Predicting Pulsar Stars

Duc Ngo - CS4300

May 6, 2020

Contents

1 Introduction

What is a Pulsar? Whenever you look at the beautiful night sky, you may find yourself looking at a stationary shiny object and there is a slim chance that you are looking at a pulsar. Pulsars frequently look like glinting stars, and they appear to sparkle with a normal cadence. Nonetheless, the light from pulsars doesn't sparkle or beat, and these celestial bodies are not stars. Pulsars are round, minimal objects that are about the size of a huge city however contain more mass than our sun or other average stars. [1]

Why are they so fascinating? Researchers are utilizing pulsars to analyze the extreme states of matter, look for planets past Earth's nearby planetary group and measure cosmic distances. Pulsars likewise could assist researchers with finding gravitational waves, which could guide the way to energetic cosmic events like collisions between supermassive black holes. Pulsar is mysterious and still new to the field of science which is why we are studying them. These celestial objects act so differently on an extreme scale that we never experience here on Earth. They are by definition is an extreme science. [1]

Motivation Artificial intelligence could be the perfect tool for exploring the Universe since finding a specific celestial object is a slow and tedious job. With this in mind, the motivation of this project is to try to write supervised learning algorithms to predict the class of pulsars with binary classification. Binary classification is a machine learning technique that categorizes whether the element is either true or false OR 1 or 0.

Google Colab `https://colab.research.google.com/drive/1cOrM_J8jWFs0k9VgNbJEcGPoJ74ZGF-m`

2 Dataset

The "Predicting a Pulsar Star" dataset was obtained from the Kaggle Data Science [4][5]. The pulsar candidate data were obtained during the High Time Resolution Universe (HTRU) survey. It contains 16,259 apocryphal (negative) examples caused by RFI/noise and 1,639 real pulsars (positive) examples which totals to 17,898 samples in the dataset. These samples have all been checked by human annotators. Each row lists the variables first that described by 8 continuous variables, and a single class label as the final entry. The class labels used are 0 (negative) and 1 (positive). The input features are for the following fields:

1. Mean of the integrated profile.
2. The standard deviation of the integrated profile.
3. Excess kurtosis of the integrated profile.
4. The skewness of the integrated profile.
5. Mean of the DM-SNR curve.
6. The standard deviation of the DM-SNR curve.
7. Excess kurtosis of the DM-SNR curve.
8. The skewness of the DM-SNR curve.
9. Class

2.1 Visualization of the distribution of each input features

The histogram plot of each info highlights indicating their most extreme and least value as well as how they are distributed can be found in the pictures given underneath.

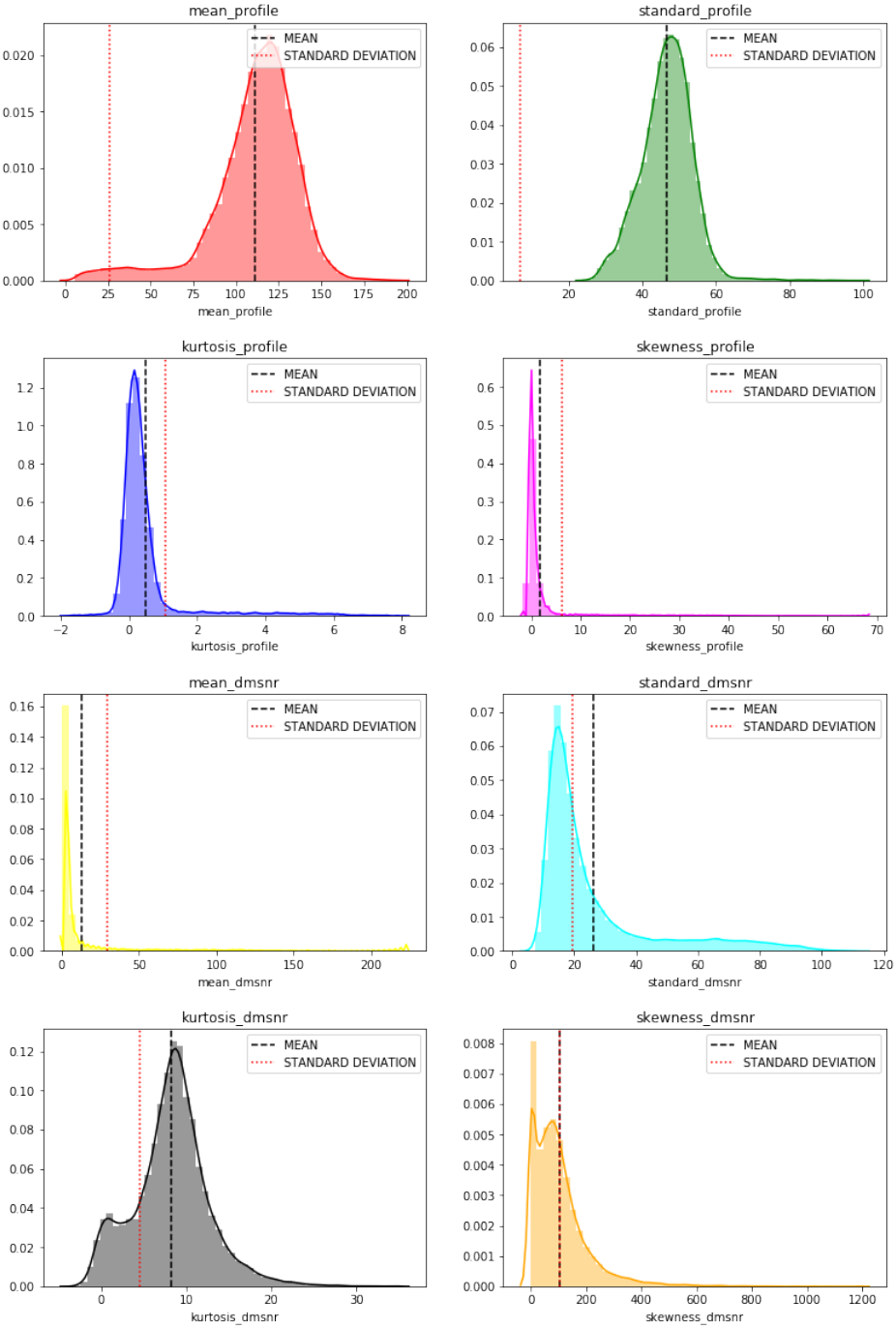

Figure 1: Input Data Distribution Histograms

	mean_profile	standard_profile	kurtosis_profile	skewness_profile	mean_dmsnr	standard_dmsnr	kurtosis_dmsnr	skewness_dmsnr	target
count	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000
mean	111.079968	46.549532	0.477857	1.770279	12.614400	26.326515	8.303556	104.857709	0.091574
std	25.652935	6.843189	1.064040	6.167913	29.472897	19.470572	4.506092	106.514540	0.288432
min	5.812500	24.772042	-1.876011	-1.791886	0.213211	7.370432	-3.139270	-1.976976	0.000000
25%	100.929688	42.376018	0.027098	-0.188572	1.923077	14.437332	5.781506	34.960504	0.000000
50%	115.078125	46.947479	0.223240	0.198710	2.801839	18.461316	8.433515	83.064556	0.000000
75%	127.085938	51.023202	0.473325	0.927783	5.464256	28.428104	10.702959	139.309331	0.000000
max	192.617188	98.778911	8.069522	68.101622	223.392140	110.642211	34.539844	1191.000837	1.000000

Figure 2: Input Feature Statistics

2.2 Distribution of the output labels

Notice that the data is imbalanced and may need to be resampled (such as oversampling, undersampling, or generate synthetic samples), but for now it is acceptable.

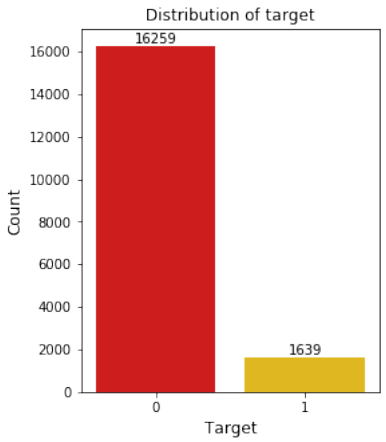

Figure 3: Output Data Distribution Histogram

3 Data Processing

3.1 Data Splitting

The data was randomly shuffled and then the dataset was split into training and validation, where 80% of the dataset was allocated for training and 20% was allocated for validation.

3.2 Data Normalization

Before data mining itself, data preprocessing plays a crucial role. As can be seen, the data was not distributed uniformly. In this manner, we have to pre-process the data with normalization techniques. Normalization makes the optimization problem more numerically stable and makes training less sensitive to the scale of features, so we can better solve for coefficients. When applying normalization, all the values are all now between 0 and 1, and the outliers are eliminate but remain visible within our normalized data. There are two normalization techniques that can be utilized and each of them has its own consequences, but for now, any of them is sufficient.

Mean Normalization Formula

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Min-Max Normalization Formula

$$X_{new} = \frac{X - X_{mean}}{X_{max} - X_{min}}$$

3.3 Normalized Data

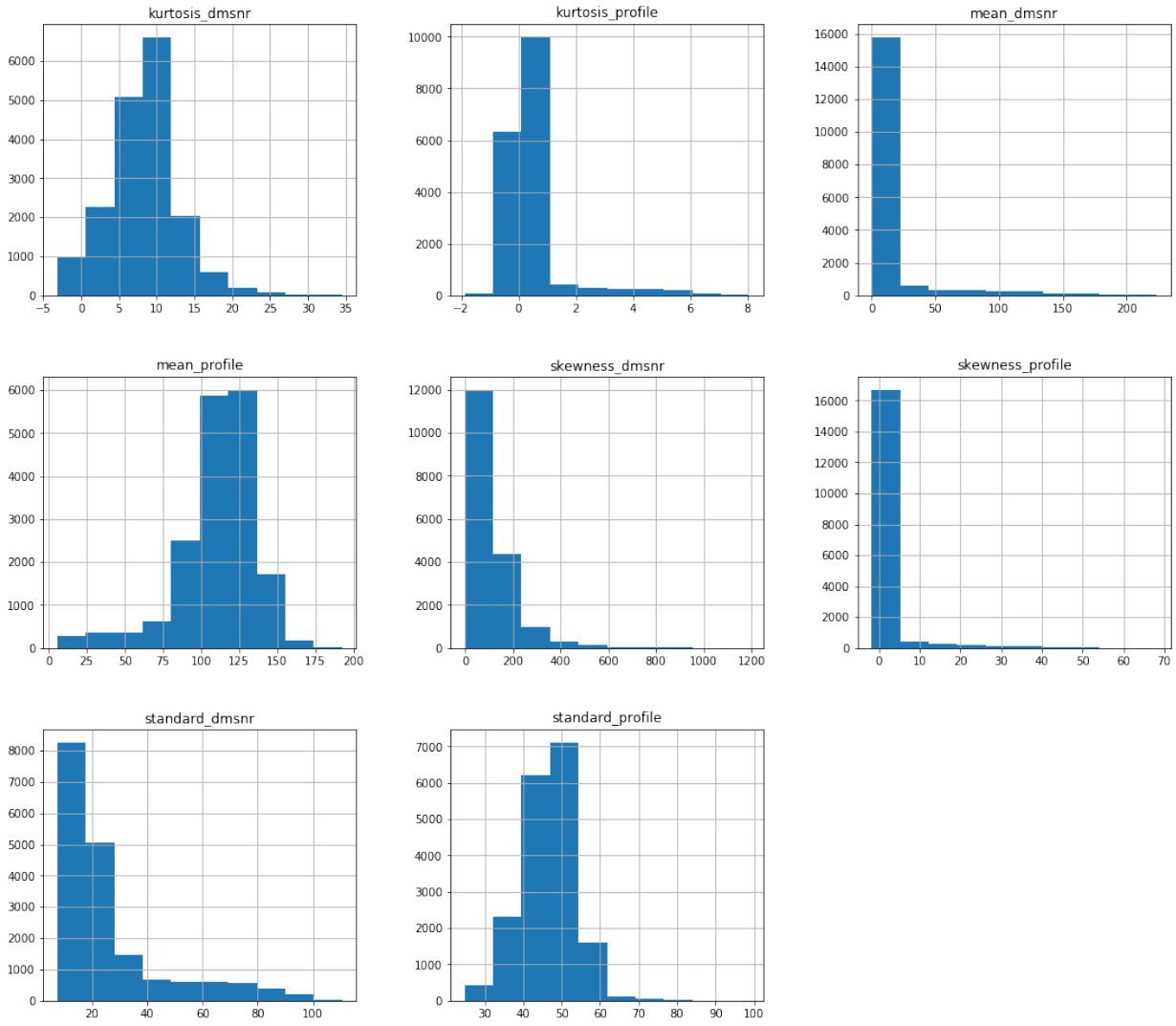

Figure 4: Input Data Before Normalization (Min-Max)

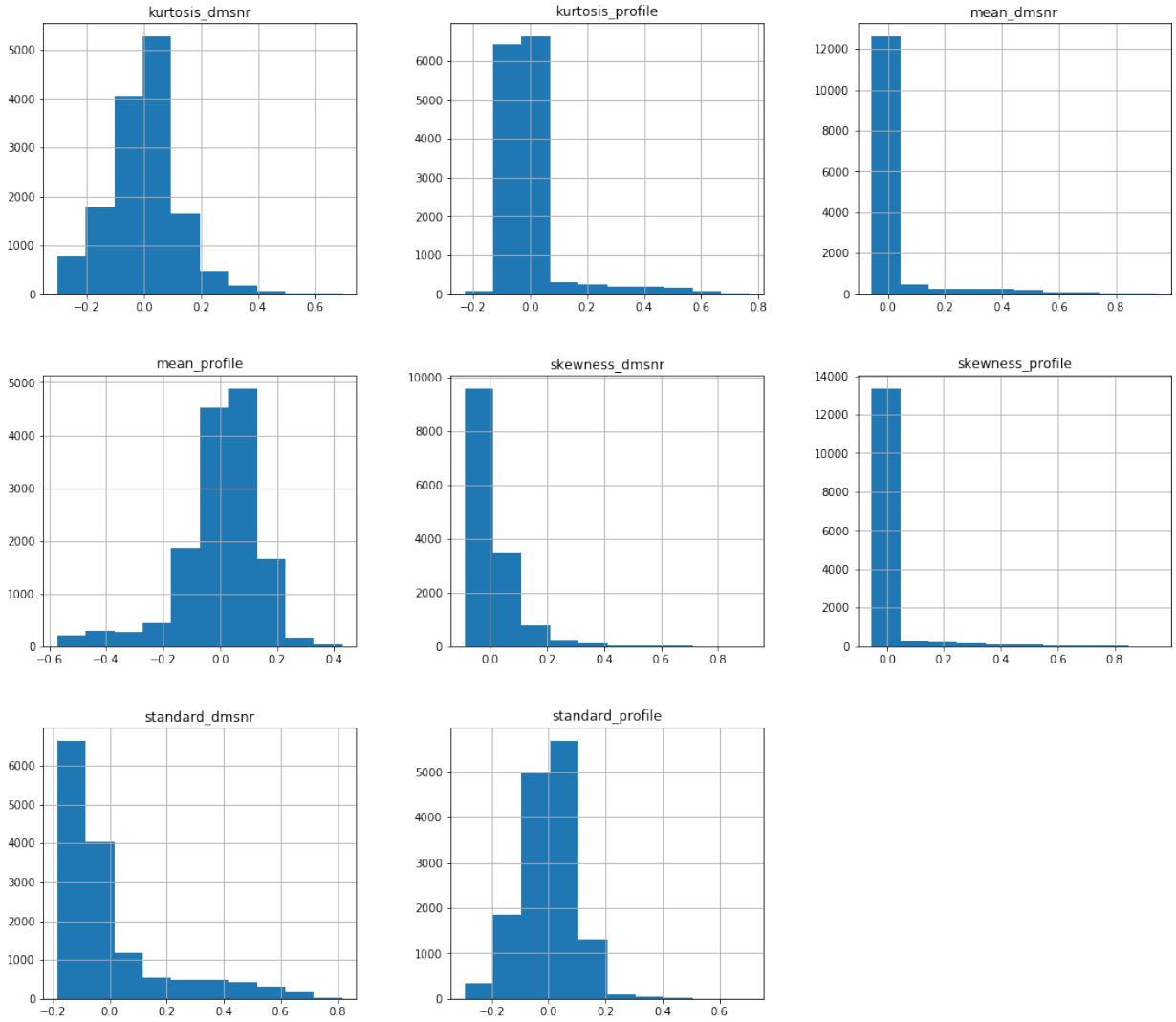

Figure 5: Input Data After Normalization (Min-Max)

4 Modelling

A feed forward artificial neural network architectures was used to create the model.

NOTE: Data is shuffle. Thus, the result will vary every time. All models were compiled and fit on May 5, 2020.

4.1 Selected Neural Network Architecture

The first model is a baseline model (act as a control) that has a basic architecture of one input and one output layer. It will then gradually increase by one hidden layer and up to 2 hidden layers at the end. An early stopping technique was used during the training.

4.1.1 Baseline Model Performance

Hidden	Accuracy	Loss
0 Layer	97.91	0.07
1 Layer	97.91	0.07
2 Layer	97.99	0.07

Table 1: performance comparison for different hidden layers

As can be seen from the above table, all the basic architecture somewhat perform relatively the same (2 hidden layers just happen to perform better during this iteration). In the end, only one hidden layer will be applied to other architecture.

4.2 Learning Curve of Baseline Model

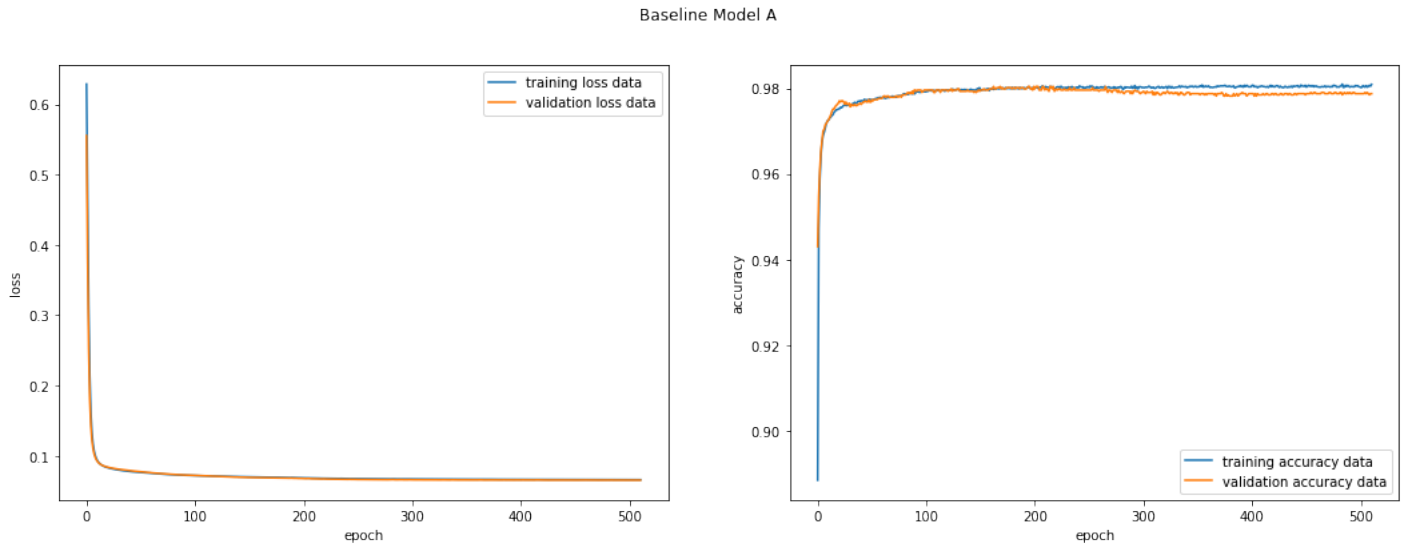

Figure 6: curve showing change in loss/accuracy vs epoch

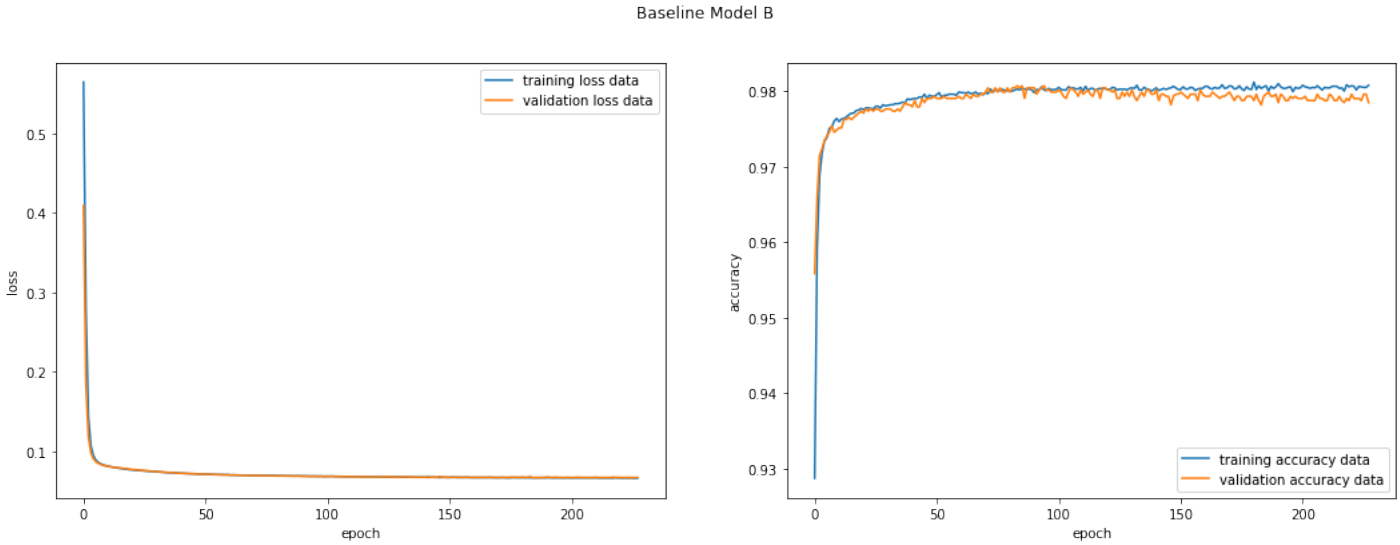

Figure 7: curve showing change in loss/accuracy vs epoch

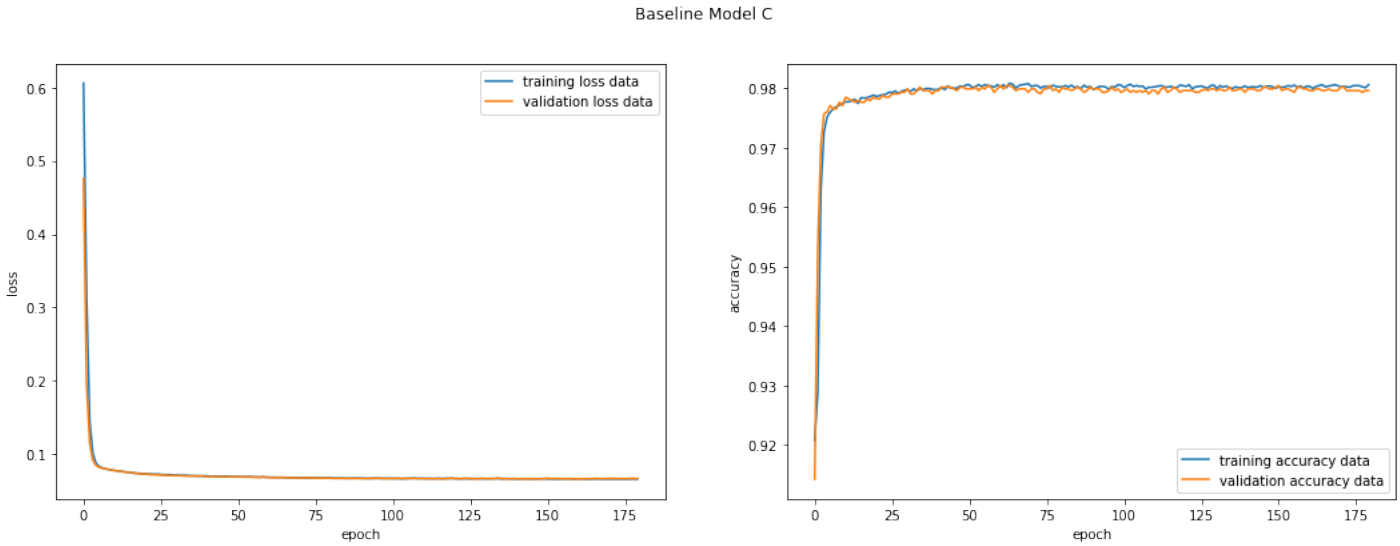

Figure 8: curve showing change in loss/accuracy vs epoch

4.3 Modifying Activation Function

The linear activation function performed poorer in comparison with a sigmoid activation function. In general, linear activation is configured with mean squared error (MSE) or mean absolute error (MAE) loss function and these functions are usually a bad choice (depend on a dataset) for binary classification problems. When using MSE, it is assumed that the underlying data has been generated from a normal distribution or a bell-shaped curve. However, in reality, the dataset that classified into two categories is not from a normal distribution. Lastly, the MSE function is non-convex for binary classification which means it is not guaranteed to minimize the loss function.[3] The performance comparison can be seen in the table shown below.

Activation Function	Accuracy	MAE	Loss
Linear (Output)	_	0.03	0.02
Linear (All)	_	0.09	0.10
Sigmoid (Output)	98.02	_	0.06
Sigmoid (All)	97.99	_	0.07
Leaky Relu + Sigmoid	97.85	_	0.07

Table 2: performance comparison for different activation function

4.4 Learning Curve of Model using Linear Activation (last neuron)

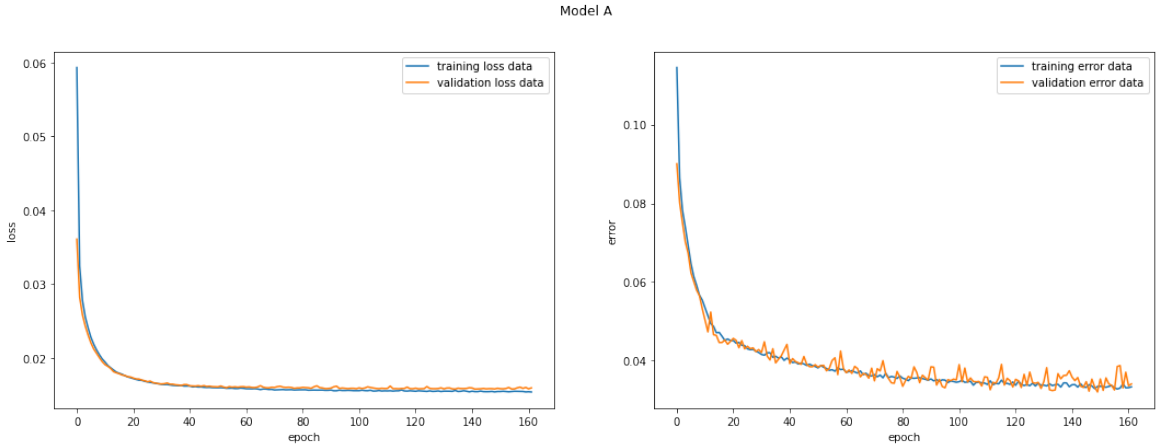

Figure 9: curve showing change in loss/error vs epoch

4.5 Learning Curve of Model using Linear Activation (all neuron)

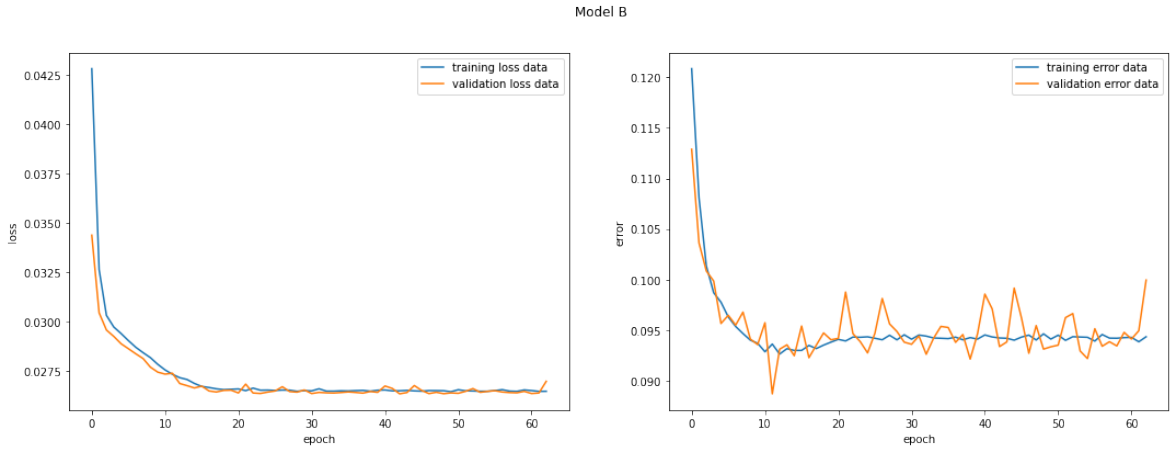

Figure 10: curve showing change in loss/error vs epoch

4.6 Learning Curve of Model using Sigmoid Activation (last neuron)

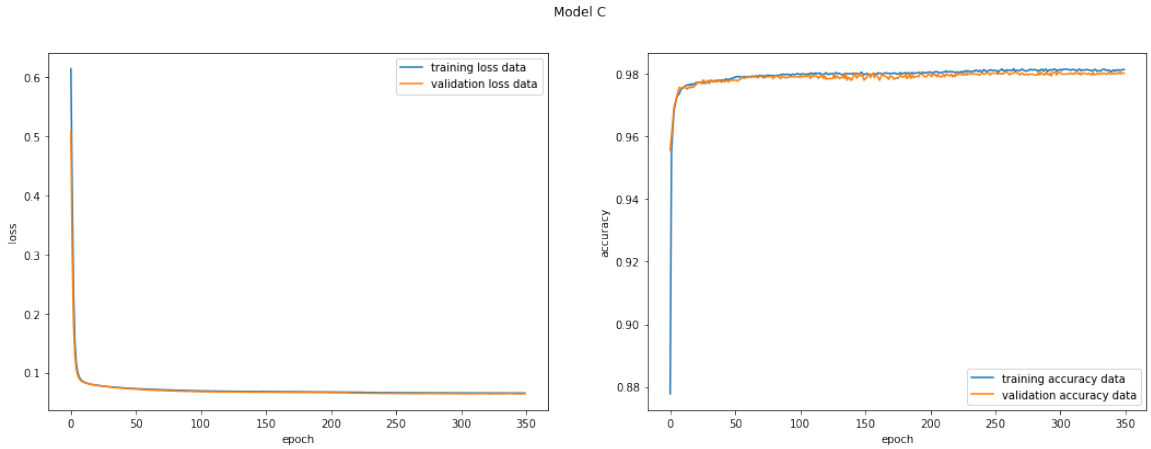

Figure 11: curve showing change in loss/accuracy vs epoch

4.7 Learning Curve of Model using Sigmoid Activation (all neuron)

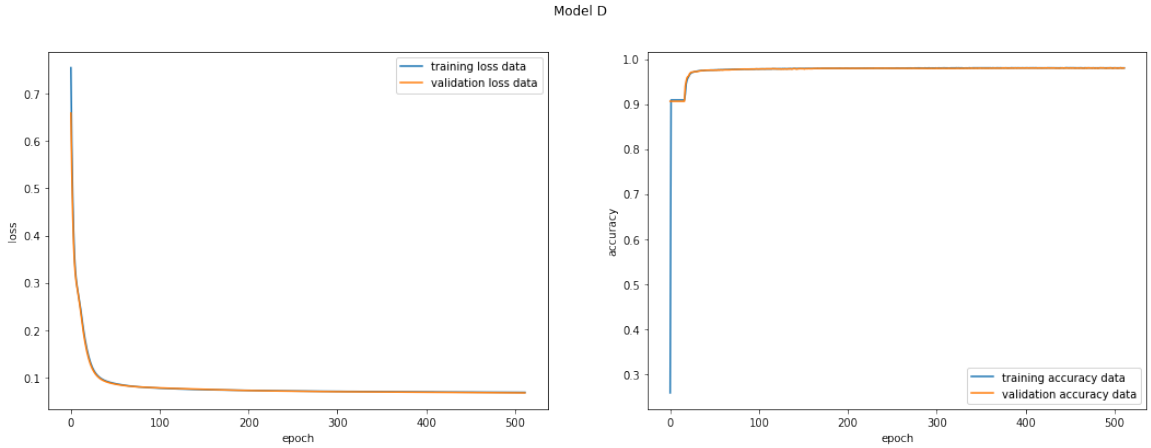

Figure 12: curve showing change in loss/accuracy vs epoch

4.8 Learning Curve of Model using Sigmoid with Leaky Relu activation

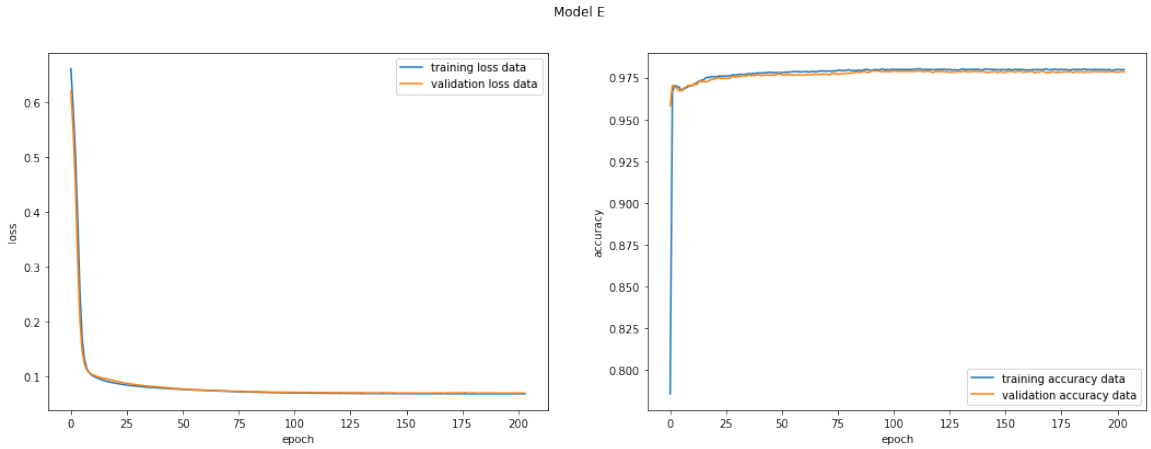

Figure 13: curve showing change in loss/accuracy vs epoch

4.9 Learning Curve of Overfitting

The number of neurons in each layer was increased by a factor of 10 to overfit the model.

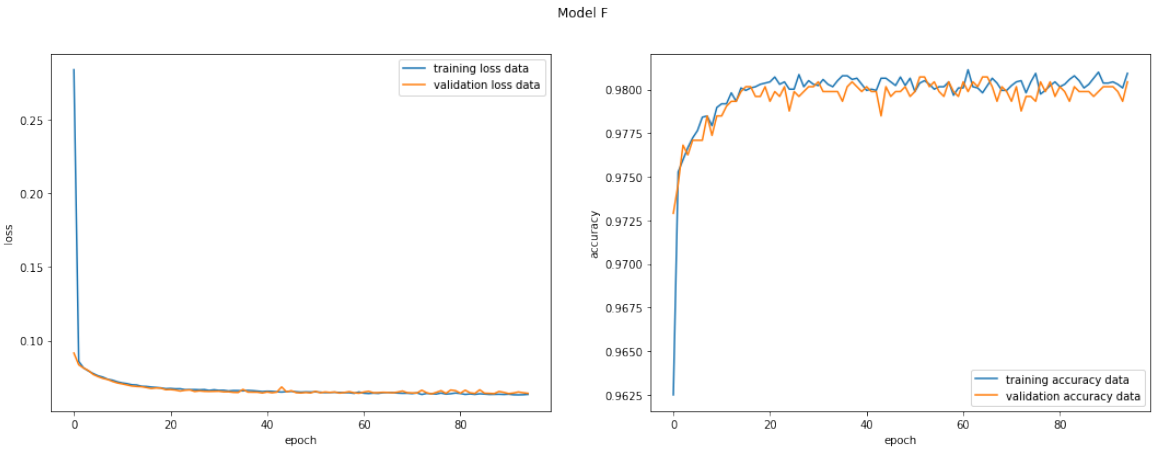

Figure 14: curve showing change in loss/accuracy vs epoch

	Accuracy	Loss
Overfitting	97.96	0.06

Table 3: performance of overfitting model

5 Model Evaluation

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	Precision	Recall	F1-Score
Baseline A	92.23	84.82	0.88
Baseline B	91.43	85.71	0.88
Baseline C	92.58	85.42	0.89
Sigmoid (last)	92.33	86.01	0.89
Sigmoid (all)	93.14	84.82	0.89
Leaky + Sigmoid	90.85	88.71	0.88
Overfitting	92.83	84.82	0.89

Table 4: predictions evaluation of all model

A useful tool when predicting the probability of a binary outcome is the Receiver Operating Characteristic curve or ROC curve. The area covered by the curve is the area between the red line and the axis. This area covered is AUC. The bigger the area covered, the better the machine learning models are at distinguishing the given classes. In other words, the AUC can be used as a summary of the model skill. The ideal value for AUC is 1.[2]

5.1 Test Accuracy

The closer the graph is to the top and left-hand borders, the more accurate the test. Likewise, the closer the graph to the diagonal, the less accurate the test. In a perfect test, it would go straight from zero up the top-left corner and then straight across the horizontal. The figure is given below shows the receiver operating characteristic curve for the baseline model, the linear model, and the sigmoid model.

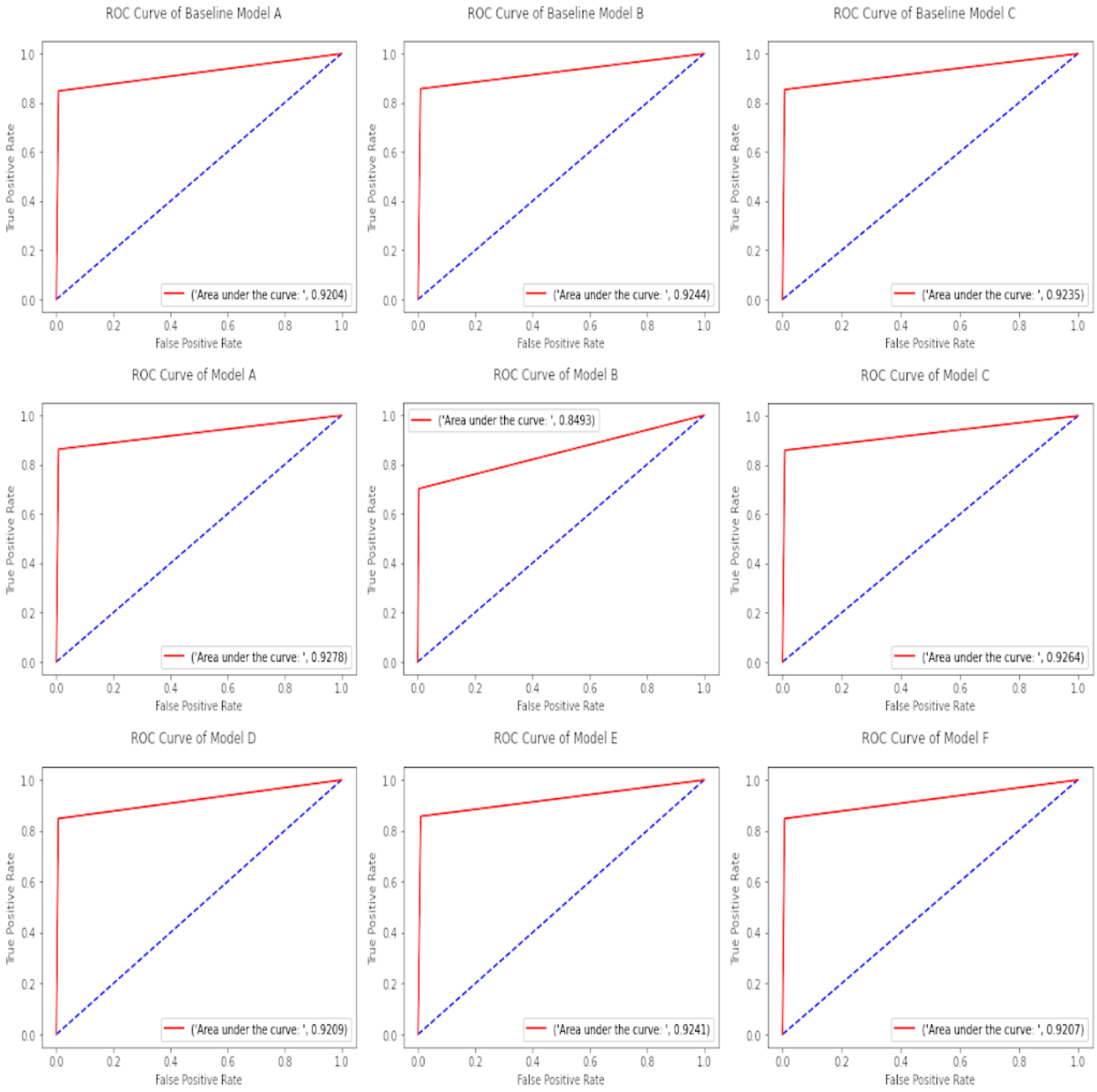

Figure 15: Receiver Operating Characteristic Curve for all model

5.2 Custom Function and Keras Function

After the model was trained, all the weights were extracted. A custom function/method was build to serves as the model. The extracted weights were then applied to the custom function to predict the outputs. The goals are to verify if the custom predictions that yield are the same as the trained model. The comparison will be on the sigmoid model (last neuron).

```
Layer #0
Weights:
 [[ 0.1 -0.4 -4.6  1.0  0.2 -0.6 -1.0  0.9]
 [-1.3 -1.8 -3.3  1.1  0.7 -1.4 -1.1  0.9]
 [ 0.1 -0.2 -0.3 -0.3  0.0 -0.3 -0.4  0.0]
 [ 0.5 -0.1  0.4  0.1  0.1 -0.2 -0.1  0.1]
 [-0.6  0.7 -2.9  0.8  0.5  0.6 -1.0  0.7]
 [ 0.1  0.7 -2.9 -0.1  0.3 -2.3 -0.4  0.7]
 [ 0.4  0.5  2.1 -0.6  0.2  0.4 -0.4 -1.1]
 [ 0.8 -0.5  0.8  0.7  0.4 -0.1  1.0  0.5]]
Bias:
 [ 0.3  0.0 -0.1 -0.2  0.4  0.3  0.7 -0.2]

Layer #1
Weights:
 [[ 2.1  1.1 -0.2 -0.3  1.6  1.4 -0.2 -1.3]
 [ 0.3 -0.3  0.6 -0.5 -0.6 -0.1 -0.2  0.2]
 [-1.9 -1.3  0.2  0.3  1.2 -1.7 -0.5 -0.5]
 [-1.8 -2.4  0.1 -0.4 -1.0 -1.9  1.3  2.2]]
Bias:
 [ 0.3 -0.1 -0.4  0.5]

Layer #2
Weights:
 [[-2.1  0.5  3.5  2.8]]
Bias:
 [-0.2]

Accuracy: 98.02%
Precision: 92.33%
Recall: 86.01%
F1-score: 0.89

X=[-0.0 -0.0 -0.0 -0.0 -0.0 -0.1  0.0  0.0], Predicted=[False]
X=[-0.1  0.1  0.0 -0.0  0.0  0.2 -0.2 -0.1], Predicted=[ True]
X=[ 0.0  0.0 -0.0 -0.0 -0.1 -0.1  0.2  0.1], Predicted=[False]
X=[-0.0  0.1 -0.0 -0.0 -0.0 -0.1 -0.0 -0.0], Predicted=[False]
X=[ 0.1 -0.0 -0.0 -0.0 -0.0 -0.1  0.0 -0.0], Predicted=[False]
X=[-0.1 -0.1  0.0 -0.0 -0.1 -0.1  0.1  0.1], Predicted=[False]
X=[-0.0 -0.0  0.0 -0.0 -0.0  0.1 -0.1 -0.1], Predicted=[False]
X=[-0.1 -0.2  0.0  0.0 -0.0 -0.1  0.0 -0.0], Predicted=[False]
X=[-0.0  0.0 -0.0 -0.0 -0.0 -0.1 -0.0 -0.0], Predicted=[False]
X=[ 0.2 -0.1 -0.1 -0.0  0.8  0.1 -0.2 -0.1], Predicted=[False]
```

Figure 16: custom function result

```
3580/3580 [==============================] - 0s 45us/sample - loss: 0.0643 - acc: 0.9802
loss: 0.06
acc: 98.02%

[[3220   24]
 [  47  289]]
98.02%

Accuracy: 98.02%
Precision: 92.33%
Recall: 86.01%
F1-score: 0.89

X=[-0.0 -0.0 -0.0 -0.0 -0.0 -0.1  0.0  0.0], Predicted=[False]
X=[-0.1  0.1  0.0 -0.0  0.0  0.2 -0.2 -0.1], Predicted=[ True]
X=[ 0.0  0.0 -0.0 -0.0 -0.1 -0.1  0.2  0.1], Predicted=[False]
X=[-0.0  0.1 -0.0 -0.0 -0.0 -0.1 -0.0 -0.0], Predicted=[False]
X=[ 0.1 -0.0 -0.0 -0.0 -0.0 -0.1  0.0 -0.0], Predicted=[False]
X=[-0.1 -0.1  0.0 -0.0 -0.1 -0.1  0.1  0.1], Predicted=[False]
X=[-0.0 -0.0  0.0 -0.0 -0.0  0.1 -0.1 -0.1], Predicted=[False]
X=[-0.1 -0.2  0.0  0.0 -0.0 -0.1  0.0 -0.0], Predicted=[False]
X=[-0.0  0.0 -0.0 -0.0 -0.0 -0.1 -0.0 -0.0], Predicted=[False]
X=[ 0.2 -0.1 -0.1 -0.0  0.8  0.1 -0.2 -0.1], Predicted=[False]
```

Figure 17: keras function result

6 Feature Importance Analysis

As of now, 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.

6.1 Significance of individual features

Five input features (standard profile, mean dmsnr, standard dmsnr, kurtosis dmsnr, and skewness dmsnr) slightly impact the overall accuracy of the model. We can see the graph below for the performance.

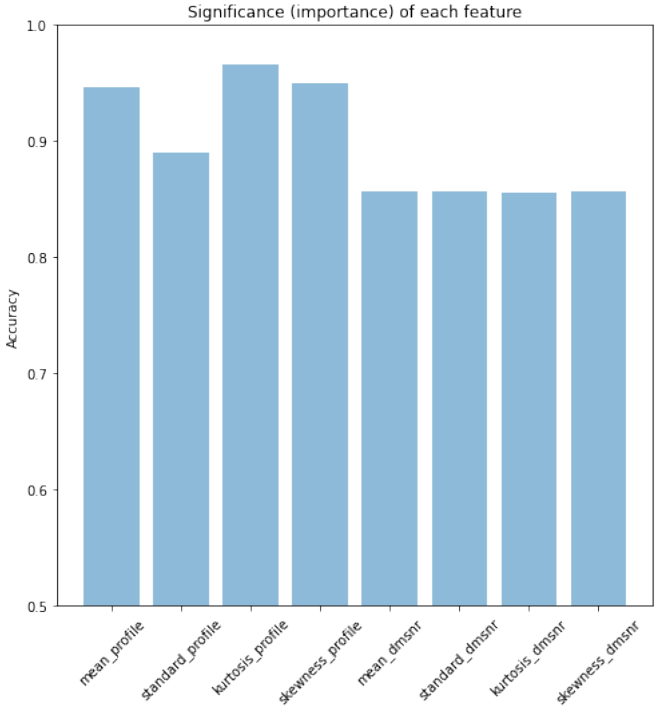

Figure 18: significance of individual features

```
2020/05/05 10:23:59 PM | mean_profile | 0.9467312348668281
2020/05/05 10:24:33 PM | standard_profile | 0.8899236356863476
2020/05/05 10:25:14 PM | kurtosis_profile | 0.9657291860681692
2020/05/05 10:25:34 PM | skewness_profile | 0.9495250512199664
2020/05/05 10:25:50 PM | mean_dmsnr | 0.8569566027193146
2020/05/05 10:26:16 PM | standard_dmsnr | 0.8569566027193146
2020/05/05 10:26:41 PM | kurtosis_dmsnr | 0.8552803129074316
2020/05/05 10:27:12 PM | skewness_dmsnr | 0.8569566027193146
```

Figure 19: significance of individual features data

6.2 Performance after removing less important features

There was no significant drop (at all) in the performance after removing less important features one at a time. We can see the graph below for the performance.

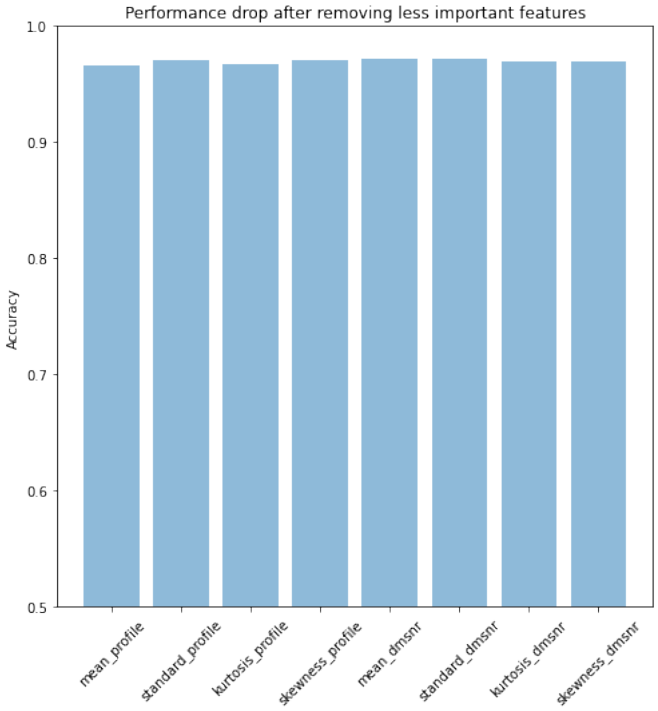

Figure 20: performance after removing less important features

```
2020/05/05 10:27:33 PM | mean_profile | 0.9662879493387968
2020/05/05 10:28:27 PM | standard_profile | 0.9707580555038182
2020/05/05 10:29:46 PM | kurtosis_profile | 0.9672192214565096
2020/05/05 10:30:26 PM | skewness_profile | 0.9701992922331906
2020/05/05 10:31:24 PM | mean_dmsnr | 0.9715030731979885
2020/05/05 10:32:10 PM | standard_dmsnr | 0.9713168187744459
2020/05/05 10:32:58 PM | kurtosis_dmsnr | 0.9696405289625628
2020/05/05 10:33:40 PM | skewness_dmsnr | 0.9688955112683926
```

Figure 21: performance after removing less important features data

6.3 Performance of two features

Even after removing two input features, there was no significant drop (at all) in the performance after removing less important features one at a time. We can see the graph below for the performance.

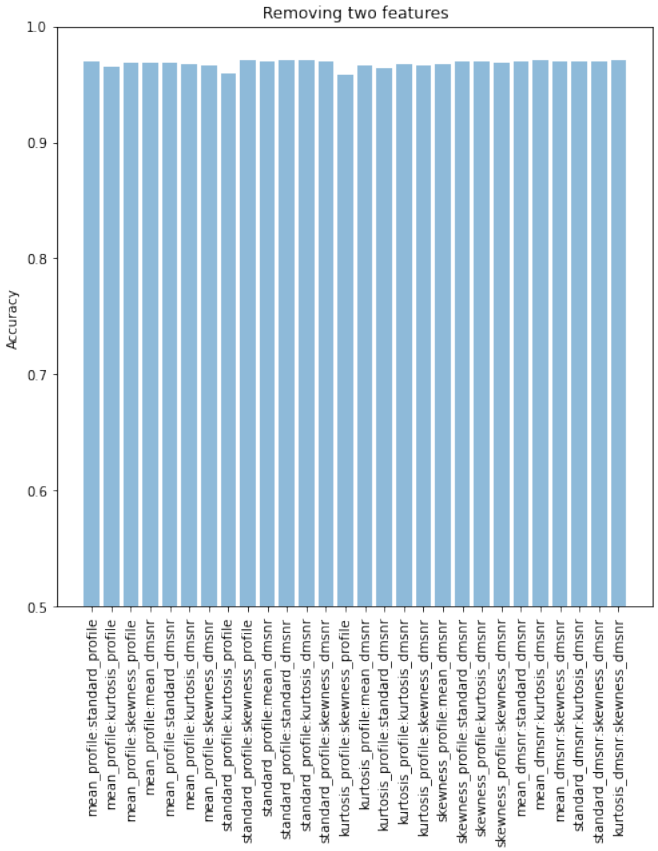

Figure 22: performance of two features

```
2020/05/05 10:34:13 PM | mean_profile:standard_profile | 0.9694542745390203
2020/05/05 10:35:08 PM | mean_profile:kurtosis_profile | 0.9647979139504563
2020/05/05 10:36:11 PM | mean_profile:skewness_profile | 0.9681504935742223
2020/05/05 10:36:48 PM | mean_profile:mean_dmsnr | 0.9685230024213075
2020/05/05 10:37:16 PM | mean_profile:standard_dmsnr | 0.9681504935742223
2020/05/05 10:37:53 PM | mean_profile:kurtosis_dmsnr | 0.9674054758800521
2020/05/05 10:38:13 PM | mean_profile:skewness_dmsnr | 0.9661016949152542
2020/05/05 10:38:30 PM | standard_profile:kurtosis_profile | 0.9593965356677221
2020/05/05 10:39:12 PM | standard_profile:skewness_profile | 0.9705718010802756
2020/05/05 10:39:54 PM | standard_profile:mean_dmsnr | 0.9692680201154777
2020/05/05 10:40:50 PM | standard_profile:standard_dmsnr | 0.9707580555038182
2020/05/05 10:41:36 PM | standard_profile:kurtosis_dmsnr | 0.970385546656733
2020/05/05 10:42:35 PM | standard_profile:skewness_dmsnr | 0.9692680201154777
2020/05/05 10:43:00 PM | kurtosis_profile:skewness_profile | 0.9584652635500093
2020/05/05 10:44:07 PM | kurtosis_profile:mean_dmsnr | 0.9662879493387968
2020/05/05 10:44:53 PM | kurtosis_profile:standard_dmsnr | 0.9634941329856584
2020/05/05 10:46:06 PM | kurtosis_profile:kurtosis_dmsnr | 0.9674054758800521
2020/05/05 10:47:10 PM | kurtosis_profile:skewness_dmsnr | 0.9664742037623394
2020/05/05 10:47:56 PM | skewness_profile:mean_dmsnr | 0.9672192214565096
2020/05/05 10:48:37 PM | skewness_profile:standard_dmsnr | 0.9690817656919352
2020/05/05 10:49:11 PM | skewness_profile:kurtosis_dmsnr | 0.9690817656919352
2020/05/05 10:49:53 PM | skewness_profile:skewness_dmsnr | 0.9688955112683926
2020/05/05 10:51:11 PM | mean_dmsnr:standard_dmsnr | 0.970013037809648
2020/05/05 10:52:14 PM | mean_dmsnr:kurtosis_dmsnr | 0.9701992922331906
2020/05/05 10:53:23 PM | mean_dmsnr:skewness_dmsnr | 0.9698267833861054
2020/05/05 10:54:12 PM | standard_dmsnr:kurtosis_dmsnr | 0.9698267833861054
2020/05/05 10:55:12 PM | standard_dmsnr:skewness_dmsnr | 0.9690817656919352
2020/05/05 10:56:25 PM | kurtosis_dmsnr:skewness_dmsnr | 0.9711305643509033
```

Figure 23: performance of two features data

7 Verify Model with XGBoost

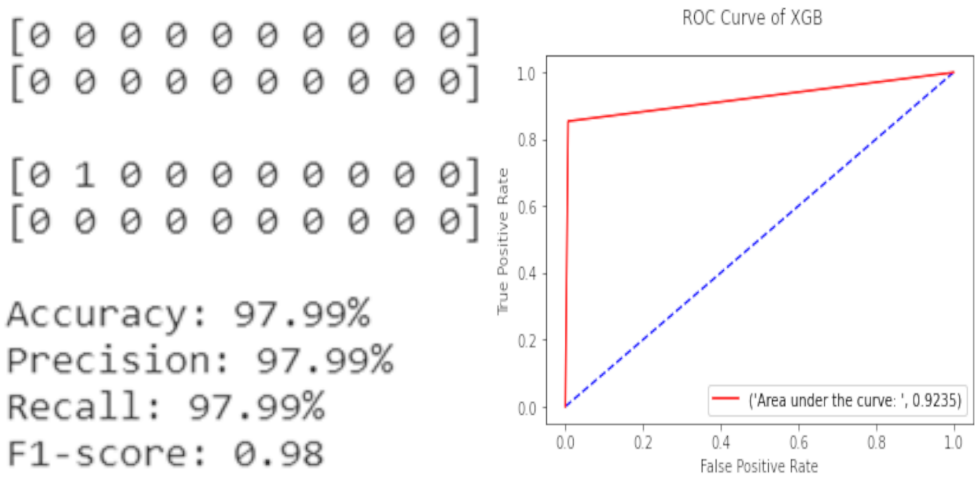

Figure 24: XGB Evaluation and ROC

8 Challenges Faced

In the early phase of this project, I didn't use an early stopping technique which in consequence causing training error and time. In addition, I also faced some challenges in determining whether all the neural networks were sufficient. To solve this dilemma, I did a little research on the receiver operating characteristic curve (ROC) and confusion matrix.

9 Future Improvement

As we can see above, XGBoost still outperforms the neural network model. For this purpose, we may able to improve the performance of the neural network by doing cross-validation, more hyperparameter tuning afterward, and apply an ensemble algorithm [6]. And as always, all the project phases can be reproduced using the Google Colab notebook or in the GitHub repository.

- **[Google Colab]** `https://colab.research.google.com/drive/1cOrM_J8jWFs0k9VgNbJEcGPoJ74ZGF-m`
- **[GitHub]** `https://github.com/zegster/artificial-intelligence`

10 Conclusion

To summarize, the motivation of this project is to try to developed a neural network model to predict the class of pulsars with binary classification. Binary classification is a machine learning technique that categorizes whether the element is either true or false OR 1 or 0. During the training phase, I tested different activation function pattern and recorded their effects on the performances. In the end, XGBoost model still out performance a neural network. Thus, further improvement of this project is required for a better result by doing cross-validation, more hyperparameter tuning afterward, and apply an ensemble algorithm.

References

[1] Calla Cofield. *What Are Pulsars?* Apr. 2016. URL: https://www.space.com/32661-pulsars.html. (accessed: 2.20.2020).

[2] Guest Contributor. *Understanding ROC Curves with Python*. Feb. 2019. URL: https://stackabuse.com/understanding-roc-curves-with-python/. (accessed: 3.16.2020).

[3] Hong Jing. *Why Linear Regression is not suitable for Classification*. May 2019. URL: https://jinglescode.github.io/datascience/2019/05/07/why-linear-regression-is-not-suitable-for-classification/. (accessed: 3.16.2020).

[4] Dr Robert Lyon. *HTRU2 Data Set*. Feb. 2017. URL: https://archive.ics.uci.edu/ml/datasets/HTRU2. (accessed: 2.20.2020).

[5] Pavan Raj. *Predicting a Pulsar Star*. May 2018. URL: https://www.kaggle.com/pavanraj159/predicting-a-pulsar-star. (accessed: 2.20.2020).

[6] Vadim Smolyakov. *Ensemble Learning to Improve Machine Learning Results*. Aug. 2017. URL: https://blog.statsbot.co/ensemble-learning-d1dcd548e936. (accessed: 5.5.2020).