Faculty of Engineering, Environment and Computing
6006CEM Machine Learning and Related Applications
Student ID: 8897758

Coventry GitHub Repository URL:

https://github.coventry.ac.uk/valente3/6006CEM_2021s1_8897758_SV.git

Dataset(s) URL(s):

Prediction Pulsar Star: https://www.kaggle.com/colearninglounge/predicting-pulsar-starintermediate

Faculty of Engineering, Environment and Computing
6006CEM Machine Learning and Related Applications
Student ID: 8897758

Table of Contents

Academic Report

Introduction

According to astronomers, Pulsar stars are highly magnetized compact stars that rotate extremely rapidly and emit electromagnetic radiation through their poles. They belong to a subclass of Neutron stars, which originate from a supernova explosion. This happens when its core goes through a gravitational collapse and compresses until there is no more space between neutrons. These stars were proposed by Walter Baade and Fritz Zwicky in 1934. However, they were only observed in 1967 by Jocelyn Burnell and Antony Hewish, (Cofield, 2016).

Though, to date, less than 2,000 have been found when is predicted to exist about a billion neutron stars. The main reason for this is age. Most Neutron stars are billions of years old, which translates into cooler temperatures and slower rotation speed. The lack of energy enables its emissions, mostly radio waves, to reach Hearth, making them invisible, (Naeye, 2007).

With a Universe in constate state of expansion, astrology is experiencing exponential growth in data collection and complexity. Thus, Machine Learning (ML) algorithms are essential to automate the detection and classification of these astronomical objects, (Baron, 2019). Thus, in this project ML data analysis techniques, such as Feature Selection, and classification algorithms, such as Logistic Regression and KNN, are used to develop an efficient system to detect and classify Neutron stars as a pulsar.

Dataset

Dataset Description

The dataset used in this project was the *HTRU2* dataset*,* which was later renamed *Predicting pulsar Star.* The featured data was extracted by Dr Robert Lyon during the High Time Resolution Universe Survey from candidate files using the PulsarFeatureLab tool at Manchester University. The data gathered includes 16,259 spurious examples caused by RFI/noise and 1,639 real pulsar examples, (Lyon et all, 2017).

Each star is described by eight variables and a single class variable
Variables: **Mean of Integrated Profile, Standard Deviation of the Integrated Profile, Excess Kurtosis of the Integrated Profile, Skewness of the Integrated Profile, Mean of the DM-SNR Curve, Standard Deviation of the DM-SNR Curve, Excess Kurtosis of the DM-SNR Curve, Skewness of the DM-SNR Curve and Class (target_class).** Figure 1 demonstrates the variable in the dataset.

```
--------------------------------------------------
Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12528 entries, 0 to 12527
Data columns (total 9 columns):
 #   Column                                         Non-Null Count  Dtype
---  ------                                         --------------  -----
 0    Mean of the integrated profile                12528 non-null  float64
 1    Standard deviation of the integrated profile  12528 non-null  float64
 2    Excess kurtosis of the integrated profile     10793 non-null  float64
 3    Skewness of the integrated profile            12528 non-null  float64
 4    Mean of the DM-SNR curve                      12528 non-null  float64
 5    Standard deviation of the DM-SNR curve        11350 non-null  float64
 6    Excess kurtosis of the DM-SNR curve           12528 non-null  float64
 7    Skewness of the DM-SNR curve                  11903 non-null  float64
 8   target_class                                   12528 non-null  float64
dtypes: float64(9)
memory usage: 881.0 KB
None
```

Figure 1 - data.info() command in the dataset HTRU2

Data Analysis

Before applying any ML algorithm, it is required for the data to be analysed regarding its shape, variables, columns and data type. Therefore, after importing the data, data.shape, data.isna()sum() and data.info() are applied, figure 1 and 2. This concludes that the dataset includes 12528 rows and 9 columns, only three columns present null values: **Excess Kurtosis of the Integrated Profile** (1735), **Standard Deviation of the DM-SNR Curve** (1178) and **Skewness of the DM-SNR Curve** (625).

```
------------------------------------------------------------
Dataset's Shape:  (12528, 9)
------------------------------------------------------------
Null Values:
 Mean of the integrated profile                    0
 Standard deviation of the integrated profile      0
 Excess kurtosis of the integrated profile      1735
 Skewness of the integrated profile                0
 Mean of the DM-SNR curve                          0
 Standard deviation of the DM-SNR curve         1178
 Excess kurtosis of the DM-SNR curve               0
 Skewness of the DM-SNR curve                    625
target_class                                       0
dtype: int64
------------------------------------------------------------
```

Figure 2 - data.shape and data.isna().sum() applied to the HTRU2 dataset

For a deeper analysis of null values, data.head() was applied for an overview, which followed a msno.matrix(data) for a graphical synopsis of the influence of the null values within the dataset, figure 3.

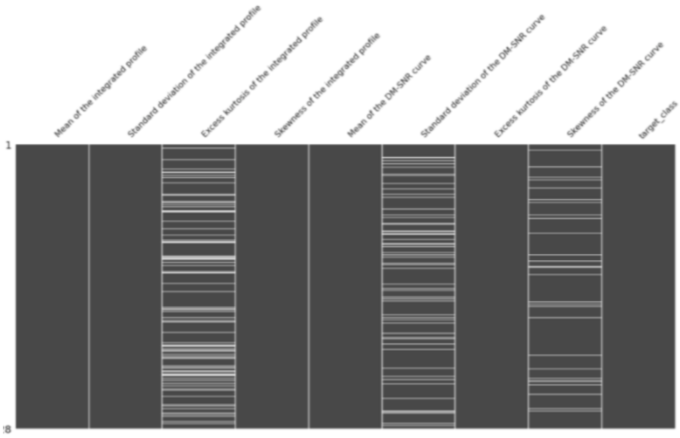

Figure 3 - msno.matrix(data) represents graphically the null values in a dataset

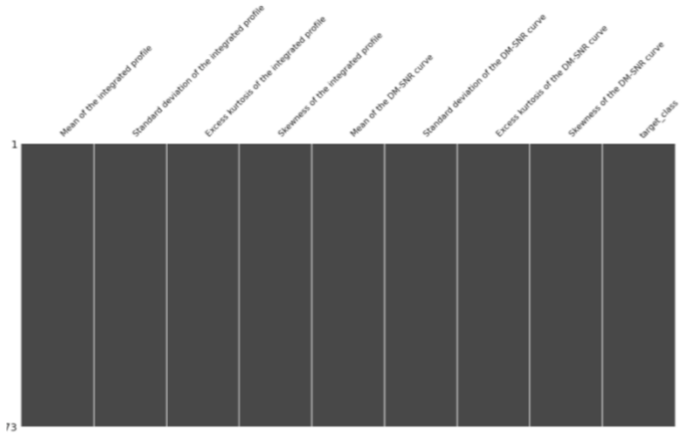

Figure 4 - msno.matrix(data) represents graphically the lack of null values in a dataset

Pre-Processing

Handling Null Values

Building an ML algorithm is a multistep process, in which information pre-processing is a part of guaranteeing data quality, it influences the model's learning process. These operations are divided into data engineering for converting and preparing the raw data, and feature engineering to tune the prepared data. This prepares the data to create features feed to the models.

While analysing the data, figure 2 and 3, null values were identified. Thus, by applying the data.dropna(inplace=True) the rows that included Nan values were deleted. Figure 4 confirms the deletion of the null values. Additionally, data.shape proves the row reduction to 9273.

Handling Categorical Values

Categorial values are aggregated data into groups, such as gender, rather than in numeric format. Since ML algorithms are not designed to deal with data groups, this data is converted into numeric. However, the measurements of the HTRU2 dataset are already of numerical type, discarding the need for this process. Figure 5 represents the data distribution of the target_class column.

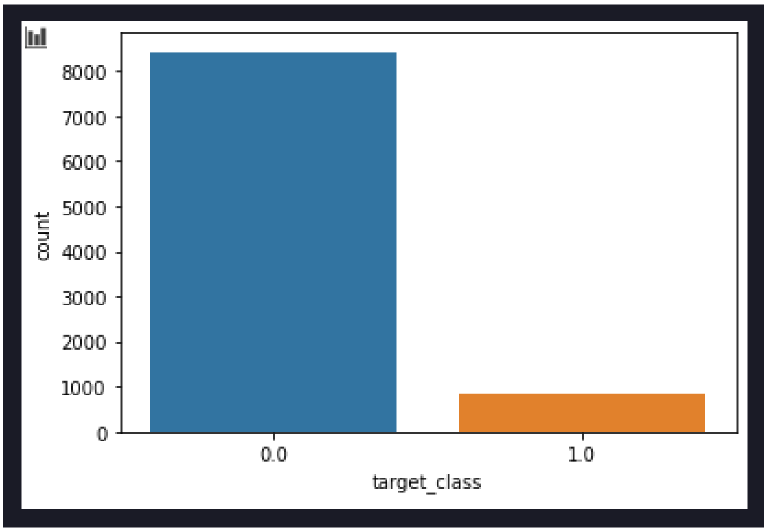

Figure 5 - Barchart represent the quantity of pulsar stars and non pulsar stars in the dataset

Feature Selection

Feature Selection is the process of tunning down the number of features based on their correlation between each other as some features are less relevant or too similar, (Chandrashekar et all, 2014). Figure 6 represents the correlated data. This provides a way of reducing computation time and improving the model's performance. Thus, a heatmap is ideal to visualize the data correlation.

```
#FEATURE SELECTION
#finds correlations between data

data_corr = data.corr()
data_corr.head()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
Mean of the integrated profile	1.000000	0.554197	-0.872497	-0.734920	-0.299984	-0.307431	0.236010	0.146103	-0.675819
Standard deviation of the integrated profile	0.554197	1.000000	-0.528370	-0.542560	-0.011061	-0.059486	0.036907	0.030959	-0.368223
Excess kurtosis of the integrated profile	-0.872497	-0.528370	1.000000	0.944715	0.421126	0.436362	-0.344571	-0.216748	0.790866
Skewness of the integrated profile	-0.734920	-0.542560	0.944715	1.000000	0.415570	0.415902	-0.328328	-0.204109	0.704743
Mean of the DM-SNR curve	-0.299984	-0.011061	0.421126	0.415570	1.000000	0.796449	-0.614526	-0.353186	0.407043

Figure 6 - Correlated data

Feature selection is applied manually or automatically. For this project an automatic approach was implemented via a *for loop* comparing all features to each other and dropping the ones with correlation higher than 85%. Figure 7 and 8 express the correlation between the features before and after feature selection.

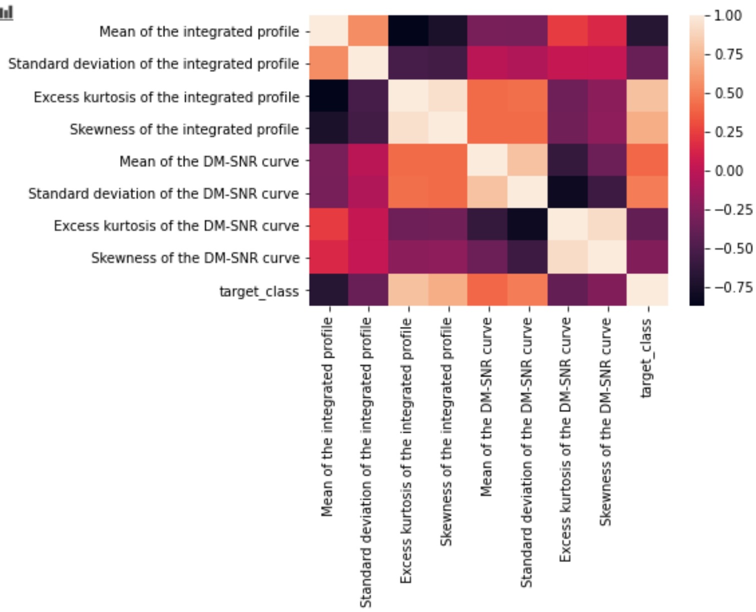

Figure 7 - Heatmap of correlated data before feature selection

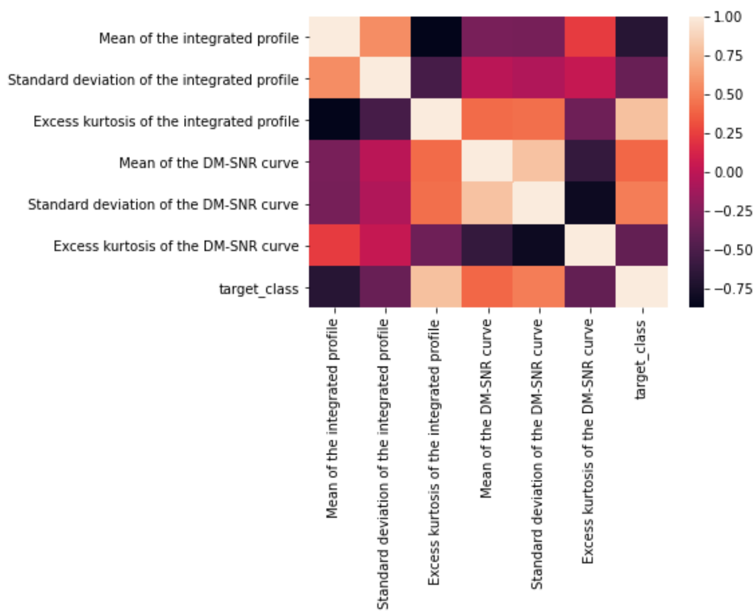

Figure 8 - Heatmap of correlated data after feature selection

After Feature Selection, the method automatically deletes one of the features of the pair with high correlation measures from the dataset: **Skewness of the Integrated Profile** and **Skewness of the DM-SNR Curve.** To very the consequences of feature selection in the learning models, all models will be applied to raw data and tunned data.

Data scaling and Splitting

Data scaling aligns all values in the same range and magnitude to avoid variables from dominating over others. Scaling allows for a better understanding of the models while improving their accuracy. According to Saini (2019), Feature scaling has a greater impact on some algorithms, such as KNN, PCA and Gradient Descent.

There are two methods of data scaling, normalization and standardization. In this system, StandardScaler() is employed as a standardization methodology to scale data_X after the taget_class (data_y) drops from the data.

Later, data splitting regroups the data into two subsets: a training set and a testing set, with train_test_split() function. The training set is then responsible for training the learning models to predict an output, while the testing set focuses on testing the quality of the model. The splitting avoids miscalculations, thus decreasing the risk of imprecise predictions. For data analysis, a PCA was plotted is in figure 9 to evaluate the variance of pulsar stars and non-pulsar stars. In this case, non-pulsar stars represent a higher variance than pulsars.

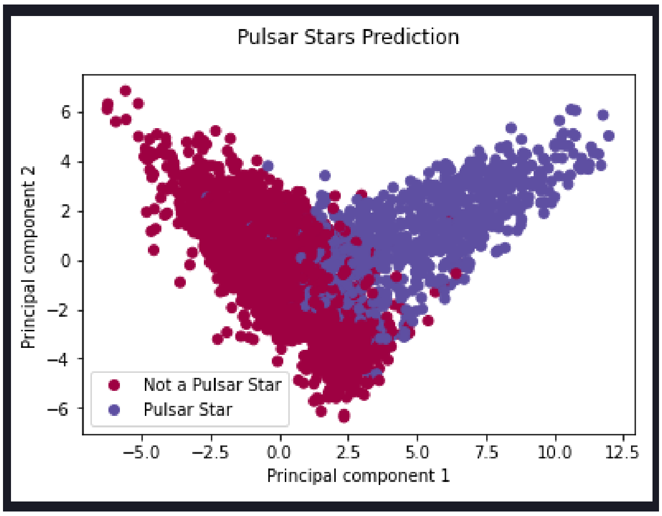

Figure 9 - PCA plot after data scaling and splitting

Implementation

After the pre-processing, the data is suitable for integration in the learning models: Logistic Regression (LR) and K-Neighbors Classifier (KNN). Additionally, to complement the data tunning, hyperparameter optimisation is implemented using two methods: Grid Search (GS) and Random Search (RS).

To compare the effectiveness of the learning models, raw data (data) and feature selected data (data_f) are used directly with the models. Furthermore, the hyperparameter optimization is included later. Table 1 demonstrates all the experimented combinations.

	Algorithms/Data
None	LR+data
	KNN+data
	LR+data_f
	KNN+data_f
Grid Search (GS)	LR+data
	KNN+data
	LR+data_f
	KNN+data_f
Random Search (RS)	LR+data
	KNN+data
	LR+data_f
	KNN+data_f

Table 1 - Learning models and data applied

Even though twelve algorithms were applied, all of them present the same basis. After data scaling and splitting, the model is trained and the target_class from the testing set is predicted with model.predict(X_test).

```
#training the model
model_LR = LogisticRegression()
model_LR = model_LR.fit(X_train, y_train)
y_train_pred = model_LR.predict(X_train)
y_test_pred = model_LR.predict(X_test)
```

Figure 10 - Training process LR

Then Cross-Validation technique evaluates the generalisation aptitude of a classification model after the hyperparameter optimization through the cross_val_score() method (figure 11). Cross-Validation is considered a model validation technique built to prevent overfitting and selection bias by providing an estimate on the new data.

```
#acuracy results from models
def accuracy_results(model, X_train, y_train, X_test, y_test, y_test_pred):

    #evaluate a score by cross-validation
    scores = cross_val_score(model, X_test, y_test, cv=5, scoring='accuracy')
```

Figure 11 - Cross-Validation technique

Finally, the accuracy is measured and the confusion matrix calculated. Confusion_matrix() calculates the matrix, while sns.heatmap(conf_matrix, annot=True) plotted with a heatmap. This process describes the performance of the learning model with the testing set (y_test) and the prediction of the testing set (y_test_pred).

Hyperparameter optimization is not present in all algorithms as a reference is necessary to compare and explore their implications and influence. Thus, GS and RS were applied to both LR and KNN with data and data_f. For this some parameters are required to be set. In figure 12 and 13 reveal the parameters of the hyperparameters techniques, which include: **solvers**, **penalties** and **penalty strength** (c_param). GS and RS test the model with every combination so parameters and select the most efficient one to be used.

```
#GRID SEARCH

#Defining parameters
#defining solvers optimises the algorithm
solvers = ['newton-cg', 'lbfgs', 'liblinear']
#penalises the hyperparameter
penalty = ['l1', 'l2']
#strengh of penalty
c_param = [100, 10, 1.0, 0.1, 0.01, 0.001]

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
param_GS = dict(solver=solvers, penalty=penalty, C=c_param)
model_GS = GridSearchCV(estimator=LogisticRegression(), param_grid=param_GS, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)
```

Figure 12 - Grid Seard parameters

```
#RANDOM SEARCH

#Defining parameters
solvers = ['newton-cg', 'lbfgs', 'liblinear']
penalty = ['l1', 'l2']
c_param = [100, 10, 1.0, 0.1, 0.01, 0.001]

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
param_RS = dict(solver=solvers, penalty=penalty, C=c_param)

model_RS = RandomizedSearchCV(LogisticRegression(), param_RS, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)
```

Figure 13 - Random Search parameters

After the hyperparameter optimization completes, the model variable is used to train the model and obtain the predicted value of X_test.

Results and evaluations

Each learning module was assessed according to its Model Training Accuracy model.score(X_train, y_train), Model Testing Accuracy model.score(X_test, y_test), Maximum Scaled Accuracy accuracy_score(y_test, y_test_pred), Cross-Validation Accuracy scores.mean() and Process Time. Although, the models built with GS and RS also measured the Best Accuracy model.best_score_ and Best Hyperparameters model.best_params_. Additionally, for each model, a confusion matrix was plotted using a heatmap.

Logistic Regression (LR)

Figure 14 and 15 illustrate the influence of data treatment and processing. LR+data_f implements data treatment to remove rows that include null values and data processing through feature selection. Even though this decreases the processing time, it also reduces the accuracies by less than 1%.

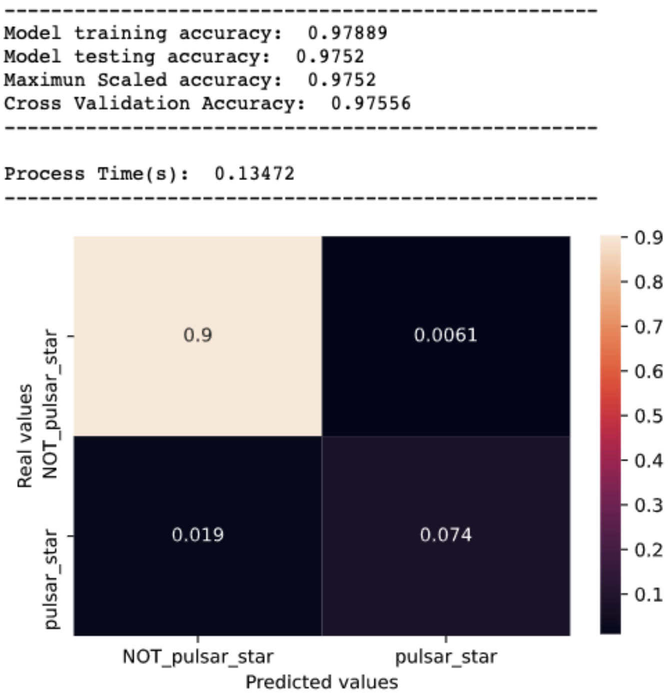

Figure 14 - Confusion Matrix of LR with data

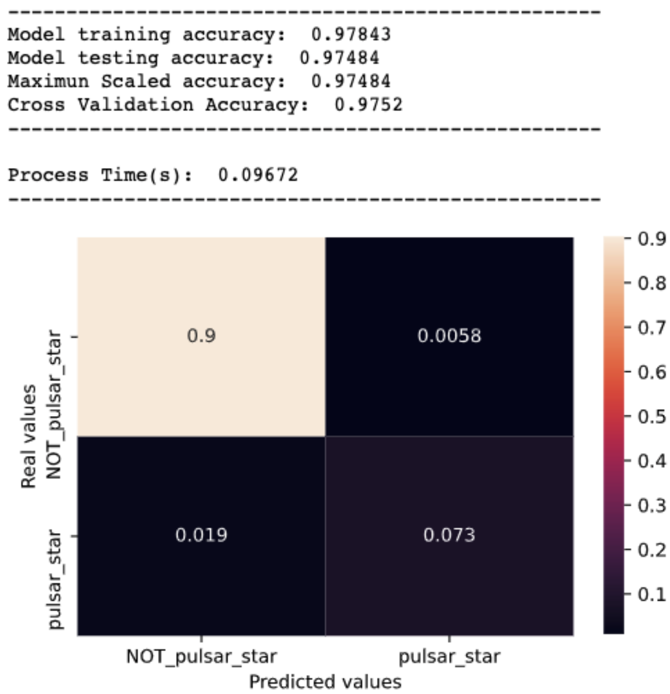

Figure 15 - Confusion Matrix of LR with data_f

Grid Search (GS)

These two models apply LR and GS to raw data (data) and processed data (data_f). Thus, figure 16 and 17 illustrated the impact of the GS technique with an LR algorithm. Both models identified the same parameters for the best accuracy, but LR+GS+data_f is 5 seconds faster. Accuracies for both models are identical with a difference of less than 0.1% approximately.

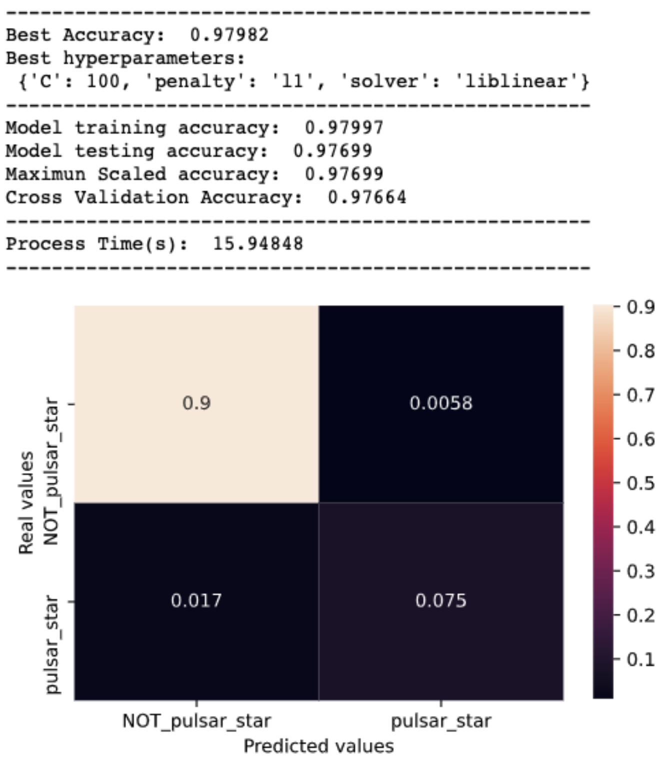

Figure 16 - Confusion Matrix of LR+GS with data

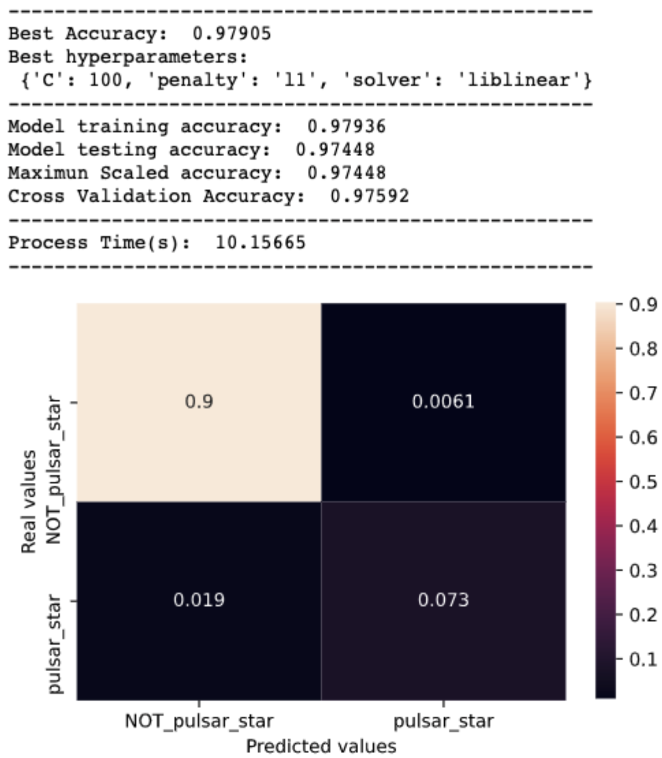

Figure 17 - Confusion Matrix of LR+GS with data_f

Random Search (RS)

This primary analysis and comparison provide a better understanding of the importance of pre_processing data in learning models. Similarly, both algorithms use the same hyperparameters, reaching an accuracy of approximately 97%. However, LR+RS+data_f requires 1 second less to complete.

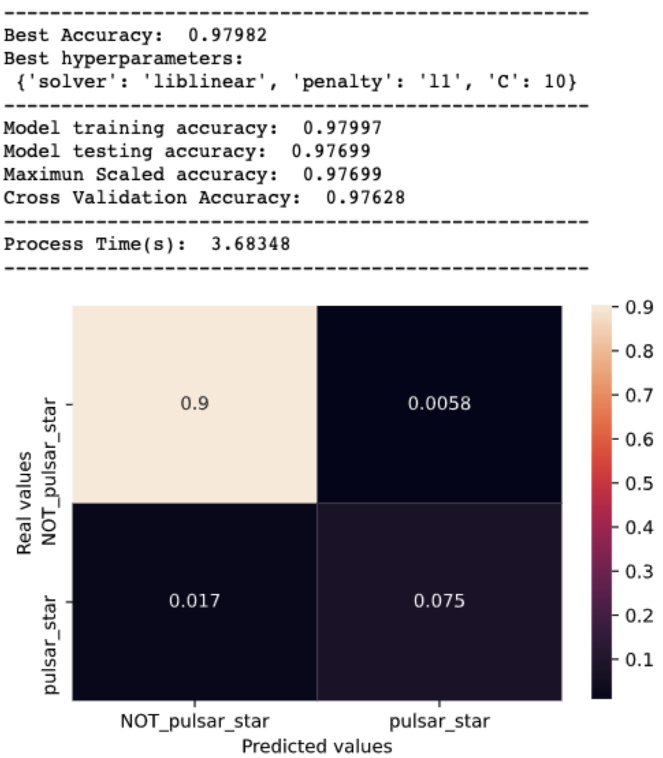

Figure 18 - Confusion Matrix of LR+RS with data

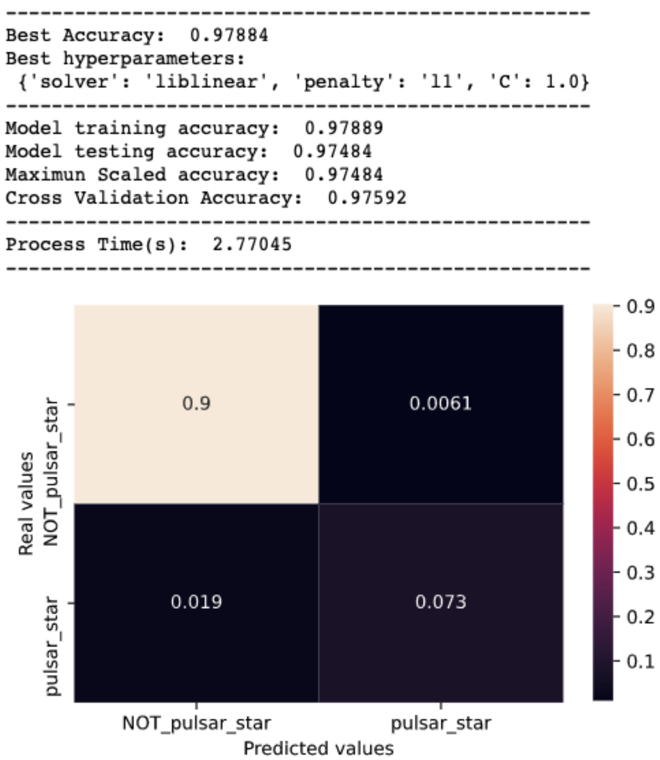

Figure 19 - Confusion Matrix of LR+RS with data_f

Table 2 agglomerates the results from LR for secondary analysis to compare GS and RS. The analysis of every technique applied to LR recognises the best accuracy is achieved with liblinear solver and with l1 penalty. As expected, the model with the least processing time applies RS, which demonstrates the accuracy varying from 2.77s to 10.16s. Algorithms without hyperparameter optimization are not considered as they suffer from overfitting.

	Best Acuracy	Best Params	Training Accuracy	Testing Acuracy	Max Scaled Acuracy	Cross-Val Acuracy	Process Time (s)
LR+data	-	-	0.97889	0.9752	0.9752	0.97556	0.20307
LR+data_f	-	-	0.97843	0.97484	0.97484	0.9752	0.07301
LR+GS+data	0.97982	{'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}	0.97997	0.97699	0.97699	0.97664	15.94848
LR+GS+data_f	0.97905	{'C': 100, 'penalty': 'l1', 'solver': 'liblinear'}	0.97936	0.97448	0.97448	0.97592	10.15665
LR+RS+data	0.97982	{'solver': 'liblinear', 'penalty': 'l1', 'C': 10}	0.97997	0.97699	0.97699	0.97628	3.68348
LR+RS+data_f	0.97884	{'solver': 'liblinear', 'penalty': 'l1', 'C': 1.0}	0.97889	0.97484	0.97484	0.97592	2.77045

Table 2 - Results from LR models

K-Nearest Neighbor Classifier (KNN)

The next figures compare the accuracies of raw data with pre-processed data, but with the KNN model instead of LG. KNN algorithms manage to surpass LG by approximately 1% when measuring the accuracies.

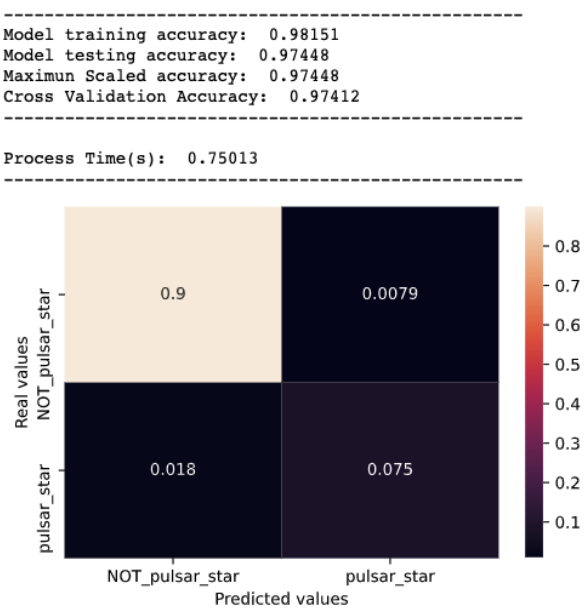

Figure 20 - Confusion Matrix of KNN with data

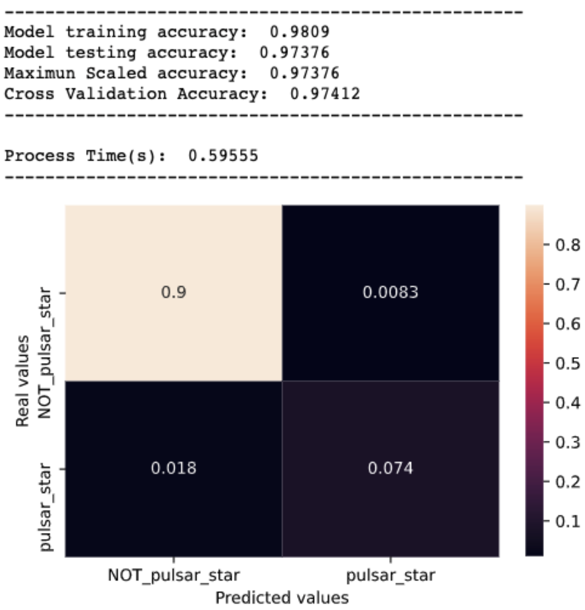

Figure 21 - Confusion Matrix of KNN with data_f

Grid Search (GS)

Unlike LR, KNN recognises the best accuracy to be accomplished by the minkowski metric. However, even though the accuracies differ by less than 1%, different hyperparameters were used. N_neighbors was set to 5 and 12 in figure 22 and 23 correspondingly. The weights parameters varied from uniformly distributed, in figure 22, to inverse their distance of weight points (distance), figure 23.

```
----------------------------------------------------
Best Accuracy:  0.97905
Best hyperparameters:
 {'metric': 'minkowski', 'n_neighbors': 5, 'weights': 'uniform'}
----------------------------------------------------
Model training accuracy:  0.98151
Model testing accuracy:  0.97448
Maximun Scaled accuracy:  0.97448
Cross Validation Accuracy:  0.97592
----------------------------------------------------
Process Time(s):  76.48149
----------------------------------------------------
```

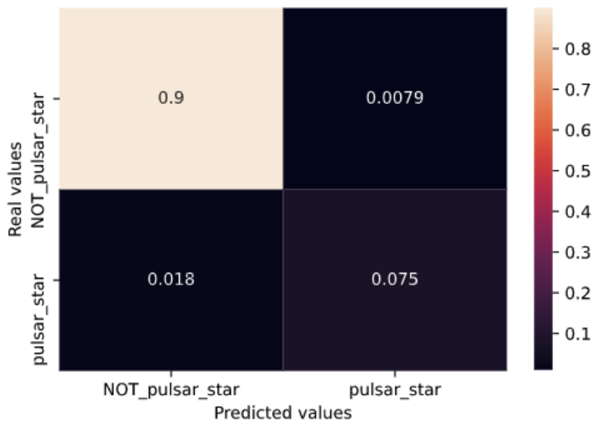

Figure 22 - Confusion Matrix of KNN+GS with data

```
----------------------------------------------------
Best Accuracy:  0.97864
Best hyperparameters:
 {'metric': 'minkowski', 'n_neighbors': 12, 'weights': 'distance'}
----------------------------------------------------
Model training accuracy:  1.0
Model testing accuracy:  0.97556
Maximun Scaled accuracy:  0.97556
Cross Validation Accuracy:  0.97592
----------------------------------------------------
Process Time(s):  71.16165
----------------------------------------------------
```

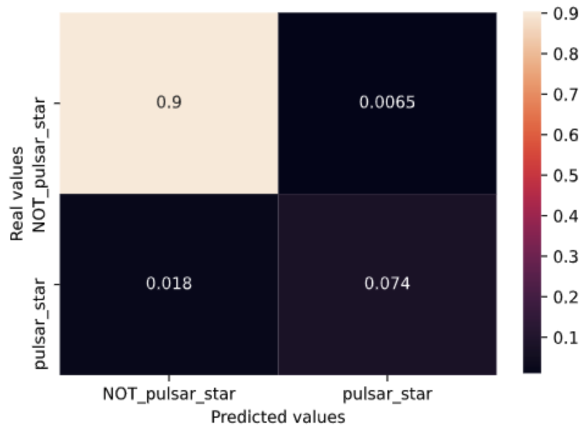

Figure 23 - Confusion Matrix of KNN+GS with data_f

Random Search (RS)

With the application of the RS technique, the lack of variations is noticeable as the only distinction is de number of n_neighbors, which varies from 13 to 15 for the best accuracy possible.

```
----------------------------------------------------
Best Accuracy:  0.97797
Best hyperparameters:
 {'weights': 'distance', 'n_neighbors': 13, 'metric': 'minkowski'}
----------------------------------------------------
Model training accuracy:  1.0
Model testing accuracy:  0.9752
Maximun Scaled accuracy:  0.9752
Cross Validation Accuracy:  0.97376
----------------------------------------------------
Process Time(s):  4.62636
----------------------------------------------------
```

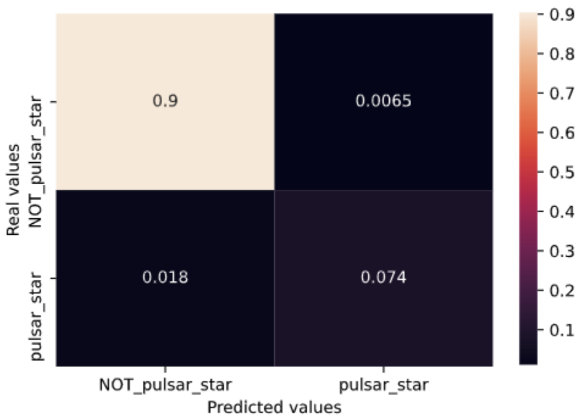

Figure 24 - Confusion Matrix of KNN+RS with data

```
----------------------------------------------------
Best Accuracy:  0.97802
Best hyperparameters:
 {'weights': 'distance', 'n_neighbors': 15, 'metric': 'minkowski'}
----------------------------------------------------
Model training accuracy:  1.0
Model testing accuracy:  0.97556
Maximun Scaled accuracy:  0.97556
Cross Validation Accuracy:  0.97592
----------------------------------------------------
Process Time(s):  4.0876
----------------------------------------------------
```

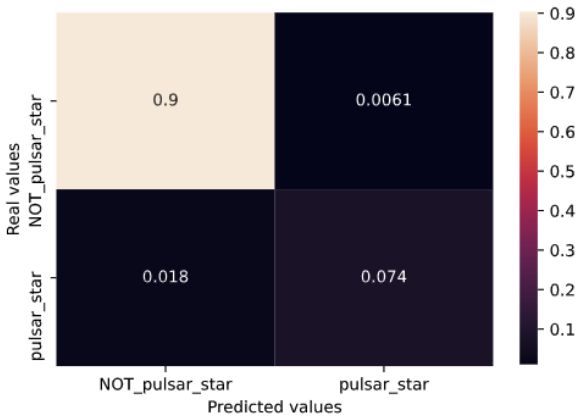

Figure 25 - Confusion Matrix of KNN+RS with data_f

Lastly, GS and RS portray a large difference in process time, with GS taking 67 additional seconds than RS. However, there is no correlation between processing time and accuracies as both obtain approximately 97% whith similar parameters.

	Best Acuracy	Best Params	Training Accuracy	Testing Acuracy	Max Scaled Acuracy	Cross-Val Acuracy	Process Time (s)
KNN+data	-	-	0.98151	0.97448	0.97448	0.97412	0.87197
KNN+data_f	-	-	0.9809	0.97376	0.97376	0.97412	0.78958
KNN+GS+data	0.97905	'metric': 'minkowski', 'n_neighbors': 5, 'weights': 'uniform'}	0.98151	0.97448	0.97448	0.97592	76.48149
KNN+GS+data_f	0.97864	{'metric': 'minkowski', 'n_neighbors': 12, 'weights': 'distance'}	1.0	0.97556	0.97556	0.97592	71.16165
KNN+RS+data	0.97797	{'weights': 'distance', 'n_neighbors': 13, 'metric': 'minkowski'}	1.0	0.9752	0.9752	0.97376	4.62636
KNN+RS+data_f	0.97802	{'weights': 'distance', 'n_neighbors': 15, 'metric': 'minkowski'}	1.0	0.97556	0.97556	0.97592	4.0876

Table 3 - Measurements of KNN models

Conclusion

Both models, LR and KNN, performed comparably as most accuracies 97%. However, the one that least performed was the KNN classifier.

Each model where hyperparameter optimization is applied, GS and RS, revealed distinctions in processing time at the expenses of accuracy. Even though there was an accuracy decline, this revealed to be minimal, of less than 1% in general. All fo the models reached an accuracy above 95%.

Thus, with this project, it is observable the influences of data missing values, feature selection, hyperparameters optimization and the learning models on the efficiency of this system.

Bibliography

Cofield, C. (2016) *What Are Pulsars?.* Available at: https://www.space.com/32661-pulsars.html (Accessed: 28 April 2021).

Naeye, R. (2007) *NASA - Neutron Stars*. Available at: https://www.nasa.gov/mission_pages/GLAST/science/neutron_stars.html (Accessed: 28 April 2021).

Baron, D. (2019) *Machine Learning in Astronomy: A Practical Overview*. Tel-Aviv University, Israel. arXiv:1904.07248

Lyon, R.J., Stappers, B. W., Cooper, S., Brooke, J. M., and Knowles, J. D. (2017) *Fifty Years of Pulsar Candidate Selection: From simple filters to a new principled real-time classification approach*, Monthly Notices of the Royal Astronomical Society 459 (1), 1104-1123, DOI: 10.1093/mnras/stw656

Chandrashekar, G. and Sahin, F. (2014) 'A Survey on Feature Selection Methods'. *Computers & Electrical Engineering* 40 (1), 16-28

Saini, R. (2019) *Feature Scaling- Why it is required?.* Available at: https://medium.com/@rahul77349/feature-scaling-why-it-is-required-8a93df1af310 (Accessed: 29 April 2021).

Appendix A

< A suggested checklist for you, for full details please refer to the coursework brief >

1. The following naming convention is used for both Coventry GitHub Repository and this report
   StudentID-Initials yes
   For example, for a student Alan Turing whose student ID was 1234567, it should be 1234567-AT
2. Coventry GitHub Repository: added to the top of this report. yes
3. Dataset(s) URL(s): added to the top of this report yes
4. Coventry OneDrive URL (if applicable): added to the top of this report
5. The GitHub Repository includes:
   - The selected dataset(s) yes
   - Source-code (.ipynb) yes
   - Demonstration video (.mp4) yes
6. Source-code added as text in Appendix B (below) yes

Appendix B

```
'''Predicting Pulsar Stars'''
#binary classification algorithm

#Load Models
import time
import numpy as np
import pandas as pd
import seaborn as sns
import missingno as msno
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA
from sklearn import neighbors, metrics
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV, cross_val_score, train_test_split, RepeatedStratifiedKFold

#shows visualization in line -> replaces plt.show
get_ipython().run_line_magic('matplotlib', 'inline')

#IMPORTING DATA
data = pd.read_csv('Datasets/pulsar_star_dataset/pulsar_data_train.csv')

'''DATA TREATMENT'''
#describes testing set shape, null values anda data info

print("--------------------------------------------------")
print("Dataset's Shape: ", data.shape)

print("-------------------------------------------------- ")
print("Null Values: ")
print(data.isna().sum())

print("-------------------------------------------------- ")
print("Data Info: ")
print(data.info())

#As all columns are relevant datapoints, none are droped
data.head()

#demonstrates Nan values within the dataset
#white strips represents Nan values in a column
msno.matrix(data)

#drops rows with Nan values
data.dropna(inplace=True)
```

```
#demonstrates the data shape to confirm the treated data has at least 1000 entrie
print("-------------------------------------------------")
print("Dataset's Shape: ", data.shape)

print("------------------------------------------------- ")
print("Null Values: ")
print(data.isna().sum())

print("------------------------------------------------- ")
print("Data Info: ")
print(data.info())

#As all columns are relevant datapoints, none are droped
data.head()

#demonstrates Nan values within the dataset
#lack of white strips represents the lack of Nan values
msno.matrix(data)

#As target_class is already binary, theres not need to transform it into categortical values
#gives general infor about the data
data.describe().T

#quantifies how many pulsar stars exist in the training set
sns.countplot(x=data['target_class'],label="pulsar_star")

#FEATURE SELECTION
#finds correlations between data
data_corr = data.corr()
data_corr.head()

#heatmap analyses the feature correlation
def heatmap(data):
  plt.figure()
  sns.heatmap(data_corr)

heatmap(data_corr)

#cheecks and eliminates one of the features that have a correlation of over .85
corr_columns = np.full((data_corr.shape[0],), True, dtype=bool)

for i in range(data_corr.shape[0]):
  for j in range(i+1, data_corr.shape[0]):
    if data_corr.iloc[i,j] >= 0.85:
      if corr_columns[j]:
        corr_columns[j] = False

selected_columns = data.columns[corr_columns]
data_f = data[selected_columns]


#To check there are no correlation between features with values over .85
```

```
data_corr = data_f.corr()
heatmap(data_corr)

#PCA PLOT
def PCA_Plot(data):

   #defining variables
   data_X = data.iloc[:,0:-1].values
   data_y = pd.DataFrame(data_f.iloc[:,-1].values, columns=['target_class'])


   #Scale X values to remove mean and improve accuracy
   X_std = StandardScaler().fit_transform(data_X)


   #PCA
   #Tripathi, A. (2019) A Complete Guide to Principal Component Analysis – PCA in Machine Learning, Data Science Duniya.
   #Available at: https://ashutoshtripathi.com/2019/07/11/a-complete-guide-to-principal-component-analysis-pca-in-machine-learning/
(Accessed: 27 April 2021).
   pca = PCA(n_components=2)
   principalComponents = pca.fit_transform(X_std)
   principalDf = pd.DataFrame(data=principalComponents , columns = ['principal component 1', 'principal component 2'])
   finalDf = pd.concat([principalDf, data_y], axis = 1)


   #PCA_Plot
   plt.figure()
   plt.xlabel('Principal component 1')
   plt.ylabel('Principal component 2')
   plt.suptitle("Pulsar Stars Prediction")
   labels = ["Not a Pulsar Star","Pulsar Star"]
   scatter = plt.scatter(data=finalDf, x="principal component 1", y="principal component 2", c="target_class",cmap='Spectral', label = labels)
   plt.legend(handles=scatter.legend_elements()[0], labels=labels)

PCA_Plot(data)


def train_test_set(data):

   #defining variables
   data_X = data.iloc[:,0:-1].values
   data_y = data.iloc[:,-1].values

   #scale dataf_X values to remove mean and improve accuracy
   #not applying scaling on y_train and y_test since their values are already 0 and 1.
   X_scaler = StandardScaler().fit_transform(data_X)

   #defining training and testing variables
   X_train, X_test, y_train, y_test = train_test_split(X_scaler, data_y, test_size=0.3, random_state=0)

   return (X_train, X_test, y_train, y_test)
```

```
#acuracy results from models
def accuracy_results(model, X_train, y_train, X_test, y_test, y_test_pred):

    #evaluate a score by cross-validation
    scores = cross_val_score(model, X_test, y_test, cv=5, scoring='accuracy')

    print("------------------------------------------------- ")
    print("Model training accuracy: ", round(model.score(X_train, y_train), 5))
    print("Model testing accuracy: ", round(model.score(X_test, y_test), 5))
    print("Maximun Scaled accuracy: ", round(accuracy_score(y_test, y_test_pred), 5))
    print("Cross Validation Accuracy: ", round(scores.mean(), 5))
    print("-------------------------------------------------")


#LEARNING MODELS

'''Logistic Regression'''
def logistic_reg(data):

    #start process time
    start_time = time.time()
    #gets vars dataf_X, dataf_y, X_train, X_test, y_train, y_test with data
    X_train, X_test, y_train, y_test = train_test_set(data)


    #training the model
    model_LR = LogisticRegression()
    model_LR = model_LR.fit(X_train, y_train)
    X_train_pred = model_LR.predict(X_train)
    y_test_pred = model_LR.predict(X_test)


    #Confusion Matrix Normalized
    conf_matrix = confusion_matrix(y_test, y_test_pred, normalize='all')

    #prints accuracies results and scores
    accuracy_results(model_LR, X_train, y_train, X_test, y_test, y_test_pred)

    #end process time
    end_time = time.time()
    print("Process Time(s): ", round(end_time-start_time, 5))
    print("------------------------------------------------- ")

    #heatmap plot of CONFUSION MATRIX
    labels = ['NOT_pulsar_star', 'pulsar_star']
    heatmap = sns.heatmap(conf_matrix, annot=True)
    heatmap.set_xticklabels(labels)
    heatmap.set_yticklabels(labels)
    heatmap.set(ylabel="Real values", xlabel="Predicted values")

#Logistic Regression with original data
logistic_reg(data)
```

```
#Logistic Regression with data with feature scalling (data_f)
logistic_reg(data_f)

#Logistic Regression with grid search
def logistic_reg_grid_search(data):

    #start process time
    start_time = time.time()

    #gets vars dataf_X, dataf_y, X_train, X_test, y_train, y_test with data
    X_train, X_test, y_train, y_test = train_test_set(data)


    #GRID SEARCH

    #Defining parameters
    #defining solvers optimises the algorithm
    solvers = ['newton-cg', 'lbfgs', 'liblinear']
    #penalises the hyperparameter
    penalty = ['l1', 'l2']
    #strengh of penalty
    c_param = [100, 10, 1.0, 0.1, 0.01, 0.001]

    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    param_GS = dict(solver=solvers, penalty=penalty, C=c_param)
    model_GS = GridSearchCV(estimator=LogisticRegression(), param_grid=param_GS, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)

    #fitting the model
    model_GS = model_GS.fit(X_train, y_train)
    X_train_pred = model_GS.predict(X_train)
    y_test_pred = model_GS.predict(X_test)

    #Confusion Matrix Normalized
    conf_matrix = confusion_matrix(y_test, y_test_pred, normalize='all')

    #prints accuracies results and scores
    print("---------------------------------------------------- ")
    print("Best Accuracy: ", round(model_GS.best_score_, 5))
    print("Best hyperparameters: ", model_GS.best_params_)
    accuracy_results(model_GS, X_train, y_train, X_test, y_test, y_test_pred)

    #end process time
    end_time = time.time()
    print("Process Time(s): ", round(end_time-start_time, 5))
    print("---------------------------------------------------- ")

    #heatmap plot of CONFUSION MATRIX
    labels = ['NOT_pulsar_star', 'pulsar_star']
    heatmap = sns.heatmap(conf_matrix, annot=True)
    heatmap.set_xticklabels(labels)
    heatmap.set_yticklabels(labels)
    heatmap.set(ylabel="Real values", xlabel="Predicted values")
```

```
#Logistic regression with original data and grid search
logistic_reg_grid_search(data)

#Logistic regression with feature selection data and grid search
logistic_reg_grid_search(data_f)

#Logistic Regression with random search
def logistic_reg_random_search(data):

    #start process time
    start_time = time.time()

    #gets vars dataf_X, dataf_y, X_train, X_test, y_train, y_test with data
    X_train, X_test, y_train, y_test = train_test_set(data)


    #RANDOM SEARCH

    #Defining parameters
    solvers = ['newton-cg', 'lbfgs', 'liblinear']
    penalty = ['l1', 'l2']
    c_param = [100, 10, 1.0, 0.1, 0.01, 0.001]

    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    param_RS = dict(solver=solvers, penalty=penalty, C=c_param)

    model_RS = RandomizedSearchCV(LogisticRegression(), param_RS, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)

    #fitting the model
    model_RS = model_RS.fit(X_train, y_train)
    X_train_pred = model_RS.predict(X_train)
    y_test_pred = model_RS.predict(X_test)

    #Confusion Matrix Normalized
    conf_matrix = confusion_matrix(y_test, y_test_pred, normalize='all')

    #prints accuracies results and scores
    print("-------------------------------------------------- ")
    print("Best Accuracy: ", round(model_RS.best_score_, 5))
    print("Best hyperparameters: ", model_RS.best_params_)
    accuracy_results(model_RS, X_train, y_train, X_test, y_test, y_test_pred)

    #end process time
    end_time = time.time()
    print("Process Time(s): ", round(end_time-start_time, 5))
    print("-------------------------------------------------- ")

    #heatmap plot of CONFUSION MATRIX
    labels = ['NOT_pulsar_star', 'pulsar_star']
    heatmap = sns.heatmap(conf_matrix, annot=True)
    heatmap.set_xticklabels(labels)
    heatmap.set_yticklabels(labels)
```

```
    heatmap.set(ylabel="Real values", xlabel="Predicted values")

#Logistic regression with original data and random search
logistic_reg_random_search(data)

#Logistic regression with feature selection data and random search
logistic_reg_random_search(data_f)

'''KNeighbours'''
def KN_Neighbors(data):

    #start process time
    start_time = time.time()

    #gets vars dataf_X, dataf_y, X_train, X_test, y_train, y_test with data
    X_train, X_test, y_train, y_test = train_test_set(data)

    #training the model
    model_KN = KNeighborsClassifier()
    model_KN = model_KN.fit(X_train, y_train)
    X_train_pred = model_KN.predict(X_train)
    y_test_pred = model_KN.predict(X_test)

    #Confusion Matrix Normalized
    conf_matrix = confusion_matrix(y_test, y_test_pred, normalize='all')

    #prints accuracies results and scores
    accuracy_results(model_KN, X_train, y_train, X_test, y_test, y_test_pred)

    #end process time
    end_time = time.time()
    print("Process Time(s): ", round(end_time-start_time, 5))
    print("-------------------------------------------------- ")

    #heatmap plot of CONFUSION MATRIX
    labels = ['NOT_pulsar_star', 'pulsar_star']
    heatmap = sns.heatmap(conf_matrix, annot=True)
    heatmap.set_xticklabels(labels)
    heatmap.set_yticklabels(labels)
    heatmap.set(ylabel="Real values", xlabel="Predicted values")

#KNNeighbors with original data
KN_Neighbors(data)

#KNNeighbors with feature selection data
KN_Neighbors(data_f)

#KNNeighbors with grid search
def KN_Neighbors_grid_search(data):

    #start process time
    start_time = time.time()
```

```
    #gets vars dataf_X, dataf_y, X_train, X_test, y_train, y_test with data
    X_train, X_test, y_train, y_test = train_test_set(data)


    #GRID SEARCH

    #Defining parameters
    n_neighbors = range(1, 31)
    #checks uniform or distance
    weights = ['uniform', 'distance']
    #metrics
    metric = ['euclidian', 'manhattan', 'minkowski']

    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    param_GS = dict(n_neighbors=n_neighbors, weights=weights, metric=metric)
    model_GS = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=param_GS, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)

    #fitting the model
    model_GS = model_GS.fit(X_train, y_train)
    X_train_pred = model_GS.predict(X_train)
    y_test_pred = model_GS.predict(X_test)

    #Confusion Matrix Normalized
    conf_matrix = confusion_matrix(y_test, y_test_pred, normalize='all')

    #prints accuracies results and scores
    print("------------------------------------------------ ")
    print("Best Accuracy: ", round(model_GS.best_score_, 5))
    print("Best hyperparameters: ", model_GS.best_params_)
    accuracy_results(model_GS, X_train, y_train, X_test, y_test, y_test_pred)

    #end process time
    end_time = time.time()
    print("Process Time(s): ", round(end_time-start_time, 5))
    print("------------------------------------------------ ")

    #heatmap plot of CONFUSION MATRIX
    labels = ['NOT_pulsar_star', 'pulsar_star']
    heatmap = sns.heatmap(conf_matrix, annot=True)
    heatmap.set_xticklabels(labels)
    heatmap.set_yticklabels(labels)
    heatmap.set(ylabel="Real values", xlabel="Predicted values")

#KNNeighbors with original data with grid search
KN_Neighbors_grid_search(data)

#KNNeighbors with feature selection data with grid search
KN_Neighbors_grid_search(data_f)

#KNNeighbors with random search
def KN_Neighbors_random_search(data):

    #start process time
```

```
    start_time = time.time()

    #gets vars dataf_X, dataf_y, X_train, X_test, y_train, y_test with data
    X_train, X_test, y_train, y_test = train_test_set(data)


    #Defining parameters
    n_neighbors = range(1, 31, 2)
    #checks uniform or distance
    weights = ['uniform', 'distance']
    #metrics
    metric = ['euclidian', 'manhattan', 'minkowski']

    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    param_RS = dict(n_neighbors=n_neighbors, weights=weights, metric=metric)
    model_RS = RandomizedSearchCV(KNeighborsClassifier(), param_RS, n_jobs=-1, cv=cv, scoring='accuracy', error_score=0)

    #fitting the model
    model_RS = model_RS.fit(X_train, y_train)
    X_train_pred = model_RS.predict(X_train)
    y_test_pred = model_RS.predict(X_test)

    #Confusion Matrix Normalized
    conf_matrix = confusion_matrix(y_test, y_test_pred, normalize='all')

    #prints accuracies results and scores
    print("-------------------------------------------------- ")
    print("Best Accuracy: ", round(model_RS.best_score_, 5))
    print("Best hyperparameters: ", model_RS.best_params_)
    accuracy_results(model_RS, X_train, y_train, X_test, y_test, y_test_pred)

    #end process time
    end_time = time.time()
    print("Process Time(s): ", round(end_time-start_time, 5))
    print("-------------------------------------------------- ")

    #heatmap plot of CONFUSION MATRIX
    labels = ['NOT_pulsar_star', 'pulsar_star']
    heatmap = sns.heatmap(conf_matrix, annot=True)
    heatmap.set_xticklabels(labels)
    heatmap.set_yticklabels(labels)
    heatmap.set(ylabel="Real values", xlabel="Predicted values")

#KNNeighbors with original data with random search
KN_Neighbors_random_search(data)

#KNNeighbors with feature selection data and random search
KN_Neighbors_random_search(data_f)
```