Predicting Pulsar Stars
Model Selection and Evaluation

Duc Ngo - CS4300

March 17, 2020

Contents

1 Introduction

What is a Pulsar? From Earth, 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 the sun. [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. We study pulsars because they are so oddly fascinating and so different from anything we 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 slow and tedious work. With this in mind, the motivation of this project is to try to write supervised learning algorithms to predict the class of pulsars with logistic regression. Logistic regression is a measurement of the relationship between the categorical dependent variable and one or more independent variables.

Google Colab `https://colab.research.google.com/drive/1yscRn_wJqhPNDB4SjhWdvAQ3PSFghxzr`

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.

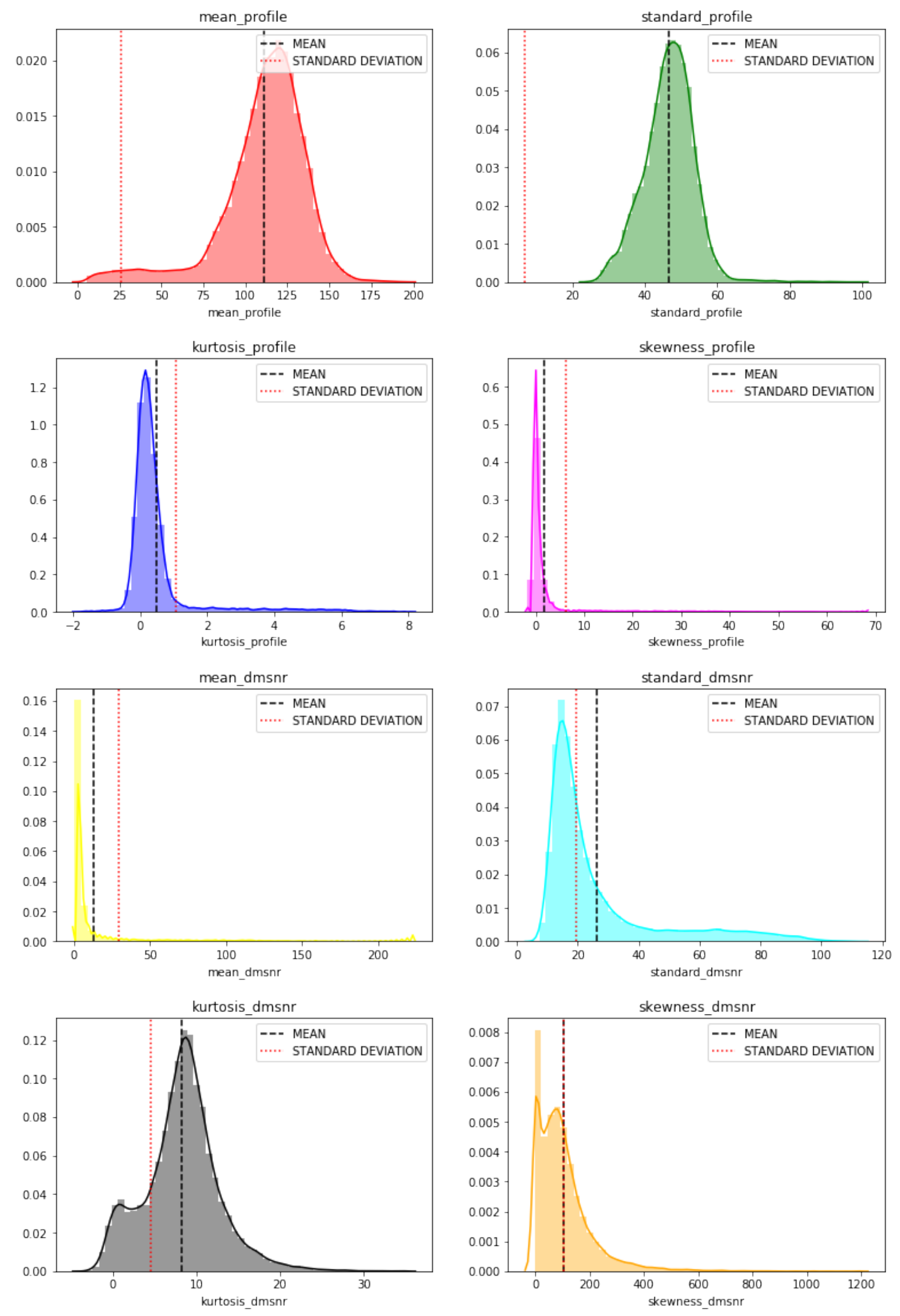
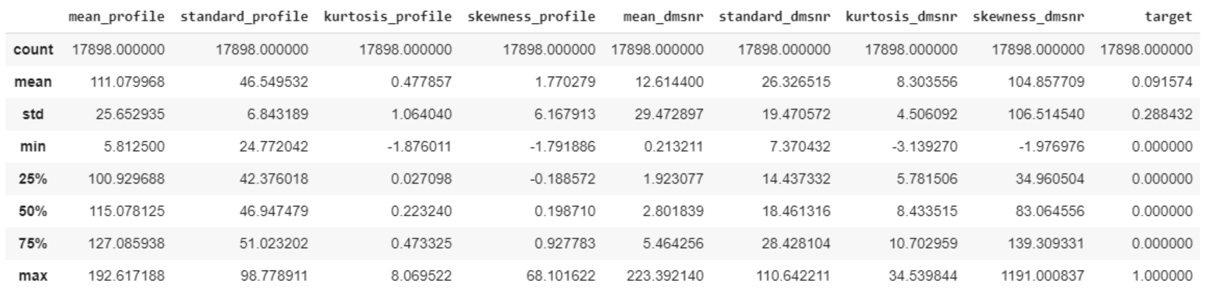


Figure 1: Input Data Distribution Histograms



	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.

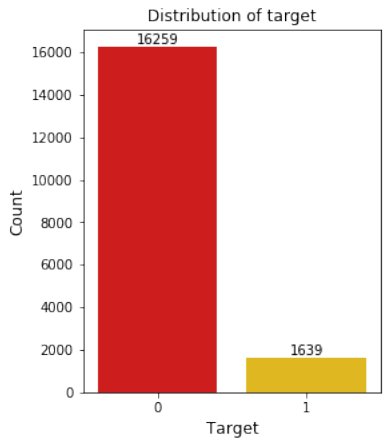


Figure 3: Output Data Distribution Histogram

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

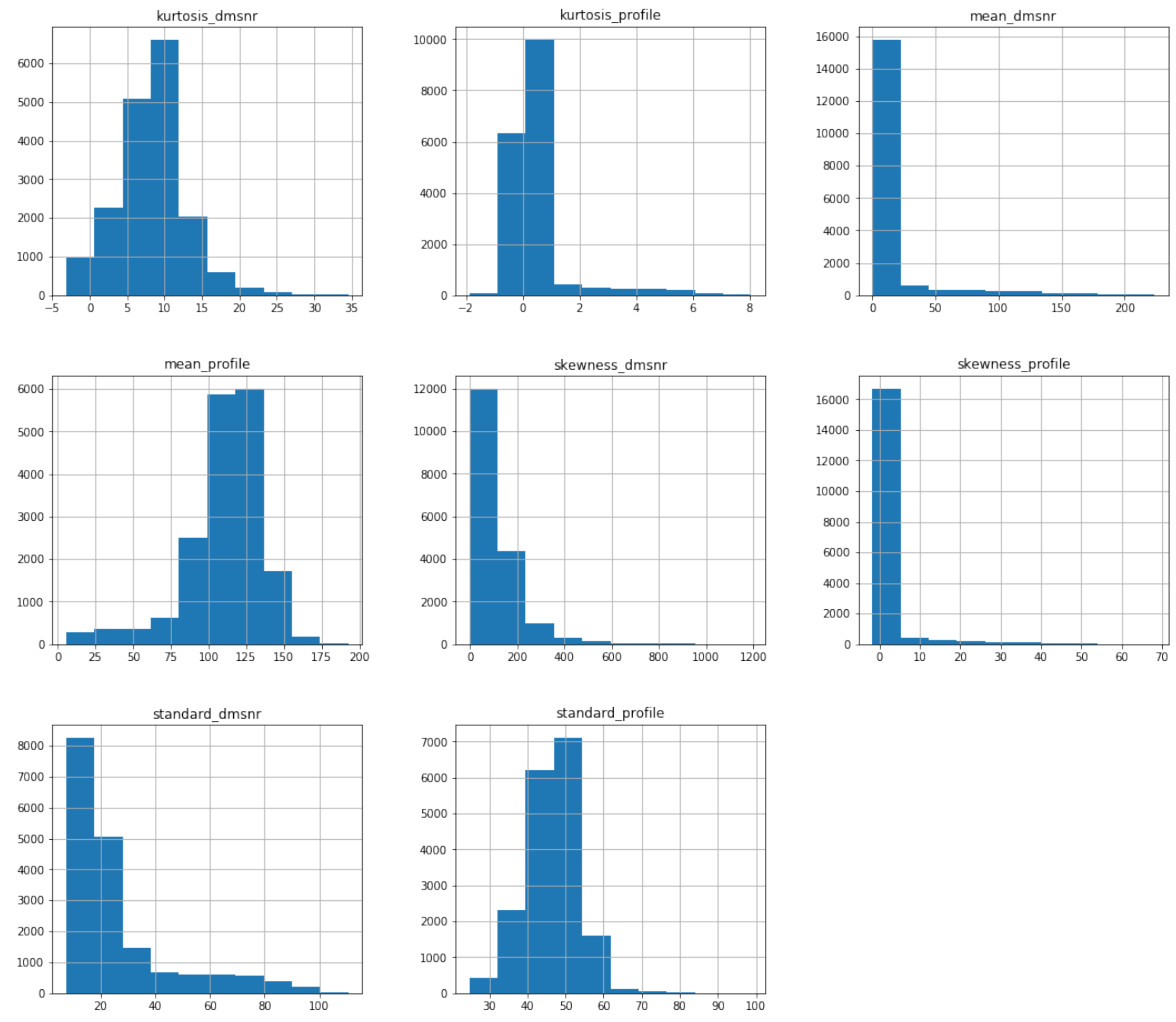


Figure 4: Input Data Before Normalization (Min-Max)

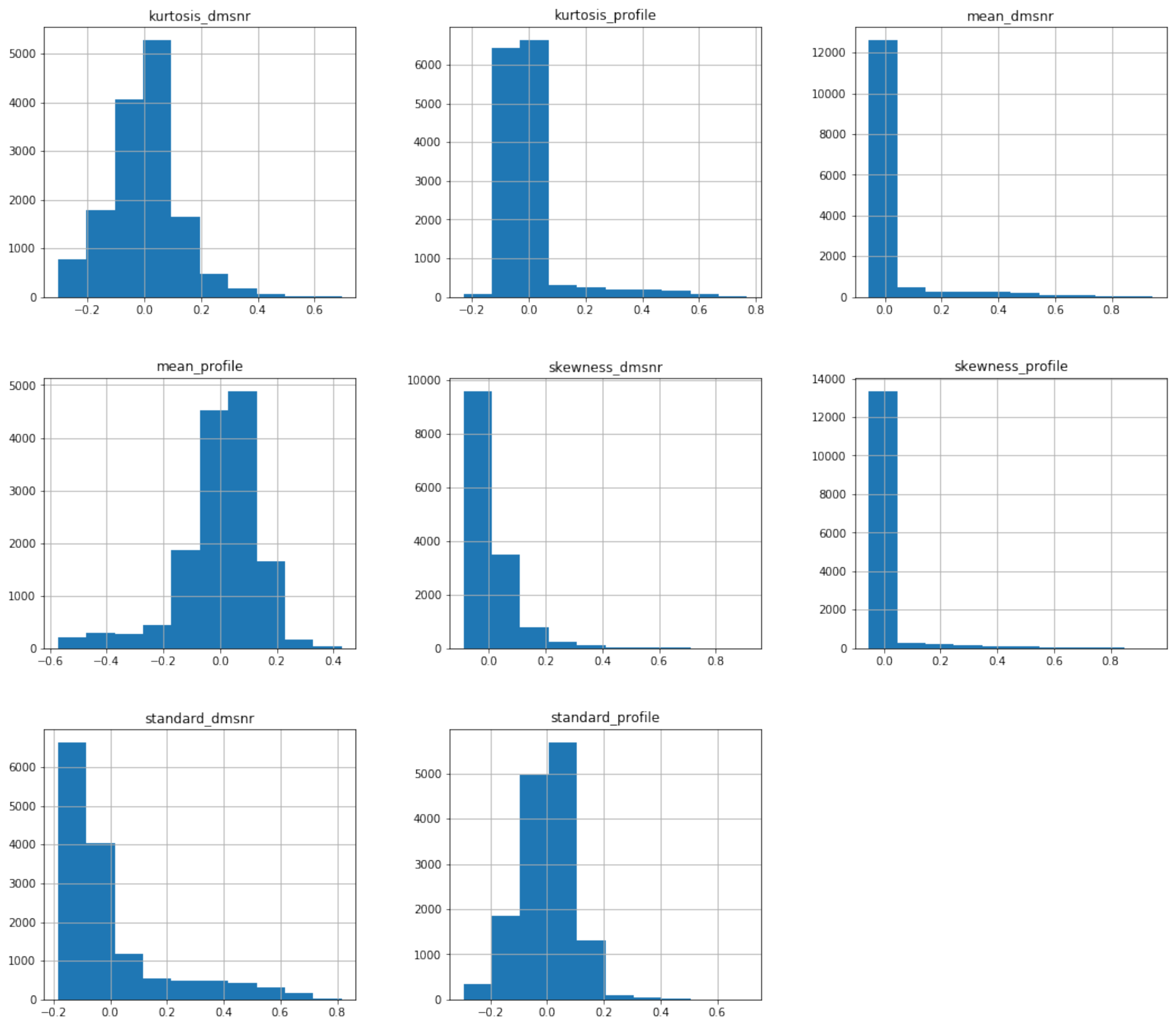


Figure 5: Input Data After Normalization (Min-Max)

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 March 17, 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.

4.1.1 Baseline Model Performance

Hidden	Accuracy	Loss
0 Layer	98.07	6.68
1 Layer	98.16	6.46
2 Layer	98.16	6.32

Table 1: performance comparison for different hidden layers

As can be seen from the above table, the basic architecture with just one hidden layer overall is performing better than other architecture with zero or multiple hidden layers for all datasets. Thus, only one hidden layer will be applied to other architecture.

4.2 Learning Curve of Baseline Model

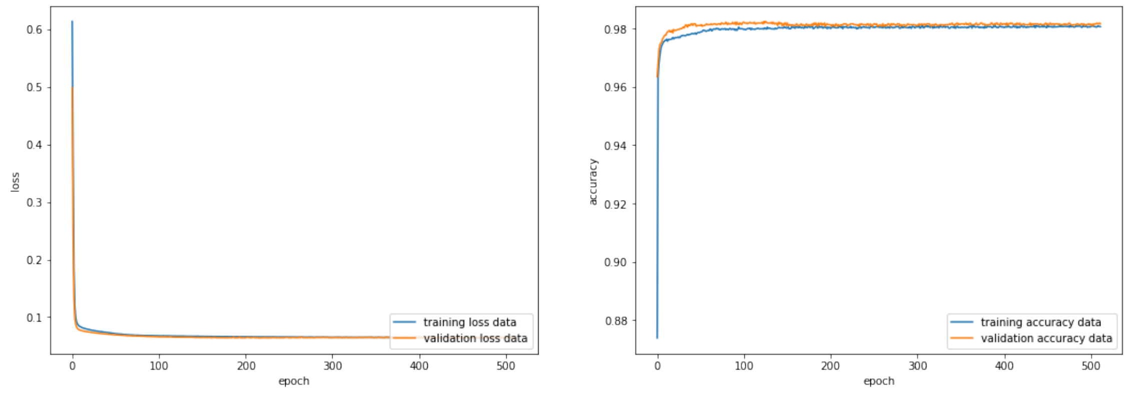


Figure 6: curve showing change in loss/accuracy vs epoch

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)	_	3.39	1.48
Linear (All)	_	9.28	2.52
Sigmoid (Output)	98.10	_	6.74
Sigmoid (All)	98.13	_	6.73

Table 2: performance comparison for different activation function

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

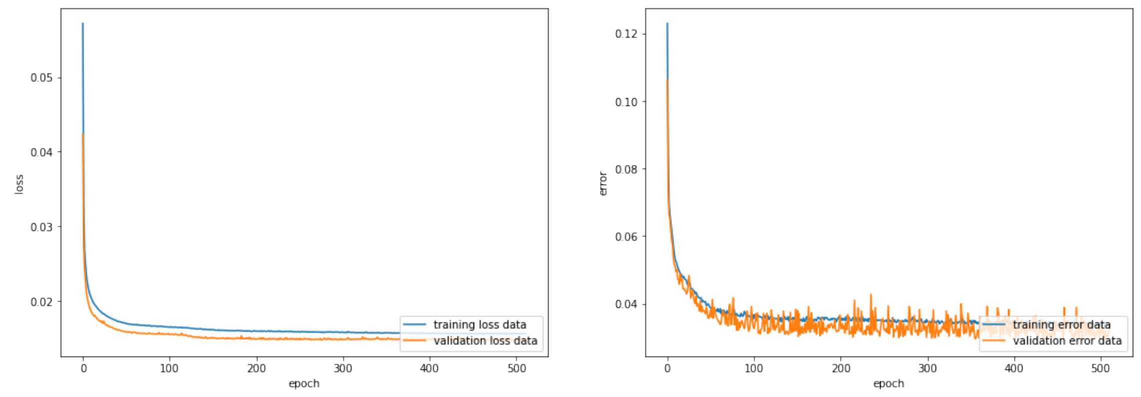


Figure 7: curve showing change in loss/error vs epoch

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

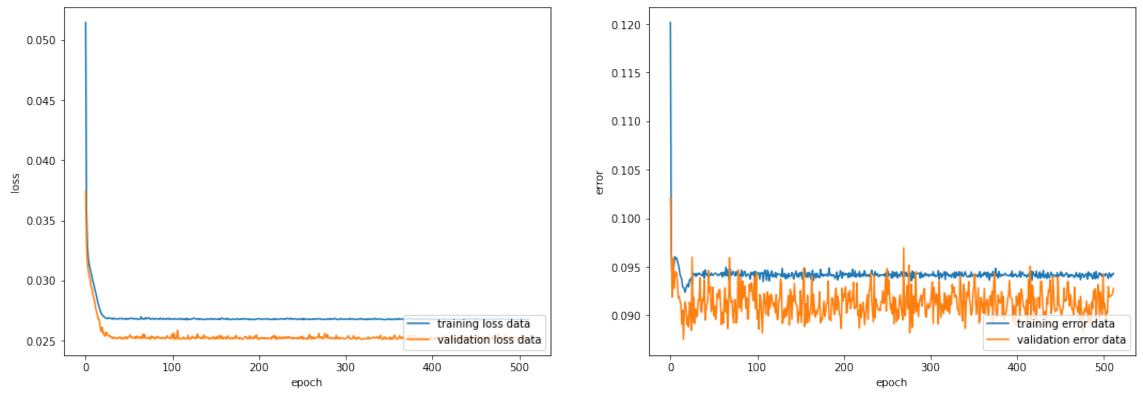


Figure 8: curve showing change in loss/error vs epoch

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

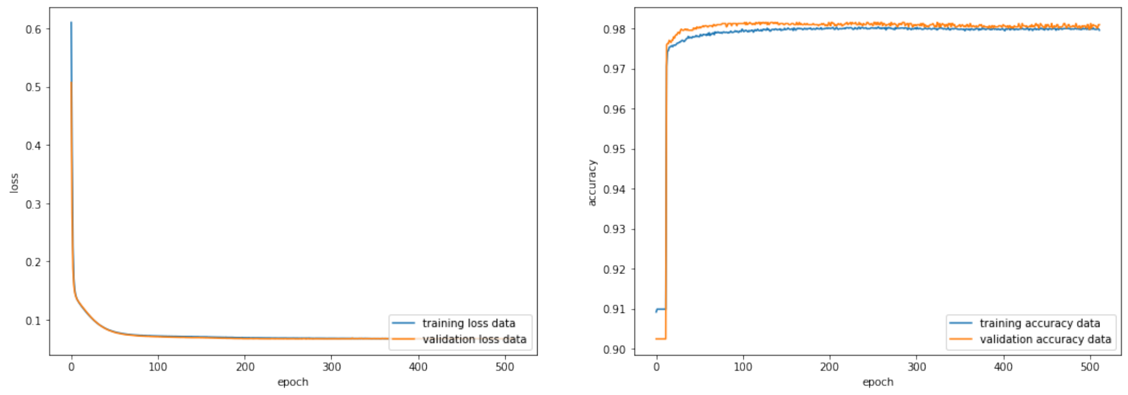


Figure 9: curve showing change in loss/accuracy vs epoch

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

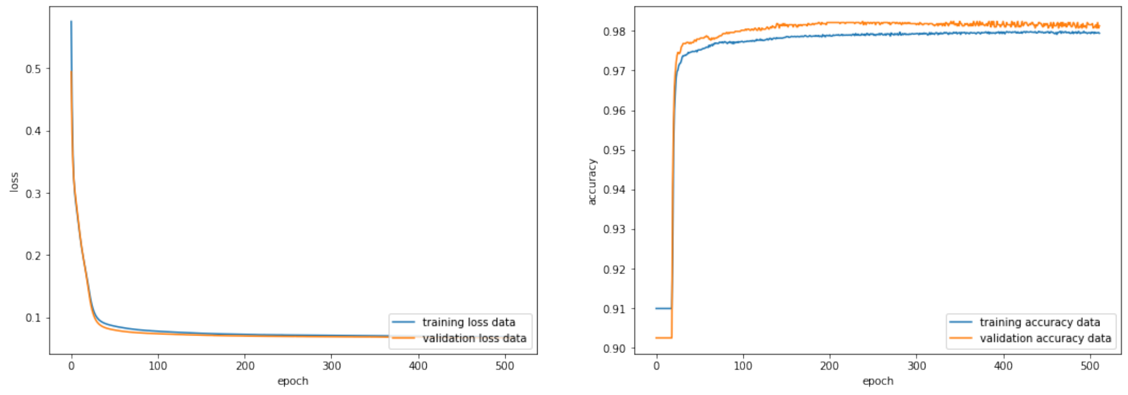


Figure 10: curve showing change in loss/accuracy vs epoch

4.8 Overfitting

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

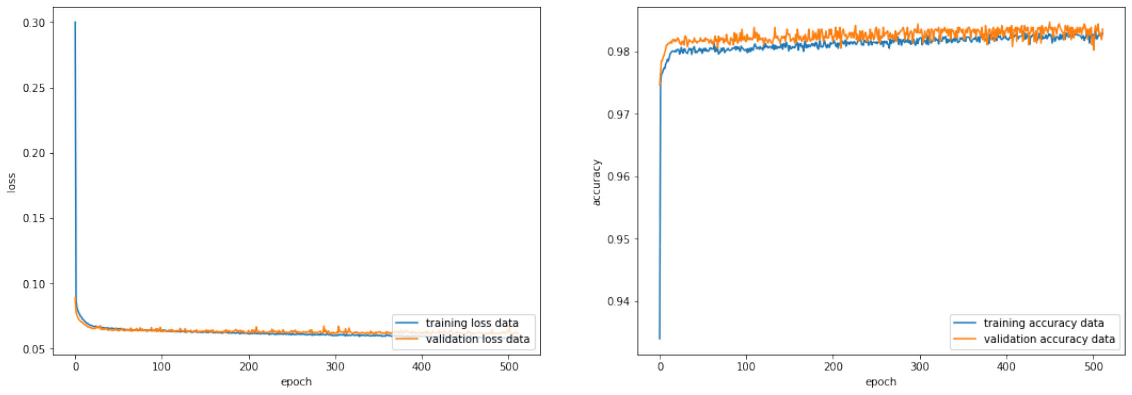


Figure 11: curve showing change in loss/accuracy vs epoch

	Accuracy	Loss
Overfitting	98.35	6.14

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	94.08	86.53	0.90
Linear (last)	94.14	87.39	0.91
Linear (all)	96.31	74.79	0.84
Sigmoid (last)	93.50	86.53	0.90
Sigmoid (all)	93.79	86.53	0.90
Overfitting	94.48	88.25	0.91

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. A perfect test 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.

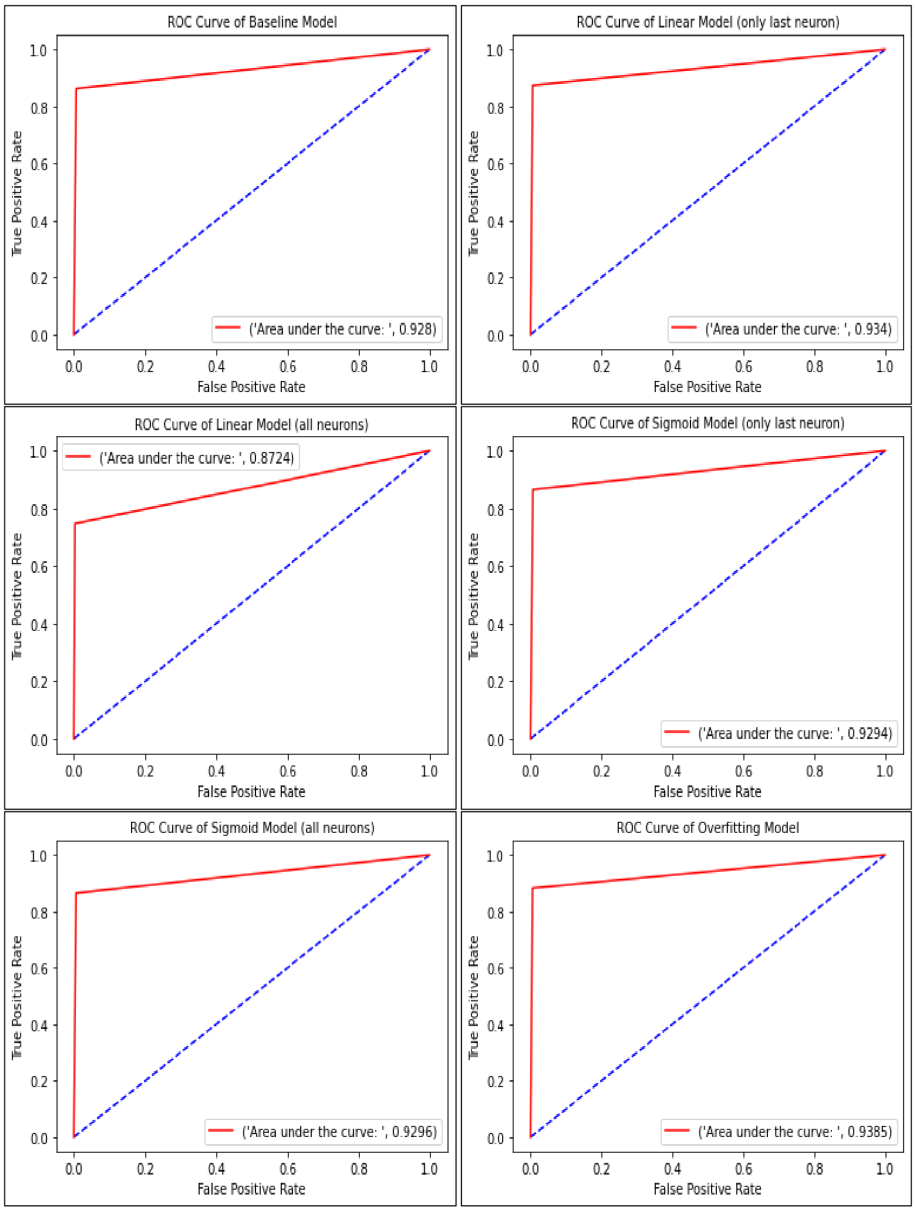
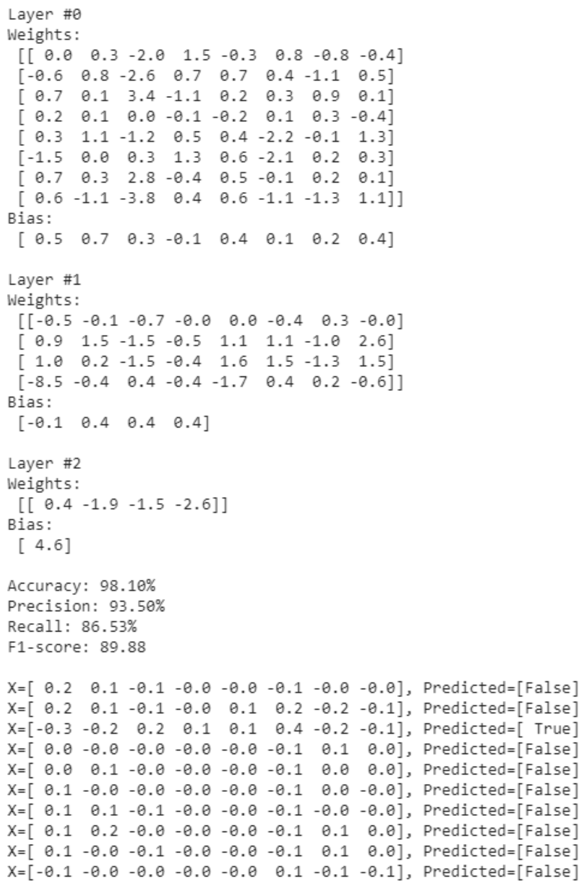


Figure 12: Receiver Operating Characteristic Curve for all model

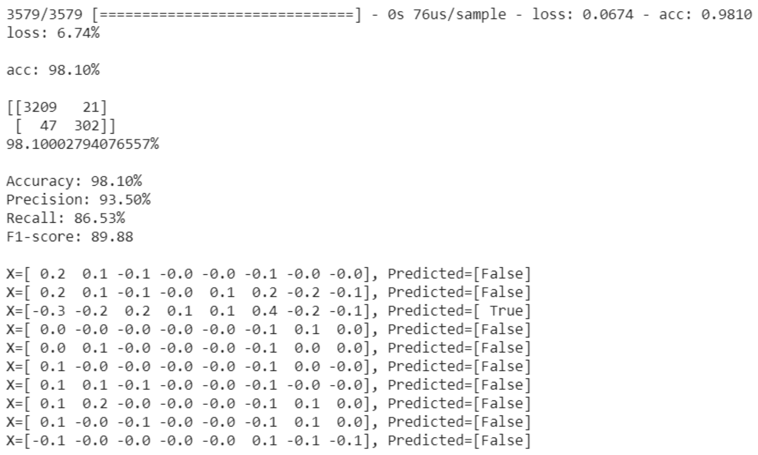
5.2 Custom Function and Keras Function

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:
 [[ 0.0  0.3 -2.0  1.5 -0.3  0.8 -0.8 -0.4]
 [-0.6  0.8 -2.6  0.7  0.7  0.4 -1.1  0.5]
 [ 0.7  0.1  3.4 -1.1  0.2  0.3  0.9  0.1]
 [ 0.2  0.1  0.0 -0.1 -0.2  0.1  0.3 -0.4]
 [ 0.3  1.1 -1.2  0.5  0.4 -2.2 -0.1  1.3]
 [-1.5  0.0  0.3  1.3  0.6 -2.1  0.2  0.3]
 [ 0.7  0.3  2.8 -0.4  0.5 -0.1  0.2  0.1]
 [ 0.6 -1.1 -3.8  0.4  0.6 -1.1 -1.3  1.1]]
Bias:
 [ 0.5  0.7  0.3 -0.1  0.4  0.1  0.2  0.4]

Layer #1
Weights:
 [[-0.5 -0.1 -0.7 -0.0  0.0 -0.4  0.3 -0.0]
 [ 0.9  1.5 -1.5 -0.5  1.1  1.1 -1.0  2.6]
 [ 1.0  0.2 -1.5 -0.4  1.6  1.5 -1.3  1.5]
 [-8.5 -0.4  0.4 -0.4 -1.7  0.4  0.2 -0.6]]
Bias:
 [-0.1  0.4  0.4  0.4]

Layer #2
Weights:
 [[ 0.4 -1.9 -1.5 -2.6]]
Bias:
 [ 4.6]

Accuracy: 98.10%
Precision: 93.50%
Recall: 86.53%
F1-score: 89.88

X=[ 0.2  0.1 -0.1 -0.0 -0.0 -0.1 -0.0 -0.0], Predicted=[False]
X=[ 0.2  0.1 -0.1 -0.0  0.1  0.2 -0.2 -0.1], Predicted=[False]
X=[-0.3 -0.2  0.2  0.1  0.1  0.4 -0.2 -0.1], Predicted=[ True]
X=[ 0.0 -0.0 -0.0 -0.0 -0.0 -0.1  0.1  0.0], 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 -0.1 -0.0 -0.0 -0.1 -0.0 -0.0], Predicted=[False]
X=[ 0.1  0.2 -0.0 -0.0 -0.0 -0.1  0.1  0.0], Predicted=[False]
X=[ 0.1 -0.0 -0.1 -0.0 -0.0 -0.1  0.1  0.0], Predicted=[False]
X=[-0.1 -0.0 -0.0 -0.0 -0.0  0.1 -0.1 -0.1], Predicted=[False]
```

Figure 13: custom function result



```
3579/3579 [==============================] - 0s 76us/sample - loss: 0.0674 - acc: 0.9810
loss: 6.74%

acc: 98.10%

[[3209   21]
 [  47  302]]
98.10002794076557%

Accuracy: 98.10%
Precision: 93.50%
Recall: 86.53%
F1-score: 89.88

X=[ 0.2  0.1 -0.1 -0.0 -0.0 -0.1 -0.0 -0.0], Predicted=[False]
X=[ 0.2  0.1 -0.1 -0.0  0.1  0.2 -0.2 -0.1], Predicted=[False]
X=[-0.3 -0.2  0.2  0.1  0.1  0.4 -0.2 -0.1], Predicted=[ True]
X=[ 0.0 -0.0 -0.0 -0.0 -0.0 -0.1  0.1  0.0], 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 -0.1 -0.0 -0.0 -0.1 -0.0 -0.0], Predicted=[False]
X=[ 0.1  0.2 -0.0 -0.0 -0.0 -0.1  0.1  0.0], Predicted=[False]
X=[ 0.1 -0.0 -0.1 -0.0 -0.0 -0.1  0.1  0.0], Predicted=[False]
X=[-0.1 -0.0 -0.0 -0.0 -0.0  0.1 -0.1 -0.1], Predicted=[False]
```

Figure 14: keras function result

6 To be continue...

6.1 To be continue...

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).