Phase wise report on Pulsar candidate predictions

Prathyusha Velupula

November 29, 2023

Contents

List of Figures

1 PHASE 1 :Data analysis and preparation

The dataset for this project is chosen from here. HTRU2 is a data set which describes a sample of pulsar candidates collected during the High Time Resolution Universe Survey (South). The data set shared here contains 16,259 spurious examples caused by RFI/noise, and 1,639 real pulsar examples. These examples have all been checked by human annotators. Each candidate is described by 8 continuous variables. The first four are simple statistics obtained from the integrated pulse profile (folded profile). This is an array of continuous variables that describe a longitude-resolved version of the signal that has been averaged in both time and frequency. The remaining four variables are similarly obtained from the DM-SNR curve . These are summarised below:

1. Mean of the integrated profile.
2. Standard deviation of the integrated profile.
3. Excess kurtosis of the integrated profile.
4. Skewness of the integrated profile.
5. Mean of the DM-SNR curve.
6. Standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.
8. Skewness of the DM-SNR curve.

HTRU 2 Summary: 17,898 total examples ,1,639 positive examples and 16,259 negative examples.

The data is presented in two formats: CSV and ARFF (used by the WEKA data mining tool). Candidates are stored in both files in separate rows. Each row lists the variables first, and the class label is the final entry. The class labels used are 0 (negative) and 1 (positive).

Please not that the data contains no positional information or other astronomical details. It is simply feature data extracted from candidate files using the PulsarFeatureLab tool

1.1 Cleaning of data

The dataset downloaded is cleaned and updated using methods given here. The updated csv file is then loaded as a pandas dataframe in the jupyter notebook.

1.2 Processing and analysing of data

1.2.1 Visualization of distribution of input varibales

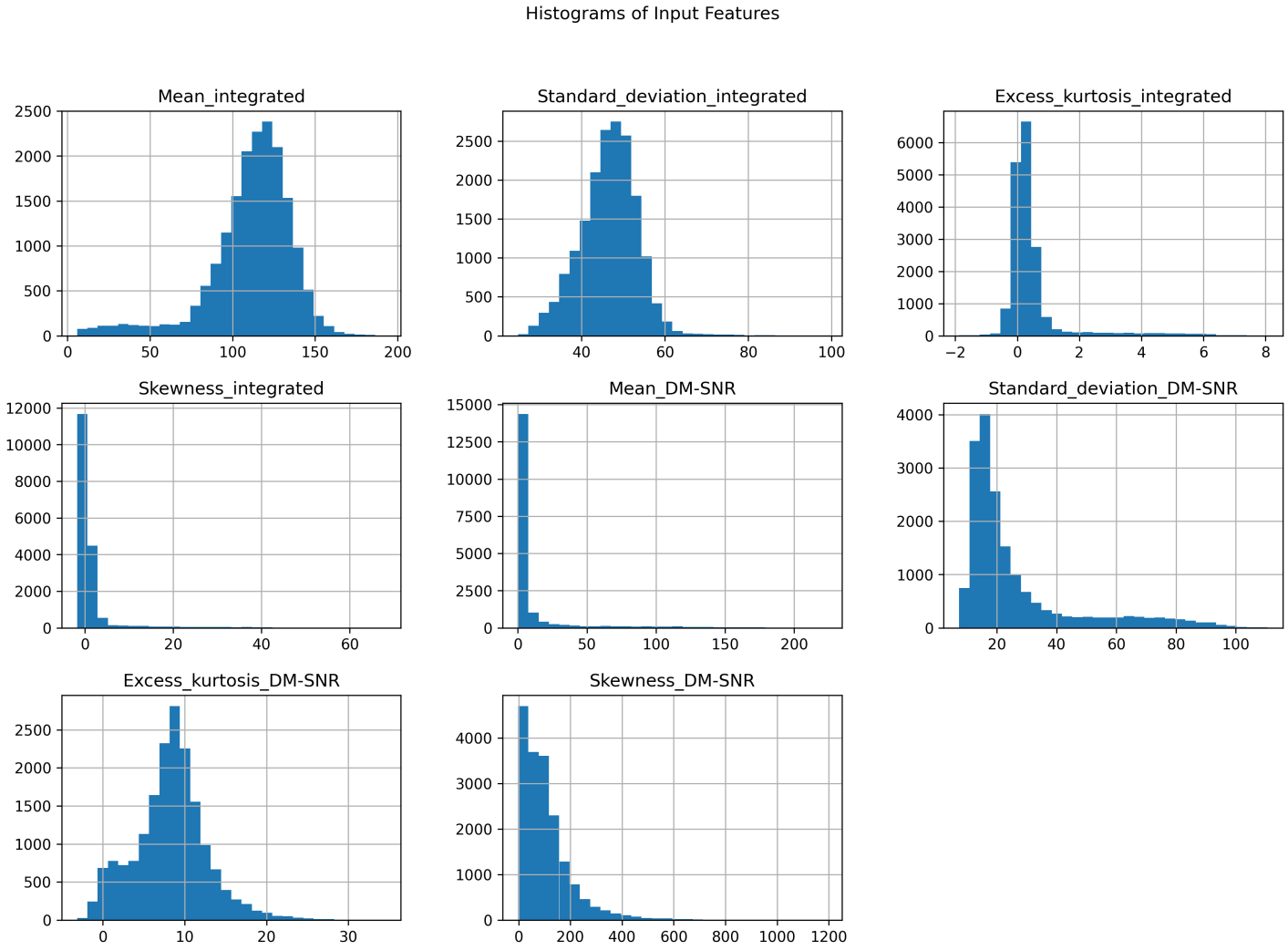

Figure 1: Histograms of Distribution of input features - before normalization

1.2.2 Distribution of Output label

Below figure shows how class feature is distributed (see **Figure 2**).

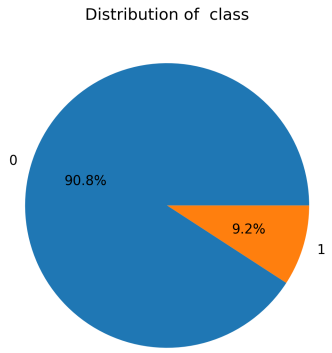

Figure 2: Pie diagram of Distribution of class label

1.2.3 Data after normalization

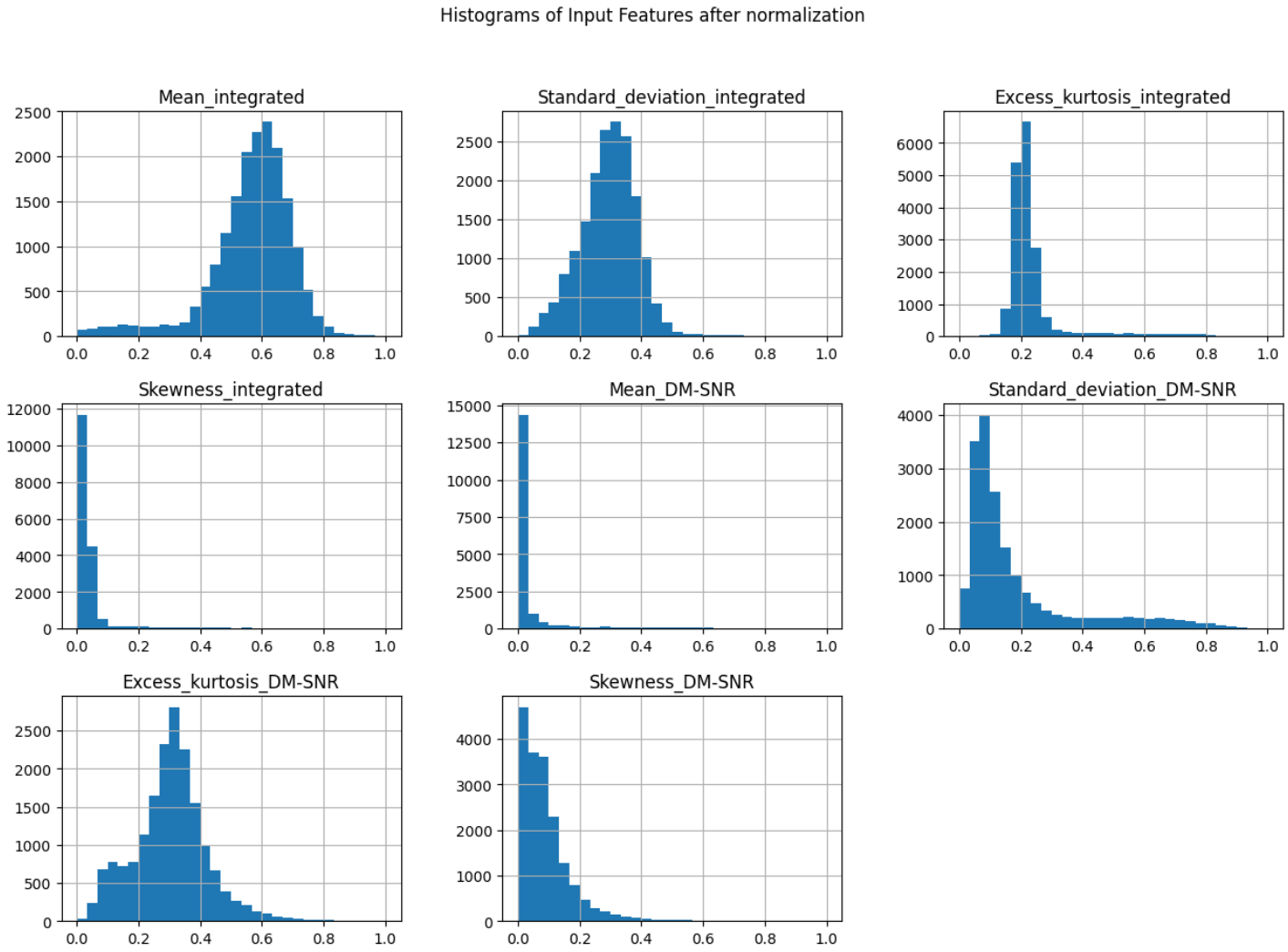

Figure 3: Histograms of Distribution of input features - after normalization

1.2.4 First 10 rows of data before and after normalization

Mean_integrated	Standard_deviation_integrated	Excess_kurtosis_integrated	Skewness_integrated	Mean_DM-SNR	Standard_deviation_DM-SNR	Excess_kurtosis_DM-SNR	Skewness_DM-SNR	Class
140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0
93.570312	46.698114	0.531905	0.416721	1.636288	14.545074	10.621748	131.394004	0
119.484375	48.765059	0.031460	-0.112168	0.999164	9.279612	19.206230	479.756567	0
130.382812	39.844056	-0.158323	0.389540	1.220736	14.378941	13.539456	198.236457	0
107.250000	52.627078	0.452688	0.170347	2.331940	14.486853	9.001004	107.972506	0
107.257812	39.496488	0.465882	1.162877	4.079431	24.980418	7.397080	57.784738	0

Figure 4: First 10 rows of data - before normalization

Mean_integrated	Standard_deviation_integrated	Excess_kurtosis_integrated	Skewness_integrated	Mean_DM-SNR	Standard_deviation_DM-SNR	Excess_kurtosis_DM-SNR	Skewness_DM-SNR	Class
0.721342	0.417687	0.165043	0.015627	0.013382	0.113681	0.294986	0.063890	0
0.517628	0.460908	0.235415	0.018268	0.006560	0.072524	0.364015	0.108443	0
0.520346	0.196868	0.221138	0.040677	0.013030	0.139188	0.288624	0.054610	0
0.700933	0.437884	0.181750	0.016534	0.015368	0.131583	0.266348	0.046581	0
0.443854	0.214847	0.249044	0.041712	0.004327	0.039684	0.462029	0.213369	0
0.469784	0.296271	0.242110	0.031600	0.006376	0.069473	0.365216	0.111797	0
0.608507	0.324200	0.191792	0.024033	0.003522	0.018487	0.593047	0.403808	0
0.666848	0.203657	0.172710	0.031211	0.004514	0.067865	0.442652	0.167827	0
0.543014	0.376384	0.234145	0.028075	0.009493	0.068910	0.322202	0.092164	0
0.543055	0.198961	0.235472	0.042275	0.017323	0.170521	0.279634	0.050095	0

Figure 5: First 10 rows of data - after normalization

2 PHASE 2 : Build a model to overfit the entire dataset

In this phase we build a model to overfit the data . We train the model with different number of neurons at a time and observe the changes in accuracy . We continue to increase number of layers and neurons to increase the accuracy. We also use logistic regression and random baseline model for evaluation purposes. Below are the models used along with the number of layers and neurons used in each model :

Model Name	Number of layer	Number of Neurons
Single layer Model	1	1
Multi layer model	2	9
Multi layer model 2	3	25
Multi layer model 3	4	113
Multi layer model 4	5	241
Multi layer model 5	5	481
Multi layer model extended	5	241

Multi layer extended model is the model where output feature also added to input features , which has 100 percent accuracy .

Below is the image which shows model accuracy with respect to 100 epochs each.

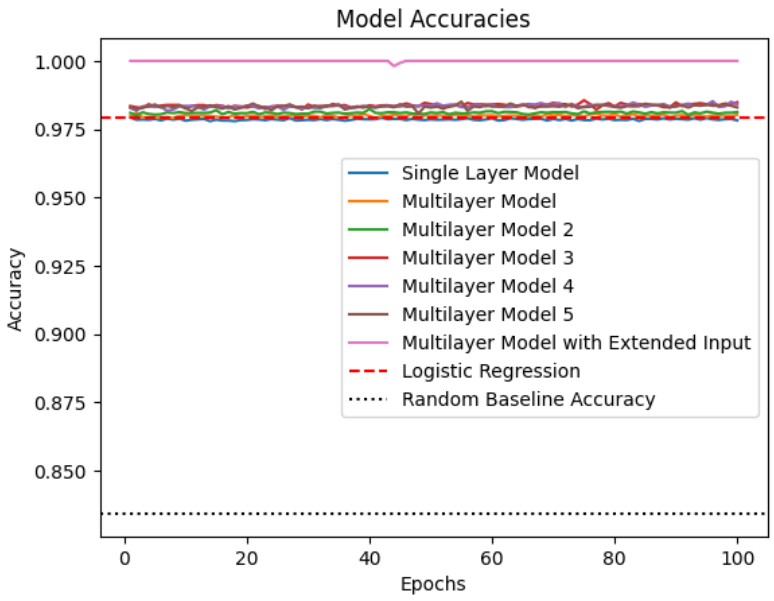

Figure 6: Model Accuracies

In this phase , we also train logistic regression model and Random base line classifier models .

Below is the bar graphs which compares accuracy of all models

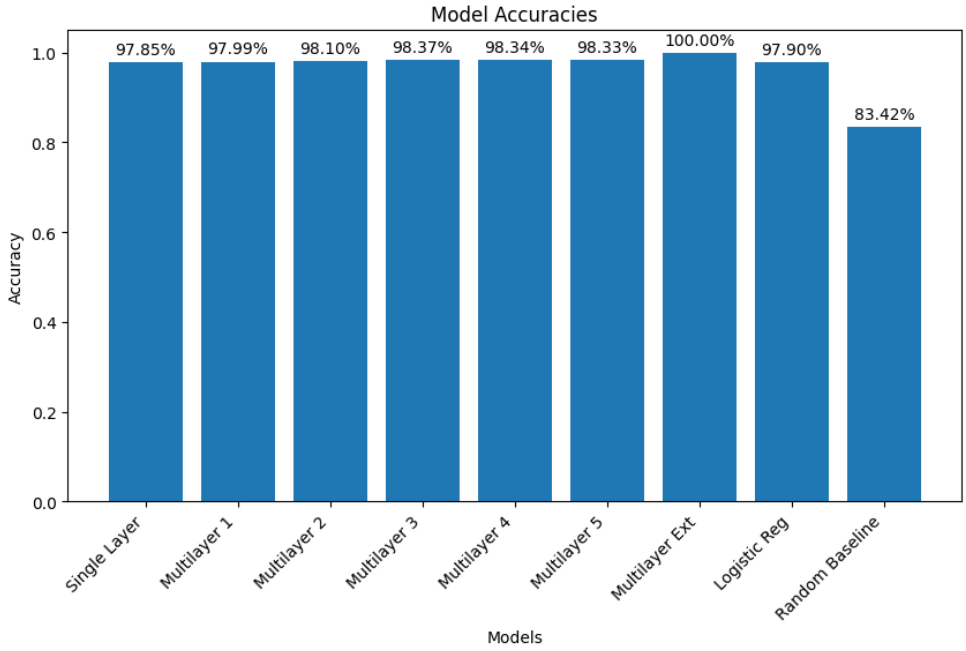

Figure 7: Model Accuracies comparision

3 Phase 3: Model selection evaluation

3.1 Data shuffling and splitting

The data in this section is split and shuffled, but it hasn't been normalized in any way. Without using normalization, we build the model using the raw data. To make sure there is no bias or innate order in the data, which could cause the model to perform poorly, we first shuffle the data set. In order to train the model on one set of data and assess its performance on an additional, unseen set, we next divided the data into training and validation sets.
After a random shuffling of the data, the data set was divided into training and validation, with 80 percent of the data set going toward training and 20 percent toward validation.

```
Training data (X_train):
[[ 1.00007812e+02  5.24918887e+01  3.33387566e-01 ...  3.36414718e+01
   4.69597949e+00  2.20442193e+01]
 [ 1.17960938e+02  4.37531075e+01  1.53131872e-01 ...  1.34146268e+01
   1.12862886e+01  1.63607278e+02]
 [ 1.15429688e+02  5.76933818e+01  2.85553040e-01 ...  4.66429823e+01
   3.03574881e+00  8.58165212e+00]
 ...
 [ 1.22281250e+02  5.02327298e+01  2.20314595e-01 ...  1.61928694e+01
   9.35133314e+00  9.91549284e+01]
 [ 1.44429688e+02  5.69876198e+01 -3.84979299e-01 ...  5.93916851e+01
  -1.93011796e+00  3.00115705e+00]
 [ 1.23414062e+02  4.70092290e+01  2.51153285e-01 ...  9.49738177e+01
  -5.64459740e-02 -1.92726590e+00]]

Training labels (y_train):
[0 0 0 ... 0 0 0]

Validation data (X_val):
[[ 1.54843750e+02  4.54801532e+01 -2.35065112e-01 ...  6.98043459e+01
  -9.61326595e-01 -8.82046466e-01]
 [ 1.36664062e+02  5.79159247e+01 -8.72293580e-02 ...  2.14114434e+01
   6.65706084e+00  4.99741235e+01]
 [ 1.02320312e+02  4.93748313e+01  4.61476528e-01 ...  2.60970314e+01
   8.22055070e+00  6.76263480e+01]
 ...
 [ 1.62500000e+01  3.08686734e+01  6.02651104e+00 ...  9.34186619e+01
   1.03484210e-02 -1.56773020e+00]
 [ 1.26992188e+02  5.45119677e+01 -4.39659100e-02 ...  1.71115954e+01
   1.08879706e+01  1.26140947e+02]
 [ 1.03742188e+02  4.24597688e+01  2.16400606e-01 ...  1.24261744e+01
   1.50022228e+01  2.46426074e+02]]

Validation labels (y_val):
[0 0 0 ... 1 0 0]
```

Figure 8: Data showing model accuracy and loss

3.2 Model Selection

3.2.1 Single Layer Neural Network

The first model is a baseline model (act as a control) that has a basic architecture of one input and one output layer. An early stopping technique was used during the training.
Create a sequential model object named single-layer-model.
Add a dense layer with a single neuron and sigmoid activation function to the model. The input dimension is set to 8, as there are 8 input features. Compile the model with the Adam optimizer, binary cross-entropy loss function, and accuracy as the evaluation metric. We get: Single Layer Model Training Accuracy: 97.78
Single Layer Model Validation Accuracy: 97.85
Recall: 84.85
Precision: 91.21
F1 Score: 87.91

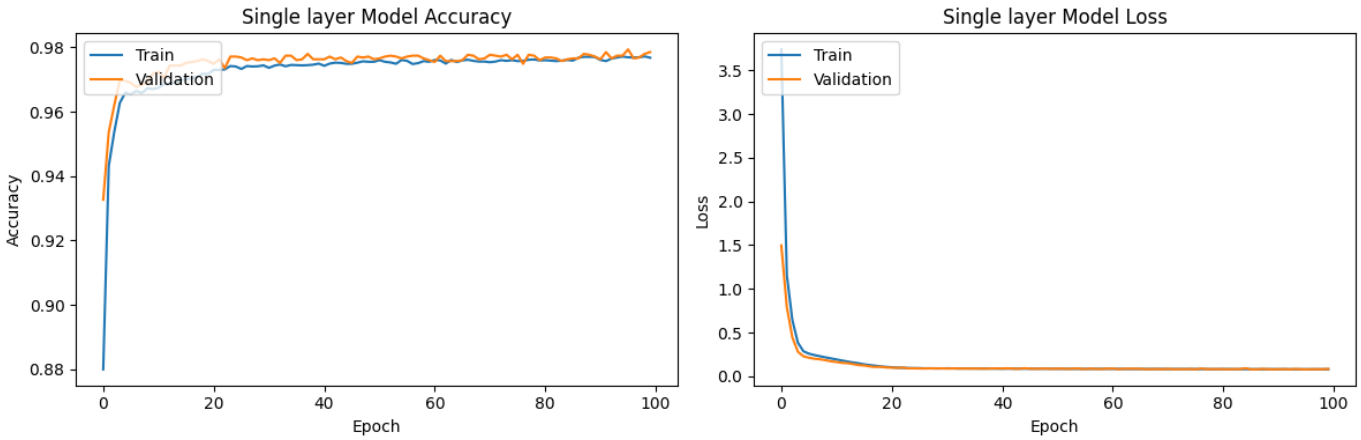

Figure 9: Single layer model accuracy and loss

3.2.2 Multi layer Neural Network

Define a sequential neural network model (multi layer model) with L1 and L2 regularization. The model has an input dimension of 8 and consists of three layers:
The first layer has 16 neurons with ReLU activation and L1 L2 regularization.
The second layer has 8 neurons with ReLU activation and L1 L2 regularization.
The third layer has 1 neuron with sigmoid activation for binary classification.
The model is compiled using the 'adam' optimizer, binary cross-entropy loss, and accuracy as the metric.
Multi layer model Training Accuracy: 96.95
Multi layer model Validation Accuracy: 97.07
Recall: 0.72
Precision: 0.94
F1 Score: 0.82

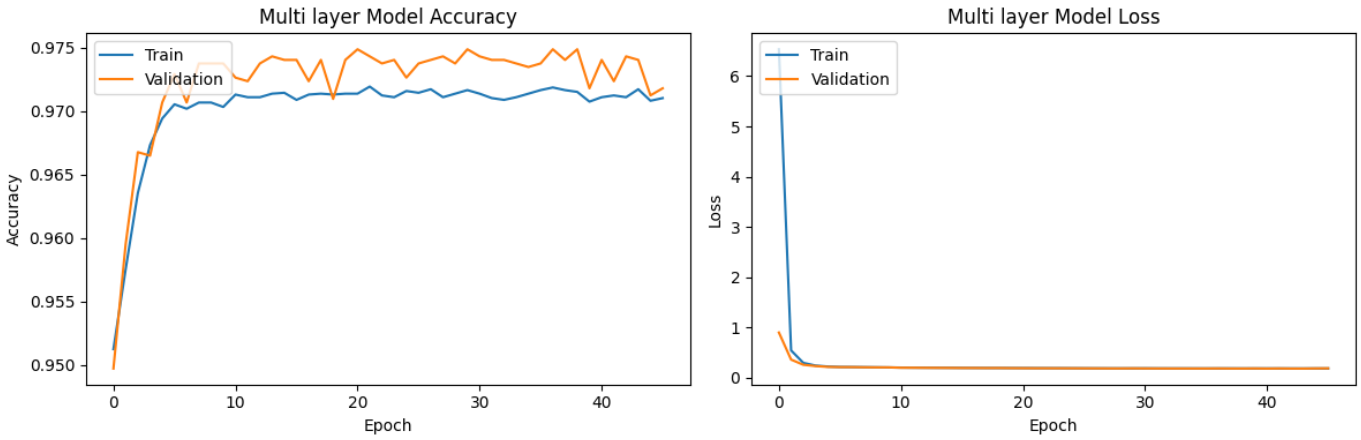

Figure 10: Multi layer model accuracy and loss

3.2.3 Logistic Regression

Create and train a logistic regression model: A" Logistic-Regression model is created with a higher maximum number of iterations (max iter=1000) than the default value (which is 100) to allow the model to converge. The model is then trained using the fit method with the training data (X train, y train).
Logistic Regression Model Training Accuracy: 97.91
Logistic Regression Model Validation Accuracy: 97.96
Recall: 0.84
Precision: 0.93
F1 Score: 0.88

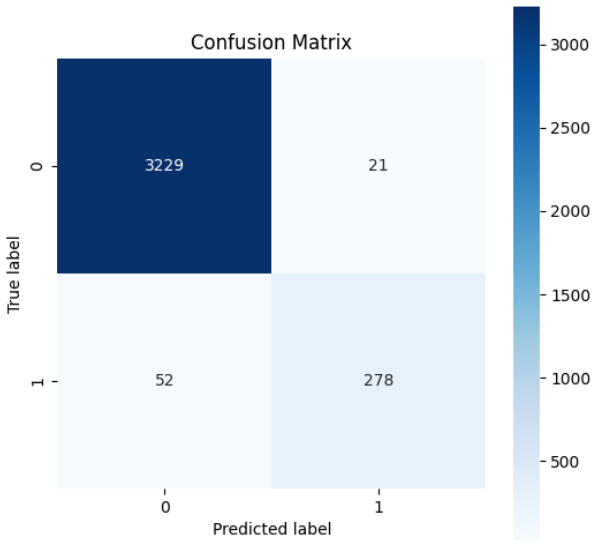

Figure 11: Confusion matrix of logistic Regression

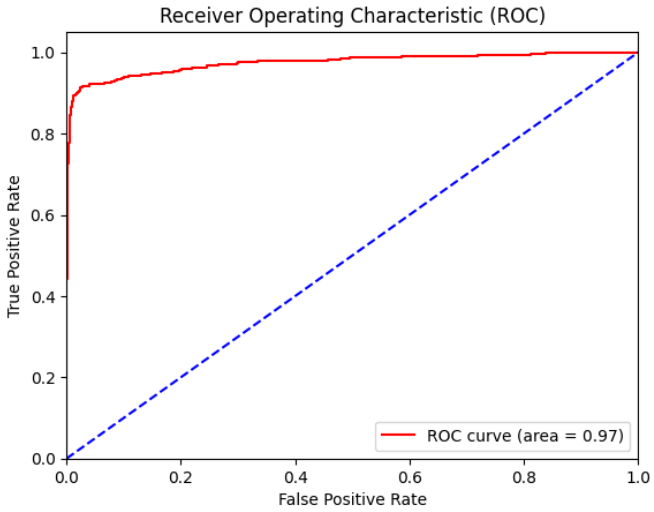

Figure 12: ROC of Logistic regression

3.2.4 Random Baseline Classifier

A random baseline classifier is a simple model that makes predictions by randomly guessing the class labels, without considering the input features. It serves as a basic point of comparison for more complex models, allowing you to assess whether your actual model is performing better than random chance. In the context of the provided code, a stratified random baseline classifier is used, which generates predictions by respecting the training set's class distribution.
Here, Import the necessary libraries and functions.Create and train a stratified random baseline classifier using the training data.Then make predictions on the validation set using the trained classifier. Calculate and print the accuracy of the random baseline classifier on the validation set. By comparing the performance of this random baseline classifier with other models, you can evaluate if your actual models are learning useful patterns in the data or just performing at the level of random chance.
Random Baseline Model Training Accuracy: 83.29
Random Baseline Model Validation Accuracy: 83.21
Recall: 0.06
Precision: 0.07
F1 Score: 0.07

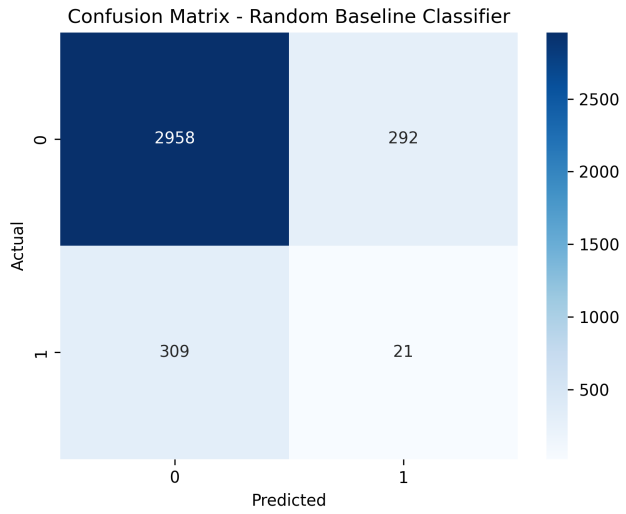

Figure 13: Confusion matrix of Random baseline model

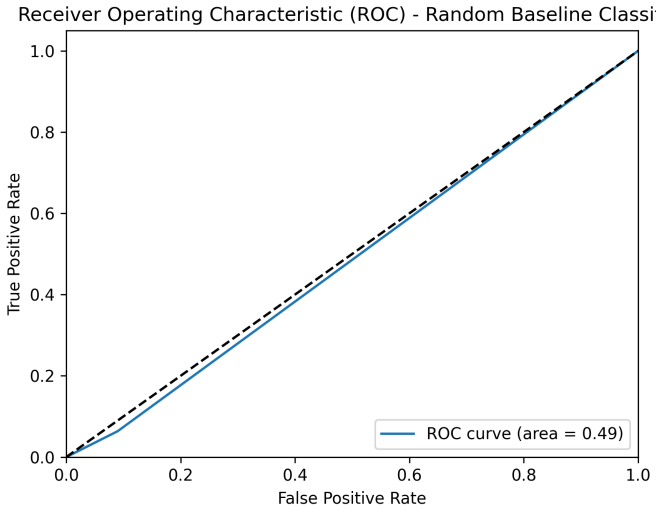

Figure 14: ROC of Random baseline model

3.2.5 Custom Prediction Function Model

This model code is for evaluating the performance of a binary classification model. It uses a custom prediction functionto make predictions on the training and validation data.
If the model is a tree-based model (e.g., XGBoost), the prediction function uses the predict proba method to get the predicted probabilities. If the model is a linear model (e.g., logistic regression), the prediction function uses the model's weights and bias to calculate the predicted probabilities using the sigmoid activation function.
Custom prediction model Training Accuracy: 98.26 percent
Custom prediction model Validation Accuracy: 98.07 percent
Recall: 86.97 percent
Precision: 91.69 percent
F1 Score: 89.27 percent

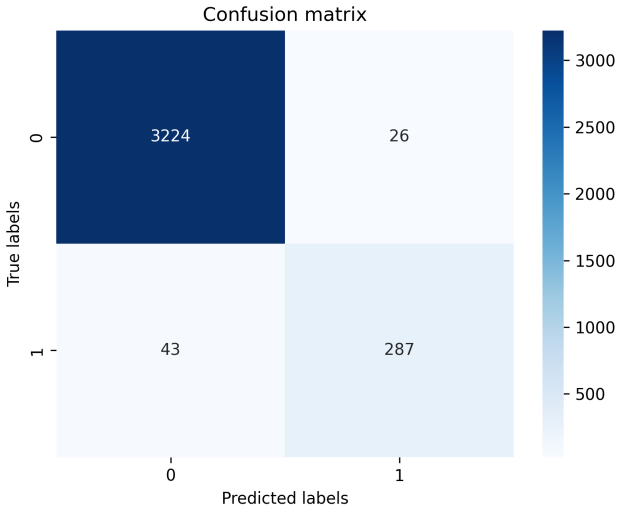

Figure 15: Confusion matrix of custom prediction model

3.3 Model Performances

Accuracy comparisons of all the built models are below

Model Name	Training Accuracy	Validation Accuracy
Single layer Model	97.78	97.85
Multi layer model	96.95	97.07
Logistic Regression model	97.91	97.96
Random Baseline model	83.29	83.21
Custom Prediction Model	98.26	98.07

The table shows the performance of different machine learning models on a task where they have to classify data. The models were trained on a specific set of data, and their accuracy was measured to see how well they could classify new data.

The accuracy of a model is a measure of how well it performs on the task. The training accuracy shows how well the model did on the data it was trained on, while the validation accuracy shows how well it can classify new data that it hasn't seen before.
Looking at the table, we can see that the Custom Model had the highest training accuracy of 98.26% and highest validation accuracy of 98.07% . The Random Baseline Classifier performed the worst, with a training accuracy of 83.29% and a validation accuracy of 83.21%. This suggests that it wasn't able to learn much from the training data.
Below is the bar graph which shows Accuracy levels of each model

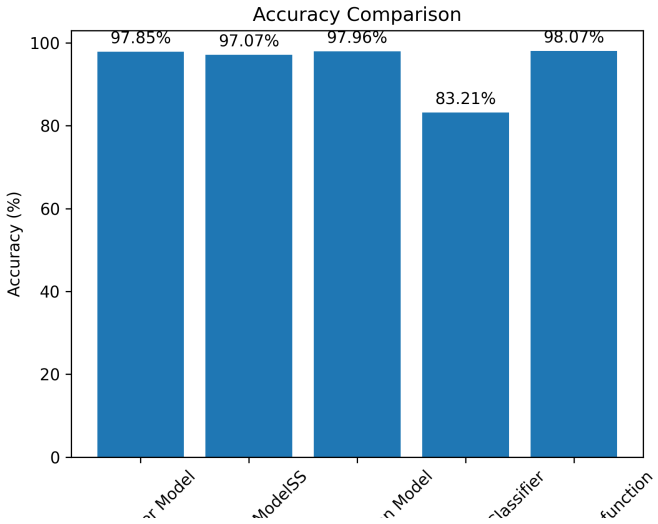

Figure 16: Comparisions of accuracies

Overall, the accuracy measures give us an idea of how well each model performed on the classification task and how well they can generalize to new data. It's important to choose a model with a good validation accuracy, as this tells us how well it will perform on new data.

4 Model Evaluation

We used three essential classification model metrics to evaluate. The table given below shows the precision, recall and f1 score for the neural network model.
1. Precision: what proportion of positive identifications was actually correct?
2. Recall: what proportion of actual positives was identified correctly?
3. F1-Score: evaluation metric for classification algorithms, where the best value is at 1 and the worst is at 0.

Model Name	Recall	Precision	F-1 Score
Single layer Model	84.85%	91.21%	87.91%
Multi layer model	72%	94%	82%
Logistic Regression model	84%	93%	88%
Random Baseline model	60%	70%	70%
Custom Prediction Model	86.97%	91.69%	89.27%

5 Phase 4: Feature importance and reduction

5.1 Feature Importance Analysis

As we have a pretty good idea of which model to use, the number of epochs, and a reasonable validation set to use for the network architecture. The next step is to find out which input features is redundant or insignificant.

5.1.1 Forward Selection with Check pointing

After selecting the appropriate model, number of epochs and validation set, the next step is to determine the importance of individual input features
Feature 0 - Mean of the integrated profile,
Feature 1 - Standard deviation of the integrated profile,
Feature 2 - Excess kurtosis of the integrated profile,
Feature 3 - Skewness of the integrated profile,
Feature 4 - Mean of the DM-SNR curve,
Feature 5 - Standard deviation of the DM-SNR curve,
Feature 6 - Excess kurtosis of the DM-SNR curve Skewness of the DM-SNR curve.
Feature 7 - Skewness of the DM-SNR curve.
A graph illustrating the performance of the model with respect to these features is provided below.
The code performs feature importance analysis and reduction. It trains single-feature models and calculates their validation accuracies. The validation accuracies are plotted to determine the significance of each input feature. Based on the plot, five input features have a small impact on the overall accuracy of the model. Then, the unimportant features are removed and the validation accuracy of the reduced-feature models are compared.
The plot shows the validation accuracy versus the number of removed features. The feature reduction technique used is forward selection, where features are removed one by one based on their significance.

The accuracy of each feature when built individually

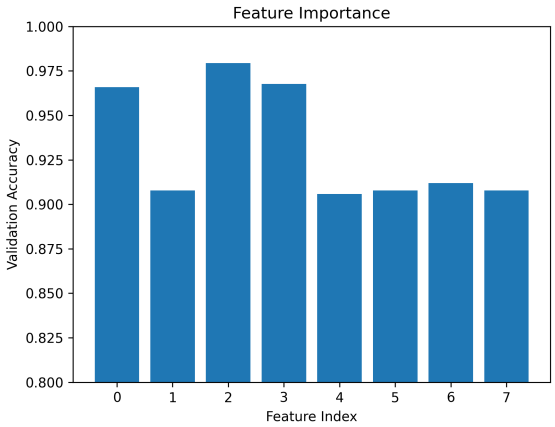

Figure 17: Significance of each feature

Performance after removing each feature one by one

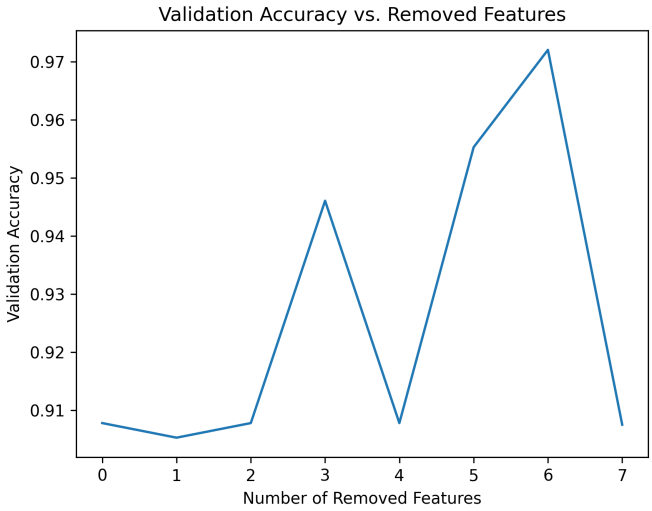

Figure 18: Graph between validation accuracy and each feature

5.1.2 Feature Reduction using LIME and SHAP

Two well-liked methods for elucidating machine learning model predictions are SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations). While SHAP is based on the idea of game theory and provides global explanations by attributing the contribution of each feature to the final prediction, LIME is a model-agnostic method that produces local explanations for individual predictions. These methods offer insightful information about the features that matter most and how a model generates its predictions.

Two methods are used in the code to implement feature importance analysis: individual feature importance and SHAP. The code trains multiple single-feature models and assesses each one's validation accuracy to determine the importance of each individual feature. The code ranks the features using SHAP values after training an XGBoost model. The validation accuracy of gradually lowering the number of top features in the model is then compared by the code. Plotting the validation accuracy for each method is the last step.
After that, eliminate features that aren't necessary, and evaluate the reduced feature models' validation accuracy. Features were removed iteratively in decreasing order of significance after being sorted according to their SHAP importance scores. Train a single-layer neural network with each reduced feature set, then assess the network's accuracy using the validation set.
In order to identify the ideal number of top features to include in the model, we lastly plotted the validation accuracy for each reduced feature set.

Feature Importance based on SHAP values

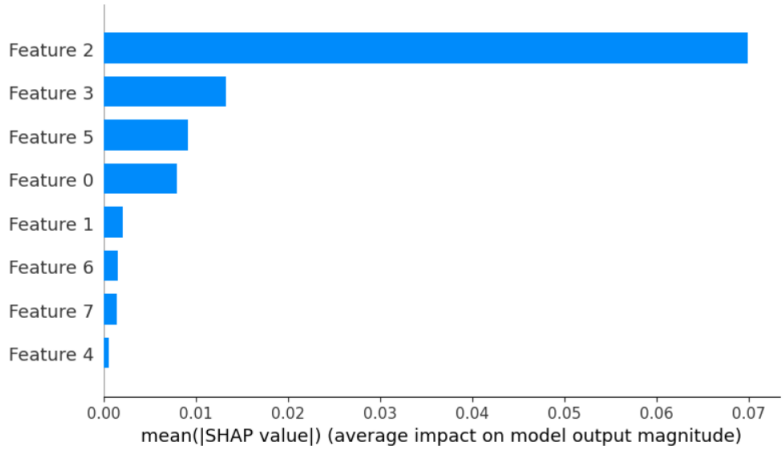

Figure 19: The figure shows the "Excess kurtosis of the integrated profile" is the most important input feature when compared with other input feature.

Performance after removing less important feature

6 Challenges faced

As the data turned to be more overfitting than expected , I faced challenges while training multilayer neural networks in determining how many layers, neurons and epochs are sufficient for training. While coding the checkpoint function, I also ran into some issues because my developed model initially failed to save the best accuracy; however, after a few tries, the best accuracy began to save into the.h5 file. Another important task that needed to be completed was plotting the ROC and the confusion matrix.

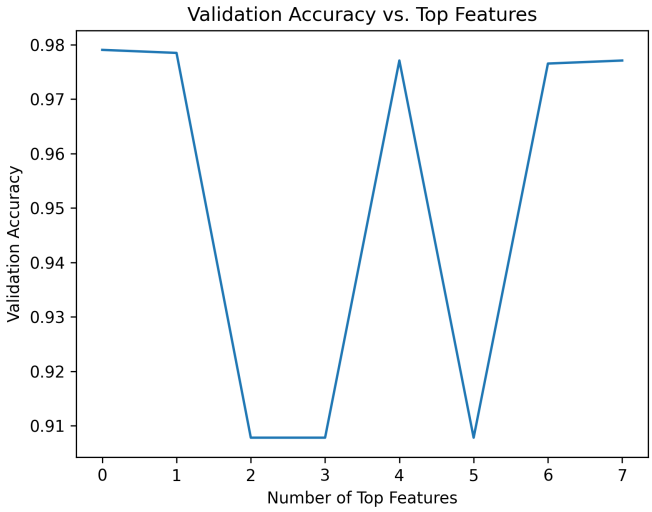

Figure 20: Performance after removing each least important feature