A PROJECT REPORT ON
CUSTOMER CHURN MODELLING USING ANN

Submitted by

Pranav Kalikate

What is ANN?

Artificial Neural Networks or ANN is an information processing model that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements (neurons) working in unison to solve a specific problem.

What is Activation Function?

The activation function is the process applied to the weighted input value once it enters the neuron. The neurons are activated in a way that the impact of each neuron's activation is limited by weights. Different activation functions are,

1. Threshold function
2. Sigmoid function
3. Rectifier function
4. Hyperbolic tangent function (tanh)

What are Weights?

Weights are a pivotal factor in a Neural Network's functioning. Weights are how Neural Networks learn. Based on each weight, the Neural Network decides what information is important, and what isn't. The weight determines which signals get passed along or not, or to what extent a signal gets passed along. The weights are what you will adjust through the process of learning.

What is Cost Function?

The cost function tells us the error in our prediction. Our aim is to minimize the cost function. The lower the cost function, the closer $\hat{Y}$ is to Y, and hence, the closer our output value to our actual value. A lower cost function means higher accuracy for our Network.

Steps in Building your ANN

1. Data preprocessing.
2. Creating the structure of ANN.
    - Add input layer and first hidden layer.
    - Add more hidden layers according to the requirement.
    - Select input & hidden nodes, activation function, optimizer, loss, and performance metrics.
    - Compile the ANN
    - Fit the ANN to training set.
    - Evaluate the model performance.
    - Adjust the optimization parameters or model if needed.

Problem Description

Here we have the first 10 rows of our sample dataset. The sample dataset we have is of a fictional bank but it is very realistic. There are 10,000 customers or rows and the columns represents the information like customers Gender, Age, Balance etc. We also have a column 'Exited' which gives an information about whether or not a customer leaves a bank.

First Ten rows:

Index	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101349	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.9	1	0	1	112543	0
2	3	15619304	Onio	502	France	Female	42	8	159661	3	1	0	113932	1
3	4	15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.6	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125511	1	1	1	79084.1	0
5	6	15574012	Chu	645	Spain	Male	44	8	113756	2	1	0	149757	1
6	7	15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
7	8	15656148	Obinna	376	Germany	Female	29	4	115047	4	1	0	119347	1
8	9	15792365	He	501	France	Male	44	4	142051	2	0	1	74940.5	0
9	10	15592389	H?	684	France	Male	27	2	134604	1	1	1	71725.7	0
10	11	15767821	Bearce	528	France	Male	31	6	102017	2	0	0	80181.1	0

Here the bank is seeing some unusual churn at higher rate and they want to assess and address this problem. My goal here is to go through the dataset, build geo demographic segmentation model to tell the bank which of the customers are at a higher risk of leaving.

Building a model

Part 1 Data Preprocessing

- **Importing the libraries**

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- **Importing the dataset**

```
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[:, 3:13].values
y = dataset.iloc[:, 13].values
```

```
In [3]: X
Out[3]:
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)
```

y - NumPy array

	0
0	1
1	0
2	1
3	0
4	0
5	1
6	0
7	1
8	0
9	0
10	0

- **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])
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	0	0	619	0	42	2	0	1	1	1	101349
1	0	0	1	608	0	41	1	83807.9	1	0	1	112543
2	1	0	0	502	0	42	8	159661	3	1	0	113932
3	1	0	0	699	0	39	1	0	2	0	0	93826.6
4	0	0	1	850	0	43	2	125511	1	1	1	79084.1
5	0	0	1	645	1	44	8	113756	2	1	0	149757
6	1	0	0	822	1	50	7	0	2	1	1	10062.8
7	0	1	0	376	0	29	4	115047	4	1	0	119347
8	1	0	0	501	1	44	4	142051	2	0	1	74940.5
9	1	0	0	684	1	27	2	134604	1	1	1	71725.7
10	1	0	0	528	1	31	6	102017	2	0	0	80181.1

- **Avoiding dummy variable trap**

  X = X[:, 1:]

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	619	0	42	2	0	1	1	1	101349
1	0	1	608	0	41	1	83807.9	1	0	1	112543
2	0	0	502	0	42	8	159661	3	1	0	113932
3	0	0	699	0	39	1	0	2	0	0	93826.6
4	0	1	850	0	43	2	125511	1	1	1	79084.1
5	0	1	645	1	44	8	113756	2	1	0	149757
6	0	0	822	1	50	7	0	2	1	1	10062.8
7	1	0	376	0	29	4	115047	4	1	0	119347
8	0	0	501	1	44	4	142051	2	0	1	74940.5
9	0	0	684	1	27	2	134604	1	1	1	71725.7
10	0	0	528	1	31	6	102017	2	0	0	80181.1

- **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_state = 0)

- **Feature Scaling**

  from sklearn.preprocessing import StandardScaler

  sc = StandardScaler()

  X_train = sc.fit_transform(X_train)

  X_test = sc.transform(X_test)

Part 2 Initializing the ANN

- **Importing the Keras libraries and packages**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
```

- **Initializing the ANN**

```
classifier = Sequential()
```

- **Adding the input layer, hidden layer and output layer**

```
#Adding the input layer and the first hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu', input_dim = 11))
#Adding the second hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
#Adding the third hidden layer
classifier.add(Dense(units = 6, kernel_initializer = 'uniform', activation = 'relu'))
#Adding the output layer
classifier.add(Dense(units = 1, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

- **Compiling the ANN**

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

- **Fitting the ANN to the Training set**

```
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

Steps in Training the ANN with Stochastic Gradient Descent:-

- 1. Randomly initialize the weights to a small numbers close to 0. (But not 0)
- 2. Input the first 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 weights. Propagate the results until getting the predicted result y.
- 4. Compare the predicted result to the actual result. Measure the generated error.
- 5. Back Propagation: From right to left, the error is propagated. Update the weights according to how much they are responsible for the error. The learning rate decides by how much we update the weights.
- 6. Repeat the steps 1 to 5 and update the weights after each observation or update the weights only after a batch of observations.
- 7. When the whole training set passed through the ANN that makes an epoch. Redo more epochs.

```
Epoch 90/100
8000/8000 [==============================] - 1s 186us/step - loss: 0.3365 - accuracy: 0.8620
Epoch 91/100
8000/8000 [==============================] - 1s 187us/step - loss: 0.3360 - accuracy: 0.8616
Epoch 92/100
8000/8000 [==============================] - 1s 187us/step - loss: 0.3360 - accuracy: 0.8625
Epoch 93/100
8000/8000 [==============================] - 1s 187us/step - loss: 0.3363 - accuracy: 0.8611
Epoch 94/100
8000/8000 [==============================] - 1s 186us/step - loss: 0.3355 - accuracy: 0.8616
Epoch 95/100
8000/8000 [==============================] - 1s 187us/step - loss: 0.3362 - accuracy: 0.8612
Epoch 96/100
8000/8000 [==============================] - 1s 186us/step - loss: 0.3355 - accuracy: 0.8612
Epoch 97/100
8000/8000 [==============================] - 2s 189us/step - loss: 0.3358 - accuracy: 0.8619
Epoch 98/100
8000/8000 [==============================] - 1s 186us/step - loss: 0.3351 - accuracy: 0.8634
Epoch 99/100
8000/8000 [==============================] - 1s 187us/step - loss: 0.3358 - accuracy: 0.8618
Epoch 100/100
8000/8000 [==============================] - 1s 185us/step - loss: 0.3363 - accuracy: 0.8611
Out[1]: <keras.callbacks.callbacks.History at 0x22c251aa048>
```

Accuracy obtained on training set = 0.8611 = 86%

- **Predicting the Test set results**

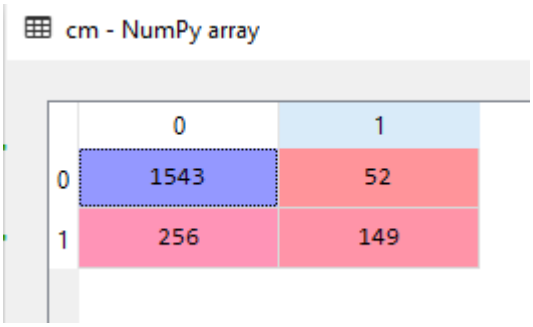
  y_pred = classifier.predict(X_test)

  y_pred = (y_pred > 0.5)

- **Making the Confusion Matrix**

  from sklearn.metrics import confusion_matrix

  cm = confusion_matrix(y_test, y_pred)



Accuracy obtained on test set = (TP+TN) / (TP+TN+FP+FN)

= 0.846= 84%

Our Model is now trained on training set and it is also performing well on test set.

- **Part 3 Obtaining a result of a single new observation**

```
 # Predicting a single new observation
"""Predict if the customer with the following informations will leave the bank:
Geography: France
Credit Score: 600
Gender: Male
Age: 40
Tenure: 3
Balance: 60000
Number of Products: 2
Has Credit Card: Yes
Is Active Member: Yes
Estimated Salary: 50000"""
new_prediction = classifier.predict(sc.transform(np.array([[0.0, 0, 600, 1, 40, 3, 60000, 2, 1, 1, 50000]])))
new_prediction = (new_prediction > 0.5)
#0.0 to avoid error
```

```
In [6]: new_prediction
Out[6]: array([[False]])
```

The person with the above information will not leave a bank.