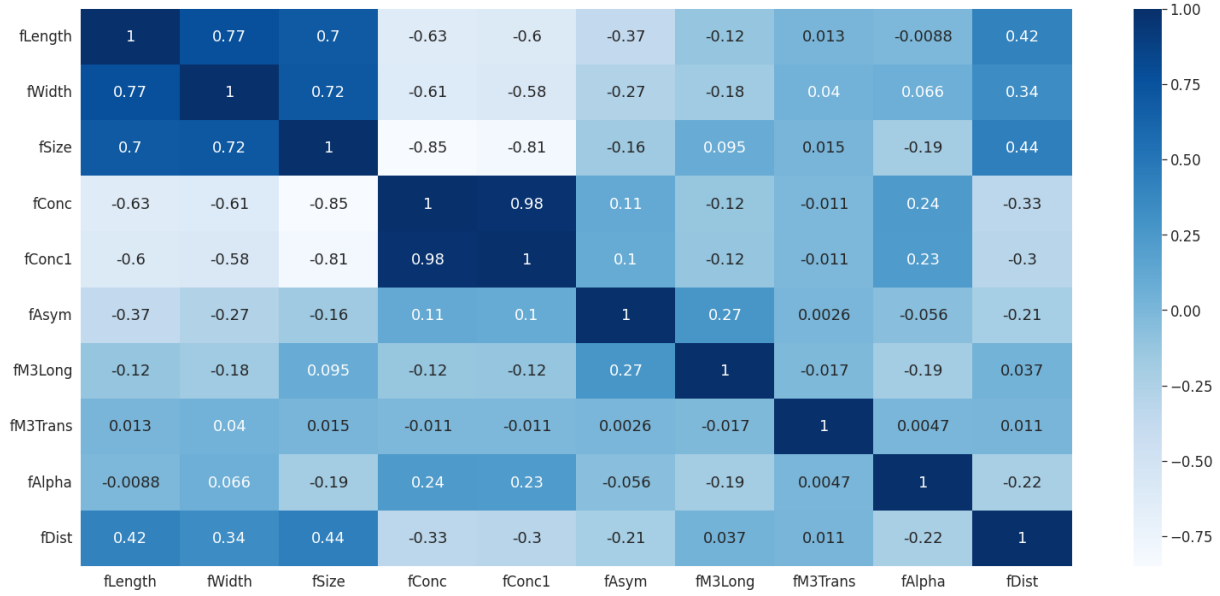


Figure 6: Heatmap representing the correlation coefficient between each pair of attributes

could reasonably expect this high value of correlation by their definitions. Hence can safely remove *fConc1*.

3 Data Preprocessing

3.1 Outliers Detection

Outliers are data points that differs significantly from other observations. These may be caused by experimental errors or by variability in the measurement. They may be object of study, but they could also cause problems during analysis. For this reason is important to locate and remove them. One way of completing this task is by using the **Local Outlier Factor(LOF)**. In anomaly detection, this technique is used to find anomalous data points by measuring the local deviation of a given data point with respect to its neighbors. It considers as outliers the samples that have a substantially lower density deviation than their neighbors.
To understand the algorithm we have to define some concepts:

- **k-distance of A** [$\text{dist}_k(A)$]: it represents the distance between its k^{th}nearest neighbor.
- **Neighborhood of A** [$N_k(A)$]: its the set of k elements within $\text{dist}_k(A)$.
- **Reachability distance from A to B** [$\text{reachdist}_k(A,B)$]: it is the true distance between A and B, if it is greater than $\text{dist}_k(B)$:

$$reachdist_k(A, B) = max(dist_k(B), dist(A, B)) \tag{2}$$

- **Local reachability distance of A** [$\text{lrd}_k(A)$]: it is the inverse of the average of the reachability distances of all the neighbors of A. We use the inverse in order to represent a density.

$$lrd_k(A) = \frac{||N_k(A)||}{\sum_{B \epsilon N_k(A)} reachdist_k(A, B)} \tag{3}$$

- **Local Outlier Factor** [LOF(A)]: it is the average of the local reachability distance values of all the neighbors of A divided by the local reachability distance of A.

$$LOF_k(A) = \frac{1}{||N_k(A)||} \frac{\sum_{B \epsilon N_k(A)} lrd_k(B)}{lrd_k(A)} \tag{4}$$

For the LOF a value of approximately 1 indicates that the object is similar to its neighbors. A value below 1 indicates a denser region , while values significantly larger than 1 indicate outliers. Hence, if the LOF of a point is higher than 1,such point may be removed from the dataset.
We apply the implementation of such algorithm provided by *sklearn*, obtaining a dataset composed of 18775 data points.

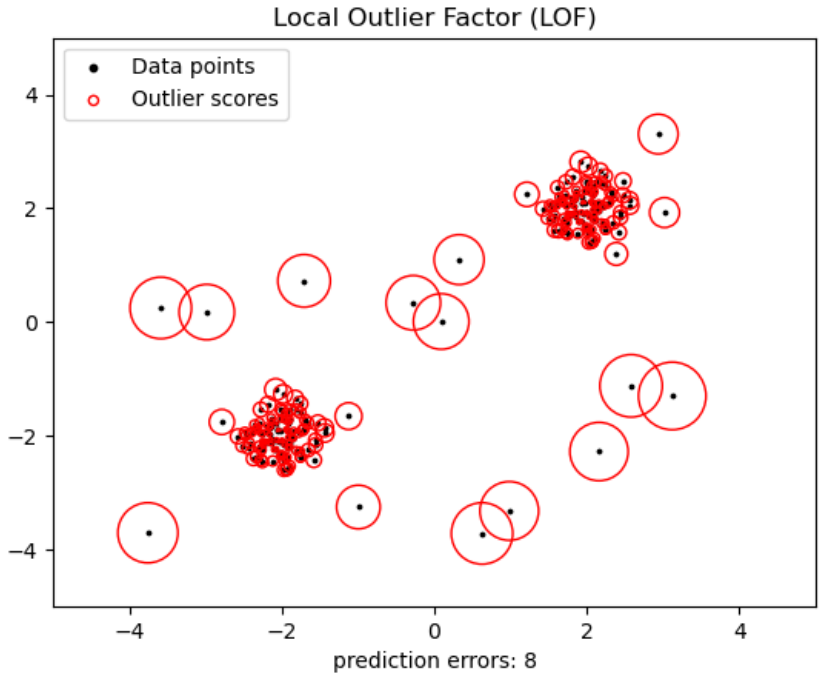


Figure 7: Example of LOF application provided by sklearn

3.2 Train-Test split

Before applying further preprocessing steps, it is important to divide the dataset into Train set and Test set. The idea in fact is to exploit the training set to learn the parameters for the preprocessing steps that will be applied later on (e.g. Inter Quartile Range) . When splitting the dataset in two sets, the idea is to randomly sample points from the original dataset. But in order to avoid generalization errors we want to preserve the distribution of the data in the dataset. Hence, we apply stratified sampling on the dataset with a 80/20 division.

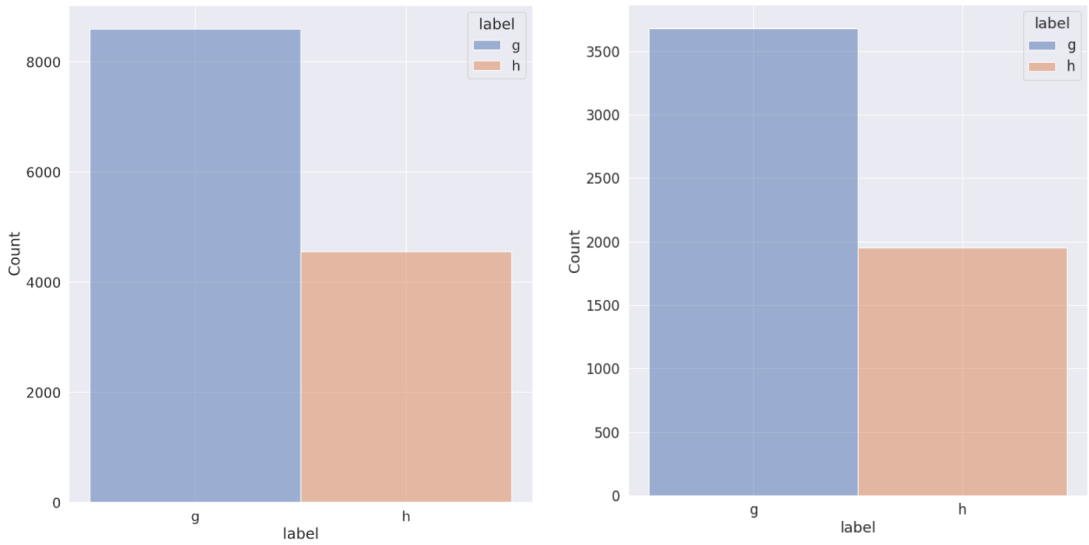


Figure 8: The distribution of the data points over the classes for both the training set and the test set. As we can see they both preserve the distribution of the original dataset.

3.3 Label Encoder

In order to work only with numerical features the first step is to encode target labels with a value between 0 and *nclasses-1*, since we have a binary problem the encoding will result:

- class g : 0
- cass h : 1

3.4 Robust Scaler

Scaling is used for making data points generalized so that the distance between them will be lower. Larger differences between the data points of input variables increase the uncertainty in the results of the model. If the values of the features are closer to each other there are chances for the algorithm to get trained well and faster instead of the data set where the data points or features values have high differences with each other will take more time to understand the data and the accuracy will be lower. The **RobustScaler** removes the median and scales the data according to the Interquartile Range(IQR). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). These are calculated on the Training set and then used also to scale the Test set.

3.5 Dimensionality reduction : PCA

Dimensionality reduction is the process of taking data in a high dimensional space and mapping it into a new space whose dimensionality is much smaller. There are different reason to apply this procedure to a dataset:

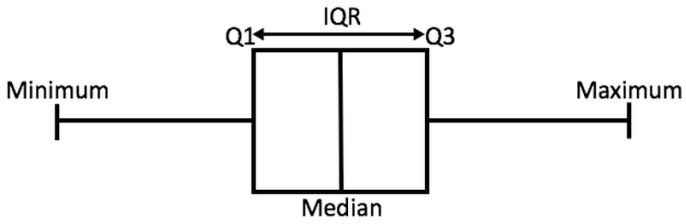


Figure 9: Interquartile Range defined over a box plot

- high dimensionality could lead to poor generalization of the learning algorithm.
- high dimensional data could lead to computational problems.
- dimensionality reduction can be used for interpretability of data and for finding meaningful structure of the data.

The method exploited in this analysis is called **Principal Component Analysis** (PCA). In PCA, both the compression and the recovery are performed by linear transformations and the method finds the linear transformations for which the differences between the recovered vectors and the original vectors are minimal in the least squared sense. Namely, let $x_1, ..., x_m$ be m vectors in R^d and a matrix $W \in R^{n,d}$. W induces a mapping x $\mapsto Wx \in R^n$ where $Wx \in R^n$ is the lower dimensionality representation of x. Then, a second matrix $U \in R^{d,n}$ can be used to recover each original vector x from its compressed version. In PCA, we find the compression matrix W and the recovering matrix U so that the total square distance between the original and recovered vectors is minimal; namely, the problem to solve is:

$$\arg\min_{(W \in R^{n,d}, U \in R^{d,n})} \sum_{i=1}^{m} ||x_i - UWx_i||^2 \tag{5}$$

Let (U,W) to be a solution of the previous formula. Then the columns of U are orthonormal and $W = U^T$.
Basing of this lemma, the problem can be rewritten as:

$$\arg\min_{(W \in R^{n,d}, U \in R^{d,n})} \sum_{i=1}^{m} ||x_i - UU^T x_i||^2 \tag{6}$$

For every $x \in R^d$ and a matrix $U \in R^{d,n}$ such that $UU^T = I$, we have that:

$$||x_i - UU^T x_i||^2 = ||x||^2 - trace(U^T x x^T U) \tag{7}$$

Hence, the problem becoms:

$$\arg\max_{(W \in U^{d,n}: U^T U = I)} trace(U^T \sum_{i=1}^{m} x_i x_i^T U) \tag{8}$$

Let $A = \sum_{i=1}^{m} x_i x_i^T$. Since this matrix is symmetric, we can apply the spectral decomposition, such that $A = VDV^T$, where D is the diagonal matrix and $V^TV = VV^T = 1$. Hence, the element of D are the eigenvalues of A and the columns of V the eigenvectors. We assume that $D_{1,1} \geq D_{2,2} \geq ... \geq D_{d,d}$, but since A is positive semidefinite it also holds that $D_{d,d} \geq 0$. So, let $x_1, ..., x_m$ be arbitrary vectors in R^d, $A = \sum_{i=1}^{m} x_i x_i^T$ and $u_1, .., u_n$ be the eigenvectors of A corresponding to the largest n eigenvalues. The solution to the PCA is to set U to be the matrix whose columns are $u_1, ..., u_n$ and $W = U^T$.

We deploy the PCA implementation provided by *sklearn*, then we exploit this model to find out the number of principal components. We opt for a visual approach (showed in the figure below). In the plot the X-axis *number of components* against the proportion of the cumulative variance explained on the Y-axis. The plot shows that by choosing 6 components, we can explain more than the 90% of the variance, so that it could be a good measure.

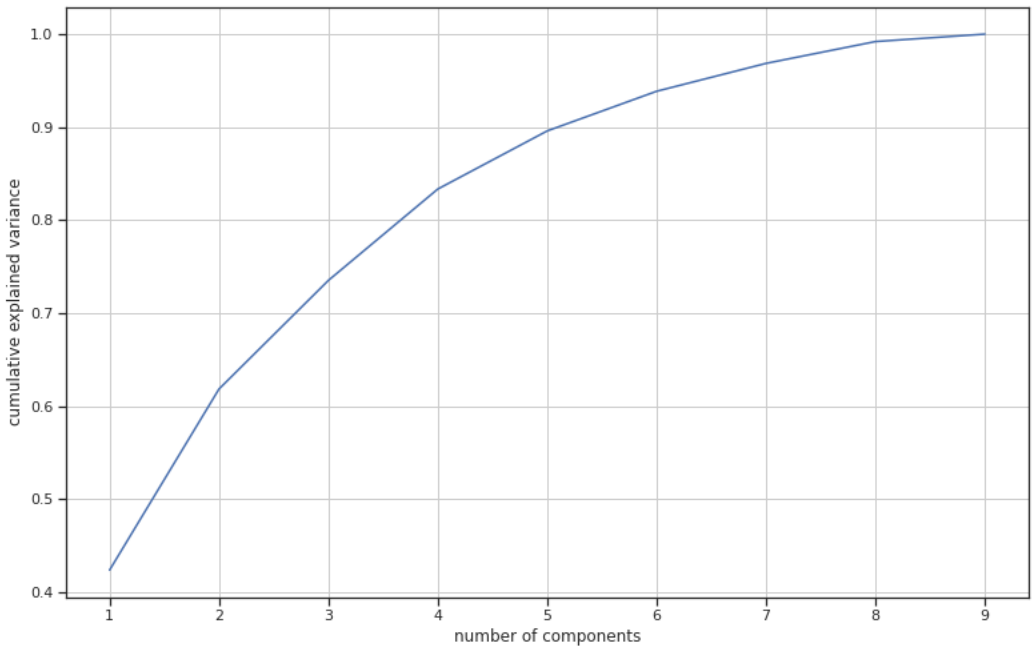


Figure 10: Cumulative variance for each number of components

4 Methods

4.1 Confusion Matrix

After the prediction we have four different possibilities for each event:

- **True Positive TP** events labelled as True and correctly predicted as True.
- **True Negative TN** events labelled as False and correctly predicted as False.
- **False Positive FP** events labelled as False and wrongly predicted as True.
- **False Negative** events labelled as True and wrongly predicted as False.

The **Confusion Matrix** is one of the most common ways to represent these values,in each row there are instances of the predicted class, and in each column there are the instances of the actual classes. The diagonal values are the ones correctly classified, while the others are the misclassified values. From these scheme we can also calculate metrics.

4.2 Metrics

In order to evaluate our models we need to define metrics. In binary classification, one of the most common metrics is *Accuracy* which provides the ratio of the number of correctly classified samples over the total number classified sample. In order to provide a more comprehensive analysis we are also going to use the *Recall* and *F1 score* measures. In terms of binary classification can be described as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{12}$$

4.2.1 ROC curve

A **receiver operating characteristic curve**, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

$$TPR = \frac{TP}{TP + FN} \tag{13}$$

$$FPR = \frac{FP}{TN + FP} \tag{14}$$

Through this curve we can obtain a measure that is most often used for model comparison, the **Area under the curve**(AUC). As hinted by the name this measure is obtained by calculating the area under the curve plotted in the graph.

4.3 Over Sampling: SMOTE

As previously shown from the Data Visualization paragraph, the dataset is unbalanced having: 12332 **gamma** records where and 6636 **hadron** records. An unbalanced dataset presents many problems for the analysis as our model could not be able to predict correctly the records representing the minority class. To overcome this problem there are two main techniques: Oversampling and undersampling. Because the undersampling the majority class would undermine the main objective of classifying the gamma signal, in this analysis we will use oversampling of the minority class by exploiting **SMOTE** (Synthetic Minority Oversampling Technique).
The basic idea is to selects a minority class instance a at random and finds its k nearest minority

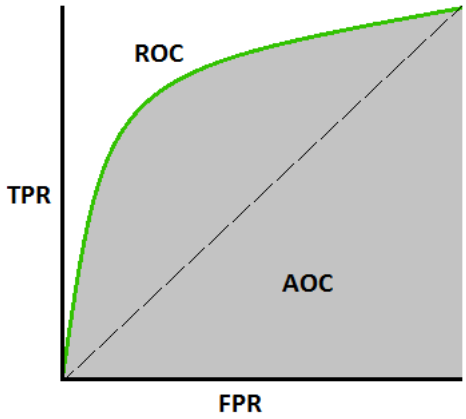


Figure 11: example of ROC curve and the AUC

class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b.

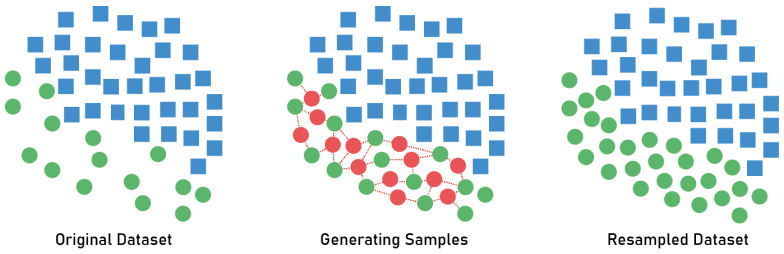


Figure 12: example of point generation with SMOTE

In the end the idea is to exploit this technique to have an even number of records for both classes. As we can see from the figures, representing the class distribution before and after the application of SMOTE algorithm.

4.4 Cross validation

In order to provide an estimate of the true error, we use resampling techniques during the training. Previously the dataset was already split into traing set and test set while preserving the distribution of the dataset in the test set. In order to provide a further improvement this step needs to be carried out also during the training. Hence we introduce the **Stratified k-fold cross validation**. This technique is used for training and also to find the best parameters for a model. It splits the training set into k sets, *k-1* of them are used as training set and the other one is used as *validation set*. This process is repeated k times by changing each time the validation set taken into account.
In order to preserve the distribution for the validation set, the idea is to apply SMOTE during each iteration. This provides the over sampling of the minority class while also preserving the distribution of the validation set during the whole training phase.
Furthermore, the CV is also used to obtain the best hyperparameters for the models. In fact during the hyperparameter tuning, the hyper-parameter values (taken from a selected list) giving the highest

CV score are chosen for the final model, whose final evaluation is then assessed on the test set. This overall procedure is called Grid Search Cross Validation.

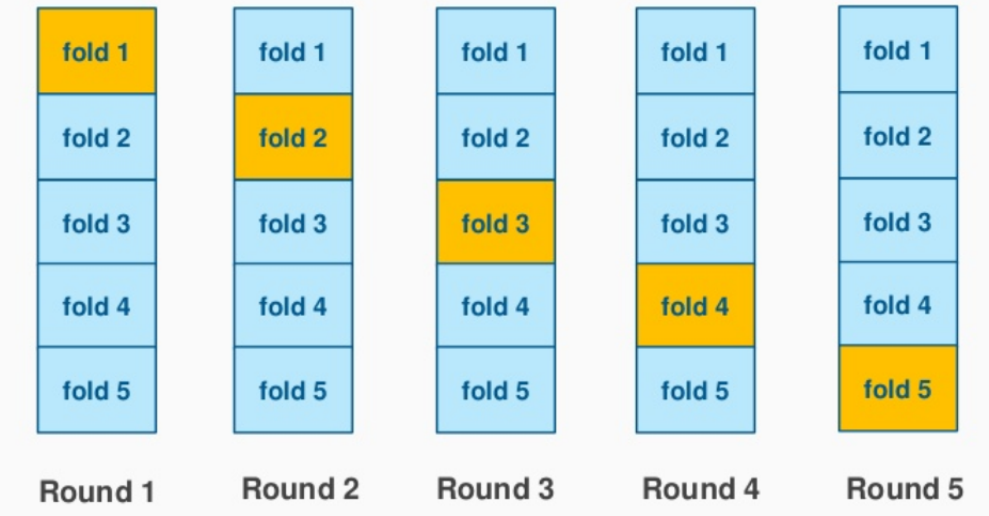


Figure 13: Example of Cross Validation algorithm

4.5 Empirical Risk Minimization

Given:

- a set of object $\mathcal{X}$
- a set of labels $\mathcal{Y}$
- an unknown distribution $\mathcal{D}$ from which objects are sampled
- a labelling function $f(\cdot) : \mathcal{X} \mapsto \mathcal{Y}$
- a finite sequence of pairs composed by objects and labels $S = ((x_1, y_1), ..., (x_m, y_m))$
- a predictor rule $h : \mathcal{X} \mapsto \mathcal{Y}$

We can define the true error of a prediction rule as:

$$L_{\mathcal{D},f}(h) = P_{x\sim\mathcal{D}}[h(x) \neq f(x)] = \mathcal{D}(\{x : h(x) \neq f(x)\}) \tag{15}$$

The error of h is the probability of randomly choosing an example x for which $h(x) \neq f(x)$. However,since we do not know neither D nor $f(\cdot)$, we cannot use this formula. Hence ,since the learner can use only S, it makes sens to search for a solution working on the training set. We can define the training error:

$$L_s(h) = \frac{|\{i \in [m] : h(x_i) \neq y_i\}|}{m} \tag{16}$$

The goal of the learning paradigm called *Empirical Risk Minimization* (ERM) is to find a prediction rule h that minimizes the error $L_s(h)$.

4.6 Bias-Complexity trade off

The training data can mislead the learner and result in overfitting. If we restrict the search space to some hypothesis class $\mathcal{H}$ we might overcome this problem. The hypothesis class can be viewed as reflecting some prior knowledge about the task. The reason why there is a need for prior knowledge is the **No Free Lunch Theorem** (NFL).

The NFL theorem states that given:

- a learning algorithm A for the task of binary classification with respect to the 0-1 loss over a domain $\mathcal{X}$
- m be any number smaller than $|\mathcal{X}|/2$ representing a training set size

Then, there exist a distribution $\mathcal{D}$ over $\mathcal{X}$ x {0,1} such that:

1. There exist a function $f : \mathcal{X} \mapsto \{0, 1\}$ with $L_{\mathcal{D}}(f) = 0$.
2. With probability of at least 1/7 over the choice of $S \sim \mathcal{D}^m$ we have that $L_{\mathcal{D}}(A(S)) \geq 1/8$.

This theorem asserts that for every learner, there exists at least one task on which it fails, even if there exists another learner f which can successfully solve that task. Therefore to avoid the distribution that will cause us to fail when learning a specific task, we exploit the prior knowledge. The latter can be expressed by restricting our hypothesis class. The problem rises during the choice of a good hypothesis class. Because cannot choose the class of all the functions over the given domain, but we want to ensure that the chosen class includes the hypothesis with no error at all or that the smallest achievable is rather small. This is called **Bias-Complexity tradeoff**.

The error of an $ERM_{\mathcal{H}}$ can be decomposed in to two components as follows:

$$L_{\mathcal{D}}(h_S) = \epsilon_{app} + \epsilon_{est} \text{ where: } \epsilon_{app} = \min : h \in \mathcal{H} L_{\mathcal{D}}(h),\ \epsilon_{est} = L_{\mathcal{D}}(h_S) - \epsilon_{app}.$$

The two components can be defined as:

- The **Approximation error**-ϵ_{app}: it represents the minimum risk achievable by a predictor in the hypothesis class. It does not depend on the sample size. This term measures how much risk we have because we restrict ourselves to a specific class, namely, how much *inductive bias*. Enlarging $\mathcal{H}$ can reduce the approximation error.
- The **Estimation error**ϵ_{est}: it is the difference between the approximation error and the error achieved by h_S. The estimation error results because the empirical risk (i.e., training error) is only an estimate of the true risk, and so the predictor minimizing the empirical risk is only an estimate of the predictor minimizing the true risk. It depends on both the **complexity** of the hypothesis class and also on the trainig set size. Namely it increases with $|\mathcal{H}|$ and decreases with m.

Since our goal is to minimize the total risk, we face a tradeoff. Choosing $\mathcal{H}$ to be a very rich class decreases the approximation error but at the same time might increase the estimation error, as a rich $\mathcal{H}$ might lead to overfitting. On the other hand, choosing $\mathcal{H}$ to be a very small set reduces the estimation error but might increase the approximation error or, in other words, might lead to underfitting. The idea is that the prior knowledge of the specific problem enables us to design hypothesis classes for which both the approximation error and the estimation error are not too large.

5 Models

5.1 Logistic Regression

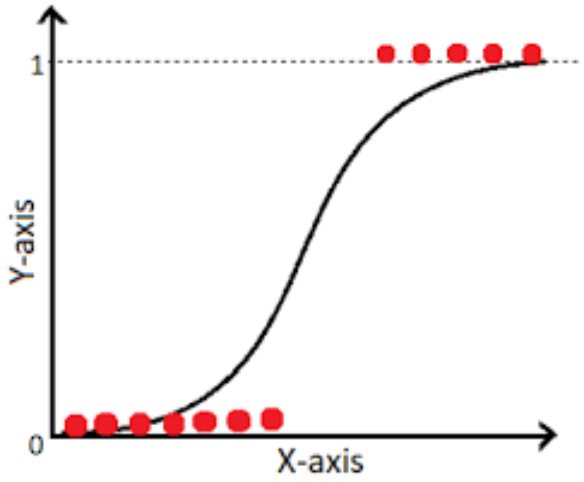


Figure 14: example of Logistic regression curve

In logistic regression we learn a family of functions h from R^d to the interval [0,1]. However, the logistic regression is commonly used for classification tasks. In fact we can think as the outcome *h(x)* as the **probability** of x is 1.

To perform the classification, the algorithm learns from a training set, a vector of weights and a bias term. Each weight w_i is a real number associated with one input feature x_i. The weight w_i represents how important that input feature is to the classification decision. The bias term β, also called *intercept*, is a real number added to the weighted inputs. To make a decision on a test instance, after we have learned the weights in the training phase, the classifier first multiplies each x_i by its weight w_i, sums up the weighted features and adds the bias term β. The resulting single number z expresses the weighted sum of the evidence for the class.

$$z = \sum_{i=1}^{n}(w_i x_i) + \beta \tag{17}$$

Since z can assume positive or negative values, the algorithm computes $\phi_{sig}(z)$, to get a probability (between 0 and 1). So, the final formulation is the one that follows:

$$P[Y = 1] = \phi(wX + \beta) = \frac{1}{1 + e^{-(wX+\beta)}} \tag{18}$$

$$P[Y = 0] = 1 - P[Y = 1] \tag{19}$$

To transform this value in prediction, we need to define a **decision boundary** to make Logistic Regression becoming a classification algorithm, following the formula below.

$$\hat{Y} = \begin{cases} 1 \; if \; P[Y = 1|X] \geq 0.5 \\ 0 \; if \; P[Y = 1|X] < 0.5 \end{cases} \tag{20}$$

In the end, we need to specify a loss function. We should define how bad it is to predict some $h_w(x) \in [0, 1]$ given that the true label is $y \in \{\pm 1\}$. Clearly, we would like that $h_w(x)$ would be large if $y = 1$ and that $1 - h_w(x)$ would be large if $y = -1$. The logistic loss function used in logistic regression penalizes h_w based on the log of $1 + e^{-y(wX+\beta)}$ that is:

$$\ell(h_w(x, y)) = log(1 + e^{-y(wX+\beta)}) \tag{21}$$

5.1.1 Parameters

The following parameters are used for the Grid search cross validation:

penalty	["l1","l2"]
solver	"liblinear"
C	[0.6, 0.8, 1.0, 1.2, 1.5]

5.2 Random Forest

Random Forest is a classifier based on an ensemble of **Decision tree**. The main idea used is *bootstrap aggregation sampling*, or also called bagging, which combine multiple prediction functions learned from n different datasets. These datasets are bootstrapped from the training set, bootstrap is a sampling technique that consist in sampling uniformly the samples from the training set at random with replacement. But firstly, we need to understand how decision trees work.

5.2.1 Decision Tree

A decision tree is a predictor, $h : \mathcal{X} \mapsto \mathcal{Y}$, that predicts the label associated with an instance x by traveling from a root node of a tree to a leaf. At each node on the root-to-leaf path, the successor child is chosen on the basis of a splitting of the input space. Usually, the splitting is based on one of the features of x or on a predefined set of splitting rules. The algorithm at each step chooses the best predictor to split according to some criteria. The most commons ones are:

- The **information gain**, which is the difference between the entropy before and after the splitting.

$$Entropy = \sum_{i=1}^{K} -p_i * \log_2 p_i \tag{22}$$

- The **Gini index**, which is a measure of the total variance across the N classes. It takes small values when $\hat{p}_{mk}$'s are close to zero or one.

$$Gini = \sum_{i=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{23}$$

Both these measures are a level of impurity of the node, so the lower they are, the better it is.

Among all algorithms, decision trees have several advantages. First, they are simple to understand and interpret, since they do not require statistical knowledge; second, compared to other techniques, take less effort for data preparation and lastly, require less data cleaning.

Concerning the disadvantages, decision trees are largely unstable compared to other decision predictors: a small change in the data can result in a major change in the structure of the tree, which can convey a different result.

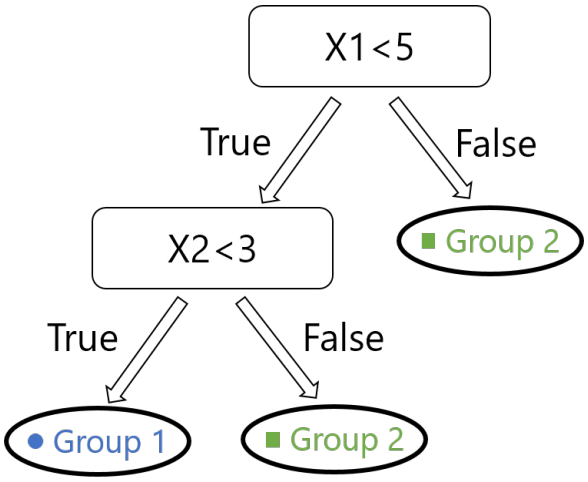


Figure 15: example of Decision tree

5.2.2 Bootstrap aggregation

Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the variance of a statistical learning method. Given a set of n independent observations $Z_1, Z_2, ..., Z_n$ each with σ^2, the variance of the mean $\bar{Z}$ of the observations is given by $\frac{\sigma^2}{n}$, which means that averaging a set of observations reduces the variance. Hence, it would be useful to have access to multiple training set. In order to do so, we can bootstrap, by taking repeated samples from the (single) training data set. The idea is to train a model for each bootstrapped training data sets and then average all the predictions to obtain the result, and this is called bagging.

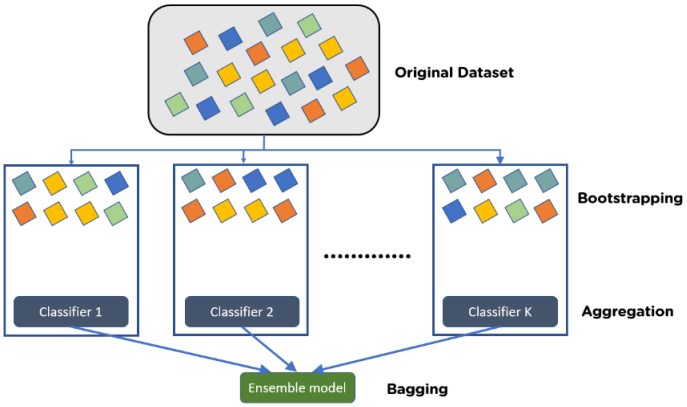


Figure 16: Bootstrap aggregation

5.2.3 Random Forest

When building these decision trees, if there exists a feature that provides a good split, it will be selected by all the decision trees, ending up with an highly correlated predictions. Random forests not only apply bagging, but to solve the aforementioned problem, they apply a technique to decorrelate the trees. So the idea is to build different decision trees on the bootstrapped training samples, but when building these decision trees, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of predictors.
These models provide an improvement in terms of preventing overfitting and provides more stability but less interpretability due to the high number of trees.

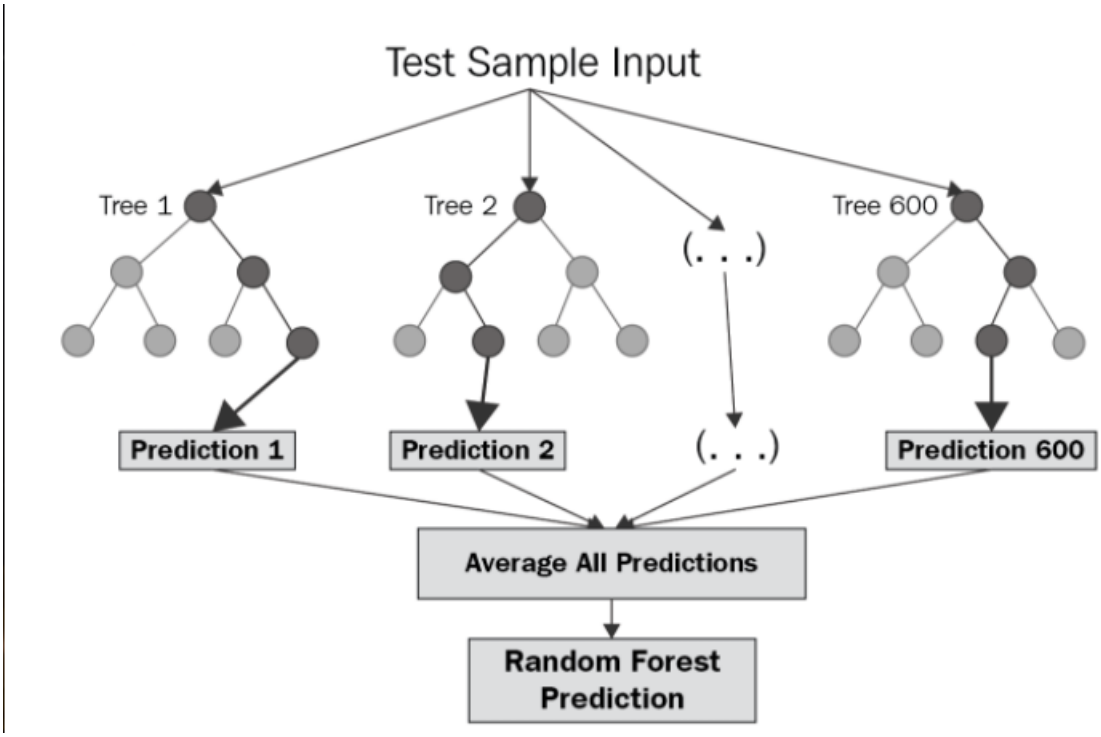


Figure 17: example of Randon Forest

5.2.4 Parameters

The following parameters are used for the Grid search cross validation:

n_estimators	[50, 100, 150, 200]
criterion	["gini", "entropy"]
max_depth	[10,20,50]

5.3 SVM

The support vector machine paradigm (SVM) is a very useful machine learning tool for learning linear predictors in high dimensional feature spaces. The SVM algorithmic paradigm tackles the sample complexity challenge by searching for large margins separators. Let S = $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$ be a training set of examples, where each $\mathbf{x}_i \in R^d$ and $y_i \in (\pm 1)$. We say that this training set is linearly

separable, if there exists a halfspace, $(\mathbf{w}, b)$, such that $y_i = sign(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$ for all i. Alternatively, this condition can be rewritten as:

$$\forall i \in [m], y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$$

.

The objective of **Hard-SVM** algorithm is to find the hyperplane whose margin with respect to the training set is as minimum as possible. Defining the distance between a point $\mathbf{x}$ and the hyperplane $\langle \mathbf{w}, \mathbf{x} \rangle$ as $|(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)|$, the closest point in the training set to the hyperplane is:

$$\min_{i \in [m]} (\langle \mathbf{w}, \mathbf{x}_i \rangle + b)$$

Therefore, the algorithm rule is:

$$\arg\max_{(\mathbf{w},b):||w||=1} \min_{i \in [m]} (\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \quad s.t \quad \forall i \in [m], y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0$$

This formula can be rewritten as a quadratic optimization, considering the maximization of the margin of the hyperplane.

$$(\mathbf{w}_0, b_0) = \arg\min_{(\mathbf{w},b)} ||\mathbf{w}||^2 \quad s.t. \ \forall i, y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$$

The final output of the algorithm is indeed the **separating hyperplane with the largest margin**.

The Hard-SVM formulation assumes that the training set is linearly separable, which is a really strong assumption. **Soft-SVM** can be viewed as a relaxation of the Hard-SVM that can be applied if the training data is not linearly separable. This algorithm enforces a relaxation of hard constraints introducing slack variables $\xi_1, ...\xi_m$, which measures how much the constraint $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$ is being violated by $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i$. Soft-SVM jointly minimize the norm of $\mathbf{w}$ and the average of ξ_i.

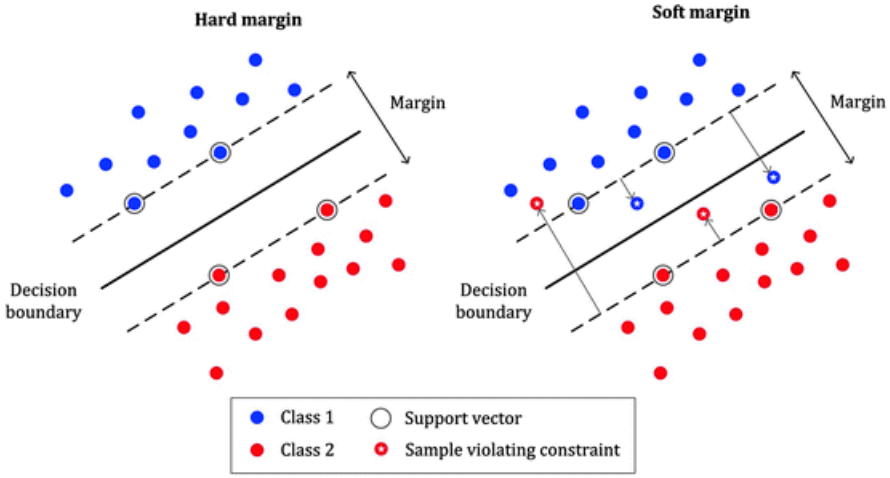


Figure 18: examples of Hard-SVM and Soft-SVM

5.3.1 Kernel trick

Kernel based algorithms and in particular kernel-SVM are very useful and popular tools. To make the class of halspaces more expressive, we can first map the original instance space into another space and then learn a halfspace in that space. The basic paradigm is the one that follows.

1. Given a domain set $\mathcal{X}$ and a learning task, choose a mapping $\psi : \mathcal{X} \to \mathcal{F}$, for some feature space $\mathcal{F}$.
2. Given a sequence of labeled examples, S = $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$ create the image sequence $\hat{S} = (\psi(\mathbf{x}_1), y_1), ..., (\psi(\mathbf{x}_m), y_m)$.
3. Train a linear predictor h over $\hat{S}$.
4. Predict the label of a test point, $\mathbf{x}$, to be $h(\psi(\mathbf{x}))$.

The success of this paradigm consists in choosing a good ψ for a given learning task: that is, a ψ that will make the image of data distribution linearly separable in the feature space, thus making the resulting algorithm a good learner for a given task. Using a nonlinear mapping ψ, we can enrich the class of halfspaces, but since this solution can be too heavy in terms of computational requirements, we can adopt as solution the **kernel based learning**.
Given an embedding ψ of some domain space $\mathcal{X}$ into some Hilbert space, we define the Kernel function $K(\mathbf{x}, \mathbf{x'}) = \langle \psi(\mathbf{x}), \psi(\mathbf{x'}) \rangle$. We can think $\mathcal{K}$ as specifying similarity between instances and of the embedding ψ as mapping the domain set $\mathcal{X}$ into a space where similarities are realized as inner products. This means that these algorithms can be carried out just on the basis of the values of the kernel function over pairs of domain points. The main advantage of such algorithms is that they implement linear separators in high dimensional feature spaces without having to specify points in that space or expressing ψ explicitly.

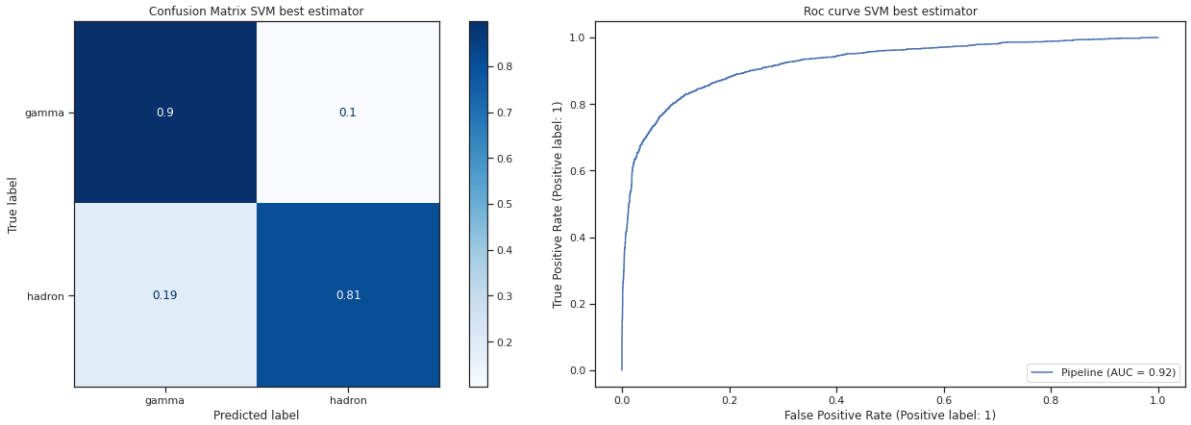
5.3.2 Parameters

The following parameters are used for the Grid search cross validation:

gamma	["auto","scale"]
kernel	["linear","rbf","poly"]
C	[0.6, 0.8, 1.0, 1.2, 1.5]

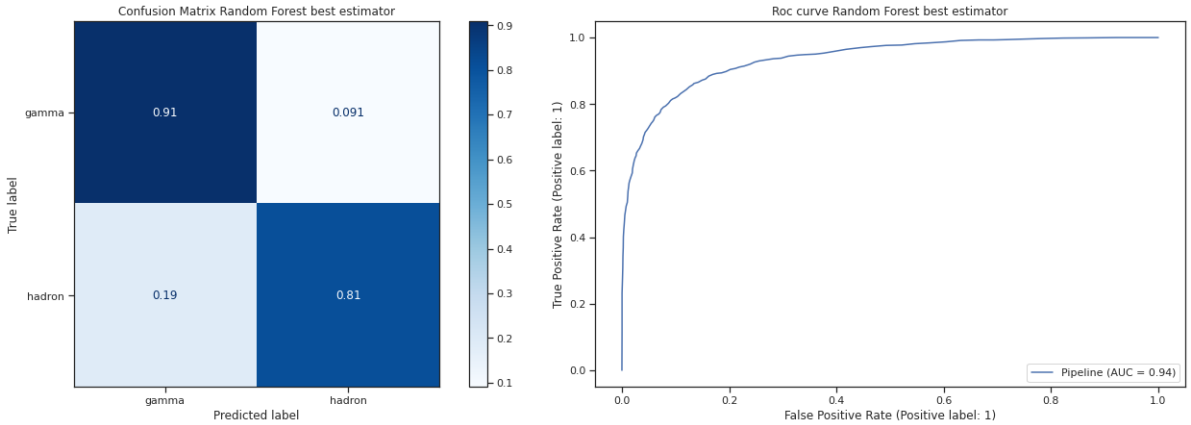
6 Training and results

In these section are presented the selected parameters, for each model, that produce the better performances. In addition, the confusion matrix and the RoC are also reported.
From the analysis we can see that using the PCA provides lower performances when compared to the respective model, this is understandable since the with n=6 components we explain approximately 94% of the variance, also the number of features is not significantly high to provide an improvement. We can also see that the best results are provided from the **Random Forest**, while the **Logistic Regression** fails when compared to the other two models.



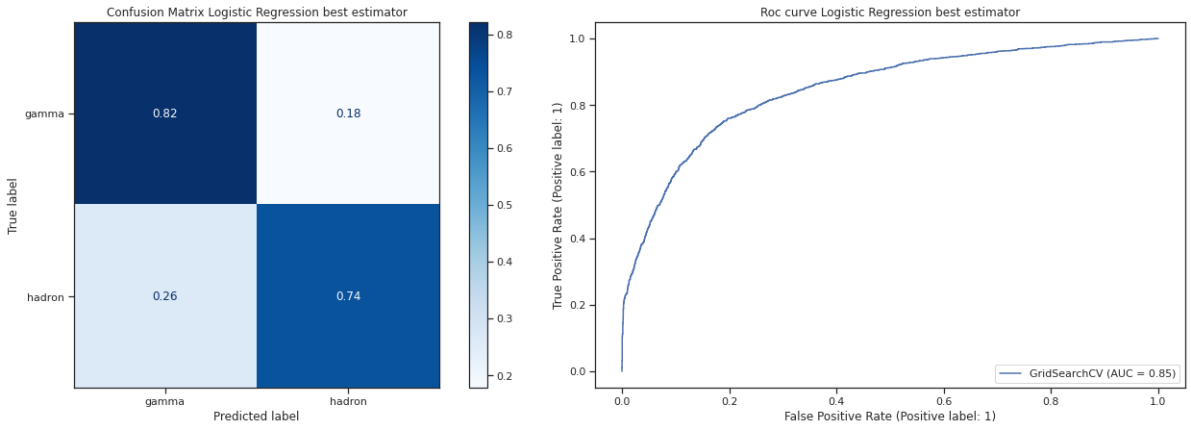
(a) Confusion Matrix (b) ROC curve

Figure 19: Confusion Matrix and ROC curve for SVM



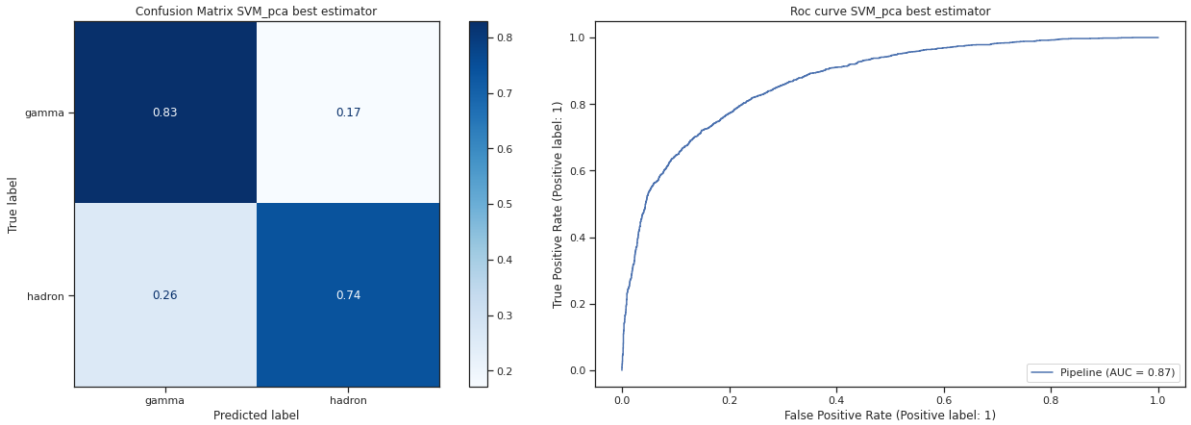
(a) Confusion Matrix (b) ROC curve

Figure 20: Confusion Matrix and ROC curve for Random Forest



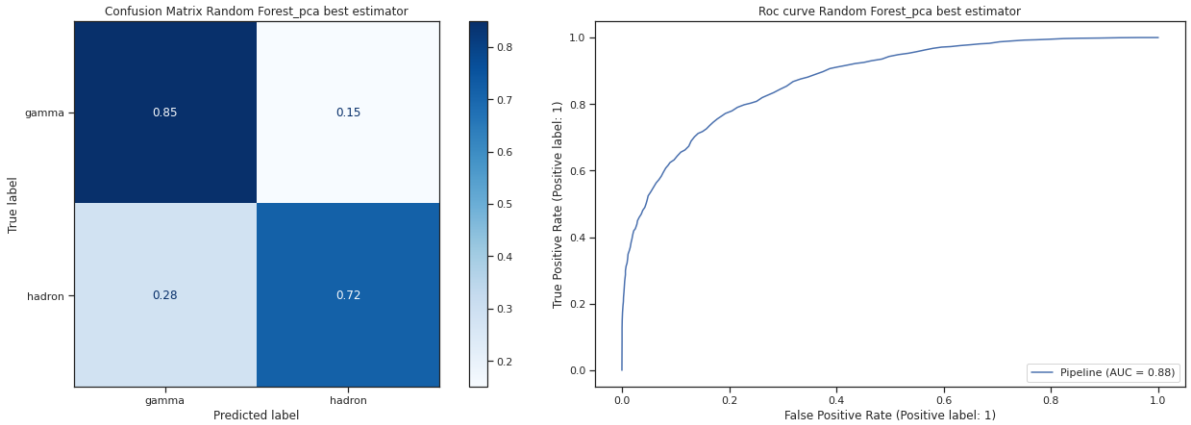
(a) Confusion Matrix (b) ROC curve

Figure 21: Confusion Matrix and ROC curve for Logistic Regression



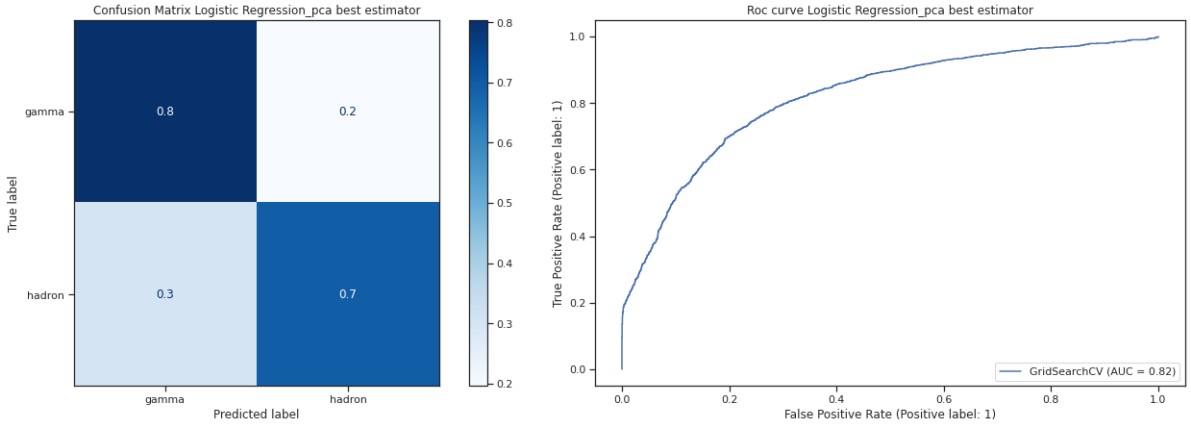
(a) Confusion Matrix (b) ROC curve

Figure 22: Confusion Matrix and ROC curve for SVM with PCA



(a) Confusion Matrix (b) ROC curve

Figure 23: Confusion Matrix and ROC curve for Random Forest with PCA



(a) Confusion Matrix (b) ROC curve

Figure 24: Confusion Matrix and ROC curve for Logistic Regression with PCA

Model	Pameters selected	AUC
SVM	C : 1.5 gamma : 'auto' kernel : 'rbf'	0.92
Random Forest	criterion : 'gini' max_depth : 50 n_estimators : 100	0.94
Logistic Regression	C : 1 penalty : 'l2' solver : 'liblinear'	0.85
SVM with PCA	C : 1.0 gamma : 'auto' kernel : rbf	0.87
Random forest with PCA	criterion : 'entropy' max_depth : 50 n_estimators : 100	0.88
Logistic Regression with PCA	C : 1.2 penalty : 'l2' solver : 'liblinear'	0.82