Assignment-5 ANN

October 22, 2018

1 Artificial Neural Network for Customer's Churn Prediction

Churn's prediction could be a great asset in the business strategy for retention applying before the exit of customers and We will create a real model with python, applied on a bank environment. This model will tell us if the customer is going or not to exit from the bank. Churn rate (sometimes called attrition rate), in its broadest sense, is a measure of the number of individuals or items moving out of a collective group over a specific period.

2 What are neural networks?

Neural networks, commonly known as Artificial Neural Networks (ANN) are quite a simulation of human brain functionality in machine learning (ML) problems. ANNs shall be noted not as a solution for all the problems that arise, but would provide better results with many other techniques altogether for various ML tasks. Most common use of ANNs are clustering and classification, which can be used for regression tasks as well.

3 Let's begin

So, In our dataset we would be dealing with Churn Modeling i.e. we would be writing a Artificial Neural Network to find out reasons as to why and which customers are actually leaving the bank and their dependencies on one another. This is a classification problem 0-1 classification(1 if customer Leaves and 0 if customer stays).

```
In [42]: #Lets import our python libraries and our dataset creating X for our independent vari
         # Importing the libraries
         import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
```

```
In [43]: # Importing the dataset
         dataset = pd.read_csv('Bank_Churn.csv')
         dataset.head(5)
```

```
Out[43]:    RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
         0          1    15634602  Hargrave          619    France  Female   42
         1          2    15647311      Hill          608     Spain  Female   41
```

```
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

In [44]:
```
#Looking at the features we can see that row no.,name will have no relation with a cu
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values
```

In [45]:
```
X
```

Out[45]:
```
array([[619, 'France', 'Female', ..., 1, 1, 101348.88],
       [608, 'Spain', 'Female', ..., 0, 1, 112542.58],
       [502, 'France', 'Female', ..., 1, 0, 113931.57],
       ...,
       [709, 'France', 'Female', ..., 0, 1, 42085.58],
       [772, 'Germany', 'Male', ..., 1, 0, 92888.52],
       [792, 'France', 'Female', ..., 1, 0, 38190.78]], dtype=object)
```

Lets start from the data: The dependent variable (Exited), the value that we are going to predict, will be the exit of the customer from the bank (binary variable 0 if the customer stays and 1 if the client exit).

The independent variables will be

Credit Score: reliability of the customer Geography: where is the customer from Gender: Male or Female Age Tenure: number of years of customer history in the company Balance: the money in the bank account Number of products of the customer in the bank Credit Card: if the customer has or not the CC Active: if the customer is active or not Estimated Salary: estimation of salary based on the entries Once we have quality data and selected the right target, we will prepare the data for the model. In general we do not need big data to create a model, but if our variables are significant, modelling with hundreds of data could work. We need always to test test our models to check if everything works correctly. Let's say for our example to work with 10.000 rows dataset We will split our entire dataset in 2 parts. The bigger part, that will be 80% of data, will be used for the training of the model, while the remaining 20% will be used to test the model and have its accuracy.

But our model needs in input numerical data, so, we need to encode categorical data into numerical data. In This case we have Geography (France, Spain and Germany) and Gender (Male and Female). For Geography we will have 0,1,2 instead of France, Spain and Gemany and 0,1 instead of Gender.

```
In [46]: # Encoding categorical data
         from sklearn.preprocessing import LabelEncoder, OneHotEncoder
         labelencoder_X_1 = LabelEncoder()
         X[:, 1] = labelencoder_X_1.fit_transform(X[:, 1])
         labelencoder_X_2 = LabelEncoder()
         X[:, 2] = labelencoder_X_2.fit_transform(X[:, 2])
```

```
In [47]: # Now creating Dummy variables
         onehotencoder = OneHotEncoder(categorical_features = [1])
         X = onehotencoder.fit_transform(X).toarray()
         X = X[:, 1:]
```

```
In [48]: X
```

```
Out[48]: array([[0.0000000e+00, 0.0000000e+00, 6.1900000e+02, ..., 1.0000000e+00,
                 1.0000000e+00, 1.0134888e+05],
                [0.0000000e+00, 1.0000000e+00, 6.0800000e+02, ..., 0.0000000e+00,
                 1.0000000e+00, 1.1254258e+05],
                [0.0000000e+00, 0.0000000e+00, 5.0200000e+02, ..., 1.0000000e+00,
                 0.0000000e+00, 1.1393157e+05],
                ...,
                [0.0000000e+00, 0.0000000e+00, 7.0900000e+02, ..., 0.0000000e+00,
                 1.0000000e+00, 4.2085580e+04],
                [1.0000000e+00, 0.0000000e+00, 7.7200000e+02, ..., 1.0000000e+00,
                 0.0000000e+00, 9.2888520e+04],
                [0.0000000e+00, 0.0000000e+00, 7.9200000e+02, ..., 1.0000000e+00,
                 0.0000000e+00, 3.8190780e+04]])
```

```
In [49]: # Splitting the dataset into the Training set and Test set
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_sta
```

Before training it is important to apply feature scaling to data. It is an important step because it standardize the data and give them the same scale also for a faster computation.

```
In [50]: #Feature scaling
         from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         X_train = sc.fit_transform(X_train)
         X_test = sc.transform(X_test)
```

Now we can start to create our Artificial Neural Network and for doing this we will need to import the Keras library

Listing out the steps involved in training the ANN with Stochastic Gradient Descent:-

1)Randomly initialize the weights to small numbers close to 0(But not 0) 2)Input the 1st observation of your dataset in the Input Layer, each Feature in one Input Node 3)Forward-Propagation from Left to Right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights.Propagate the activations until getting the predicted result y. 4)Compare the predicted result with the actual result. Measure the generated error. 5)Back-Propagation: From Right to Left, Error is back propagated.Update the weights according to how much they are responsible for the error.The Learning Rate tells us by how much such we update the weights. 6)Repeat Steps 1 to 5 and update the weights after each observation(Reinforcement Learning). Or: Repeat Steps 1 to 5 but update the weights only after a batch of observations(Batch Learning) 7)When the whole training set is passed through the ANN.That completes an Epoch. Redo more Epochs.

```
In [51]: # Importing the Keras libraries and packages
         import keras
         from keras.models import Sequential
         from keras.layers import Dense
```

We need now to define the number of hidden layers, the number of nodes for each hidden layers. We know that our input layers will have as many nodes as our independent variables (in our example we have 11 independent variables) and as output layers we will have our y, so 1 dependent variable.

```
In [52]: # Initialising the ANN
         classifier = Sequential()
```

We will use the Dense function A tip for the definition of the number of nodes of hidden layers, based on experiments, is to choose the average between the input and the output layers. In this case we will have 6 nodes (11+1)/2.

```
In [53]: # Adding the input layer and the first hidden layer
         classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu', input_dim

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

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

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

```
C:\Users\Rahul\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: UserWarning: Update your `

C:\Users\Rahul\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: UserWarning: Update your `
  """
C:\Users\Rahul\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: Update your `
```

```
In [54]: # Fitting the ANN to the Training set
         classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)
```

```
C:\Users\Rahul\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: UserWarning: The `nb_epoch
```

```
Epoch 1/100
8000/8000 [==============================] - 1s 141us/step - loss: 0.4862 - acc: 0.7956
Epoch 2/100
8000/8000 [==============================] - 1s 100us/step - loss: 0.4276 - acc: 0.7960
Epoch 3/100
8000/8000 [==============================] - 1s 108us/step - loss: 0.4220 - acc: 0.7960
Epoch 4/100
8000/8000 [==============================] - 1s 108us/step - loss: 0.4183 - acc: 0.8222
Epoch 5/100
8000/8000 [==============================] - 1s 93us/step - loss: 0.4162 - acc: 0.8255
Epoch 6/100
8000/8000 [==============================] - 1s 93us/step - loss: 0.4137 - acc: 0.8287
Epoch 7/100
8000/8000 [==============================] - 1s 92us/step - loss: 0.4128 - acc: 0.8309
Epoch 8/100
8000/8000 [==============================] - 1s 95us/step - loss: 0.4115 - acc: 0.8319
Epoch 9/100
8000/8000 [==============================] - 1s 94us/step - loss: 0.4104 - acc: 0.8340
Epoch 10/100
8000/8000 [==============================] - 1s 94us/step - loss: 0.4095 - acc: 0.8320
Epoch 11/100
8000/8000 [==============================] - 1s 95us/step - loss: 0.4085 - acc: 0.8332
Epoch 12/100
8000/8000 [==============================] - 1s 99us/step - loss: 0.4078 - acc: 0.8345
Epoch 13/100
8000/8000 [==============================] - 1s 101us/step - loss: 0.4074 - acc: 0.8335
Epoch 14/100
8000/8000 [==============================] - 1s 105us/step - loss: 0.4061 - acc: 0.8352
Epoch 15/100
8000/8000 [==============================] - 1s 97us/step - loss: 0.4062 - acc: 0.8334
Epoch 16/100
8000/8000 [==============================] - 1s 110us/step - loss: 0.4060 - acc: 0.8334
Epoch 17/100
8000/8000 [==============================] - 1s 102us/step - loss: 0.4055 - acc: 0.8340
Epoch 18/100
8000/8000 [==============================] - 1s 114us/step - loss: 0.4049 - acc: 0.8341
Epoch 19/100
8000/8000 [==============================] - 1s 116us/step - loss: 0.4048 - acc: 0.8350
Epoch 20/100
8000/8000 [==============================] - 1s 116us/step - loss: 0.4049 - acc: 0.8339
Epoch 21/100
8000/8000 [==============================] - 1s 112us/step - loss: 0.4043 - acc: 0.8347
Epoch 22/100
8000/8000 [==============================] - 1s 99us/step - loss: 0.4040 - acc: 0.8360
```

```
Epoch 23/100
8000/8000 [==============================] - 1s 100us/step - loss: 0.4037 - acc: 0.8365
Epoch 24/100
8000/8000 [==============================] - 1s 97us/step - loss: 0.4033 - acc: 0.8366
Epoch 25/100
8000/8000 [==============================] - 1s 92us/step - loss: 0.4037 - acc: 0.8344
Epoch 26/100
8000/8000 [==============================] - 1s 94us/step - loss: 0.4033 - acc: 0.8342
Epoch 27/100
8000/8000 [==============================] - 1s 94us/step - loss: 0.4028 - acc: 0.8350
Epoch 28/100
8000/8000 [==============================] - 1s 94us/step - loss: 0.4029 - acc: 0.8332
Epoch 29/100
8000/8000 [==============================] - 1s 95us/step - loss: 0.4026 - acc: 0.8347
Epoch 30/100
8000/8000 [==============================] - 1s 96us/step - loss: 0.4026 - acc: 0.8346
Epoch 31/100
8000/8000 [==============================] - 1s 96us/step - loss: 0.4025 - acc: 0.8345
Epoch 32/100
8000/8000 [==============================] - 1s 96us/step - loss: 0.4019 - acc: 0.8340
Epoch 33/100
8000/8000 [==============================] - 1s 96us/step - loss: 0.4021 - acc: 0.8342
Epoch 34/100
8000/8000 [==============================] - 1s 98us/step - loss: 0.4019 - acc: 0.8366
Epoch 35/100
8000/8000 [==============================] - 1s 99us/step - loss: 0.4016 - acc: 0.8341
Epoch 36/100
8000/8000 [==============================] - 1s 101us/step - loss: 0.4022 - acc: 0.8342
Epoch 37/100
8000/8000 [==============================] - 1s 97us/step - loss: 0.4017 - acc: 0.8350
Epoch 38/100
8000/8000 [==============================] - 1s 108us/step - loss: 0.4012 - acc: 0.8355
Epoch 39/100
8000/8000 [==============================] - 1s 118us/step - loss: 0.4016 - acc: 0.8335
Epoch 40/100
8000/8000 [==============================] - 1s 113us/step - loss: 0.4010 - acc: 0.8359
Epoch 41/100
8000/8000 [==============================] - 1s 107us/step - loss: 0.4013 - acc: 0.8347
Epoch 42/100
8000/8000 [==============================] - 1s 109us/step - loss: 0.4015 - acc: 0.8340
Epoch 43/100
8000/8000 [==============================] - 1s 102us/step - loss: 0.4013 - acc: 0.8345
Epoch 44/100
8000/8000 [==============================] - 1s 93us/step - loss: 0.4011 - acc: 0.8366
Epoch 45/100
8000/8000 [==============================] - 1s 95us/step - loss: 0.4012 - acc: 0.8356
Epoch 46/100
8000/8000 [==============================] - 1s 94us/step - loss: 0.4011 - acc: 0.8350
```

```
Epoch 47/100
8000/8000 [==============================] - 1s 98us/step - loss: 0.4006 - acc: 0.8336
Epoch 48/100
8000/8000 [==============================] - 1s 98us/step - loss: 0.4010 - acc: 0.8342
Epoch 49/100
8000/8000 [==============================] - 1s 95us/step - loss: 0.4009 - acc: 0.8356
Epoch 50/100
8000/8000 [==============================] - 1s 99us/step - loss: 0.4010 - acc: 0.8350
Epoch 51/100
8000/8000 [==============================] - 1s 99us/step - loss: 0.4005 - acc: 0.8339
Epoch 52/100
8000/8000 [==============================] - 1s 98us/step - loss: 0.4006 - acc: 0.8352
Epoch 53/100
8000/8000 [==============================] - 1s 94us/step - loss: 0.4005 - acc: 0.8355
Epoch 54/100
8000/8000 [==============================] - 1s 95us/step - loss: 0.4006 - acc: 0.8369
Epoch 55/100
8000/8000 [==============================] - 1s 97us/step - loss: 0.4006 - acc: 0.8351
Epoch 56/100
8000/8000 [==============================] - 1s 96us/step - loss: 0.4002 - acc: 0.8350
Epoch 57/100
8000/8000 [==============================] - 1s 95us/step - loss: 0.4009 - acc: 0.8352
Epoch 58/100
8000/8000 [==============================] - 1s 111us/step - loss: 0.3998 - acc: 0.8350
Epoch 59/100
8000/8000 [==============================] - 1s 114us/step - loss: 0.4008 - acc: 0.8341
Epoch 60/100
8000/8000 [==============================] - 1s 113us/step - loss: 0.4005 - acc: 0.8349
Epoch 61/100
8000/8000 [==============================] - ETA: 0s - loss: 0.3996 - acc: 0.834 - 1s 113us/ste
Epoch 62/100
8000/8000 [==============================] - 1s 100us/step - loss: 0.4004 - acc: 0.8357
Epoch 63/100
8000/8000 [==============================] - 1s 98us/step - loss: 0.4000 - acc: 0.8349
Epoch 64/100
8000/8000 [==============================] - 1s 100us/step - loss: 0.4006 - acc: 0.8335
Epoch 65/100
8000/8000 [==============================] - 1s 115us/step - loss: 0.4003 - acc: 0.8354
Epoch 66/100
8000/8000 [==============================] - 1s 107us/step - loss: 0.4003 - acc: 0.8347
Epoch 67/100
8000/8000 [==============================] - 1s 107us/step - loss: 0.4001 - acc: 0.8342
Epoch 68/100
8000/8000 [==============================] - 1s 121us/step - loss: 0.4004 - acc: 0.8337
Epoch 69/100
8000/8000 [==============================] - 1s 122us/step - loss: 0.3999 - acc: 0.8355
Epoch 70/100
8000/8000 [==============================] - 1s 113us/step - loss: 0.4001 - acc: 0.8340
```

```
Epoch 71/100
8000/8000 [==============================] - 1s 110us/step - loss: 0.4000 - acc: 0.8342
Epoch 72/100
8000/8000 [==============================] - 1s 111us/step - loss: 0.3993 - acc: 0.8342
Epoch 73/100
8000/8000 [==============================] - 1s 106us/step - loss: 0.4002 - acc: 0.8339
Epoch 74/100
8000/8000 [==============================] - 1s 113us/step - loss: 0.4001 - acc: 0.8346
Epoch 75/100
8000/8000 [==============================] - 1s 109us/step - loss: 0.4000 - acc: 0.8356
Epoch 76/100
8000/8000 [==============================] - 1s 124us/step - loss: 0.4000 - acc: 0.8345
Epoch 77/100
8000/8000 [==============================] - 1s 126us/step - loss: 0.3999 - acc: 0.8342
Epoch 78/100
8000/8000 [==============================] - 1s 125us/step - loss: 0.3999 - acc: 0.8349
Epoch 79/100
8000/8000 [==============================] - 1s 125us/step - loss: 0.3998 - acc: 0.8352
Epoch 80/100
8000/8000 [==============================] - 1s 109us/step - loss: 0.3998 - acc: 0.8342
Epoch 81/100
8000/8000 [==============================] - 1s 108us/step - loss: 0.3996 - acc: 0.8347
Epoch 82/100
8000/8000 [==============================] - 1s 103us/step - loss: 0.3999 - acc: 0.8340
Epoch 83/100
8000/8000 [==============================] - 1s 99us/step - loss: 0.3997 - acc: 0.8346
Epoch 84/100
8000/8000 [==============================] - 1s 98us/step - loss: 0.4001 - acc: 0.8355
Epoch 85/100
8000/8000 [==============================] - 1s 98us/step - loss: 0.3998 - acc: 0.8332
Epoch 86/100
8000/8000 [==============================] - 1s 98us/step - loss: 0.4001 - acc: 0.8351
Epoch 87/100
8000/8000 [==============================] - 1s 99us/step - loss: 0.3994 - acc: 0.8345
Epoch 88/100
8000/8000 [==============================] - 1s 104us/step - loss: 0.3994 - acc: 0.8341
Epoch 89/100
8000/8000 [==============================] - 1s 100us/step - loss: 0.3995 - acc: 0.8361
Epoch 90/100
8000/8000 [==============================] - 1s 105us/step - loss: 0.4000 - acc: 0.8350
Epoch 91/100
8000/8000 [==============================] - 1s 101us/step - loss: 0.3999 - acc: 0.8346
Epoch 92/100
8000/8000 [==============================] - 1s 101us/step - loss: 0.3996 - acc: 0.8350
Epoch 93/100
8000/8000 [==============================] - 1s 100us/step - loss: 0.3995 - acc: 0.8350
Epoch 94/100
8000/8000 [==============================] - 1s 103us/step - loss: 0.3993 - acc: 0.8351
```

```
Epoch 95/100
8000/8000 [==============================] - 1s 117us/step - loss: 0.3999 - acc: 0.8340
Epoch 96/100
8000/8000 [==============================] - 1s 117us/step - loss: 0.3996 - acc: 0.8356
Epoch 97/100
8000/8000 [==============================] - 1s 117us/step - loss: 0.3999 - acc: 0.8344
Epoch 98/100
8000/8000 [==============================] - 1s 105us/step - loss: 0.3997 - acc: 0.8347
Epoch 99/100
8000/8000 [==============================] - 1s 102us/step - loss: 0.3997 - acc: 0.8341
Epoch 100/100
8000/8000 [==============================] - 1s 100us/step - loss: 0.3998 - acc: 0.8344
```

```
Out[54]: <keras.callbacks.History at 0xa0e8d42ba8>
```

```
In [55]: # Making the predictions and evaluating the model
         y_pred = classifier.predict(X_test)
         y_pred = (y_pred > 0.5)
```

```
In [56]: # Creating the Confusion Matrix
         from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
```

```
In [57]: cm
```

```
Out[57]: array([[1547,   48],
                [ 261,  144]], dtype=int64)
```

```
In [58]: Accuracy = (1547+144)/(48+261+1547+144)
         Accuracy
```

```
Out[58]: 0.8455
```

The Accuracy of the Neural Network Model is 84.5% which is decent enough.