Report 3 : Machine Learning methods application in an astrophysics dataset

Piero Paialunga and Simone Chierichini

October 2020

Contents

Abstract

Support Vector Machine and Decision Tree algorithms have been applied in order to predict the Sharp sign of the stars observed from Hubble Telescope. The Principal Component Analysis method permitted the algorithm to be performed with low computational cost. Nonetheless an high accuracy value (82% on 3 classes classification algorithm) has been obtained.

Chapter 1

Introduction

An important set of techniques used in physics is called Machine Learning. By the terms Machine Learning it is intended to mention all the techniques where the computer is instructed to perform a specific task without being specifically programmed to do so. A typical example where Machine Learning is extremely helpful is digit recognition that is the recognition of an hand-written number by the computer. If all the variables of hand writing would be written down the final code would be really long, difficult to interpret and probably not efficient. Machine Learning methods, on the other hand, let the computer learn from the data, and by its own correction, find the better way to classify the digits.

As it has implicitly being said, Machine Learning methods learn by using data. For this reason, as physics is an experimental science and it requires and produces experiments, the Machine Learning techniques can be used and developed. In this report, Machine Learning techniques are applied to an astrophysics dataset. In this particular context of physics, a large amount of data can be analyzed. In these kind of the so called "big data" scenario, in order to reduce the computational stress and increase the performances some dimensionality reduction techniques can be applied. Two different kind of Machine Learning algorithms can be defined:

- **Supervised Learning** is the set of all the techniques where the objective, known as target, is given as an input to the algorithm and it has to learn how to predict the target when it is unknwon
- **Unsupervised Learning** is the set of all the techniques where the target is unknown and the algorithm has to find cluster or groups based by similarities metrics.

The choice of which kind of algorithms to use is drastically determined by the final task. In this specific report Supervised Learning have been used and describe to perform a classification task. This means that the target is a discrete variable representing a specific class and the algorithm has to learn its best classification model from the data. More specifically, given certain information about a star (see Data Description) the algorithms will learn to classify the Sharp, that is a variable that describes how much broader the star's profile appears compared to the PSF profile.

Chapter 2

Data Description

The dataset consists in a table of 51480 rows x 9 columns in which the number of rows represents the length of observation sample, and columns represents the features related to each observation:

- **ID**: Target Identifier
- **X**: X Detector Position
- **Y**: Y Detector Position
- **F606W**: Input magnitude for the F606W band (V band)
- **F814W**: Input magnitude for the F814W band (I band)
- Two columns of **error**: The first for the F606W band input magnitude, the second for the F814W band input magnitude, respectively representing the F606W and F814W uncertainty.
- **Chi**: Goodness-of-fit statistic
- **Sharp**: Describes how much broader the object's profile appears compared to the PSF profile

The first information about the text file is that the error value (the uncertainties of the input values) should be considered in a relative sense, together with a certain number of artificial star tests that have been reported separately in specific datasets. A second important information about the same file is that the magnitude is measured in the Vega system [5]. This is an apparent magnitude that uses $m_{ref} = m_{Vega} = 0$ as $I_{ref} = I_{Vega} = K$. The magnitude is thus computed using the following expression:

$$m_1 - m_{ref} = -2.5 \times \log(\frac{I_1}{I_{ref}})$$

Chapter 3

Methods Description

3.1 Machine Learning framework

As it was already discussed in Report 1, even if the sharp values describe how much broader the object's profile appears compared to the PSF profile, some sharp values are negative.

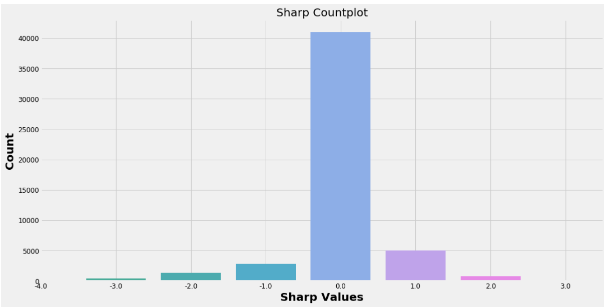

Figure 3.1: As it is possible to see from the count-plot, the Sharp distribution is not entirely positive.

This anomalous property of the dataset was subject of study during the Report n.1. In that scenario, it has been considered the hypothesis that the stars with negative Sharp were the ones with the profile shape that was narrower than the PSF profile. Nonetheless, the analysis of Report 1 has been done by considering the FITS image file that permitted to obtain graphical information about the peak. In absence of this information, **the Supervised Machine Learning approach aims to inspect the dataset and understand if it possible to predict the sharp sign by only looking at the text data**. By using this approach, **a quantitative method to distinguish the stars with positive or negative sharp could be obtained with a certain accuracy**. As the final goal is to distinguish the Positive, Negative or null Sharp of the stars, **the Machine Learning methods have been applied to perform a 2 (Non Negative and Negative) and 3 (Positive, Zero and Negative) classification tasks**. The two Machine Learning methods that have been applied are:

- Support Vector Machines
- Decision Tree / Random Forest

Another important application of Machine Learning to this dataset relies on the **dimensionality reduction**. In fact, the dataset that has been studied is considerably large both in rows (50000+) and in columns (8). **The Principal Component Analysis (PCA) method has**

been applied in order to reduce the number of columns, exploring other dimensions that are able to capture an important part of the variance. This method has been extremely helpful during the training part of Support Vector Machines, as the algorithm is computationally expensive.

3.1.1 Classification Evaluation Metrics

After the classification process, the next step consists in evaluating how the classification has performed on the dataset.
This evaluation is done by the usage of the Confusion Matrix [4]. This matrix is represented by the same number of columns and of rows: one for each class of the dataset. The sum of the element in a row indicates the number of element for each classes. The sum of the element in a column indicates the number of element for each classes following the algorithm classification method. The diagonal elements are the elements that belongs to one classes and are correctly classified as elements of that specific class by the algorithm. The non-diagonal elements are mis-classifications of the algorithm. An immediate definition of the accuracy of the algorithm is the ratio between the sum of the diagonal elements and the sum of all the elements of the confusion matrix. In fact, if this ratio is equal to 1, this means that all the points are classified correctly. This measure is known as **accuracy**.
Nonetheless this measurement is not the only one that needs to be considered while evaluating a classification algorithm. Two important metrcis are also **precision** and **recall**.
The recall metrics indicates how much the algorithm is able to "cover" the class and classify in the correct way the points belonging to that class. For this reason this measurement is also called **coverage** and it is represented by the following expression:

$$R = TP/(TP + FN)$$

Where TP is the so called true positive number i.e. the number of points that are correctly classified as belonging to the selected class (each class has its recall and precision values). FN is the number that are classified as not belonging to the class but they actually do.
The precision metrics indicates how much the algorithm is reliable (or precis, as the name suggests) when it classifies a point as belonging to that class. In fact, a classification is "precise" when few points are classified to belong to that specific class and they don't actually do. As these points are called to be false positives the precision is represented by the following equation:

$$P = TP/(TP + FP)$$

Where FP is the number of false positives happened to be during the classification.
These measurement are deeper than the naive accuracy value, as they highlights different metrics of the algorithm. These metrics are particularly relevant when the dataset target is not balanced and in the dataset an higher percentage of elements belongs to a class rather than another. An efficient example would be to considered a 2 class classification and the dataset presents the same class for the 98% of the elements and a naive classifier that chose this class

for the entire dataset, It's accuracy would be almost 1, but no prediction ability is shown by the classifier.
In this particular framework, both the two classes (Negative/Non Negative) and the three classes dataset (Positive/Negative/Null) are not balanced, as it is possible to verfiy from Figure 3.3 and Figure 3.3. For this reason, together with the accuracy measurement, recall and precision have been computed for each classes. Nonetheless, this imbalance didn't dramatically effect the prevision as the dataset is considerably large, and even the 39% of the point is a sufficient high number.

Figure 3.2: **3 classes dataset**. As it is possible to see from the plot, the three classes dataset target is not balanced.

3.2 Principal Component Analysis

The Principal Component Analysis method is used in order to reduce the dimension of a dataset. One possible definition of PCA is to consider it as an orthogonal projection of the data onto a lower dimensional linear space [1]. This variance optimization can guarantee that the axes that are chosen by

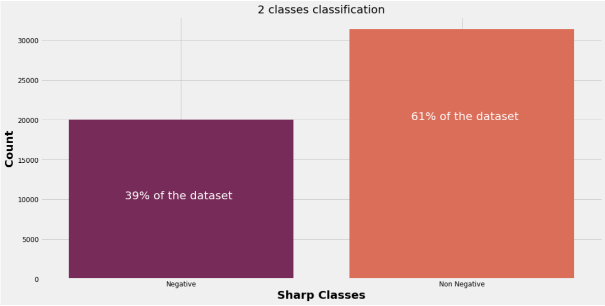

Figure 3.3: **2 classes dataset**. As it is possible to see from the plot, the two classes dataset target is not balanced.

the PCA algorithm contains an high portion of information about the previous features. Nonetheless, the global information about the dataset is lost when a PCA is applied and the dimensionality is reduced. For this reason, it is important to check the value of a quantity called **the explained variance ratio** [3]. This quantity is important to give an idea of the variance that has been discarded during the PCA process. As it is possible to see from Figure 3, almost the entire Variance (Variance Ratio equal to 1) has captured by a three component P.C.A.

If a blind P.C.A. is applied on the original dataset, a not so informative decomposition is applied. In fact, two of the three components that are captured are the spatial coordinates (X and Y) of the original dataset. **This decomposition is suggesting that the X and Y coordinates are highly informative by themselves as they contain high variance and cannot be projected on another dimension that can capture enough variance**.
Even if this result is reasonable, the P.C.A. method becomes really useful in obtaining an high informative but yet reduced dataset

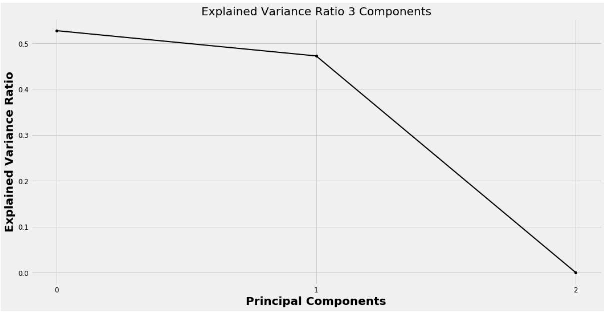

Figure 3.4: **Explained Variance Ratio Plot**. As it is possible to see from the Explained Variance Ratio plot, almost all the variance has been captured by a three component P.C.A.

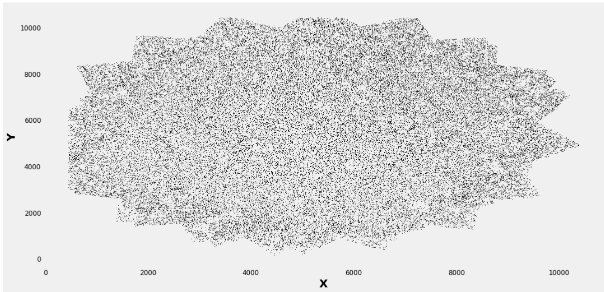

Figure 3.5: **X vs Y Scatter Plot**.

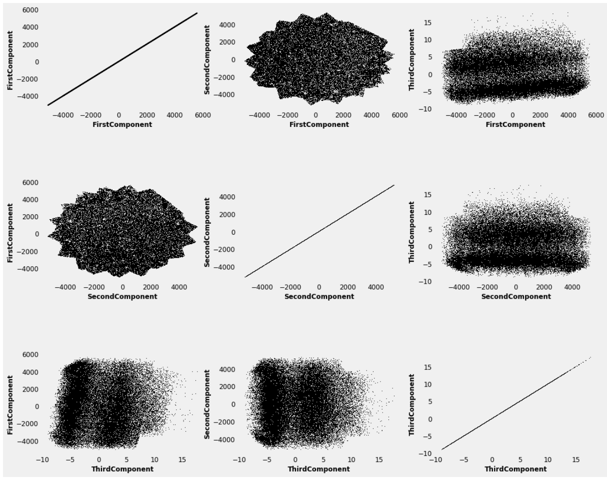

Figure 3.6: **Three Components Principal Component Analysis**. As it is possible to see, FirstComponent vs SecondComponent plot shows the same scatterplot of the X vs Y scatter plot. The first and second P.C.A. Components collapse on the spatial coordinates.

when the decomposition is performed on the non-spatial features dataset. In fact, as it is possible to see from Figure 3.5, this decomposition permitted to obtain non trivial decomposition axes.
In general, even if the Principal Component Analysis is often a powerful tool to decrease the dimensionality of your data, the application of this method is usually accompanied by a loss of explainability. This meas that even if the dataset features are easy to interpret when they are furnished (e.g. F606W is the input magnitude for the F606W band) , the application of the Principal Component Analysis outputs features that are difficult to interpret and understand (e,g, FirstComponent is none of the original features). Nonetheless **in this particular scenario, it has been possible to reconstruct the P.C.A. output components in terms of a linear combination of the original features** [2].

The first important thing to notice is that **the ThirdComponent vs FirstComponent plot resembles the Stellar Color (Flux Difference) vs 814nm Flux plot (Figure 3.6)**. That indicates that one of the component (ThirdComponent) of the P.C.A. is closely related to the flux difference of the original dataset and the other one (FirstComponent) is related to the flux. **The same reasoning could be done for the SecondComponent thus high-**

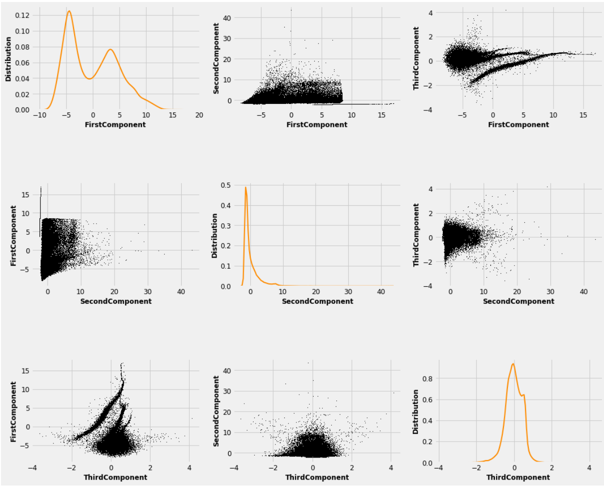

Figure 3.7: **Three Components Principal Component Analysis on the dataset deprived of its spatial coordinates**. As it is possible to see, this P.C.A. permitted to obtain non trivial new features.

lighting how a correlation could be retrieved between the SecondComponent and the Chi feature. To quantitatively define this correlation, the following nomenclature reference has been used:

- χ : the Chi variable
- F_{606} : the 606 nm flux
- F_{814} : the 814 nm flux
- $S_C = F_{814} - F_{606}$: the two fluxes difference
- E_{606}: the 606 nm flux error
- E_{814}: the 814 nm flux error

An acceptable approximation of the ThirdComponent feature (T) is given by the following equation:

$$S_C \approx 1.31 \times T - 1.24$$

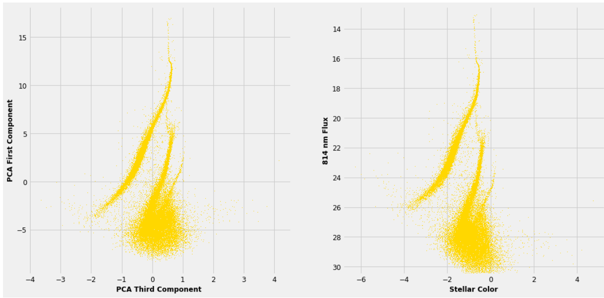

Figure 3.8: **Comparison between the FirstComponent vs ThirdComponent and the 814 nm flux vs Stellar Color plots** . As it is possible to see, the shape of the two plots is notably similar.

In fact the RMSE between the real Stellar Color and its approximation ($RMSE_T$) is the following:

$$RMSE_T = 0.095 = 0.02 \times \max\{S_C\}$$

The goodness of this approximation can be appreciated by the plot of Figure 3.7

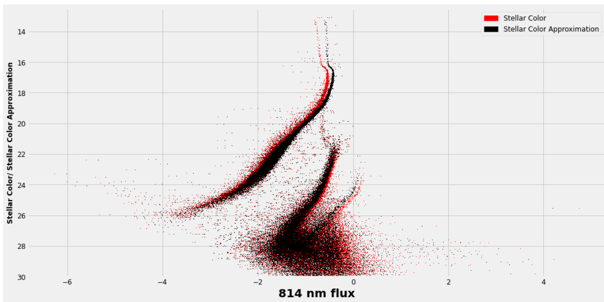

Figure 3.9: **Stellar Color and Stellar Color approximation plot** . As it is possible to see, the Stellar Color is notably close to its approximation.

As it was already explained, the SecondComponent (S) is related with χ . In particular the following approximation can be considered:

$$\chi \approx S \times 0.96 + 2.276$$

In this case, the RMSE between the real χ value and its approximation ($RMSE_S$) is the following:

$$RMSE_S = 0.24 = 0.005 \times \max\{\chi\}$$

Plotting the χ approximation and its real values, the plot of Figure 3.8 has been obtained, and it graphically shows the goodness of the approximation itself.

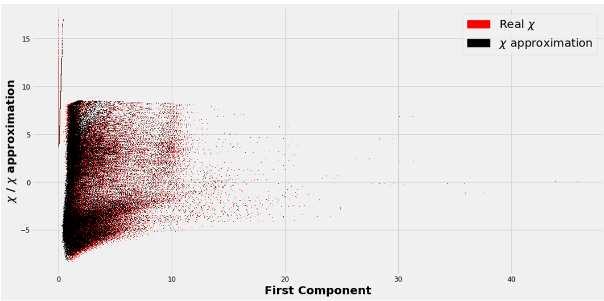

Figure 3.10: **χ and χ approximation plot** . As it is possible to see, the χ value is notably close to its approximation.

The multiplicative and additive factors that have been used so far are obtained by attempts and still permitted to have low RMSE values. A more rigorous method has been applied to obtain the FirstComponent (F) expression, as it has been more difficult to understand. The following expression has been considered:

$$F \approx A \times F_{606} + B + C \times E_{606} + D \times E_{814} + E \times F_{814}$$

And the set of parameters $\{A, B, C, D, E\}$ has been optimized in order to obtain the lowest RMSE between the real First Component and its approximation.

$$1 * A = -1.4, B = +18.6, C = -1, D = -1, E = 10^{-14} \approx 0$$

This parameters set permits to have the following RMSE computed between the First-Component feature and its approximation:

$$RMSE_F = 0.456 = 0.03 \times \max\{F\}$$

The plot in Figure 3.9 is obtained and it highlights the similarity between the First-Component feature and its approximation.

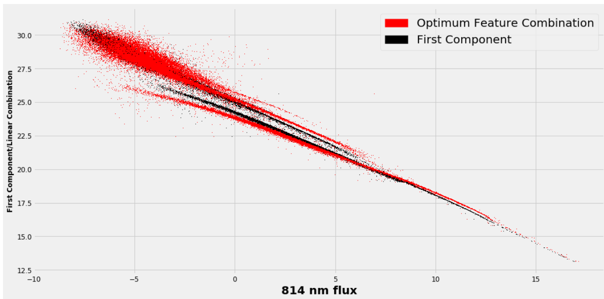

Figure 3.11: **F and F approximation plot** . As it is possible to see, the F feature is notably close to its approximation.

3.3 The Datasets

As the P.C.A. method is applied, two datasets are now obtained:

1. The original dataset, with 7 features + the target
2. The P.C.A. dataset, with 3 components + 2 spatial features + the target

Nonetheless, when using a Machine Learning approach, it is important to furnish to the algorithm informative features more than a large quantities of data. In fact, if the features that are used are not informative, the algorithm may be "confused" by

all this information thus not being able to perform at its best.
Principal Component Analysis performs a dimensionality reduction thus loosing by definition a portion of the original information. Nonetheless, a more specific quantity that needs to be considered while using the Machine Learning approach is the **Mutual Information** [6]. The Mutual Information between two variables ($I(x,y)$) can be considered as the Kullback-Leibler divergence between the distribution of both events in x and y variables ($p(x,y)$) and the product of the distribution of both events considering these events independent by each other ($p(x)p(y)$) [1].

$$I(x,y) = KL = KL(p(x,y)|p(x)p(y))$$

With:

$$KL = -\int\int p(x,y)\ln(\frac{p(x)p(y)}{p(x,y)})dxdy$$

This quantity becomes helpful while considering one of the variable as a feature and the other one as the target. In fact, as the algorithms methods are often computationally expensive, it could be smart to reduce the computational stress by using only the most informative features.
The three most informative features of the second dataset are, with little surprise, the following ones:

	Mutual Information P.C.A. Data
FirstComponent	0.216071
SecondComponent	0.198161
ThirdComponent	0.070408

Figure 3.12: **Mutual Information of the Principal Component Analysis dataset (dataset n.2)**. The three most informative features are the three components coming out from the Principal Component Analysis method.

On the other hand, the three most informative features of the first dataset are the following ones:

	Mutual Information Original Data
F814W	0.201580
Chi	0.203029
F606W	0.211351

Figure 3.13: **Mutual Information of the original dataset (dataset n.1)**. The three most informative features are the two fluxes and the Chi variable.

In order to help the algorithms to obtain the best performance possible, in addition to the first two datasets, the reduced datasets have been used too, obtaining a total of 4 datasets:

1. The original complete dataset, with 7 features + the target

2. The P.C.A. complete dataset, with 3 components + 2 spatial features + the target

3. The original reduced dataset, with 3 features (two fluxes and Chi) + the target

4. The P.C.A. reduced dataset, with 3 features (the three components) + the target

In particular, these 4 datasets has been to test the most computationally expensive algorithm (Linear Support Vector Machine algorithm) and determine which dataset is best suited to perform the classification task.

3.4 Support Vector Machines

3.4.1 Theoretical Background

The first method that has been applied to perform the classification task is the Support Vector Machines method [1]. The Support Vector Machines performs the classification (target t) of a point $\mathbf{x}$. Each point $\mathbf{x}$ can be in fact considered as a vector, with a certain target $t_x \in \{-1, 1\}$ that is associated to it in the following sense :

- $t_x = -1$ means that the first classes is associated with the $\mathbf{x}$ vector.
- $t_x = 1$ means that the first classes is associated with the $\mathbf{x}$ vector.

The Support Vector Machines algorithm uses the following expression to classify the points:

$$y(\mathbf{x}) = \mathbf{w^T x} + b$$

In this equation, $\mathbf{x}$ is the vector that needs to be classified, $\mathbf{w}$ and b are two parameters that needs optimized and $y(\mathbf{x})$ is the classification value that needs to be considered on its sign:

- $y(\mathbf{x}) > 0$ the algorithm classify $\mathbf{x}$ as a point belonging to the second class ($t = 1$)
- $y(\mathbf{x}) < 0$ the algorithm classify $\mathbf{x}$ as a point belonging to the second class ($t = -1$)

According to the way the problem has been described, a correct classification appears when $t_x y(\mathbf{x}) > 0$, as the sign of the algorithm classification is equal to the sign of the real classification. In this sense, $\mathbf{w}$ and b represent the parameters of the separation hyper-plane, and they need to be optimized in order to classify the points in the right class.
The optimization of this parameter follows the idea of maximum margin. In fact, the principle that is behind the Support Vector Machine is that it is more safe to assume that a point belong to the class determined by the separation hyperplane when the distance between the point and the hyperplane itself is large. The assumption can be understood by considering its contrary: if all the points are too close to the considered hyperplane, it is dangerous to assume that hyperplane as the classification plane as small differences between the points coordinates can comport a classification change. An important method to increase the SVM performance is called kernel trick.
The math behind this optimization problem suggests that the optimum values of this parameters can be obtained by maximazing a function called lagrangian (L) that considers the points only by their mutual internal product. In more technical terms, this means that L considers all the points ($\mathbf{x}$) of the dataset the following expression, called kernel:

$$k(\mathbf{x_n}, \mathbf{x_m}) = \mathbf{x_n} \cdot \mathbf{x_m}$$

Moreover, each classification task in a specific space $\mathbf{x} \in X$ can be considered in a transformed space, with the ϕ transform function $\phi(\mathbf{x}) \in \Phi(X)$. It is then possible to consider this kernel function as the internal product of a different space of the original features. This method is known as "kernel trick".
A powerful kernel that has been used in this report is called "Gaussian" kernel, and it is given by the following equation:

$$k(\mathbf{x_n}, \mathbf{x_m}) = exp(-\frac{||\mathbf{x_n} - \mathbf{x_m}||^2}{2\sigma^2})$$

The simple underlying idea of this trick is that a dataset may be not "linearly separable" in a specific domain, but it could be in another one. Kernels permit to implicitly visit other domains and apply the classification algorithm. When the kernel is simply the internal product with no internal transformation the Support Vector Machine algorithm is called Linear. Both the Gaussian Kernel and the Linear one have been used during this report. Another important parameters that can increase the performance of the algorithm is knwon as C.
The strong assumption that has been implicitly taken so far is that it is possible to find in the original or the transformed space an hyperplane that is able to exactly classify all the points in the dataset. Speaking in technical terms, the SVM that does not allow wrong classifications is called an "hard-margin classification". In practice, however, the exact separation may not be possible because two different classified points may be really close to each other. Moreover, even if it is possible to find the exact separation, it could be a separation with a really low margin value, and it could perform poorly on the test set. For these reasons, the C parameters is considered while performing the Support Vector Machine algorithm. In fact, the C value is adopted to control the penalty that is given to a mistake (wrong classification) by the algorithm. If $C \to \infty$ the penalty that is given to a wrong classification is extremely high, thus collapsing on the previous method, the so called hard margin. In general, when $C > 0$ a non infinite penalty is given to the wrong classification, thus making the algorithm more "relaxed" on judging the mistakes. As this parameter can dramatically effect the performance of the algorithm, it has been considered in the hyperparameter tuning part of the work.

3.4.2 Two Classes classification

The four datasets described in section 3.3 has been used in order to test which dataset was more suitable to the classification task. In particular **the Linear Support Vector Machine algorithm has been applied on a two classes classification task : Negative/Non Negative sharp**. In particular, each one of these 4 datasets has been divided in the following parts:

- Train+Validation Set: Random 80% of the dataset
- Test Set: The remaining 20% of the dataset

Moreover, the Train+Validation set has been subdivided into 2 parts:

- Train set: Random 80% of the Train+Validation set

- Validation set: The remaining 20% of the Train+Validation set.

The Train-Validation split has been used to tune the C values within a certain pre-fixed range of values:

1. $C \in \{0, 1\}$
2. $C \in \{1, 10\}$
3. $C \in \{10, 100\}$

For each of the three set, the C values that permitted to have the best accuracy (in that set) has been collected.
These three best values have been tested on the Test Set, and a final best accuracy has been obtained for all the 4 datasets.
The entire process has been repeated 5 times to prove the statistical consistency of the method, obtaining almost identical results.
The accuracy for the four datasets reveal that the best performance has been obtained by using the P.C.A. reduced dataset (dataset n.4). Nontheless, it is important to notice that all the accuracy are close to each other as the minimum accuracy is 61.2% and the maximum accuracy is 63.3%.
A Non Linear Support Vector Machine (Gaussian Kernel) has been applied too. Nonetheless, as this algorithm is computationally expensive, only two features (the two most informative: FirstComponent and SecondComponent) of the best dataset of the previous approach (P.C.A.) has been used. While the previous method has been used to select the best dataset, this one was meant to obtain an higher performance result. For this reason, a rigid train-test split has been used, thus permitting at the algorithm to be trained on a large portion of the dataset. Hyperparameter tuning on the C parameter has been made at this stage too. As a first approach, a range-like division similar to the Linear SVM method has been used, but the best accuracy was always obtained by the larger values of the range. For this reason, the best C value has been considered to be $C = 10^5$. In other words, the hard margin method was the best one in this scenario. The plot in Figure 3.10 the difference between the real targets and the predicted ones has been shown. The SVM algorithm is able to apply a good distintion in the extreme zones of the dataset, but it is not able to determine well the interior (expect for the ellipsoidal curve) as the classification distributions overlap.

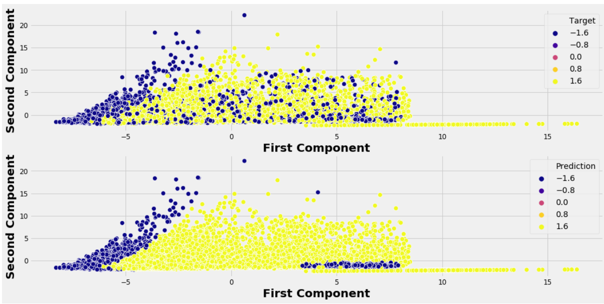

Figure 3.14: **Real Classification vs Predicted Classification**. As it is possible to see, the SVM performs well in the extreme part of the dataset, but are not able to discern the classification on the interior, as the classification distributions overlap.

The same phenomenon has been shown in Figure 3.13 where the decision boundaries of the SVM methods are shown. As it is possible to notice, the algorithm is "confused" by the overlap and consider even really small circles as decision boundary points.

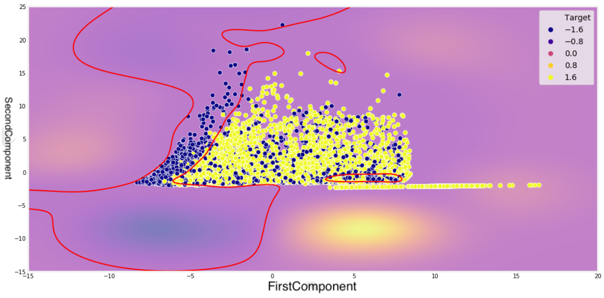

Figure 3.15: **SVM Decision Boundary**. The decision boundary of the algorithm is almost exact at the extremal zones of the dataset, but is not correct in the interior.

The accuracy of this method is 72.2% . **A Non Linear Support Vector Machine two classes classification has been applied on the reduced P.C.A. dataset too, thus considering three features.** Nonetheless, as it was already been said multiple times, the computational power impose some restrictions. In fact, as opposed to the previous methods, a small portion of the dataset has been used in the training+validation set (10% of the data) and a large part of the dataset (90% of the data) has been predicted. The division between Train and Validation Set has been made by considering them to be of equal size. **In order to avoid overfitting, the train,validation and test sets have been iteratively changed 5 times while the parameter hypertuning was made. Each time a the best parameter (lowest RMSE) was extracted and the parameter value that had been taken the most of the times has been considered as optimal. This method has been applied both for the kernel type and the C parameter.** The optimum kernel was the gaussian one, while the best C parameter obtained was $C_{opt} = 39$. The optimal classification gave a total accuracy of 71.7%, obtained extracting the values from the following confusion matrix.

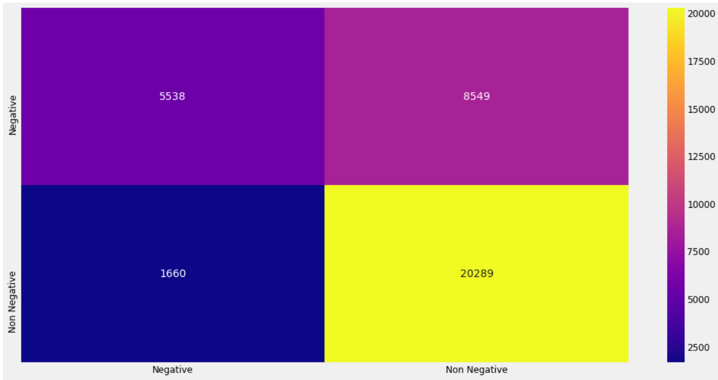

Figure 3.16: **Three Features, Two Classes classification confusion matrix**. As it is possible to appreciate, the performance of the algorithm is better on the Negative class in terms of precision, while it is worst in terms of recall.

The recall of the first class is 39% and it considerably worse than the one on the second class, that is 77% but the latter performs worse than the first in terms of precision as the first class precision is 70% while the second class precision if 92%. The results are summarized in Figure 3.17.

	Negative	Non Negative
Precision	0.769380	0.703551
Recall	0.393128	0.924370

Figure 3.17: **Three Features, Two Classes Classification Evalutaion Metrics**.

3.4.3 Three Classes classification

Even if the results were not particularly encouraging in terms of accuracy, precision and recall, a three classes classification has been performed. In fact, in chapter 3.2 of the Principal Component Analysis emerged that it could be appromaximated the First Component as the -1.4 times the 606 nm flux summed with a constant bias and the negative sum of errors (even if the error is generally significantly lower than its relative flux). The Second Component was again really close to the chi feature. With a non rigorous approximation, it is then possible to consider the First Component as the 606 nm flux. This reasoning can be used to connect what it has been observed during the Report 1, where the 606 nm vs Sharp plot revealed that the Sharp equals to 0 usually place themselves in a narrow extremal zone at low flux values. This could be shown directly from Figure 3.18, where low flux variables correspond to absolute low zero Sharp values.

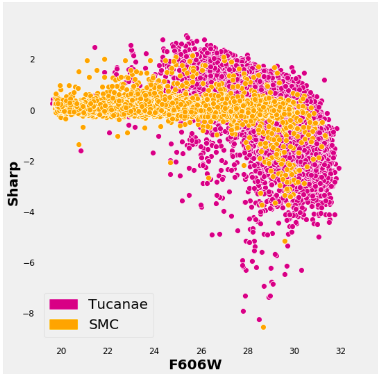

Figure 3.18: **Sharp vs 606 nm flux**. Zero sharp values correspond to low. In Report 1, a distinction (not relevant in this work) about the SMC and Tucanae has been done.

Another important reasoning that can lead to the belief that high flux values correspond to Zero Sharp is the following. During the Data Mining process of the First Report, the Chi variable has been studied in detail. As it was already highlighted in chapter 3.2 and it is possible to appreacite from Figure 3.19 and Figure 3.20, the Chi vs Flux plot is closely related to the SecondComponent vs -FirstComponent plot. Moreover, the Chi variable has been highlighted to be strongly important in terms of determining the values of Sharp when Chi is null. In fact all the points with Chi=0 are the ones with null Sharp.

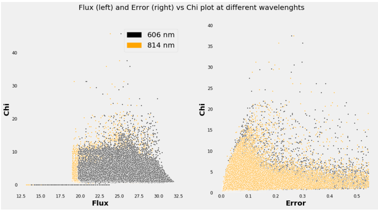

Figure 3.19: **Chi vs Flux and Chi vs Error plots** As the Chi can be approximated with the Second Component and the 606 nm flux with the First Component changed by its sign, the Chi vs Flux plot can be considered as a good approximation of the first two component of the PCA.

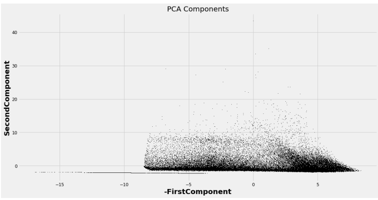

Figure 3.20: **SecondComponent vs -FirstComponent plot** The FirstComponent feature, changed by its sign and rigidly translated, can be approximated with the 606 nm flux, while the Second Component can be approximated with the Chi variable.

This reasoning leads to the final idea that almost all the points with null Sharp are placed in that narrow area with the highest First Component values. This idea find its confirm on the plot from Figure 3.21. This phenomenon makes the Machine Learning method easier to apply, as it requires a simple hyperplane to separe the Null/Not Null sharp values. In this sense, the final accuracy shouldn't dramatically change when a new class is considered as this new class could be classified almost exactly with low effort.

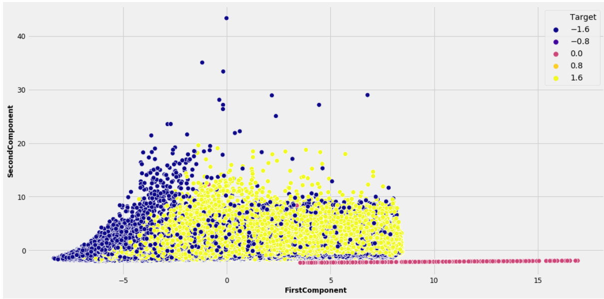

Figure 3.21: **SecondComponent vs -FirstComponent plot** The SecondComponent vs FirstComponent plot shows that almost all the points with null Sharp are located in a narrow zone with the highest FirstComponent values.

The three classes classification has been performed by using a Non Linear Support Vector Machine algorithm on the dataset n.4. The exact same process as the Non Linear Support Vector Machine two classes classification on the reduced P.C.A. dataset (the last method of the previous subsection) has been applied. The resulting optimum kernel is again the gaussian one, while the best C parameter is 17. As it was expected the resulting accuracy is 71.6% and is extremely similar to the one of the Non Linear Support Vector Machine algorithm on the dataset n.4. (71.7%). In particular, the confusion matrix highlight how the zero class is almost exactly classified, while the other two classes classification perform almost at the same level of the previous classifier. This phenomenon is highlighted by the performances in terms of precision and recall that are displayed in Figure 3.23.
As no significant difference between the two accuracy has been reported, the Support Vector Machine method can be considered

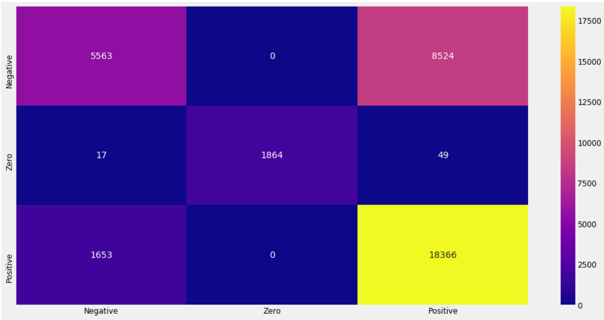

Figure 3.22: **Three classes classification Confusion Matrix** The zero class is almost exactly classified, while the other two classes classification perform almost at the same level of the previous classifier.

	Negative	Positive	Zero
Precision	0.769114	0.917428	0.965803
Recall	0.394903	0.681763	1.000000

Figure 3.23: **Three classes classification evaluation metrics** The zero class is almost exactly classified and it has almost perfect precision and recall. The precision and recall values of the other two classes are almost identical to the one of the previous classifier.

to be working on three classes instead of two. Moreover, **as the best accuracy has been obtained on a two features Non Linear Support Vector Machine, this method should be considered as the optimum Support Vector Machine method for this specific task.**

3.5 Decision Trees

Decision tree is a model widely used in machine learning because it offers high quality in terms of performance, but it's also easily interpretable, it is generally a prediction modeling technique and a decision-supporting tool. It uses a tree-like representation or design and decision model to get accurate inferences. Decision tree, as the name suggests, it has a tree flowchart-like structure that works on the principle of conditions. It is efficient and has strong algorithms used for predictive analysis. the main attributes are as follows:

- **Root Node**: This attribute is used for dividing the data into two or more sets. The feature attribute in this node is selected based on Attribute Selection Techniques.
- **Branch or Sub-Tree**: A part of the entire decision tree is called a branch or sub-tree.
- **Splitting**: Dividing a node into two or more sub-nodes based on if-else conditions.
- **Decision Node**: After splitting the sub-nodes into further sub-nodes, then it is called the decision node.
- **Leaf or Terminal Node**: This is the end of the decision tree where it cannot be split into further sub-nodes.
- **Pruning**: Removing a sub-node from the tree is called pruning

The very basic goal of decision trees is to develop a model that predicts the value of a target by taking some attributes into account and making decisions accordingly. Every internal node holds a “test” on an attribute, branches hold the conclusion of the test and every leaf node means the class label. This is the most used algorithm when it comes to supervised models due to stability and reliability.

The decisions generally depend on if and else conditional statements. Decision trees use various algorithms to recognize the most significant variables, the split, and the best possible value as a result that produces a further subpopulation set. Decisions are made by exploiting some concepts related to information theory; More specifically the information content in an observation describes how surprising it is, given the distribution it comes from. More formally, let x $\in \mathcal{X}$ be an observation coming from a random variable with probability mass function p: $\mathcal{X} \to [0,1]$. Then the information content of x is $I(x) = -\log p(x)$. In addition, an important element used in information theory is called Entropy which is a measure of the uncertainty associated with a random variable.

Given a set of examples D is possible to compute the original entropy of the dataset such as:

$$H(D) = -\sum_{i=1}^{|C|} p(c_i) log(p(c_i))$$

where C is the set of desired class. it is possible to define also the Entropy of an attribute A_i(i.e. Attribute 'Chi' in our dataset). Let the attribute A_i be, with ν values, the root of the current tree, this will partition D into ν subsets $D_1, D_2, ..., D_\nu$. The expected entropy if A_i is used as the current rootis defined as:

$$H_{A_i}(D) = \sum_{i=1}^{\nu} \frac{D_i}{D} H(D_i)$$

The splits of the dataset are made on the basis of a function that takes into account the information carried by each attribute; the best known and widely used are:

- **Infromation gain**: Choose the attribute with the *highest gain* to branch/split the current tree.

  $$IG(D, A_i) = H(D) - H_{A_i}(D)$$

- **Gini Impirity**: It's a measure of the probability of misclassifying an observation. It's calculated as:

  $$G = \sum_{i=1}^{|C|} p(c_i)(1 - p(c_i))$$

  where C is the number of classes and $p(c_i)$ is the probability of randomly picking an element of class i.

The heuristic is to choose the attribute with the maximum **Information Gain** or **Gain Ratio** based on information theory, the objective is to reduce the impurity or uncertainty in data as much as possible. A subset of data is *pure* if all instances belong to the same class. In summary, the main steps of the tree algorithms are:

1. Tree is constructed in a top-down recursive manner
2. At start, all the training examples are at the root
3. Examples are partitioned recursively based on selected attributes

4. Attributes are selected on the basis of an impurity function (e.g., information gain, Gini index)

this class of algorithms is generally equipped with stop conditions, i.e. all examples for a given node belong to the same class, there are no remaining attributes for further partitioning or the number of branches/leafs is fixed.

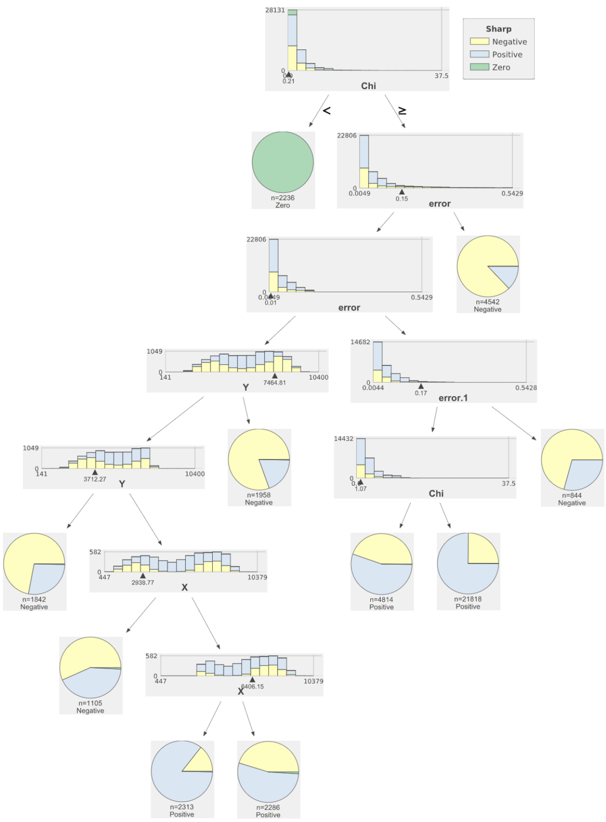

Figure 3.24: Decision Tree visualization. Leaf nodes are pie chart while the decision nodes are displayed as histograms.

This process is illustrated in figure [3.24], which shows a Decision Tree built on our dataset by setting the maximum number of leaves equal to 10 as the stop criterion. The trees were built using a sklearn function called DecisionTreeClassifier. The decision tree, although in its most standard configuration, managed to obtain an accuracy in the 3-class classification task slightly higher than that of the SVM, approximately 74%. In figure [3.25] is shown the related confusion matrix. Surely with a little parameter

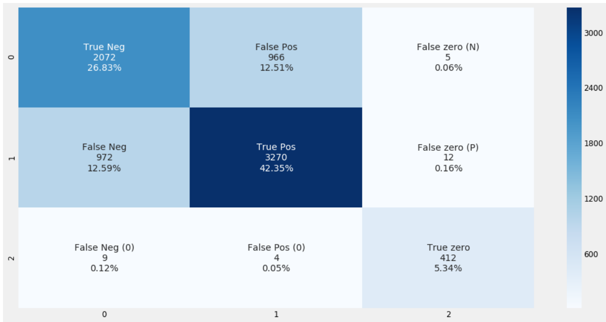

Figure 3.25: Decision Tree Confusion matrix.

tuning it is possible to further improve this result, but it was decided to directly optimize an ensemble-type model that could exploit even more the power of these models.

3.5.1 Ensamble Methods and Random Forest

Ensemble methods is a machine learning technique that combines several base models in order to produce one optimal predictive model. Usually decision trees are used in BAGGing (**B**ootstrap **AGG**regating) methods. Given a sample of data, multiple bootstrapped subsamples are pulled; typically given a standard training set D of size n, bagging generates m new training sets D_i, each of size n, by sampling from

D uniformly and with replacement (replacement could cause some observations to be repeated in each D_i). A Decision Tree is

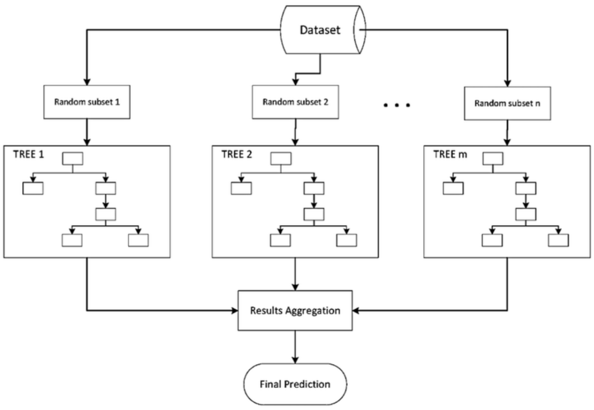

Figure 3.26: Bagging example: A Decision Tree is formed on each bootstrapped sample. The results of each tree are aggregated to yield the strongest, most accurate predictor.

formed on each of the bootstrapped subsamples. After each subsample Decision Tree has been formed, an algorithm is used to aggregate over the Decision Trees to form the most efficient predictor. An example is showed in figure [3.26]. Random Forest are a particular class of Bagging models, because the decision to split is based on a random selection of features; this allows to have a subsampling of all different trees.

3.5.2 Building the model

We therefore tried to fulfill the described classification task using a Random Forest model, again using the sklearn function RandomForestClassifier. For this purpose, the dataset has been divided into training and test sets, and a validation process has been applied to the training set to configure the model properly before testing. In par-

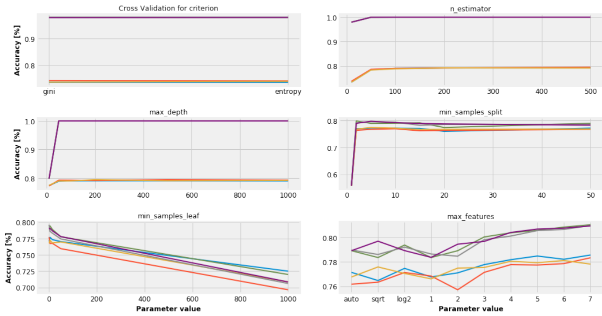

Figure 3.27: Cross validation subplots. Every plot shows 3 cross validation steps for a single feature.

ticular, 3 cross validation steps have been carried out for each tuned feature and the best value is used to test the next one. The parameter tested are the following:

- **criterion**: The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Note: this parameter is tree-specific.
- **Number of estimators**: The number of trees in the forest.
- **max depth**: The maximum depth of the tree.
- **min samples split**: The minimum number of samples required to split an internal node.
- **min samples leaf**: The minimum number of samples required to be at a leaf node.
- **max features**: The number of features to consider when looking for the best split.

Once the results seem encouraging, the accuracy grows up to about 80%. One of the reasons why decision trees are often used is, as we have said, the ease of interpretation; in fact, these models offer the possibility, by going up the tree, to understand exactly what were the decisions that led to the final result. However, when used in this way through ensemble methods, the ability to interpret the model is lacking mainly due to the size and large number of individual trees in the model. Nevertheless, exploiting the aspects listed above related to information theories, the random forest classifier is able to establish a ranking of features based on their importance in decisions [3.28]. The fact that Chi is the most important feature for classification is not surprising, because we know it is related to the value of sharp. In addition, the **error** appears to be more informative in this case than the **X** and **Y** coordinates. To better visualize the classification, the projection of the dataset on the two most relevant directions is shown in [3.29]. The comparison between the predictions of the model and the actual values highlights the limits of this approach. Although the performers are superior to those of the SVM, the Random Forest fails to characterize the points close to zero, where the three classes (above all positive and negative Sharp) mix more.

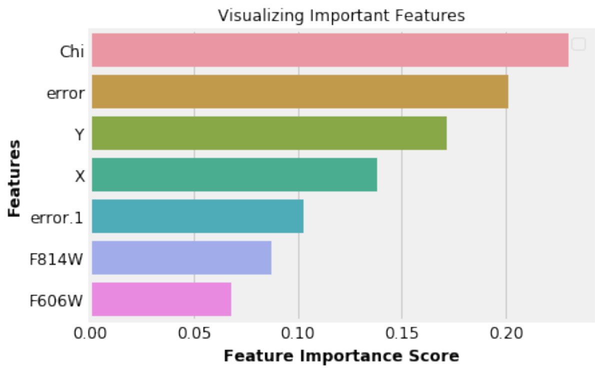

Figure 3.28: Ranking of features according to Random Forest Classifier.

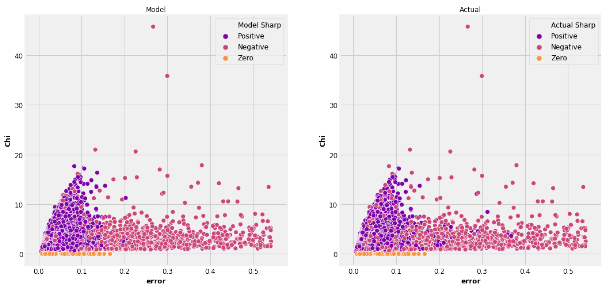

Figure 3.29: Scatterplot of the projection of the test set on the two most relevant directions according to the model. On the left the model prediction, on the right the actual tagets.

3.5.3 Best Model

The last step in the analysis carried out on this dataset consists in a sort of boosting-like method. Usually with the term Boosting, we mean a set of classifiers identified as "*Weak learners*" that used together (typically through a weighted sum) perform better; the result is a better model called "*Strong learner*". In our case we have tried to improve the performance of the optimum Support Vector Machine described in chapter 3.4.3 using the Random Forest classifier, which has been seen to perform better on the dataset under analysis. In particular, a Random Forest was fitted on the data belonging to the area of the principal component space in which the SVM was unable to classify the data properly. Since the dataset to train the model is smaller than in the

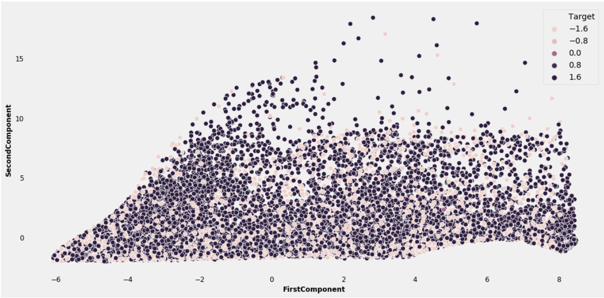

Figure 3.30: Scatter plot of test data wrongly classied by SVM.

previous cases, the model was trained using only 30 % of the isolated data, in order to have a test set of appreciable dimensions. Furthermore, the idea is to try to look at the dataset using a different method using more dimensions, since with the SVM we are limited at the computational level. Basically, the Random Forest Classifier described in the previous section was applied to the portion of the test set that could not be separated using the optimal SMV method, following the idea just described. The model built in this way allowed to reach an accuracy of about 82 %. In fact, it cannot go

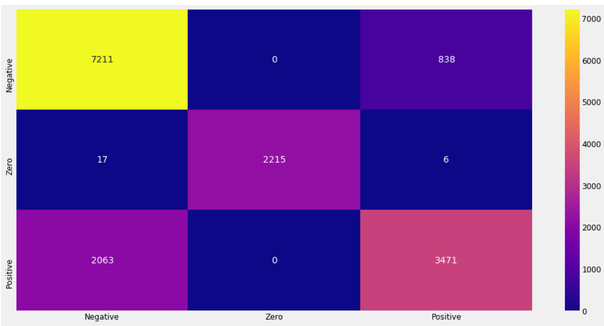

Figure 3.31: Confusion matrix of the Boosted model.

that far beyond the random forest in terms of performance because it is basically a 'targeted' application of the same algorithm, however this process showed excellent potential, since the training, for practical reasons, was limited.

Chapter 4

Conclusion

In this report, several Machine Learning approaches have been used in an anstrophysics framework.
The dataset used consisted in a table of 51480 rows x 9 columns in which the number of rows represents the length of observation sample, and columns represents the features related to each observation:

- **ID**: Target Identifier
- **X**: X Detector Position
- **Y**: Y Detector Position
- **F606W**: Input magnitude for the F606W band (V band)
- **F814W**: Input magnitude for the F814W band (I band)
- Two columns of **error**: The first for the F606W band input magnitude, the second for the F814W band input magnitude, respectively representing the F606W and F814W uncertainty.
- **Chi**: Goodness-of-fit statistic
- **Sharp**: Describes how much broader the object's profile appears compared to the PSF profile

As the Sharp values are not all positive like they should be considering their physical sense, the interest of this work has been the classification of the Sharp sign. In particular two different task has been performed:

1. Negative/Non Negative Sharp classification
2. Negative/Null/Positive Sharp classification

This tasks have been performed by the usage of two different algorithms:

1. The Support Vector Machines
2. The Decision Trees/ Random Forest

As the dataset consisted in general of more than 51000×8 values, the Principal Component Analysis method has been applied in order to perform a dimensionality reduction. A rare case of P.C.A. explainability has been commented and accurately described. With this method, four dataset has been obtained:

1. The original complete dataset, with 7 features + the target

2. The P.C.A. complete dataset, with 3 components + 2 spatial features + the target
3. The original reduced dataset, with 3 features (two fluxes and Chi) + the target
4. The P.C.A. reduced dataset, with 3 features (the three components) + the target

This datasets have all been tested on Support Vector Machines Linear algorithms by using a cross validation method to hypertune the parameters. This approach permitted to state that the best dataset to this specific task was the dataset n.4, that has been subject of ulterior studies. The dataset has been used in the following way:

1. Two Features two classes classification
2. Three Features two classes classification
3. Three Features three classes classification

All these classifications have been made by using a Train/Validation/Test set in order to hypertune the parameters. The methods used an opportune proportion for the three sets in order to meet the computational power available. As the three classes classification performed almost at the same level of the two classes classification, while two features performed better than 3, a two features three classes non linear Support Vector Machine method has been considered as the optimum support vector machine method for this specific task with accuracy $\approx 72\%$.

The Decision Tree method has been then considered. In particular a simple usage of this algorithm permitted to outclass the previous method by obtaining an higher accuracy value ($\approx 74\%$). This was encouraging and pushed to use a bagging ensemble method callled Random Forest. A careful hypeparameter tuning of Random Forest permitted to have an higher accuracy (almost 80%) and permitted to obtain important results about the features importance. The most important one is that the trees extract a lot of their decisional ability from the Chi value. This consideration has physical sense, as the Chi variable is the fit between the stars and the hypothesis shape. The final model that has been used was a boosting ensemble model between the optimum SVM model and the optimum Random Forest model. In particular, the most difficult to predict zone of the Support Vector Machine was the one given by the points that were classified to be with positive Sharp. A portion of these point has been considered as the Training set of the optimum Random Forest model. The remaining part has been predicted with the trained optimum Random Forest model. This last ensemble method permitted to obtain the highest accuracy of the work (almost 82%).

Chapter 5

Appendix

In this section, the codes that permitted to obtain the ouput shown in the report have been reported. The codes are collected in their relative GitHub page. The description of the notebook is the following:

1. "MachineLearningUtility.ipynb"
   In this notebook a single plot has been presented, showing the reason why a Machine Learning approach has been followed

2. "PCA.ipynb"
   The Principal Component Analysis approach has been performed and described. The curios case of P.C.A. explainability has been deepened here and in this article

3. "Linear.ipynb"
   Four dataset has been used to develop a Support Vector Machine linear algorithm:

   - P.C.A. most informative features dataset
   - P.C.A. dataset and other features dataset
   - Original dataset
   - Most informative features original dataset

4. "NonLinearTwoComponents.ipynb"
   Stated that P.C.A. dataset performs better, a non linear Support Vector Machine Algorithm has been applied

5. "NonLinearThreeComponentsTwoC.ipynb"
   The three component P.C.A. dataset has been here used to perform a non linear Support Vector Machine Algorithm for two classes classification.

6. "NonLinearThreeComponentsThreeC.ipynb"
   The three component P.C.A. dataset has been here used to perform a non linear Support Vector Machine Algorithm for three classes classification

7. "DecisionTree.ipynb"
   In this notebook, the decision Tree and Random Forest approach has been used to perform a three classes classification algorithm, hyperparameters tuning has been made in order to get the best random forest algorithm

8. ”Classification.ipynb”
The best hypertuned SVM algorithm coming from the previous notebooks has been used to perform a first classification. Then this classification has been bosted by performing the best Random Forest algorithm from the previous notebook, that has been applied in the most wrong prediction area of the Support Vector Machine algorithm. A total of 82% of accuracy has been obtained.

5.1 MachineLearningUtili

```python
#!/usr/bin/env python
# coding: utf-8

# # Machine Learning Utility

# In[1]:


#Importing the libraries to watch the 'fits' image and get the data array
import astropy
#import plotly.graph_objects as go
from astropy.io import fits
#Importing a library that is useful to read the original file
import pandas as pd
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
#Importing a visual library with some illustrative set up
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib import cm
import numpy as np
from sklearn.decomposition import PCA
import math
import seaborn as sns
from sklearn.linear_model import LogisticRegression
plt.style.use('fivethirtyeight')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
#plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (16, 8)
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
plt.rcParams["axes.grid"] = False


# In[2]:


data=pd.read_csv('star.txt',sep='\s+')


# In[3]:


data.head()


# In[19]:
```

```
sns.countplot(data.Sharp.round())
plt.xlim(5)
plt.grid(True)
plt.title('Sharp Countplot',
    fontsize=20)
plt.xlabel('Sharp Values',fontsize
    =20)
plt.ylabel('Count',fontsize=20)
```

5.2 PCA.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# # P.C.A.

# In[31]:


#Importing a library that is useful
     to read the original file
import pandas as pd
import matplotlib.pyplot as plt
#Importing a visual library with
    some illustrative set up
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib import cm
import numpy as np
from sklearn.utils.testing import
    ignore_warnings
from sklearn.exceptions import
    ConvergenceWarning
from sklearn.decomposition import
    PCA
import math
import seaborn as sns
plt.style.use('fivethirtyeight')
plt.rcParams['font.family'] = 'sans
    -serif'
plt.rcParams['font.serif'] = '
    Ubuntu'
plt.rcParams['font.monospace'] = '
    Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] =
    'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] =
    12
plt.rcParams['ytick.labelsize'] =
    12
plt.rcParams['legend.fontsize'] =
    12
plt.rcParams['figure.titlesize'] =
    12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation']
     = 'none'
plt.rcParams['figure.figsize'] =
    (16, 8)
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] =
    8
plt.rcParams["axes.grid"] = False


# In[3]:


#Importing the dataset
data=pd.read_csv('star.txt',sep='\s
    +')


# In[4]:


data.head()


# In[5]:


#Sharp is the target so it's not
    included
notar=data.drop(columns=['Sharp','#
    ID'])


# In[6]:


#P.C.A. transformation
```

```
pca=PCA(n_components=3)
pca=pca.fit(notar)
pca_data=pd.DataFrame(pca.transform(notar))


# In[7]:


#Explained variance ratio computation
VAR=pca.explained_variance_ratio_


# In[8]:


plt.title('Explained Variance Ratio 3 Components',fontsize=20)
plt.ylabel('Explained Variance Ratio',fontsize=20)
plt.xlabel('Principal Components',fontsize=20)
plt.plot(VAR,marker='.',color='k')
plt.xticks(np.arange(0,3,1))
plt.grid(True)


# In[9]:


#Displaying the new components.
pca=PCA(n_components=3)
pca=pca.fit(data.drop(columns=['Sharp','#ID']))
pca_data=pd.DataFrame(pca.transform(data.drop(columns=['Sharp','#ID'])))
pca_data=pca_data.rename(columns={0:'FirstComponent',1:'SecondComponent',2:'ThirdComponent'})
COL_NAMES=pca_data.columns.tolist()
k=1
for i in range(3):
    col=COL_NAMES[i]
    for j in range(3):
        if k==1:
            plt.subplot(3,3,k)
            plt.subplots_adjust(left=0.025, bottom=0.1, right=0.9, top=1.5, wspace=0.2, hspace=0.7)

            plt.plot(pca_data[col],pca_data[COL_NAMES[j]],color='k')
            plt.xlabel(COL_NAMES[i])
            plt.ylabel(COL_NAMES[j])
        else:
            plt.subplot(3,3,k)
            plt.subplots_adjust(left=0.025, bottom=0.1, right=0.9, top=1.5, wspace=0.2, hspace=0.7)

            plt.plot(pca_data[col],pca_data[COL_NAMES[j]],',',color='k')
            plt.xlabel(COL_NAMES[i])
            plt.ylabel(COL_NAMES[j])

        k=k+1


# In[10]:


#Two variables are now represented as X and Y coordinates
plt.plot(data.X,data.Y,',',color='black')
plt.xlabel('X',fontsize=20)
plt.ylabel('Y',fontsize=20)


# In[11]:


#The last 5 entries of P.C.A. component data
pca_data.tail()
```

```
# In[12]:


#The correlation of the original variables
notar.corr()


# In[13]:


#The correlation of the P.C.A. variables
pca_data.corr()


# # P.C.A. Excluding Space

# In[14]:


#The same P.C.A. has been applied excluding the space coordinates
notar=data.drop(columns=['Sharp','#ID','X','Y'])
pca=PCA(n_components=3)
pca=pca.fit(notar)
pca_data=pd.DataFrame(pca.transform(notar))
pca_data=pca_data.rename(columns={0:'FirstComponent',1:'SecondComponent',2:'ThirdComponent'})
COL_NAMES=pca_data.columns.tolist()
k=1
q=0
for i in range(3):
    col=COL_NAMES[i]
    for j in range(3):
        if k==1 or k==5 or k==9:
            plt.subplot(3,3,k)
            plt.subplots_adjust(left=0.025, bottom=0.1, right=0.9, top=1.5, wspace=0.2, hspace=0.7)

            sns.kdeplot(pca_data[COL_NAMES[i]],color='darkorange')
            #g._legend.remove()
            plt.grid(True)
            plt.xlabel('Values')
            plt.legend([],[], frameon=False)

            plt.xlabel(COL_NAMES[i])
            plt.ylabel('Distribution')
            #plt.ylabel(COL_NAMES[j])
        else:
            plt.subplot(3,3,k)
            plt.subplots_adjust(left=0.025, bottom=0.1, right=0.9, top=1.5, wspace=0.2, hspace=0.7)

            plt.plot(pca_data[col],pca_data[COL_NAMES[j]],',',color='k')
            plt.xlabel(COL_NAMES[i])
            plt.ylabel(COL_NAMES[j])
            plt.grid(True)

        k=k+1


# In[15]:


#Plotting First Component and Third Component, to the left
plt.subplot(1,2,1)
plt.plot(pca_data['ThirdComponent'],pca_data['FirstComponent'],',',color='gold')
plt.xlabel('PCA Third Component ')
plt.ylabel('PCA First Component')
plt.grid(True)
plt.subplot(1,2,2)
#Plotting stellar color and flux, to the right
plt.ylim(30.5,12.5)
plt.ylabel('814 nm Flux')
plt.xlabel('Stellar Color')
```

```python
plt.plot(np.array(data['F814W']-data['F606W']),data.F814W,',',color='gold')
plt.grid(True)


# In[32]:


import matplotlib.patches as mpatches


# In[33]:


#Plotting First Component and 814 flux in black
plt.plot(-pca_data.FirstComponent,data.F814W,',',color='k')
#Plotting a complex feature linear combination in red
plt.plot((data.F606W-26.5)*1.3+5*data.error,data.F814W,',',color='red')
plt.grid(True)

plt.xlabel('814 nm flux',fontsize=20)
plt.ylabel('Stellar Color/ Modified Third Component', fontsize=12)
red_patch = mpatches.Patch(color='red', label='Feature Combination')
black_patch = mpatches.Patch(color='black', label='- First Component')
plt.legend(handles=[red_patch, black_patch])


# In[34]:


#Optimization process to get the exact right coefficient of the linear combination
from sklearn.metrics import mean_squared_error
A=np.arange(1,3,0.1)
B=np.arange(-30.5,-22.5,0.5)
C=np.arange(-15,15,1)
orig=-pca_data.FirstComponent
max_pca=np.abs(-pca_data.FirstComponent.max())
RMSE=[]
TRIPLET=[]
for a in A:

    for b in B:
        for c in C:
            recons=(data.F606W+b)*a+c*data.error
            RMSE.append(np.sqrt(mean_squared_error(recons,orig)))
            TRIPLET.append([a,b,c])
a_opt=TRIPLET[np.array(RMSE).argmin()][0]
b_opt=TRIPLET[np.array(RMSE).argmin()][1]
c_opt=TRIPLET[np.array(RMSE).argmin()][2]
r_opt=(data.F606W+b_opt)*a_opt+c_opt*data.error
D=np.arange(-10,10,0.1)
BEST_RMSE=[]
for d in D:
    recons=r_opt+d*data.F814W
    BEST_RMSE.append(np.sqrt(mean_squared_error(recons,orig)))
BEST_RMSE=np.array(BEST_RMSE)
d_opt=D[BEST_RMSE.argmin()]
r_opt=(data.F606W+b_opt)*a_opt+c_opt*data.error+d_opt*data.F814W
E=np.arange(-10,10,1)
BEST_RMSE=[]
for e in E:
    recons=r_opt+e*data['error.1']
    BEST_RMSE.append(np.sqrt(mean_squared_error(recons,orig)))
BEST_RMSE=np.array(BEST_RMSE)
e_opt=E[np.array(BEST_RMSE).argmin()]
r_opt_first=r_opt+e_opt*data['error
```

```
.1']


# In[35]:


print ('The lowest RMSE is '+ str(BEST_RMSE.min()))


# In[36]:


#Displaying the best values to obtain the linear combination with less R.M.S.E.
print ('The best parameters are \n')
print((a_opt,b_opt,c_opt,d_opt,e_opt))


# In[37]:


r_opt=-r_opt


# In[40]:


#Plotting the optimum feature combination and the first component of P.C.A.
plt.plot(pca_data.FirstComponent,data.F814W,',',color='k',label='First Component')
plt.plot(r_opt,data.F814W,',',color='red', label='Linear Combination')
plt.grid(True)
plt.xlabel('814 nm flux',fontsize=20)
plt.ylabel('First Component/Linear Combination',fontsize=12)
red_patch = mpatches.Patch(color='red', label='Optimum Feature Combination')
black_patch = mpatches.Patch(color='black', label='First Component')
plt.legend(handles=[red_patch,black_patch],fontsize=20)


# In[48]:


#Plotting a linear combination of Second Combonent + bias (y) vs First component (black)
#Plotting Chi vs First component (red)
plt.plot(np.array(data.Chi),pca_data['FirstComponent'],',',color='red')
plt.plot(pca_data.SecondComponent*0.96+2.276,pca_data['FirstComponent'],',',color='black')
plt.grid(True)
red_patch = mpatches.Patch(color='red', label='First Component')
black_patch = mpatches.Patch(color='black', label='Modified Second Component')
plt.xlabel('First Component',fontsize=20)
plt.ylabel('Chi/ Modified Second Component', fontsize=12)
plt.legend(handles=[red_patch,black_patch])


# In[41]:


#Optimum parameters for the second component
r_opt_sec=pca_data.SecondComponent*0.96+2.276


# In[42]:


#Computing the RMSE between the real second component and the
```

```
    linear combination
second_RMSE=np.sqrt(
    mean_squared_error(pca_data.
    SecondComponent*0.96+2.276,data
    .Chi))/data.Chi.max()


# In[43]:


print('The RMSE for the
    reconstructed second component
    is ' + str(second_RMSE))


# In[42]:


import matplotlib.patches as
    mpatches
#Plotting 814nm flux vs Stellar
    color (red) or a linear
    combination of Third Component
    + bias (black)
plt.ylim(30.0,12.5)
plt.plot(np.array(data['F814W']-
    data['F606W']),data.F814W,',',
    color='red',label='Stellar
    Color')
plt.plot(-1.24+1.31*pca_data.
    ThirdComponent,data.F814W,',',
    color='k',label='Modified Third
     Component')
plt.grid(True)
red_patch = mpatches.Patch(color='
    red', label='Stellar Color')
black_patch = mpatches.Patch(color=
    'black', label='Modified Third
    Component')
plt.xlabel('814 nm flux',fontsize
    =20)
plt.ylabel('Stellar Color/ Modified
     Third Component', fontsize=12)
plt.legend(handles=[red_patch,
    black_patch])


# In[44]:


r_opt_third=-1.24+1.31*pca_data.
    ThirdComponent


# In[45]:


third_RMSE=np.sqrt(
    mean_squared_error(-1.24+1.31*
    pca_data.ThirdComponent,np.
    array(data.F814W-data.F606W)))


# In[46]:


print ('The RMSE between the Third
    Component and its
    reconstruction is '+ str(
    third_RMSE))


# In[57]:


#Stacking together the coordinates
    and the sharp
pca_data['X']=data.X
pca_data['Y']=data.Y


# In[58]:


pca_data['Sharp']=data.Sharp


# In[59]:


pca_data.head()


# In[60]:


#pca_data=pca_data.drop(columns=['
    Sharp'])
```

5.3 Linear.ipynb

```python
#!/usr/bin/env python
# coding: utf-8

# # 4 Dataset Classification

# In[2]:


#Importing the libraries to watch the 'fits' image and get the data array
import astropy
#import plotly.graph_objects as go
from astropy.io import fits
#Importing a library that is useful to read the original file
import pandas as pd
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
#Importing a visual library with some illustrative set up
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib import cm
import numpy as np
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.decomposition import PCA
import math
import seaborn as sns
from sklearn.linear_model import LogisticRegression
plt.style.use('fivethirtyeight')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
#plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (16, 8)
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
plt.rcParams["axes.grid"] = False


# In[3]:


#Importing the dataset
data=pd.read_csv('star.txt',sep='\s+')


# In[4]:


#Displaying the first 5 rows
data.head()


# In[5]:


#Dataset without Sharp (target) and #ID
notar=data.drop(columns=['Sharp','#ID'])


# # P.C.A. Excluding Space

# In[7]:
```

```
#Excluding Space and performing P.C.A.
notar=data.drop(columns=['X','Y'])


# In[8]:


notar.head()


# In[9]:


pca=PCA(n_components=3)
pca=pca.fit(notar)
pca_data=pd.DataFrame(pca.transform(notar))


# In[10]:


pca_data=pca_data.rename(columns={0:'FirstComponent',1:'SecondComponent',2:'ThirdComponent'})


# In[11]:


#Stacking the X and Y coordinates to the P.C.A.
pca_data['X']=data.X
pca_data['Y']=data.Y


# In[12]:


#And the Sharp
pca_data['Sharp']=data.Sharp


# In[13]:


#Displaying the first rows
pca_data.head()


# In[60]:


#pca_data=pca_data.drop(columns=['Sharp'])


# # 2-class: Positive or Negative

# In[16]:


#2 class classification preprocessing
#The sharp is considered with its sign
#The 0 sharp are now considered as positive

data['SharpSign']=data.Sharp.apply(np.sign)
data['SharpSign']=data['SharpSign'].replace(0,1)


# In[24]:


#From Sklearn import library for mutual information
from sklearn.feature_selection import mutual_info_classif as mi


# In[25]:


#Take columns and compute the Mutual Information
COL=data.columns.tolist()[:-2]


# In[34]:
```

```
MI=mi(data.drop(columns=['Sharp','#ID']),np.array(data.SharpSign).reshape(-1,1))


# In[37]:


MI_data=pd.DataFrame({'MutualInformation':MI})


# In[44]:


MI_data.index=data.drop(columns=['Sharp','#ID']).columns.tolist()


# In[52]:


#Excluding the trivial correlation of the Sharp Sign with itself
#three most informative feature are the two errors and the flux feature
MI_data.sort_values(by='MutualInformation',ascending=False).head(4)


# In[55]:


#Computing the same thing with the P.C.A. dataset
COL=pca_data.columns.tolist()[:-1]
MI=mi(pca_data,np.array(data.SharpSign).reshape(-1,1))


# In[61]:


MI_data=pd.DataFrame({'MutualInformation':MI})
MI_data.index=COL+['Sharp']


# In[62]:


#Excluding the trivial correlation of the Sharp Sign with itself
#the three most informative feature are the three components
MI_data.sort_values(by='MutualInformation',ascending=False).head(4)


# # 1. PCA Linear Classifier

# In[10]:


pca_data['Target']=data['SharpSign']


# In[303]:


pca_data.head()


# In[11]:


#Importing linear SVM classifier and model selection train test split
from sklearn.svm import LinearSVC as SVC
from sklearn.model_selection import train_test_split


# In[325]:


#The dataset
X=pca_data.drop(columns=['Sharp','Target'])
```

```


# In[326]:


#The target
y=pca_data.Target


# In[327]:


#Test-(validation+train) split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)


# In[328]:


#Validation-Train split
X_traint,X_train_val,y_traint, y_train_val= train_test_split(
    X_train, y_train, test_size=0.2, random_state=42)


# In[ ]:


#Computing the best coefficient on the validation set, C range=0-1
#Training on the training set
C_SCORE=[]
c=np.arange(0.2,1.2,0.2)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val, y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[346]:


#Printing the best C in terms of accuracy on the validation set between 0 and 1
MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[347]:


#Computing the best coefficient on the validation set, C range=1-10
#Training on the training set
C_SCORE=[]
c=np.arange(1,11,1)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val, y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[349]:


#Printing the best C in terms of accuracy on the validation set between 1 and 10

MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


```

```
# In[343]:


#Computing the best coefficient on the validation set, C range=10 and 100
#Training on the training set

C_SCORE=[]
c=np.arange(10,110,10)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[344]:


#Printing the best C in terms of accuracy on the validation set between 10 and 100

MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[386]:


#Taking the best three possibilities for the three ranges
OPT_C=[0.6,7,30]


# In[388]:


#Testing them on the training set and see which is the best
FIN_SCORE=[]
for opt_C in OPT_C:
    clf=SVC(C=opt_c)
    clf.fit(X_train,y_train)
    fin_score=clf.score(X_test,y_test)
    FIN_SCORE.append(fin_score)


# In[391]:


FIN_SCORE=np.array(FIN_SCORE)


# In[392]:


fin_score=FIN_SCORE.max()


# In[394]:


print ('The PCA dataset gave a best classification with ' + str(fin_score*100)+ '% of accuracy with a linear classifier')


# # 1.2 Dataset Linear Classifier

# In[413]:


#Same process as before, the only difference is that we used the entire dataset
#And not the P.C.A.
X=data.drop(columns=['SharpSign'])


# In[414]:


X=pca_data.drop(columns=['Sharp','Target'])

```

```python

# In[415]:


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)


# In[416]:


X_traint,X_train_val,y_traint,y_train_val= train_test_split(
    X_train, y_train, test_size=0.2, random_state=42)


# In[417]:


C_SCORE=[]
c=np.arange(0.2,1.2,0.2)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[418]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[419]:


C_SCORE=[]
c=np.arange(1,11,1)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[420]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[421]:


C_SCORE=[]
c=np.arange(10,110,10)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[422]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
```

```
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[423]:


OPT_C=[1,70]


# In[424]:


FIN_SCORE=[]
for opt_C in OPT_C:
    clf=SVC(C=opt_c)
    clf.fit(X_train,y_train)
    fin_score=clf.score(X_test,y_test)
    FIN_SCORE.append(fin_score)


# In[425]:


FIN_SCORE=np.array(FIN_SCORE)


# In[426]:


fin_score=FIN_SCORE.max()


# In[427]:


print ('The PCA dataset gave a best classification with ' + str(fin_score*100)+ '% of accuracy with a linear classifier')


# # 1.3 Best Features Classification Original Data

# In[49]:



#Same process as before, but only the most informative feature has been used (error,606flux and error)
opt_data=data.drop(columns=['#ID','X','Y','Chi','Sharp','F814W'])


# In[50]:


opt_data.head()


# In[436]:


X=opt_data.drop(columns=['SharpSign'])


# In[437]:


data['SharpSign']=data.Sharp.apply(np.sign)
data['SharpSign']=data['SharpSign'].replace(0,1)
y=data.SharpSign


# In[438]:


X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size=0.2, random_state=42)


# In[439]:


X_traint,X_train_val,y_traint,y_train_val= train_test_split(
 X_train, y_train, test_size=0.2, random_state=42)

```

```python
# In[440]:


C_SCORE=[]
c=np.arange(0.2,1.2,0.2)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[441]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[442]:


C_SCORE=[]
c=np.arange(1,11,1)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[443]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[444]:


C_SCORE=[]
c=np.arange(10,110,10)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[445]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[447]:


OPT_C=[0.2,1,40]


# In[448]:


FIN_SCORE=[]
for opt_C in OPT_C:
    clf=SVC(C=opt_c)
    clf.fit(X_train,y_train)
```

```
    fin_score=clf.score(X_test, y_test)
    FIN_SCORE.append(fin_score)


# In[449]:


FIN_SCORE=np.array(FIN_SCORE)


# In[450]:


fin_score=FIN_SCORE.max()


# In[451]:


print ('The PCA dataset gave a best classification with ' + str(fin_score*100)+ '% of accuracy with a linear classifier')


# # 1.4 Best Feature Classification PCA data

# In[452]:


#Same process as before, but only the most informative P.C.A. components have been used
opt_data=pca_data[['FirstComponent','SecondComponent','ThirdComponent']]


# In[453]:


X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)


# In[454]:


X_traint,X_train_val,y_traint,y_train_val= train_test_split(
    X_train, y_train, test_size=0.2, random_state=42)


# In[455]:


C_SCORE=[]
c=np.arange(0.2,1.2,0.2)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + ' coefficient has been adopted' )


# In[456]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min())+ ' and ' +str(c.max())+ '\n')
print ('the best score has been obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[457]:


C_SCORE=[]
c=np.arange(1,11,1)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + '
```

```
    coefficient has been adopted' )


# In[458]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min
    ())+ ' and ' +str(c.max())+ '\n
    ')
print ('the best score has been
    obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[459]:


C_SCORE=[]
c=np.arange(10,110,10)
for c_value in c:
    clf=SVC(C=c_value)
    clf.fit(X_traint,y_traint)
    sc=clf.score(X_train_val,
    y_train_val)
    C_SCORE.append(sc)
    print(str(c_value) + '
    coefficient has been adopted' )


# In[460]:


MIN_VAL=np.array(C_SCORE).max()
MIN_C=c[np.array(C_SCORE).argmax()]
print ('In the range ' + str(c.min
    ())+ ' and ' +str(c.max())+ '\n
    ')
print ('the best score has been
    obtained with ' + str(MIN_C))
print ('and it is ' + str(MIN_VAL))


# In[461]:


OPT_C=[0.2,1,70]


# In[462]:


FIN_SCORE=[]
for opt_C in OPT_C:
    clf=SVC(C=opt_c)
    clf.fit(X_train,y_train)
    fin_score=clf.score(X_test,
    y_test)
    FIN_SCORE.append(fin_score)


# In[463]:


FIN_SCORE=np.array(FIN_SCORE)


# In[464]:


fin_score=FIN_SCORE.max()


# In[465]:


print ('The PCA dataset gave a best
    classification with ' + str(
    fin_score*100)+ '% of accuracy
    with a linear classifier')
```

5.4 NonLinearTwoComponent.ipy

```
#!/usr/bin/env python
# coding: utf-8

# # 4 Dataset Classification

# In[4]:


#Importing the libraries to watch
    the 'fits' image and get the
    data array
import astropy
#import plotly.graph_objects as go
```

```
from astropy.io import fits
#Importing a library that is useful to read the original file
import pandas as pd
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
#Importing a visual library with some illustrative set up
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib import cm
import numpy as np
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.decomposition import PCA
import math
import seaborn as sns
from sklearn.linear_model import LogisticRegression
plt.style.use('fivethirtyeight')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
#plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (16, 8)
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
plt.rcParams["axes.grid"] = False


# In[5]:


#Importing the dataset
data=pd.read_csv('star.txt',sep='\s+')


# In[6]:


#First five entries
data.head()


# In[7]:


#Dropping the target and the ID
notar=data.drop(columns=['Sharp','#ID'])


# # P.C.A. Excluding Space

# In[8]:


notar=data.drop(columns=['Sharp','#ID','X','Y'])


# In[9]:


notar.head()


# In[10]:


#Applying P.C.A. to the non spatial
```

```python
    features
pca=PCA(n_components=3)
pca=pca.fit(notar)
pca_data=pd.DataFrame(pca.transform(notar))


# In[11]:


#Renaming them
pca_data=pca_data.rename(columns={0:'FirstComponent',1:'SecondComponent',2:'ThirdComponent'})


# In[12]:


#Stacking together the spatial coordinates
pca_data['X']=data.X
pca_data['Y']=data.Y


# In[13]:


#And the target
pca_data['Sharp']=data.Sharp


# In[14]:


pca_data.head()


# # 2. Best method non linear

# In[15]:


#SVM applied to the first two component
opt_data=pca_data[['FirstComponent','SecondComponent']]


# In[31]:


#Two target classification
#Taking the Sharp and considering only its sign
data['SharpSign']=np.sign(data.Sharp)


# In[33]:


#As it is two class, the 0 values are set together with the positive ones.
data[data['SharpSign']==0].SharpSign=np.ones(len(data[data['SharpSign']==0]))


# In[85]:


#Target set as dharp sign
opt_data['Target']=data.SharpSign


# In[74]:


#Dataset without target
X=opt_data.drop(columns=['Target'])


# In[75]:


#Scatterplot of First and Second Component, together with the target
sns.scatterplot(opt_data.FirstComponent,opt_data.SecondComponent,hue=opt_data.Target,palette='plasma')
plt.grid(True)


```

```python
# In[78]:


#Importing sklearn libraries for SVC and model selection
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split


# In[87]:


#Considering the target y
y=opt_data.Target


# In[88]:


#Considering the train test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)


# In[485]:


#Checking the best C value for SVM between 0 and 1
c_list=np.arange(0.10,1.1,0.10)
k=0
FIN_SCORE=[]
for c in c_list:
    clf=SVC(C=c, kernel='rbf')
    clf.fit(X_train,y_train)
    fin_score=clf.score(X_test,y_test)
    FIN_SCORE.append(fin_score)

    if k==1:
        print('20% of the C values inspected')
    if k==4:
        print('50% of the C values inspected')
    if k==7:
        print ('80% of the C values inspected')
    if k == 9:
        print ('100% of the C values inspected \n')
        print ('Process completed')
    k=k+1


# In[487]:


#Computing the accuracy
FIN_SCORE=np.array(FIN_SCORE)


# In[488]:


#Checking the best C(best accuracy)
fin_score=FIN_SCORE.max()
i=FIN_SCORE.argmax()
c_max=c_list[i]


# In[500]:


print ('The best classification is done between '+ str(c_list.min()) + ' and ' + str(c_list.max()) + '\n')
print('is ' + str(c_max) +', obtaining the following accuracy: '+str(fin_score*100)+'%')


# In[501]:


#Same process, but changing C between 1 and 10
c_list=np.arange(1.,11,1.)
k=0
FIN_SCORE=[]
for c in c_list:
    clf=SVC(C=c, kernel='rbf')
    clf.fit(X_train,y_train)
```

```
    fin_score=clf.score(X_test, y_test)
    FIN_SCORE.append(fin_score)

    if k==1:
        print('20% of the C values inspected')
    if k==4:
        print('50% of the C values inspected')
    if k==7:
        print ('80% of the C values inspected')
    if k == 9:
        print ('100% of the C values inspected \n')
        print ('Process completed')
    k=k+1


# In[504]:


FIN_SCORE=np.array(FIN_SCORE)
fin_score=FIN_SCORE.max()
i=FIN_SCORE.argmax()
c_max=c_list[i]
print ('The best classification is done between '+ str(c_list.min()) + ' and ' + str(c_list.max()) + '\n')
print('is ' + str(c_max) +', obtaining the following accuracy: '+str(fin_score*100)+'%')


# In[506]:


#Same process but changing C between 10 and 50
c_list=np.arange(10,60,10)
k=0
FIN_SCORE=[]
for c in c_list:
    clf=SVC(C=c, kernel='rbf')
    clf.fit(X_train,y_train)
    fin_score=clf.score(X_test, y_test)
    FIN_SCORE.append(fin_score)
    if k==0:
        print('20% of the C values inspected')
    if k==1:
        print('40% of the C values inspected')
    if k==2:
        print('60% of the C values inspected')
    if k==3:
        print('80% of the C values inspected')
    if k==4:
        print('100% of the C values inspected')
    k=k+1


# In[508]:


FIN_SCORE=np.array(FIN_SCORE)
fin_score=FIN_SCORE.max()
i=FIN_SCORE.argmax()
c_max=c_list[i]
print ('The best classification is done between '+ str(c_list.min()) + ' and ' + str(c_list.max()) + '\n')
print('is ' + str(c_max) +', obtaining the following accuracy: '+str(fin_score*100)+'%')


# In[510]:


#Same process between 100 and 500
c_list=np.arange(100,600,100)
k=0
FIN_SCORE=[]
for c in c_list:
    clf=SVC(C=c, kernel='rbf')
    clf.fit(X_train,y_train)
    fin_score=clf.score(X_test, y_test)
```

```
    FIN_SCORE.append(fin_score)
    if k==0:
        print('20% of the C values inspected \n')
    if k==1:
        print('40% of the C values inspected \n')
    if k==2:
        print('60% of the C values inspected \n')
    if k==3:
        print('80% of the C values inspected \n')
    if k==4:
        print('100% of the C values inspected \n')
        print ('Process completed')
    k=k+1


# In[512]:


FIN_SCORE=np.array(FIN_SCORE)
fin_score=FIN_SCORE.max()
i=FIN_SCORE.argmax()
c_max=c_list[i]
print ('The best classification is done between '+ str(c_list.min()) + ' and ' + str(c_list.max()) + '\n')
print('is ' + str(c_max) +', obtaining the following accuracy: '+str(fin_score*100)+'%')


# In[513]:


#Same process with C=1000
clf=SVC(C=1000, kernel='rbf')
clf.fit(X_train,y_train)
fin_score=clf.score(X_test,y_test)


# In[516]:


print('The score for C=1000 is ' + str(fin_score*100) +'%')


# In[517]:


#And C=10000
clf=SVC(C=10000, kernel='rbf')
clf.fit(X_train,y_train)
fin_score=clf.score(X_test,y_test)


# In[518]:


print('The score for C=10000 is ' + str(fin_score*100) +'%')


# In[90]:


#And C=1000000
clf=SVC(C=100000, kernel='rbf')
clf.fit(X_train,y_train)
fin_score=clf.score(X_test,y_test)


# In[91]:


#The best one is the last one, with 72%+ of accuracy
print('The score for C=100000 is '+str(fin_score*100) +'% \n')
print ('SVM best score: 72.2%')


# In[92]:


#Prediction computed on the test set
pred=clf.predict(X_test)


# In[93]:
```

```

#Target and Prediction comparison displayed
pred_data=pd.DataFrame()
pred_data['FirstComponent']= X_test['FirstComponent']
pred_data['SecondComponent']=X_test['SecondComponent']
pred_data['Target']=y_test
pred_data['Prediction']=pred


# In[94]:


pred_data.head()


# In[95]:


pred_data.to_csv('SVMprediction.csv')


# In[541]:


#Subplot of the prediction and the real target
plt.subplot(2,1,1)
sns.scatterplot(pred_data.FirstComponent,pred_data.SecondComponent,hue=pred_data.Target,palette='plasma')
plt.grid(True)
plt.xlabel('First Component', fontsize=20)
plt.ylabel('Second Component', fontsize=20)
plt.subplot(2,1,2)
sns.scatterplot(pred_data.FirstComponent,pred_data.SecondComponent,hue=pred_data.Prediction,palette='plasma')
plt.grid(True)
plt.xlabel('First Component', fontsize=20)
plt.ylabel('Second Component', fontsize=20)


# In[111]:


#Decision boundary plot
import matplotlib.cm as cm
xx, yy = np.meshgrid(np.linspace(-15, 20, 500),
                     np.linspace(-15, 25, 500))
Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

fig = plt.figure(figsize=(16,8))
fig.patch.set_facecolor('white')
ax = fig.gca()
imshow_handle = plt.imshow(Z, interpolation='nearest',
        extent=(xx.min(), xx.max(), yy.min(), yy.max()), aspect='auto',
        origin='lower', alpha=.5, cmap='plasma')
contours = plt.contour(xx, yy, Z, levels=[0], linewidths=2,
        linetypes='--', colors='red')
sns.scatterplot(pred_data.FirstComponent,pred_data.SecondComponent,hue=pred_data.Target,palette='plasma')
plt.xlabel('$x_1$', fontsize=14)
plt.ylabel('$x_2$', fontsize=14)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)
#plt.xlim(-3, 3)
#plt.ylim(-3, 3)
plt.legend()
plt.show()
```

5.5 NonLinearThreeComponentT

```
#!/usr/bin/env python
# coding: utf-8

```

```python
# # 4 Dataset Classification

# In[1]:


#Importing the libraries to watch the 'fits' image and get the data array
import astropy
#import plotly.graph_objects as go
from astropy.io import fits
#Importing a library that is useful to read the original file
import pandas as pd
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
#Importing a visual library with some illustrative set up
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib import cm
import numpy as np
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.decomposition import PCA
import math
import seaborn as sns
from sklearn.linear_model import LogisticRegression
plt.style.use('fivethirtyeight')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
#plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (16, 8)
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
plt.rcParams["axes.grid"] = False


# In[2]:


#Importing the dataset
data=pd.read_csv('star.txt',sep='\s+')


# In[3]:


data.head()


# In[4]:


#Excluding Sharp and #ID
notar=data.drop(columns=['Sharp','#ID'])


# # P.C.A. Excluding Space

# In[5]:


#Excluding Space Features in order to perform the P.C.A.
notar=data.drop(columns=['X','Y'])


```

```
# In[6]:


notar.head()


# In[7]:


pca=PCA(n_components=3)
pca=pca.fit(notar)
pca_data=pd.DataFrame(pca.transform(notar))


# In[8]:


pca_data=pca_data.rename(columns={0:'FirstComponent',1:'SecondComponent',2:'ThirdComponent'})


# In[9]:


#Stacking together the spatial coordinates
pca_data['X']=data.X
pca_data['Y']=data.Y


# In[10]:


#And the sharp
pca_data['Sharp']=data.Sharp


# In[11]:


pca_data.head()


# In[13]:


#pca_data=pca_data.drop(columns=['Sharp'])


# # Best method 3 Features

# In[12]:


#Two classification pre process, the Sharp is considered by its sign
data['SharpSign']=np.sign(data.Sharp)


# In[13]:


#The 0 values are considered as positive
data[data['SharpSign']==0]['SharpSign']=np.ones(len(data[data['SharpSign']==0]))


# In[14]:


#Three features are considered, the most informative
opt_data=pca_data[['FirstComponent','SecondComponent','ThirdComponent']]
opt_data['Target']=data.SharpSign


# In[15]:


#Target
y=opt_data.Target


# In[16]:


#dataset
X=opt_data.drop(columns=['Target'])
```

```


# In[17]:


#Importing SVM and train test split from sklearn model selection
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split


# In[18]:


#(Train+validation)/Test split with wide test set (90%)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.9, random_state=42)


# In[19]:


#Train/Validation split at half
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.5, random_state=42)


# In[20]:


#List of kernels
K_LIST=['linear', 'poly', 'rbf', 'sigmoid']


# In[ ]:


#Validation with CV_number=5 has been performed to choose the best kernel
BEST_KERNEL=[]
k=0
for i in range(5):
    FIN_SCORE=[]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.9, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train, test_size=0.5, random_state=42)
    print('Split Done')
    for ker in K_LIST:
        clf=SVC(kernel=ker)
        clf.fit(X_train,y_train)
        fin_score=clf.score(X_val,y_val)
        FIN_SCORE.append(fin_score)
        k=k+1
        print(ker + ' Kernel has been explored')
    FIN_SCORE=np.array(FIN_SCORE)

    BEST_KERNEL.append(K_LIST[FIN_SCORE.argmax()])
    print('Cross validation ' + str(i) + ' out of 4 \n')


# In[25]:


sns.countplot(np.array(BEST_KERNEL))
plt.xlabel('Chosen Kernel',fontsize=20)


# In[39]:


#Best kernel is chosen 5 times out of 5
best_kernel='rbf'


# In[97]:


#Cross validation on C values
```

```
c_list=np.arange(0.5,50.5,0.5)
k=0
PERC=['20%','40%','60%','80%','100%']
K=[20,40,60,80,100]
BEST_C=[]
for i in range(5):
    FIN_SCORE=[]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.9, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train, test_size=0.5, random_state=42)
    for c in c_list:
        k=k+1
        clf=SVC(C=c,kernel=best_kernel)
        clf.fit(X_train,y_train)
        fin_score=clf.score(X_val,y_val)
        FIN_SCORE.append(fin_score)
        #k=k+1
        if k in K:
            ind=K.index(k)
            print (PERC[ind] + ' of the C values has been explored')
    FIN_SCORE=np.array(FIN_SCORE)
    BEST_C.append(c_list[FIN_SCORE.argmax()])
    print('Cross validation ' + str(i) + ' out of 4 \n')


# In[100]:


sns.countplot(BEST_C)
plt.xlabel('Chosen C')
plt.ylabel('Count')


# In[101]:


FIN_SCORE=np.array(FIN_SCORE)


# In[102]:


#39 is chosen 5 times out of 5
best_c=c_list[FIN_SCORE.argmax()]


# In[107]:


#Train test split 70/30
X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size=0.7, random_state=42)


# In[108]:


#Prediction with best parameters
clf=SVC(kernel=best_kernel,C=best_c)
clf.fit(X_train,y_train)
fin_score=clf.score(X_test,y_test)


# In[119]:


prediction=clf.predict(X_test.drop(columns=['Target']))


# In[113]:


print('The final score with 3 feature is ' + str(fin_score*100) +'% ')


# In[121]:


#Comparing prediction and target
test_data=X_test.copy()
```

```python
test_data['Target']=y_test
test_data['Prediction']=prediction


# In[169]:


#Confusion matrix
import itertools
from string import ascii_uppercase
from sklearn.metrics import
    confusion_matrix

y_test=test_data.Target
predic = prediction

columns = ['Negative','Non Negative
    ']

confm = confusion_matrix(y_test,
    predic)
df_cm = pd.DataFrame(confm.astype(
    float), index=columns, columns=
    columns)

ax = sns.heatmap(df_cm, cmap='
    plasma',annot=True,fmt='g')


# In[170]:


#Defining precision and recall out
    of the confusion matrix
def precision(confusion):
    TP=confusion[0][0]
    TN=confusion[1][1]
    FP=confusion[0][1]
    FN=confusion[1][0]
    pres_a=TP/(TP+FN)
    pres_b=TN/(TN+FP)
    return [pres_a,pres_b]


# In[172]:


def recall(confusion):
    TP=confusion[0][0]
    TN=confusion[1][1]
    FP=confusion[0][1]
    FN=confusion[1][0]
    rec_a=TP/(TP+FP)
    rec_b=TN/(TN+FN)
    return [rec_a,rec_b]


# In[176]:


#Display the statistics
def statistics(confusion):
    stat=pd.DataFrame({'Negative':[
   precision(confusion)[0],recall(
   confusion)[0]],'Non Negative':[
   precision(confusion)[1],recall(
   confusion)[1]]})
    stat.index=['Precision','Recall
   ']
    return stat


# In[177]:


statistics(confm)
```

5.6 NonLinearThreeComponentT

```python
#!/usr/bin/env python
# coding: utf-8

# # 4 Dataset Classification

# In[3]:


#Importing the libraries to watch
   the 'fits' image and get the
   data array
import astropy
#import plotly.graph_objects as go
from astropy.io import fits
#Importing a library that is useful
    to read the original file
import pandas as pd
import pylab as plb
```

```python
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
#Importing a visual library with some illustrative set up
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib import cm
import numpy as np
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.decomposition import PCA
import math
import seaborn as sns
from sklearn.linear_model import LogisticRegression
plt.style.use('fivethirtyeight')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
#plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (16, 8)
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
plt.rcParams["axes.grid"] = False


# In[4]:


#Importing the dataset
data=pd.read_csv('star.txt',sep='\s+')


# In[6]:


data.head()


# In[7]:


#Excluding the target and the #ID
notar=data.drop(columns=['Sharp','#ID'])


# # P.C.A. Excluding Space
# In[6]:


#Excluding the target, the #ID and the spatial coordinates in order to apply the P.C.A.
notar=data.drop(columns=['Sharp','#ID','X','Y'])


# In[9]:


notar.head()


# In[7]:


pca=PCA(n_components=3)
pca=pca.fit(notar)
pca_data=pd.DataFrame(pca.transform(notar))
```

```


# In[8]:


pca_data=pca_data.rename(columns={0:'FirstComponent',1:'SecondComponent',2:'ThirdComponent'})


# In[57]:


#Stacking the X and Y coordinates
pca_data['X']=data.X
pca_data['Y']=data.Y


# In[58]:


#And the Sharp
pca_data['Sharp']=data.Sharp


# In[59]:


pca_data.head()


# In[60]:


#pca_data=pca_data.drop(columns=['Sharp'])


# # Best method 3 feature 3 classes

# In[12]:


#Considering three feature from the P.C.A. and three classes classification
opt_data=pca_data[['FirstComponent','SecondComponent','ThirdComponent']]
data['SharpSign']=data.Sharp.apply(np.sign)
opt_data['Target']=data['SharpSign']


# In[183]:


#Target and data
X=opt_data.drop(columns=['Target'])
y=opt_data.Target


# In[184]:


#Importing SVM and Model selection for train test split
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split


# In[185]:


#(Train+Validation)-Test with wide test set (90% of the data)
X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size=0.9, random_state=42)


# In[186]:


#Train Validation split with 50% of values
X_train, X_val, y_train, y_val = train_test_split(
 X_train, y_train, test_size=0.5, random_state=42)


# In[187]:

```

```python

#Computing the best kernel on the validation, training on the training set
K_LIST=['linear', 'poly', 'rbf', 'sigmoid']


# In[188]:


#Training and test validation varies 5 times in order to pick the best one
BEST_KERNEL=[]
k=0
for i in range(5):
    FIN_SCORE=[]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.9, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train, test_size=0.5, random_state=42)
    k=0
    for ker in K_LIST:
        clf=SVC(kernel=ker)
        clf.fit(X_train,y_train)
        fin_score=clf.score(X_val, y_val)
        FIN_SCORE.append(fin_score)
        k=k+1
        print(ker + ' Kernel has been explored')
    FIN_SCORE=np.array(FIN_SCORE)

    BEST_KERNEL.append(K_LIST[FIN_SCORE.argmax()])
    print('Cross validation ' + str(i) + ' out of 4 \n')


# In[190]:


CV_DATA=pd.DataFrame({'CV Number':np.arange(1,6,1),'Choosen Kernel': BEST_KERNEL})


# In[191]:


sns.countplot(CV_DATA['Choosen Kernel'])


# In[192]:


best_kernel='rbf'


# In[194]:


#The same form of validation is made for the C value
c_list=np.arange(0.5,50.5,0.5)
k=0
PERC=['20%','40%','60%','80%','100%']
K=[20,40,60,80,100]
BEST_C=[]
for i in range(5):
    FIN_SCORE=[]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.9, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train, test_size=0.5, random_state=42)
    k=0
    for c in c_list:
        k=k+1
        clf=SVC(C=c,kernel=best_kernel)
        clf.fit(X_train,y_train)
        fin_score=clf.score(X_val, y_val)
        FIN_SCORE.append(fin_score)
        #k=k+1
        if k in K:
            ind=K.index(k)
```

```
            print (PERC[ind] + ' of the C values has been explored')
    FIN_SCORE=np.array(FIN_SCORE)
    BEST_C.append(c_list[FIN_SCORE.argmax()])
    print('Cross validation ' + str(i) + ' out of 4 \n')


# In[195]:


sns.countplot(BEST_C)
plt.xlabel('Chosen C')
plt.ylabel('Count')


# In[196]:


FIN_SCORE=np.array(FIN_SCORE)


# In[197]:


best_c=c_list[FIN_SCORE.argmax()]


# In[14]:


best_c=17
best_kernel='rbf'


# In[199]:


#Train-test split at 70% has been made, and the performance are computed
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.7, random_state=42)


# In[200]:


#Computing the score
clf=SVC(kernel=best_kernel,C=best_c)
clf.fit(X_train,y_train)
fin_score=clf.score(X_test,y_test)


# In[203]:


#The prediction
prediction=clf.predict(X_test)


# In[204]:


print('The final score with 3 feature is ' + str(fin_score*100) +'% ')


# In[205]:


test_data=X_test.copy()
test_data['Target']=y_test
test_data['Prediction']=prediction


# In[214]:


#And the confusion matrix
import itertools
from string import ascii_uppercase
from sklearn.metrics import confusion_matrix

y_test=test_data.Target
predic = prediction

columns = ['Negative','Zero','Positive']

confm = confusion_matrix(y_test,
```

```
predic)
df_cm = pd.DataFrame(confm.astype(float), index=columns, columns=columns)

ax = sns.heatmap(df_cm, cmap='plasma',annot=True,fmt='g')


# In[227]:


#Defining precision and recall on three classes classification
def precision(confusion,clas):
    if clas=='Negative':
        TP=confusion[0][0]
        FN=confusion[1][0]+confusion[2][0]
        pres=TP/(TP+FN)
    if clas=='Positive':
        TP=confusion[2][2]
        FN=confusion[2][0]+confusion[2][1]
        pres=TP/(TP+FN)
    if clas=='Zero':
        TP=confusion[1][1]
        FN=confusion[1][0]+confusion[1][2]
        pres=TP/(TP+FN)
    return pres


# In[235]:


def recal(confusion,clas):
    if clas=='Negative':
        TP=confusion[0][0]
        FP=confusion[0][1]+confusion[0][2]
        rec=TP/(TP+FP)
    if clas=='Positive':
        TP=confusion[2][2]
        FP=confusion[0][2]+confusion[1][2]
        rec=TP/(TP+FP)
    if clas=='Zero':
        TP=confusion[1][1]
        FP=confusion[0][1]+confusion[2][1]
        rec=TP/(TP+FP)
    return rec


# In[236]:


precision(confm,'Negative'),precision(confm,'Positive'),precision(confm,'Zero')


# In[237]:


recal(confm,'Negative'),recal(confm,'Positive'),recal(confm,'Zero')


# In[238]:


#Summary of the Precision and Recall for each classes function
def statistics(confusion):
    neg=[precision(confm,'Negative'),recal(confm,'Negative')]
    pos=[precision(confm,'Positive'),recal(confm,'Positive')]
    zero=[precision(confm,'Zero'),recal(confm,'Zero')]
    stats=pd.DataFrame({'Negative':neg,'Positive':pos,'Zero':zero})
    stats.index=['Precision','Recall']
    return stats


# In[239]:


statistics(confm)
```

5.7 Decision Tree.ipynb

```python
#!/usr/bin/env python
# coding: utf-8

# # Decision Tree

# In[68]:


#Importing a library that is useful to read the original file
import pandas as pd
#Importing a visual library with some illustrative set up
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import matplotlib.colors as mcolors
from matplotlib import cm
import math
plt.style.use('fivethirtyeight')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (16, 8)
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
colors = ['xkcd:pale orange', 'xkcd:sea blue', 'xkcd:pale red', 'xkcd:sage green', 'xkcd:terra cotta', 'xkcd:dull purple', 'xkcd:teal', 'xkcd: goldenrod', 'xkcd:cadet blue',
'xkcd:scarlet']
cmap_big = cm.get_cmap('Spectral', 512)
cmap = mcolors.ListedColormap(cmap_big(np.linspace(0.7, 0.95, 256)))
bbox_props = dict(boxstyle="round,pad=0.3", fc=colors[0], alpha=.5)


# In[2]:


from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
from pydot import graph_from_dot_data
import pandas as pd
import numpy as np


# Load dataset

# In[3]:


data_original=pd.read_csv('/Users/Simone/Desktop/Programmi/DDA/hlsp_deep47tuc_hst_acs_47tuc_f606w-f814w_v1_catalog.txt', delimiter='\s+')
```

```
data_original.head()


# Drop the #ID column

# In[4]:


Sharp = data_original.Sharp
data = data_original.drop(columns = ['#ID','Sharp'])


# In[5]:


data.head()


# create target column, 3 classes : positive, negative and zero Sharp

# In[6]:


data['target'] = np.zeros_like(len(data))


# In[7]:


len(data.target[Sharp > 0])


# In[8]:


data.target[Sharp > 0] = 1


# In[9]:


data.target[Sharp == 0] = 2


# Let's split dataset in training and test set, we use a categorical encoding, this way we can add more classes if needed

# In[10]:


feature_names = ['X','Y','F606W','error','F814W','error.1','Chi']


# In[11]:


X = pd.DataFrame(data, columns=feature_names)
y = pd.Categorical.from_codes(data.target, ['Negative', 'Positive', 'Zero'])


# In[12]:


X.head()


# we use one-hot encoding where sharps value targets are vectors :
# Negative -> [100]
# Positive -> [010]
# Zero -> [001]
#

# In[13]:


y = pd.get_dummies(y)
y.head()


# Splito dataset in train and test

# In[15]:


X_train, X_test, y_train, y_test =
```

```
    train_test_split(X, y, train_size = 0.85, random_state=1)


# First of all we check how a standard decision tree performs on our dataset

# In[16]:


dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)


# we can print the output wuth the following lines, however without limitations on leaf number we expect to have a veri huge tree.

# In[ ]:


dot_data = StringIO()
export_graphviz(dt, out_file=dot_data, feature_names=feature_names)
(graph, ) = graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())


# Lets'check confusion matrix and accuracy of a standard decision tree

# In[19]:


y_pred = dt.predict(X_test)


# Lets visualize Confusion matrix

# In[65]:


species = np.array(y_test).argmax(axis=1)
predictions = np.array(y_pred).argmax(axis=1)
cf_matrix = confusion_matrix(species, predictions)

group_names = ['True Neg','False Pos','False zero (N)','False Neg',
               'True Pos', 'False zero (P)','False Neg (0)','False Pos (0)','True zero']
group_counts = ['{0:0.0f}'.format(value) for value in
                cf_matrix.flatten()]
group_percentages = ['{0:.2%}'.format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(3,3)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')


# In[45]:


from sklearn.metrics import accuracy_score

accuracy_score(y_test, y_pred)


# Decision tree works alredy better then SVM Alredy, maybe a little bit of parameter tuning would help.
# However its interesting checking if a boosting of this method could lead to better results, lets consider a random forest
```

```
    classifier

# # Random Forest

# In[46]:


#Import Random Forest Model
from sklearn.ensemble import
    RandomForestClassifier

#Create a Gaussian Classifier
rf=RandomForestClassifier()

# Train the model on training data
rf.fit(X_train, y_train);


# In[50]:


# Use the forest's predict method
    on the test data
rf_y_pred = rf.predict(X_test)


# In[51]:


accuracy_score(y_test, rf_y_pred)


# it seems to work slightly better,
     trought RandomForestClassifier
     we could check a sort of
    feature ranking

# In[52]:


feature_imp = pd.Series(rf.
    feature_importances_,index=
    feature_names).sort_values(
    ascending=False)


# In[66]:


import seaborn as sns
get_ipython().run_line_magic('
    matplotlib', 'inline')
# Creating a bar plot
sns.barplot(x=feature_imp, y=
    feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance
    Score')
plt.ylabel('Features')
plt.title("Visualizing Important
    Features")
plt.legend()
plt.show()


# Random forest is in general more
    interesting then a single tree,
     so we are going to tune
    parameter for this model,
    hoping in a significative
    increase in terms of
    performance.

# In[57]:


from sklearn.model_selection import
     validation_curve


# we are going to check most of the
     features wich are fixed in the
     standard
    RandomForestClassifier function
     throught a validation process

# In[59]:


num_estNum = [10, 50, 100, 200,500]
train_scoreNum, test_scoreNum =
    validation_curve(

    RandomForestClassifier(),
                                 X =
     X_train, y = y_train,

    param_name = 'n_estimators',
```

```
                                                    param_range = num_estNum , cv = 3)


# Greater is the number of estimators , better seems to be the performance in test set

# In[73]:


plt.plot(num_estNum,test_scoreNum, label = 'Test')
plt.plot(num_estNum,train_scoreNum, label = 'Train')
plt.grid(True)
plt.title('Cross Validation for n_estimator')
plt.ylabel('Accuracy [%]')
plt.xlabel('Parameter value')
plt.legend()
plt.show()


# we do this for differesnt features , saving best results

# In[74]:


num_estDepth = [10, 50, 100, 200,500,1000]
train_scoreDepth, test_scoreDepth= validation_curve(
                                                    RandomForestClassifier(n_estimators=200),
                                                    X = X_train, y = y_train,
                                                    param_name = 'max_depth',
                                                    param_range = num_estDepth, cv = 3)


# In[77]:


plt.plot(num_estDepth, test_scoreDepth,label = 'Test')
plt.plot(num_estDepth, train_scoreDepth,label = 'Train')
plt.grid(True)
plt.title('Cross Validation for max_depth')
plt.ylabel('Accuracy [%]')
plt.xlabel('Parameter value')
plt.legend()
plt.show()


# In[79]:


num_estSplit = [1.,2, 5,10, 15, 20,50]
train_scoreSplit, test_scoreSplit= validation_curve(
                                                    RandomForestClassifier(max_depth = 10),
                                                    X = X_train, y = y_train,
                                                    param_name = 'min_samples_split',
                                                    param_range = num_estSplit, cv = 3)


# In[84]:


plt.plot(num_estSplit, test_scoreSplit,label = 'Test')
plt.plot(num_estSplit, train_scoreSplit,label = 'Train')
plt.grid(True)
plt.title('Cross Validation for min_samples_split')
plt.ylabel('Accuracy [%]')
plt.xlabel('Parameter value')
plt.legend()
```

```
plt.show()


# In[85]:


num_estLeaf = [1,5,15,50,1000]
train_scoreLeaf, test_scoreLeaf= validation_curve(
    RandomForestClassifier(max_depth = 10, min_samples_split=15),
                                                          X = X_train, y = y_train,
    param_name = 'min_samples_leaf',
    param_range = num_estLeaf, cv = 3)


# In[88]:


plt.plot(num_estLeaf,test_scoreLeaf, label = 'Test')
plt.plot(num_estLeaf, train_scoreLeaf,label = 'Train')
plt.grid(True)
plt.title('Cross Validation for min_samples_leaf')
plt.ylabel('Accuracy [%]')
plt.xlabel('Parameter value')
plt.legend()
plt.show()


# In[ ]:





# In[ ]:





# In[87]:


num_est = ['auto', 'sqrt', 'log2',1,2,3,4,5,6,7]
train_scoreMaxFeat, test_scoreMaxFeat= validation_curve(
    RandomForestClassifier(max_depth = 10, min_samples_split=15, min_samples_leaf=2),
                                                          X = X_train, y = y_train,
    param_name = 'max_features',
    param_range = num_est, cv = 3)


# In[89]:


plt.plot(num_est,test_scoreMaxFeat, label = 'Test')
#plt.plot(num_est, train_scoreMaxFeat,label = 'Train')
plt.grid(True)
plt.title('Cross Validation for max_features')
plt.ylabel('Accuracy [%]')
plt.xlabel('Parameter value')
plt.legend()
plt.show()


# At this point we are ready to build the forest with tuned parameters

# In[91]:


#Import Random Forest Model
```

```
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
rf=RandomForestClassifier(n_estimators=500, max_features = 6, max_depth = 50, min_samples_split=15, min_samples_leaf=1)

# Train the model on training data
rf.fit(X_train, y_train);


# In[92]:


# Use the forest's predict method on the test data
rf_y_pred = rf.predict(X_test)


# In[93]:


accuracy_score(y_test, rf_y_pred)


# we could print one of the trees usend in the forest (if the lenght of the tree is not specified, the result could be a large PNG image)

# In[ ]:


# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rf.estimators_[5]
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot
# Pull out one tree from the forest
tree = rf.estimators_[5]
# Export the image to a dot file
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_names, rounded = True, precision = 1)
# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')
Image(graph.create_png())


# Let's check Feature importance at this level

# In[94]:


feature_imp = pd.Series(rf.feature_importances_,index=feature_names).sort_values(ascending=False)


# In[95]:


import seaborn as sns
get_ipython().run_line_magic('matplotlib', 'inline')
# Creating a bar plot
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()


# 3d Plots of the dataset projecion on the 3 most important features, comparing model(1st plot) with actual labels (second)
```

```python

# In[96]:


Results = X_test.copy()
decoded_model = np.argmax(rf_y_pred, axis=1)
decoded_target = np.argmax(np.asarray(y_test), axis=1)


# In[97]:


Results['actual'] = decoded_target
Results['model'] = decoded_model


# In[ ]:



import plotly.express as px

fig = px.scatter_3d(Results, x='error', y='Y', z='Chi',
                    color='model')
fig.show()


# In[ ]:



import plotly.express as px

fig = px.scatter_3d(Results, x='error', y='Y', z='Chi',
                    color='actual')
fig.show()


# In[ ]:
```

5.8 Classification.ipynb

```python
#!/usr/bin/env python
# coding: utf-8

# # Best Classification

# In[6]:


#Importing the libraries to watch the 'fits' image and get the data array
import astropy
#import plotly.graph_objects as go
from astropy.io import fits
#Importing a library that is useful to read the original file
import pandas as pd
import pylab as plb
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy import asarray as ar,exp
#Importing a visual library with some illustrative set up
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
from matplotlib import cm
import numpy as np
from sklearn.utils.testing import ignore_warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.decomposition import PCA
import math
import seaborn as sns
from sklearn.linear_model import LogisticRegression
plt.style.use('fivethirtyeight')
plt.rcParams['font.family'] = 'sans-serif'
plt.rcParams['font.serif'] = 'Ubuntu'
plt.rcParams['font.monospace'] = 'Ubuntu Mono'
plt.rcParams['font.size'] = 14
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.labelweight'] = 'bold'
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['xtick.labelsize'] =
```

```
    12
plt.rcParams['ytick.labelsize'] = 12
#plt.rcParams['legend.fontsize'] = 12
plt.rcParams['figure.titlesize'] = 12
plt.rcParams['image.cmap'] = 'jet'
plt.rcParams['image.interpolation'] = 'none'
plt.rcParams['figure.figsize'] = (16, 8)
plt.rcParams['lines.linewidth'] = 2
plt.rcParams['lines.markersize'] = 8
plt.rcParams["axes.grid"] = False


# In[4]:


#Importing the dataset and displaying the first 5 rows
data=pd.read_csv('star.txt',sep='\s+')


# In[6]:


data.head()


# In[7]:


#Dropping the Sharp and the #ID from the dataset
notar=data.drop(columns=['Sharp','#ID'])


# # P.C.A. Excluding Space

# In[6]:


#Dropping the X and the Y to perform PCA
notar=data.drop(columns=['X','Y'])


# In[9]:


notar.head()


# In[7]:


#Performing the P.C.A.
pca=PCA(n_components=3)
pca=pca.fit(notar)
pca_data=pd.DataFrame(pca.transform(notar))


# In[8]:


pca_data=pca_data.rename(columns={0:'FirstComponent',1:'SecondComponent',2:'ThirdComponent'})


# In[57]:


pca_data['X']=data.X
pca_data['Y']=data.Y


# In[58]:


pca_data['Sharp']=data.Sharp


# In[59]:


pca_data.head()


# In[60]:

```

```

#pca_data=pca_data.drop(columns=['Sharp'])


# # Best method 2 features 3 classes

# In[59]:


#Two features, three classes preprocessing
opt_data=pca_data[['FirstComponent','SecondComponent']]


# In[60]:


data['SharpSign']=data.Sharp.apply(np.sign)
opt_data['Target']=data['SharpSign']


# In[14]:


#plotting the First two component, together with the target
sns.scatterplot(opt_data.FirstComponent,opt_data.SecondComponent,hue=opt_data.Target,palette='plasma')
plt.grid(True)


# In[63]:


#Dataset and target
X=opt_data.drop(columns=['Target'])
y=opt_data.Target


# In[20]:


#Importing the SVM and the train test split
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split


# In[62]:


# (Train+validation)/ test split at 90%
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.9, random_state=42)


# In[18]:


# Train/Validation split at 50%
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size=0.5, random_state=42)


# In[19]:


K_LIST=['linear', 'poly', 'rbf', 'sigmoid']


# In[20]:


#Validation on kernels
BEST_KERNEL=[]
k=0
for i in range(5):
    FIN_SCORE=[]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.9, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(
```

```
        X_train, y_train, test_size=0.5, random_state=42)
    for ker in K_LIST:
        clf=SVC(kernel=ker)
        clf.fit(X_train,y_train)
        fin_score=clf.score(X_val,y_val)
        FIN_SCORE.append(fin_score)
        k=k+1
        print(ker + ' Kernel has been explored')
    FIN_SCORE=np.array(FIN_SCORE)

    BEST_KERNEL.append(K_LIST[FIN_SCORE.argmax()])
    print('Cross validation ' + str(i) + ' out of 4 \n')


# In[7]:


BEST_KERNEL=['rbf','rbf','rbf','rbf','rbf']


# In[9]:


sns.countplot(BEST_KERNEL)
plt.xlabel('Chosen Kernel',fontsize=20)


# In[22]:


best_kernel='rbf'


# In[23]:


#Validation on C values
c_list=np.arange(0.5,50.5,0.5)
k=0
PERC=['20%','40%','60%','80%','100%']
K=[20,40,60,80,100]
BEST_C=[]
for i in range(5):
    FIN_SCORE=[]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.9, random_state=42)
    X_train, X_val, y_train, y_val = train_test_split(
        X_train, y_train, test_size=0.5, random_state=42)
    k=0
    for c in c_list:
        k=k+1
        clf=SVC(C=c,kernel=best_kernel)
        clf.fit(X_train,y_train)
        fin_score=clf.score(X_val,y_val)
        FIN_SCORE.append(fin_score)
        #k=k+1
        if k in K:
            ind=K.index(k)
            print (PERC[ind] + ' of the C values has been explored')
    FIN_SCORE=np.array(FIN_SCORE)
    BEST_C.append(c_list[FIN_SCORE.argmax()])
    print('Cross validation ' + str(i) + ' out of 4 \n')


# In[24]:


sns.countplot(BEST_C)
plt.xlabel('Chosen C')
plt.ylabel('Count')


# In[25]:


FIN_SCORE=np.array(FIN_SCORE)


# In[15]:

```

```
#Best parameters
best_c=14.5
best_kernel='rbf'


# In[64]:


#Train/Test rigid split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.7, random_state=42)


# In[65]:


#Fit with the best parameters
clf=SVC(kernel=best_kernel,C=best_c)
clf.fit(X_train,y_train)
fin_score=clf.score(X_test,y_test)


# In[66]:


prediction=clf.predict(X_test)


# In[67]:


#Results
print('The final score with 2 feature is ' + str(fin_score*100) +'% ')


# In[68]:


#Target/prediction comparison
pred_data=X_test.copy()
pred_data['Target']=y_test
pred_data['Prediction']=prediction


# In[34]:


#plotting the decision surfaces
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
def make_meshgrid(x, y, h=.4):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out


# In[32]:


#renaming
y=y_test


# In[35]:


fig, ax = plt.subplots()
# title for the plots
title = ('Decision surface of linear SVC ')
# Set-up grid for plotting.
X0, X1 = X_test['FirstComponent'], X_test['SecondComponent']
xx, yy = make_meshgrid(X0, X1)

plot_contours(ax, clf, xx, yy, cmap='plasma', alpha=0.8)
```

```
ax.scatter(X0, X1, c=y, cmap='plasma', s=20, edgecolors='k')
ax.set_ylabel('Second Component')
ax.set_xlabel('First Component')


violet_patch = mpatches.Patch(color='navy', label='Sharp<0')
yellow_patch = mpatches.Patch(color='gold', label='Sharp>0')
pink_patch = mpatches.Patch(color='magenta', label='Sharp=0')

plt.legend(handles=[violet_patch, yellow_patch,pink_patch])

ax.set_xticks(())
ax.set_yticks(())
ax.set_title('Decision Surface', fontsize=20)
#ax.legend()
plt.show()


# In[36]:


test_data=pred_data


# In[37]:


#Confusion matrix
import itertools
from string import ascii_uppercase
from sklearn.metrics import confusion_matrix

y_test=test_data.Target
predic = prediction

columns = ['Negative','Zero','Positive']

confm = confusion_matrix(y_test, predic)
df_cm = pd.DataFrame(confm.astype(float), index=columns, columns=columns)

ax = sns.heatmap(df_cm, cmap='plasma',annot=True,fmt='g')


# In[103]:


#Defining precision and recall for three classes classification
def precision(confusion,clas):
    if clas=='Negative':
        TP=confusion[0][0]
        FN=confusion[1][0]+confusion[2][0]
        pres=TP/(TP+FN)
    if clas=='Positive':
        TP=confusion[2][2]
        FN=confusion[2][0]+confusion[2][1]
        pres=TP/(TP+FN)
    if clas=='Zero':
        TP=confusion[1][1]
        FN=confusion[1][0]+confusion[1][2]
        pres=TP/(TP+FN)
    return pres


# In[104]:


def recal(confusion,clas):
    if clas=='Negative':
        TP=confusion[0][0]
        FP=confusion[0][1]+confusion[0][2]
        rec=TP/(TP+FP)
    if clas=='Positive':
        TP=confusion[2][2]
        FP=confusion[0][2]+confusion[1][2]
        rec=TP/(TP+FP)
    if clas=='Zero':
        TP=confusion[1][1]
        FP=confusion[0][1]+confusion[2][1]
        rec=TP/(TP+FP)
```

```
    return rec


# In[41]:


precision(confm,'Negative'),precision(confm,'Positive'),precision(confm,'Zero')


# In[42]:


recal(confm,'Negative'),recal(confm,'Positive'),recal(confm,'Zero')


# In[44]:


#Defining statistics summary
def statistics(confusion):
    neg=[precision(confm,'Negative'),recal(confm,'Negative')]
    pos=[precision(confm,'Positive'),recal(confm,'Positive')]
    zero=[precision(confm,'Zero'),recal(confm,'Zero')]
    stats=pd.DataFrame({'Negative':neg,'Positive':pos,'Zero':zero})
    stats.index=['Precision','Recall']
    return stats


# In[45]:


statistics(confm)


# # Best Method

# In[70]:


#On the yellow data, where the algorithm performs poorly, a Random Forest algorithm is applied
wrong_data=pred_data[pred_data['Prediction']==1.0].drop(columns=['Target'])
wrong_target=pred_data[pred_data['Prediction']==1.0].Target


# In[71]:


#Plotting the difficult to predict data points
sns.scatterplot(wrong_data.FirstComponent,wrong_data.SecondComponent,hue=wrong_target)


# In[72]:


#Implement random forest on a split set
X=data.loc[X_test.index].drop(columns=['#ID','Sharp','SharpSign'])
y=data.loc[X_test.index].SharpSign
X_train, X_test, y_train, y_test = train_test_split(
 X, y, test_size=0.2, random_state=42)


# In[73]:


#Import Random Forest Model
from sklearn.ensemble import RandomForestClassifier



#Create a Gaussian Classifier
rf=RandomForestClassifier(n_estimators=500,
```

```
    min_samples_split = 20,
    max_features = 5)


# Train the model on training data
rf.fit(X_train, y_train)


# In[74]:


feature_names = ['X','Y','F606W','error','F814W','error.1','Chi']
# Creating a bar plot
feature_imp = pd.Series(rf.feature_importances_,index=feature_names).sort_values(ascending=False)
sns.barplot(x=feature_imp, y=feature_imp.index)
# Add labels to your graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Visualizing Important Features")
plt.legend()
plt.show()


# In[75]:


#Collecting the results of the random forest on this middle area
#And the SVM on the -1 and 0 points
#Combining the features for the Random Forest results
Results=X_test.copy()
Results['Target']=y_test
Results['Pred']=rf.predict(X_test)
Results['FirstComponent']=opt_data['FirstComponent'].loc[Results.index]
Results['SecondComponent']=opt_data['SecondComponent'].loc[Results.index]


# In[76]:


#Doing the same for the SVM results
good_ones=pred_data[(pred_data.Prediction==0.) |(pred_data.Prediction==-1.0)]


# In[79]:


for feat in feature_names:
     good_ones[feat]=data[feat].loc[good_ones.index]


# In[80]:


good_ones=good_ones.rename(columns={'Prediction':'Pred'})


# In[81]:


#Building the total resume
Results=Results.append(good_ones)


# In[82]:


#Computing accuracy
from sklearn.metrics import accuracy_score
acc=accuracy_score(Results.Pred, Results.Target)


# In[84]:


#Computing the confusion matrix
import itertools
from string import ascii_uppercase
```

```
from sklearn.metrics import confusion_matrix

y_test=Results.Target
predic = Results.Pred

columns = ['Negative','Zero','Positive']

confm = confusion_matrix(y_test, predic)
df_cm = pd.DataFrame(confm.astype(float), index=columns, columns=columns)

ax = sns.heatmap(df_cm, cmap='plasma',annot=True,fmt='g')


# In[105]:


#Using the statistics function to see the precisions and the recall
def statistics(confusion):
    neg=[precision(confm,'Negative'),recal(confm,'Negative')]
    pos=[precision(confm,'Positive'),recal(confm,'Positive')]
    zero=[precision(confm,'Zero'),recal(confm,'Zero')]
    stats=pd.DataFrame({'Negative':neg,'Positive':pos,'Zero':zero})
    stats.index=['Precision','Recall']
    return stats


# In[106]:


statistics(confm)


# In[94]:


#Summary of all the methods
Tot_res=pd.DataFrame({'Performance':[71,74,80,82]})
Tot_res.index=['SVM','Decision Tree','Random Forest','Ensemble Learning']
Tot_res


# In[100]:


#Barplot of all the methods performances
sns.barplot(x=Tot_res.index,y=Tot_res.Performance,palette='plasma')
plt.xlabel('Method',fontsize=20)
plt.ylabel('Accuracy (%)',fontsize=20)
plt.grid(True)
```

Bibliography

[1] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[2] Piero Paialunga. *P.C.A. meets explainability*. 2020. URL: https://towardsdatascience.com/p-c-a-meets-explainability-ba1ba5e4636.

[3] Roman Cheplyaka. *Explained variance in PCA*. 2017. URL: https://ro-che.info/articles/2017-12-11-pca-explained-variance#:~:text=The%20total%20variance%20is%20the,divide%20by%20the%20total%20variance..

[4] Wikipedia contributors. *Confusion matrix — Wikipedia, The Free Encyclopedia*. [Online; accessed 8-December-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=980584181.

[5] Wikipedia contributors. *Magnitude (astronomy) — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Magnitude_(astronomy)&oldid=983656469. [Online; accessed 16-October-2020]. 2020.

[6] Wikipedia contributors. *Mutual information — Wikipedia, The Free Encyclopedia*. [Online; accessed 7-December-2020]. 2020. URL: https://en.wikipedia.org/w/index.php?title=Mutual_information&oldid=991412922.