AI Tensorflow.

Submitted by – Kartik Nagras

Dataset Explanation.

The Dataset is of Bank Customers which includes Information of Customers having attributes such as **Customer_ID ,Surname ,Credit_Score ,Geography ,Gender ,Age ,Tenure_With Bank ,Account Balance Number_Of_Products , Salary,Exited(Is Active Member or not) etc.**

dataset - DataFrame

Index	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
1678	1679	15569178	Kharlamov	570	France	Female	18	4	8.28e+04	1	1	0	7.18e+04	0
7722	7723	15570086	Lynch	684	Germany	Male	18	9	9.05e+04	1	0	1	4.78e+03	0
8522	8523	15619892	Page	644	Spain	Male	18	8	0	2	1	0	5.92e+04	0
2136	2137	15621893	Bellucci	727	France	Male	18	4	1.34e+05	1	1	1	4.69e+04	0
9501	9502	15634146	Hou	835	Germany	Male	18	2	1.43e+05	1	1	1	1.18e+05	0
9572	9573	15641688	Collier	644	Spain	Male	18	7	0	1	0	1	5.96e+04	1
3330	3331	15657439	Chao	738	France	Male	18	4	0	2	1	1	4.78e+04	0
3512	3513	15657779	Boylan	806	Spain	Male	18	3	0	2	1	1	8.7e+04	0
3686	3687	15665327	Cattaneo	706	France	Male	18	2	1.76e+05	2	1	0	1.3e+05	0
9526	9527	15665521	Chiazagomekpele	642	Germany	Male	18	5	1.11e+05	2	0	1	1.01e+04	0
9520	9521	15673180	Onyekaozulu	727	Germany	Female	18	2	9.38e+04	2	1	0	1.26e+05	0
9029	9030	15722701	Bruno	594	Germany	Male	18	1	1.33e+05	1	1	0	1.68e+05	0
9782	9783	15728829	Weigel	509	France	Male	18	7	1.03e+05	1	1	0	1.72e+05	0
3517	3518	15757821	Burgess	771	Spain	Male	18	1	0	2	0	0	4.15e+04	0
2141	2142	15758372	Wallace	674	France	Male	18	7	0	2	1	1	5.58e+04	1
7334	7335	15759133	Vaguine	616	France	Male	18	6	0	2	1	1	2.73e+04	0
1619	1620	15770309	McDonald	656	France	Male	18	10	1.52e+05	1	0	1	1.27e+05	0

Business Question:

The Bank wants to predict which Customers will Leave and which customers will stay ?

The Bank want's us to create a Churn Model for the Bank.

Creating ANN.

We create the Churn Model using 'Keras' with Backend as 'Tensorflow'.

- First we will initialize our Artificial Neural Network.

```
# Initialising the ANN
classifier = Sequential()
```

As this is a classification problem we use classifier and initialize it Sequentially.

Sequential Model is a linear stack of layers.

- Then we add the input layer and first hidden layer.

```
# Adding the input layer and the first hidden layer
classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim = 11))
```

Dense function is used to create Layers i.e.(Input layer and First Hidden Layer) having parameters -

Output_Dim = Number of nodes we want to add in Hidden Layer.

Init = Randomly initialize the weights uniformly close to 0.

Activation = Activation function we want to choose in our Hidden Layer

Input_dim =Number of Independent variables in our dataset.

- Now we add another hidden layer with same number of nodes.

```
# Adding the second hidden layer
classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu'))
```

- Now we add output Layer.

```
# Adding the output layer
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))
```

Here output_dim =1 because we have only one dependent variable or outcome variable i.e Whether the customer will leave the bank or not .
We use sigmoid as activation function because this will give us the probability of customers leaving the bank or not from 0 to 1.

- Now we will compile our ANN.

```
# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

Here, we use compile function to compile our Layers having parameters –
Optimizer – Algorithm to find optimal weights .
Loss – Function to compile ANN.
Metrics – We us 'Accuracy' because we want find the accuracy of our ANN model.

- We will now fit our ANN model to our training set.

```
# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
```

Batch_size = Total number of Training example present in a single batch.
Epoch = Number of times dataset is passed forward and backward.

Results –

```
Epoch 1/100
8000/8000 [==============================] - 2s 287us/step - loss: 0.4788 - acc: 0.7956
Epoch 2/100
8000/8000 [==============================] - 1s 137us/step - loss: 0.4260 - acc: 0.7960
Epoch 3/100
8000/8000 [==============================] - 1s 135us/step - loss: 0.4206 - acc: 0.8050
Epoch 4/100
8000/8000 [==============================] - 1s 135us/step - loss: 0.4176 - acc: 0.8247
Epoch 5/100
8000/8000 [==============================] - 1s 133us/step - loss: 0.4150 - acc: 0.8277
Epoch 6/100
8000/8000 [==============================] - 1s 158us/step - loss: 0.4132 - acc: 0.8290
Epoch 7/100
8000/8000 [==============================] - 1s 153us/step - loss: 0.4113 - acc: 0.8314
Epoch 8/100
8000/8000 [==============================] - 1s 132us/step - loss: 0.4103 - acc: 0.8297
Epoch 9/100
8000/8000 [==============================] - 1s 133us/step - loss: 0.4090 - acc: 0.8326
Epoch 10/100
```

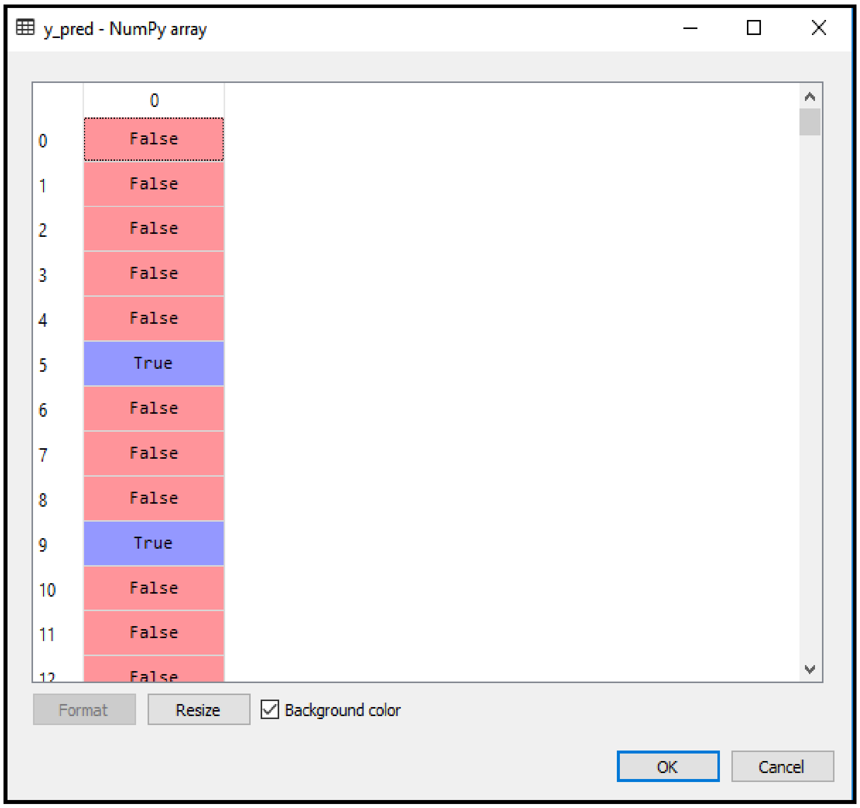
```
Epoch 90/100
8000/8000 [==============================] - 1s 148us/step - loss: 0.3997 - acc: 0.8342
Epoch 91/100
8000/8000 [==============================] - 1s 155us/step - loss: 0.4001 - acc: 0.8355
Epoch 92/100
8000/8000 [==============================] - 1s 182us/step - loss: 0.3994 - acc: 0.8360
Epoch 93/100
8000/8000 [==============================] - 1s 155us/step - loss: 0.3995 - acc: 0.8357
Epoch 94/100
8000/8000 [==============================] - 1s 144us/step - loss: 0.3997 - acc: 0.8365
Epoch 95/100
8000/8000 [==============================] - 1s 153us/step - loss: 0.3998 - acc: 0.8349
Epoch 96/100
8000/8000 [==============================] - 1s 145us/step - loss: 0.3990 - acc: 0.8345
Epoch 97/100
8000/8000 [==============================] - 1s 143us/step - loss: 0.3997 - acc: 0.8361
Epoch 98/100
8000/8000 [==============================] - 1s 135us/step - loss: 0.3996 - acc: 0.8357
Epoch 99/100
8000/8000 [==============================] - 1s 135us/step - loss: 0.3990 - acc: 0.8369
Epoch 100/100
8000/8000 [==============================] - 1s 136us/step - loss: 0.3999 - acc: 0.8344
Out[42]: <keras.callbacks.History at 0x2c9d0dccdd8>
```

Using training dataset to test our ANN model we get accuracy of **79.56%** after 1 epoch cycle,and our accuracy increases to **83.34%** after 100 epoch cycle.

Now we test our model on testing data set, where we convert the probability of customers who will leave or not using a threshold of 0.5 into True and False.

```python
# Part 3 - Making the predictions and evaluating the model

# Predicting the Test set results
# For Customers having probability < 0.5  = Leave the bank(False)
# For Customers having probability > 0.5  = Stay with the bank(True)
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```



We calculate the confusion matrix to evaluate our model to count number of correct predictions vs number of incorrect predictions.

	0	1
0	1553	42
1	273	132

Out of 2000 customers data in our testing dataset we get 1,685 as correct predictions and 315 as incorrect predictions.

Accuracy = Number of correct prediction / Total Data.

=1,685/2000

=84.25%

Conclusion-

Now the Bank has this Churn Model ready, and they use this model to predict which customer will leave the bank or not.

They can apply this model on any number of customers data and they will get accuracy of their prediction of **84%** which is pretty good.