POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Data Spaces

Classification of interstellar objects

Relatore
Prof. Francesco Vaccarino

Candidati
Mattia Cara
Nicolò Chiapello

Settembre 2019

Contents

Chapter 1

Introduction

1.1 Overview

The current report represents the final assignment of the *Data Spaces* course at *Politecnico di Torino*.

Introduction to the problem

Since the dawn of time, human beings have gazed up at the sky full of curiosity. Amazed by such beautiful view, they started wondering who deserve to dwell in that place. They thought that the deities resided in the Vault of Heaven, to monitor the mankind from above. Afterward, they started to considering the sky in a more critical way, and noticed that it is extremely helpful in orienting on large expanses without landmarks. Greater scientific rigor came when the telescope allowed an embryonic space exploration: the heliocentric theory came out leading to a huge earthquake in the society of that time.

Since forever, humanity tried to elicit the secrets of the firmament and, since forever, it has been the edge of the human intellect. Now more than ever, the race to the outer space is a topical and disruptive subject with serious scientific, political and recently also military, implications. This interest in the extraterrestrial environment and the latent rush to mineral resources, lead many telescopes to turn upwards. Nowadays, the scientific instrumentation is far more advanced than the Galilei and Copernico ones; still, the task is so complex that is requested the contribution of any scientific branch. In particular, the *data science* is helpful in analyzing the data, collected by space probes or terrestrial observatories, in order to find out recognizable patterns and take a glimpse beyond the curtain of the unknown.

Report structure

The current work analyzes a dataset of spatial data related to observed sky objects. The raw samples comes from two different types of analysis:

- photometric data
- spectral data

Those information are combined together and linked to the type of the sky object itself. For the aim of this report, we will consider this labelling as the ground-truth of the real nature of those celestial bodies.

Having access to an already assigned `class` feature, the current one is a typical **supervised learning** problem:

$$y = f(\vec{x})$$

It is possible to infer the f mapping function, from input data $\vec{x}$ to result y, based on example input-output pairs (already labelled).

In particular, the chosen dataset is eligible for **classification** tasks: creating a model (i.e. f) able to determine the right type of sky object (i.e. y) according to its photometric and spectral features (i.e. $\vec{x}$). Several classification techniques will be analyzed in this report: after a brief theoretical review, their results (mainly charts) will be exposed. The selected algorithms are the following:

- k-nearest neighbour (kNN)
- logistic regression
- support vector machine (SVM)
- tree based methods and random forest

For each of them some standard operations will be performed, in the following order:

1. preprocessing: cleaning the data and reducing the dataset dimensionality
2. parameter tuning: defining the model *hyperparameters*, that best suits the training set, according to a *validation-set approach*
3. model training: fitting the chosen model to data, by reducing a *loss function*
4. performance evaluation: classifying the testing set and checking the correspondence between the provided results and the actual class labels

At the end, the classifiers results will be compared together in order to find the best solution, but also their strengths, drawback and limitations.

1.2 Used tools

In performing the current analysis, we rely on some available technology; especially from the ICT field.

Programming language

Due to the computer science background of the two authors of this work, we decided to exploit the Python programming language. This choice allows us to have a better control on the operations we are actually performing, more flexibility and an improved code structure (e.g. *object oriented programming* paradigm).

Frameworks

Riding the wave of interest in *Machine Learning* applications, Google provides a powerful tool for researchers and students: Google Colab (short for *Colaboratory*). It is a web application that provides a *plug-and-play* coding environment (based on Jupiter notebook), that allows a first-hand experience on *ML*, without complex setup. It also provides a remote and free access to a GPU (NVIDIA with Cuda library), great for parallelize *Deep Learning* tasks (e.g. *Convolutional Neural Network*) exploiting Pytorch or TensorFlow frameworks.

Furthermore, *Colab* provides good integration with *VCS* systems. We setup a GitHub repository for storing the necessary code and collaborate efficiently on the same project.

Custom tools

Working on this project in two people, we decided to implement a class, named `CommonTools.py`, that holds methods that have been used by the both of us.

Indeed it **provides the dataset**, "arranged" in four different ways:

- raw data set: all the columns considered;
- most relevant data set: most relevant columns coming from the feature selection using random forest;
- "most meaningful" data set: the subset of predictors that we think have the biggest meaning while dealing with this topic;
- PCA data set: the first five principal components of the PCA.

We have decided to introduce the "most meaningful" data set because the feature selection analysis have included a few features related to the instruments used during the observation (such as `specobjd` and `plate`) and also the date of the observation (`mjd`) (see section 1.3). In our opinion, is really hard to believe that an information related to a date can influence the class of a heavenly body. We have supposed that this high correlation between the final class and these features, apparently not meaningful, is due to observations performed on the same day with the same machinery during some researches aimed to find only a certain type of body.
For the PCs choose number, refer to the PCA description (see section 2.2).

We have also a method for the **grid search**, used while tuning the hyperparameters, for the accuracy and the confusion matrix.

Our case studio presents three different classes and `quasar` is way less numerous then the other two. Because of this the accuracy is for sure important but does not always represent a good metric of your algorithm: is it possible to achieve a very good accuracy even if all the records of the minority class are wrongly labeled. To overcome this problem we have provided also the **confusion matrix**: on its diagonal we have the percentage of how many times each class is correctly assigned and this is the best measure to be used in this case.

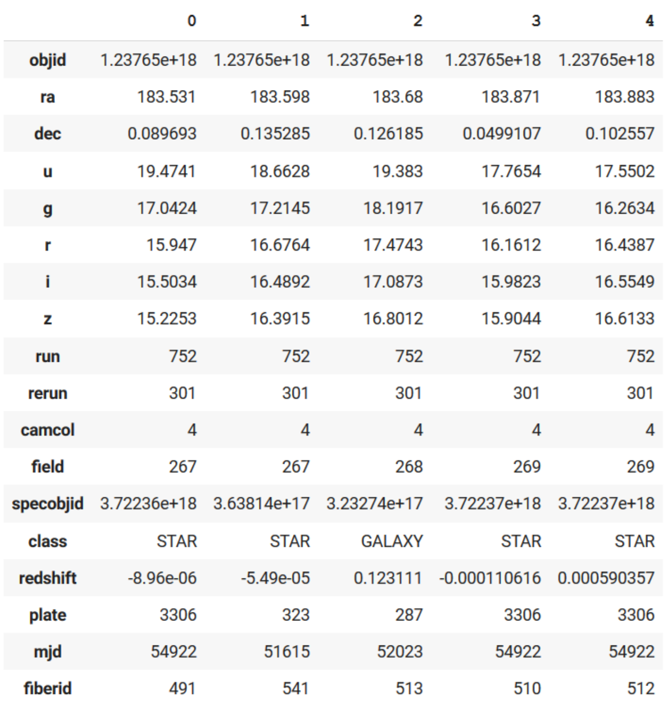
1.3 Dataset exploration

Let dive more deeply in the technical aspects and start familiarize with the actual data.

Dataset origin

The chosen dataset is the Sloan Digital Sky Survey DR14 one. It comes from the Kaggle archive and it was specifically proposed as a classification task to distinguish between different types of objects.

Let take a first glance at some data points (transposed for sake of presentation), in order to get the feeling of the dataset composition:



	0	1	2	3	4
objid	1.23765e+18	1.23765e+18	1.23765e+18	1.23765e+18	1.23765e+18
ra	183.531	183.598	183.68	183.871	183.883
dec	0.089693	0.135285	0.126185	0.0499107	0.102557
u	19.4741	18.6628	19.383	17.7654	17.5502
g	17.0424	17.2145	18.1917	16.6027	16.2634
r	15.947	16.6764	17.4743	16.1612	16.4387
i	15.5034	16.4892	17.0873	15.9823	16.5549
z	15.2253	16.3915	16.8012	15.9044	16.6133
run	752	752	752	752	752
rerun	301	301	301	301	301
camcol	4	4	4	4	4
field	267	267	268	269	269
specobjid	3.72236e+18	3.63814e+17	3.23274e+17	3.72237e+18	3.72237e+18
class	STAR	STAR	GALAXY	STAR	STAR
redshift	-8.96e-06	-5.49e-05	0.123111	-0.000110616	0.000590357
plate	3306	323	287	3306	3306
mjd	54922	51615	52023	54922	54922
fiberid	491	541	513	510	512

Figure 1.1. Dataset overview (original)

Feature description

The data consists of 10,000 observations of space taken by the SDSS (Sloan Digital Sky Survey) an extremely detailed three-dimensional maps of the Universe. Every observation is described by 18 feature columns, one of which is exploited as the target class label.

In particular, the `class` variable has a categorical `string` type (easily mappable to an `integer`) taking the following values:

- galaxy
- quasar

- star

The dataset results from a query which joins two tables (according to the type of analysis mentioned above): photometric and spectral data. Each sample is an observation and the overall describing features (both *predictors* and *response* variables) are the following:

- `PhotoObj`: photometric data
  - `objid`: object identifier (primary key of the `PhotoObj` table)
  - `ra`: J2000 Right Ascension (r-band)
  - `dec`: J2000 Declination (r-band)
  - `u`, `g`, `r`, `i`, `z`: Thuan-Gunn astronomic magnitude system
  - `run`: run number
  - `rereun`: rerun number
  - `camcol`: camera column
  - `field`: field number
- `SpecObj`: spectral data
  - `specobjid`: object identifier (primary key of the `SpecObj` table)
  - `redshift`: final redshift
  - `plate`: plate number
  - `mjd`: MJD of observation
  - `fiberid`: fiber ID
  - `class`: target object class (galaxy, star or quasar object)

The `objid` and `SpecObjid` fields are sequential numerical identifiers that allows to address a specific object. They were used to join the two tables together.

Right ascension (abbreviated `ra`) is the angular distance measured eastward along the celestial equator from the Sun at the March equinox to the hour circle of the point above the earth in question. When paired with declination (abbreviated `dec`), these astronomical coordinates specify the direction of a point on the celestial sphere (traditionally called in English the skies or the sky) in the equatorial coordinate system.

In the Thuan-Gunn astronomic magnitude system, the `u`, `g`, `r`, `i`, `z` values represent the response of the 5 bands of the telescope.

Run, rerun, camcol and field are features which describe a field within an image taken by the SDSS. A field is basically a part of the entire image corresponding to 2048 by 1489 pixels. A field can be identified by: - `run` number, which identifies the specific scan, - the camera column, or `camcol`, a number from 1 to 6, identifying the scanline within the run, and - the `field` number. The field number typically starts at 11 (after an initial rampup time), and can be as large as 800 for particularly long runs. - An additional number, `rerun`, specifies how the image was processed.

In physics, `redshift` happens when light or other electromagnetic radiation from an object is increased in wavelength, or shifted to the red end of the spectrum.

Each spectroscopic exposure employs a large, thin, circular metal plate that positions optical fibers via holes drilled at the locations of the images in the telescope focal plane. These fibers then feed into the spectrographs. Each plate has a unique serial number, which is called `plate` in spectral views.

The `MJD`, that stands for Modified Julian Date, represents the date that a given piece of SDSS data (image or spectrum) was taken. It is simply a time reference of the observation.

The SDSS spectrograph uses optical fibers to direct the light at the focal plane from individual objects to the slithead. Each object is assigned a corresponding `fiberID`.

As the name already says, the `class` column is the response we are trying to determine, in the classification tasks. It is a categorical field that can take only three values (i.e. galaxy, star or quasar).

Feature selection

Even though the previously exposed features are all numerical (thus useful for a methematical approach), several of them are not really meaningful for the aim of classification: some are instrumentation-related and other are simply technical identifiers.

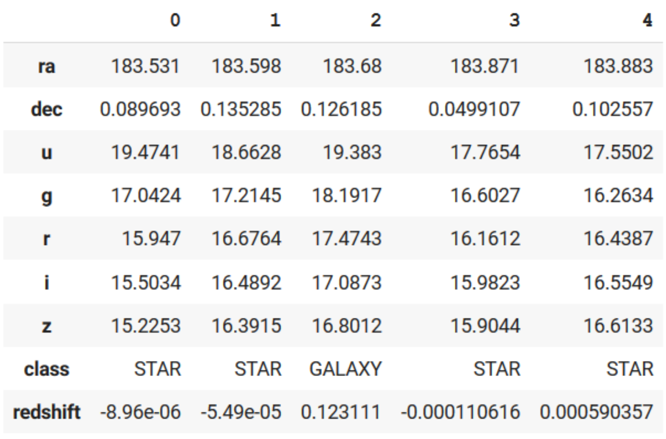
For better handling the dataset and provide a significative input to the algorithms we will apply, we repeat the analysis with a dataset (i.e. the **most meaningful dataset**) in which the following features will be discarded:

- `objid`, `SpecObjid`: numerical ID of the celestial bodies
- `run`, `rerun`, `camcol`, `field`: parameters used to identify from which scan data are taken
- `plate`, `mjd`, `fiberid`: parameters related to the used laboratory equipment

After this selection, the obtained dataset is composed by only 9 features (including the class label) and looks like as in figure 1.2 (transposed for sake of presentation).

On the other hand, we also repeated the analysis with the **most relevant dataset** selected in a more formal way.

In order to understand which are the most relevant features we have used a random forest classifier. It is possible, indeed, to retrieve which is the feature weight through this algorithm because every time a new tree is built only a subset of columns is chosen. After the training phase is terminated these weights are evaluated depending on the splits order done in each tree: if a feature is selected to be used during the first split it means that it is more relevant with respect to the others. We show down here the chart related to what we have just depicted (see figure 1.3).



	0	1	2	3	4
ra	183.531	183.598	183.68	183.871	183.883
dec	0.089693	0.135285	0.126185	0.0499107	0.102557
u	19.4741	18.6628	19.383	17.7654	17.5502
g	17.0424	17.2145	18.1917	16.6027	16.2634
r	15.947	16.6764	17.4743	16.1612	16.4387
i	15.5034	16.4892	17.0873	15.9823	16.5549
z	15.2253	16.3915	16.8012	15.9044	16.6133
class	STAR	STAR	GALAXY	STAR	STAR
redshift	-8.96e-06	-5.49e-05	0.123111	-0.000110616	0.000590357

Figure 1.2. Dataset overview (filtered)

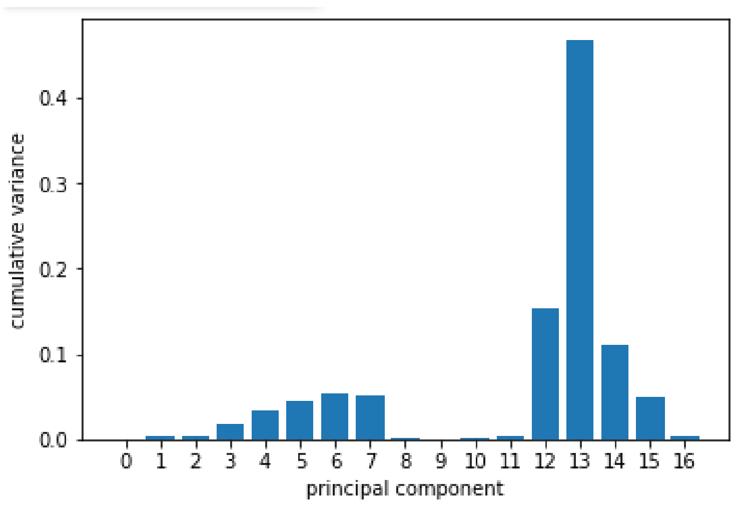


Figure 1.3. Feature relevance

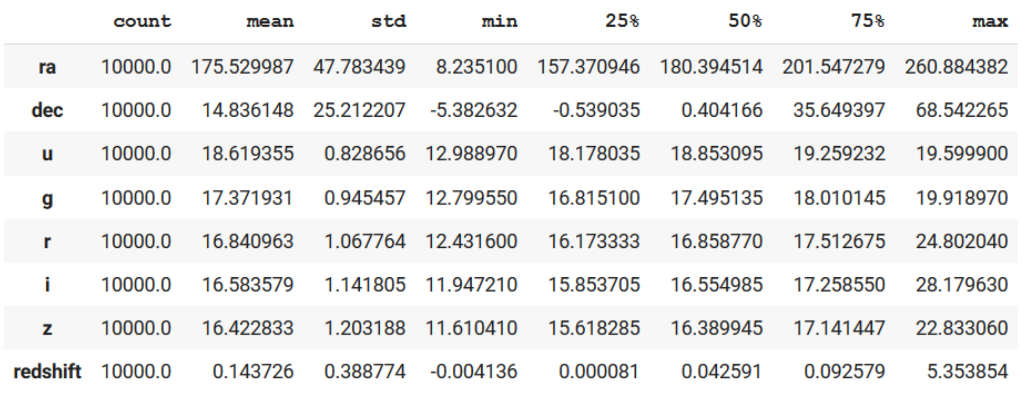
Statistical overview of the data

Considering the following relevant statistical measures:

- `count`: number of samples
- `mean`: mean of the attribute among all samples
- `std`: standard deviation of this attribute
- `min`: minimal value of this attribute
- 25%: lower percentile
- 50%: median
- 75%: upper percentile

- `max`: maximal value of this attribute

let evaluate them for each column of the dataset (except for the class label):



	count	mean	std	min	25%	50%	75%	max
ra	10000.0	175.529987	47.783439	8.235100	157.370946	180.394514	201.547279	260.884382
dec	10000.0	14.836148	25.212207	-5.382632	-0.539035	0.404166	35.649397	68.542265
u	10000.0	18.619355	0.828656	12.988970	18.178035	18.853095	19.259232	19.599900
g	10000.0	17.371931	0.945457	12.799550	16.815100	17.495135	18.010145	19.918970
r	10000.0	16.840963	1.067764	12.431600	16.173333	16.858770	17.512675	24.802040
i	10000.0	16.583579	1.141805	11.947210	15.853705	16.554985	17.258550	28.179630
z	10000.0	16.422833	1.203188	11.610410	15.618285	16.389945	17.141447	22.833060
redshift	10000.0	0.143726	0.388774	-0.004136	0.000081	0.042591	0.092579	5.353854

Figure 1.4. Dataset description

While the previous table summarizes properly the characteristics of the features, it is hard to interpret in that format. Let show some graphs in order to visualize the main trends at a glance.

First of all, let plot the distribution of the response variable between the three different classes:

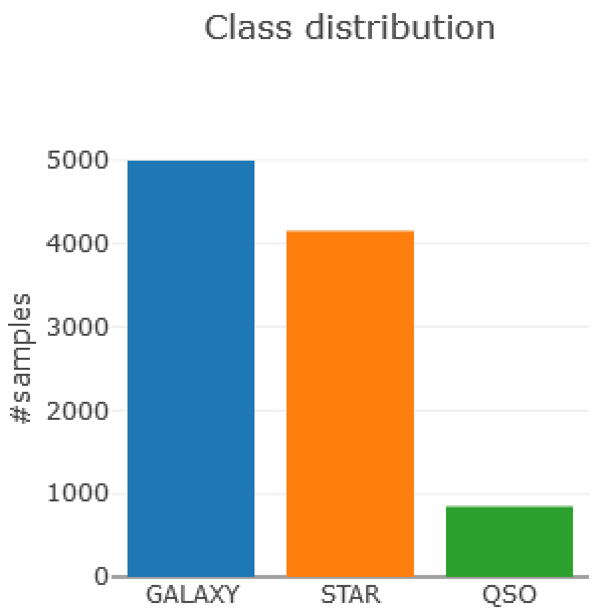


Figure 1.5. Class distribution

It is immediately possible to notice that the class distribution is not balanced: the `galaxy` and `star` classes almost split in half the 10,000 samples, leaving only the crumbs to the `quasar`. Later on (Chapter 8) we will analyse techniques to overcome this problem.

Part I

Preprocessing

Chapter 2

Principal component analysis

2.1 Theory recall

The **PCA (Principal Component Analysis)** is an algorithm that has as goal to reduce the number of dimensions keeping the highest percentage of information possible (represented by the variance). It is usually implemented because working in a high-dimensional space can lead to a problem, the so-called curse of dimensionality. This is a phenomena that involves having data very sparse and this is something that should be avoided because sparsity is an issue for algorithms that deal with statistical significance. Indeed having samples very far away among them means that the distance between any couple will be very huge and it will lose significance but distance is a key aspect for many methods.

To solve this scenario the principal component analysis is very helpful. A new set of components will be chosen such that:

- each component is orthogonal to previous ones;
- as much information as possible is preserved.

There are many different ways to implement this algorithm. One of these is based on exploiting the **variance-covariance matrix $\boldsymbol{\Sigma}$** and its **eigenvalues** and **eigenvectors**:

- given your data $\{x_1, \cdots, x_m\}$, computer the covariance matrix $\Sigma = \dfrac{1}{m}\sum_{i=1}^{m}(x_i - \bar{x}) \cdot (x - \bar{x})^T$ $\left(\text{where } \bar{x} = \dfrac{1}{m}\sum_{i=1}^{m} x_i\right)$;
- each eigenvector will be a new basis in the new data space and its importance is represented by the its eigenvalue, such that the bigger it is the more importance it has;
- pick up only the first k eigenvectors depending on the result.

It is critical to have data in the standard form because otherwise one feature, that has an order of magnitude high, could appear way more relevant than the other.

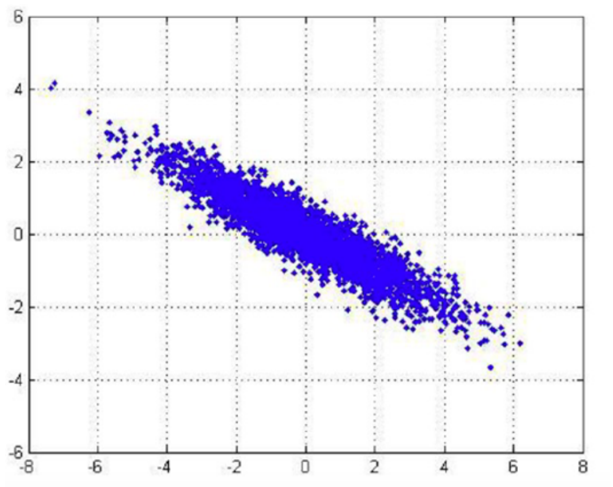


Figure 2.1. PCA - original distribution

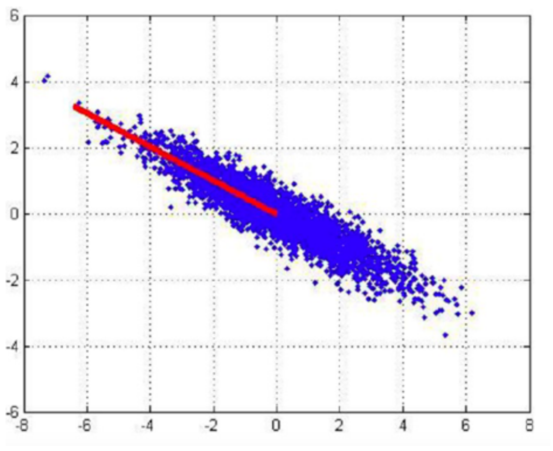


Figure 2.2. PCA - first PC

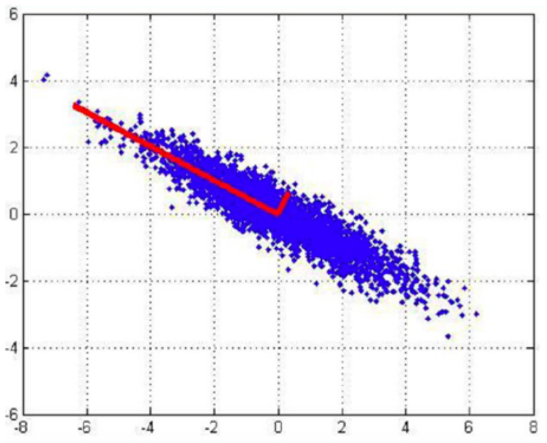


Figure 2.3. PCA - first two PCs

The number of eigenvectors to be selected is different on every problem and it can be evaluated with the help of graphs representing the variance associated to each component (see Figure 2.4).

Is it possible to notice that after a certain point each component contributes to the variance with a small percentage and that the difference from the following one is very low; that point is the **elbow point** and usually it represents a good number of component to choose, considering also that, in order to increment the percentage of variance of few points, you need to add many more new axis. Another criteria to decide the number of components is to look at the cumulative variance (dashed line in the second chart of figure 2.4) that represents how much information you are bringing on.

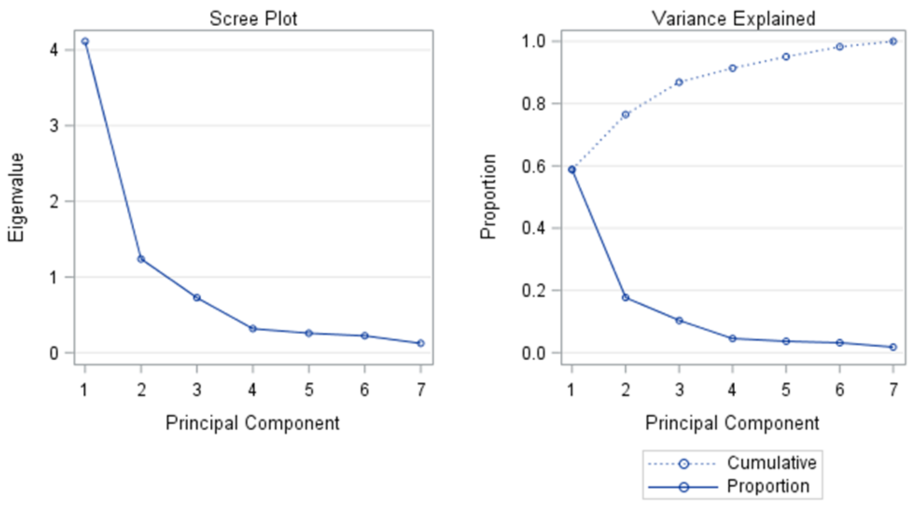


Figure 2.4. PCA - explained variance

2.2 Practical results

Cumulative variance

The cumulative sum in our graph is the following (see Figure 2.5), and we have decided to select just the first 5 components (even if the number of dimensions is not high we have chosen to implement it anyway for the sake of learning how it works and how it can be implemented).

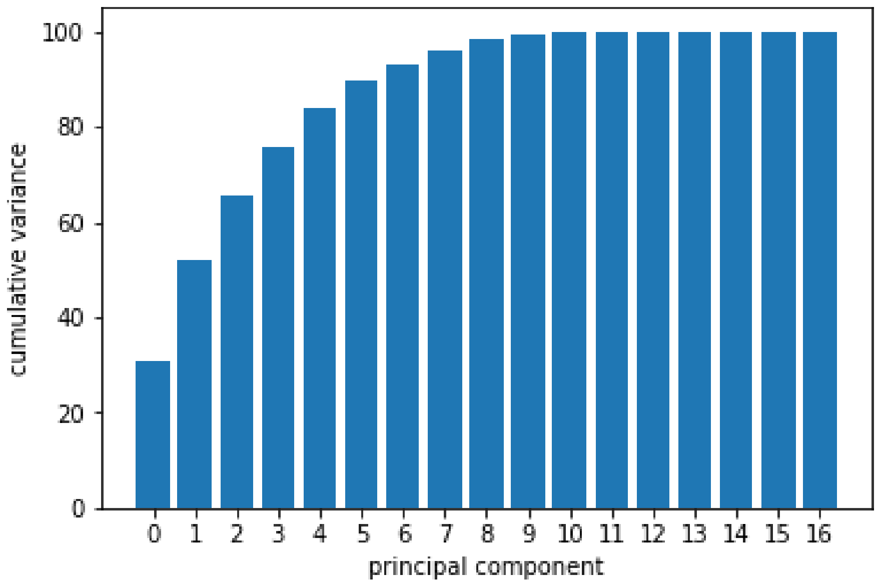


Figure 2.5. PCA - cumulative variance

First two PCs

In the following chart (see figure 2.6) we plotted the first two principal components so that it is possible to show how the data is dislocated. In our case the first two axis only bring with them only around 50% of variance and for this reason we should not expect a great result. Indeed, as we can see, points of three different classes are mixed all together, even if it is possible to recognize some patterns or some areas in which only class is present. In the most left part of the chart there are mostly green points (representing quasar records), while stars and galaxies are overlapped, creating three sort of "stripes" in the right-center part; we have assumed that this behavior could be provoked by the fact that galaxies are composed for the majority of stars and so they should appear similar one to each other.

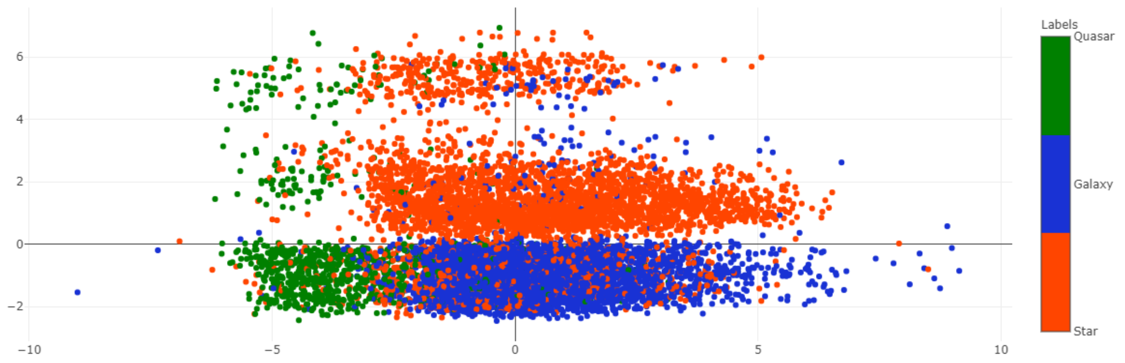


Figure 2.6. PCA - first 2 PCs

Part II

Classification

Chapter 3

k-nearest neighbours

3.1 Theory recall

The ***k*NN (*k*-nearest neighbor)** is an instance based classifier because in order to provide the result it has to scan the database, find the closest points to your sample and then assign the correct label. For this reason it requires:

- the data set with all the previous records already classified;
- a *distance metric* that is used in order to compute the distance between a couple of samples;
- the value of k neighbors to consider at each time.

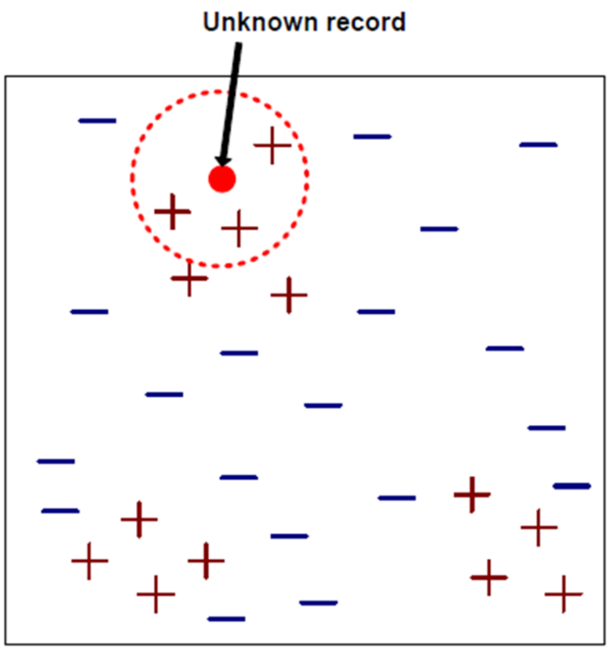


Figure 3.1. kNN - new classification

When a new sample requires to be labeled this algorithm has to evaluate all the possible distances with all the other records: for this reason this algorithm is particularly slower compared to other algorithms such as SVM in which, to obtain a result, just a simple computation is required. After this first phase the k-nearest neighbors are identified and then, using a majority vote schema, the class is decided. It is possible to assign a **weight** to each label so that they are accounted in a different way during the voting; for instance you can use the distance as a scaling factor such that closer records will be more meaningful.

Choosing parameters

While implementing this algorithm two different parameters has to be chosen carefully:

- the distance metric;
- the value of k.

The kNN is heavily based on the **distance metric**, thus running the same scenario with different metrics will lead to different results. Choosing the correct distance is often a hard task because sometimes there is not a "natural" way of defining a distance between two objects. Whenever this happens is possible to use the *Minkowski distance* (generalization of both the *Euclidean distance* and the *Manhattan distance*):

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \tag{3.1}$$

The **value of *k*** plays a big role because selecting a wrong value of it can lead to a misclassification. There are two possible mistakes:

- value of k is too large (figure 3.2): points far away are considered. Here `"x"` is labeled as `"-"` while probably is more correct to be labeled as `"+"`;
- value of k is too small (figure 3.3): the class is decided by a noise point.

Drawbacks

The kNN algorithm has scaling issues. Like every algorithm that depends on a metric, having an attribute that presents values larger than the other predictors can dominate during the computation of the distance and because of this it is suggested to normalize the data set before running the training phase.

3.2 Practical results

While analyzing our data set with the kNN algorithm we could not find any appropriate distance metric; this is one of the scenarios in which having the knowledge of an expert could be really useful. Due to this fact we have decided to take the "safest" approach: while tuning the number of k neighbors, we have also considered the grade p of the Minkowski distance as a hyperparameter to be tuned. For

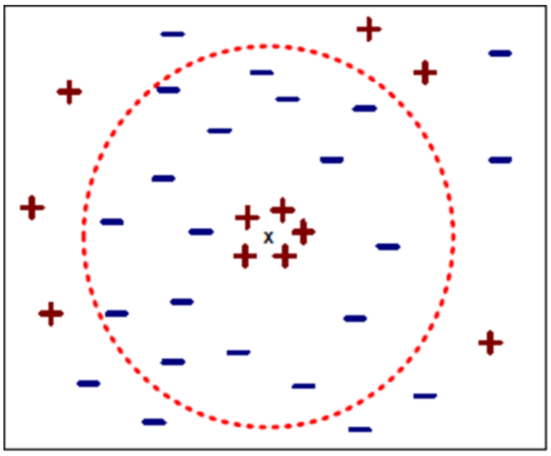


Figure 3.2. kNN - large k

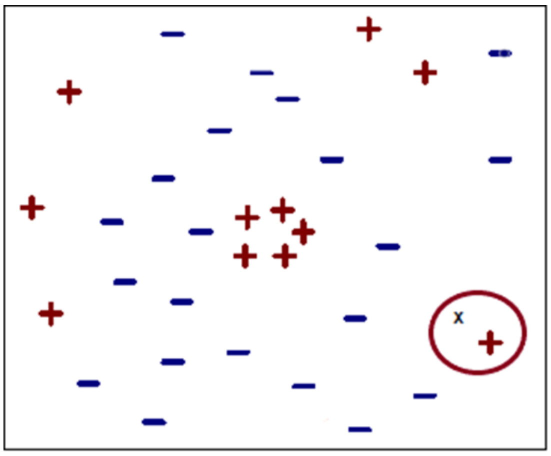


Figure 3.3. kNN - small k

this reason, we have plotted the results using a heat map to find which is the best combination of this two numbers.

To validate our models, we have used the cross-validation approach, with 10 folds.

Complete dataset

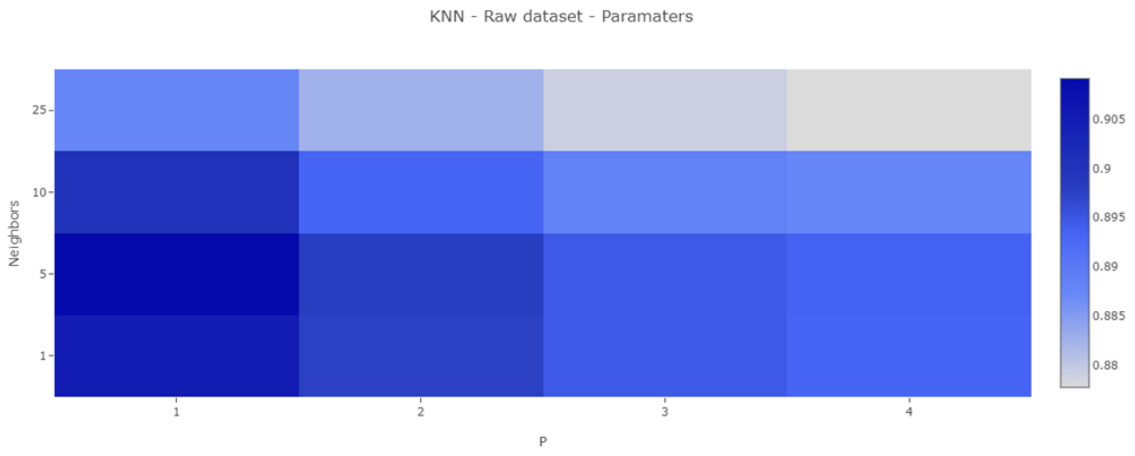


Figure 3.4. Complete dataset - hyperparameters heatmap

On the raw dataset (i.e. considering all the features), the best model has the following parameters and reaches the accuracy below:

$$\begin{cases} k_{best} = 1 \\ p_{best} = 1 \end{cases} \longrightarrow \texttt{accuracy} = 0.9177$$

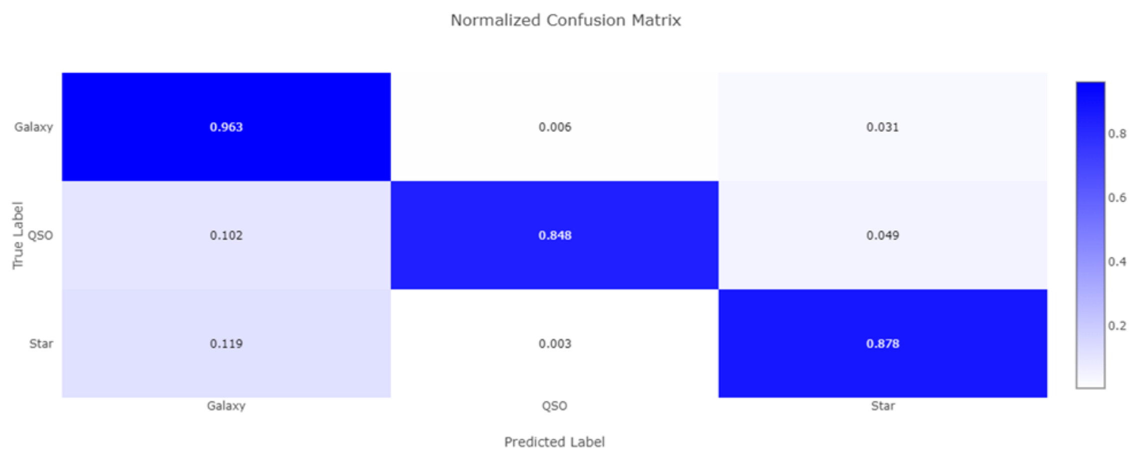


Figure 3.5. Complete dataset - confusion matrix

Most relevant dataset

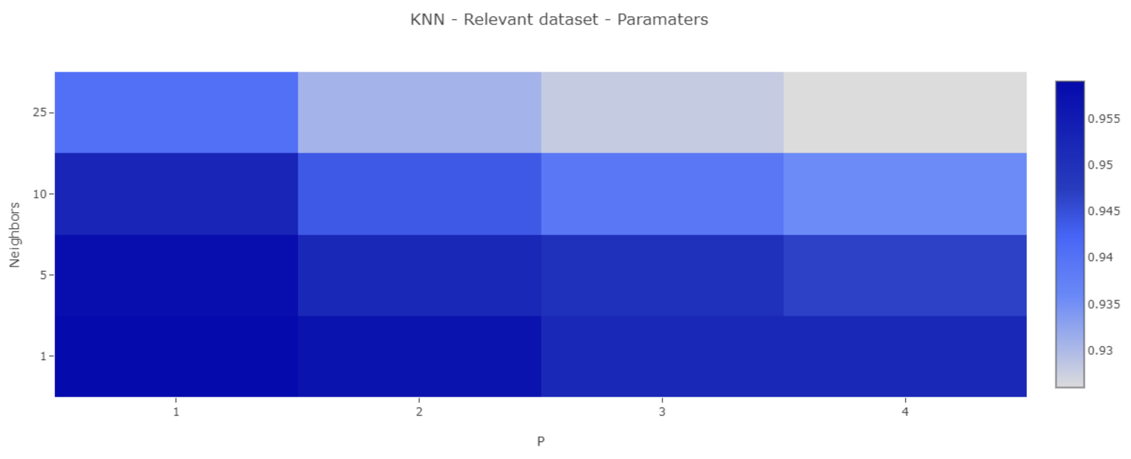


Figure 3.6. Most relevant dataset - hyperparameters heatmap

On the most relevant dataset (i.e. from random forest analysis), the best model has the following parameters and reaches the accuracy below:

$$\begin{cases} k_{best} = 1 \\ p_{best} = 1 \end{cases} \longrightarrow \texttt{accuracy} = 0.9660$$

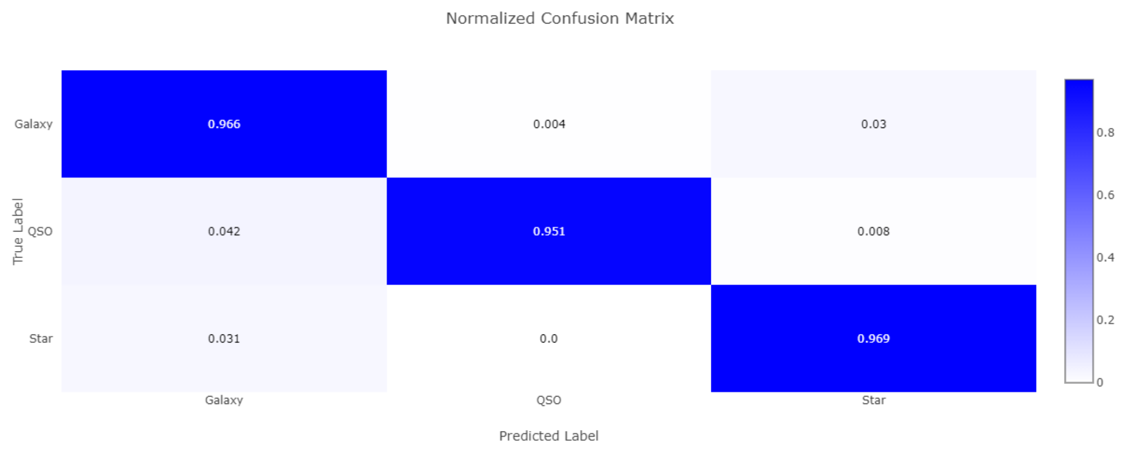


Figure 3.7. Most relevant dataset - confusion matrix

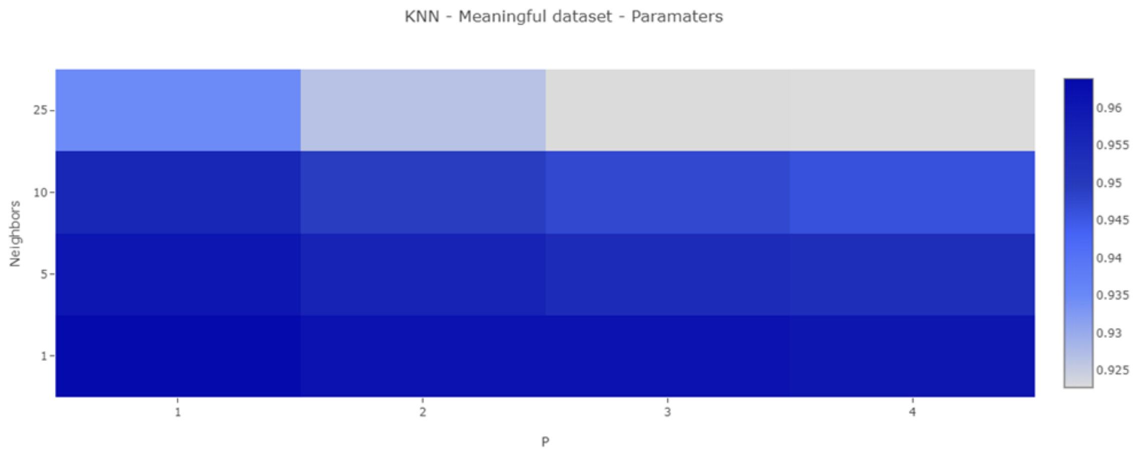


Figure 3.8. Most meaningful dataset - hyperparameters heatmap

Most meaningful dataset

On the most meaningful dataset (i.e. our feature selection intuition), the best model has the following parameters and reaches the accuracy below:

$$\begin{cases} k_{best} = 1 \\ p_{best} = 1 \end{cases} \longrightarrow \texttt{accuracy} = 0.9670$$

PCA dataset

On the PCA dataset (i.e. first 5 PCs), the best model has the following parameters and reaches the accuracy below:

$$\begin{cases} k_{best} = 10 \\ p_{best} = 1 \end{cases} \longrightarrow \texttt{accuracy} = 0.8580$$

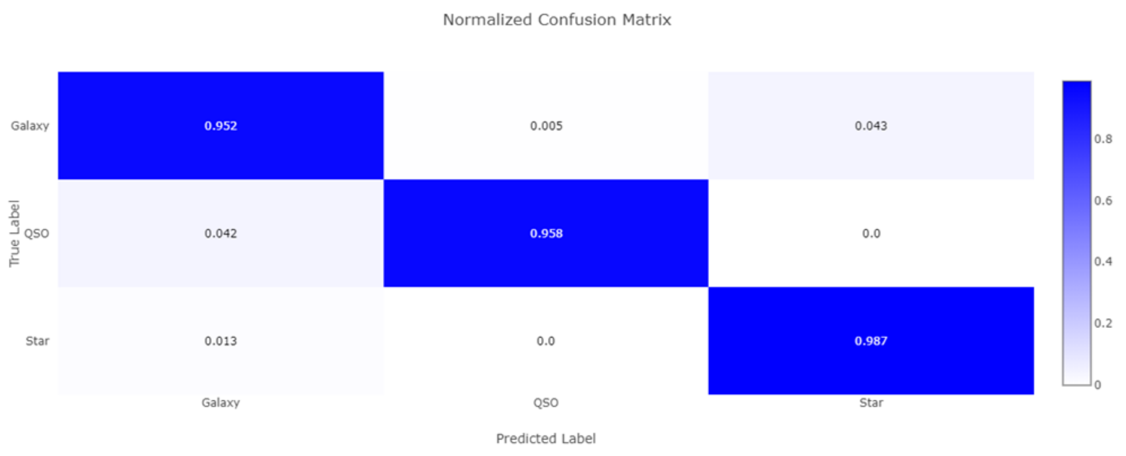


Figure 3.9. Most meaningful dataset - confusion matrix

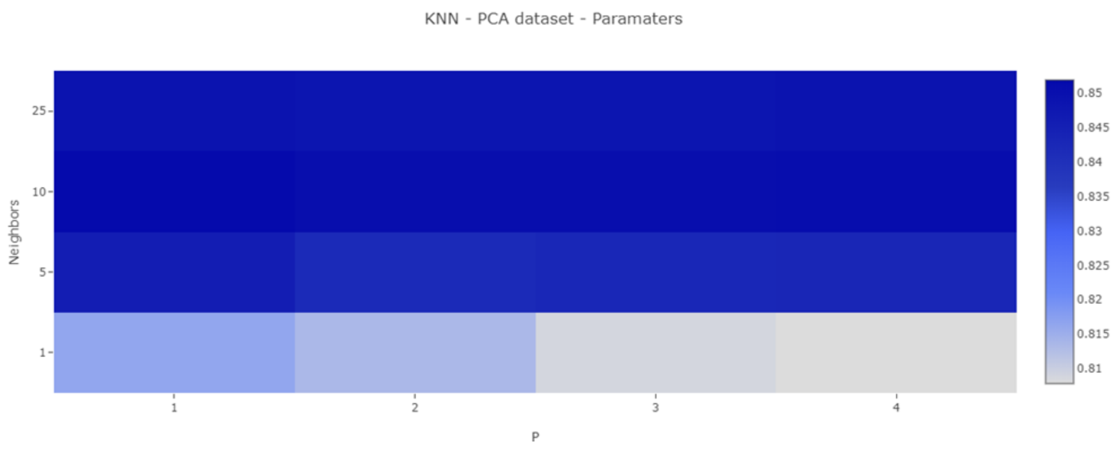


Figure 3.10. PCA dataset - hyperparameters heatmap

Conclusion

In conclusion we can observe that all the three first models returns a pretty high accuracy but most important of all is the result coming from the confusion matrix. Indeed the most important task of this work is to find and correctly identify all the records associated to a `QSO` and the second and third model are the best in this job. The model based on the PCA data performs poorly compared to the previous ones.

The best classification accuracy obtained with the kNN algorithm is 96.70%.

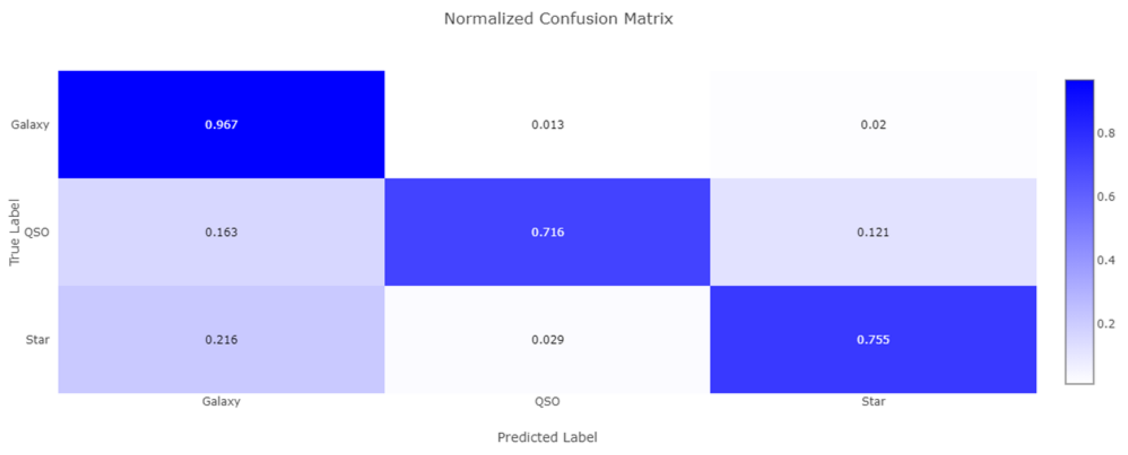


Figure 3.11. PCA dataset - confusion matrix

Chapter 4

Logistic regression

4.1 Theory recall

The **logistic model** is used to model the probability of a certain class in a binary scenario. This can also be extended to model several classes of events; each target would be assigned a probability between 0 and 1 and the sum adding to one.

The **logistic regression** is a *supervised learning* algorithm that we will use, despite its name, to classify provided samples in specified classes.

Linear regression

The **linear regression** is an algorithm able to linearly model the relationship between a *scalar response* (i.e. *dependent variable*) and an *explanatory variable* (i.e. *independent variable*).

In case there are more than one independent variables, the procedure falls under the name of *multivariate linear regression* and constitute a finer ML technique.

For sake of simplicity, we will cover the binary situation. In a *supervised learning* scenario, we already have the mapping between the explanatory variable x and the scalar response y. We also assume that those are related by a linear relationship having the following form:

$$y = w \cdot x + b \tag{4.1}$$

where the parameters have the following meaning:

- $\underline{y}$: dependent variable we want to predict
- $\underline{w}$: slope
- $\underline{x}$: independent variable
- $\underline{b}$: bias

One of the main linear regression features it is worth to underline, is that the response variable y is a **continuous feature**: this model is able to handle non-discrete values (e.g. weight, income).

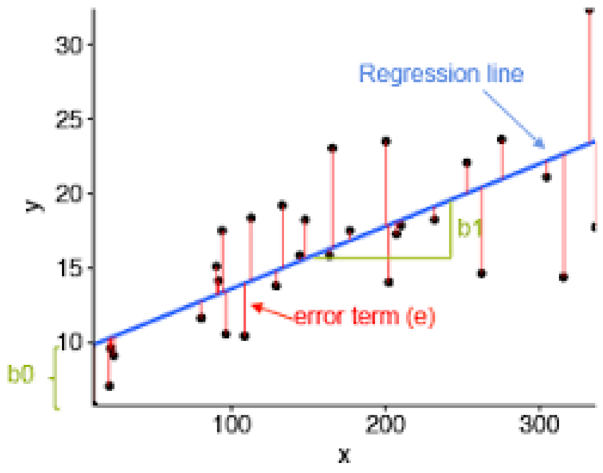


Figure 4.1. Linear regression - terminology

In order to find the best fit for the given data, we look for the line that minimizes the **least square error**, defined as follows:

$$L_2 = \sum_{i=0}^{n} (a_i - b_i)^2 \tag{4.2}$$

Applied to the current situation, the a_i term is the actual scalar response (for the given value x_i) and b_i is the estimated value ($w \cdot x_i + b$) using the current line (defined by parameters w and b). Minimizing the least square error process is formalized as follows:

$$\underset{(w,b)}{\arg\min} \sum_{i=0}^{n} [y_i - (w \cdot x_i + b)]^2 \tag{4.3}$$

that lead to the following solution for estimate the needed line parameters:

$$\begin{cases} b = \dfrac{\sum_{i=0}^{n} (y_i - w \cdot x_i)}{n} \\ w = \dfrac{\sum_{i=0}^{n} x_i \cdot (y_i - b)}{\sum_{i=0}^{n} {x_i}^2} \end{cases} \tag{4.4}$$

Once computed the interpolating line, we could exploit it to **estimate new values**: given a new sample x_i, we can assign its corresponding y_i value, by exploiting the linear model:

$$x_i \longrightarrow y_i = w \cdot x_i + b$$

This procedure is the linear regression part that is actually considered ML flavour.

Key concept

Even though the linear regression could be used for classification, it only handles continuous values; while the logistic regression is more suited for this purpose, because discriminates between a binary case {0,1}.

Instead of inferring a straight line (growing at infinity), the logistic regression draw a **S shaped curve** that models two well defined cases linked by a smooth transitional phase. The edges could be mapped to the two class label {0,1}, while the middle part represents a region of uncertainty.

The logistic regression output is bounded in [0,1], thus has a *probabilistic interpretation*.

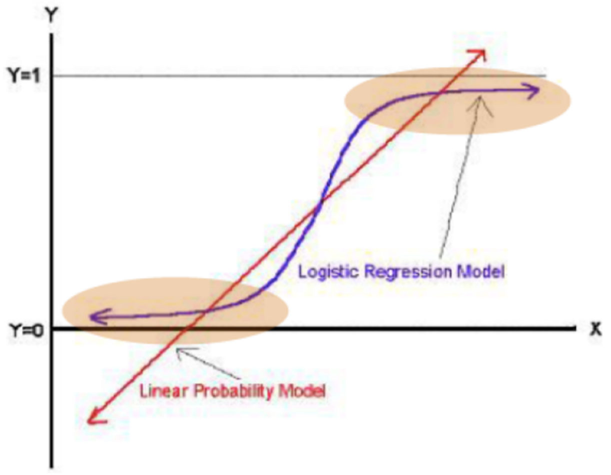


Figure 4.2. Linear regression vs logistic regression classification

Another main difference between linear regression and logistic regression is the way they are computed (see details section 4.1). While the linear regression looks for the line that minimizes the *least square error*, the logistic regression exploits **MLE (Maximum Likelihood Estimator)**: it choose the parameters θ that maximize the probability of having the observed data D.

$$\hat{\theta}_{MLE} = \arg\max_{\theta} P(D|\theta) \tag{4.5}$$

Formal model

To overcome the linear regression problems in classifications, in the logistic regression we substitute the linear function with the **sigmoid function**:

$$g(h) = \frac{1}{1+e^{-h}} \tag{4.6}$$

The general classification problem $P(y|X;\theta)$, using the sigmoid function for binary classification, becomes:

$$\begin{cases} P(y=0|X;\theta) = g(w^TX) = \dfrac{1}{1+e^{w^TX}} \\ P(y=1|X;\theta) = 1 - g(w^TX) = \dfrac{e^{w^TX}}{1+e^{w^TX}} \end{cases} \tag{4.7}$$

Note that θ parameter was substituted with w and that we considered a probabilistic scenario (i.e. sum of binary cases probability is 1).

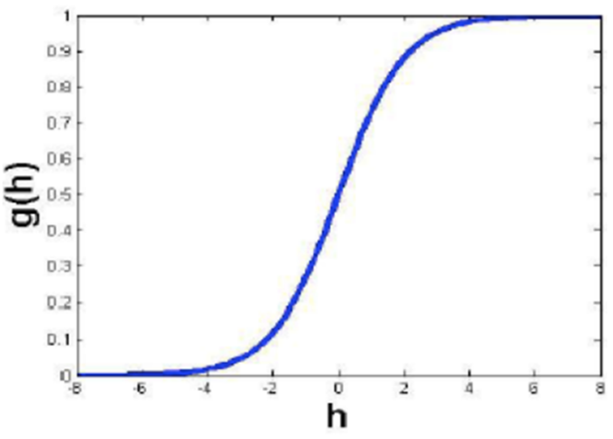


Figure 4.3. Sigmoid function

Defining the probability in terms of $P(y|X;\theta)$ do not need Bayes rule anymore. This is the main difference between logistic and linear regression.

The likelihood of the data, given the model, is the following (where $g(\cdot)$ is the sigmoid function $g(w^T X)$):

$$\mathcal{L}(y|X;w) = \prod_{i=1}^{n} (1 - g(X_i;w))^{y_i} \cdot g(X_i;w)^{(1-y_i)} \tag{4.8}$$

To maximize the likelihood, we state that its derivative w.r.t. w is zero. In order to simplify the computations, due to the exponential form of $g(\cdot)$, we can consider the **log likelihood** (i.e. Eulerian logarithm of $\mathcal{L}(y|X;w)$):

$$\begin{aligned} ll(y|X;w) &= log[\mathcal{L}(y|X;w)] \\ &= \sum_{i=1}^{n} [y_i w^T X_i - log(1 + e^{w^T X_i})] \end{aligned} \tag{4.9}$$

We considered the partial derivative w.r.t. each component w^jf the vector w of parameter to set. We need to state this quantity to zero, trying to minimize the function composed by the true label (i.e. y_i) minus the prediction of the label itself (i.e. $P(y_i = 1|X_i;w)$ thus we are minimizing the loss function:

$$\frac{\partial}{\partial w^j} ll(w) = 0 \tag{4.10}$$

The computation above has no a close form solution, but luckily it is a concave function, hence we could exploit the **gradient descend method**: an iterative optimization algorithm to find the minimum of a function.
Give a function z (having at least one minimum) defined as:

$$z = x(y - g(w;X))$$

the aim of the gradient descent algorithm is to opposite direction, with respect to the slope, in order to decrease z and find its minimum. But we need to be careful

and not to go too much, otherwise we would go beyond the optimal w.
The value of the slope is defined as follow:

$$slope = \frac{\partial z}{\partial w}$$

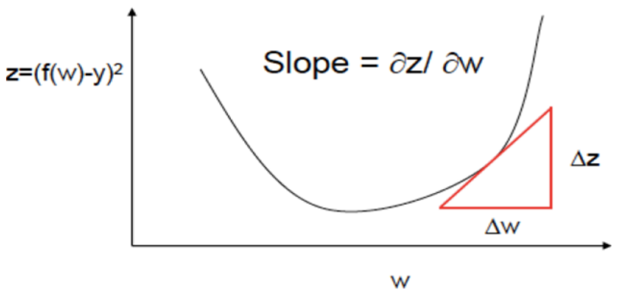


Figure 4.4. Gradient descent

4.2 Practical results

Custom tools

Being the logistic regression analysis, we created a structured way to perform it: we build a Python class (`LogisticRegression_Analysis`) providing all the needed methods, based both on:

- `CommonTools.py`
- `LogisticRegression` from `sklearn.linear_model`

The constructor acquires the `predictors` X and the related `responses` y (supervised learning flavour) and immediately splits them in training set (to infer the model), a validation set (used for parameter tuning) and a testing set (used for accuracy evaluation).

Once instantiated, a `LogisticRegression_Analysis` object provides the following facilities:

- parameters tuning:
  - penalties:
    - L1
    - L2
  - solvers:
    - liblinear
    - newton-cg
    - lbfgs
    - sag

    * saga

- computing accuracy
- plotting confusion matrix
- plotting the learning curve

Furthermore, the `LogisticRegression_Analysis` class provides standard ranges in which tuning the parameter `C` (derived from common knowledge on the field). It also stores the estimator obtained by the last training.

Complete dataset

Exploiting the raw dataset (i.e. considering all the features), the best logistic regression solver is `lbfgs` with the following parameter:

$$\text{lbfgs solver:} \begin{cases} \texttt{C} = 1000 \\ \text{penalty} = \texttt{l2} \end{cases} \longrightarrow \texttt{accuracy} = 0.9900$$

The confusion matrix (see figure 4.5) shows that the stars are predicted perfectly, the galaxies are identified with high precision, while the smallest accuracy (still a meaningful 96%) is on the quasars that have the smallest number of samples on which being trained on.

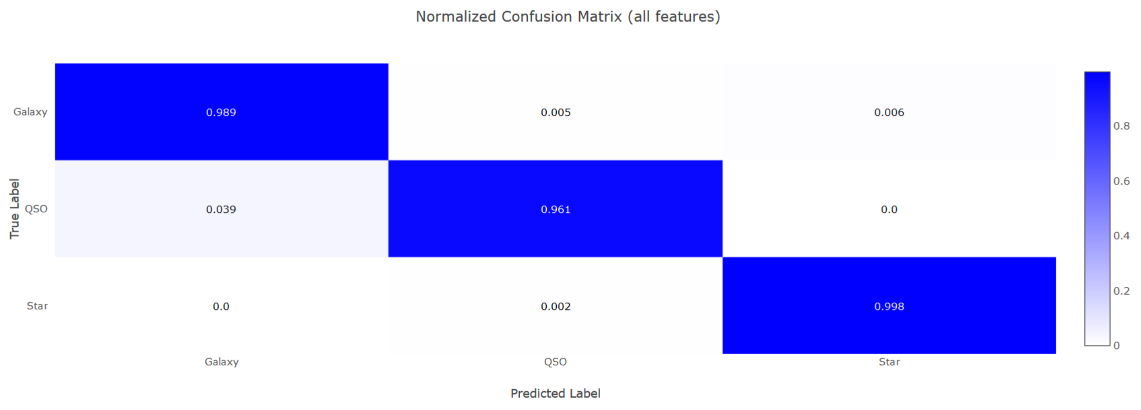


Figure 4.5. Confusion matrix - complete dataset

Most relevant dataset

Exploiting the most relevant dataset (i.e. from random forest analysis), the best logistic regression solver is still the `lbfgs`:

$$\text{lbfgs solver:} \begin{cases} \texttt{C} = 1000 \\ \text{penalty} = \texttt{l2} \end{cases} \longrightarrow \texttt{accuracy} = 0.9900$$

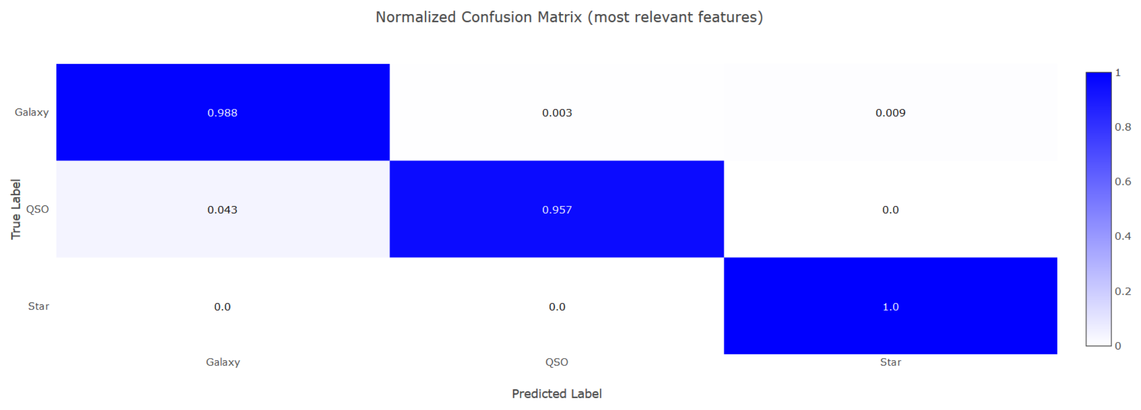


Figure 4.6. Confusion matrix - most relevant dataset

Most meaningful dataset

Exploiting the most meaningful dataset (i.e. our feature selection intuition), the best logistic regression solver is still the `lbfgs`, but with slightly worse results:

$$\text{lbfgs solver:}\quad \begin{cases} \texttt{C} = 1000 \\ \text{penalty} = \texttt{l2} \end{cases} \quad \longrightarrow \texttt{accuracy} = 0.9880$$

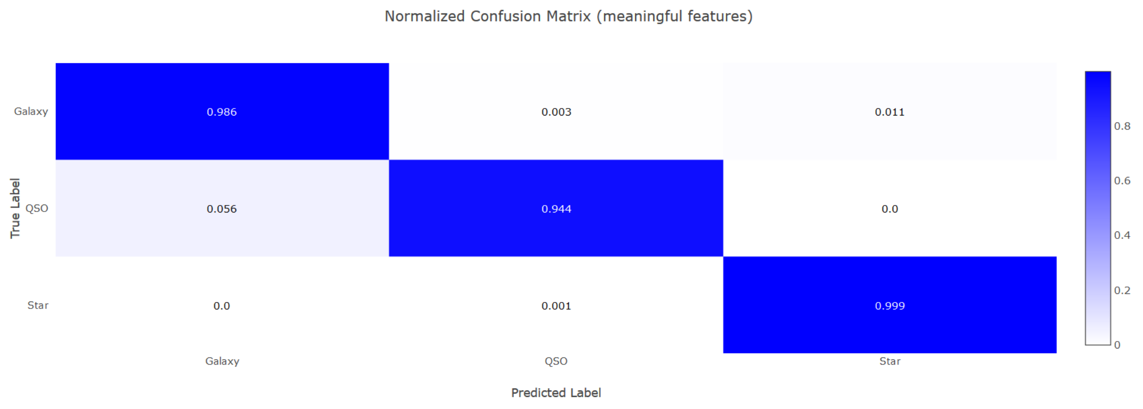


Figure 4.7. Confusion matrix - most meaningful dataset

PCA dataset

The PCA dataset (i.e. first 5 PCs) is the only one exploiting a different solver, but with worse results:

$$\text{saga solver:}\quad \begin{cases} \texttt{C} = 100 \\ \text{penalty} = \texttt{l1} \end{cases} \quad \longrightarrow \texttt{accuracy} = 0.8466$$

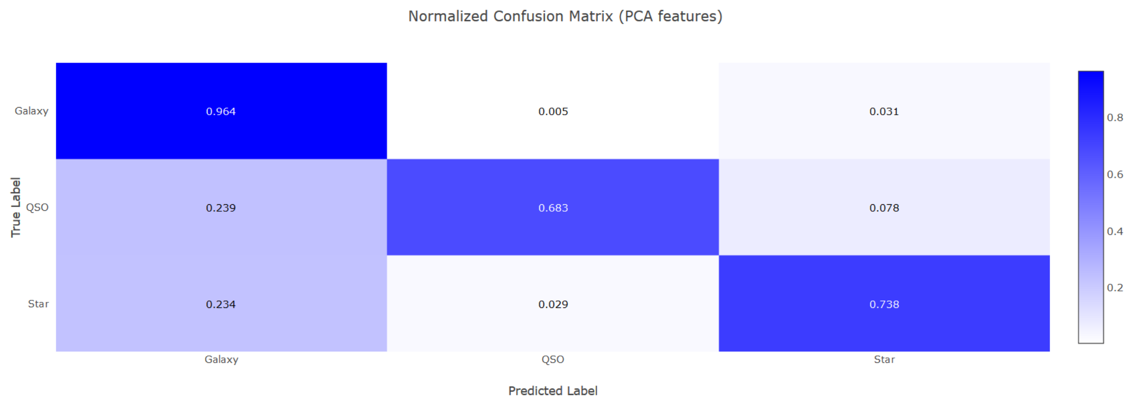


Figure 4.8. Confusion matrix - PCA dataset

Conclusions

Considering the repeated analysis with different solvers, in most of the cases the best optimization algorithm is the `lbfgs` one. Only the PCA dataset differs, because lives in a different feature space.

On the other side, considering the repeated analysis with different datasets, we can state that (for the logistic regression application) there are no relevant differences between considering or discarding the selected features, but considering only the first 5 PCs decreases significantly the performances.

The best classification accuracy obtained with the logistic regression algorithm is 99.00%.

Chapter 5

Support vector machines

5.1 Theory recall

The **SVM (Support Vector Machine)** is a *supervised learning* algorithm able to linearly classify the provided dataset.

It is also able to work in a *multinomial classification* scenario, but for sake of simplicity, we will start considering the *binomial classification* case. The multi-class situation will be considered in the specified section (chapter 5.1).

In order to handle also the most complex situations, the SVM works also on the non-linearly separable dataset, exploiting the *kernel trick* (chapter 5.1).

Key concept

While the *perceptron* provides infinite possible separator hyperplanes (image 5.1), the SVM basic idea is to choose the best possibility among those ones.

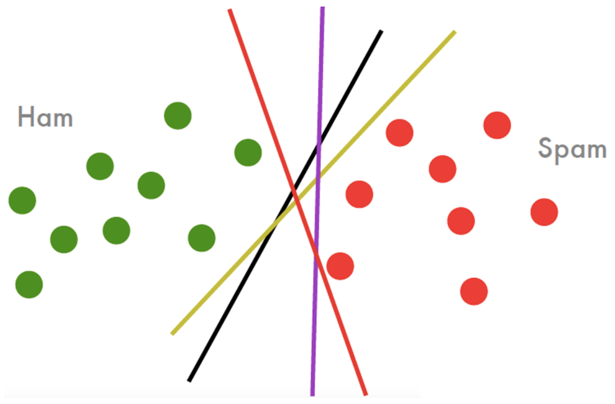


Figure 5.1. Perceptron - separating hyperplanes

Therefore, the SVM consist in an *optimization problem*: pick up the best separating hyperplane $\vec{w}$ that maximizes the **margin** ρ.

The choice of selecting the $\vec{w}$ that is as far as possible from the two classes (at training time) provide the best possible robustness against noise and outliers (at testing time). See image 5.2.

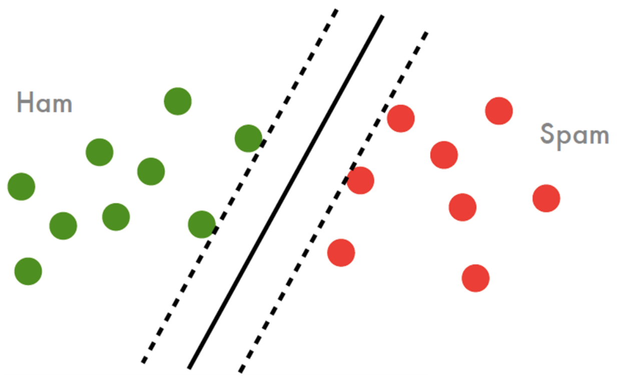


Figure 5.2. SVM - maximize the margin

Formal model

The *perceptron* algorithm, for each data point $\vec{x_i}$ provides a classification guess $f(\vec{x_i})$, given by a decision function having the following form:

$$f(\vec{x_i}) = \sigma(\langle \vec{w}, \vec{x_i} \rangle + b) \tag{5.1}$$

where the parameters have the following meaning:

- $\underline{\vec{x_i}}$: sample data point
- $\underline{f(\cdot)}$: decision function
- $\underline{\vec{w}}$: orthogonal vector describing the separating hyperplane
- $\underline{b}$: bias term (linear offset)
- $\underline{\sigma(\cdot)}$: non-linear function (e.g. *step function* or *sigmoid function*)

This result should be compared to the actual class label y_i (if available) to determine if the classification is correct or not (classifier accuracy).

The SVM decision function has the same form of the perceptron one, but the former aims to maximize the given quantity.

The separating hyperplane creates different regions, according to the result of $f(\vec{x})$:

$$\begin{cases} \langle \vec{w}, \vec{x} \rangle + b = 0 & \text{hyperplane} \\ \langle \vec{w}, \vec{x} \rangle + b \leq -1 & \text{class } c_1 \\ \langle \vec{w}, \vec{x} \rangle + b \geq +1 & \text{class } c_2 \end{cases} \tag{5.2}$$

Being SVM an optimization problem about maximizing the margin and being the margin defined by the closest data points, the SVM itself is defined by those vectors. Due to their importance, they gain a specific name: **Support Vectors**, because they "support" the classification model (image 5.3).

Learning the SVM could be formulated as the following constrained optimization problem:

$$\max_{\vec{w},b} \frac{1}{||\vec{w}||} \text{ subject to } y_i[\langle \vec{x_i}, \vec{w} \rangle + b] \geq 1 \tag{5.3}$$

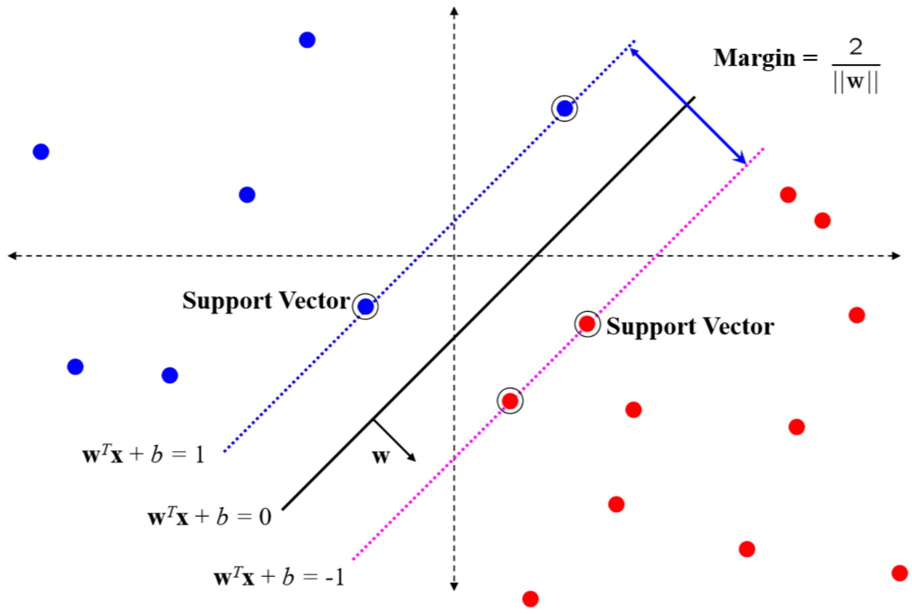

Margin = $\frac{2}{||\mathbf{w}||}$
Support Vector
Support Vector
$\mathbf{w}^T\mathbf{x} + b = 1$
$\mathbf{w}^T\mathbf{x} + b = 0$
$\mathbf{w}^T\mathbf{x} + b = -1$
$\mathbf{w}$

Figure 5.3. SVM - terminology

or equivalently:

$$\min_{\vec{w},b} ||\vec{w}||^2 \text{ subject to } y_i[\langle \vec{x}_i, \vec{w}\rangle + b] \geq 1 \tag{5.4}$$

that is a quadratic optimization problem, subject to linear constraints and there is a unique minimum.

This *primal problem* could be not trivial to solve, thus we can move to the *dual problem* thanks to the *duality principle* and then solve it thanks to the *Lagrangian multipliers*.

Multi-class SVM

Generally speaking, in classification tasks there are two main problem types, according to the number of involved classes to distinguish:

- binary classification: two target classes only, $y_i = \{+1, -1\}$
- multinomial classification: three or more target classes

The SVM, as well as many other classification algorithm, is well suited for the simplest situation (binary classifier), because it is also the one providing mathematical certainties.

To extend this safe ground to the situation of having many classes (e.g. the SDSS dataset), there are two main approaches. Both of them rely on the rationale of returning to a binary scenario:

- **One-vs-One** (OvO): train $K(K-1)/2$ binary classifiers (for a K-way multiclass problem); each receives the samples of a pair of classes from the original training set, and must learn to distinguish these two classes
- **One-vs-All** (OvA): train a single classifier per class, with the samples of that class as positive samples and all other samples as negatives

Kernel SVM

The vanilla SVM algorithm is able to handle linear cases, but fails when the classes are not linearly separable. A typical example of non-linearity is the *XOR problem* (image 5.4) in which there are no possible linear separating hyperplanes that lead all plus/minus to the same side.

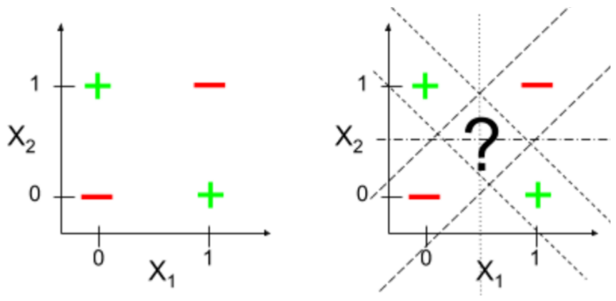


Figure 5.4. XOR problem

To properly address those situations, we can perform a mapping to an higher-dimensional space in which the points are linearly separable. The *mapping function* is formally defined as:

$$\begin{aligned} \phi : \quad & \mathbb{R}^n \to \mathbb{R}^m \quad \text{with } m > n \\ & x \to \phi(x) \end{aligned} \tag{5.5}$$

that defines a **feature space** $\phi(x)$ in which we can replace the scalar product $\langle x, x' \rangle$ with its featured version $\langle \phi(x), \phi(x') \rangle$ (image 5.5).

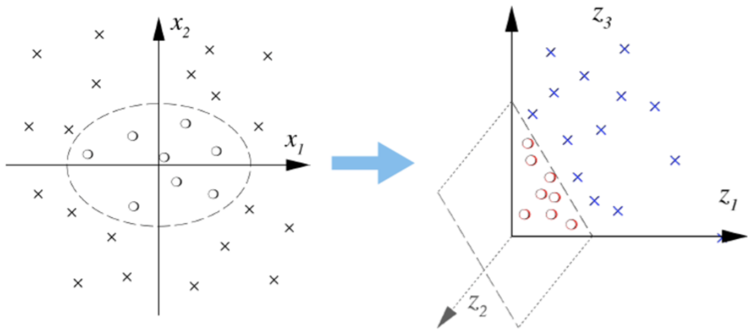


Figure 5.5. Quadratic features

In order to improve mapping performances, it is possible to rely on the **kernel trick**: making the algorithm nonlinear by substituting the scalar products $\langle \phi(x), \phi(x') \rangle$ with a kernel function $k(x, x')$ that represents a scalar product in some other higher dimensional spaces (not caring which it space is). Thus we bypass the mapping $\phi(\cdot)$ (no need for finding it) focusing only on the kernel function.

$$\langle x, x' \rangle \xrightarrow{\text{feature map}} \langle \phi(x), \phi(x') \rangle \xrightarrow{\text{kernel function}} k(x, x')$$

More formally, a **kernel function** $k(\cdot)$ is a symmetric function in its arguments:

$$k: \quad \mathcal{X} \times \mathcal{X} \to \mathbb{R} \tag{5.6}$$

and for some feature maps $\phi(\cdot)$ holds the following property:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \tag{5.7}$$

In order to create a **kernel SVM** we could exploit several kernel functions. The ones used during this work are the following:

- linear: $k(x, x') = \langle x, x' \rangle$
- polynomial: $k(x, x') = (\gamma \cdot \langle x, x' \rangle + r)^d$ with $c \geq 0, d \in \mathbb{R}$
- rbf: $k(x, x') = e^{-\gamma \cdot ||x - x'||^2}$
- sigmoid: $k(x, x') = \tanh(\gamma \cdot \langle x, x' \rangle + c)$

The correspondence between the math coefficients and the actual `sklearn` parameters is summarized in table 5.1.

Formulas:	`sklearn`:
d	`degree`
r	`coef0`
γ	`gamma`

Table 5.1. Coefficients correspondence

The table 5.2 lists the coefficients relevant for each kernel:

	`C`	`gamma`	`coef0`	`degree`
linear	✓			
rbf	✓	✓		
sigmoid	✓	✓	✓	
polynomial	✓	✓	✓	✓

Table 5.2. Kernel coefficients

5.2 Practical results

Custom tools

Being the SVM analysis, we created a structured way to perform it: we build a Python class (`SVM_Analysis`) providing all the needed methods, based both on:

- `CommonTools.py`

- `svm` from `sklearn`

The constructor acquires the **`predictors`** X and the related **`responses`** y (supervised learning flavour) and immediately splits them in training set (to infer the model), a validation set (used for parameter tuning) and a testing set (used for accuracy evaluation).

Once instantiated, a `SVM_Analysis` object provides the following facilities:

- parameters tuning:
  - linear kernel
    - one vs one
    - one vs all
  - rbf kernel
  - sigmoid kernel
  - polynomial kernel
  - among all kernels
- computing accuracy
- plotting confusion matrix
- plotting the learning curve

Furthermore, the `SVM_Analysis` class provides standard ranges in which tuning the parameters (derived from common knowledge on the field). It also stores the estimator obtained by the last training.

Complete dataset

Exploiting the raw dataset (i.e. considering all the features), the best SVM kernel is the linear one (one-vs-one fashion) with the following parameters:

$$\text{linear kernel: } \texttt{C} = 1000 \longrightarrow \texttt{accuracy} = 0.9943$$

The confusion matrix (see figure 5.6) shows that the stars are predicted perfectly, the galaxies are identified with high precision, while the smallest accuracy (still a meaningful 96%) is on the quasars that have the smallest number of samples on which being trained on.

Most relevant dataset

Exploiting the most relevant dataset (i.e. from random forest analysis), results remains the same. Refer to the previous section (see section 5.2).

Most meaningful dataset

Also analysing the the most meaningful dataset (i.e. our feature selection intuition), the considerations are exactly the same as the two cases before(see section 5.2).

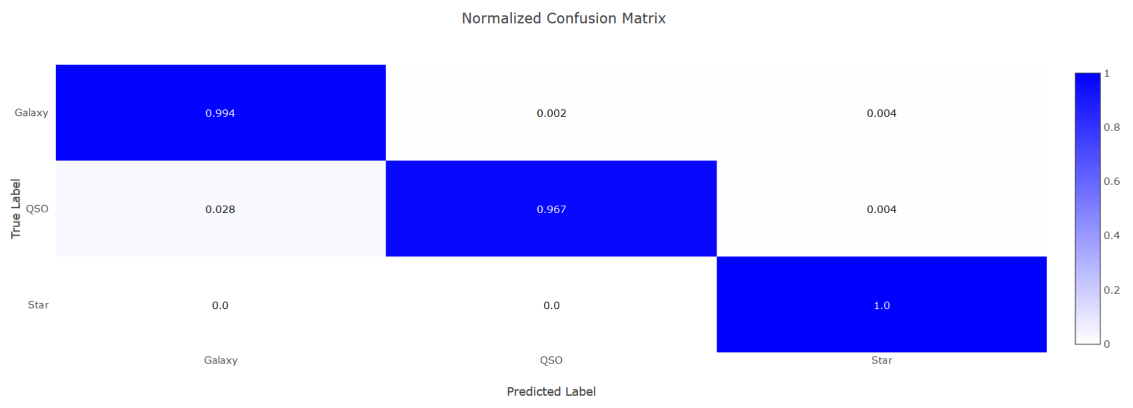


Figure 5.6. Confusion matrix - complete dataset

PCA dataset

Starting from the PCA dataset (i.e. first 5 PCs), the best SVM kernel is the RBF one with the following parameters:

$$\text{RBF kernel: } \begin{cases} \texttt{C} = 1000 \\ \texttt{gamma} = 0.001 \end{cases} \longrightarrow \texttt{accuracy} = 0.9943$$

Despite the changes in the applied kernel, the the confusion matrix (see figure 7.1) remains the same and so do the related considerations.

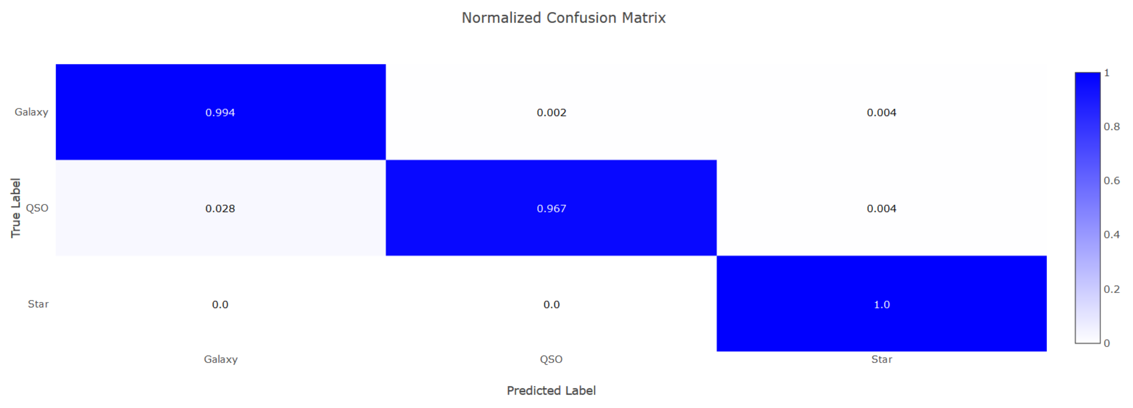


Figure 5.7. Confusion matrix - PCA dataset

Conclusions

Considering the repeated analysis with different kernels, in most of the cases the best estimator is the linear one. This could mean that the dataset is probably linearly separable (besides PCA case in which the projection space is no longer linear).

On the other side, considering the repeated analysis with different datasets, we can state that (for the SVM application) there are no relevant differences between considering or discarding the selected features.

The best classification accuracy obtained with the SVM algorithm is 99.43%.

Chapter 6

Random forest

6.1 Theory recall

Random forest is an algorithm that comes from the classification tree one and it implements solutions in order to overcome all the problems related to it. Indeed, whenever we decide to exploit classification trees, we need to take in consideration many different issues:

- at each step of the iterations during the building phase, the algorithm has to pick up which is the best attribute of the data set to be used during the split. This choice will lead to have more homogeneous nodes (they will hold samples belonging to the same class) depending on the measure of impurity that has been implemented, such as *entropy GINI index*. This choice is the key point of the algorithm and it is really complex because it is possible to have an infinite number of possible splits and it can be performed on different attributes or group of attributes. Because of this usually the approach that is adopted is greedy: the cost of training is reduced but it may lead to a non-optimal solution;

- its performance is affected by missing values: if the algorithm has never seen a particular instance then this will be most likely be misclassified because of the way in which the tree has been built;

- the basic stopping criteria is to have nodes containing only samples of the same class. This represents a scenario of an overfitted solution because we have an error on the training set equals to 0 (each record will be classified correctly) but the error on the test set will be very large (we are only label to address data that we have already seen). To overcome this problem it is possible to stop the expansion of the tree before it becomes a fully-grown tree: the algorithm will be stopped as soon as every node contains just a few samples or whenever the following split does not provide a big improvement. There is also the possibility to perform this operation after the tree has been completed "trimming" the nodes in a bottom-up fashion. This operation is fundamental because number of instances at the leaf nodes could be too small to make any statistically significant decision.

So the main problem related to trees is that they strongly depends on the dataset and on the boundary decision: adopting a greedy algorithm implies that running multiple times the same algorithm will always lead to the same final solution.

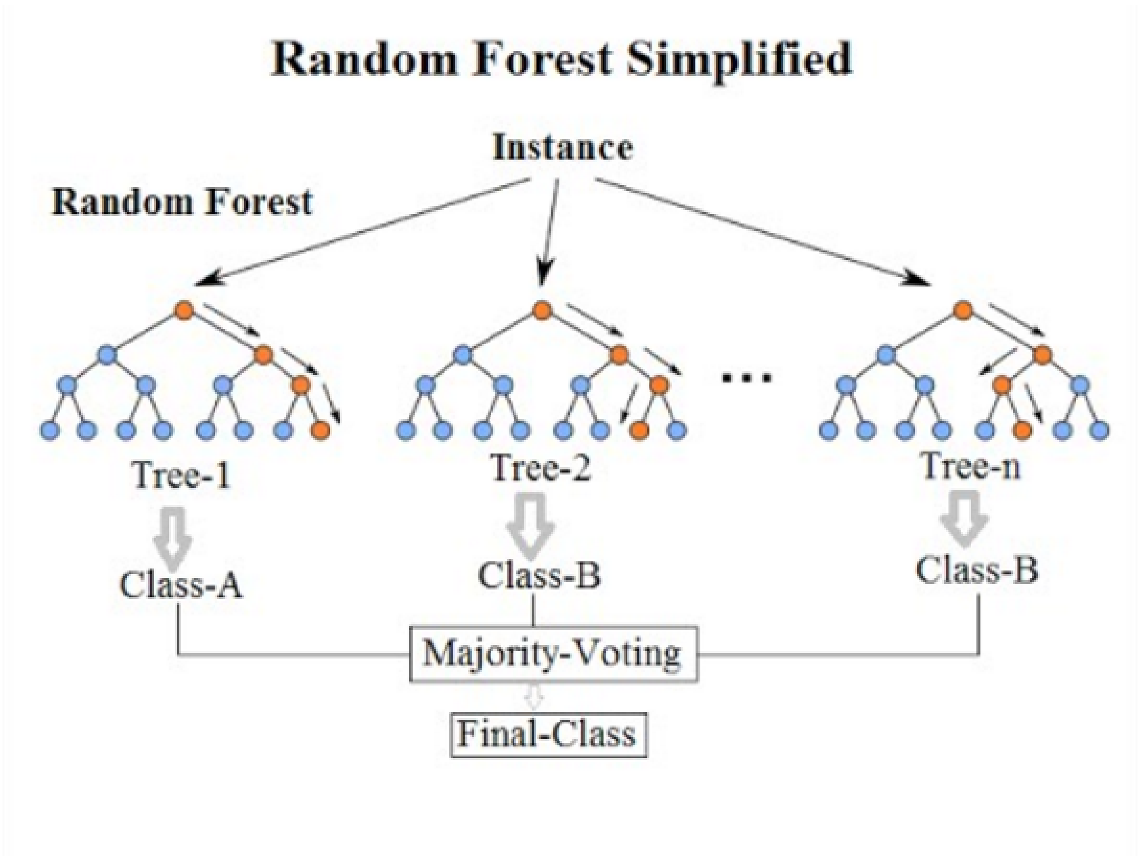


Figure 6.1. Random forest

Improvement techniques

In order to obtain a better result, we have decided to implement a random forest: combining the results coming from multiple trees allows to achieve a better solution. All these trees are trained taking in account two important aspects:

- **bagging**: instead of running the training phase always on the same data set, each time every tree will use a different group of samples obtained using **bootstrap**. In this way every tree will be generated using a different data set and so all the split decisions will be different;
- **decorrelation**: every tree, instead of being built using all the predictors, will use a random selection of them, each time different. The number of predictors that will be chosen is equals to the square root of the total number. Applying this decision will solve the problem of having a particular feature that is dominant with respect to the others because it won't be present in

every tree; in this way is it possible to understand how the tree will grow in absence of this predictor and how the other predictors are correlated among them.

In order to understand which will be the final class to assign to a sample a *majority vote schema* is applied: each tree will provide a class and the one that has received the biggest number of votes will be chosen as the final result.

Estimators

Random forest, due to its implementation, allows to use the **out-of-the bag error estimation**.

Thanks to the main feature of the bagging process, that is picking each time a random subset of samples, it is possible to directly test the performance of your model on the records that are not involved in the training phase. Observations not involved during the fit of the algorithm are referred as the *out-of-bag (OOB) observations*.

Drawbacks

Random forest has a minor drawback that is a lost of interpretability with respect to the classification tree. Classification trees are really easy to read and to explain, just following all the splits will immediately tell you which is the final result, without doing any computation. While implementing random forest we should read a large number of trees and then decide on their vote which will be the class chosen for the record; this task is very long to compute by a human being and also when the number of trees become very large, the space required to represent them is too waste.

6.2 Practical results

Complete dataset

Exploiting the raw dataset (i.e. considering all the features), the best random forest parameters are the following:

$$\begin{cases} \texttt{\#trees} = 50 \\ \texttt{\#features} = 4 \end{cases} \longrightarrow \texttt{accuracy} = 0.9903$$

Most relevant dataset

Exploiting the most relevant dataset (i.e. from random forest analysis), the best random forest parameters are the following:

$$\begin{cases} \texttt{\#trees} = 100 \\ \texttt{\#features} = 5 \end{cases} \longrightarrow \texttt{accuracy} = 0.9910$$

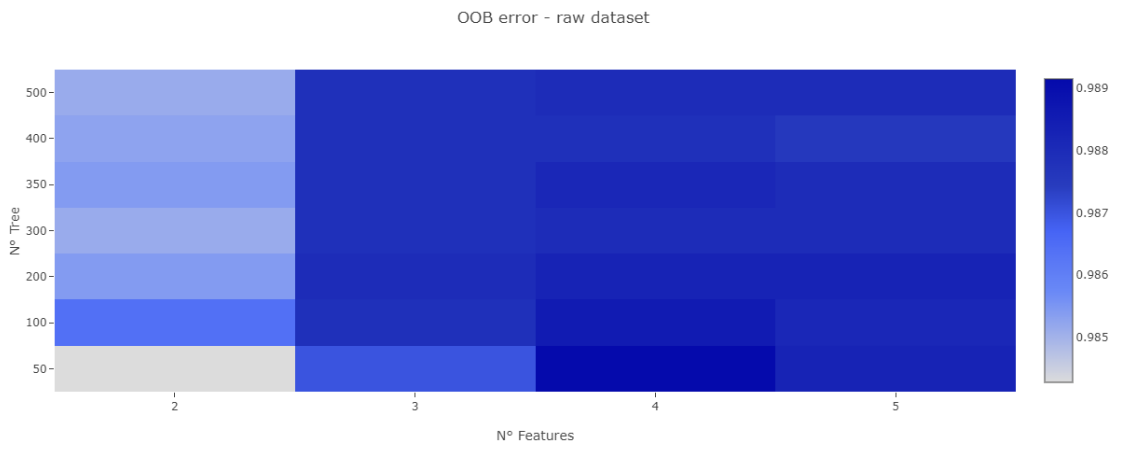


Figure 6.2. Heatmap - complete dataset

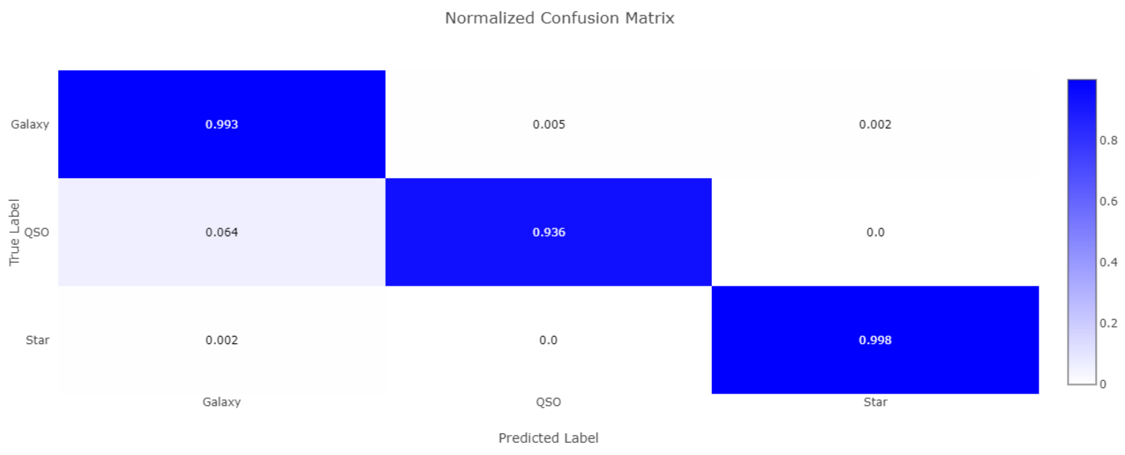


Figure 6.3. Confusion matrix - complete dataset

Most meaningful dataset

Exploiting the most meaningful dataset (i.e. our feature selection intuition), the best random forest parameters are the following:

$$\begin{cases} \texttt{\#trees} = 400 \\ \texttt{\#features} = 5 \end{cases} \longrightarrow \texttt{accuracy} = 0.9930$$

PCA dataset

The PCA dataset (i.e. first 5 PCs) has noticeable different performances:

$$\begin{cases} \texttt{\#trees} = 350 \\ \texttt{\#features} = 3 \end{cases} \longrightarrow \texttt{accuracy} = 0.8713$$

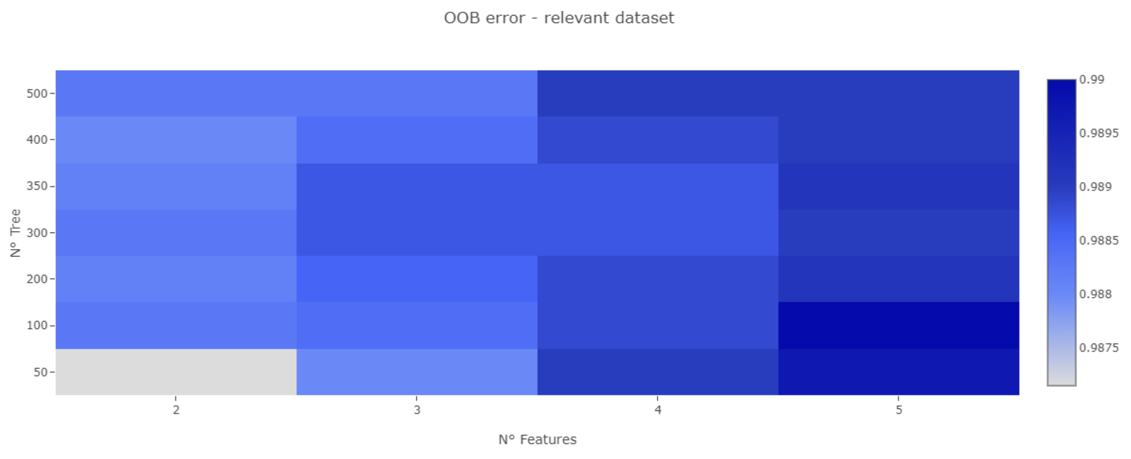


Figure 6.4. Heatmap - most relevant dataset

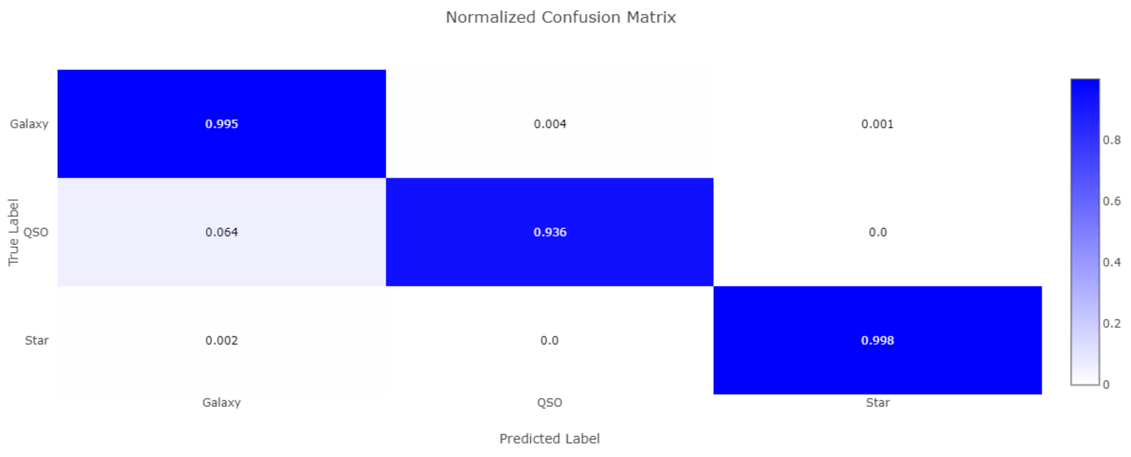


Figure 6.5. Confusion matrix - most relevant dataset

Conclusions

Random forest has proven to be a good solution for analyzing this dataset, indeed the first three cases score very well, with a precision that goes over the 99%. As usual, the only model that does not perform well is the PCA one, with a precision inferior than the 90%. Hyperparameters among the different implementations are pretty different, mostly in the number of trees generated. On top of this it is possible to adopt a different solution depending on your need: having 50 trees (in the first case) can be manually interpreted easily, compared to the last scenario where we need to generate 400 different trees; also while testing, having a smaller number of estimators can speed up the process, trading off a small amount of precision (tenth of percentage).

The best classification accuracy obtained with the random forest algorithm is 99.30%.

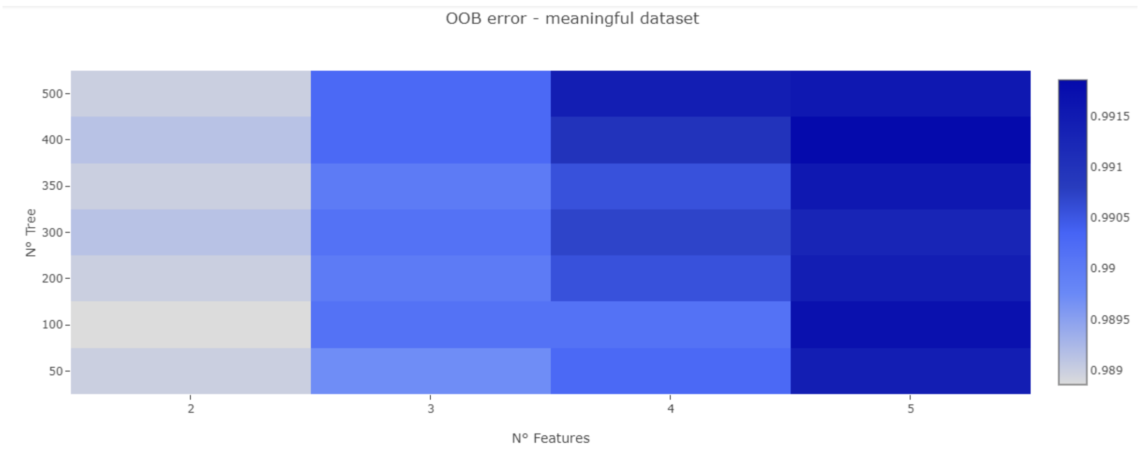


Figure 6.6. Heatmap - most meaningful dataset

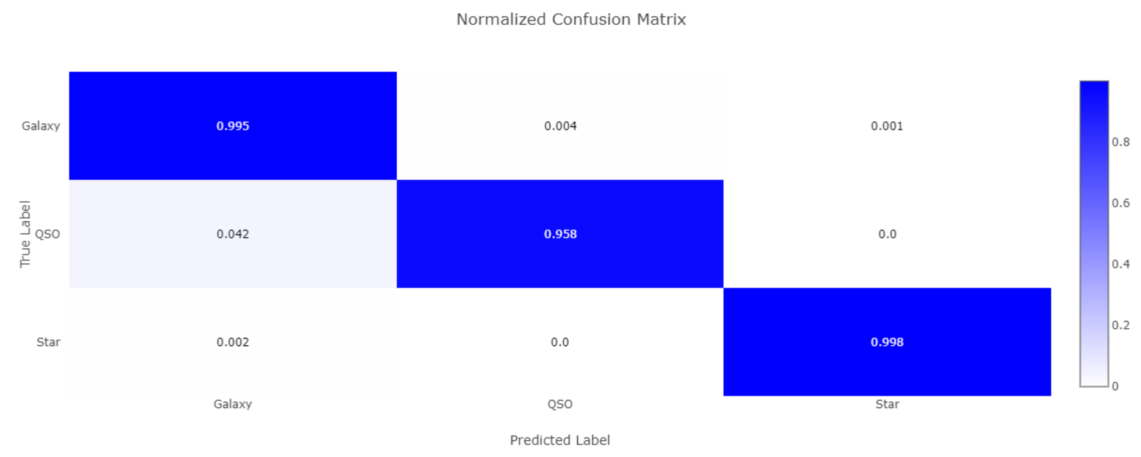


Figure 6.7. Confusion matrix - most meaningful dataset

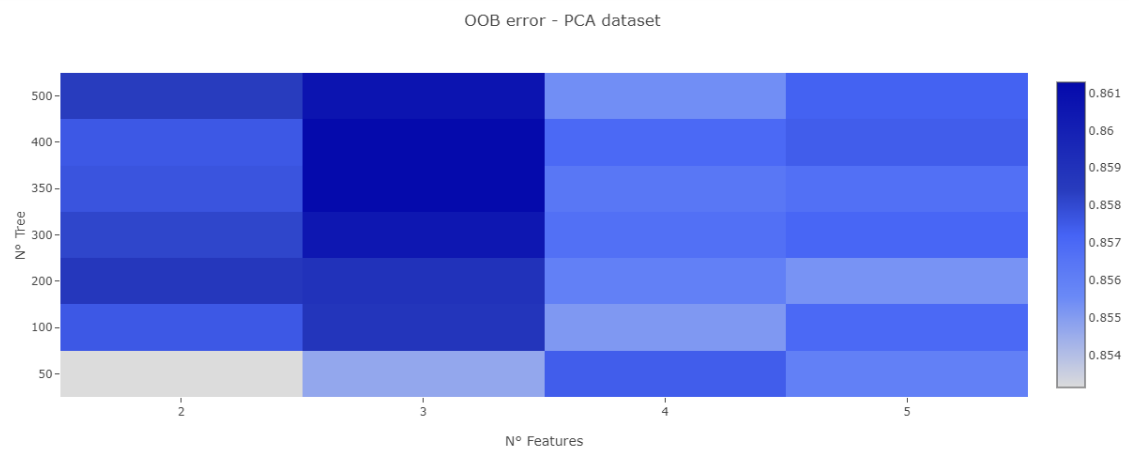


Figure 6.8. Heatmap - PCA dataset

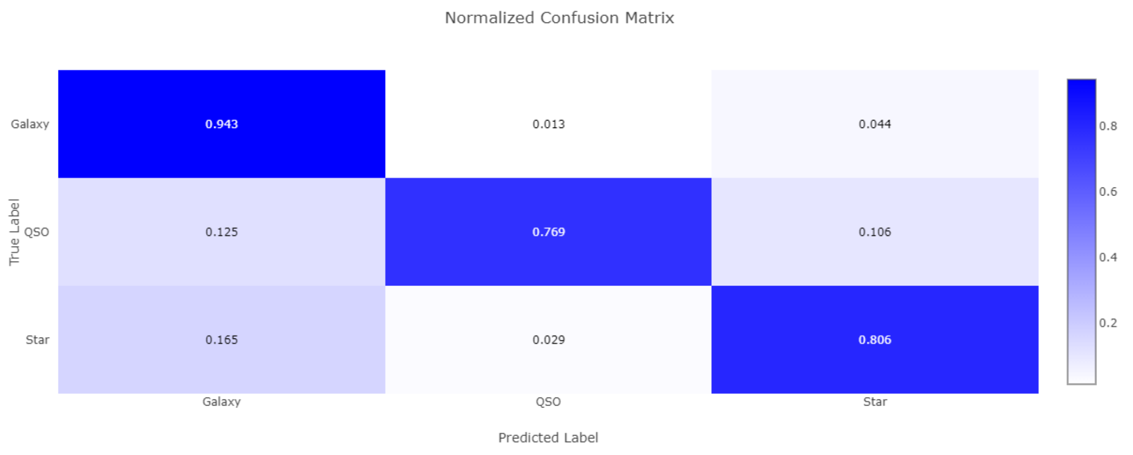


Figure 6.9. Confusion matrix - PCA dataset

Part III

Further analysis

Chapter 7

Learning curve

7.1 Theory recall

Learning curve is a graphical representation of how the precision of the model benefits from running that particular algorithm using more samples. Indeed, even if it is true that having a lot of data is always a positive aspect, sometimes the performance won't increase beyond a certain value even if more records are provided, like in the following picture, where we can notice how after using 1100 samples we don't have any improvement.

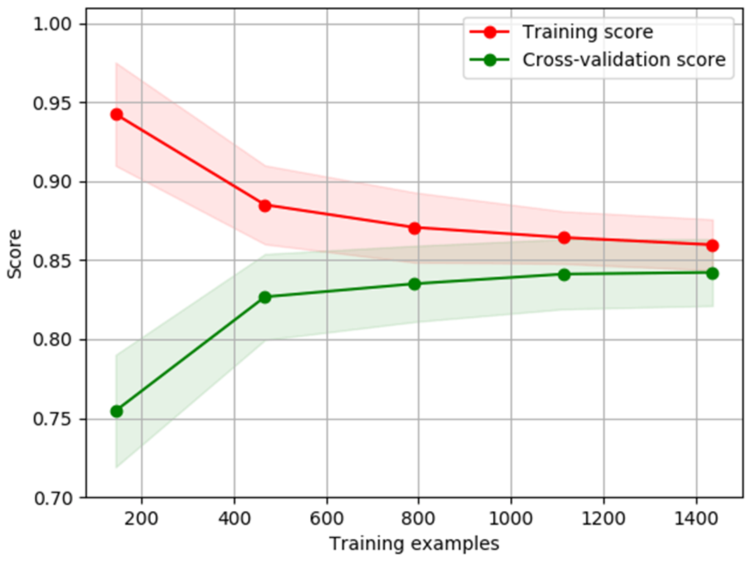


Figure 7.1. Learning curve

The shade nearby the two splines is the variance of the result: the thinner it is the more accurate and unbiased the result it will be.

This kind of analysis can be helpful in order to understand how many samples are enough, so that is possible to save training time and it also gives a hint on the current situation of the training phase: if the spline is not stabilized yet to a value, it may be useful to increment the number of data because it is likely to be able to reach a better performance; if the training spline has a really high score while the validation one scores poorly it probably means that we are overfitting our model, that is we are building a model that is really good at identifying only samples already seen and that will mistake every new record.

7.2 Practical results

*k*NN

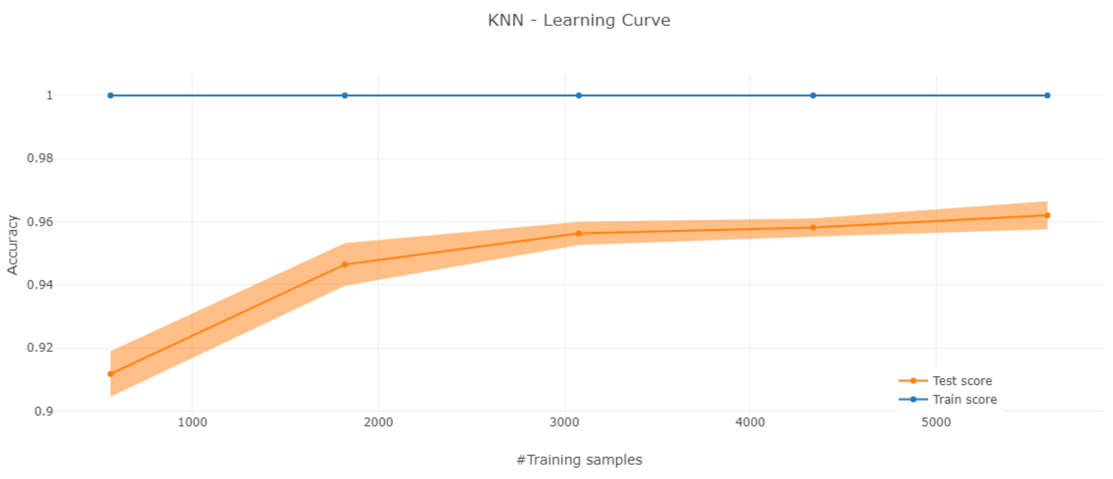


Figure 7.2. Learning curve - *k*NN

Logistic regression

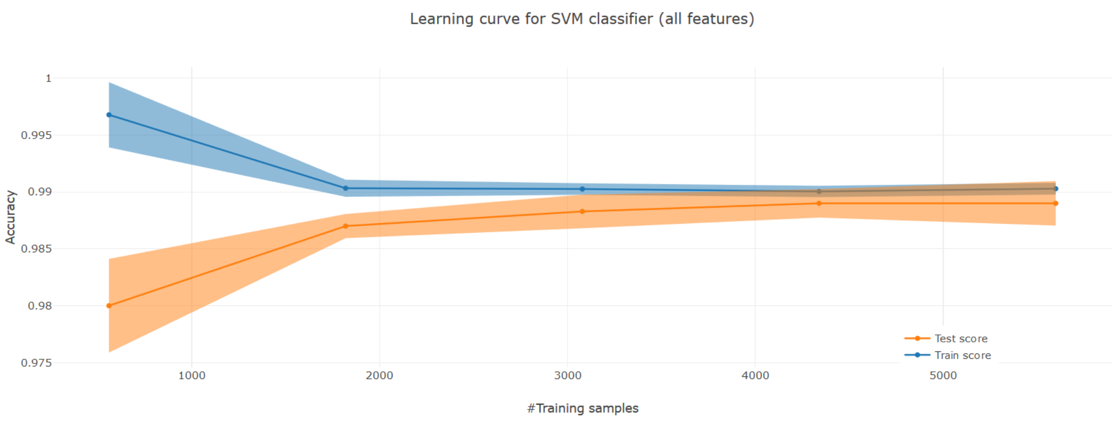


Figure 7.3. Learning curve - logistic regression

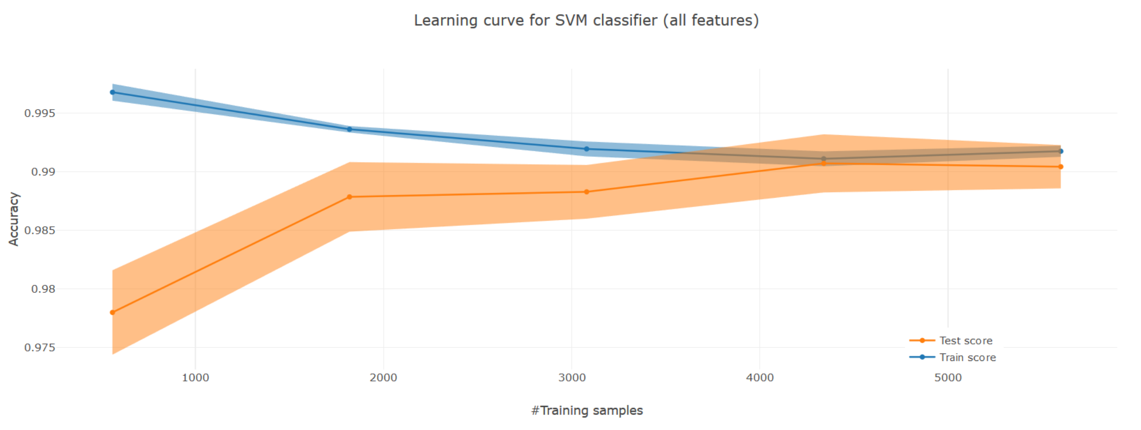


Figure 7.4. Learning curve - SVM

SVM

Random forest

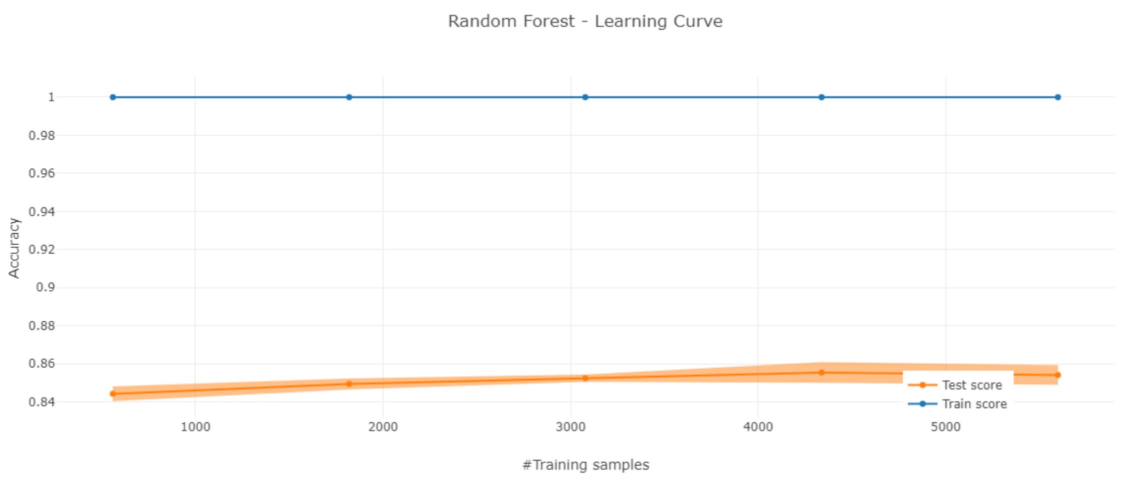


Figure 7.5. Learning curve - random forest

Conclusion

For all the applied techniques, the spline stabilizes around a certain values. Thanks to that we can assume that the dataset comprehend enough statistics in order to achieve a proper result that is also unbiased. It is also possible to reduce the used number of samples and still perform well.

Chapter 8

Balancing techniques

8.1 Theory recall

As we discussed before, the considered one is an **unbalanced dataset** because the frequency of the records belonging to the class “quasar” is lower with respect of the other two. Usually this is a problem while fitting algorithms because, if the implementation is performed without considering this aspect, the score could result very good even if all the instances of the minority class are classified wrongly.

Let’s suppose to have 1000 total records and only 10 samples of these belongs to a class. Building a classifier that always assign the label of the bigger class will have an accuracy of 990/1000 that overall is very impressive. To solve this problem is possible to introduce a different weight inside the loss function, in such a way, data of the minority class has a major impact during the training phase.

It is also suggested to use the correct measure to validate the model. Using as score accuracy is not very representative while, for example, the confusion matrix is very helpful because you are able to understand how your model performs differently for each class.

There are two other ways to solve the class imbalance: oversampling and undersampling. They both deal with the problem varying the number of instances of the classes, the first one introducing new samples, the latter removing them from the majority class.

SMOTE (Synthetic Minority Over-sampling Technique) is a technique that is able to generate new synthetic points so that they resemble to actual real points.

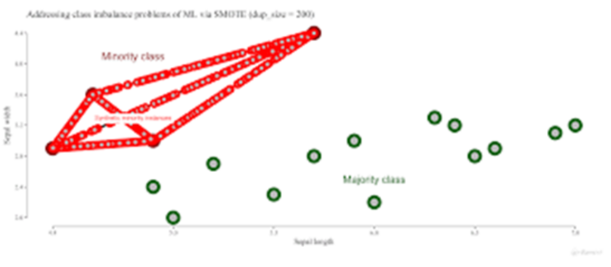


Figure 8.1. SMOTE

The main idea is that these points will be located on top of the distance between closest neighbors of the minority class, like in the previous picture, where new samples can be created where the red dots are.

To actuate undersampling instead is enough to randomly pick from the majority class the same number of samples of the minority class so that it will be balanced.

It is important to underline that, while using both techniques, is required to perform the cross-validation phase before the data set is changed, otherwise this will influence the score and the validation of the algorithm.

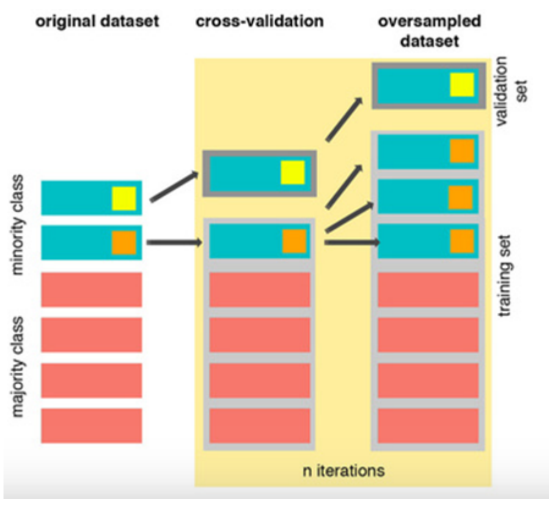


Figure 8.2. Right approach

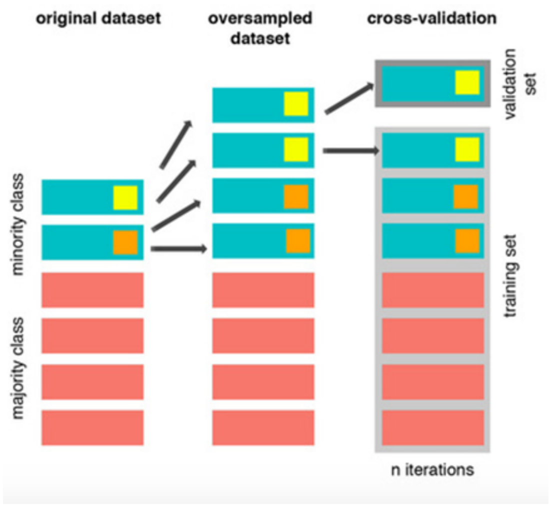


Figure 8.3. Wrong approach

8.2 Practical results

At first, while preprocessing the data set, we thought that we could have used these techniques but in the end we didn't because we saw that all the results were optimal, both in accuracy and analyzing the confusion matrix.

We still wanted to explain this aspect because we have spent some time trying to understand if this was useful in our case.

Part IV

Comparison

Chapter 9

Conclusion

Performances

In order to compare the performances of various applied classifiers, in the following table we summarize the obtained accuracies:

	complete dataset	most relevant dataset	most meaningful dataset	PCA dataset
kNN	91.77 %	96.60 %	96.70 %	85.80 %
Logistic regression	99.00 %	99.00 %	98.80 %	84.66 %
SVM	99.43 %	99.43 %	99.43 %	99.43 %
Random forest	99.03 %	99.10 %	99.30 %	87.13 %

Table 9.1. Overall performances comparison

It is possible to observe from the previous table that all the performances are extremely good, higher than 90% in most of the cases (and often around 99%).

PCA impact

As predicted previously, the lowest scores are achieved by the models obtained on the dataset preprocessed using PCA, proving that for this case it has been useless trying to implement it. In the other three cases, using a different dataset does not provide a huge impact on the final result.

Quasar classification

Beside the accuracy, the best result is that all our models are capable of distinguishing quasar from the other celestial bodies, this is the crucial task of this project because they are in minority and because of this they represent the hardest class to correctly be addressed.

Feature selection impact

We have also verified our hypothesis that the feature selection obtained through random forest had included predictors not very meaningful, such as the calendar date:

differences of precision between these models is very little and the absolutely best is performed by the one that we have named “most meaningful” dataset. Usually this kind of issues are solved by the presence of a *domain expert* that is capable of giving instructions related to which characteristics are useful or less.