Pulsar Candidate Recognition using Machine Learning

Susheela Dahiya, Siddhi Patil, Punnam Satpathy, Pragyan Jyoti Dutta

Abstract

In recent years, the application of Machine Learning in the field of Astronomy has increased exponentially. Detection of Pulsar stars is a significant research topic in the field of Radio Astronomy. The enormous amount of data provided by the various Astronomical surveys makes it almost impossible to analyze the data manually, hence Machine Learning techniques are used to automate the process. In this paper, we have analyzed the pulsar star dataset from the High Time Resolution Universe - II (HTRU2) survey and classified the candidates into a pulsar or not a pulsar using ML techniques like DecisionTree, RandomClassifier, SVM,KNN, Logistic Regression and applied an ANN to it and tried to explain the mathematics behind each model.

Introduction

Pulsars belong to the broader category of neutron stars. Neutron stars are thought to be formed when a supernova occurs and are essentially the dense "dead" remains of a highly luminous and massive star that burst due to uncontrolled nucleosynthesis. A pulsar is a neutron star that rotates and has strong magnetic fields. Its magnetic poles produce electromagnetic radiation beams. The radio waves from pulsars are seen as brief bursts that occur in well-defined time intervals. Pulsars are different from other neutron stars in that they revolve and release periodic pulses of electromagnetic radiation as a result of a mismatch of their rotation and magnetic field axes. Millions of pulsar candidates have now been discovered as a result of several pulsar searches. If these enormous amounts of data are categorised manually by experts in the respective fields, it will take a lot of work.Since its conception, machine learning has become more sophisticated, and it has been successfully used to a variety of astronomical research fields, including pulsar candidate screening. The important machine learning techniques that may be employed in pulsar candidate recognition studies are discussed in this work. Techniques including Artificial Neural Networks, Support Vector Machines, and Decision Tree Classifiers are used in the approaches illustrated for usage in this study. Techniques like Synthetic Minority Oversampling Technique, or SMOTE, have been used to remedy the data imbalance. Principle Component Analysis and SMOTE have also been combined. The HTRU2 data collection, which describes a selection of pulsar candidates gathered during the High Time Resolution Universe Survey (South), is the dataset being utilised for study [1].

History

In 1934, Walter Baade and Fritz Zwicky proposed the existence of neutron stars putting forward the argument that a small, dense star consisting of primarily of neutron stars would result in a supernova. Based on the idea of magnetic flux conservation from magnetic main sequence stars, Lodewijk Woltjer proposed in 1964 that such neutron stars might contain magnetic fields as large as 10^{14} to 10^{16} gausses (=10^{10} to 10^{12} teslas). In 1967, shortly before the discovery of pulsars, Franco Pacini suggested that a rotating neutron star with a magnetic field would emit radiation, and even noted that such energy could be pumped into a supernova remnant around a neutron star, such as the Crab Nebula.

Jocelyn Bell and Anthony Hewish made the first Pulsar Star discovery while utilising the Mullard Radio Astronomy Observatory's instrument to look into radio signals coming from a fixed location in the sky. While examining data from the Radio Telescope captured on August 6, 1967, they uncovered this discovery. Bell and Hewish resolved the signals as a series of pulses spaced evenly every 1.33 seconds using a quick recorder. Due to the researchers' initial concern that the discovered signal might have come from an extraterrestrial source, it was initially known as LGM-1 ('Little Green Men'), however it was later given the name PSR B1919+21. Numerous pulsar candidates and confirmed pulsar stars were subsequently found.

Anthony Hewish was subsequently awarded the Nobel Prize for his contribution towards the discovery of pulsar stars.

The HTRU-2 Dataset

- **About the Dataset**

Legitimate pulsar examples make up a small portion of the positive class, whereas false examples make up the majority of the negative class. The data includes both genuine pulsar samples and erroneous ones brought on by RFI and/or noise. All of these instances have been reviewed by human annotators.

The variables are listed first in the row, and the class label is the last item. Class labels 0 (negative) and 1 (positive) are used . Eight continuous variables and one class variable are used to describe each candidate. Simple statistics were generated from the integrated pulse profile for the first four (folded profile). This is an array of continuous variables that represents a time- and frequency-averaged, longitude-resolved version of the signal. These are summarised below:

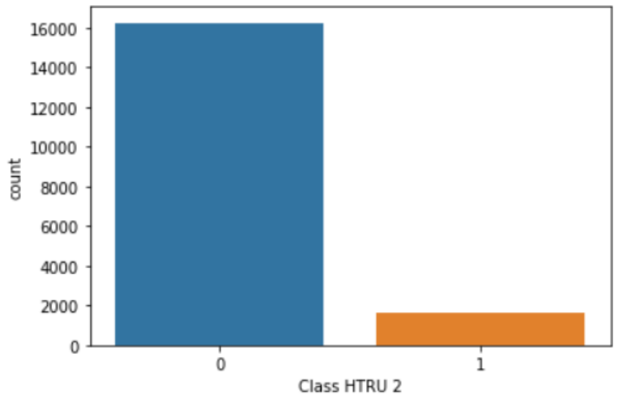
- *Mean of the integrated profile.*
- *Standard deviation of the integrated profile.*
- *Excess kurtosis of the integrated profile.*
- *Skewness of the integrated profile.*
- *Mean of the DM-SNR curve.*
- *Standard deviation of the DM-SNR curve.*
- *Excess kurtosis of the DM-SNR curve.*
- *Skewness of the DM-SNR curve.*
- *Class*

- **Challenges offered by the Dataset:**

We observe that the target class in the HTRU dataset is highly imbalanced. i.e., the occurrence of one of the classes is very high compared to the other classes present. In other words, there is a bias or skewness towards the majority class present in the target. In our case the number of negatives is much greater than the number of positives in the dataset.

```
0    90.84
1     9.16
Name: Class HTRU 2, dtype: float64
```

```
<AxesSubplot:xlabel='Class HTRU 2', ylabel='count'>
```



- **Why to overcome class imbalance?**

We got an idea of what class imbalance is in the previous section but now the question arises, why do we need to overcome this issue? Generally, most Machine Learning algorithms assume that the data is evenly distributed in each class. In such cases of class imbalance , the algorithm is more biased to predict the majority class, i.e. the 'not a pulsar star' in our case. The algorithm won't have enough data to learn about the patterns present in the minority class and that would make our model highly inaccurate.

Method 1: RandomClassifier with class_weight='balanced':

The first approach adopted by the team while trying to overcome class imbalance was to use a RandomClassifier alongwith the "class weight = 'balanced' " parameter wherein the classes are automatically weighted inversely to how frequestly they appear in the dataset. More specifically,

$$W_j = n / kn_j$$

where $\mathbf{w_j}$ is the class weight to **j** , **n** is the number of observations, $\mathbf{n_j}$ is the number of observations in class **j** and **k** is the total number of classes.

Method 2: RandomClassifier with RandomOversampler':

Random oversampling involves randomly duplicating examples from the minority class and adding them to the training dataset. Examples from the training dataset are selected randomly with replacement. This means that examples from the minority class can be chosen and added to the new "*more balanced*" training dataset multiple times; they are selected from the original training dataset, added to the new training dataset, and then returned or "*replaced*" in the original dataset, allowing them to be selected again.

This technique can be effective for those machine learning algorithms that are affected by a skewed distribution and where multiple duplicate examples for a given class can influence the fit of the model. This might include algorithms that iteratively learn coefficients, like artificial neural networks that use stochastic gradient descent. It can also affect models that seek good splits of the data, such as support vector machines and decision trees.

It might be useful to tune the target class distribution. In some cases, seeking a balanced distribution for a severely imbalanced dataset can cause affected algorithms to overfit the minority class, leading to increased generalization error. The effect can be better performance on the training dataset, but worse performance on the holdout or test dataset.

Method 3: RandomClassifier with RandomUnderSampler:

Random undersampling involves randomly selecting examples from the majority class to delete from the training dataset.

This has the effect of reducing the number of examples in the majority class in the transformed version of the training dataset. This process can be repeated until the desired class distribution is achieved, such as an equal number of examples for each class.

This approach may be more suitable for those datasets where there is a class imbalance although a sufficient number of examples in the minority class, such a useful model can be fit.

A limitation of undersampling is that examples from the majority class are deleted that may be useful, important, or perhaps critical to fitting a robust decision boundary. Given that examples are deleted randomly, there is no way to detect or preserve "good" or more information-rich examples from the majority class.

Method 4:Synthetic Minority Oversampling Technique(SMOTE):

Machine learning models typically struggle to learn and generalize successfully if they are trained on unbalanced data, which is data where one or more classes is disproportionately represented (usually under-represented). As a result, we must take action to overcome this; often, this entails modifying the learning data.

One way to balance out uneven datasets is to oversample the minority group. Even though these examples don't provide any new understanding to the model, the simplest approach is cloning instances from the minority class. Instead, new ones can be made by using the existing ones as a guide. Data augmentation for the minority class is known as the Synthetic Minority Oversampling Technique, or SMOTE. In their 2002 study titled "SMOTE: Synthetic Minority Over-sampling Technique," Nitesh Chawla, et al., described this technique in their paper, "SMOTE: Synthetic Minority Over-sampling Technique."

SMOTE selects examples in the feature space that are near to one another, draws a line connecting the examples, and then creates a new sample at a location along the line.

To be more precise, a random representative from the minority class is initially picked. Next, k nearest neighbors for that example are located (k is normally equal to 5). A synthetic example is made at a randomly chosen position in feature space between two instances and their randomly chosen neighbor.

The strategy works because it generates convincing new synthetic instances from the minority class that are substantially near in feature space to already existing examples from the minority class.

The downside of this approach is that synthetic examples are constructed without taking into account the majority class, which might lead to misleading examples if there is a significant overlap between the classes.

Techniques Used

- ### Performance Matrix

In our report we have evaluated using the assessment metrics like Accuracy and area under the curve. Due to the highly imbalanced data, accuracy may not be the most apt implementation and hence precision and recall based on the confusion matrix are further used for the same.

- **Accuracy = (TN+TP)/(TP+FP+TN+FN)**
- **Precision = TP/(TP+FP)**
- **Recall = TP/(TP+FN)**
- **f1 score = 2*(Precision*Recall)/(Precision + Recall)**

where, **TN = True Negatives, TP = True Positives, FN = False Negatives, FP = False Positives**

Ideally Accuracy and Precision should be close to one while f1 score becomes higher for a better model and takes both precision and recall into consideration**.**

In this paper, we proposed a combined solution to classify imbalanced data, which successfully reduces dimensionality, and balances the minority class using a combination of Principle Component Analysis (PCA) and Synthetic Minority Oversampling Techniques. Precision and Recall should both be closer to 1 for a good model.

- ### RandomForestClassifier

A random forest is a meta estimator that employs averaging to increase predicted accuracy and reduce overfitting after fitting numerous decision tree classifiers to distinct dataset subsamples.

Numerous distinct decision trees are built by random forests (RF) during training. To determine the final prediction—the mean prediction for regression or the mode of the classes for classification—predictions from all trees are combined. They are known as ensemble approaches because they combine results to reach a decision.

- ### KNearestNeighbors

KnearestNeighbors also known as KNN or k-NN is a non-parametric supervised learning classifier, which means that it does not makin assumptions on the underlying data, and uses proximity to make predictions or classifications about grouping of an individual data point

The goal of the k-nearest neighbor algorithm is to identify the nearest neighbors of a given query point, so that we can assign a class label to that point.

- ### LogisticRegression

Logistic regression is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set. A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables.

- ### Support Vector Machine

SVM acts on data by creating hyperplanes such that accuracy and robustness can be maximized. This is carried out using different Kernels like rbf, polynomial, sigmoid etc. Usually rbf can be employed for datasets whose distribution is completely unknown while the polynomial might be more successful for known datasets. The C value or the distance between 2 outliers minimized, is also incorporated within the algorithm to include variance and reduce error %.

- ### Artificial Neural Network

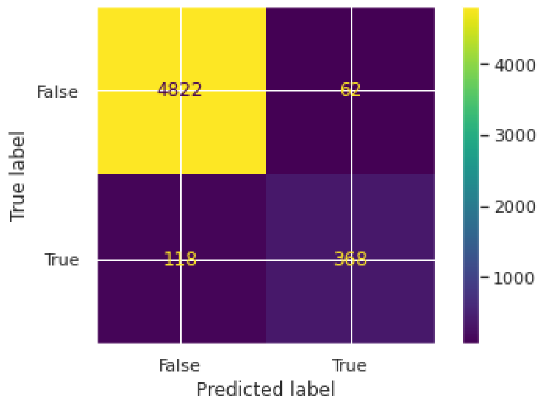
A node layer of an artificial neural network (ANN) consists of an input layer, one or more hidden layers, and an output layer. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer. Otherwise, no data is sent to the network's next tier. Each each node has its own linear regression model, comprising of input data, weights, a bias (or threshold), and an output.

Observations

- **RandomClassifer:**

Applying RandomClassifier on our dataset alongwith SMOTE gave us a model accuracy of 97% and recall of 76% while predicting positives and a precision of 86%. Our model was able to predict 368 True Positives, 4822 True Negatives, 62 False negatives along with 118 False positives.

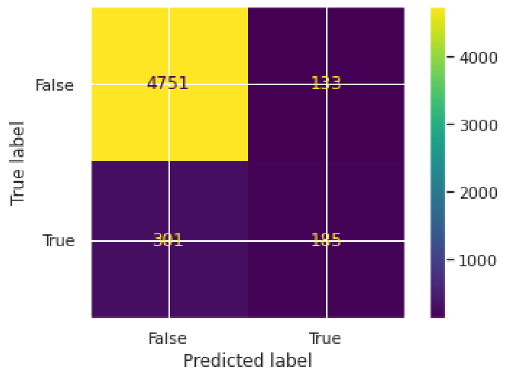
```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      4884
           1       0.86      0.76      0.80       486

    accuracy                           0.97      5370
   macro avg       0.92      0.87      0.89      5370
weighted avg       0.97      0.97      0.97      5370
```



- **Logistic Regression:**

Applying Logistic Regression on our dataset gave us a model accuracy of 92% and recall of 38% while predicting positives and a precision of 58%. Our model was able to predict 185 True Positives, 4751 True Negatives, 133 False negatives along with 301 False positives.

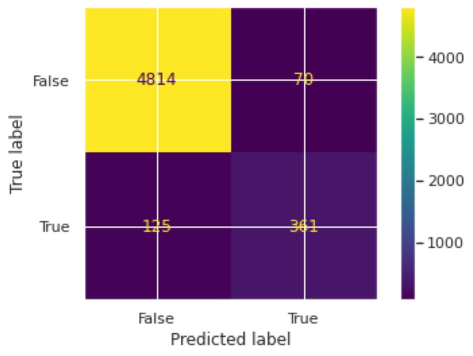
```
              precision    recall  f1-score   support

           0       0.94      0.97      0.96      4884
           1       0.58      0.38      0.46       486

    accuracy                           0.92      5370
   macro avg       0.76      0.68      0.71      5370
weighted avg       0.91      0.92      0.91      5370
```



- **K Nearest Neighbors:**

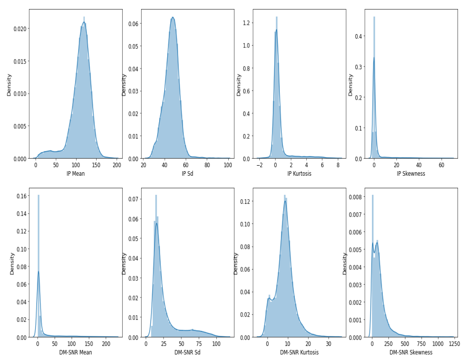
Applying Logistic Regression on our dataset gave us a model accuracy of 96% and recall of 74% while predicting positives and a precision of 84%. Our model was able to predict 361 True Positives, 4814 True Negatives, 125 False negatives along with 70 False positives.

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      4884
           1       0.84      0.74      0.79       486

    accuracy                           0.96      5370
   macro avg       0.91      0.86      0.88      5370
weighted avg       0.96      0.96      0.96      5370
```

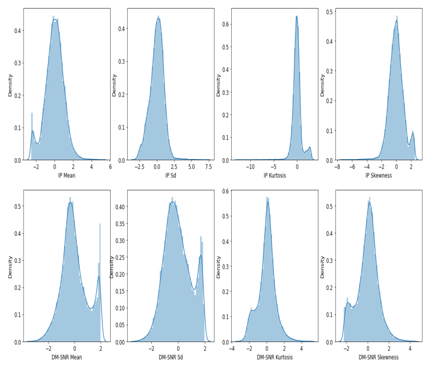


- **SVM:**

We applied this kernel to the raw data as well as applied gaussian function by applying the yeo-johnson power transform. The linear kernel is a specialized polynomial kernel of the order one. The raw data as seen is skewed both positively and negatively. Yeo – Johnson is an essential power transform method that acts by inflating the low variance data and deflating the high variance.

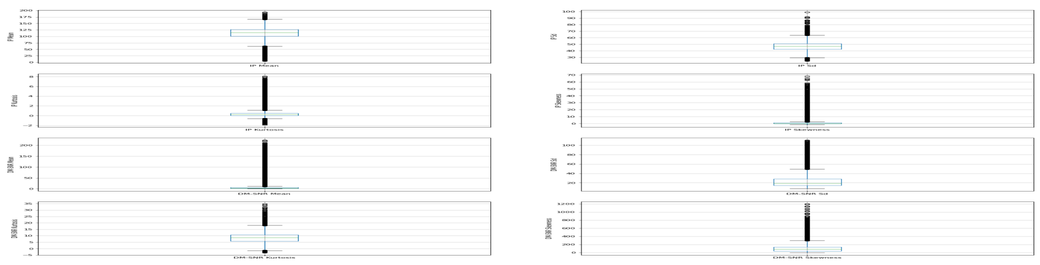


Original data distribution



Gaussian Distribution

Converting the data to Gaussian essentially makes it easier for machine learning strategies to assume the cost function to be a sum of the independent variables or the input features which may have an intrinsically gaussian property. The 'C' value is increased for 0 to 1000 for each kernel. Generally increasing the C value suggests more outliers being taken into consideration if they are located at a larger distance. Based on the graph below, the outliers are spread out pretty unevenly thus suggesting that we should probably see a better value performance at higher values of c.



The figure below is SVM applied over the original dataset to compare between the SMOTE and PCA performance as well. As noticed by the accuracy percentage, the values between c=1 and c=100 were the maximized accuracy with values either decreasing or stagnating after c=100. This is possibly due to the high imbalance in the data which creates biased skewness with high variance for every test run.

c=[0,100,1000]	**Original Dataset**			**Gaussian Dataset**		
	rbf	**Linear**	**Polynomial**	**rbf**	**Linear**	**Polynomial**
c=1	0.9827	0.9830	0.9807	0.9838	0.9818	0.9841
c=100	0.9827	0.885	0.9824	0.9841	0.9819	0.9841
c=1000	0.9816	0.885	0.9807	0.9841	0.9802	0.9826

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3306
           1       0.93      0.84      0.88       274

    accuracy                           0.98      3580
   macro avg       0.96      0.92      0.94      3580
weighted avg       0.98      0.98      0.98      3580

Classification accuracy : 0.9830
Classification error : 0.0170
Precision : 0.9949
Recall or Sensitivity : 0.9868
False Positive Rate : 0.0688
Specificity : 0.9312
```

This further suggests that the best kernel for the HTRU dataset is the Polynomial after the transformation with an improvement of 0.002 from the non gaussian data rbf kernel.

- **Artificial Neural Networks (ANN)**
  - **Feature Engineering and ANN**

Data Manipulation

When we visualized the data, we saw that two columns :- "mean_ip_squared", "mean_ip_squared" seemed to have a distribution that was almost normal. In order to alter the values, we choose to square them.

```python
# Squaring the `mean_ip` column

pulsar["Mean IP squared"] = pulsar["Mean of the integrated profile"].apply(lambda x: x**2)

# Squaring the `sd_ip` column

pulsar["SD IP squared"]   = pulsar["Standard deviation of the integrated profile"].apply(lambda x: x**2)
```

	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	Class HTRU 2	Mean IP squared	SD IP squared
Mean of the integrated profile	1.000000	0.547137	-0.873898	-0.738775	-0.298841	-0.307016	0.234331	0.144033	-0.673181	0.972187	0.494977
Standard deviation of the integrated profile	0.547137	1.000000	-0.521435	-0.539793	0.006869	-0.047632	0.029429	0.027691	-0.363708	0.519775	0.990401
Excess kurtosis of the integrated profile	-0.873898	-0.521435	1.000000	0.945729	0.414368	0.432880	-0.341209	-0.214491	0.791591	-0.757326	-0.460159
Skewness of the integrated profile	-0.738775	-0.539793	0.945729	1.000000	0.412056	0.415140	-0.328843	-0.204782	0.709528	-0.588606	-0.474847
Mean of the DM-SNR curve	-0.298841	0.006869	0.414368	0.412056	1.000000	0.796555	-0.615971	-0.354269	0.400876	-0.191857	0.051190
Standard deviation of the DM-SNR curve	-0.307016	-0.047632	0.432880	0.415140	0.796555	1.000000	-0.809786	-0.575800	0.491535	-0.214294	-0.009830
Excess kurtosis of the DM-SNR curve	0.234331	0.029429	-0.341209	-0.328843	-0.615971	-0.809786	1.000000	0.923743	-0.390816	0.157530	-0.001059
Skewness of the DM-SNR curve	0.144033	0.027691	-0.214491	-0.204782	-0.354269	-0.575800	0.923743	1.000000	-0.259117	0.097513	0.010278
Class HTRU 2	-0.673181	-0.363708	0.791591	0.709528	0.400876	0.491535	-0.390816	-0.259117	1.000000	-0.571943	-0.318202
Mean IP squared	0.972187	0.519775	-0.757326	-0.588606	-0.191857	-0.214294	0.157530	0.097513	-0.571943	1.000000	0.475585
SD IP squared	0.494977	0.990401	-0.460159	-0.474847	0.051190	-0.009830	-0.001059	0.010278	-0.318202	0.475585	1.000000

We choose to divide the characteristics into two subgroups rather than having a single set.

- original features
- squared features

```
# Defining the `mean_ip` * `sd_ip` colum
# Defining the `ex_kurt_ip` * `skew_ip` colum

pulsar["Mean * SD IP"] = pulsar["Mean of the integrated profile"] * pulsar["Standard deviation of the integrated profile"]
pulsar["Excess kurtosis * Skewness IP"] = pulsar["Excess kurtosis of the integrated profile"] * pulsar["Skewness of the integrated profile"]

# Defining the `mean_dmsnr` * `sd_dmsnr` colum
# Defining the `ex_kurt_dmsnr` * `skew_dmsnr` colum

pulsar["Mean * SD DM-SNR"] = pulsar["Mean of the DM-SNR curve"] * pulsar["Standard deviation of the DM-SNR curve"]
pulsar["Excess kurtosis * Skewness DM-SNR"] = pulsar["Excess kurtosis of the DM-SNR curve"] * pulsar["Skewness of the DM-SNR curve"]
```

```
Index(['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',
       'Class HTRU 2', 'Mean IP squared', 'SD IP squared', 'Mean * SD IP',
       'Excess kurtosis * Skewness IP', 'Mean * SD DM-SNR',
       'Excess kurtosis * Skewness DM-SNR'],
      dtype='object')
```

We came to the conclusion that the best method for forecasting a pulsar is a feed forward neural network due to the high imbalance of the classes.

We determined that we are better off with two subsets: the original features and original features with squared columns. Before defining our X and Y variables, we must first specify the subsets.

```
# List of the original features

original_features    = ['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',
                        'Class HTRU 2']

# List of the original features with `mean_ip` and `sd_ip` squared

manipulated_features = ['Mean IP squared',
                        'SD IP squared',
                        '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',
                        'Class HTRU 2']
```

Because each characteristic is on a distinct scale, we will scale our data. Additionally, if all features are scaled, it will be simpler for the neural network to locate global minima.

The train-test split is used to divide up the data so that we may set aside an unobserved piece of it for model testing. Additionally, we will stratify on y to maintain the distribution of classes and establish a random state for repeatability.

```
# For the original data

X_train, X_test, y_train, y_test = train_test_split(X_ss,
                                                    y,
                                                    random_state = 42,
                                                    stratify     = y)

# For the original data with squared features

X_sq_train, X_sq_test, y_sq_train, y_sq_test = train_test_split(X_sq_ss,
                                                                y_sq,
                                                                random_state = 42,
                                                                stratify     = y_sq )
```

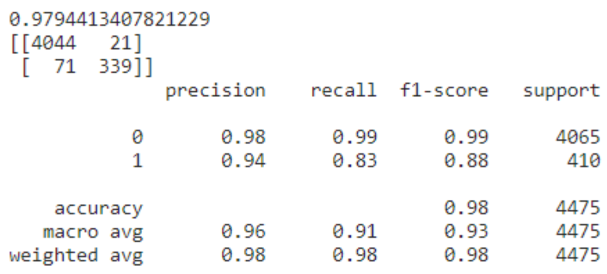
- **Metrics**

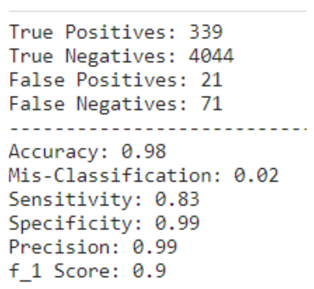
A confusion matrix contrasts the real and predicted y values so that we can understand how the model did on each class, and it provides us with a summary of how our model categorised the test data. The configuration of each confusion matrix is the same:

	Predicted Negative	Predicted Positive
Actual Negative	True Negative	False Positive
Actual Positive	False Negative	True Positive

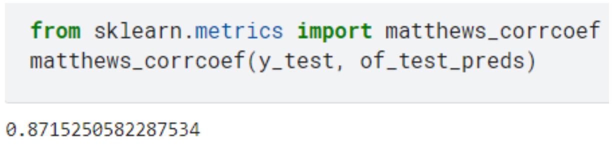
We will create two scores, specificity and sensitivity, based on the confusion matrix; we will discuss what these values represent.
Since accuracy just counts the number of entries that were successfully categorised, it is not the most useful result for us. Instead, we want to assess how well the projected positives (pulsars) and negatives performed (non-pulsars). The following scores will be used:

Metric	Definition	Scale
Accuracy	The number of correctly made predictions	0 to 1
Specificity	How many negative predictions are correct	0 to 1
Sensitivity	How many positive predictions are correct (also known as recall)	0 to 1
ROC-AUC Score	A measure of the model's ability to distinguish classes	0.5 to 1
Matthew Correlation Coefficient	A measure of how correlated the results and true values are	-1 to 1


```
0.9794413407821229
[[4044   21]
 [  71  339]]
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      4065
           1       0.94      0.83      0.88       410

    accuracy                           0.98      4475
   macro avg       0.96      0.91      0.93      4475
weighted avg       0.98      0.98      0.98      4475
```

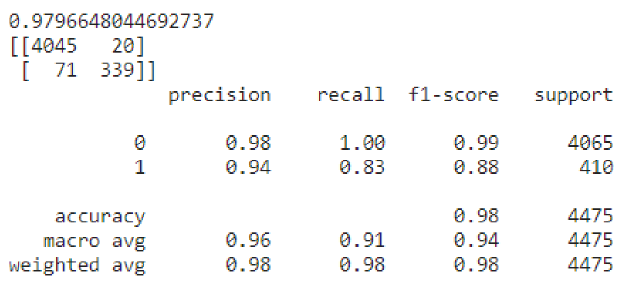

```
True Positives: 339
True Negatives: 4044
False Positives: 21
False Negatives: 71
--------------------------
Accuracy: 0.98
Mis-Classification: 0.02
Sensitivity: 0.83
Specificity: 0.99
Precision: 0.99
f_1 Score: 0.9
```

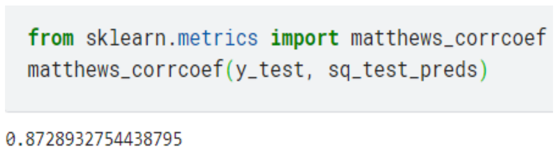
	Predicted Negative	Predicted Positive
Actual Negative	4044	21
Actual Positive	71	339


```
from sklearn.metrics import matthews_corrcoef
matthews_corrcoef(y_test, of_test_preds)
```
```
0.8715250582287534
```

Since overfitting is a very serious issue with neural networks, we were worried about the model. The model was often very little underfit and there was seldom any overfitting. Overall, despite the fact that the sensitivity was very poor due to the small number of pulsars in the data, we were extremely happy with our results.

Given that the negative class accounts for 91% of the data, it is not surprising that the model performed so well with genuine negatives. The false negatives were also quite low, which is crucial because these stars are really significant and we want to have as few false negatives as possible. The model also performed well with true positives.


```
0.9796648044692737
[[4045   20]
 [  71  339]]
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      4065
           1       0.94      0.83      0.88       410

    accuracy                           0.98      4475
   macro avg       0.96      0.91      0.94      4475
weighted avg       0.98      0.98      0.98      4475
```


```
from sklearn.metrics import matthews_corrcoef
matthews_corrcoef(y_test, sq_test_preds)
```
```
0.8728932754438795
```

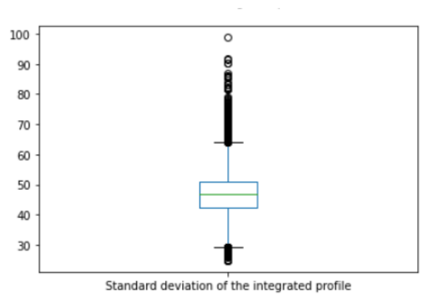
```
True Positives: 339
True Negatives: 4045
False Positives: 20
False Negatives: 71
--------------------------
Accuracy: 0.98
Mis-Classification: 0.02
Sensitivity: 0.83
Specificity: 1.0
Precision: 1.0
f_1 Score: 0.9
```

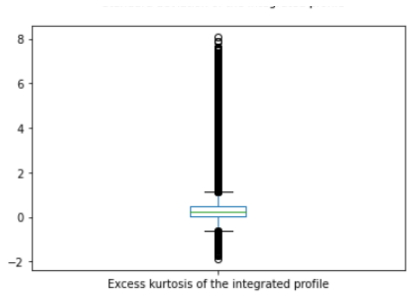
	Predicted Negative	Predicted Positive
Actual Negative	4045	20
Actual Positive	71	339

These results are marginally better than the metrics for the original features; squaring the two features only marginally enhanced the model.

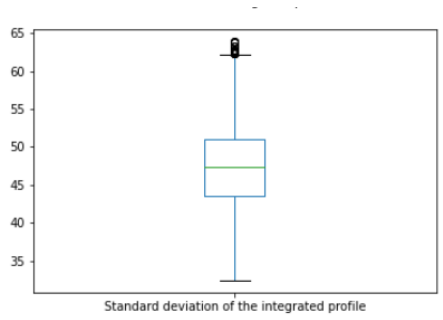
The matrix in this model was almost identical to the previous one, except it has one more true negative and one less false positive.

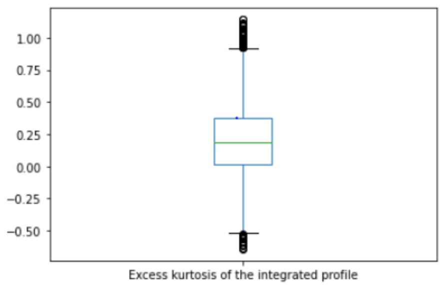
- **Outlier Correction and Imbalanced Data**





```python
def outliers(df,ft):
    Q1 = df[ft].quantile(0.25)
    Q3 = df[ft].quantile(0.75)
    IQR = Q3 - Q1

    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    ls = df.index[(df[ft] < lower_bound)| (df[ft] > upper_bound)]

    return ls
```





```python
round(pulsar["Class HTRU 2"].value_counts(normalize = True)*100, 2)
```

```
0    90.84
1     9.16
Name: Class HTRU 2, dtype: float64
```

```python
round(df_cleaned["Class HTRU 2"].value_counts(normalize = True)*100, 2)
```

```
0    98.88
1     1.12
Name: Class HTRU 2, dtype: float64
```

```
df_cleaned.shape
```

```
(13320, 9)
```

```
pulsar.columns
pulsar.shape
```

```
(17898, 9)
```

Outlier Correction lead to more imbalance in the dataset

- **SMOTE with Logistic Regression**

	Predicted Non-Pulsar	Predicted Pulsar
Actual Non-Pulsar	3165	87
Actual Pulsar	29	299

- **ANN with Layers**

```
model = Sequential([
    Dense(units=20, input_dim = X_train.shape[1], activation='relu'),
    Dense(units=24,activation='relu'),
    Dropout(0.5),
    Dense(units=20,activation='relu'),
    Dense(units=24,activation='relu'),
    Dense(1, activation='sigmoid')
])
model.summary()
```

- **SMOTE with ANN**

```
0    16259
1    16259
Name: Class HTRU 2, dtype: int64
```

```
scoreNew = model.evaluate(X, y)
print('Test Accuracy: {:.2f}%\nTest Loss: {}'.format(scoreNew[1]*100,scoreNew[0]))
```

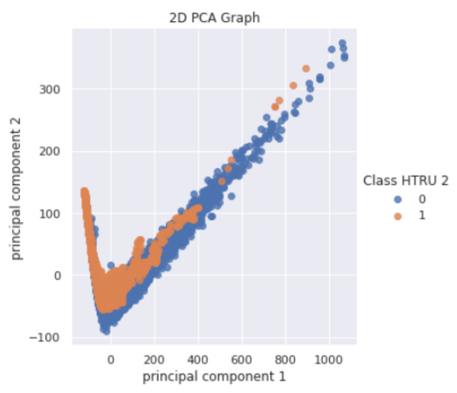
```
560/560 [==============================] - 1s 2ms/step - loss: 0.1559 - accuracy: 0.9614
Test Accuracy: 96.14%
Test Loss: 0.15585562586784363
```

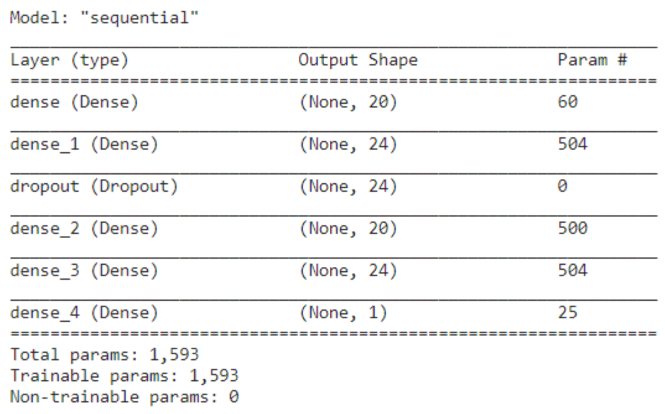
```
print(classification_report(y_test2, y_pred2.round()))
```

```
              precision    recall  f1-score   support

           0       0.99      0.97      0.98     16259
           1       0.74      0.89      0.81      1639

    accuracy                           0.96     17898
   macro avg       0.86      0.93      0.89     17898
weighted avg       0.97      0.96      0.96     17898
```

SMOTE -> PCA -> ANN

	principal component 1	principal component 2	Class HTRU 2
0	30.415686	-52.502637	0
1	72.904940	-13.690186	0
2	11.115845	-32.993010	0
3	10.003232	-57.136322	0
4	186.317816	40.018868	0





```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 20)                60
_________________________________________________________________
dense_1 (Dense)              (None, 24)                504
_________________________________________________________________
dropout (Dropout)            (None, 24)                0
_________________________________________________________________
dense_2 (Dense)              (None, 20)                500
_________________________________________________________________
dense_3 (Dense)              (None, 24)                504
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 25
=================================================================
Total params: 1,593
Trainable params: 1,593
Non-trainable params: 0
```

Accuracy	**Specificity**	**Sensitivity**	**Matthews Corr. Coef.**	**Accuracy**
Original Features	0.99	0.83	0.8715250582287534	0.979441341
Squared Features	1.0	0.83	0.8728932754438795	0.979664804
SMOTE with logistic	0.97	0.91	0.8230234248976278	0.967597765
ANN with layers	0.98	0.92	0.8389804799847553	0.97150838
SMOTE with ANN 24	0.97	0.89	0.7920796266752241	0.961448207
SMOTE with ANN 12	0.97	0.88	0.8502182677845082	0.923226732
SMOTE PCA ANN	0.81	0.94	0.754456473739111	0.874128741

Conclusions

- **PCA before SMOTE**

[0,1]	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**	**KNN**	**ANN**
Accuracy	0.95	0.97	0.96	0.92	0.96	0.92
Precision	0.97, 0.71	0.98, 0.86	0.97, 0.84	0.94, 0.58	0.97, 0.84	-
Recall	0.97, 0.71	0.99, 0.76	0.99, 0.73	0.97, 0.38	0.99, 0.74	-

- **SMOTE before PCA**

[0,1]	Decision Tree	Random Forest	SVM	Logistic Regression	KNN	ANN
Accuracy	0.90	0.93	0.90	0.90	0.92	0.90
Precision	0.90, 0.89	0.92, 0.93	0.89, 0.92	0.87, 0.93	0.92, 0.92	-
Recall	0.89, 0.90	0.93, 0.92	0.92, 0.89	0.93, 0.86	0.92, 0.92	-

Based on our research we found Random forest to be the best performer based on accuracy. The models were applied to the dataset treated with SMOTE and then PCA. The next best models would be KNN and SVM.

However, the models performed the best in context of precision and recall when we applied Random forest after the dataset was treated with SMOTE and the PCA.

References

- R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, J. D. Knowles, 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

  R. J. Lyon, HTRU2, DOI: 10.6084/m9.figshare.3080389.v1.

- Wang, Y., Pan, Z., Zheng, J. *et al.* A hybrid ensemble method for pulsar candidate classification. *Astrophys Space Sci* 364, 139 (2019). https://doi.org/10.1007/s10509-019-3602-4
- Predicting pulsar stars using a random tree boosting voting classifier (RTB-VC) F.
  F. Rustam, A. Mehmood, S. Ullah, M. Ahmad, D. Muhammad Khan, G.S. Choi, B.-W. On,
  Predicting pulsar stars using a random tree boosting voting classifier (RTB-VC),
  Astronomy and Computing,
  Volume 32,
  2020,
  100404,
  ISSN 2213-1337,
  https://doi.org/10.1016/j.ascom.2020.100404.
  (https://www.sciencedirect.com/science/article/pii/S2213133720300585)
- Patel A, Taghavi M, Bakhtiyari K, JúNior JC (2013) An intrusion detection and prevention system in cloud computing: a systematic review. J Netw Comput Appl 36:25–41

- De la Hoz E, De La Hoz E, Ortiz A, Ortega J, Prieto B (2015) PCA filtering and probabilistic SOM for network intrusion detection. Neurocomputing 164:71–81

- Hosseinzadeh, M., Rahmani, A.M., Vo, B. *et al.* Improving security using SVM-based anomaly detection: issues and challenges. *Soft Comput* 25, 3195–3223 (2021). https://doi.org/10.1007/s00500-020-05373-x

- V. Cherkassky and Y. Ma, “Practical selection of SVM parameters and noise estimation for SVM regression,” Neural Netw., vol. 17, no. 1, pp. 113–126, 2004.

- G. Mountrakis, J. Im, and C. Ogole, “Support vector machines in remote sensing: A review,” ISPRS J. Photogramm. Remote Sens., vol. 66, no. 3, pp. 247–259, 2011.

- Naseriparsa, Mehdi & Riahi Kashani, Mohammad. (2014). Combination of PCA with SMOTE Resampling to Boost the Prediction Rate in Lung Cancer Dataset. International Journal of Computer Applications. 77. 10.5120/13376-0987.
- Braytee, A., Liu, W., Kennedy, P. (2016). A Cost-Sensitive Learning Strategy for Feature Extraction from Imbalanced Data. In: Hirose, A., Ozawa, S., Doya, K., Ikeda, K., Lee, M., Liu, D. (eds) Neural Information Processing. ICONIP 2016. Lecture Notes in Computer Science(), vol 9949. Springer, Cham. https://doi.org/10.1007/978-3-319-46675-0_9