

Report 3 : Machine Learning methods application in an astrophysics dataset

Piero Paialunga and Simone Chierichini

October 2020

Contents

1	Introduction	1
2	Data Description	2
3	Methods Description	3
3.1	Machine Learning framework	3
3.1.1	Classification Evaluation Metrics	4
3.2	Principal Component Analysis	5
3.3	The Datasets	8
3.4	Support Vector Machines	10
3.4.1	Theoretical Background	10
3.4.2	Two Classes classification	11
3.4.3	Three Classes classification	14
3.5	Decision Trees	16
3.5.1	Ensamble Methods and Random Forest	18
3.5.2	Building the model	19
3.5.3	Best Model	20
4	Conclusion	22
5	Appendix	24
5.1	MachineLearningUtility.ipynb	25
5.2	PCA.ipynb	26
5.3	Linear.ipynb	32
5.4	NonLinearTwoComponent.ipynb	41
5.5	NonLinearThreeComponentTwoC.ipynb	47
5.6	NonLinearThreeComponentThreeC.ipynb	52
5.7	Decision Tree.ipynb	58
5.8	Classification.ipynb	65

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 unknown
- **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:

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

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.

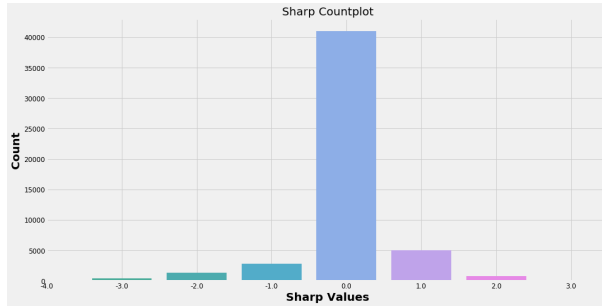


Figure 3.1: As it is possible to see from the count-plot, the Sharp distribution is not entirely positive.

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 metrics 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 precise, 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 verify 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.

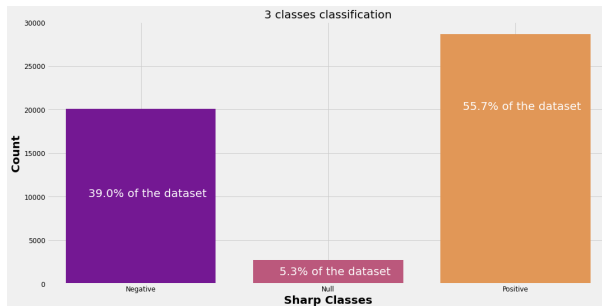


Figure 3.2: **3 classes dataset**. As it is possible to see from the plot, the three classes dataset target is not balanced.

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

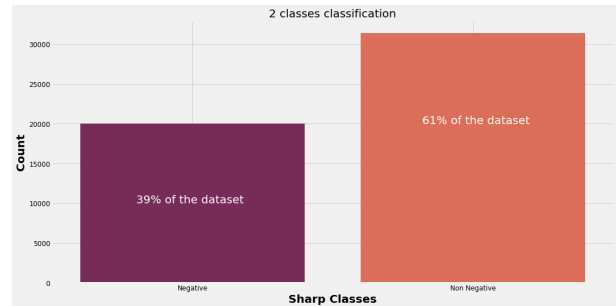


Figure 3.3: **2 classes dataset**. As it is possible to see from the plot, the two classes dataset target is not balanced.

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

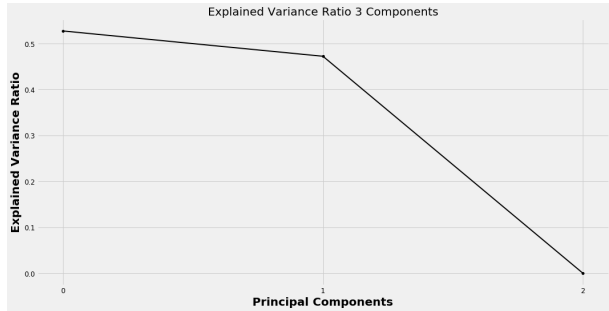


Figure 3.4: **Explained Variance Ratio Plot.** As it is possible to see from the Explained Variance Ratio plot, almost all the variance has been captured by a three component P.C.A.

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 means 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**

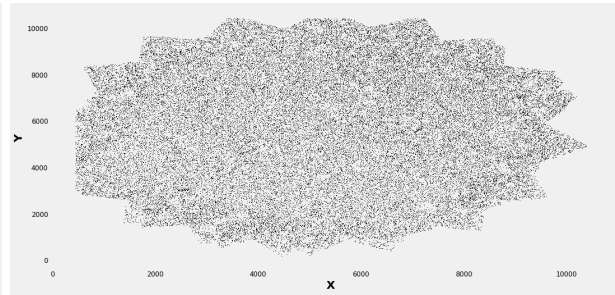


Figure 3.5: **X vs Y Scatter Plot.**

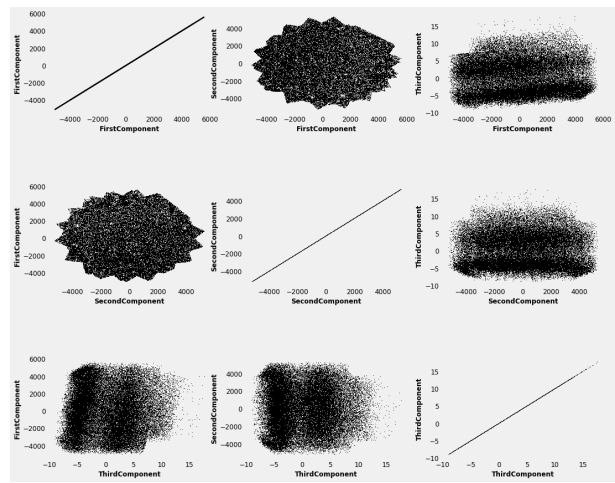


Figure 3.6: **Three Components Principal Component Analysis.** As it is possible to see, FirstComponent vs SecondComponent plot shows the same scatterplot of the X vs Y scatter plot. The first and second P.C.A. Components collapse on the spatial coordinates.

nent 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-**

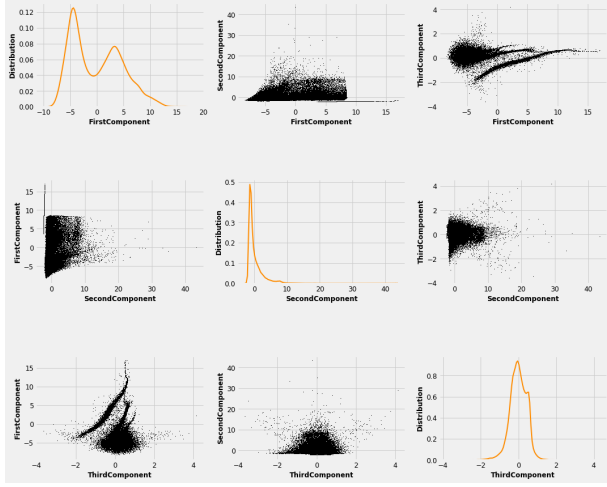


Figure 3.7: **Three Components Principal Component Analysis on the dataset deprived of its spatial coordinates.** As it is possible to see, this P.C.A. permitted to obtain non trivial new features.

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 Third-Component feature (T) is given by the following equation:

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

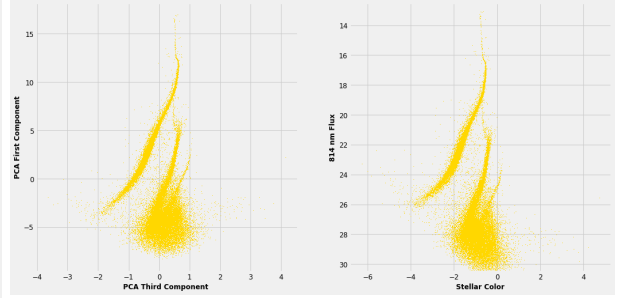


Figure 3.8: **Comparison between the FirstComponent vs ThirdComponent and the 814 nm flux vs Stellar Color plots .** As it is possible to see, the shape of the two plots is notably similar.

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

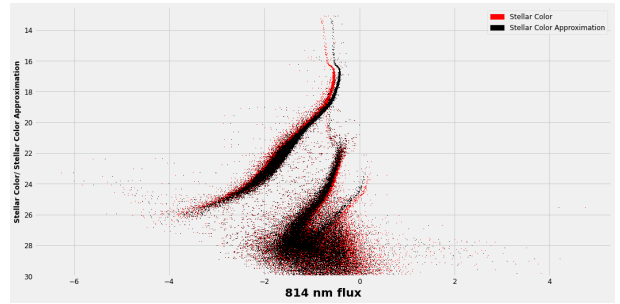


Figure 3.9: **Stellar Color and Stellar Color approximation plot .** As it is possible to see, the Stellar Color is notably close to its approximation.

As it was already explained, the Second-Component (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.

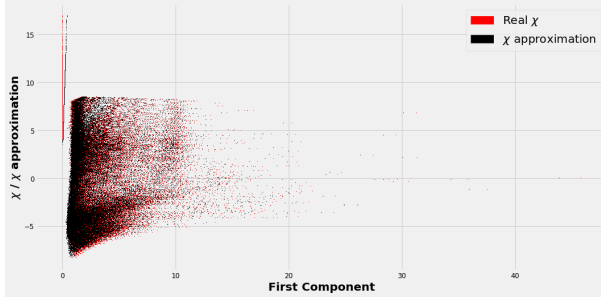


Figure 3.10: χ and χ approximation plot . As it is possible to see, the χ value is notably close to its approximation.

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.

$$1 \times A = -1.4, B = +18.6, C = -1, D = -1, E = 10^{-14} \text{ kg}$$

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.

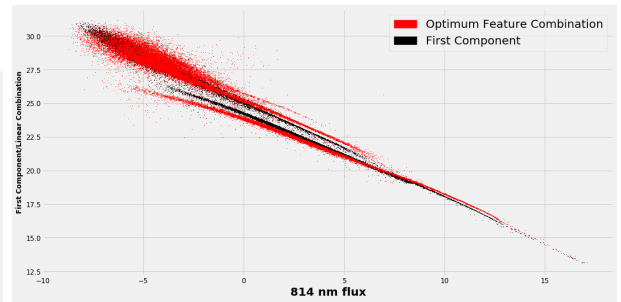


Figure 3.11: F and F approximation plot . As it is possible to see, the F feature is notably close to its approximation.

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 losing 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:

$$KL = - \int \int p(x, y) \ln \left(\frac{p(x)p(y)}{p(x, y)} \right) dx dy$$

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:

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

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:

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.

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.

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:

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

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 maximizing 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:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left(-\frac{\|\mathbf{x}_n - \mathbf{x}_m\|^2}{2\sigma^2}\right)$$

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 known 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 $C \rightarrow \infty$ 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). Nonetheless, 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 distinction 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.

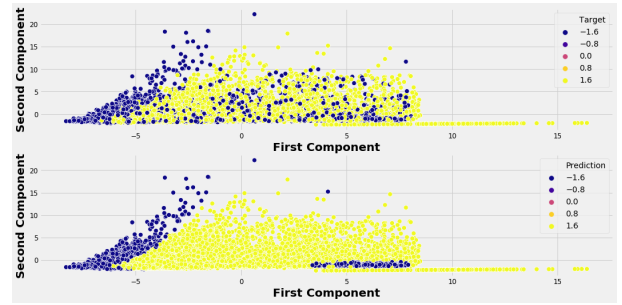


Figure 3.14: Real Classification vs Predicted Classification. As it is possible to see, the SVM performs well in the extreme part of the dataset, but are not able to discern the classification on the interior, as the classification distributions overlap.

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.

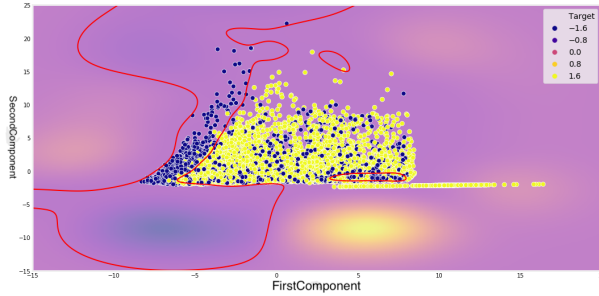


Figure 3.15: **SVM Decision Boundary.** The decision boundary of the algorithm is almost exact at the extremal zones of the dataset, but is not correct in the interior.

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 param-

eter obtained was $C_{opt} = 39$. The optimal classification gave a total accuracy of 71.7%, obtained extracting the values from the following confusion matrix.

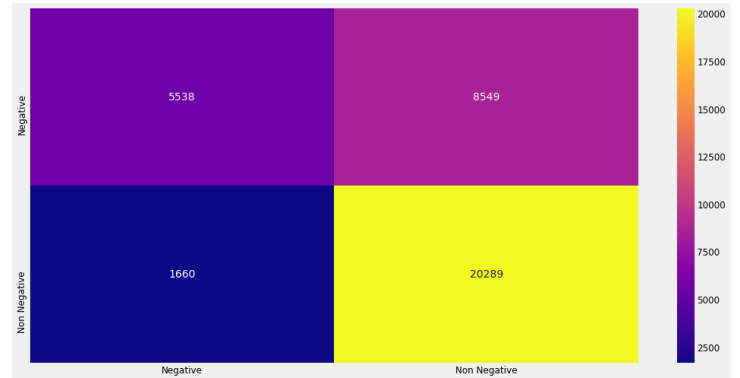


Figure 3.16: **Three Features, Two Classes classification confusion matrix.** As it is possible to appreciate, the performance of the algorithm is better on the Negative class in terms of precision, while it is worst in terms of recall.

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 approximated 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.

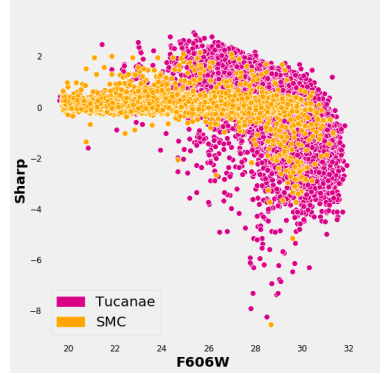


Figure 3.18: **Sharp vs 606 nm flux.** Zero sharp values correspond to low. In Report 1, a distinction (not relevant in this work) about the SMC and Tucanae has been done.

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 appreciate 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.

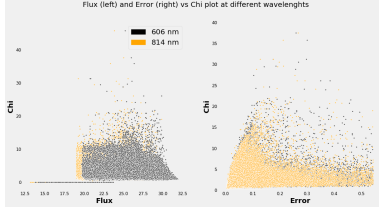


Figure 3.19: **Chi vs Flux and Chi vs Error plots** As the Chi can be approximated with the Second Component and the 606 nm flux with the First Component changed by its sign, the Chi vs Flux plot can be considered as a good approximation of the first two component of the PCA.

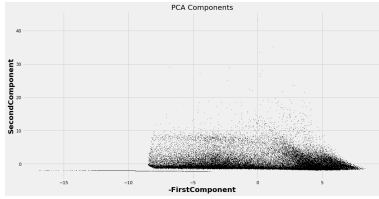


Figure 3.20: **SecondComponent vs -FirstComponent plot** The FirstComponent feature, changed by its sign and rigidly translated, can be approximated with the 606 nm flux, while the Second Component can be approximated with the Chi variable.

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 separate 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.

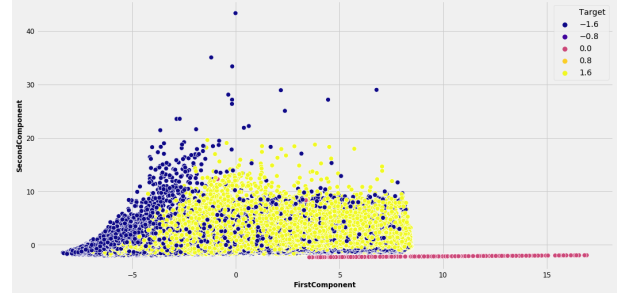


Figure 3.21: **SecondComponent vs -FirstComponent plot** The SecondComponent vs FirstComponent plot shows that almost all the points with null Sharp are located in a narrow zone with the highest FirstComponent values.

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

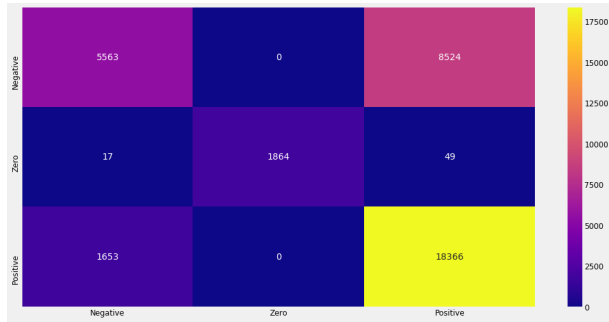


Figure 3.22: **Three classes classification Confusion Matrix** The zero class is almost exactly classified, while the other two classes classification perform almost at the same level of the previous classifier.

	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 $p: \mathcal{X} \rightarrow [0, 1]$. 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, \dots, D_ν . The expected entropy if A_i is used as the

current root is 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:

- **Information 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 Impurity:** 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

- 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.

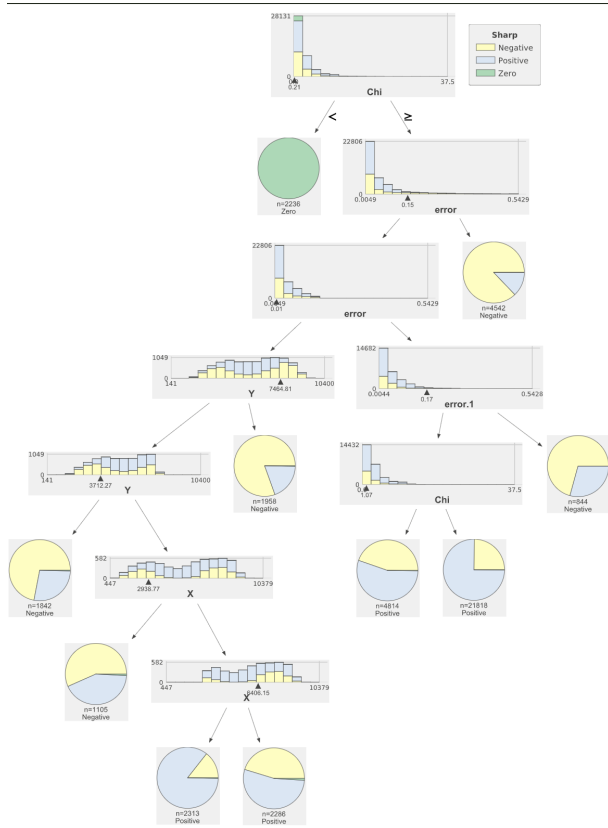


Figure 3.24: Decision Tree visualization. Leaf nodes are pie chart while the decision nodes are displayed as histograms.

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

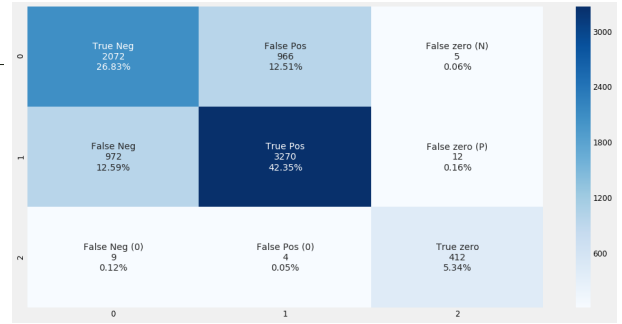


Figure 3.25: Decision Tree Confusion matrix.

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 Ensemble 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

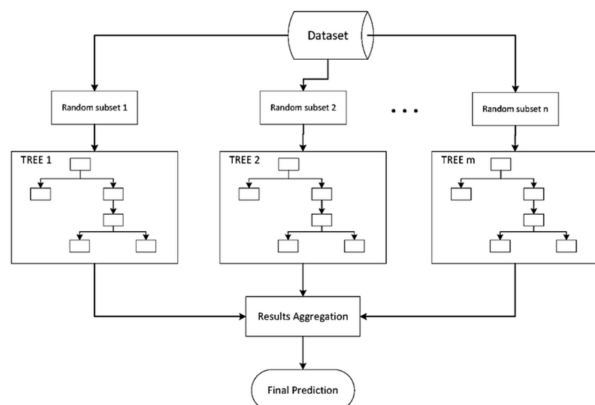


Figure 3.26: Bagging example: A Decision Tree is formed on each bootstrapped sample. The results of each tree are aggregated to yield the strongest, most accurate predictor.

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-

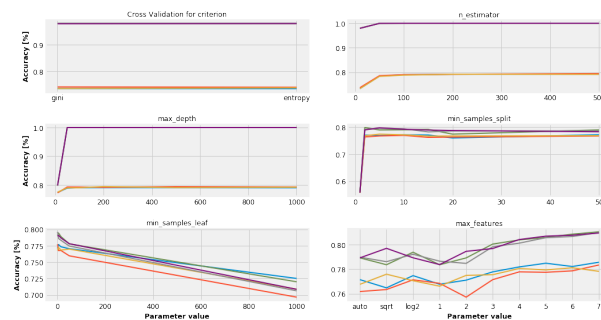


Figure 3.27: Cross validation subplots. Every plot shows 3 cross validation steps for a single feature.

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

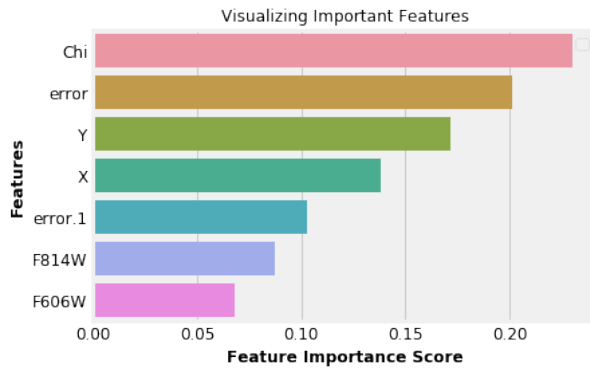


Figure 3.28: Ranking of features according to Random Forest Classifier.

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. Al-

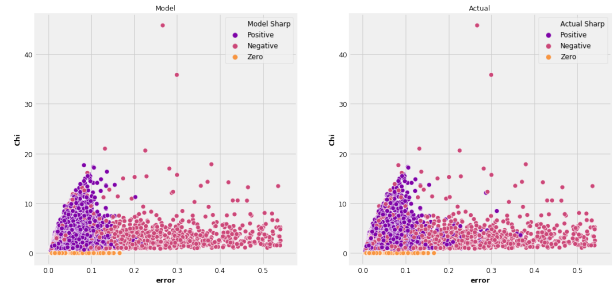


Figure 3.29: Scatterplot of the projection of the test set on the two most relevant directions according to the model. On the left the model prediction, on the right the actual targets.

though 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.

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

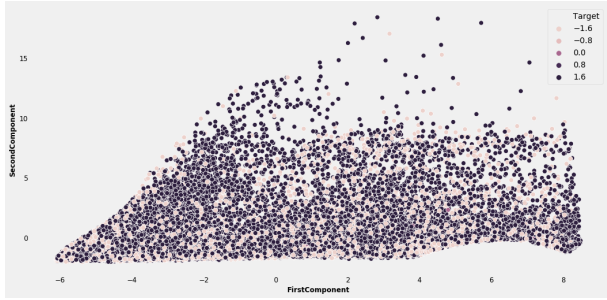


Figure 3.30: Scatter plot of test data wrongly classied by SVM.

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

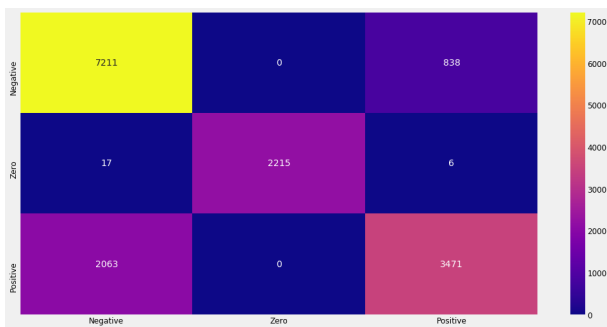


Figure 3.31: Confusion matrix of the Boosted model.

that far beyond the random forest in terms of performance because it is basically a 'tar-

geted' 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 anastro-physics 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 called Random Forest. A careful hyperparameter 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 output 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 curious 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 boosted 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

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # Machine Learning Utility
5
6 # In[1]:
7
8
9 #Importing the libraries to watch
   the 'fits' image and get the
   data array
10 import astropy
11 #import plotly.graph_objects as go
12 from astropy.io import fits
13 #Importing a library that is useful
   to read the original file
14 import pandas as pd
15 import pylab as plb
16 import matplotlib.pyplot as plt
17 from scipy.optimize import
   curve_fit
18 from scipy import asarray as ar,exp
19 #Importing a visual library with
   some illustrative set up
20 import matplotlib.pyplot as plt
21 import matplotlib.colors as mcolors
22 from matplotlib import cm
23 import numpy as np
24 from sklearn.decomposition import
   PCA
```

```
25 import math
26 import seaborn as sns
27 from sklearn.linear_model import
   LogisticRegression
28 plt.style.use('fivethirtyeight')
29 plt.rcParams['font.family'] = 'sans
   -serif'
30 plt.rcParams['font.serif'] = '
   Ubuntu'
31 plt.rcParams['font.monospace'] = '
   Ubuntu Mono'
32 plt.rcParams['font.size'] = 14
33 plt.rcParams['axes.labelsize'] = 12
34 plt.rcParams['axes.labelweight'] =
   'bold'
35 plt.rcParams['axes.titlesize'] = 12
36 plt.rcParams['xtick.labelsize'] =
   12
37 plt.rcParams['ytick.labelsize'] =
   12
38 #plt.rcParams['legend.fontsize'] =
   12
39 plt.rcParams['figure.titlesize'] =
   12
40 plt.rcParams['image.cmap'] = 'jet'
41 plt.rcParams['image.interpolation']
   = 'none'
42 plt.rcParams['figure.figsize'] =
   (16, 8)
43 plt.rcParams['lines.linewidth'] = 2
44 plt.rcParams['lines.markersize'] =
   8
45 plt.rcParams["axes.grid"] = False
46
47
48 # In[2]:
49
50
51 data=pd.read_csv('star.txt',sep='\s
   +')
52
53
54 # In[3]:
55
56
57 data.head()
58
59
60 # In[19]:
```

```

61 plt.rcParams['axes.labelweight'] =
62     'bold'
63 sns.countplot(data.Sharp.round())
64 plt.xlim(5)
65 plt.grid(True)
66 plt.title('Sharp Countplot',
67     fontsize=20)
68 plt.xlabel('Sharp Values',fontsize
69     =20)
70 plt.ylabel('Count',fontsize=20)
71
72 plt.rcParams['axes.labelweight'] =
73     'bold'
74 plt.rcParams['axes.titlesize'] = 12
75 plt.rcParams['xtick.labelsize'] =
76     12
77 plt.rcParams['ytick.labelsize'] =
78     12
79 plt.rcParams['legend.fontsize'] =
80     12
81 plt.rcParams['figure.titlesize'] =
82     12
83 plt.rcParams['image.cmap'] = 'jet'
84 plt.rcParams['image.interpolation']
85     = 'none'
86 plt.rcParams['figure.figsize'] =
87     (16, 8)
88 plt.rcParams['lines.linewidth'] = 2
89 plt.rcParams['lines.markersize'] =
90     8
91 plt.rcParams["axes.grid"] = False

```

5.2 PCA.ipynb

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # # P.C.A.
5
6  # In[31]:
7
8
9  #Importing a library that is useful
10     to read the original file
11 import pandas as pd
12 import matplotlib.pyplot as plt
13 #Importing a visual library with
14     some illustrative set up
15 import matplotlib.pyplot as plt
16 import matplotlib.colors as mcolors
17 from matplotlib import cm
18 import numpy as np
19 from sklearn.utils.testing import
20     ignore_warnings
21 from sklearn.exceptions import
22     ConvergenceWarning
23 from sklearn.decomposition import
24     PCA
25 import math
26 import seaborn as sns
27 plt.style.use('fivethirtyeight')
28 plt.rcParams['font.family'] = 'sans
29     -serif'
30 plt.rcParams['font.serif'] = '
31     Ubuntu'
32 plt.rcParams['font.monospace'] = '
33     Ubuntu Mono'
34 plt.rcParams['font.size'] = 14
35 plt.rcParams['axes.labelsize'] = 12
36
37 plt.rcParams['axes.labelweight'] =
38     'bold'
39 plt.rcParams['axes.titlesize'] = 12
40 plt.rcParams['xtick.labelsize'] =
41     12
42 plt.rcParams['ytick.labelsize'] =
43     12
44 plt.rcParams['legend.fontsize'] =
45     12
46 plt.rcParams['figure.titlesize'] =
47     12
48 plt.rcParams['image.cmap'] = 'jet'
49 plt.rcParams['image.interpolation']
50     = 'none'
51 plt.rcParams['figure.figsize'] =
52     (16, 8)
53 plt.rcParams['lines.linewidth'] = 2
54 plt.rcParams['lines.markersize'] =
55     8
56 plt.rcParams["axes.grid"] = False
57
58 # In[3]:
59
60 #Importing the dataset
61 data=pd.read_csv('star.txt',sep='\s
62     +')
63
64 # In[4]:
65
66 data.head()
67
68 # In[5]:
69
70 #Sharp is the target so it's not
71     included
72 notar=data.drop(columns=['Sharp','#
73     ID'])
74
75 # In[6]:
76
77 #P.C.A. transformation

```

```

66 pca=PCA(n_components=3) 104
67 pca=pca.fit(notar)
68 pca_data=pd.DataFrame(pca.transform 105
    (notar))
69 106
70 107
71 # In[7]: 108
72 109
73 110
74 #Explained variance ratio 111
    computation
75 VAR=pca.explained_variance_ratio_
76 112
77 113
78 # In[8]: 114
79 115
80 116
81 plt.title('Explained Variance Ratio 117
    3 Components',fontsize=20)
82 plt.ylabel('Explained Variance 118
    Ratio',fontsize=20)
83 plt.xlabel('Principal Components', 119
    fontsize=20)
84 plt.plot(VAR,marker='.',color='k') 120
85 plt.xticks(np.arange(0,3,1)) 121
86 plt.grid(True) 122
87 123
88 124
89 # In[9]: 125
90 126
91 127
92 #Displaying the new components. 128
93 pca=PCA(n_components=3) 129
94 pca=pca.fit(data.drop(columns=[' 130
    Sharp','#ID']))
95 pca_data=pd.DataFrame(pca.transform 131
    (data.drop(columns=['Sharp','#
    ID'])))
96 pca_data=pca_data.rename(columns 132
    ={0: 'FirstComponent',1: '
    SecondComponent',2: '
    ThirdComponent'})
97 COL_NAMES=pca_data.columns.tolist() 133
98 k=1 134
99 for i in range(3): 135
100     col=COL_NAMES[i]
101     for j in range(3):
102         if k==1:
103             plt.subplot(3,3,k)
104                 plt.subplots_adjust(
105                     left=0.025, bottom=0.1, right
106                     =0.9, top=1.5, wspace=0.2,
107                     hspace=0.7)
108                 plt.plot(pca_data[col],
109                     pca_data[COL_NAMES[j]],color='k
110                     ')
111                 plt.xlabel(COL_NAMES[i
112                     ])
113                 plt.ylabel(COL_NAMES[j
114                     ])
115                 else:
116                     plt.subplot(3,3,k)
117                     plt.subplots_adjust(
118                         left=0.025, bottom=0.1, right
119                         =0.9, top=1.5, wspace=0.2,
120                         hspace=0.7)
121                     plt.plot(pca_data[col],
122                         pca_data[COL_NAMES[j]],',',
123                         color='k')
124                     plt.xlabel(COL_NAMES[i
125                         ])
126                     plt.ylabel(COL_NAMES[j
127                         ])
128                     k=k+1
129
130 # In[10]:
131
132 #Two variables are now represented
    as X and Y coordinates
133 plt.plot(data.X,data.Y,',',color='
    black')
134 plt.xlabel('X',fontsize=20)
135 plt.ylabel('Y',fontsize=20)
136
137 # In[11]:
138
139 #The last 5 entries of P.C.A.
    component data
140 pca_data.tail()

```

```

136 # In[12]:
137
138
139 #The correlation of the original
    variables
140 notar.corr()
141
142
143 # In[13]:
144
145
146 #The correlation of the P.C.A.
    variables
147 pca_data.corr()
148
149
150 # # P.C.A. Excluding Space
151
152 # In[14]:
153
154
155 #The same P.C.A. has been applied
    excluding the space coordinates
156 notar=data.drop(columns=['Sharp','#
    ID','X','Y'])
157 pca=PCA(n_components=3)
158 pca=pca.fit(notar)
159 pca_data=pd.DataFrame(pca.transform(
    notar))
160 pca_data=pca_data.rename(columns
    ={0: 'FirstComponent',1: '
    SecondComponent',2: '
    ThirdComponent'})
161 COL_NAMES=pca_data.columns.tolist()
162 k=1
163 q=0
164 for i in range(3):
165     col=COL_NAMES[i]
166     for j in range(3):
167         if k==1 or k==5 or k==9:
168             plt.subplot(3,3,k)
169             plt.subplots_adjust(
                left=0.025, bottom=0.1, right
                =0.9, top=1.5, wspace=0.2,
                hspace=0.7)
170
171             sns.kdeplot(pca_data[
                COL_NAMES[i]],color='darkorange
                ')
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
        #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')

```

```

206 plt.plot(np.array(data['F814W']- data['F606W']),data.F814W,',',
207          color='gold')
208 plt.grid(True)
209
210 # In[32]:
211
212 import matplotlib.patches as
213 mpatches
214
215 # In[33]:
216
217 #Plotting First Component and 814
218 flux in black
219 plt.plot(-pca_data.FirstComponent,
220 data.F814W,',',color='k')
221 #Plotting a complex feature linear
222 combination in red
223 plt.plot((data.F606W-26.5)*1.3+5*
224 data.error,data.F814W,',',color
225 ='red')
226 plt.grid(True)
227
228 plt.xlabel('814 nm flux',fontsize
229 =20)
230
231 plt.ylabel('Stellar Color/ Modified
232 Third Component', fontsize=12)
233
234 red_patch = mpatches.Patch(color='
235 red', label='Feature
236 Combination')
237
238 black_patch = mpatches.Patch(color=
239 'black', label='- First
240 Component')
241
242 plt.legend(handles=[red_patch,
243 black_patch])
244
245 # In[34]:
246
247 #Optimization process to get the
248 exact right coefficient of the
249 linear combination
250
251 from sklearn.metrics import
252 mean_squared_error
253
254 A=np.arange(1,3,0.1)
255 B=np.arange(-30.5,-22.5,0.5)
256 C=np.arange(-15,15,1)
257 orig=-pca_data.FirstComponent
258 max_pca=np.abs(-pca_data.
259 FirstComponent.max())
260 RMSE=[]
261 TRIPLET=[]
262 for a in A:
263     for b in B:
264         for c in C:
265             recons=(data.F606W+b)*a
266             +c*data.error
267             RMSE.append(np.sqrt(
268 mean_squared_error(recons,orig)
269 ))
270             TRIPLET.append([a,b,c])
271 a_opt=TRIPLET[np.array(RMSE).argmin
272 ()][0]
273 b_opt=TRIPLET[np.array(RMSE).argmin
274 ()][1]
275 c_opt=TRIPLET[np.array(RMSE).argmin
276 ()][2]
277 r_opt=(data.F606W+b_opt)*a_opt+
278 c_opt*data.error
279 D=np.arange(-10,10,0.1)
280 BEST_RMSE=[]
281 for d in D:
282     recons=r_opt+d*data.F814W
283     BEST_RMSE.append(np.sqrt(
284 mean_squared_error(recons,orig)
285 ))
286 BEST_RMSE=np.array(BEST_RMSE)
287 d_opt=D[BEST_RMSE.argmin()]
288 r_opt=(data.F606W+b_opt)*a_opt+
289 c_opt*data.error+d_opt*data.
290 F814W
291 E=np.arange(-10,10,1)
292 BEST_RMSE=[]
293 for e in E:
294     recons=r_opt+e*data['error.1']
295     BEST_RMSE.append(np.sqrt(
296 mean_squared_error(recons,orig)
297 ))
298 BEST_RMSE=np.array(BEST_RMSE)
299 e_opt=E[np.array(BEST_RMSE).argmin
300 ()]
301 r_opt_first=r_opt+e_opt*data['error

```



```

        .1']
271
272
273 # In[35]:
274
275
276 print ('The lowest RMSE is ' + str(
    BEST_RMSE.min()))
277
278
279 # In[36]:
280
281
282 #Displaying the best values to
    obtain the linear combination
    with less R.M.S.E.
283 print ('The best parameters are \n'
    )
284 print((a_opt,b_opt,c_opt,d_opt,
    e_opt))
285
286
287 # In[37]:
288
289
290 r_opt=-r_opt
291
292
293 # In[40]:
294
295
296 #Plotting the optimum feature
    combination and the first
    component of P.C.A.
297 plt.plot(pca_data.FirstComponent,
    data.F814W,',',color='k',label=
    'First Component')
298 plt.plot(r_opt,data.F814W,',',color=
    'red', label='Linear
    Combination')
299 plt.grid(True)
300 plt.xlabel('814 nm flux',fontsize
    =20)
301 plt.ylabel('First Component/Linear
    Combination',fontsize=12)
302 red_patch = mpatches.Patch(color='
    red', label='Optimum Feature
    Combination')
303 black_patch = mpatches.Patch(color=
    'black', label='First Component
    ')
304 plt.legend(handles=[red_patch,
    black_patch],fontsize=20)
305
306
307 # In[48]:
308
309
310 #Plotting a linear combination of
    Second Component + bias (y) vs
    First component (black)
311 #Plotting Chi vs First component (
    red)
312 plt.plot(np.array(data.Chi),
    pca_data['FirstComponent'],',',
    color='red')
313 plt.plot(pca_data.SecondComponent
    *0.96+2.276,pca_data['
    FirstComponent'],',',color='
    black')
314 plt.grid(True)
315 red_patch = mpatches.Patch(color='
    red', label='First Component')
316 black_patch = mpatches.Patch(color=
    'black', label='Modified Second
    Component')
317 plt.xlabel('First Component',
    fontsize=20)
318 plt.ylabel('Chi/ Modified Second
    Component', fontsize=12)
319 plt.legend(handles=[red_patch,
    black_patch])
320
321
322 # In[41]:
323
324
325 #Optimum parameters for the second
    component
326 r_opt_sec=pca_data.SecondComponent
    *0.96+2.276
327
328
329 # In[42]:
330
331
332 #Computing the RMSE between the
    real second component and the

```

```

linear combination
333 second_RMSE=np.sqrt(
    mean_squared_error(pca_data.
        SecondComponent*0.96+2.276,data
        .Chi))/data.Chi.max()
334
335
336 # In[43]:
337
338
339 print('The RMSE for the
    reconstructed second component
    is ' + str(second_RMSE))
340
341
342 # In[42]:
343
344
345 import matplotlib.patches as
    mpatches
346 #Plotting 814nm flux vs Stellar
    color (red) or a linear
    combination of Third Component
    + bias (black)
347 plt.ylim(30.0,12.5)
348 plt.plot(np.array(data['F814W']-
    data['F606W']),data.F814W,',',
    color='red',label='Stellar
    Color')
349 plt.plot(-1.24+1.31*pca_data.
    ThirdComponent,data.F814W,',',
    color='k',label='Modified Third
    Component')
350 plt.grid(True)
351 red_patch = mpatches.Patch(color='
    red', label='Stellar Color')
352 black_patch = mpatches.Patch(color='
    black', label='Modified Third
    Component')
353 plt.xlabel('814 nm flux',fontsize
    =20)
354 plt.ylabel('Stellar Color/ Modified
    Third Component', fontsize=12)
355 plt.legend(handles=[red_patch,
    black_patch])
356
357
358 # In[44]:
359
360
361 r_opt_third=-1.24+1.31*pca_data.
    ThirdComponent
362
363
364 # In[45]:
365
366
367 third_RMSE=np.sqrt(
    mean_squared_error(-1.24+1.31*
    pca_data.ThirdComponent,np.
    array(data.F814W-data.F606W)))
368
369
370 # In[46]:
371
372
373 print ('The RMSE between the Third
    Component and its
    reconstruction is '+ str(
    third_RMSE))
374
375
376 # In[57]:
377
378
379 #Stacking together the coordinates
    and the sharp
380 pca_data['X']=data.X
381 pca_data['Y']=data.Y
382
383
384 # In[58]:
385
386
387 pca_data['Sharp']=data.Sharp
388
389
390 # In[59]:
391
392
393 pca_data.head()
394
395
396 # In[60]:
397
398
399 #pca_data=pca_data.drop(columns=['
    Sharp'])

```

5.3 Linear.ipynb

```
1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # 4 Dataset Classification
5
6 # In[2]:
7
8
9 #Importing the libraries to watch
   the 'fits' image and get the
   data array
10 import astropy
11 #import plotly.graph_objects as go
12 from astropy.io import fits
13 #Importing a library that is useful
   to read the original file
14 import pandas as pd
15 import pylab as plb
16 import matplotlib.pyplot as plt
17 from scipy.optimize import
   curve_fit
18 from scipy import asarray as ar,exp
19 #Importing a visual library with
   some illustrative set up
20 import matplotlib.pyplot as plt
21 import matplotlib.colors as mcolors
22 from matplotlib import cm
23 import numpy as np
24 from sklearn.utils.testing import
   ignore_warnings
25 from sklearn.exceptions import
   ConvergenceWarning
26 from sklearn.decomposition import
   PCA
27 import math
28 import seaborn as sns
29 from sklearn.linear_model import
   LogisticRegression
30 plt.style.use('fivethirtyeight')
31 plt.rcParams['font.family'] = 'sans
   -serif'
32 plt.rcParams['font.serif'] = '
   Ubuntu'
33 plt.rcParams['font.monospace'] = '
   Ubuntu Mono'
34 plt.rcParams['font.size'] = 14
35 plt.rcParams['axes.labelsize'] = 12
36 plt.rcParams['axes.labelweight'] =
   'bold'
37 plt.rcParams['axes.titlesize'] = 12
38 plt.rcParams['xtick.labelsize'] =
   12
39 plt.rcParams['ytick.labelsize'] =
   12
40 #plt.rcParams['legend.fontsize'] =
   12
41 plt.rcParams['figure.titlesize'] =
   12
42 plt.rcParams['image.cmap'] = 'jet'
43 plt.rcParams['image.interpolation']
   = 'none'
44 plt.rcParams['figure.figsize'] =
   (16, 8)
45 plt.rcParams['lines.linewidth'] = 2
46 plt.rcParams['lines.markersize'] =
   8
47 plt.rcParams["axes.grid"] = False
48
49
50 # In[3]:
51
52
53 #Importing the dataset
54 data=pd.read_csv('star.txt',sep='\s
   +')
55
56
57 # In[4]:
58
59
60 #Displaying the first 5 rows
61 data.head()
62
63
64 # In[5]:
65
66
67 #Dataset without Sharp (target) and
   #ID
68 notar=data.drop(columns=['Sharp','#
   ID'])
69
70
71 # # P.C.A. Excluding Space
72
73
74 # In[7]:
```

```

74
75
76 #Excluding Space and performing P.C
    .A.
77 notar=data.drop(columns=['X','Y'])
78
79
80 # In[8]:
81
82
83 notar.head()
84
85
86 # In[9]:
87
88
89 pca=PCA(n_components=3)
90 pca=pca.fit(notar)
91 pca_data=pd.DataFrame(pca.transform
    (notar))
92
93
94 # In[10]:
95
96
97 pca_data=pca_data.rename(columns
    ={0:'FirstComponent',1:'
    SecondComponent',2:'
    ThirdComponent'})
98
99
100 # In[11]:
101
102
103 #Stacking the X and Y coordinates
    to the P.C.A.
104 pca_data['X']=data.X
105 pca_data['Y']=data.Y
106
107
108 # In[12]:
109
110
111 #And the Sharp
112 pca_data['Sharp']=data.Sharp
113
114
115 # In[13]:
116
117
118 #Displaying the first rows
119 pca_data.head()
120
121
122 # In[60]:
123
124
125 #pca_data=pca_data.drop(columns=['
    Sharp'])
126
127
128 # # 2-class: Positive or Negative
129
130 # In[16]:
131
132
133 #2 class classification
    preprocessing
134 #The sharp is considered with its
    sign
135 #The 0 sharp are now considered as
    positive
136
137 data['SharpSign']=data.Sharp.apply(
    np.sign)
138 data['SharpSign']=data['SharpSign'
    ].replace(0,1)
139
140
141 # In[24]:
142
143
144 #From Sklearn import library for
    mutual information
145 from sklearn.feature_selection
    import mutual_info_classif as
    mi
146
147
148 # In[25]:
149
150
151 #Take columns and compute the
    Mutual Information
152 COL=data.columns.tolist()[:-2]
153
154
155 # In[34]:

```

```

156
157
158 MI=mi(data.drop(columns=['Sharp','#
159 ID']),np.array(data.SharpSign).
160 reshape(-1,1))
161
162 # In[37]:
163
164 MI_data=pd.DataFrame({'
165 MutualInformation':MI})
166
167 # In[44]:
168
169
170 MI_data.index=data.drop(columns=['
171 Sharp','#ID']).columns.tolist()
172
173 # In[52]:
174
175
176 #Excluding the trivial correlation
177 of the Sharp Sign with itself
178 #the three most informative feature
179 are the two errors and the
180 flux feature
181 MI_data.sort_values(by='
182 MutualInformation',ascending=
183 False).head(4)
184
185 # In[55]:
186
187
188 #Computing the same thing with the
189 P.C.A. dataset
190 COL=pca_data.columns.tolist()[:-1]
191 MI=mi(pca_data,np.array(data.
192 SharpSign).reshape(-1,1))
193
194 MI_data.index=COL+['Sharp']
195
196 # In[62]:
197
198
199 #Excluding the trivial correlation
200 of the Sharp Sign with itself
201 #the three most informative feature
202 are the three components
203 MI_data.sort_values(by='
204 MutualInformation',ascending=
205 False).head(4)
206
207
208 # # 1. PCA Linear Classifier
209
210 # In[10]:
211
212 pca_data['Target']=data['SharpSign']
213
214
215 # In[303]:
216
217 pca_data.head()
218
219 # In[11]:
220
221
222 #Importing linear SVM classifier
223 and model selection train test
224 split
225 from sklearn.svm import LinearSVC
226 as SVC
227 from sklearn.model_selection import
228 train_test_split
229
230 # In[325]:
231
232
233 #The dataset
234 X=pca_data.drop(columns=['Sharp','
235 Target'])

```

```

232
233 # In[326]:
234
235
236 #The target
237 y=pca_data.Target
238
239
240 # In[327]:
241
242
243 #Test-(validation+train) split
244 X_train, X_test, y_train, y_test = train_test_split(
245     X, y, test_size=0.2,
246     random_state=42)
247
248 # In[328]:
249
250
251 #Validation-Train split
252 X_train_val, X_train_val, y_train_val, y_train_val = train_test_split(
253     X_train, y_train, test_size
254     =0.2, random_state=42)
255
256 # In[ ]:
257
258
259 #Computing the best coefficient on
260 the validation set, C range=0-1
261 #Training on the training set
262 C_SCORE=[]
263 c=np.arange(0.2,1.2,0.2)
264 for c_value in c:
265     clf=SVC(C=c_value)
266     clf.fit(X_train_val,y_train_val)
267     sc=clf.score(X_train_val,
268     y_train_val)
269     C_SCORE.append(sc)
270     print(str(c_value) + '
271     coefficient has been adopted' )
272
273 # In[346]:
274
275
276 #Printing the best C in terms of
277 accuracy on the validation set
278 between 0 and 1
279 MIN_VAL=np.array(C_SCORE).max()
280 MIN_C=c[np.array(C_SCORE).argmax()]
281 print ('In the range ' + str(c.min
282     ())+ ' and ' +str(c.max())+ '\n
283     ')
284 print ('the best score has been
285     obtained with ' + str(MIN_C))
286 print ('and it is ' + str(MIN_VAL))
287
288 # In[347]:
289
290
291 #Computing the best coefficient on
292 the validation set, C range
293 =1-10
294 #Training on the training set
295 C_SCORE=[]
296 c=np.arange(1,11,1)
297 for c_value in c:
298     clf=SVC(C=c_value)
299     clf.fit(X_train_val,y_train_val)
300     sc=clf.score(X_train_val,
301     y_train_val)
302     C_SCORE.append(sc)
303     print(str(c_value) + '
304     coefficient has been adopted' )
305
306 # In[349]:
307
308
309 #Printing the best C in terms of
310 accuracy on the validation set
311 between 1 and 10
312 MIN_VAL=np.array(C_SCORE).max()
313 MIN_C=c[np.array(C_SCORE).argmax()]
314 print ('In the range ' + str(c.min
315     ())+ ' and ' +str(c.max())+ '\n
316     ')
317 print ('the best score has been
318     obtained with ' + str(MIN_C))
319 print ('and it is ' + str(MIN_VAL))
320
321
322

```

```

309 # In[343]:
310
311
312 #Computing the best coefficient on
    the validation set, C range=10
    and 100
313 #Training on the training set
314
315 C_SCORE=[]
316 c=np.arange(10,110,10)
317 for c_value in c:
318     clf=SVC(C=c_value)
319     clf.fit(X_train,y_train)
320     sc=clf.score(X_train_val,
321                 y_train_val)
322     C_SCORE.append(sc)
323     print(str(c_value) + '
324           coefficient has been adopted' )
325
326 # In[344]:
327
328 #Printing the best C in terms of
    accuracy on the validation set
    between 10 and 100
329
330 MIN_C=c[np.array(C_SCORE).argmax()]
331 print ('In the range ' + str(c.min
332       ())+ ' and ' +str(c.max())+ '\n
333       ')
334
335 print ('the best score has been
336       obtained with ' + str(MIN_C))
337
338 print ('and it is ' + str(MIN_VAL))
339
340 # In[386]:
341
342
343 #Taking the best three
    possibilities for the three
    ranges
344 OPT_C=[0.6,7,30]
345
346 # In[388]:
347
348 #Testing them on the training set
    and see which is the best
349 FIN_SCORE=[]
350 for opt_C in OPT_C:
351     clf=SVC(C=opt_c)
352     clf.fit(X_train,y_train)
353     fin_score=clf.score(X_test,
354                         y_test)
355     FIN_SCORE.append(fin_score)
356
357 # In[391]:
358
359 FIN_SCORE=np.array(FIN_SCORE)
360
361 # In[392]:
362
363
364 fin_score=FIN_SCORE.max()
365
366 # In[394]:
367
368
369 print ('The PCA dataset gave a best
370       classification with ' + str(
371       fin_score*100)+ '% of accuracy
372       with a linear classifier')
373
374 # # 1.2 Dataset Linear Classifier
375
376 # In[413]:
377
378 #Same process as before, the only
    difference is that we used the
    entire dataset
379 #And not the P.C.A.
380 X=data.drop(columns=['SharpSign'])
381
382
383 # In[414]:
384
385
386 X=pca_data.drop(columns=['Sharp',
387                          'Target'])

```

```

388
389 # In[415]:
390
391
392 X_train, X_test, y_train, y_test =
393     train_test_split(
394         X, y, test_size=0.2,
395         random_state=42)
396
397
398
399 X_train, X_train_val, y_train,
400     y_train_val = train_test_split(
401         X_train, y_train, test_size
402         =0.2, random_state=42)
403
404
405 # In[417]:
406
407 C_SCORE=[]
408 c=np.arange(0.2,1.2,0.2)
409 for c_value in c:
410     clf=SVC(C=c_value)
411     clf.fit(X_train,y_train)
412     sc=clf.score(X_train_val,
413                 y_train_val)
414     C_SCORE.append(sc)
415     print(str(c_value) + '
416     coefficient has been adopted' )
417
418 # In[418]:
419
420 MIN_VAL=np.array(C_SCORE).max()
421 MIN_C=c[np.array(C_SCORE).argmax()]
422 print ('In the range ' + str(c.min
423     ())+ ' and ' +str(c.max())+ '\n
424     ')
425 print ('the best score has been
426     obtained with ' + str(MIN_C))
427 print ('and it is ' + str(MIN_VAL))
428
429 # In[419]:
430
431 C_SCORE=[]
432 c=np.arange(1,11,1)
433 for c_value in c:
434     clf=SVC(C=c_value)
435     clf.fit(X_train,y_train)
436     sc=clf.score(X_train_val,
437                 y_train_val)
438     C_SCORE.append(sc)
439     print(str(c_value) + '
440     coefficient has been adopted' )
441
442 # In[420]:
443
444 MIN_VAL=np.array(C_SCORE).max()
445 MIN_C=c[np.array(C_SCORE).argmax()]
446 print ('In the range ' + str(c.min
447     ())+ ' and ' +str(c.max())+ '\n
448     ')
449 print ('the best score has been
450     obtained with ' + str(MIN_C))
451 print ('and it is ' + str(MIN_VAL))
452
453 # In[421]:
454
455 C_SCORE=[]
456 c=np.arange(10,110,10)
457 for c_value in c:
458     clf=SVC(C=c_value)
459     clf.fit(X_train,y_train)
460     sc=clf.score(X_train_val,
461                 y_train_val)
462     C_SCORE.append(sc)
463     print(str(c_value) + '
464     coefficient has been adopted' )
465
466 # In[422]:
467
468 MIN_VAL=np.array(C_SCORE).max()
469 MIN_C=c[np.array(C_SCORE).argmax()]
470 print ('In the range ' + str(c.min
471     ())+ ' and ' +str(c.max())+ '\n
472     ')

```



```

468 print ('the best score has been
      obtained with ' + str(MIN_C))
469 print ('and it is ' + str(MIN_VAL))
470
471
472 # In[423]:
473
474
475 OPT_C=[1,70]
476
477
478 # In[424]:
479
480
481 FIN_SCORE=[]
482 for opt_C in OPT_C:
483     clf=SVC(C=opt_c)
484     clf.fit(X_train,y_train)
485     fin_score=clf.score(X_test,
486                          y_test)
486     FIN_SCORE.append(fin_score)
487
488
489 # In[425]:
490
491
492 FIN_SCORE=np.array(FIN_SCORE)
493
494
495 # In[426]:
496
497
498 fin_score=FIN_SCORE.max()
499
500
501 # In[427]:
502
503
504 print ('The PCA dataset gave a best
      classification with ' + str(
      fin_score*100)+ '% of accuracy
      with a linear classifier')
505
506
507 # # 1.3 Best Features
      Classification Original Data
508
509 # In[49]:
510
511
512 #Same process as before, but only
      the most informative feature
      has been used (error,606flux
      and error)
513 opt_data=data.drop(columns=['#ID',
      'X','Y','Chi','Sharp','F814W'])
514
515
516 # In[50]:
517
518
519 opt_data.head()
520
521
522 # In[436]:
523
524
525 X=opt_data.drop(columns=['SharpSign
      '])
526
527
528 # In[437]:
529
530
531 data['SharpSign']=data.Sharp.apply(
      np.sign)
532 data['SharpSign']=data['SharpSign'
      ].replace(0,1)
533 y=data.SharpSign
534
535
536 # In[438]:
537
538
539 X_train, X_test, y_train, y_test =
      train_test_split(
      X, y, test_size=0.2,
      random_state=42)
540
541
542 # In[439]:
543
544
545
546 X_train,X_train_val,y_train,
      y_train_val= train_test_split(
      X_train, y_train, test_size
      =0.2, random_state=42)
547
548

```

```

549
550 # In[440]:
551
552
553 C_SCORE=[]
554 c=np.arange(0.2,1.2,0.2)
555 for c_value in c:
556     clf=SVC(C=c_value)
557     clf.fit(X_train,y_train)
558     sc=clf.score(X_train_val,
559                 y_train_val)
559     C_SCORE.append(sc)
560     print(str(c_value) + '
561           coefficient has been adopted' )
562
563 # In[441]:
564
565
566 MIN_VAL=np.array(C_SCORE).max()
567 MIN_C=c[np.array(C_SCORE).argmax()]
568 print ('In the range ' + str(c.min
569      ())+ ' and ' +str(c.max())+ '\n
570      ')
569 print ('the best score has been
570      obtained with ' + str(MIN_C))
571 print ('and it is ' + str(MIN_VAL))
572
573 # In[442]:
574
575
576 C_SCORE=[]
577 c=np.arange(1,11,1)
578 for c_value in c:
579     clf=SVC(C=c_value)
580     clf.fit(X_train,y_train)
581     sc=clf.score(X_train_val,
582                 y_train_val)
582     C_SCORE.append(sc)
583     print(str(c_value) + '
584           coefficient has been adopted' )
585
586 # In[443]:
587
588
589 MIN_VAL=np.array(C_SCORE).max()
590 MIN_C=c[np.array(C_SCORE).argmax()]
591
592 print ('In the range ' + str(c.min
593      ())+ ' and ' +str(c.max())+ '\n
594      ')
595 print ('the best score has been
596      obtained with ' + str(MIN_C))
597 print ('and it is ' + str(MIN_VAL))
598
599 # In[444]:
600
601
602 C_SCORE=[]
603 c=np.arange(10,110,10)
604 for c_value in c:
605     clf=SVC(C=c_value)
606     clf.fit(X_train,y_train)
607     sc=clf.score(X_train_val,
608                 y_train_val)
609     C_SCORE.append(sc)
610     print(str(c_value) + '
611           coefficient has been adopted' )
612
613 # In[445]:
614
615
616 MIN_VAL=np.array(C_SCORE).max()
617 MIN_C=c[np.array(C_SCORE).argmax()]
618 print ('In the range ' + str(c.min
619      ())+ ' and ' +str(c.max())+ '\n
620      ')
621 print ('the best score has been
622      obtained with ' + str(MIN_C))
623 print ('and it is ' + str(MIN_VAL))
624
625 # In[447]:
626
627
628 OPT_C=[0.2,1,40]
629
630 # In[448]:
631
632
633 FIN_SCORE=[]
634 for opt_C in OPT_C:
635     clf=SVC(C=opt_c)
636     clf.fit(X_train,y_train)

```

```

632     fin_score=clf.score(X_test,
633                          y_test)
634     FIN_SCORE.append(fin_score)
635
636 # In[449]:
637
638
639 FIN_SCORE=np.array(FIN_SCORE)
640
641
642 # In[450]:
643
644
645 fin_score=FIN_SCORE.max()
646
647
648 # In[451]:
649
650
651 print ('The PCA dataset gave a best
        classification with ' + str(
        fin_score*100)+ '% of accuracy
        with a linear classifier')
652
653
654 # # 1.4 Best Feature Classification
        PCA data
655
656 # In[452]:
657
658
659 #Same process as before, but only
        the most informative P.C.A.
        components have been used
660 opt_data=pca_data[['FirstComponent',
        'SecondComponent',
        'ThirdComponent']]
661
662
663 # In[453]:
664
665
666 X_train, X_test, y_train, y_test =
        train_test_split(
667     X, y, test_size=0.2,
        random_state=42)
668
669
670 # In[454]:
671
672
673 X_train,X_train_val,y_train,
        y_train_val= train_test_split(
674     X_train, y_train, test_size
        =0.2, random_state=42)
675
676
677 # In[455]:
678
679
680 C_SCORE=[]
681 c=np.arange(0.2,1.2,0.2)
682 for c_value in c:
683     clf=SVC(C=c_value)
684     clf.fit(X_train,y_train)
685     sc=clf.score(X_train_val,
        y_train_val)
686     C_SCORE.append(sc)
687     print(str(c_value) + '
        coefficient has been adopted' )
688
689
690 # In[456]:
691
692
693 MIN_VAL=np.array(C_SCORE).max()
694 MIN_C=c[np.array(C_SCORE).argmax()]
695 print ('In the range ' + str(c.min
        ())+ ' and ' +str(c.max())+ '\n
        ')
696 print ('the best score has been
        obtained with ' + str(MIN_C))
697 print ('and it is ' + str(MIN_VAL))
698
699
700 # In[457]:
701
702
703 C_SCORE=[]
704 c=np.arange(1,11,1)
705 for c_value in c:
706     clf=SVC(C=c_value)
707     clf.fit(X_train,y_train)
708     sc=clf.score(X_train_val,
        y_train_val)
709     C_SCORE.append(sc)
710     print(str(c_value) + '

```

```

711         coefficient has been adopted' )
712
713 # In[458]:
714
715 MIN_VAL=np.array(C_SCORE).max()
716 MIN_C=c[np.array(C_SCORE).argmax()]
717 print ('In the range ' + str(c.min()
718       )+ ' and ' +str(c.max())+ '\n
719       ')
720 print ('the best score has been
721       obtained with ' + str(MIN_C))
722 print ('and it is ' + str(MIN_VAL))
723
724 # In[459]:
725
726 C_SCORE=[]
727 c=np.arange(10,110,10)
728 for c_value in c:
729     clf=SVC(C=c_value)
730     clf.fit(X_train,y_train)
731     sc=clf.score(X_train_val,
732                y_train_val)
733     C_SCORE.append(sc)
734     print(str(c_value) + '
735           coefficient has been adopted' )
736
737 # In[460]:
738
739 MIN_VAL=np.array(C_SCORE).max()
740 MIN_C=c[np.array(C_SCORE).argmax()]
741 print ('In the range ' + str(c.min()
742       )+ ' and ' +str(c.max())+ '\n
743       ')
744 print ('the best score has been
745       obtained with ' + str(MIN_C))
746 print ('and it is ' + str(MIN_VAL))
747
748 # In[461]:
749
750 OPT_C=[0.2,1,70]
751
752 # In[462]:
753
754 FIN_SCORE=[]
755 for opt_C in OPT_C:
756     clf=SVC(C=opt_c)
757     clf.fit(X_train,y_train)
758     fin_score=clf.score(X_test,
759                        y_test)
760     FIN_SCORE.append(fin_score)
761
762 # In[463]:
763
764 FIN_SCORE=np.array(FIN_SCORE)
765
766 # In[464]:
767
768 fin_score=FIN_SCORE.max()
769
770 # In[465]:
771
772 print ('The PCA dataset gave a best
773       classification with ' + str(
774       fin_score*100)+ '% of accuracy
775       with a linear classifier')

```

5.4 NonLinearTwoComponent.ipynb

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # 4 Dataset Classification
5
6 # In[4]:
7
8
9 #Importing the libraries to watch
10 the 'fits' image and get the
11 data array
12
13 import astropy
14 #import plotly.graph_objects as go

```

```

12 from astropy.io import fits
13 #Importing a library that is useful
    to read the original file
14 import pandas as pd
15 import pylab as plb
16 import matplotlib.pyplot as plt
17 from scipy.optimize import
    curve_fit
18 from scipy import asarray as ar,exp
19 #Importing a visual library with
    some illustrative set up
20 import matplotlib.pyplot as plt
21 import matplotlib.colors as mcolors
22 from matplotlib import cm
23 import numpy as np
24 from sklearn.utils.testing import
    ignore_warnings
25 from sklearn.exceptions import
    ConvergenceWarning
26 from sklearn.decomposition import
    PCA
27 import math
28 import seaborn as sns
29 from sklearn.linear_model import
    LogisticRegression
30 plt.style.use('fivethirtyeight')
31 plt.rcParams['font.family'] = 'sans
    -serif'
32 plt.rcParams['font.serif'] = '
    Ubuntu'
33 plt.rcParams['font.monospace'] = '
    Ubuntu Mono'
34 plt.rcParams['font.size'] = 14
35 plt.rcParams['axes.labelsize'] = 12
36 plt.rcParams['axes.labelweight'] =
    'bold'
37 plt.rcParams['axes.titlesize'] = 12
38 plt.rcParams['xtick.labelsize'] =
    12
39 plt.rcParams['ytick.labelsize'] =
    12
40 #plt.rcParams['legend.fontsize'] =
    12
41 plt.rcParams['figure.titlesize'] =
    12
42 plt.rcParams['image.cmap'] = 'jet'
43 plt.rcParams['image.interpolation']
    = 'none'
44 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

```

```

    features
89 pca=PCA(n_components=3)
90 pca=pca.fit(notar)
91 pca_data=pd.DataFrame(pca.transform(notar))
92
93
94 # In[11]:
95
96
97 #Renaming them
98 pca_data=pca_data.rename(columns={0: 'FirstComponent', 1: '
    SecondComponent', 2: '
    ThirdComponent'})
99
100
101 # In[12]:
102
103
104 #Stacking together the spatial
    coordinates
105 pca_data['X']=data.X
106 pca_data['Y']=data.Y
107
108
109 # In[13]:
110
111
112 #And the target
113 pca_data['Sharp']=data.Sharp
114
115
116 # In[14]:
117
118
119 pca_data.head()
120
121
122 # # 2. Best method non linear
123
124 # In[15]:
125
126
127 #SVM applied to the first two
    component
128 opt_data=pca_data[['FirstComponent',
    'SecondComponent']]
129

```

```

130
131 # In[31]:
132
133
134 #Two target classification
135 #Taking the Sharp and considering
    only its sign
136 data['SharpSign']=np.sign(data.
    Sharp)
137
138
139 # In[33]:
140
141
142 #As it is two class, the 0 values
    are set together with the
    positive ones.
143 data[data['SharpSign']==0].
    SharpSign=np.ones(len(data[data
    ['SharpSign']==0]))
144
145
146 # In[85]:
147
148
149 #Target set as dharp sign
150 opt_data['Target']=data.SharpSign
151
152
153 # In[74]:
154
155
156 #Dataset without target
157 X=opt_data.drop(columns=['Target'])
158
159
160 # In[75]:
161
162
163 #Scatterplot of First and Second
    Component, together with the
    target
164 sns.scatterplot(opt_data.
    FirstComponent, opt_data.
    SecondComponent, hue=opt_data.
    Target, palette='plasma')
165 plt.grid(True)
166
167

```

```

168 # In[78]:
169
170
171 #Importing sklearn libraries for
172 SVC and model selection
173 from sklearn.svm import SVC
174 from sklearn.model_selection import
175 train_test_split
176
177
178
179 #Considering the target y
180 y=opt_data.Target
181
182
183 # In[88]:
184
185
186 #Considering the train test split
187 X_train, X_test, y_train, y_test =
188 train_test_split(
189 X, y, test_size=0.2,
190 random_state=42)
191
192
193 # In[485]:
194
195 #Checking the best C value for SVM
196 between 0 and 1
197 c_list=np.arange(0.10,1.1,0.10)
198 k=0
199 FIN_SCORE=[]
200 for c in c_list:
201     clf=SVC(C=c, kernel='rbf')
202     clf.fit(X_train,y_train)
203     fin_score=clf.score(X_test,
204 y_test)
205     FIN_SCORE.append(fin_score)
206
207     if k==1:
208         print('20% of the C values
209 inspected')
210     if k==4:
211         print('50% of the C values
212 inspected')
213     if k==7:
214         print('80% of the C values
215 inspected')
216     if k == 9:
217         print('100% of the C
218 values inspected \n')
219         print('Process completed')
220         k=k+1
221
222
223 # In[487]:
224
225
226 #Computing the accuracy
227 FIN_SCORE=np.array(FIN_SCORE)
228
229
230 # In[488]:
231
232
233 #Checking the best C(best accuracy)
234 fin_score=FIN_SCORE.max()
235 i=FIN_SCORE.argmax()
236 c_max=c_list[i]
237
238
239 # In[500]:
240
241
242 print('The best classification is
243 done between ' + str(c_list.min
244 ()) + ' and ' + str(c_list.max
245 ()) + '\n')
246 print('is ' + str(c_max) + ',
247 obtaining the following
248 accuracy: ' + str(fin_score*100)+
249 '%')
250
251 # In[501]:
252
253
254 #Same process, but changing C
255 between 1 and 10
256 c_list=np.arange(1.,11,1.)
257 k=0
258 FIN_SCORE=[]
259 for c in c_list:
260     clf=SVC(C=c, kernel='rbf')
261     clf.fit(X_train,y_train)

```

249	fin_score=clf.score(X_test,	286	y_test)
	y_test)	287	FIN_SCORE.append(fin_score)
250	FIN_SCORE.append(fin_score)	288	if k==0:
251			print('20% of the C values
252	if k==1:	289	inspected')
253	print('20% of the C values	290	if k==1:
	inspected')		print('40% of the C values
254	if k==4:	291	inspected')
255	print('50% of the C values	292	if k==2:
	inspected')		print('60% of the C values
256	if k==7:	293	inspected')
257	print('80% of the C values	294	if k==3:
	inspected')		print('80% of the C values
258	if k == 9:	295	inspected')
259	print('100% of the C	296	if k==4:
	values inspected \n')		print('100% of the C values
260	print('Process completed')	297	inspected')
261	k=k+1	298	k=k+1
262		299	
263		300	# In[508]:
264	# In[504]:	301	
265		302	
266		303	FIN_SCORE=np.array(FIN_SCORE)
267	FIN_SCORE=np.array(FIN_SCORE)	304	fin_score=FIN_SCORE.max()
268	fin_score=FIN_SCORE.max()	305	i=FIN_SCORE.argmax()
269	i=FIN_SCORE.argmax()	306	c_max=c_list[i]
270	c_max=c_list[i]	307	print('The best classification is
271	print('The best classification is	308	done between ' + str(c_list.min
	done between ' + str(c_list.min		()) + ' and ' + str(c_list.max
	() + ' and ' + str(c_list.max		()) + '\n')
	() + '\n')		
272	print('is ' + str(c_max) +',	309	print('is ' + str(c_max) +',
	obtaining the following	310	obtaining the following
	accuracy: '+str(fin_score*100)+	311	accuracy: '+str(fin_score*100)+
	%,')	312	%,')
273		313	
274		314	#Same process but changing C
275	# In[506]:	315	between 10 and 50
276		316	c_list=np.arange(100,600,100)
277		317	k=0
278	#Same process but changing C	318	FIN_SCORE=[]
	between 10 and 50	319	for c in c_list:
279	c_list=np.arange(10,60,10)	320	clf=SVC(C=c, kernel='rbf')
280	k=0	321	clf.fit(X_train,y_train)
281	FIN_SCORE=[]		fin_score=clf.score(X_test,
282	for c in c_list:		y_test)
283	clf=SVC(C=c, kernel='rbf')		
284	clf.fit(X_train,y_train)		
285	fin_score=clf.score(X_test,		


```

322     FIN_SCORE.append(fin_score)
323     if k==0:
324         print('20% of the C values inspected \n')
325     if k==1:
326         print('40% of the C values inspected \n')
327     if k==2:
328         print('60% of the C values inspected \n')
329     if k==3:
330         print('80% of the C values inspected \n')
331     if k==4:
332         print('100% of the C values inspected \n')
333         print ('Process completed')
334     k=k+1
335
336
337 # In[512]:
338
339
340 FIN_SCORE=np.array(FIN_SCORE)
341 fin_score=FIN_SCORE.max()
342 i=FIN_SCORE.argmax()
343 c_max=c_list[i]
344 print ('The best classification is done between ' + str(c_list.min()) + ' and ' + str(c_list.max()) + '\n')
345 print('is ' + str(c_max) + ', obtaining the following accuracy: '+str(fin_score*100)+'%')
346
347
348 # In[513]:
349
350
351 #Same process with C=1000
352 clf=SVC(C=1000, kernel='rbf')
353 clf.fit(X_train,y_train)
354 fin_score=clf.score(X_test,y_test)
355
356
357 # In[516]:
358
359
360 print('The score for C=1000 is ' + str(fin_score*100) + '%')
361
362
363 # In[517]:
364
365 #And C=10000
366 clf=SVC(C=10000, kernel='rbf')
367 clf.fit(X_train,y_train)
368 fin_score=clf.score(X_test,y_test)
369
370
371 # In[518]:
372
373 print('The score for C=10000 is ' + str(fin_score*100) + '%')
374
375
376
377
378 # In[90]:
379
380
381 #And C=1000000
382 clf=SVC(C=100000, kernel='rbf')
383 clf.fit(X_train,y_train)
384 fin_score=clf.score(X_test,y_test)
385
386
387 # In[91]:
388
389 #The best one is the last one, with 72%+ of accuracy
390 print('The score for C=100000 is ' + str(fin_score*100) + '% \n')
391 print ('SVM best score: 72.2%')
392
393
394
395 # In[92]:
396
397 #Prediction computed on the test set
398 pred=clf.predict(X_test)
399
400
401
402 # In[93]:
403

```

```

404                                     fontsize=20)
405 #Target and Prediction comparison displayed
406 pred_data=pd.DataFrame()
407 pred_data['FirstComponent']= X_test['FirstComponent']
408 pred_data['SecondComponent']=X_test['SecondComponent']
409 pred_data['Target']=y_test
410 pred_data['Prediction']=pred
411
412
413 # In[94]:
414
415
416 pred_data.head()
417
418
419 # In[95]:
420
421
422 pred_data.to_csv('SVMprediction.csv')
423
424
425 # In[541]:
426
427
428 #Subplot of the prediction and the real target
429 plt.subplot(2,1,1)
430 sns.scatterplot(pred_data.
    FirstComponent,pred_data.
    SecondComponent,hue=pred_data.
    Target,palette='plasma')
431 plt.grid(True)
432 plt.xlabel('First Component',
    fontsize=20)
433 plt.ylabel('Second Component',
    fontsize=20)
434 plt.subplot(2,1,2)
435 sns.scatterplot(pred_data.
    FirstComponent,pred_data.
    SecondComponent,hue=pred_data.
    Prediction,palette='plasma')
436 plt.grid(True)
437 plt.xlabel('First Component',
    fontsize=20)
438 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()

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

```

5.5 NonLinearThreeComponentT

```

4 # # 4 Dataset Classification
5
6 # In[1]:
7
8
9 #Importing the libraries to watch
   the 'fits' image and get the
   data array
10 import astropy
11 #import plotly.graph_objects as go
12 from astropy.io import fits
13 #Importing a library that is useful
   to read the original file
14 import pandas as pd
15 import pylab as plb
16 import matplotlib.pyplot as plt
17 from scipy.optimize import
   curve_fit
18 from scipy import asarray as ar,exp
19 #Importing a visual library with
   some illustrative set up
20 import matplotlib.pyplot as plt
21 import matplotlib.colors as mcolors
22 from matplotlib import cm
23 import numpy as np
24 from sklearn.utils.testing import
   ignore_warnings
25 from sklearn.exceptions import
   ConvergenceWarning
26 from sklearn.decomposition import
   PCA
27 import math
28 import seaborn as sns
29 from sklearn.linear_model import
   LogisticRegression
30 plt.style.use('fivethirtyeight')
31 plt.rcParams['font.family'] = 'sans
   -serif'
32 plt.rcParams['font.serif'] = '
   Ubuntu'
33 plt.rcParams['font.monospace'] = '
   Ubuntu Mono'
34 plt.rcParams['font.size'] = 14
35 plt.rcParams['axes.labelsize'] = 12
36 plt.rcParams['axes.labelweight'] =
   'bold'
37 plt.rcParams['axes.titlesize'] = 12
38 plt.rcParams['xtick.labelsize'] =
   12
39 plt.rcParams['ytick.labelsize'] =
   12
40 #plt.rcParams['legend.fontsize'] =
   12
41 plt.rcParams['figure.titlesize'] =
   12
42 plt.rcParams['image.cmap'] = 'jet'
43 plt.rcParams['image.interpolation']
   = 'none'
44 plt.rcParams['figure.figsize'] =
   (16, 8)
45 plt.rcParams['lines.linewidth'] = 2
46 plt.rcParams['lines.markersize'] =
   8
47 plt.rcParams["axes.grid"] = False
48
49 # In[2]:
50
51 #Importing the dataset
52 data=pd.read_csv('star.txt',sep='\s
   +')
53
54 # In[3]:
55
56 data.head()
57
58 # In[4]:
59
60 #Excluding Sharp and #ID
61 notar=data.drop(columns=['Sharp','#
   ID'])
62
63 # # P.C.A. Excluding Space
64
65 # In[5]:
66
67 #Excluding Space Features in order
   to perform the P.C.A.
68 notar=data.drop(columns=['X','Y'])
69
70
71
72
73
74
75
76
77
78

```

```

79 # In[6]:
80
81
82 notar.head()
83
84
85 # In[7]:
86
87
88 pca=PCA(n_components=3)
89 pca=pca.fit(notar)
90 pca_data=pd.DataFrame(pca.transform
    (notar))
91
92
93 # In[8]:
94
95
96 pca_data=pca_data.rename(columns
    ={0: 'FirstComponent', 1: '
    SecondComponent', 2: '
    ThirdComponent'})
97
98
99 # In[9]:
100
101
102 #Stacking together the spatial
    coordinates
103 pca_data['X']=data.X
104 pca_data['Y']=data.Y
105
106
107 # In[10]:
108
109
110 #And the sharp
111 pca_data['Sharp']=data.Sharp
112
113
114 # In[11]:
115
116
117 pca_data.head()
118
119
120 # In[13]:
121
122
123 #pca_data=pca_data.drop(columns=['
    Sharp'])
124
125
126 # # Best method 3 Features
127
128 # In[12]:
129
130
131 #Two classification pre process,
    the Sharp is considered by its
    sign
132 data['SharpSign']=np.sign(data.
    Sharp)
133
134
135 # In[13]:
136
137
138 #The 0 values are considered as
    positive
139 data[data['SharpSign']==0]['
    SharpSign']=np.ones(len(data[
    data['SharpSign']==0]))
140
141
142 # In[14]:
143
144
145 #Three features are considered, the
    most informative
146 opt_data=pca_data[['FirstComponent'
    , 'SecondComponent', '
    ThirdComponent']]
147 opt_data['Target']=data.SharpSign
148
149
150 # In[15]:
151
152
153 #Target
154 y=opt_data.Target
155
156
157 # In[16]:
158
159
160 #dataset
161 X=opt_data.drop(columns=['Target'])

```

```

162
163
164 # In[17]:
165
166
167 #Importing SVM and train test split
168     from sklearn.model_selection
169     from sklearn.svm import SVC
170     from sklearn.model_selection import
171         train_test_split
172
173 # In[18]:
174
175 #(Train+validation)/Test split with
176     wide test set (90%)
177 X_train, X_test, y_train, y_test =
178     train_test_split(
179         X, y, test_size=0.9,
180         random_state=42)
181
182 # In[19]:
183
184 #Train/Validation split at half
185 X_train, X_val, y_train, y_val =
186     train_test_split(
187         X_train, y_train, test_size
188         =0.5, random_state=42)
189
190 # In[20]:
191
192 #List of kernels
193 K_LIST=['linear', 'poly', 'rbf', '
194     sigmoid']
195
196 # In[ ]:
197
198 #Validation with CV_number=5 has
199     been performed to choose the
200     best kernel
201
202 BEST_KERNEL=[]
203 k=0
204
205 for i in range(5):
206     FIN_SCORE=[]
207     X_train, X_test, y_train,
208     y_test = train_test_split(
209         X, y, test_size=0.9,
210         random_state=42)
211     X_train, X_val, y_train, y_val
212     = train_test_split(
213         X_train, y_train, test_size
214         =0.5, random_state=42)
215     print('Split Done')
216     for ker in K_LIST:
217         clf=SVC(kernel=ker)
218         clf.fit(X_train,y_train)
219         fin_score=clf.score(X_val,
220             y_val)
221         FIN_SCORE.append(fin_score)
222         k=k+1
223     print(ker + ' Kernel has
224         been explored')
225     FIN_SCORE=np.array(FIN_SCORE)
226
227     BEST_KERNEL.append(K_LIST[
228         FIN_SCORE.argmax()])
229     print('Cross validation ' + str
230         (i) + ' out of 4 \n')
231
232 # In[25]:
233
234 sns.countplot(np.array(BEST_KERNEL)
235 )
236 plt.xlabel('Chosen Kernel',fontsize
237     =20)
238
239 # In[39]:
240
241 #Best kernel is chosen 5 times out
242     of 5
243 best_kernel='rbf'
244
245 # In[97]:
246
247 #Cross validation on C values

```

```

239 c_list=np.arange(0.5,50.5,0.5) 277
240 k=0 278
241 PERC=['20%', '40%', '60%', '80%', '100%'] # In[102]: 279
242 280
243 K=[20,40,60,80,100] 281
244 BEST_C=[] 282 #39 is chosen 5 times out of 5
245 for i in range(5): 283 best_c=c_list[FIN_SCORE.argmax()]
246 FIN_SCORE=[] 284
247 X_train, X_test, y_train, 285
248 y_test = train_test_split( 286 # In[107]:
249 X, y, test_size=0.9, 287
250 random_state=42) 288
251 X_train, X_val, y_train, y_val 289 #Train test split 70/30
252 = train_test_split( 290 X_train, X_test, y_train, y_test =
253 X_train, y_train, test_size 291 train_test_split(
254 =0.5, random_state=42) 292 X, y, test_size=0.7,
255 for c in c_list: 293 random_state=42)
256 k=k+1 294
257 clf=SVC(C=c,kernel= 295 # In[108]:
258 best_kernel) 296
259 clf.fit(X_train,y_train) 297
260 fin_score=clf.score(X_val, 298 #Prediction with best parameters
261 y_val) 299 clf=SVC(kernel=best_kernel,C=best_c
262 FIN_SCORE.append(fin_score) 300 )
263 #k=k+1 301 clf.fit(X_train,y_train)
264 if k in K: 302 fin_score=clf.score(X_test,y_test)
265 ind=K.index(k) 303
266 print (PERC[ind] + ' of 304 # In[119]:
267 the C values has been explored 305
268 ') 306
269 FIN_SCORE=np.array(FIN_SCORE) 307
270 BEST_C.append(c_list[FIN_SCORE 308
271 argmax()]) 309
272 print('Cross validation ' + str 310 prediction=clf.predict(X_test.drop(
273 (i) + ' out of 4 \n') 311 columns=['Target']))
274 312
275 # In[100]: 313
276 314
277 315
278 316
279 317
280 318 # In[121]:
281 319
282 320
283 321
284 322
285 323
286 324
287 325
288 326
289 327
290 328
291 329
292 330
293 331
294 332
295 333
296 334
297 335
298 336
299 337
300 338
301 339
302 340
303 341
304 342
305 343
306 344
307 345
308 346
309 347
310 348
311 349
312 350
313 351
314 352
315 353
316 354
317 355
318 356
319 357
320 358
321 359
322 360
323 361
324 362
325 363
326 364
327 365
328 366
329 367
330 368
331 369
332 370
333 371
334 372
335 373
336 374
337 375
338 376
339 377
340 378
341 379
342 380
343 381
344 382
345 383
346 384
347 385
348 386
349 387
350 388
351 389
352 390
353 391
354 392
355 393
356 394
357 395
358 396
359 397
360 398
361 399
362 400
363 401
364 402
365 403
366 404
367 405
368 406
369 407
370 408
371 409
372 410
373 411
374 412
375 413
376 414
377 415
378 416
379 417
380 418
381 419
382 420
383 421
384 422
385 423
386 424
387 425
388 426
389 427
390 428
391 429
392 430
393 431
394 432
395 433
396 434
397 435
398 436
399 437
400 438
401 439
402 440
403 441
404 442
405 443
406 444
407 445
408 446
409 447
410 448
411 449
412 450
413 451
414 452
415 453
416 454
417 455
418 456
419 457
420 458
421 459
422 460
423 461
424 462
425 463
426 464
427 465
428 466
429 467
430 468
431 469
432 470
433 471
434 472
435 473
436 474
437 475
438 476
439 477
440 478
441 479
442 480
443 481
444 482
445 483
446 484
447 485
448 486
449 487
450 488
451 489
452 490
453 491
454 492
455 493
456 494
457 495
458 496
459 497
460 498
461 499
462 500
463 501
464 502
465 503
466 504
467 505
468 506
469 507
470 508
471 509
472 510
473 511
474 512
475 513
476 514
477 515
478 516
479 517
480 518
481 519
482 520
483 521
484 522
485 523
486 524
487 525
488 526
489 527
490 528
491 529
492 530
493 531
494 532
495 533
496 534
497 535
498 536
499 537
500 538
501 539
502 540
503 541
504 542
505 543
506 544
507 545
508 546
509 547
510 548
511 549
512 550
513 551
514 552
515 553
516 554
517 555
518 556
519 557
520 558
521 559
522 560
523 561
524 562
525 563
526 564
527 565
528 566
529 567
530 568
531 569
532 570
533 571
534 572
535 573
536 574
537 575
538 576
539 577
540 578
541 579
542 580
543 581
544 582
545 583
546 584
547 585
548 586
549 587
550 588
551 589
552 590
553 591
554 592
555 593
556 594
557 595
558 596
559 597
560 598
561 599
562 600
563 601
564 602
565 603
566 604
567 605
568 606
569 607
570 608
571 609
572 610
573 611
574 612
575 613
576 614
577 615
578 616
579 617
580 618
581 619
582 620
583 621
584 622
585 623
586 624
587 625
588 626
589 627
590 628
591 629
592 630
593 631
594 632
595 633
596 634
597 635
598 636
599 637
600 638
601 639
602 640
603 641
604 642
605 643
606 644
607 645
608 646
609 647
610 648
611 649
612 650
613 651
614 652
615 653
616 654
617 655
618 656
619 657
620 658
621 659
622 660
623 661
624 662
625 663
626 664
627 665
628 666
629 667
630 668
631 669
632 670
633 671
634 672
635 673
636 674
637 675
638 676
639 677
640 678
641 679
642 680
643 681
644 682
645 683
646 684
647 685
648 686
649 687
650 688
651 689
652 690
653 691
654 692
655 693
656 694
657 695
658 696
659 697
660 698
661 699
662 700
663 701
664 702
665 703
666 704
667 705
668 706
669 707
670 708
671 709
672 710
673 711
674 712
675 713
676 714
677 715
678 716
679 717
680 718
681 719
682 720
683 721
684 722
685 723
686 724
687 725
688 726
689 727
690 728
691 729
692 730
693 731
694 732
695 733
696 734
697 735
698 736
6
```

```

320 test_data['Target']=y_test
321 test_data['Prediction']=prediction
322
323
324 # In[169]:
325
326
327 #Confusion matrix
328 import itertools
329 from string import ascii_uppercase
330 from sklearn.metrics import
    confusion_matrix
331
332 y_test=test_data.Target
333 predic = prediction
334
335 columns = ['Negative','Non Negative
    ']
336
337 confm = confusion_matrix(y_test,
    predic)
338 df_cm = pd.DataFrame(confm.astype(
    float), index=columns, columns=
    columns)
339
340 ax = sns.heatmap(df_cm, cmap='
    plasma',annot=True,fmt='g')
341
342
343 # In[170]:
344
345
346 #Defining precision and recall out
    of the confusion matrix
347 def precision(confusion):
348     TP=confusion[0][0]
349     TN=confusion[1][1]
350     FP=confusion[0][1]
351     FN=confusion[1][0]
352     pres_a=TP/(TP+FN)
353     pres_b=TN/(TN+FP)
354     return [pres_a,pres_b]
355
356
357 # In[172]:
358
359
360 def recall(confusion):
361     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]
362
363
364
365
366
367
368
369
370 # In[176]:
371
372
373 #Display the statistics
374 def statistics(confusion):
375     stat=pd.DataFrame({'Negative':[
    precision(confusion)[0],recall(
    confusion)[0]],'Non Negative':[
    precision(confusion)[1],recall(
    confusion)[1]]})
    stat.index=['Precision','Recall
    ']
    return stat
376
377
378
379
380 # In[177]:
381
382
383 statistics(confm)

```

5.6 NonLinearThreeComponentT

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # 4 Dataset Classification
5
6 # In[3]:
7
8
9 #Importing the libraries to watch
    the 'fits' image and get the
    data array
10 import astropy
11 #import plotly.graph_objects as go
12 from astropy.io import fits
13 #Importing a library that is useful
    to read the original file
14 import pandas as pd
15 import pylab as plb

```

```

16 import matplotlib.pyplot as plt
17 from scipy.optimize import
    curve_fit
18 from scipy import asarray as ar,exp
19 #Importing a visual library with
    some illustrative set up
20 import matplotlib.pyplot as plt
21 import matplotlib.colors as mcolors
22 from matplotlib import cm
23 import numpy as np
24 from sklearn.utils.testing import
    ignore_warnings
25 from sklearn.exceptions import
    ConvergenceWarning
26 from sklearn.decomposition import
    PCA
27 import math
28 import seaborn as sns
29 from sklearn.linear_model import
    LogisticRegression
30 plt.style.use('fivethirtyeight')
31 plt.rcParams['font.family'] = 'sans
    -serif'
32 plt.rcParams['font.serif'] = '
    Ubuntu'
33 plt.rcParams['font.monospace'] = '
    Ubuntu Mono'
34 plt.rcParams['font.size'] = 14
35 plt.rcParams['axes.labelsize'] = 12
36 plt.rcParams['axes.labelweight'] =
    'bold'
37 plt.rcParams['axes.titlesize'] = 12
38 plt.rcParams['xtick.labelsize'] =
    12
39 plt.rcParams['ytick.labelsize'] =
    12
40 #plt.rcParams['legend.fontsize'] =
    12
41 plt.rcParams['figure.titlesize'] =
    12
42 plt.rcParams['image.cmap'] = 'jet'
43 plt.rcParams['image.interpolation']
    = 'none'
44 plt.rcParams['figure.figsize'] =
    (16, 8)
45 plt.rcParams['lines.linewidth'] = 2
46 plt.rcParams['lines.markersize'] =
    8
47 plt.rcParams["axes.grid"] = False
48
49
50 # In[4]:
51
52
53 #Importing the dataset
54 data=pd.read_csv('star.txt',sep='\s
    +')
55
56
57 # In[6]:
58
59
60 data.head()
61
62
63 # In[7]:
64
65
66 #Excluding the target and the #ID
67 notar=data.drop(columns=['Sharp','#
    ID'])
68
69
70 # # P.C.A. Excluding Space
71
72 # In[6]:
73
74
75 #Excluding the target, the #ID and
    the spatial coordinates in
    order to apply the P.C.A.
76 notar=data.drop(columns=['Sharp','#
    ID','X','Y'])
77
78
79 # In[9]:
80
81
82 notar.head()
83
84
85 # In[7]:
86
87
88 pca=PCA(n_components=3)
89 pca=pca.fit(notar)
90 pca_data=pd.DataFrame(pca.transform
    (notar))

```



```

91     ThirdComponent'])
92
93 # In[8]:
94
95
96 pca_data=pca_data.rename(columns
135     ={0: 'FirstComponent',1: '
136     SecondComponent',2: '
137     ThirdComponent'})
97
98
99 # In[57]:
100
101
102 #Stacking the X and Y coordinates
103 pca_data['X']=data.X
104 pca_data['Y']=data.Y
105
106
107 # In[58]:
108
109
110 #And the Sharp
111 pca_data['Sharp']=data.Sharp
112
113
114 # In[59]:
115
116
117 pca_data.head()
118
119
120 # In[60]:
121
122
123 #pca_data=pca_data.drop(columns=['
160     Sharp'])
124
125
126 # # Best method 3 feature 3 classes
164
127
128 # In[12]:
165
129
130
131 #Considering three feature from the
166     P.C.A. and three classes
167     classification
168
132 opt_data=pca_data[['FirstComponent',
169     , 'SecondComponent', '
170     ThirdComponent']]
133 data['SharpSign']=data.Sharp.apply(
134     np.sign)
135
136
137 # In[183]:
138
139
140 #Target and data
141 X=opt_data.drop(columns=['Target'])
142 y=opt_data.Target
143
144
145 # In[184]:
146
147
148 #Importing SVM and Model selection
149     for train test split
150 from sklearn.svm import SVC
151 from sklearn.model_selection import
152     train_test_split
153
154
155 # In[185]:
156
157
158 #(Train+Validation)-Test with wide
159     test set (90% of the data)
160 X_train, X_test, y_train, y_test =
161     train_test_split(
162         X, y, test_size=0.9,
163         random_state=42)
164
165 # In[186]:
166
167
168 #Train Validation split with 50% of
169     values
170 X_train, X_val, y_train, y_val =
171     train_test_split(
172         X_train, y_train, test_size
173         =0.5, random_state=42)
174
175 # In[187]:

```

171		Kernel': BEST_KERNEL})
172	#Computing the best kernel on the validation, training on the training set	206
		207
173	K_LIST=['linear', 'poly', 'rbf', 'sigmoid']	208 # In[191]:
		209
174		210
175		211 sns.countplot(CV_DATA['Chooosen Kernel'])
176	# In[188]:	212
177		213
178		214 # In[192]:
179	#Training and test validation varies 5 times in order to pick the best one	215
		216
180	BEST_KERNEL=[]	217 best_kernel='rbf'
181	k=0	218
182	for i in range(5):	219
		220 # In[194]:
183	FIN_SCORE=[]	221
184	X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9, random_state=42)	222
		223 #The same form of validation is made for the C value
185		224 c_list=np.arange(0.5,50.5,0.5)
186	X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.5, random_state=42)	225 k=0
		226 PERC=['20%', '40%', '60%', '80%', '100%']
187		227 K=[20,40,60,80,100]
188	k=0	228 BEST_C=[]
189	for ker in K_LIST:	229 for i in range(5):
		230 FIN_SCORE=[]
190	clf=SVC(kernel=ker)	231 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9, random_state=42)
191	clf.fit(X_train,y_train)	232
192	fin_score=clf.score(X_val, y_val)	233 X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.5, random_state=42)
		234
193	FIN_SCORE.append(fin_score)	235 k=0
194	k=k+1	236 for c in c_list:
195	print(ker + ' Kernel has been explored')	237 k=k+1
196	FIN_SCORE=np.array(FIN_SCORE)	238 clf=SVC(C=c,kernel=best_kernel)
		239
197		240
198	BEST_KERNEL.append(K_LIST[FIN_SCORE.argmax()])	241
199	print('Cross validation ' + str(i) + ' out of 4 \n')	242
200		243
201		244
202	# In[190]:	
203		
204		
205	CV_DATA=pd.DataFrame({'CV Number': np.arange(1,6,1), 'Chooosen	

```

245         print (PERC[ind] + ' of ' + str(perc))
246         the C values has been explored
247     ')
248     FIN_SCORE=np.array(FIN_SCORE)
249     BEST_C.append(c_list[FIN_SCORE.argmax()])
250     print('Cross validation ' + str(perc) + ' out of 4 \n')
251 # In[195]:
252
253 sns.countplot(BEST_C)
254 plt.xlabel('Chosen C')
255 plt.ylabel('Count')
256
257 # In[196]:
258
259 FIN_SCORE=np.array(FIN_SCORE)
260
261 # In[197]:
262
263 best_c=c_list[FIN_SCORE.argmax()]
264
265 # In[14]:
266
267 best_c=17
268 best_kernel='rbf'
269
270 # In[199]:
271
272 #Train-test split at 70% has been
273 made, and the performance are
274 computed
275 X_train, X_test, y_train, y_test = train_test_split(
276     X, y, test_size=0.7,
277     random_state=42)
278
279 # In[200]:
280
281 #Computing the score
282 clf=SVC(kernel=best_kernel,C=best_c)
283
284 clf.fit(X_train,y_train)
285 fin_score=clf.score(X_test,y_test)
286
287 # In[203]:
288
289 #The prediction
290 prediction=clf.predict(X_test)
291
292 # In[204]:
293
294 print('The final score with 3
295 feature is ' + str(fin_score
296 *100) + '% ')
297
298 # In[205]:
299
300 test_data=X_test.copy()
301 test_data['Target']=y_test
302 test_data['Prediction']=prediction
303
304 # In[214]:
305
306 #And the confusion matrix
307 import itertools
308 from string import ascii_uppercase
309 from sklearn.metrics import
310     confusion_matrix
311
312 y_test=test_data.Target
313 predic = prediction
314
315 columns = ['Negative','Zero','
316 Positive']
317
318 confm = confusion_matrix(y_test,

```

```

    predic)
330 df_cm = pd.DataFrame(confm.astype(
    float), index=columns, columns=
    columns)
331
332 ax = sns.heatmap(df_cm, cmap='
    plasma', annot=True, fmt='g')
333
334
335 # In[227]:
336
337 #Defining precision and recall on
338 three classes classification
339 def precision(confusion, clas):
340     if clas=='Negative':
341         TP=confusion[0][0]
342         FN=confusion[1][0]+
343         confusion[2][0]
344         pres=TP/(TP+FN)
345     if clas=='Positive':
346         TP=confusion[2][2]
347         FN=confusion[2][0]+
348         confusion[2][1]
349         pres=TP/(TP+FN)
350     if clas=='Zero':
351         TP=confusion[1][1]
352         FN=confusion[1][0]+
353         confusion[1][2]
354         pres=TP/(TP+FN)
355     return pres
356
357 # In[235]:
358 def recal(confusion, clas):
359     if clas=='Negative':
360         TP=confusion[0][0]
361         FP=confusion[0][1]+
362         confusion[0][2]
363         rec=TP/(TP+FP)
364     if clas=='Positive':
365         TP=confusion[2][2]
366         FP=confusion[0][2]+
367         confusion[1][2]
368         rec=TP/(TP+FP)
369     if clas=='Zero':
370         TP=confusion[1][1]
371         FP=confusion[0][1]+
372         confusion[2][1]
373         rec=TP/(TP+FP)
374     return rec
375
376 # In[236]:
377 precision(confm, 'Negative'),
378 precision(confm, 'Positive'),
379 precision(confm, 'Zero')
380
381 # In[237]:
382 recal(confm, 'Negative'), recal(confm
383 , 'Positive'), recal(confm, 'Zero'
384 )
385
386 # In[238]:
387
388 #Summary of the Precision and
389 Recall for each classes
390 function
391 def statistics(confusion):
392     neg=[precision(confm, 'Negative'
393 ), recal(confm, 'Negative')]
394     pos=[precision(confm, 'Positive'
395 ), recal(confm, 'Positive')]
396     zero=[precision(confm, 'Zero'),
397 recal(confm, 'Zero')]
398     stats=pd.DataFrame({'Negative':
399 neg, 'Positive':pos, 'Zero':zero
400 })
401     stats.index=['Precision', '
402 Recall']
403     return stats
404
405 # In[239]:
406 statistics(confm)

```

5.7 Decision Tree.ipynb

```

1 #!/usr/bin/env python
2 # coding: utf-8
3
4 # # Decision Tree
5
6 # In[68]:
7
8
9 #Importing a library that is useful
   to read the original file
10 import pandas as pd
11 #Importing a visual library with
   some illustrative set up
12 import matplotlib.pyplot as plt
13 import numpy as np
14 import seaborn as sns
15 import matplotlib.colors as mcolors
16 from matplotlib import cm
17 import math
18 plt.style.use('fivethirtyeight')
19 plt.rcParams['font.family'] = 'sans
   -serif'
20 plt.rcParams['font.serif'] = '
   Ubuntu'
21 plt.rcParams['font.monospace'] = '
   Ubuntu Mono'
22 plt.rcParams['font.size'] = 14
23 plt.rcParams['axes.labelsize'] = 12
24 plt.rcParams['axes.labelweight'] =
   'bold'
25 plt.rcParams['axes.titlesize'] = 12
26 plt.rcParams['xtick.labelsize'] =
   12
27 plt.rcParams['ytick.labelsize'] =
   12
28 plt.rcParams['legend.fontsize'] =
   12
29 plt.rcParams['figure.titlesize'] =
   12
30 plt.rcParams['image.cmap'] = 'jet'
31 plt.rcParams['image.interpolation']
   = 'none'
32 plt.rcParams['figure.figsize'] =
   (16, 8)
33 plt.rcParams['lines.linewidth'] = 2
34 plt.rcParams['lines.markersize'] =
   8
35 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',
36 'xkcd:scarlet']
37 cmap_big = cm.get_cmap('Spectral',
   512)
38 cmap = mcolors.ListedColormap(
   cmap_big(np.linspace(0.7, 0.95,
   256)))
39 bbox_props = dict(boxstyle="round,
   pad=0.3", fc=colors[0], alpha
   =.5)
40
41
42 # In[2]:
43
44
45 from sklearn.datasets import
   load_iris
46 from sklearn.tree import
   DecisionTreeClassifier
47 from sklearn.model_selection import
   train_test_split
48 from sklearn.metrics import
   confusion_matrix
49 from sklearn.tree import
   export_graphviz
50 from sklearn.externals.six import
   StringIO
51 from IPython.display import Image
52 from pydot import
   graph_from_dot_data
53 import pandas as pd
54 import numpy as np
55
56
57 # Load dataset
58
59 # In[3]:
60
61
62 data_original=pd.read_csv('/Users/
   Simone/Desktop/Programmi/DDA/
   hlsp_deep47tuc_hst_acs_47tuc_f606w
   -f814w_v1_catalog.txt',
   delimiter='\s+')

```

```

63 data_original.head()
64
65
66 # Drop the #ID column
67
68 # In[4]:
69
70
71 Sharp = data_original.Sharp
72 data = data_original.drop(columns =
    ['#ID', 'Sharp'])
73
74
75 # In[5]:
76
77
78 data.head()
79
80
81 # create target column, 3 classes :
    positive, negative and zero
    Sharp
82
83 # In[6]:
84
85
86 data['target'] = np.zeros_like(len(data))
87
88
89 # In[7]:
90
91
92 len(data.target[Sharp > 0])
93
94
95 # In[8]:
96
97
98 data.target[Sharp > 0] = 1
99
100
101 # In[9]:
102
103
104 data.target[Sharp == 0] = 2
105
106
107 # Let's split dataset in training
    and test set, we use a
    categorical encoding, this way
    we can add more classes if
    needed
108
109 # In[10]:
110
111
112 feature_names = ['X', 'Y', 'F606W', '
    error', 'F814W', 'error.1', 'Chi']
113
114
115 # In[11]:
116
117
118 X = pd.DataFrame(data, columns=
    feature_names)
119 y = pd.Categorical.from_codes(data.
    target, ['Negative', 'Positive',
    'Zero'])
120
121
122 # In[12]:
123
124
125 X.head()
126
127
128 # we use one-hot encoding where
    sharps value targets are
    vectors :
129 # Negative -> [100]
130 # Positive -> [010]
131 # Zero -> [001]
132 #
133
134 # In[13]:
135
136
137 y = pd.get_dummies(y)
138 y.head()
139
140
141 # Splitto dataset in train and test
142
143 # In[15]:
144
145
146 X_train, X_test, y_train, y_test =

```

```

147     train_test_split(X, y,
148                       train_size = 0.85, random_state
149                       =1)
150
151 # First of all we check how a
152 # standard decision tree performs
153 # on our dataset
154
155 # In[16]:
156
157 dt = DecisionTreeClassifier()
158 dt.fit(X_train, y_train)
159
160 # we can print the output with the
161 # following lines, however
162 # without limitations on leaf
163 # number we expect to have a veri
164 # huge tree.
165
166 # In[ ]:
167
168 dot_data = StringIO()
169 export_graphviz(dt, out_file=
170                 dot_data, feature_names=
171                 feature_names)
172 (graph, ) = graph_from_dot_data(
173                 dot_data.getvalue())
174 Image(graph.create_png())
175
176 # Lets'check confusion matrix and
177 # accuracy of a standard decision
178 # tree
179
180 # In[19]:
181
182 y_pred = dt.predict(X_test)
183
184 # Lets visualize Confusion matrix
185
186 # In[65]:
187
188 species = np.array(y_test).argmax(
189     axis=1)
190 predictions = np.array(y_pred).
191     argmax(axis=1)
192 cf_matrix = confusion_matrix(
193     species, predictions)
194
195 group_names = ['True Neg', 'False
196     Pos', 'False zero (N)', 'False
197     Neg',
198                 'True Pos', 'False
199     zero (P)', 'False Neg (0)', '
200     False Pos (0)', 'True zero']
201 group_counts = ['{0:0.0f}'.format(
202     value) for value in
203                 cf_matrix.flatten()]
204 group_percentages = ['{0:.2%}'.
205     format(value) for value in
206                 cf_matrix.
207     flatten()/np.sum(cf_matrix)]
208 labels = [f'{v1}\n{v2}\n{v3}' for
209     v1, v2, v3 in
210             zip(group_names,
211                 group_counts, group_percentages)
212 ]
213 labels = np.asarray(labels).reshape
214     (3,3)
215 sns.heatmap(cf_matrix, annot=labels
216             , fmt='', cmap='Blues')
217
218 # In[45]:
219
220 from sklearn.metrics import
221     accuracy_score
222
223 accuracy_score(y_test, y_pred)
224
225 # Decision tree works already better
226 # then SVM Already, maybe a
227 # little bit of parameter tuning
228 # would help.
229
230 # However its interesting checking
231 # if a boosting of this method
232 # could lead to better results,
233 # lets consider a random forest

```

<pre> 208 classifier 209 # # Random Forest 210 211 # In[46]: 212 213 214 #Import Random Forest Model 215 from sklearn.ensemble import RandomForestClassifier 216 217 #Create a Gaussian Classifier 218 rf=RandomForestClassifier() 219 220 # Train the model on training data 221 rf.fit(X_train, y_train); 222 223 224 # In[50]: 225 226 227 # Use the forest's predict method on the test data 228 rf_y_pred = rf.predict(X_test) 229 230 231 # In[51]: 232 233 234 accuracy_score(y_test, rf_y_pred) 235 236 237 # it seems to work slightly better, trough RandomForestClassifier we could check a sort of feature ranking 238 239 # In[52]: 240 241 242 feature_imp = pd.Series(rf. feature_importances_, index= feature_names).sort_values(ascending=False) 243 244 245 # In[66]: 246 247 </pre>	<pre> 248 import seaborn as sns 249 get_ipython().run_line_magic(' matplotlib', 'inline') 250 # Creating a bar plot 251 sns.barplot(x=feature_imp, y= feature_imp.index) 252 # Add labels to your graph 253 plt.xlabel('Feature Importance Score') 254 plt.ylabel('Features') 255 plt.title("Visualizing Important Features") 256 plt.legend() 257 plt.show() 258 259 260 # 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. 261 262 # In[57]: 263 264 265 from sklearn.model_selection import validation_curve 266 267 268 # we are going to check most of the features wich are fixed in the standard RandomForestClassifier function throught a validation process 269 270 # In[59]: 271 272 273 num_estNum = [10, 50, 100, 200,500] 274 train_scoreNum, test_scoreNum = validation_curve(275 276 RandomForestClassifier(), 277 X = 278 X_train, y = y_train, 279 280 param_name = 'n_estimators', </pre>
---	---


```

278 param_range = num_estNum, cv = 3)
279
280 # Greater is the number of
281 # estimators, better seems to be
282 # the performance in test set
283 # In[73]:
284
285 plt.plot(num_estNum, test_scoreNum,
286          label = 'Test')
287 plt.plot(num_estNum, train_scoreNum,
288          label = 'Train')
289 plt.grid(True)
290 plt.title('Cross Validation for
291           n_estimator')
292 plt.ylabel('Accuracy [%]')
293 plt.xlabel('Parameter value')
294 plt.legend()
295 plt.show()
296
297 # we do this for differesnt
298 # features, saving best results
299 # In[74]:
300
301 num_estDepth = [10, 50, 100,
302                 200,500,1000]
303 train_scoreDepth, test_scoreDepth=
304 validation_curve(
305     RandomForestClassifier(
306         n_estimators=200),
307     X_train, y = y_train,
308     param_name = 'max_depth',
309     param_range = num_estDepth, cv
310     = 3)
311
312 plt.plot(num_estDepth,
313          test_scoreDepth, label = 'Test')
314 plt.plot(num_estDepth,
315          train_scoreDepth, label = 'Train
316          ')
317 plt.grid(True)
318 plt.title('Cross Validation for
319           max_depth')
320 plt.ylabel('Accuracy [%]')
321 plt.xlabel('Parameter value')
322 plt.legend()
323 plt.show()
324
325 # In[79]:
326
327 num_estSplit = [1.,2, 5,10, 15,
328                 20,50]
329 train_scoreSplit, test_scoreSplit=
330 validation_curve(
331     RandomForestClassifier(
332         max_depth = 10),
333     X_train, y = y_train,
334     param_name = 'min_samples_split
335     ',
336     param_range = num_estSplit, cv
337     = 3)
338
339 # In[84]:
340
341 plt.plot(num_estSplit,
342          test_scoreSplit, label = 'Test')
343 plt.plot(num_estSplit,
344          train_scoreSplit, label = 'Train
345          ')
346 plt.grid(True)
347 plt.title('Cross Validation for
348           min_samples_split')
349 plt.ylabel('Accuracy [%]')
350 plt.xlabel('Parameter value')
351 plt.legend()

```

343	plt.show()	379	
344		380	
345		381	
346	# In[85]:	382	# In[87]:
347		383	
348		384	
349	num_estLeaf = [1,5,15,50,1000]	385	num_est = ['auto', 'sqrt', 'log2',
350	train_scoreLeaf, test_scoreLeaf=		1,2,3,4,5,6,7]
	validation_curve(386	train_scoreMaxFeat,
351			test_scoreMaxFeat=
	RandomForestClassifier(validation_curve(
	max_depth = 10,	387	
	min_samples_split=15),		RandomForestClassifier(
352	X =		max_depth = 10,
	X_train, y = y_train,		min_samples_split=15,
353			min_samples_leaf=2),
	param_name = 'min_samples_leaf'	388	X =
	,		X_train, y = y_train,
354		389	
	param_range = num_estLeaf, cv =		param_name = 'max_features',
	3)	390	
355			param_range = num_est, cv = 3)
356		391	
357	# In[88]:	392	
358		393	# In[89]:
359		394	
360	plt.plot(num_estLeaf, test_scoreLeaf,	395	
	label = 'Test')	396	plt.plot(num_est, test_scoreMaxFeat,
361	plt.plot(num_estLeaf,		label = 'Test')
	train_scoreLeaf, label = 'Train'	397	#plt.plot(num_est,
)		train_scoreMaxFeat, label = '
362	plt.grid(True)		Train')
363	plt.title('Cross Validation for	398	plt.grid(True)
	min_samples_leaf')	399	plt.title('Cross Validation for
364	plt.ylabel('Accuracy [%]')		max_features')
365	plt.xlabel('Parameter value')	400	plt.ylabel('Accuracy [%]')
366	plt.legend()	401	plt.xlabel('Parameter value')
367	plt.show()	402	plt.legend()
368		403	plt.show()
369		404	
370	# In[]:	405	
371		406	# At this point we are ready to
372			build the forest with tuned
373			parameters
374		407	
375		408	# In[91]:
376	# In[]:	409	
377		410	
378		411	#Import Random Forest Model

```

412 from sklearn.ensemble import
    RandomForestClassifier
413
414 #Create a Gaussian Classifier
415 rf=RandomForestClassifier(
    n_estimators=500, max_features
    = 6, max_depth = 50,
    min_samples_split=15,
    min_samples_leaf=1)
416
417 # Train the model on training data
418 rf.fit(X_train, y_train);
419
420
421 # In[92]:
422
423
424 # Use the forest's predict method
    on the test data
425 rf_y_pred = rf.predict(X_test)
426
427
428 # In[93]:
429
430
431 accuracy_score(y_test, rf_y_pred)
432
433
434 # we could print one of the trees
    used in the forest (if the
    length of the tree is not
    specified, the result could be
    a large PNG image)
435
436 # In[ ]:
437
438
439 # Import tools needed for
    visualization
440 from sklearn.tree import
    export_graphviz
441 import pydot
442 # Pull out one tree from the forest
443 tree = rf.estimators_[5]
444 # Import tools needed for
    visualization
445 from sklearn.tree import
    export_graphviz
446 import pydot
447 # Pull out one tree from the forest
448 tree = rf.estimators_[5]
449 # Export the image to a dot file
450 export_graphviz(tree, out_file = '
    tree.dot', feature_names =
    feature_names, rounded = True,
    precision = 1)
451 # Use dot file to create a graph
452 (graph, ) = pydot.
    graph_from_dot_file('tree.dot')
453 Image(graph.create_png())
454
455
456 # Let's check Feature importance at
    this level
457
458 # In[94]:
459
460
461 feature_imp = pd.Series(rf.
    feature_importances_, index=
    feature_names).sort_values(
    ascending=False)
462
463
464 # In[95]:
465
466
467 import seaborn as sns
468 get_ipython().run_line_magic('
    matplotlib', 'inline')
469 # Creating a bar plot
470 sns.barplot(x=feature_imp, y=
    feature_imp.index)
471 # Add labels to your graph
472 plt.xlabel('Feature Importance
    Score')
473 plt.ylabel('Features')
474 plt.title("Visualizing Important
    Features")
475 plt.legend()
476 plt.show()
477
478
479 # 3d Plots of the dataset projection
    on the 3 most important
    features, comparing model(1st
    plot) with actual labels (
    second)

```

```

480
481 # In[96]:
482
483
484 Results = X_test.copy()
485 decoded_model = np.argmax(rf_y_pred
486 , axis=1)
487 decoded_target = np.argmax(np.
488 asarray(y_test), axis=1)
489
490 # In[97]:
491
492 Results['actual'] = decoded_target
493 Results['model'] = decoded_model
494
495
496 # In[ ]:
497
498
499
500 import plotly.express as px
501
502 fig = px.scatter_3d(Results, x='
503 error', y='Y', z='Chi',
504 color='model')
505 fig.show()
506
507 # In[ ]:
508
509
510
511 import plotly.express as px
512
513 fig = px.scatter_3d(Results, x='
514 error', y='Y', z='Chi',
515 color='actual')
516 fig.show()
517
518 # In[ ]:

```

5.8 Classification.ipynb

```

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

```

```

3
4 # # Best Classification
5
6 # In[6]:
7
8 #Importing the libraries to watch
9 the 'fits' image and get the
10 data array
11 import astropy
12 #import plotly.graph_objects as go
13 from astropy.io import fits
14 #Importing a library that is useful
15 to read the original file
16 import pandas as pd
17 import pylab as plb
18 import matplotlib.pyplot as plt
19 from scipy.optimize import
20 curve_fit
21 from scipy import asarray as ar,exp
22 #Importing a visual library with
23 some illustrative set up
24 import matplotlib.pyplot as plt
25 import matplotlib.colors as mcolors
26 from matplotlib import cm
27 import numpy as np
28 from sklearn.utils.testing import
29 ignore_warnings
30 from sklearn.exceptions import
31 ConvergenceWarning
32 from sklearn.decomposition import
33 PCA
34 import math
35 import seaborn as sns
36 from sklearn.linear_model import
37 LogisticRegression
38 plt.style.use('fivethirtyeight')
39 plt.rcParams['font.family'] = 'sans
40 -serif'
41 plt.rcParams['font.serif'] = '
42 Ubuntu'
43 plt.rcParams['font.monospace'] = '
44 Ubuntu Mono'
45 plt.rcParams['font.size'] = 14
46 plt.rcParams['axes.labelsize'] = 12
47 plt.rcParams['axes.labelweight'] =
48 'bold'
49 plt.rcParams['axes.titlesize'] = 12
50 plt.rcParams['xtick.labelsize'] =

```

```

12
39 plt.rcParams['ytick.labelsize'] = 12
40 #plt.rcParams['legend.fontsize'] = 12
41 plt.rcParams['figure.titlesize'] = 12
42 plt.rcParams['image.cmap'] = 'jet'
43 plt.rcParams['image.interpolation'] = 'none'
44 plt.rcParams['figure.figsize'] = (16, 8)
45 plt.rcParams['lines.linewidth'] = 2
46 plt.rcParams['lines.markersize'] = 8
47 plt.rcParams["axes.grid"] = False
48
49
50 # In[4]:
51
52
53 #Importing the dataset and
    displaying the first 5 rows
54 data=pd.read_csv('star.txt',sep='\s
    +')
55
56
57 # In[6]:
58
59
60 data.head()
61
62
63 # In[7]:
64
65
66 #Dropping the Sharp and the #ID
    from the dataset
67 notar=data.drop(columns=['Sharp', '#
    ID'])
68
69
70 # # P.C.A. Excluding Space
71
72 # In[6]:
73
74
75 #Dropping the X and the Y to
    perform PCA
76 notar=data.drop(columns=['X','Y'])
77
78
79 # In[9]:
80
81
82 notar.head()
83
84
85 # In[7]:
86
87
88 #Performing the P.C.A.
89 pca=PCA(n_components=3)
90 pca=pca.fit(notar)
91 pca_data=pd.DataFrame(pca.transform
    (notar))
92
93
94 # In[8]:
95
96
97 pca_data=pca_data.rename(columns
    ={0: 'FirstComponent', 1: '
    SecondComponent', 2: '
    ThirdComponent'})
98
99
100 # In[57]:
101
102
103 pca_data['X']=data.X
104 pca_data['Y']=data.Y
105
106
107 # In[58]:
108
109
110 pca_data['Sharp']=data.Sharp
111
112
113 # In[59]:
114
115
116 pca_data.head()
117
118
119 # In[60]:
120

```

```

121
122 #pca_data=pca_data.drop(columns=['
    Sharp'])
123
124
125 # # Best method 2 features 3
    classes
126
127 # In[59]:
128
129
130 #Two features, three classes
    preprocessing
131 opt_data=pca_data[['FirstComponent'
    , 'SecondComponent']]
132
133
134 # In[60]:
135
136
137 data['SharpSign']=data.Sharp.apply(
    np.sign)
138 opt_data['Target']=data['SharpSign']
139
140
141 # In[14]:
142
143
144 #plotting the First two component,
    together with the target
145 sns.scatterplot(opt_data.
    FirstComponent,opt_data.
    SecondComponent,hue=opt_data.
    Target,palette='plasma')
146 plt.grid(True)
147
148
149 # In[63]:
150
151
152 #Dataset and target
153 X=opt_data.drop(columns=['Target'])
154 y=opt_data.Target
155
156
157 # In[20]:
158
159
160 #Importing the SVM and the train
    test split
161 from sklearn.svm import SVC
162 from sklearn.model_selection import
    train_test_split
163
164
165 # In[62]:
166
167
168 # (Train+validation)/ test split at
    90%
169 X_train, X_test, y_train, y_test =
    train_test_split(
170     X, y, test_size=0.9,
    random_state=42)
171
172
173 # In[18]:
174
175
176 # Train/Validation split at 50%
177 X_train, X_val, y_train, y_val =
    train_test_split(
178     X_train, y_train, test_size
    =0.5, random_state=42)
179
180
181 # In[19]:
182
183
184 K_LIST=['linear', 'poly', 'rbf', '
    sigmoid']
185
186
187 # In[20]:
188
189
190 #Validation on kernels
191 BEST_KERNEL=[]
192 k=0
193 for i in range(5):
194     FIN_SCORE=[]
195     X_train, X_test, y_train,
    y_test = train_test_split(
196         X, y, test_size=0.9,
    random_state=42)
197     X_train, X_val, y_train, y_val
    = train_test_split(

```

```

198     X_train, y_train, test_size=0.5, random_state=42)
199     for ker in K_LIST:
200         clf=SVC(kernel=ker)
201         clf.fit(X_train,y_train)
202         fin_score=clf.score(X_val, y_val)
203         FIN_SCORE.append(fin_score)
204         k=k+1
205         print(ker + ' Kernel has been explored')
206         FIN_SCORE=np.array(FIN_SCORE)
207
208         BEST_KERNEL.append(K_LIST[FIN_SCORE.argmax()])
209         print('Cross validation ' + str(i) + ' out of 4 \n')
210
211 # In[7]:
212
213
214 BEST_KERNEL=['rbf','rbf','rbf','rbf','rbf']
215
216
217 # In[9]:
218
219
220 sns.countplot(BEST_KERNEL)
221 plt.xlabel('Chosen Kernel',fontsize=20)
222
223
224 # In[22]:
225
226
227 best_kernel='rbf'
228
229
230 # In[23]:
231
232
233 #Validation on C values
234 c_list=np.arange(0.5,50.5,0.5)
235 k=0
236 PERC=['20%','40%','60%','80%','100%']
237 K=[20,40,60,80,100]
238
239
240 BEST_C=[]
241 for i in range(5):
242     FIN_SCORE=[]
243     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.9, random_state=42)
244     X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.5, random_state=42)
245     k=0
246     for c in c_list:
247         k=k+1
248         clf=SVC(C=c,kernel=best_kernel)
249         clf.fit(X_train,y_train)
250         fin_score=clf.score(X_val, y_val)
251         FIN_SCORE.append(fin_score)
252         #k=k+1
253         if k in K:
254             ind=K.index(k)
255             print (PERC[ind] + ' of the C values has been explored')
256
257     FIN_SCORE=np.array(FIN_SCORE)
258     BEST_C.append(c_list[FIN_SCORE.argmax()])
259     print('Cross validation ' + str(i) + ' out of 4 \n')
260
261 # In[24]:
262
263
264 sns.countplot(BEST_C)
265 plt.xlabel('Chosen C')
266 plt.ylabel('Count')
267
268 # In[25]:
269
270
271 FIN_SCORE=np.array(FIN_SCORE)
272
273 # In[15]:
274
275

```

```

278
279 #Best parameters
280 best_c=14.5
281 best_kernel='rbf'
282
283
284 # In[64]:
285
286
287 #Train/Test rigid split
288 X_train, X_test, y_train, y_test =
    train_test_split(
289     X, y, test_size=0.7,
        random_state=42)
290
291
292 # In[65]:
293
294
295 #Fit with the best parameters
296 clf=SVC(kernel=best_kernel,C=best_c
    )
297 clf.fit(X_train,y_train)
298 fin_score=clf.score(X_test,y_test)
299
300
301 # In[66]:
302
303
304 prediction=clf.predict(X_test)
305
306
307 # In[67]:
308
309
310 #Results
311 print('The final score with 2
    feature is ' + str(fin_score
        *100) + '% ')
312
313
314 # In[68]:
315
316
317 #Target/prediction comparison
318 pred_data=X_test.copy()
319 pred_data['Target']=y_test
320 pred_data['Prediction']=prediction
321
322
323 # In[34]:
324
325
326 #plotting the decision surfaces
327 import matplotlib.patches as
    mpatches
328 import matplotlib.pyplot as plt
329 def make_meshgrid(x, y, h=.4):
330     x_min, x_max = x.min() - 1, x.
        max() + 1
331     y_min, y_max = y.min() - 1, y.
        max() + 1
332     xx, yy = np.meshgrid(np.arange(
        x_min, x_max, h), np.arange(
        y_min, y_max, h))
333     return xx, yy
334
335 def plot_contours(ax, clf, xx, yy,
    **params):
336     Z = clf.predict(np.c_[xx.ravel(
        ), yy.ravel()])
337     Z = Z.reshape(xx.shape)
338     out = ax.contourf(xx, yy, Z, **
        params)
339     return out
340
341
342 # In[32]:
343
344
345 #renaming
346 y=y_test
347
348
349 # In[35]:
350
351
352 fig, ax = plt.subplots()
353 # title for the plots
354 title = ('Decision surface of
    linear SVC ')
355 # Set-up grid for plotting.
356 X0, X1 = X_test['FirstComponent'],
    X_test['SecondComponent']
357 xx, yy = make_meshgrid(X0, X1)
358
359 plot_contours(ax, clf, xx, yy, cmap
    ='plasma', alpha=0.8)

```



```

360 ax.scatter(X0, X1, c=y, cmap='
    plasma', s=20, edgecolors='k')
361 ax.set_ylabel('Second Component')
362 ax.set_xlabel('First Component')
363
364
365 violet_patch = mpatches.Patch(color
    ='navy', label='Sharp<0')
366 yellow_patch = mpatches.Patch(color
    ='gold', label='Sharp>0')
367 pink_patch = mpatches.Patch(color='
    magenta', label='Sharp=0')
368
369 plt.legend(handles=[violet_patch,
    yellow_patch, pink_patch])
370
371 ax.set_xticks(())
372 ax.set_yticks(())
373 ax.set_title('Decision Surface',
    fontsize=20)
374 #ax.legend()
375 plt.show()
376
377
378 # In[36]:
379
380
381 test_data=pred_data
382
383
384 # In[37]:
385
386
387 #Confusion matrix
388 import itertools
389 from string import ascii_uppercase
390 from sklearn.metrics import
    confusion_matrix
391
392 y_test=test_data.Target
393 predic = prediction
394
395 columns = ['Negative', 'Zero', '
    Positive']
396
397 confm = confusion_matrix(y_test,
    predic)
398 df_cm = pd.DataFrame(confm.astype(
    float), index=columns, columns=
    columns)
399
400 ax = sns.heatmap(df_cm, cmap='
    plasma', annot=True, fmt='g')
401
402
403 # In[103]:
404
405 #Defining precision and recall for
    three classes classification
406 def precision(confusion, clas):
407     if clas=='Negative':
408         TP=confusion[0][0]
409         FN=confusion[1][0]+
410             confusion[2][0]
411         pres=TP/(TP+FN)
412     if clas=='Positive':
413         TP=confusion[2][2]
414         FN=confusion[2][0]+
415             confusion[2][1]
416         pres=TP/(TP+FN)
417     if clas=='Zero':
418         TP=confusion[1][1]
419         FN=confusion[1][0]+
420             confusion[1][2]
421         pres=TP/(TP+FN)
422     return pres
423
424
425 # In[104]:
426 def recal(confusion, clas):
427     if clas=='Negative':
428         TP=confusion[0][0]
429         FP=confusion[0][1]+
430             confusion[0][2]
431         rec=TP/(TP+FP)
432     if clas=='Positive':
433         TP=confusion[2][2]
434         FP=confusion[0][2]+
435             confusion[1][2]
436         rec=TP/(TP+FP)
437     if clas=='Zero':
438         TP=confusion[1][1]
439         FP=confusion[0][1]+
440             confusion[2][1]
441         rec=TP/(TP+FP)

```

```

439         return rec
440
441
442 # In[41]:
443
444 precision(confm, 'Negative'),
445 precision(confm, 'Positive'),
446 precision(confm, 'Zero')
447
448 # In[42]:
449
450 recal(confm, 'Negative'), recal(confm,
451                                   'Positive'), recal(confm, 'Zero')
452
453
454 # In[44]:
455
456 #Defining statistics summary
457 def statistics(confusion):
458     neg=[precision(confm, 'Negative'),
459          recal(confm, 'Negative')]
460     pos=[precision(confm, 'Positive'),
461          recal(confm, 'Positive')]
461     zero=[precision(confm, 'Zero'),
462           recal(confm, 'Zero')]
462     stats=pd.DataFrame({'Negative': neg,
463                         'Positive': pos,
464                         'Zero': zero})
463     stats.index=['Precision', 'Recall']
464     return stats
465
466
467 # In[45]:
468
469 statistics(confm)
470
471
472 # # Best Method
473
474 # In[70]:
475
476
477
478 #On the yellow data, where the
479 algorithm performs poorly, a
480 Random Forest algorithm is
481 applied
482
483 wrong_data=pred_data[pred_data['
484 Prediction']==1.0].drop(columns
485 =['Target'])
486 wrong_target=pred_data[pred_data['
487 Prediction']==1.0].Target
488
489
490 # In[71]:
491
492 #Plotting the difficult to predict
493 data points
494 sns.scatterplot(wrong_data.
495                 FirstComponent, wrong_data.
496                 SecondComponent, hue=
497                 wrong_target)
498
499
500 # In[72]:
501
502 #Implement random forest on a split
503 set
504 X=data.loc[X_test.index].drop(
505     columns=['#ID', 'Sharp', '
506 SharpSign'])
507 y=data.loc[X_test.index].SharpSign
508 X_train, X_test, y_train, y_test =
509     train_test_split(
510         X, y, test_size=0.2,
511         random_state=42)
512
513
514 # In[73]:
515
516 #Import Random Forest Model
517 from sklearn.ensemble import
518 RandomForestClassifier
519
520
521 #Create a Gaussian Classifier
522 rf=RandomForestClassifier(
523     n_estimators=500,

```

```

min_samples_split = 20,
max_features = 5)

# In[76]:

# 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

```

```

586 from sklearn.metrics import
    confusion_matrix
587
588 y_test=Results.Target
589 predic = Results.Pred
590
591 columns = ['Negative','Zero','
    Positive']
592
593 confm = confusion_matrix(y_test,
    predic)
594 df_cm = pd.DataFrame(confm.astype(
    float), index=columns, columns=
    columns)
595
596 ax = sns.heatmap(df_cm, cmap='
    plasma',annot=True,fmt='g')
597
598
599 # In[105]:
600
601
602 #Using the statistics function to
    see the precisions and the
    recall
603 def statistics(confusion):
604     neg=[precision(confm,'Negative'
    ),recal(confm,'Negative')]
605     pos=[precision(confm,'Positive'
    ),recal(confm,'Positive')]
606     zero=[precision(confm,'Zero'),
    recal(confm,'Zero')]
607     stats=pd.DataFrame({'Negative':
    neg,'Positive':pos,'Zero':zero
    })
608     stats.index=['Precision','
    Recall']
609     return stats
610
611
612 # In[106]:
613
614
615 statistics(confm)
616
617
618 # In[94]:
619
620
621 #Summary of all the methods
622 Tot_res=pd.DataFrame({'Performance'
    : [71,74,80,82]})
623 Tot_res.index=['SVM','Decision Tree
    ','Random Forest','Ensemble
    Learning']
624 Tot_res
625
626
627 # In[100]:
628
629 #Barplot of all the methods
    performances
631 sns.barplot(x=Tot_res.index,y=
    Tot_res.Performance,palette='
    plasma')
632 plt.xlabel('Method',fontsize=20)
633 plt.ylabel('Accuracy (%)',fontsize
    =20)
634 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://roche.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](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.