Data Scientist Assignment Arya.ai
Binary Classification Using Tensorflow

Chetan V. Deshmukh

Nashik, Maharashtra

chetandeshmukh497@gmail.com

(M: 9168729917, W: 8055910806)

INTRODUCTION

Binary Classification

Binary classification refers to those classification tasks that have two class labels.

Examples include:

- Email spam detection (spam or not).
- Cancer prediction (Detected or Not Detected).
- Conversion prediction (buy or not).

Typically, binary classification tasks involve one class that is the normal state and another class that is the abnormal state.

For example "*not spam*" is the normal state and "*spam*" is the abnormal state. Another example is "*cancer not detected*" is the normal state of a task that involves a medical test and "*cancer detected*" is the abnormal state.

The class for the normal state is assigned the class label 0 and the class with the abnormal state is assigned the class label 1.

APPROACH

The goal of a binary classification problem is to create a machine learning model that makes a prediction in situations where the thing to predict can take one of just two possible values. For example, we want to predict whether a person is male (0) or female (1) based on predictor variables such as age, income, height, political party affiliation, and so on.

There are many different techniques we can use for a binary classification problem. These techniques include logistic regression, k-NN (if all predictors are numeric), naive Bayes (if all predictors are non-numeric), support vector machines (rarely used any more), decision trees and random forest, and many others. My favourite technique is to use a standard neural network.

ASSUMPTION

As the training data consists of only two classes, 0 and 1, I have assumed that the model I have built only for classification of these two object categories and thus, the model won't be able to classify other object categories. As the givendata consists of training data and test data (without ground truth), I have assumed that we cannot use any extra training data, thus, haven't used any other data or haven't increased the size.

PREREQUISITES

- Tensorflow
- Google Colaboratory
- Numpy
- Matplotlib
- Scikit-Learn
- Pandas

STEPS

1) Data Exploration and Preparation

First, I have imported Numpy, Pandas and Matplotlib then uploaded the dataset as well. The code snippet given below and it also prints a sample.

```
[ ] # Importing libraries
    import numpy as np
    import pandas as pd
    import matplotlib as plt
    import seaborn as sns
```

```
# Laod and Read Data
F = pd.read_csv('/content/training_set.csv') # Train Dataset
F
```

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X49	X50	X51	X52	X53	X54	X55	X56	X57	Y
0	0	0.00	0.00	4.34	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	1.342	0.000	0.000	1.200	2	12	0
1	1	0.00	0.56	0.56	0.00	1.12	0.56	2.25	0.00	0.00	...	0.0	0.083	0.0	0.503	0.000	0.083	16.304	148	375	1
2	2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	0.000	0.000	0.000	1.000	1	5	0
3	3	0.64	0.00	0.64	0.00	1.93	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	0.462	0.370	0.000	2.440	22	122	1
4	4	0.58	0.00	0.00	35.46	0.58	0.00	0.58	0.58	0.00	...	0.0	0.000	0.0	0.239	0.239	0.000	3.338	123	207	1

Here's a look into what the dataset looks like right now

2) Basic Preparation

In this step, we are going to perform EDA on a given dataset. The steps are described below.

2) Basic Data Exploration

Lets See what datatype we have in our dataset

```
# Know the datatypes
F.info()  # Know the datatypes Train Dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3910 entries, 0 to 3909
Data columns (total 59 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   3910 non-null   int64
 1   X1           3910 non-null   float64
 2   X2           3910 non-null   float64
 3   X3           3910 non-null   float64
 4   X4           3910 non-null   float64
 5   X5           3910 non-null   float64
```

As we can see the colomn X56 nad X57 are of integer datatype. So I have change it to float datatype

```
 55  X55         691 non-null    float64
 56  X56         691 non-null    int64
 57  X57         691 non-null    int64
dtypes: float64(55), int64(3)
memory usage: 313.2 KB
```

+ Code + Text

As data type of Coloumn X56 and X57 is of int type. we will change it to float type.

```
[ ] # converting int datatype to float
    F = F.astype({'X56':'float','X57':'float'})
    G = G.astype({'X56':'float','X57':'float'})
```

```
 54  X54         3910 non-null   float64
 55  X55         3910 non-null   float64
 56  X56         3910 non-null   float64
 57  X57         3910 non-null   float64
 58  Y           3910 non-null   int64
dtypes: float64(57), int64(2)
memory usage: 1.8 MB
```

Step 2

Except ‘Y’ all our column are of float datatype now

To know the shape of dataset

```
[ ]  print("Train Shape",F.shape)
     print("Test Shape",G.shape) # Test Data is without ground truth (i.e. No output colomn)
```

```
Train Shape (3910, 59)
Test Shape (691, 58)
```

Step 3

To know null values in Dataset

```
F.isnull().sum()
```

```
Unnamed: 0    0
X1            0
X2            0
X3            0
X4            0
X5            0
X6            0
X7            0
X8            0
```

Step 4

There is no null values in our dataset. Also we will delete a Unnamed coloumn from both training and Test Dataste as it is irrelevent to the given data.

```
F = F.drop(columns=['Unnamed: 0']) # delete coloumn unnamed:0
F
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X49	X50	X51	X52	X53	X54	X55	X56	X57	Y
0	0.00	0.00	4.34	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	1.342	0.000	0.000	1.200	2.0	12.0	0
1	0.00	0.56	0.56	0.00	1.12	0.56	2.25	0.00	0.00	0.56	...	0.0	0.083	0.0	0.503	0.000	0.083	16.304	148.0	375.0	1
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	0.000	0.000	0.000	1.000	1.0	5.0	0
3	0.64	0.00	0.64	0.00	1.93	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	0.462	0.370	0.000	2.440	22.0	122.0	1
4	0.58	0.00	0.00	35.46	0.58	0.00	0.58	0.58	0.00	0.00	...	0.0	0.000	0.0	0.239	0.239	0.000	3.338	123.0	207.0	1

Step 5

Lets perform described method which help us to see how data is spread for numerical values.

```
F.describe()
```

	X10	...	X49	X50	X51	X52	X53	X54	X55	X56	X57	Y
	910.000000	...	3910.000000	3910.000000	3910.000000	3910.000000	3910.000000	3910.000000	3910.000000	3910.000000	3910.000000	3910.000000
	0.244345	...	0.037493	0.139252	0.015876	0.272971	0.077820	0.043828	5.047150	52.338107	283.059079	0.392327
	0.667065	...	0.235054	0.276309	0.083600	0.858634	0.256991	0.452862	31.397035	204.445218	578.339858	0.488331
	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000

Step 6

3) Scale the Data

The dataset contains columns with values that are of different scales and hence not uniform or close enough. I may end up confusing the neural network, if I leave the dataset like this. Here, I need to scale the data. I have use MinmaxScaler from Scikit-Learn to fit and transform the data to make it ready for the model.

3) Preprocessing

From summery we can see that the mean value of feature X55, X57, X58 is high as compared to other feature. So we have to perform Normalization.

```
# Preprocessing allows us to noramalize our data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
F[["ScaledX55", "ScaledX56","ScaledX57"]] = scaler.fit_transform(F[["X55", "X56",'X57']])
F
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X52	X53	X54	X55	X56	X57	Y	ScaledX55	ScaledX56	ScaledX57
0	0.00	0.00	4.34	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	1.342	0.000	0.000	1.200	2.0	12.0	0	0.000182	0.000100	0.001093
1	0.00	0.56	0.56	0.00	1.12	0.56	2.25	0.00	0.00	0.56	...	0.503	0.000	0.083	16.304	148.0	375.0	1	0.013894	0.014718	0.037173
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.000	1.000	1.0	5.0	0	0.000000	0.000000	0.000398
3	0.64	0.00	0.64	0.00	1.93	0.00	0.00	0.00	0.00	0.00	...	0.462	0.370	0.000	2.440	22.0	122.0	1	0.001307	0.002103	0.012027

Step 7

As you can see that the values have been scaled to a relative closer range which is now ready for our neural network.

Removing old version of scaled feature

```
# use drop method
df = F.drop(["X55", "X56",'X57'], axis = 1)
df
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X49	X50	X51	X52	X53	X54	Y	ScaledX55	ScaledX56	ScaledX57
0	0.00	0.00	4.34	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	1.342	0.000	0.000	0	0.000182	0.000100	0.001093
1	0.00	0.56	0.56	0.00	1.12	0.56	2.25	0.00	0.00	0.56	...	0.0	0.083	0.0	0.503	0.000	0.083	1	0.013894	0.014718	0.037173
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	0.000	0.000	0.000	0	0.000000	0.000000	0.000398
3	0.64	0.00	0.64	0.00	1.93	0.00	0.00	0.00	0.00	0.00	...	0.0	0.000	0.0	0.462	0.370	0.000	1	0.001307	0.002103	0.012027
4	0.58	0.00	0.00	35.46	0.58	0.00	0.58	0.58	0.00	0.00	...	0.0	0.000	0.0	0.239	0.239	0.000	1	0.002123	0.012215	0.020475

Step 8

So now our dataset looks like this after all the transformation and changes and now we will move on to the next phase where I will start model training.

4) Training the classification model

I have assigned dependent feature to Y and independent features to X.

4) Training Of Model

Assigning Train and Validation Dataset

```
[ ]  Y = df['Y']
     X = df.drop(['Y'],axis = 1)
```

```
[ ] X
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X48	X49	X50	X51	X52	X53	X54	ScaledX55	ScaledX56	ScaledX57
0	0.00	0.00	4.34	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.0	0.000	0.0	1.342	0.000	0.000	0.000182	0.000100	0.001093
1	0.00	0.56	0.56	0.00	1.12	0.56	2.25	0.00	0.00	0.56	...	0.0	0.0	0.083	0.0	0.503	0.000	0.083	0.013894	0.014718	0.037173
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	...	0.0	0.0	0.000	0.0	0.000	0.000	0.000	0.000000	0.000000	0.000398
3	0.64	0.00	0.64	0.00	1.93	0.00	0.00	0.00	0.00	0.00	...	0.0	0.0	0.000	0.0	0.462	0.370	0.000	0.001307	0.002103	0.012027
4	0.58	0.00	0.00	35.46	0.58	0.00	0.58	0.58	0.00	0.00	...	0.0	0.0	0.000	0.0	0.239	0.239	0.000	0.002123	0.012215	0.020475

```
Y
```

```
0       0
1       1
2       0
3       1
4       1
      ..
3905    0
3906    0
3907    0
3908    1
3909    1
Name: Y, Length: 3910, dtype: int64
```

As mentioned in the assignment I have split the training data in training and validation set with ratio of 4:1.

```
[ ] from sklearn.model_selection import train_test_split
    X_train,X_val,y_train,y_val = train_test_split(X,Y,train_size=0.80,random_state=42)
```

The value range is much narrow and hence it is perfect for a neural network and now we move on to training it with Tensorflow

5) Using Tensorflow to train the classification model

The Neural Network

The following architecture was chosen at random and hence you can adjust it to whatever you want to. This model here goes from 12 different input features to the first hidden layer of 128 neurons and then 2 more hidden layers of 256 neurons. Then it ends with 1 neuron at the end and the hidden layers ReLU as the activation function and the output layer is got by using a Sigmoid function. The following code demonstrates it.

```
import tensorflow as tf
tf.random.set_seed(42)


model = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

model.compile(
    loss=tf.keras.losses.binary_crossentropy,
    optimizer=tf.keras.optimizers.Adam(lr=0.001),
    metrics=[
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
)

history = model.fit(X_train, y_train, epochs=200)
```

```
Epoch 190/200
98/98 [==============================] - 0s 3ms/step - loss: 0.0144 - accuracy: 0.9952 - precision: 0.9959 - re
Epoch 191/200
98/98 [==============================] - 0s 3ms/step - loss: 0.0106 - accuracy: 0.9965 - precision: 0.9984 - recall: 0.9927
Epoch 192/200
98/98 [==============================] - 0s 4ms/step - loss: 0.0103 - accuracy: 0.9968 - precision: 0.9992 - recall: 0.9927
Epoch 193/200
98/98 [==============================] - 0s 3ms/step - loss: 0.0083 - accuracy: 0.9968 - precision: 0.9992 - recall: 0.9927
Epoch 194/200
98/98 [==============================] - 0s 3ms/step - loss: 0.0086 - accuracy: 0.9968 - precision: 0.9984 - recall: 0.9935
Epoch 195/200
98/98 [==============================] - 0s 4ms/step - loss: 0.0080 - accuracy: 0.9974 - precision: 0.9984 - recall: 0.9951
Epoch 196/200
98/98 [==============================] - 0s 3ms/step - loss: 0.0082 - accuracy: 0.9971 - precision: 0.9992 - recall: 0.9935
Epoch 197/200
98/98 [==============================] - 0s 3ms/step - loss: 0.0083 - accuracy: 0.9971 - precision: 0.9992 - recall: 0.9935
Epoch 198/200
98/98 [==============================] - 0s 4ms/step - loss: 0.0081 - accuracy: 0.9971 - precision: 0.9984 - recall: 0.9943
Epoch 199/200
98/98 [==============================] - 0s 3ms/step - loss: 0.0082 - accuracy: 0.9965 - precision: 0.9984 - recall: 0.9927
Epoch 200/200
98/98 [==============================] - 0s 3ms/step - loss: 0.0077 - accuracy: 0.9971 - precision: 0.9975 - recall: 0.9951
```

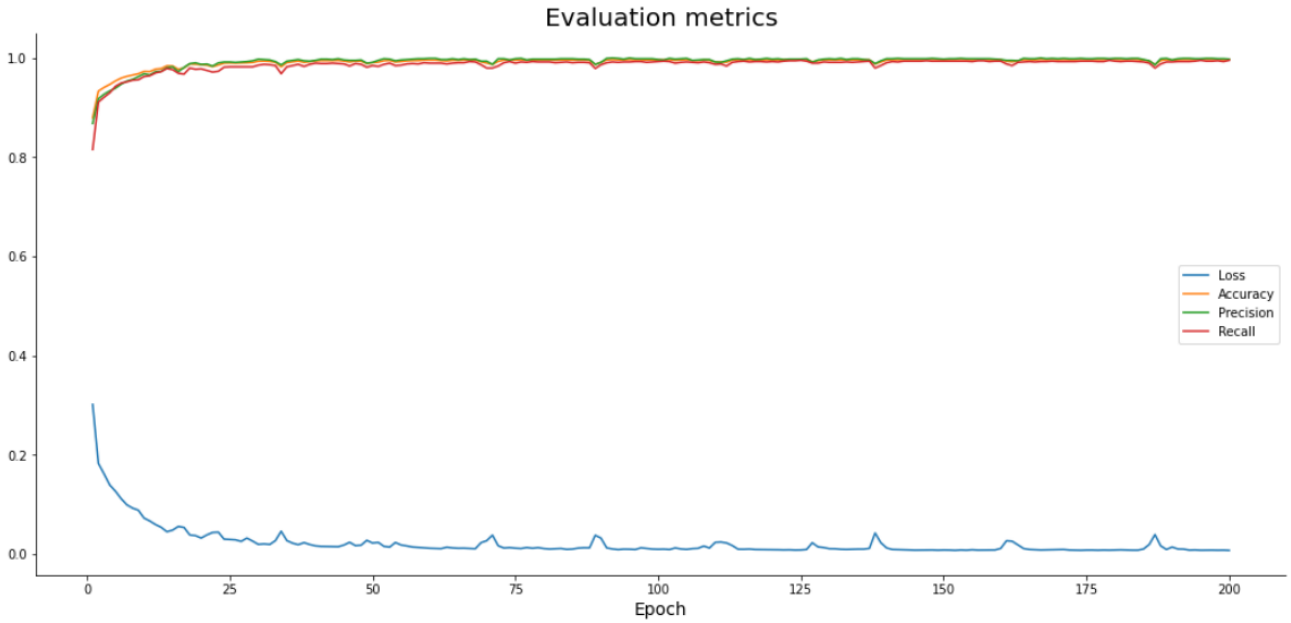
This image shows the final epochs of the model. Each epoch on average takes around 1 second on Google Colab to get trained. I have kept track of the accuracy, loss, precision, and recall function during training and saved them to history.

6) Visualization and Evaluation of the classification model using Tensorflow

```
import matplotlib.pyplot as plt
from matplotlib import rcParams

rcParams['figure.figsize'] = (18, 8)
rcParams['axes.spines.top'] = False
rcParams['axes.spines.right'] = False
```

```
plt.plot(
    np.arange(1, 201),
    history.history['loss'], label='Loss'
)
plt.plot(
    np.arange(1, 201),
    history.history['accuracy'], label='Accuracy'
)
plt.plot(
    np.arange(1, 201),
    history.history['precision'], label='Precision'
)
plt.plot(
    np.arange(1, 201),
    history.history['recall'], label='Recall'
)
plt.title('Evaluation metrics', size=20)
plt.xlabel('Epoch', size=14)
plt.legend();
```



Here in our model, we can see that it is following the trend and loss is decreasing as the other factors are increasing. The important question to solve next is whether if we are overfitting or not?

7) Predictions for Classification Model with Tensorflow

Now we move onto the prediction part where we will use predict() function to predict the output on the scaled data of testing. The following code demonstrates it.

```
predictions = model.predict(X_val)
predictions
```

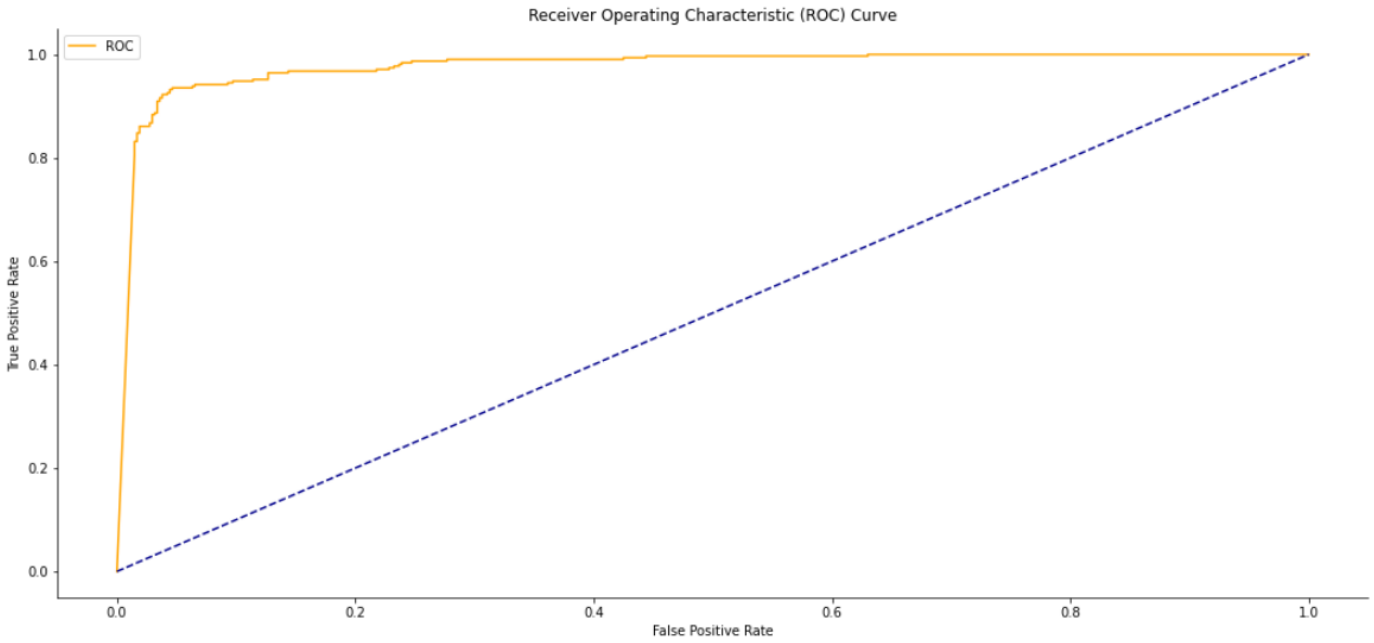
```
array([[1.14020757e-15],
       [0.00000000e+00],
       [1.00000000e+00],
       [0.00000000e+00],
       [2.75552725e-06],
       [0.00000000e+00],
       [9.08510458e-08],
       [1.00000000e+00],
       [1.00000000e+00],
       [7.18325377e-04],
       [8.05016555e-07],
```

I need to convert them to the corresponding classes and the logic is simple as if the result is more than optimal threshold, then I can assign a value of 1 or 0 otherwise. Following code to find the optimal threshold.

```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
# Computing manually fpr, tpr, thresholds and roc auc
fpr, tpr, thresholds = roc_curve(y_val, predictions)
roc_auc = auc(fpr, tpr)
print("ROC_AUC Score : ",roc_auc)
print("Function for ROC_AUC Score : ",roc_auc_score(y_val, predictions)) # Function present
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds[optimal_idx]
print("Threshold value is:", optimal_threshold)
plot_roc_curve(fpr, tpr)
```

```
ROC_AUC Score :  0.976761290940564
Function for ROC_AUC Score :  0.976761290940564
Threshold value is: 0.22272009
```

```
Threshold value is: 0.22272009
```



So now we have found the optimal threshold value, we will proceed to the next step

```
prediction_classes = [1 if prob > 0.22 else 0 for prob in np.ravel(predictions)]
prediction_classes
```

```
0,
0,
1,
1,
0,
0,
1,
1,
0,
1,
1,
0,
0,
1,
```

Now we need to move on to the evaluation of the model. We will begin with the confusion matrix which can be found by the following code.

Model evaluation on test data

```
from sklearn.metrics import confusion_matrix
#print(y_val.shape)
#predictions.shape
print(confusion_matrix(y_val, prediction_classes))
```

```
[[451  22]
 [ 20 289]]
```

Since there are False Negatives 20 and false positives 22, hence we can deduce that the recall value of the test set will be higher than the precision. The below code can be used to print all the details like precision, accuracy, and recall.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score
print(f'Accuracy: {accuracy_score(y_val, prediction_classes):.2f}')
print(f'Precision: {precision_score(y_val, prediction_classes):.2f}')
print(f'Recall: {recall_score(y_val, prediction_classes):.2f}')
```

```
Accuracy: 0.95
Precision: 0.93
Recall: 0.94
```

It is a decent model for a quick build and test. With more epochs and better data exploration, we can further enhance the model.

Prediction on Unseen Dataset

```
predictions = model.predict(Test_Dataset)
predictions
```

```
[0.00000000e+00],
[2.73665327e-17],
[1.00000000e+00],
[6.35432760e-23],
[9.99998212e-01],
[1.00000000e+00],
[7.36602009e-19],
[1.00000000e+00],
[1.57690718e-11],
[1.00000000e+00],
[4.08253074e-03],
[1.00000000e+00],
[5.26605926e-10],
```

```
prediction_classes = [1 if prob > 0.22 else 0 for prob in np.ravel(predictions)]
prediction_classes
```

```
0,
0,
1,
0,
1,
1,
0,
1,
0,
1,
0,
1,
0,
0,
0,
```

CONCLUSION

The problem of Binary Classification is an extremely importantone and can be solved in various ways by adopting various deep learning techniques. Here, I have used standard neural network for performing this task and have achieved decent first-hand results. These results can be easily interpreted by looking at the result. These results can be further improved by tuning the model hyperparameters or even by choosing some other technique.

REFERENCES

Following resources have certainly helped me in getting a first-hand solution for this task:

[1] Beginner's guide on How to Train a Classification Model with Tensorflow: https://www.analyticsvidhya.com/blog/2021/10/beginners-guide-on-how-to-train-a-classification-model-with-tensorflow/#h2_1

[2] Types of Classification Tasks in Machine Learning: https://machinelearningmastery.com/types-of-classification-in-machine-learning/#:~:text=Binary%20classification%20refers%20to%20those,prediction%20(buy%20or%20not).

[3] Binary Classification Example: Binary Classification Example. Predicting Opioid Use | by Kamil Mysiak | Towards Data Science