Politecnico di Torino

Mathematics in Machine Learning

Astronomy and Machine Learning: a case study using images from a Cherenkov gamma-ray telescope

Student name: Riccardo Prestigiacomo
email: s283392@studenti.polito.it
Professors: Francesco Vaccarino, Mauro Gasparini
Academic year: 2020-2021

Contents

1 Introduction **2**

2 Data exploration **2**
2.1 Dataset . . . 2
2.2 Dataset information . . . 2
2.3 Dataset distribution . . . 3
2.4 Statistical overview . . . 3
2.5 Correlation . . . 5
2.6 Pairplot . . . 6
2.7 Conclusions . . . 7

3 Data preprocessing **7**
3.1 Outliers detection . . . 7
3.2 Label encoder . . . 7
3.3 Standard scaler . . . 7
3.4 Dimensionality reduction . . . 7
3.4.1 PCA . . . 8

4 Setting and Metrics **8**
4.1 Oversampling . . . 8
4.1.1 SMOTE . . . 9
4.2 Stratified K fold . . . 9
4.3 The bias-variance trade off . . . 10
4.4 Confusion Matrix . . . 10
4.5 ROC curve . . . 11

5 Algorithms **13**
5.1 Logistic Regression . . . 13
5.2 Decision Tree . . . 13
5.3 Random Forest . . . 14
5.4 SVM . . . 15
5.4.1 Hard-SVM . . . 15
5.4.2 Soft-SVM . . . 16
5.4.3 Non-Linear SVM: Rbf Kernel . . . 16

6 Training and results **16**
6.1 Logistic Regression . . . 16
6.2 Decision Tree . . . 17
6.3 Random Forest . . . 18
6.4 SVM . . . 19
6.4.1 Linear SVM . . . 19
6.4.2 Kernel SVM . . . 19

7 Conclusions **20**

1 Introduction

Astronomy, astrophysics, and particle physics all have made rapid progress as observational sciences in recent years. Much of this progress is due to the development of detector technology, coupled with a parallel development of analysis methods. Confronted with a daunting challenge of extracting a small number of interesting events from an overwhelming sea of background, both having very similar characteristics, physicists have become familiar with quite sophisticated multivariate techniques.

Figure 1: Ground-based gamma-ray telescope

Ground-based gamma-ray telescopes are an example of experiments exploring a new research frontier, that are likely to benefit from the many multivariate data analysis techniques developed in recent years. In this case study, various multivariate techniques are applied to the same set of data.
Ground-based atmospheric Cherenkov telescopes using the imaging techniques are a comparatively recent addition to the panopoly of instruments used by astrophysicists. They observe high-energy gamma rays, taking advantage of the radiation emitted by charged particles as they are produced abundantly inside the electromagnetic showers initiated by the gammas, and developing in the atmosphere. This Cherenkov radiation leaks through the atmosphere and gets recorded in the detector, allowing reconstruction of shower paramters. One problem is that the number of observable Cherenkov photons for primary gammas of lower energy (below 100 GeV) becomes comparatively small, and correspondingly the problems of discrimination against background get enhanced. Optimal use of this information is critical for the success of this technique.

2 Data exploration

2.1 Dataset

For our case study, we use the 'MAGIC Gamma Telescope Data Set'. This data set contains generated data to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using the imaging technique.
The data set was generated by a Monte Carlo program, Corsika, described in: D.Heck et al., CORSIKA, A Monte Carlo code to simulate extensive air showers, Forschungszentrum Karlsruhe FZKA 6019 (1998).
The task is to *discriminate stastically images generated by primary gammas* (signal, class label g) *from the images of hadronic showers* initiated by cosmic rays in the upper atmosphere (background, class label h).
In the work presented we perform a general study, applying different multivariate classification methods to events described by 10 chosen image parameters.

2.2 Dataset information

Let's start by showing the dataset's structure along with its variables.

We have 19020 samples, whose no value is missing. In each row we have a numerical variable, and last we have a target binary variable: as this is a binary classification problem.
Each event is characterized by the following **ten parameters**:

1 *fLength*: continuous major axis of ellipse [mm]

```
RangeIndex: 19020 entries, 0 to 19019
Data columns (total 11 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   fLength   19020 non-null  float64
 1   fWidth    19020 non-null  float64
 2   fSize     19020 non-null  float64
 3   fConc     19020 non-null  float64
 4   fConc1    19020 non-null  float64
 5   fAsym     19020 non-null  float64
 6   fM3Long   19020 non-null  float64
 7   fM3Trans  19020 non-null  float64
 8   fAlpha    19020 non-null  float64
 9   fDist     19020 non-null  float64
 10  class     19020 non-null  object
dtypes: float64(10), object(1)
```

Figure 2: dataset info

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]

11 *class*: g,h gamma (signal), hadron (background)

The data consist of **two classes**: **gammas (signal)** and **hadrons (background)**.

2.3 Dataset distribution

Events were generated at shower energies from 10 GeV up to about 30 TeV, and for zenith angles from zero to 20 degrees. The samples used by all methods are identical, and consist of **12332 gamma events** and **6888 hadron events**. The dataset is not well balanced: in fact 12332 events are gamma and 6888 hadron.

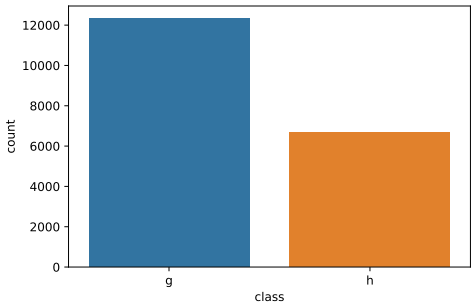


Figure 3: Dataset distribution

2.4 Statistical overview

For each attribute I analized the count, mean, standard deviation, minimum and maximum value and the quartiles.

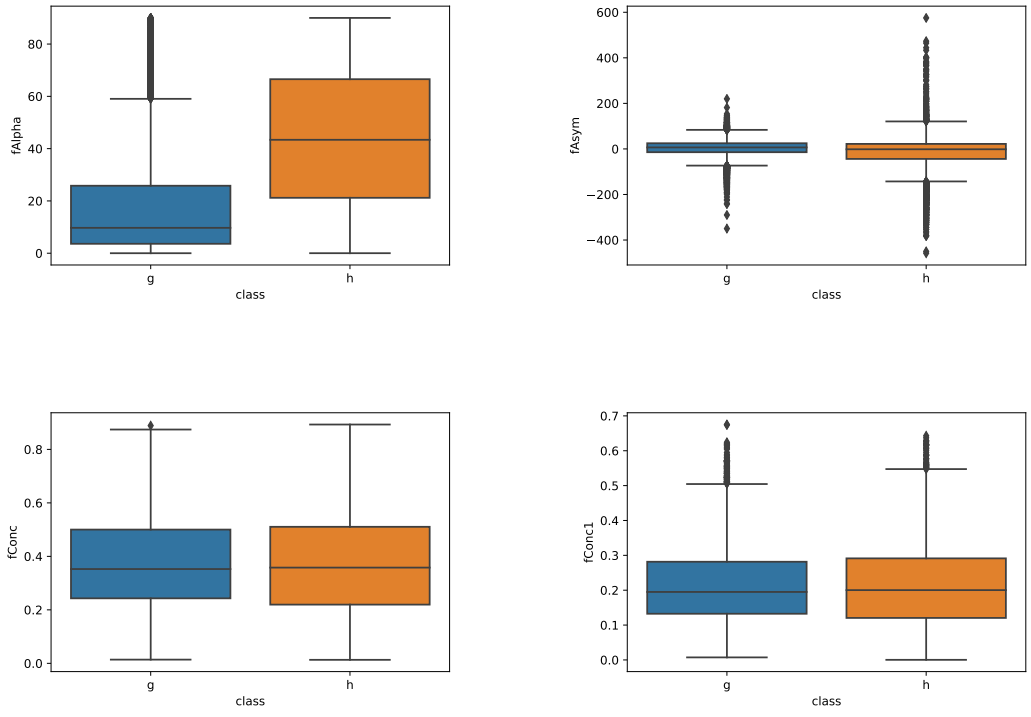
	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist
count	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000	19020.000000
mean	53.250154	22.180966	2.825017	0.380327	0.214657	-4.331745	10.545545	0.249726	27.645707	193.818026
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

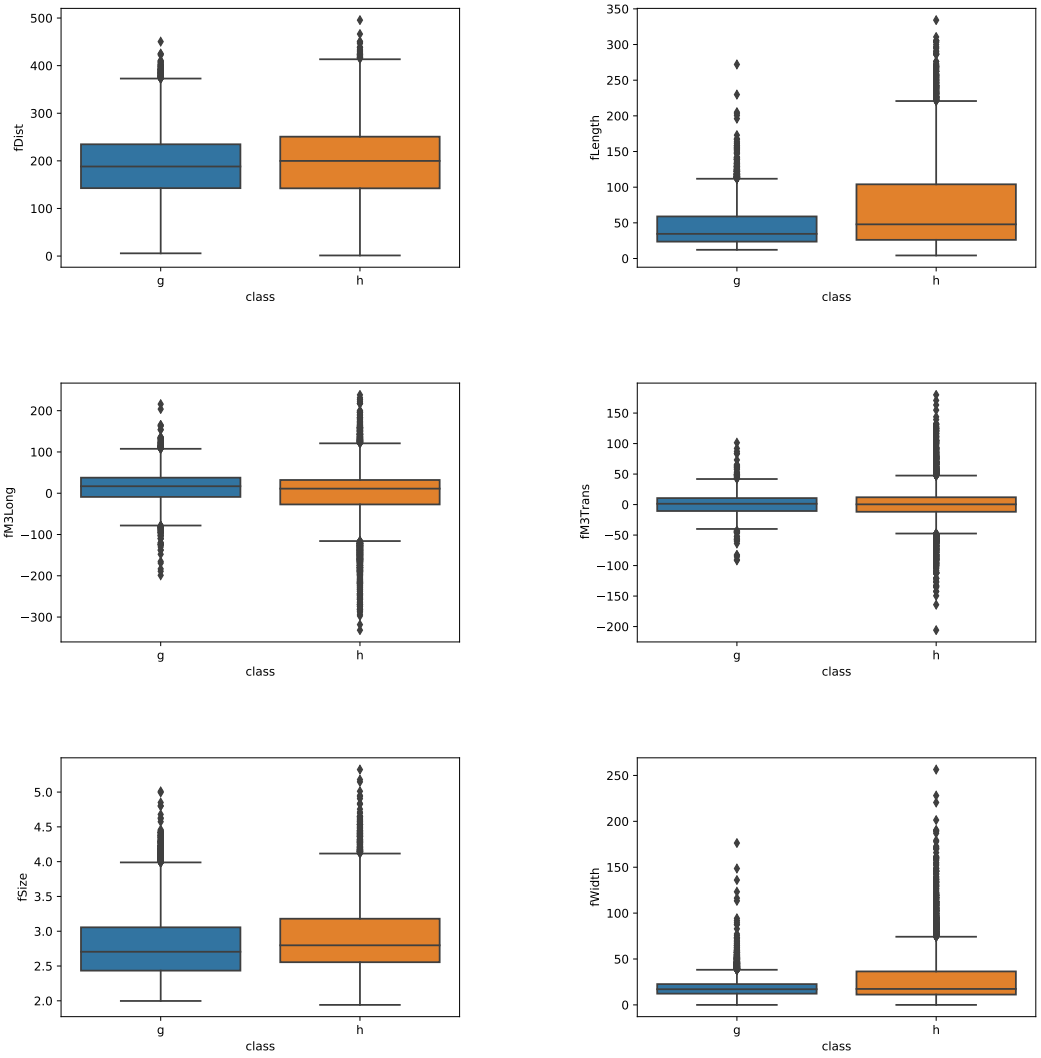
Figure 4: statistical overview

	Attribute	Domain
0	fLength	[4.2835, 334.177]
1	fWidth	[0.0, 256.382]
2	fSize	[1.9413, 5.3233]
3	fConc	[0.0131, 0.893]
4	fConc1	[0.003, 0.6752]
5	fAsym	[-457.9161, 575.2407]
6	fM3Long	[-331.78, 238.321]
7	fM3Trans	[-205.8947, 179.851]
8	fAlpha	[0.0, 90.0]
9	fDist	[1.2826, 495.561]
10	class	{g, h}

Figure 5: Attribute domain

The distributions of all attributes have been plotted as well: since all features are numerical we use boxplots.





2.5 Correlation

Correlation is a term that is a measure of the strength of a linear relationship between two quantitative variables. The most familiar measure of dependence between two quantities is the **Pearson product-moment correlation coefficient (PPMCC)**, or "Pearson's correlation coefficient", commonly called simply "**the correlation coefficient**".

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.

The population correlation coefficient between two random variables X and Y with expected values μ_X and μ_Y and standard deviations σ_X and σ_Y is defined as:

$$\rho_{X,Y} = corr(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sigma_X \sigma_Y}$$

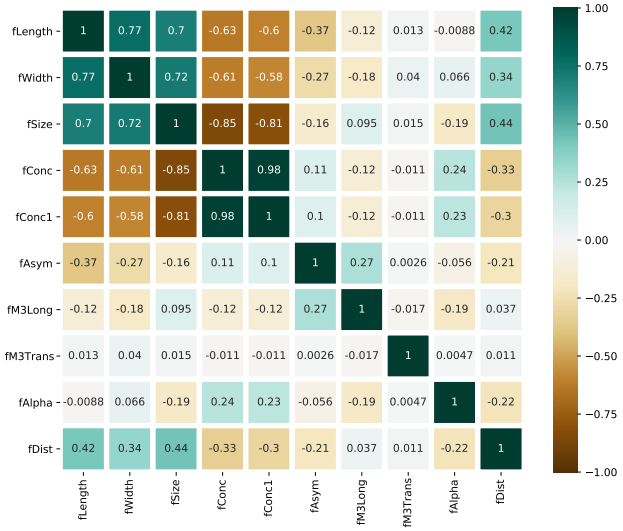


Figure 6: Correlation matrix

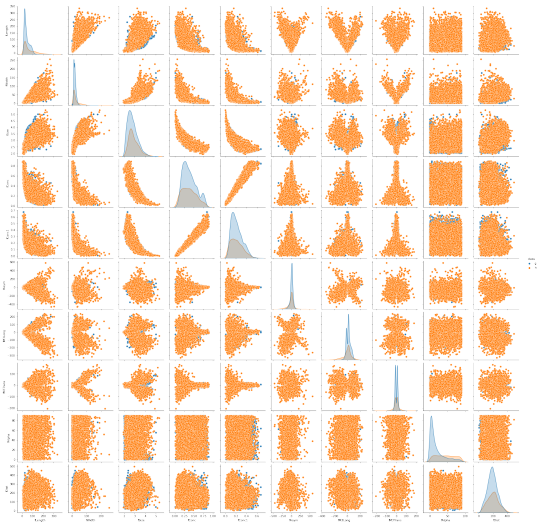


Figure 7: pairplot

2.6 Pairplot

A pair plot allows us to see both distribution of single variables, in the diagonal, and relationships between two variables.

2.7 Conclusions

- The dataset is not well balanced: in fact 12332 events are gamma and 6888 hadron. A problem could be that our model could have poor predictive performance especially for the minority class. We address this challenge with ad-hoc techniques.
- The attribute domain have a very different range: this can cause trouble to many machine learning models. For example, for the models that are based on distance computation, if one of the features has a broad range of values, the distance will be governed by this particular feature.
- From the boxplots we can noticed, for example, that there is a significant difference in the fAlpha and a slight difference in the fLength while the other boxplots are very similar. fAlpha will be more important than others features to distinguish events. Furthemore there are different outliers which can represent a problem for algorithm that are sensitive to outliers.
- The correlation coefficients between the variables *fLength-fWidth* and *fWidth-fSize* are respectively 0.77 and 0.72. This is the case of positive correlation, between *fLength-fWidth* and *fWidth-fSize* there is an increasing linear relationships: if one increase the other increase too. On the other hand, between *fConc* and *fSize* we have a negative correlation: if one increase the other decrease and viceversa.

3 Data preprocessing

3.1 Outliers detection

Outliers are those data points which differs significantly from other observations present in the dataset. It can occur because of variability in measurement and due to misinterpretation in filling data points.
To detect outliers there are several techniques: we use the **z-score method**: since our attributes have Gaussian distribution we calculate the Z-score as $data - mean/std$ and we remove points whose modulus of z-score is greater than a threshold value. This threshold value is set to 3.

3.2 Label encoder

Machine learning algorithm works only with numerical features so the first step is to encode target labels with value between 0 and nclasses-1. Since we are in a binary problem we have only two classes, we encode:

- class g: 0
- class h: 1

3.3 Standard scaler

The goal of standardization is to change the values of the variables in order to use a common scale, while preserving differences in each value range.

$$\frac{x - \mu}{\sigma}$$

We standardize the data such that each feature has mean equal to 0 and variance equal to 1. This is done such that each feature will equally contribute when using a machine learning algorithm. For example, for models based on distance metrics (e.g. KNN, SVM), if one feature has a larger values range, such feature would dominate much more in the distance computation, and we don't want that.
Moreover standardization has to be done especially if we want to apply PCA. PCA projects the original data onto the directions which maximize the variance, so if we don't standardize, PCA would interpret a feature to have more variance just because of its bigger scale. So it will erroneously proceed by projecting the data onto that direction. Conversely, by standardizing, all features would have the same variance, and so they would have the same weight for the principal components calculation.
Then we split the dataset into training and test sets, representing respectively 80% and 20% of the entire data. This is done for not overfitting the model, since the test is performed on a different set.

3.4 Dimensionality reduction

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 several reason to reduce the dimensionality of the data:

- high dimensional data could lead to computational problems.

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

The method we are going to use is called **Principal Component Analysis** (PCA). In PCA both the compression and the recovery are performed by liner 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.

3.4.1 PCA

The goal of PCA is to reduce dimensionality and then to be able to reconstruct our information by losing as little information as possible.
Let us consider m vectors in $\mathbb{R}^d$. We would like to reduce dimensionality of these vectors using a linear transformation.
A matrix $W \in \mathbb{R}^{n,d}$, where $n < d$, induces a mapping $x \to Wx$, where $Wx \in \mathbb{R}^n$ is the lower dimensionality representation of x. Then a second matrix $U \in \mathbb{R}^{d,n}$ can be used to recover original vector x from its compressed version. That is, for a compressed vector $y = Wx$,where y is in the low dimensional space $\mathbb{R}^n$, we can construct $\hat{x} = Uy$ so that $\hat{x}$ is the recovered version of x and resides in the original high dimensional space $\mathbb{R}^d$.
In PCA, we find the compression matrix W and the recovering matrix U so that the total squared distance between the original and recovered vectors is minimal, we aim at solving the problem:

$$argmin \sum_{i}^{m} \|x_i - UWx_i\|_2^2$$

How do we choose the matrix U and W?

For the following lemma: if (U, W) is a solution to the equation above, then the columns of U are orthonormal and $W = U^T$. We can rewrite our equation as $||x - UU^T||^2$ and our problem become to maximize the trace$(U^T x x^T U)$, where $U^T U = I$.
Let $A = \sum_{i=1}^{m} x_i x_i^T$, since A is symmetric we can written using its spectral value decomposition VDV^T. The elements of the diagonal of D are the eigenvalues of A and the columns of V are the corresponding eigenvectors. We assume that $D_{1,1} \geq D_{2,2} \geq .. \geq D_{d,d}$. We claim the **solution is the matrix U whose columns are the n eigenvectors of A corresponding to the largest n eigenvalues**.

In practice to choose the new number of components we will look at the cumulative explained variance ratio as shown in the plot below.

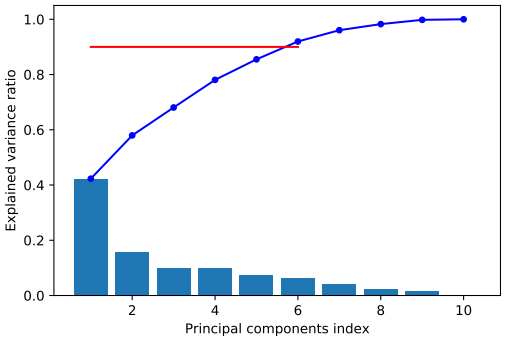


Figure 8: explained variance ratio

We select $ncomponents = 6$ since it is possible to explain more than 90% of the variance. The 90% value is represent by the red line in the figure 8.

4 Setting and Metrics

4.1 Oversampling

Oversampling techniques are a family of techniques designed to equalize an unbalanced class distribution. Such techniques generate new minority class samples and add them to the original dataset to balance out the class

distribution.

4.1.1 SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) synthesizes new minority class samples starting from the existing samples. It works by iteratively following these steps:

- First, select a random sample from the minority class
- Second, select the k nearest neighbours for the selected sample, from the minority class
- Last, one of the k neighbours is selected and a synthetic sample is generated as a point between the two selected samples

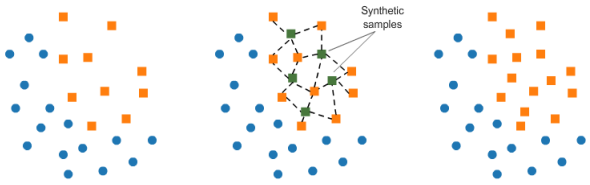


Figure 9: Smote technique

An **important consideration** that we have to do is that first we apply cross-validation: we exclude the sample to use as validation set and, after that, oversample the remaining of the minority class.

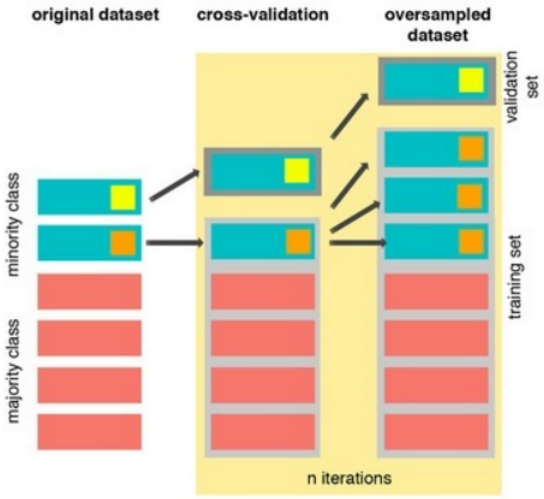


Figure 10: Sequence of operations

4.2 Stratified K fold

In order to evaluate the different models and tune their parameters, **k-fold with stratify** has been used. The entire dataset is divided in k parts and at each iteration one subsample is used as validation set while the remaining parts are used as training set. This process is repeated k times.

Since we want training and test to have similar distributions, we use stratify to maintain the proportions of the data, considering the binary classification.

Then the final score of the k-fold cross validation procedure is obtained from the average of the k models individual scores. This gives us an idea about how well the model will perform on average, and what are its overall generalization capabilities when applied to different test sets.

Last but foremost, CV is also used for finding the best model hyper-parameters (hyper-parameters 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.

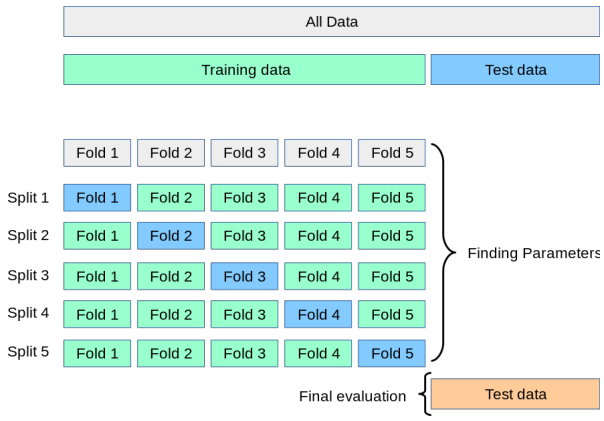


Figure 11: grid search with cross validation

4.3 The bias-variance trade off

Unless one is very careful the training data can mislead the learner, and result in overfitting. To overcome this problem, we restricted the search space to some hypothesis class H. This can be seen as reflecting some prior knowledge that the learner has about the task - a belief that one of the members of the class H is a low-error model for the task.

The **No-Free-Lunch theorem** states that no universal learner exists. To be more precise, the theorem states that for binary classification prediction tasks, for every learner there exists a distribution on which it fails. We say that the learner fails if its output hypothesis is likely to have a large risk ($\geq .3$), whereas for the same distribution, there exists another learner that will output a hypothesis with a small risk. Therefore when approaching a particular learning problem, defined by some distribution D, we should have some prior knowledge on D.

How to prevent such failures?
We can use our prior knowledge to restrict our hypothesis class.

But how should we choose a good hypothesis class?
On the one hand, we want that this class includes the hypothesis with the smallest error achievable; on the other hand, we have just seen that we cannot simply choose the richest class - the class of all functions over the given domain. In order to resolve this trade off we decompose the Empirical Risk in two:

$$L_D(h_S) = \varepsilon_{app} + \varepsilon_{est}$$

1 ε_{app}: the minimum risk achievable by a predictor in the hypothesis class. This term measures how much risk we have because we restrict ourselves to a specific class, namely, how much **inductive bias**. The approximation error does not depend on the sample size and is determined by the hypothesis class chosen. Enlarging the hypothesis class can decrease the approximation error

2 ε_{est}: the difference between the approximation error and the error achieved by the ERM predictor. The estimation error results because the 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.

Since our goal is to minimize the total risk, we face tradeoff, called the **bias-complexity** tradeoff.

One of the common things to do for understanding which of the two errors the algorithm is suffering is plotting **learning curves**. In this representation the algorithm is trained on increasing part of the training set, starting for example with 10% of the whole dataset, then with 20% and so no. At each steps the training error is calculated on both training and validation set. In this way if the trend of training error and the validation error remains constant for all the steps, then the algorithm is suffering of approximation error since it is not actually learning more, while if the validation error starts as a constant and then starts decreasing, we are in the estimation error scenario.

4.4 Confusion Matrix

In this work we are facing a binary classification problem, so the two errors possible are false-positive (FP) and false-negative (FN). In these cases, we mis-classify respectively a prediction $y = 1$, when the real value is

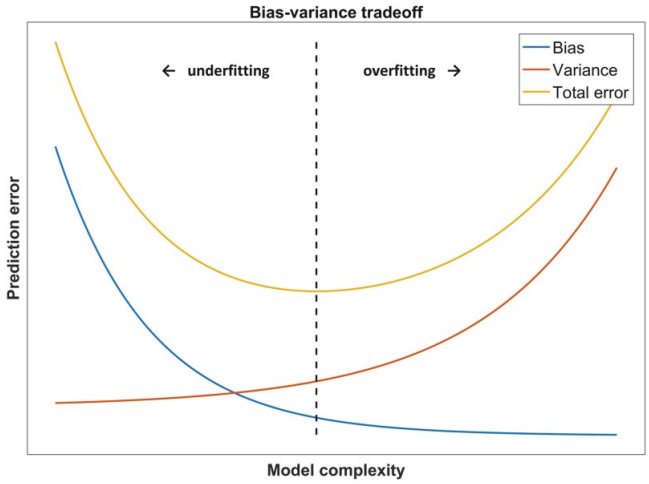


Figure 12: The bias variance trade off

0, and a prediction $y = 0$, when the real value is 1. One of the most common ways to represent these values is called Confusion Matrix, in which each row is an instance of the predictive class, and each column are the instances of the true classes. As we can see in the figure, the diagonal values on the matrix represents the correct

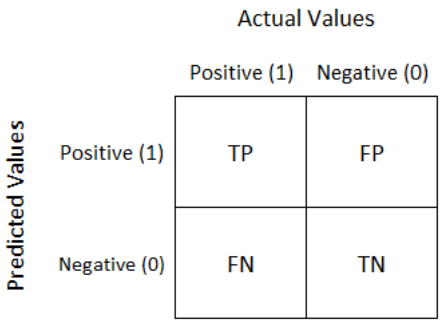


Figure 13: Confusion Matrix

classification, while the off-diagonal values are the misclassified values. From the table, we can also compute the metrics to evaluate the classification models:

- **accuracy**: defined as the number of correct prediction divided by the total number of predictions.

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

- **recall** or TPR: defined as the number of correctly positive predicted values over the total actual positives.

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

- **precision**: defined as the number of true predicted positives over the total number of positives.

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

- **F1-score**: defined as a function of both precision and recall.

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

4.5 ROC curve

Suppose we are solving a binary decision problem. Also, assume we have labeled data set, $D = \{(x_i, y_i)\}$. Let $\phi(x) = I(f(x) > \tau)$ be our decision rule, where $f(x)$ is measure of confidence that $y = 1$, and τ is some threshold parameter. For each given value of τ, we can apply our decision rule and count the number of true

positives, false positives, true negatives and false negatives that occur.
However, rather than computing the TPR and FPR for a fixed threshold τ, we can run our detector for a set of thresholds, and then plot the TPR vs FPR as an implicit function of τ. This is called a **receiver operating characteristic** or **ROC** curve.
The quality of a ROC curve is often summarized as a single number using the **area under the curve** or **AUC**. Higher AUC scores are better; the maximum is obviously 1.

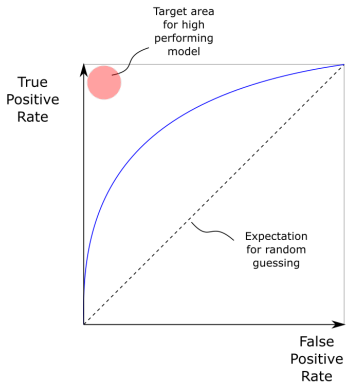


Figure 14: ROC curve

Any system can achieve the point on the bottom left, $(FPR = 0, TPR = 0)$, by setting $\tau = 1$ and thus classifying everything as negative: similarly any system can achieve the point on the top right, $(FPR = 1, TPR = 1)$, by setting $\tau = 0$ and thus classifying everything as positive. A system that perfectly separates the positives from negatives has a threshold that can achieve the top left corner, $(FPR = 0, TPR = 1)$; by varying the threshold such a system will "hug" the left axis and then the top axis, as shown in the Figure [14].

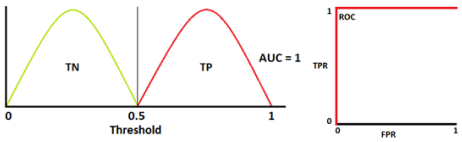


Figure 15: Example of ROC curve with AUC=1

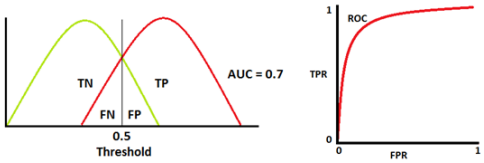


Figure 16: Example of ROC curve with AUC=0.7

5 Algorithms

5.1 Logistic Regression

In logistic regression we learn a family of functions **h** from $\mathbb{R}^d$ to the interval $[0, 1]$. However logistic regression is used for classification task: we can interpret $h(x)$ as the **probability** that the label of **x** is 1. The hypothesis class associated with logistic regression is the composition of a sigmoid function over the class of linear functions L_d. The sigmoid function used in logistic regression is the **logistic function**, defined as:

$$\phi_{sig}(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

The name sigmoid means S-shaped and it is referred to the plot of this function.

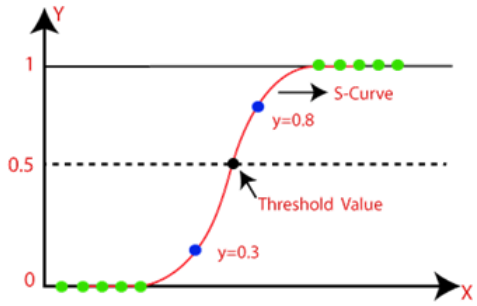


Figure 17: Logistic Regression

The hypothesis class is therefore: $H_{sig} = \left\{x \mapsto \phi_{sig}(\langle w, x\rangle) : w \in \mathbb{R}^d\right\}$.
Note that when $\langle w, x\rangle$ is very large then $\phi_{sig}(\langle w, x\rangle)$ is close to 1, whereas if $\langle w, x\rangle$ is very small then $\phi_{sig}(\langle w, x\rangle)$ is close to 0.

The logistic function simply models probability of output in terms of input and does not perform classification (it is not a classifier). A common approach is to choose a cutoff value (usually 0.5) and classifying inputs with probability greater than the cutoff as one class and below as the other. Logistic Regression is a frequent choice to make a binary classification (as in this case).

$$\hat{y} = \begin{cases} 1 \; if P[y=1|x] > 0.5 \\ 0 \; if P[y=0|x] < 0.5 \end{cases}$$

Next, 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 $1 + e^{(-y\langle w,x\rangle)}$ (log is a monotonic function). That is

$$l(h_w, (x, y)) = log(1 + e^{(-y\langle w,x\rangle)})$$

Therefore, given a training set $S = (x_1, y_1), ..., (x_m, y_m)$ the problem associated with logistic regression is

$$\arg\min_{w\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} log(1 + e^{(-y_i\langle w,x_i\rangle)})$$

5.2 Decision Tree

A decision tree is a predictor , h : X → 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**. A leaf contains a specific label.

In training phase, unfortunately, since it is computationally infeasible to consider every possible partition of the feature space, decision tree learning algorithms are based on heuristics such as a greedy approach, where the three is constructed gradually, and locally optimal decisions are made at the construction of each node. The algorithm at each step chooses the best predictor to split according to some criteria. The most commons ones are:

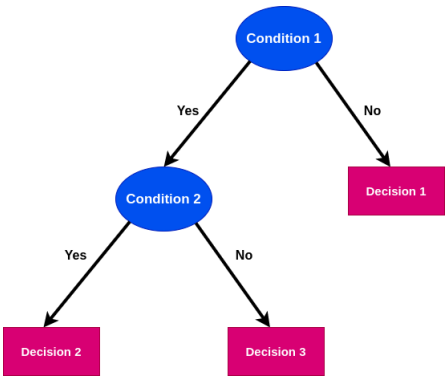


Figure 18: Decision Tree

1 **Gini Index**: $1 - \sum_{i=1}^{n} p_i^2$

2 **Cross Entropy**: $\sum_{i=1}^{n} -p_i log_2 p_i$

p_i is the probability of an object being classified to a particular class.

Advantages of a decision tree:

- Easy to understand: decision trees are very easy to read and interpret, in fact they do not require any statistical knowledge and its graphical representation is very intuitive;
- useful in Data Exploration: it is very easy to identify most significant variables and the relation between them;
- number of hyper-parameters to be tuned is low.

Disadvantages of a decision tree:

- Over-fitting: decision trees may not generalize the data well due to over-complex trees. This issue can be solved by introducing some constraints and pruning;

5.3 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 (each sample has the same probability to be picked) the samples from the training set at random with replacement.
Given a set of n independent observations $Z_1, Z_2, ..., Z_n$ each with σ^2, the variance of the mean $\overline{Z}$ of the observations is given by $\frac{\sigma^2}{n}$, which means that averaging a set of observations reduces the variance.
Therefore, $T_1, T_2, ..., T_n$ sets are sampled from the training set to create N separate models. Then, using a majority vote criterium, the most frequent class among N predictors is chosen.
In the case of Random Forest this procedure is further improved with a trick that allows to decorrelate the trees and hence further reduce the variance when averaging them. In fact, every time a split is considered during the built of tree, a random selection of m predictors is chosen among the B prediction, and the tree is allowed to use only one of these m predictors. Usually $m \approx \sqrt{B}$.

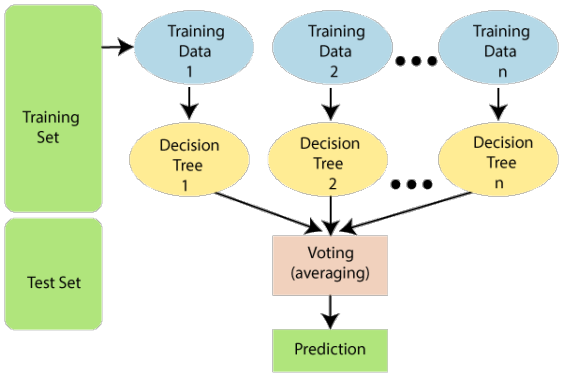


Figure 19: Random forest

5.4 SVM

SVM is a supervised learning algorithm and it is used both for classification and regression. SVM performs classification tasks by constructing hyperplanes in a multidimensional space that separates data of different class labels.

Let us consider a binary classification problem, the training set is $S = (x_1, y_1), ..., (x_m, y_m)$ where $y_i \in [\pm 1]$. We say that this **training set** is **linearly separable**, if there exists a halfspace, (w, b), such that $y_i = sign(\langle w, x_i \rangle + b) > 0$.

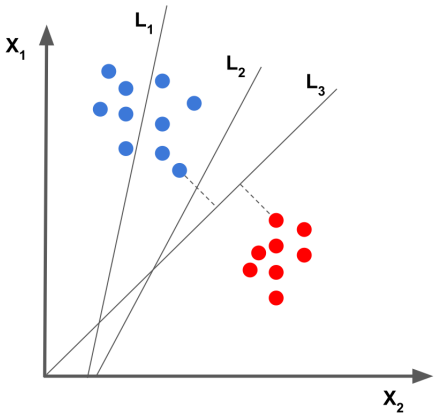


Figure 20: Support vector machine

In the figure above L_1, L_2, L_3 are hyperplanes. We notice that the best is L_3, to formalize this intuition we need the concept of **margin**. The margin of a hyperplane with respect to a training set is defined to be the minimal distance between a point in the training set and the hyperplane. If a hyperplane has a large margin, then it will still separate the training set even if we slighlty perturb each instance.

5.4.1 Hard-SVM

Hard-SVM is the learning rule in which we return an hyperplane that separate the training set with the largest possible margin. The distance between a point $\mathbf{x}$ and the hyperplane defined by (w, b) where $||w|| = 1$ is $|\langle w, x \rangle + b|$.
The closest point in the training set to the separating hyperplane is $min_{i \in [m]} |\langle w, x_i \rangle + b|$, therefore the hard-svm rule is

$$argmax \;\; min_{i \in [m]} |\langle w, x_i \rangle + b| \;\; s.t. \;\; \forall i, y_i(\langle w, x_i \rangle + b) > 0$$

The output of the SVM is indeed the separating hyperplane with the largest margin. Hard-SVM searches for $\mathbf{w}$ of minimal norm among all the vectors that separate the data and for which $|\langle w, x_i \rangle + b| \geq 1$, for all i. Therefore finding the largest margin halfspace boils down to finding $\mathbf{w}$ whose norm is minimal.

$$(w_0, b_0) = argmin||w||^2 \; such \; that \;\; \forall i, y_i(\langle w, x_i \rangle + b) \geq 1$$

.

5.4.2 Soft-SVM

The Hard-SVM assumes that the training set is linearly separable, which is rather strong assumption. Soft-SVM can be viewed as a **relaxation of the Hard-SVM rule** that can be applied even if the training set is not linearly separable. The optimization problem seen above enforces the hard constraints $\forall i, y_i(\langle w, x_i\rangle + b) \geq 1$. A relaxation is to allow the constraints to be violated for some of the examples in the training set. This can be modeled introducing nonnegative slack variables, $\epsilon_1, \epsilon_2, ..., \epsilon_m$ that measure how the constraint is being violated. Soft-SVM jointly minimizes the norm of $\mathbf{w}$ and the average of ϵ_i. The trade-off between the two terms in controlled by a parameter λ.

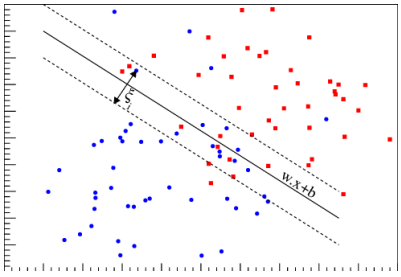


Figure 21: Soft-SVM

The problem now is to solve:

$$argmin_{w,b,\epsilon}(\lambda||w||^2 + \frac{1}{m}\sum_{i=1}^{m}\epsilon_i)$$

such that

$$\forall i, (y_i\langle w, x_i\rangle + b) \geq 1 - \epsilon_i \; and \; \epsilon_i \geq 0$$

5.4.3 Non-Linear SVM: Rbf Kernel

If we do not find a line to separate the classes, we will use the Kernel Trick that consists in projecting the points in a higher dimensional space to get a representation which is linearly separable. The other hypermeter is gamma that defines how far the influence of a single training example reaches and it is used for non linear hyperplanes. If gamma is low every point has a far reach, on the other hand a high value means that the only the closest point will contribute.
The radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms. In particular, it is commonly used in support vector machine classification.

$$k(x_i, x_j) = e^{-\gamma||x_i - x_j||^2}$$

6 Training and results

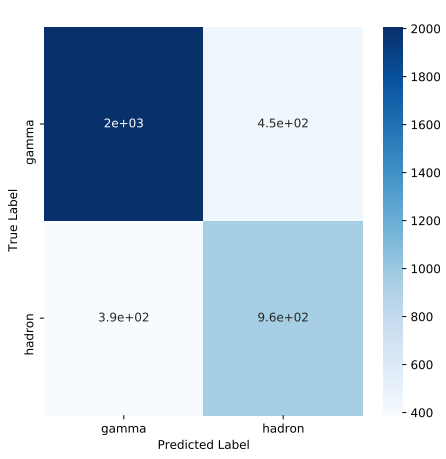
We test the algorithms on our dataset, then analyze and compare the different scores.

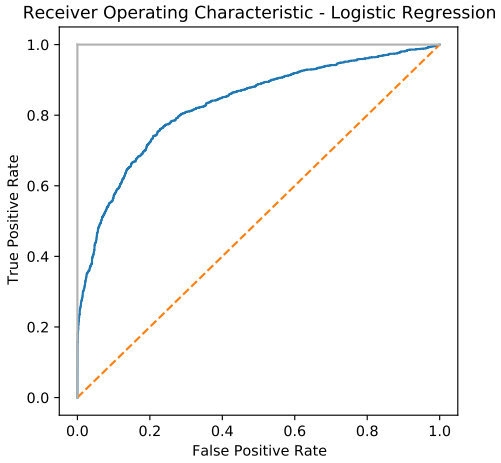
6.1 Logistic Regression

The combinations of the following parameters have been evaluated in hyperparameter tuning:

- 'penalty' : 'l1', 'l2';
- 'C': 0.001, 0.01, 0.1, 1, 10, 100;
- 'randomstate': 33, 42, 66.

The best configuration is with 'penalty' = 'l2', 'C' = 1, 'randomstate' = 33.





6.2 Decision Tree

The combinations of the following parameters have been evaluated in hyperparameter tuning:

- 'criterion' : 'gini', 'entropy';
- 'maxDepth': 2, 5, 10, 15;
- 'minImpurityDecrease': 0.001, 0.01, 0.1, 1;
- 'randomstate': 33, 42, 66.

The best configuration is with 'criterion' = 'gini', 'maxDepth' = 10, 'minImpurityDecrease' = 0.001 and 'randomstate' = 33.

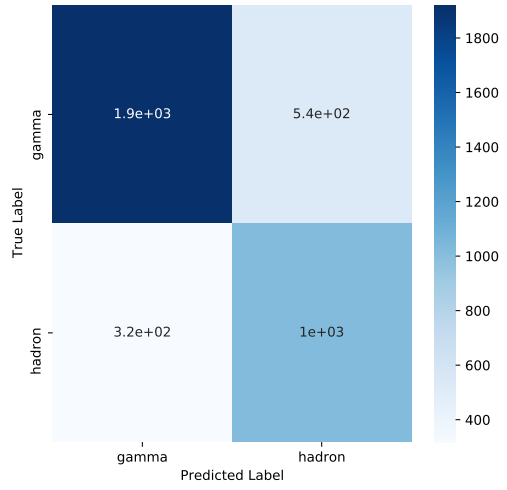


Figure 22: Confusion Matrix for decision tree.

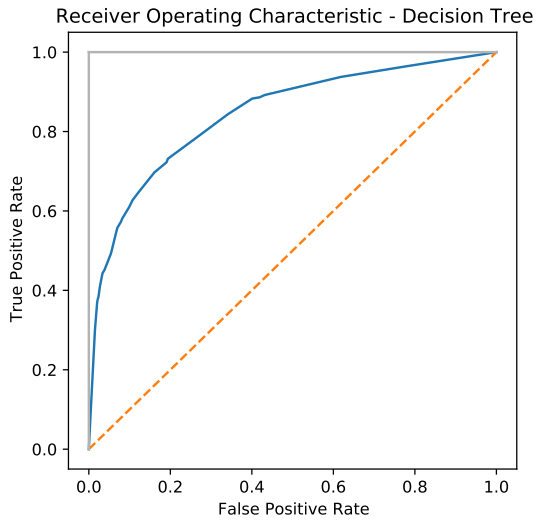


Figure 23: ROC curve for Decision Tree

In this learning curve we compare the performance of a model on training and testing data over a varying number of training instances. We can get an idea of how well the model can generalize to new data.
In this curve the training score is very high at the beginning and decreases and the cross-validation score is very low at the beginning and increases.

The figure [24] shows the representation of the Decision Tree with the parameter maxDepth set to three in order to better visualize the data. We can notice, according to our preliminary analysis, that the first attribute use to split the data is fAlpha, it has the highest gini (0.456). If we look the boxplot of such attribute we notice that it is one with the most different values for each class with respect to the other features which have more similar mean and std.

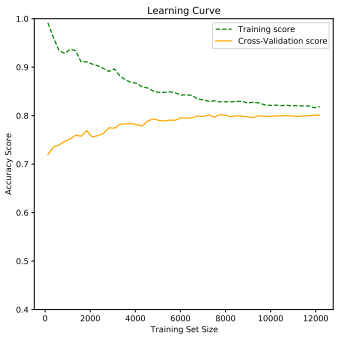


Figure 24: Learning curve for Decision Tree

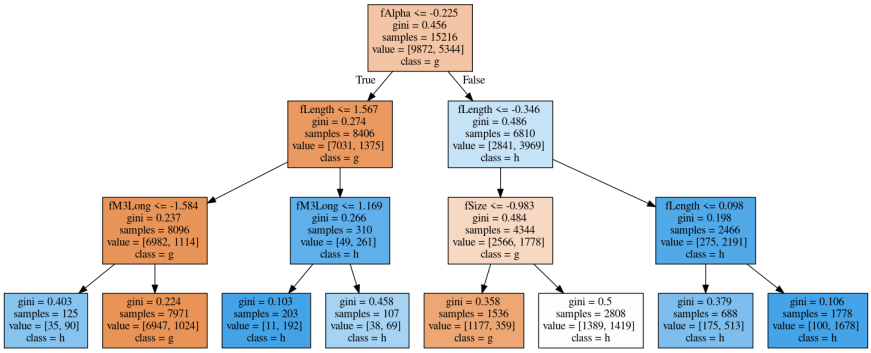


Figure 25: Decision Trees

6.3 Random Forest

The combinations of the hyperparameter is the same as the decision tree, in addition we tune the hyperparameter 'nEstimators' which represents the number of decision tree used and the best is 'nEstimators'= 500.

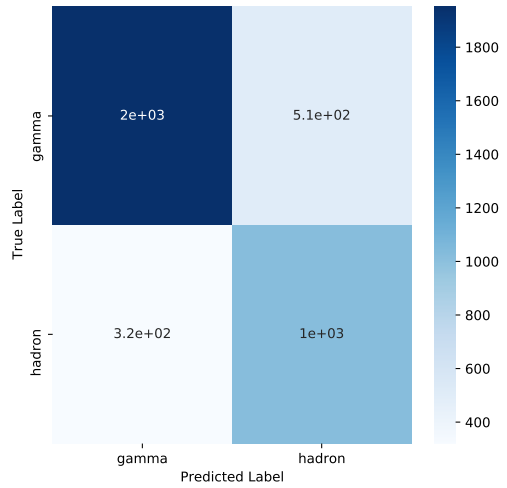


Figure 26: Confusion Matrix for Random Forest.

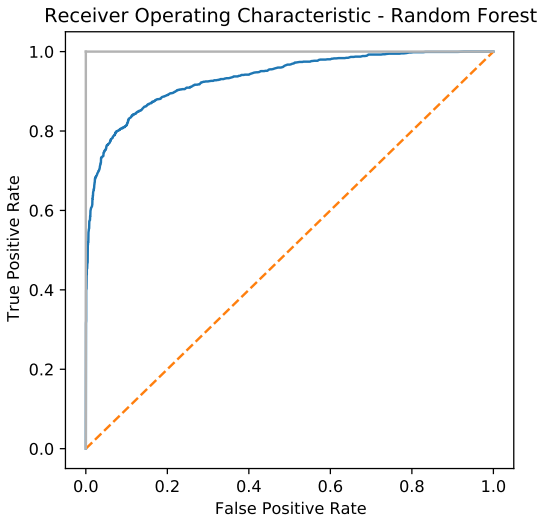


Figure 27: ROC curve for Random Forest

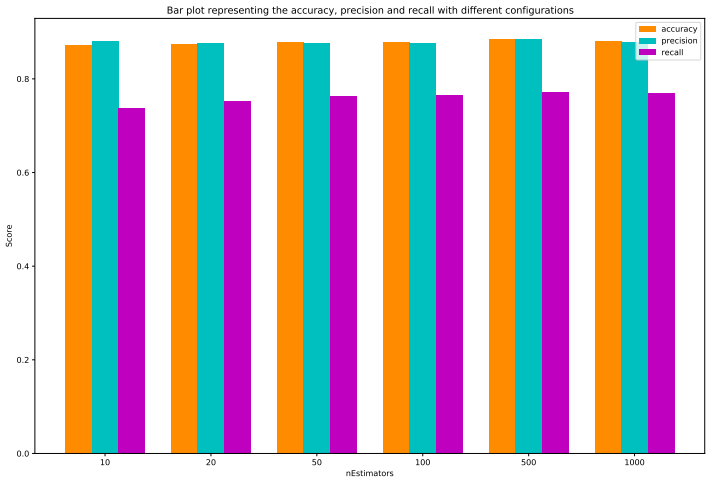


Figure 28: Score with different numbers of decision trees ensembled

6.4 SVM

6.4.1 Linear SVM

The only hyperparameter that I tuned with the linear SVM is the regularization term C. To better understand the trend of the accuracy score as changing C, I plotted the so-called validation curve, illustrated in figure 30. The selected hyperparameter is the one that maximizes the cross-validation score, in this case C = 100.

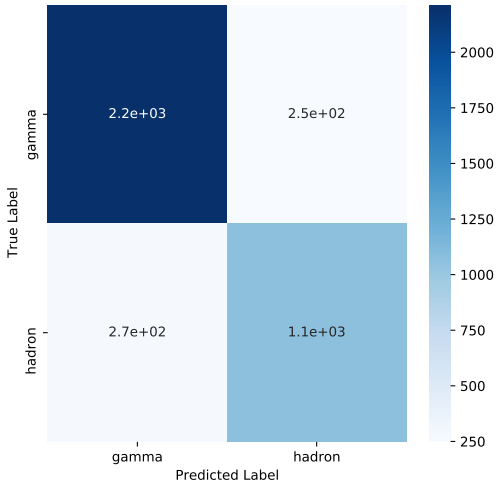


Figure 29: Confusion Matrix for Linear SVM.

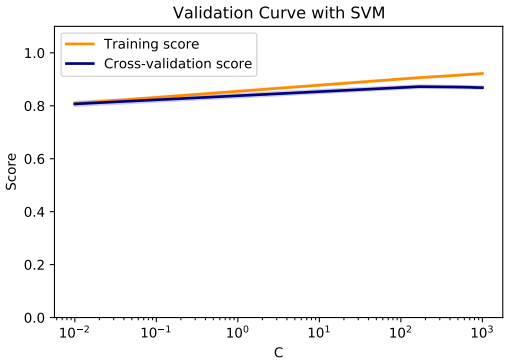


Figure 30: Validation Curve for Linear SVM

6.4.2 Kernel SVM

The RBF function contains a new hyperparameter γ , which needs to be tuned. Intuitively, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. Hence, I performed a k-fold cross-validation for different pairs of C and gamma. The best value found for C was 100 and then with that value of C I choose γ To visualize the results, I plotted a validation curve, for C=100, the best value of γ which maximize the cross-validation score is 10e-3.

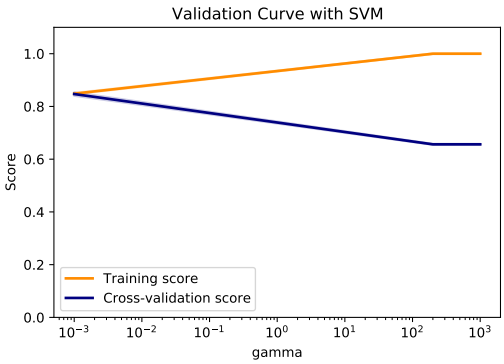


Figure 31: Validation curve with Kernel SVM

7 Conclusions

	Accuracy	Precision	Recall
Logistic Regression (no preprocessing)	0.789	0.692	0.727
Logistic Regression (or,pca,smote)	0.775	0.662	0.740
Logistic Regression (pca,smote)	0.778	0.678	0.711
Decision Tree (no preprocessing)	0.835	0.765	0.770
Decision Tree (or,pca,smote)	0.770	0.651	0.749
Decision Tree (pca,smote)	0.836	0.765	0.77
Random Forest (no preprocessing)	0.853	0.796	0.787
Random Forest (or,pca,smote)	0.773	0.644	0.799
Random Forest (pca,smote)	0.782	0.668	0.762
Linear SVM (no preprocessing)	0.863	0.820	0.787
Linear SVM (or,pca,smote)	0.798	0.684	0.798
Linear SVM (pca,smote)	0.809	0.722	0.747
Kernel SVM (no preprocessing)	0.862	0.812	0.798
Kernel SVM (or,pca,smote)	0.797	0.684	0.798
Kernel SVM (pca,smote)	0.812	0.731	0.763

Figure 32: Algorithm reults