Hands-on of Big Data Analyst with TuV Certified Qualification

Artificial Neural Network

Artificial neural networks (ANN) or connectionist systems are computing systems that are inspired by, but not necessarily identical to, the biological neural networks that constitute animal brains. *WIkipedia*.

In this practice, we will keep working on the Telco Cutomer Churn Dataset.

```
# Import Library
import pandas as pd


#Import the files to Google Colab
url = 'https://raw.githubusercontent.com/rc-dbe/bigdatacertification/master/dataset/churn_i
df_csv = pd.read_csv(url, sep=',',)

# Show 10 first Row
df_csv.head()
```

```
# Remove "Unnamed:0" Coloumn
df = df_csv.drop("Unnamed: 0", axis=1)
df.head()
# Data Frame for the output parameter 'Churn'
df_churn = df
# Data Frame for the output parameter 'Total Charges'
```

✓ 0s completed at 5:04 PM

```
df_churn.head()
```

```
df_totalCharges.head()
```

```
# Check the Data Infomation
df_churn.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 46 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   gender_0            7043 non-null   int64
 1   gender_1            7043 non-null   int64
 2   SeniorCitizen_0     7043 non-null   int64
```

```
 3   SeniorCitizen_1     7043 non-null   int64
 4   Partner_0           7043 non-null   int64
 5   Partner_1           7043 non-null   int64
 6   Dependents_0        7043 non-null   int64
 7   Dependents_1        7043 non-null   int64
 8   tenure              7043 non-null   int64
 9   PhoneService_0      7043 non-null   int64
 10  PhoneService_1      7043 non-null   int64
 11  MultipleLines_0     7043 non-null   int64
 12  MultipleLines_1     7043 non-null   int64
 13  MultipleLines_2     7043 non-null   int64
 14  InternetService_0   7043 non-null   int64
 15  InternetService_1   7043 non-null   int64
 16  InternetService_2   7043 non-null   int64
 17  OnlineSecurity_0    7043 non-null   int64
 18  OnlineSecurity_1    7043 non-null   int64
 19  OnlineSecurity_2    7043 non-null   int64
 20  OnlineBackup_0      7043 non-null   int64
 21  OnlineBackup_1      7043 non-null   int64
 22  OnlineBackup_2      7043 non-null   int64
 23  DeviceProtection_0  7043 non-null   int64
 24  DeviceProtection_1  7043 non-null   int64
 25  DeviceProtection_2  7043 non-null   int64
 26  TechSupport_0       7043 non-null   int64
 27  TechSupport_1       7043 non-null   int64
 28  TechSupport_2       7043 non-null   int64
 29  StreamingTV_0       7043 non-null   int64
 30  StreamingTV_1       7043 non-null   int64
 31  StreamingTV_2       7043 non-null   int64
 32  StreamingMovies_0   7043 non-null   int64
 33  StreamingMovies_1   7043 non-null   int64
 34  StreamingMovies_2   7043 non-null   int64
 35  Contract_0          7043 non-null   int64
 36  Contract_1          7043 non-null   int64
 37  Contract_2          7043 non-null   int64
 38  PaperlessBilling_0  7043 non-null   int64
 39  PaperlessBilling_1  7043 non-null   int64
 40  PaymentMethod_0     7043 non-null   int64
 41  PaymentMethod_1     7043 non-null   int64
 42  PaymentMethod_2     7043 non-null   int64
 43  MonthlyCharges      7043 non-null   int64
 44  TotalCharges        7043 non-null   float64
 45  Churn               7043 non-null   int64
dtypes: float64(1), int64(45)
memory usage: 2.5 MB
```

```
#Import MinMax Scaler
from sklearn.preprocessing import MinMaxScaler
#-------------------For 'Churn' as the output parameter-----------------------
# initialize min-max scaler
mm_scaler = MinMaxScaler()
column_names = df_churn.columns.tolist()
column_names.remove('Churn')
```

```
# Transform all attributes
df_churn[column_names] = mm_scaler.fit_transform(df_churn[column_names])
df_churn.sort_index(inplace=True)
df_churn.head()
```

```
#-------------------For 'TotalCharges' as the output parameter------------------
column_names_2 = df_totalCharges.columns.tolist()
column_names_2.remove('TotalCharges')

# Transform all attributes
df_totalCharges[column_names_2] = mm_scaler.fit_transform(df_totalCharges[column_names_2])
df_totalCharges.sort_index(inplace=True)
df_totalCharges.head()
```

```
# Selecting the Feature, by remove the unused feature
feature = ['Churn']
train_feature_1 = df_churn.drop(feature, axis=1)
```

```
# Set The Target
train_target_1 = df_churn["Churn"]
```

```
# Show the Feature
train_feature_1.head(5)
```

The scikit-learn library provides us with the splitter function train_test_split(). The original dataset is split into input(X) and output(Y) columns, then call the function passing both arrays and have them split into train and test subsets. Input(X) in this case is 'train_feature' and output(Y) will be 'train_target'. The size of the split is specified via the test_size argument, the value used in this case is 0.3 which denotes that 30 percent of the dataset will be allocated to the test set and 70 percent will be allocated to the training test. The shffle parameter is used to shuffle the data before splitting, its default value is 'True'.

```
# Split Data
# For 70:30 train-test ratio
from sklearn.model_selection import train_test_split, cross_val_score
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(train_feature_1,
                                                            train_target_1,
                                                            shuffle = True,
                                                            test_size=0.3,
                                                            random_state=1)
```

```
# Show the training data
X_train_1.head()
```

To train the ANN Model.We will use the MLPClassifier from Scikit Learn Library. The full documentation can be seen HERE. Below is the default parameter:

```
sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100), activation='relu',
solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant',
learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None,
tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,
early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999,
epsilon=1e-08, n_iter_no_change=10)
```

```
# Import Library
from sklearn.neural_network import MLPClassifier

# Fitting Model
mlp_1 = MLPClassifier(hidden_layer_sizes=(5), activation = 'relu', solver = 'adam',max_iter
mlp_1 = mlp_1.fit(X_train_1,y_train_1)

# Prediction to Test Dataset
y_predmlp_1 = mlp_1.predict(X_test_1)
```

```
Iteration 1, loss = 0.56286397
Iteration 2, loss = 0.51892746
Iteration 3, loss = 0.49642797
Iteration 4, loss = 0.48035651
Iteration 5, loss = 0.46834415
Iteration 6, loss = 0.45879149
Iteration 7, loss = 0.45169116
Iteration 8, loss = 0.44740694
Iteration 9, loss = 0.44462657
Iteration 10, loss = 0.44247464
Iteration 11, loss = 0.44067617
Iteration 12, loss = 0.43952082
Iteration 13, loss = 0.43801440
Iteration 14, loss = 0.43722527
Iteration 15, loss = 0.43661789
Iteration 16, loss = 0.43568316
```

```
Iteration 17, loss = 0.43519491
Iteration 18, loss = 0.43437119
Iteration 19, loss = 0.43408388
Iteration 20, loss = 0.43336906
Iteration 21, loss = 0.43323358
Iteration 22, loss = 0.43250250
Iteration 23, loss = 0.43226889
Iteration 24, loss = 0.43184781
Iteration 25, loss = 0.43156140
Iteration 26, loss = 0.43121486
Iteration 27, loss = 0.43135299
Iteration 28, loss = 0.43067097
Iteration 29, loss = 0.43062674
Iteration 30, loss = 0.43006995
Iteration 31, loss = 0.42975083
Iteration 32, loss = 0.42955768
Iteration 33, loss = 0.42964746
Iteration 34, loss = 0.42908955
Iteration 35, loss = 0.42890498
Iteration 36, loss = 0.42860162
Iteration 37, loss = 0.42842291
Iteration 38, loss = 0.42828291
Iteration 39, loss = 0.42816452
Iteration 40, loss = 0.42842211
Iteration 41, loss = 0.42815132
Iteration 42, loss = 0.42762660
Iteration 43, loss = 0.42722680
Iteration 44, loss = 0.42723554
Iteration 45, loss = 0.42730384
Iteration 46, loss = 0.42683306
Iteration 47, loss = 0.42711535
Iteration 48, loss = 0.42665229
Iteration 49, loss = 0.42658386
Iteration 50, loss = 0.42638787
Iteration 51, loss = 0.42599712
Iteration 52, loss = 0.42695583
Iteration 53, loss = 0.42635368
Iteration 54, loss = 0.42581617
Iteration 55, loss = 0.42557948
Iteration 56, loss = 0.42574805
Iteration 57, loss = 0.42572099
Iteration 58, loss = 0.42566969
```

```
print('Number of Layer =', mlp_1.n_layers_)
print('Number of Iteration =', mlp_1.n_iter_)
print('Current loss computed with the loss function =', mlp_1.loss_)
```

```
Number of Layer = 3
Number of Iteration = 163
Current loss computed with the loss function = 0.4202830677824757
```

Since it was the classification problem, we can evaluate the model using Confussion Matrix

A confusion matrix is a tabular summary of the number of correct and incorrect predictions made by a classifier. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall and F-1 score. y_test is a list that holds the actual labels. y_predmlp is a list that holds the predicted labels. metrics.confusion_matrix() takes in the list of actual labels and the list of predicted labels and calculates the confusion matrix for the given inputs. The confusion matrix has four diferent values in an array, 'True Negative(Top-Left), 'False Positive'(Top-Right), 'False Negative'(Bottom-Left) and 'True Positive'(Bottom-Right).

```
# Import the metrics class
from sklearn import metrics
```

```
# Confussion Matrix
cnf_matrixmlp = metrics.confusion_matrix(y_test_1, y_predmlp_1)
cnf_matrixmlp
```

```
array([[1397,  188],
       [ 208,  320]])
```

```
# For 10:90 train-test ratio
from sklearn.model_selection import train_test_split, cross_val_score
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(train_feature_1,
                                                            train_target_1,
                                                            shuffle = True,
                                                            test_size=0.9,
                                                            random_state=1)
```

```
# Show the training data
X_train_2.head()
```

```
# Fitting Model
mlp_2 = MLPClassifier(hidden_layer_sizes=(5), activation = 'relu', solver = 'adam',max_iter
mlp_2 = mlp_2.fit(X_train_2,y_train_2)

# Prediction to Test Dataset
y_predmlp_2 = mlp_2.predict(X_test_2)
```

```
Iteration 1, loss = 1.58277533
Iteration 2, loss = 1.51716494
Iteration 3, loss = 1.45628112
Iteration 4, loss = 1.40095440
Iteration 5, loss = 1.35070943
Iteration 6, loss = 1.30305001
Iteration 7, loss = 1.25801389
Iteration 8, loss = 1.21776879
Iteration 9, loss = 1.17886388
Iteration 10, loss = 1.14268317
Iteration 11, loss = 1.10754769
Iteration 12, loss = 1.07313057
Iteration 13, loss = 1.04054638
Iteration 14, loss = 1.00879614
Iteration 15, loss = 0.97686905
Iteration 16, loss = 0.94454777
Iteration 17, loss = 0.91262189
Iteration 18, loss = 0.87968824
Iteration 19, loss = 0.84754807
Iteration 20, loss = 0.81629250
Iteration 21, loss = 0.78719395
Iteration 22, loss = 0.75962998
Iteration 23, loss = 0.73406627
Iteration 24, loss = 0.71007822
Iteration 25, loss = 0.68951205
Iteration 26, loss = 0.67000595
Iteration 27, loss = 0.65393851
Iteration 28, loss = 0.63861816
Iteration 29, loss = 0.62627991
Iteration 30, loss = 0.61560911
Iteration 31, loss = 0.60622053
Iteration 32, loss = 0.59855971
Iteration 33, loss = 0.59219290
Iteration 34, loss = 0.58657127
Iteration 35, loss = 0.58205545
Iteration 36, loss = 0.57819639
Iteration 37, loss = 0.57452876
Iteration 38, loss = 0.57149611
Iteration 39, loss = 0.56905152
Iteration 40, loss = 0.56656440
Iteration 41, loss = 0.56437582
Iteration 42, loss = 0.56243467
Iteration 43, loss = 0.56043965
Iteration 44, loss = 0.55865076
Iteration 45, loss = 0.55681687
Iteration 46, loss = 0.55500013
```

```
Iteration 47, loss = 0.55327345
Iteration 48, loss = 0.55156101
Iteration 49, loss = 0.54981586
Iteration 50, loss = 0.54818877
Iteration 51, loss = 0.54643647
Iteration 52, loss = 0.54481386
Iteration 53, loss = 0.54312245
Iteration 54, loss = 0.54142604
Iteration 55, loss = 0.53976566
Iteration 56, loss = 0.53808081
Iteration 57, loss = 0.53648282
Iteration 58, loss = 0.53482921
```

Accuracy measures how often the model is correct.

It is calculated as: Accuracy = (TP + TN)/Total Predictions

Precision means of the positives predicted, what percentage is truly positive?

It is calculated as: Precision = TP/(TP + FP)

Recall indicates of all the positive cases, what percentage are predicted positive?

Recall = TP/(TP + FN)

F1 Score is the harmonic mean of precision and recall.

F1 Score = 2(PrceisionRecall)/(Precision + Recall)

```
###############################################################################
# 2. Train only 10% of the dataset, then run cell by cell from the top again.
#    Compare the performance and explain.

# Show the Accuracy, Precision, Recall, F1, etc.
acc_mlp_1   = metrics.accuracy_score(y_test_1, y_predmlp_1)
prec_mlp_1  = metrics.precision_score(y_test_1, y_predmlp_1)
rec_mlp_1   = metrics.recall_score(y_test_1, y_predmlp_1)
f1_mlp_1    = metrics.f1_score(y_test_1, y_predmlp_1)
kappa_mlp_1 = metrics.cohen_kappa_score(y_test_1, y_predmlp_1)

acc_mlp_2   = metrics.accuracy_score(y_test_2, y_predmlp_2)
prec_mlp_2  = metrics.precision_score(y_test_2, y_predmlp_2)
rec_mlp_2   = metrics.recall_score(y_test_2, y_predmlp_2)
f1_mlp_2    = metrics.f1_score(y_test_2, y_predmlp_2)
kappa_mlp_2 = metrics.cohen_kappa_score(y_test_2, y_predmlp_2)

print("-------------Performance with 70:30 train-test ratio-------------------")
print("Accuracy:", acc_mlp_1)
print("Precision:", prec_mlp_1)
print("Recall:", rec_mlp_1)
print("F1 Score:", f1_mlp_1)
print("Cohens Kappa Score:", kappa_mlp_1)
```

```
print("Cohens Kappa Score:", kappa_mlp_1)

print("-------------Performance with 10:90 train-test ratio-------------------")
print("Accuracy:", acc_mlp_2)
print("Precision:", prec_mlp_2)
print("Recall:", rec_mlp_2)
print("F1 Score:", f1_mlp_2)
print("Cohens Kappa Score:", kappa_mlp_2)
```

```
-------------Performance with 70:30 train-test ratio-------------------
Accuracy: 0.812588736393753
Precision: 0.6299212598425197
Recall: 0.6060606060606061
F1 Score: 0.6177606177606176
Cohens Kappa Score: 0.4936839684864034
-------------Performance with 10:90 train-test ratio-------------------
Accuracy: 0.7942893200820319
Precision: 0.6189848384970337
Recall: 0.563963963963964
F1 Score: 0.5901948460087995
Cohens Kappa Score: 0.45327019724999285
```

It is clearly seen that the model's correctness and the true performance is dropped moderately when the training data size is reduced. This is because with the 70:30 train-test ratio, the model has enough data to trian the model, before testing it on the unknown test data as compared to the 10:90 train test ratio, where in contrast the model has huge amount of unknown data.

```
######################################################################
# 3. Change the output parameter from Churn to TotalCharges, and run again to find
#    out the mean squared error (MSE) value. (hint: use Multi Regressor)
# Selecting the Feature, by remove the unused feature
feature_2 = ['TotalCharges']
train_feature_2 = df_churn.drop(feature_2, axis=1)

# Set The Target
train_target_2 = df_churn["TotalCharges"]


train_feature_2.head()
```

```
# Split Data
from sklearn.model_selection import train_test_split, cross_val_score
X_train_3, X_test_3, y_train_3, y_test_3 = train_test_split(train_feature_2,
                                                            train_target_2,
                                                            shuffle = True,
                                                            test_size=0.3,
                                                            random_state=1)
```

```
X_train_3.head()
```

```
# Import Library
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error

# Fitting Model

mlp = MLPRegressor(hidden_layer_sizes=(5), activation = 'relu', solver = 'adam',max_iter= 
mlp = mlp.fit(X_train_3,y_train_3)

# Prediction to Test Dataset
y_predmlp_3 = mlp.predict(X_test_3)
```

```
Iteration 1, loss = 0.22782005
Iteration 2, loss = 0.11589921
Iteration 3, loss = 0.07629270
Iteration 4, loss = 0.05268521
Iteration 5, loss = 0.03880269
```

```
Iteration 6, loss = 0.03214905
Iteration 7, loss = 0.02877301
Iteration 8, loss = 0.02665774
Iteration 9, loss = 0.02504460
Iteration 10, loss = 0.02377668
Iteration 11, loss = 0.02271709
Iteration 12, loss = 0.02189077
Iteration 13, loss = 0.02119426
Iteration 14, loss = 0.02059020
Iteration 15, loss = 0.02008354
Iteration 16, loss = 0.01962719
Iteration 17, loss = 0.01921016
Iteration 18, loss = 0.01884202
Iteration 19, loss = 0.01852718
Iteration 20, loss = 0.01821053
Iteration 21, loss = 0.01792961
Iteration 22, loss = 0.01769687
Iteration 23, loss = 0.01749315
Iteration 24, loss = 0.01729315
Iteration 25, loss = 0.01711827
Iteration 26, loss = 0.01694462
Iteration 27, loss = 0.01679128
Iteration 28, loss = 0.01665337
Iteration 29, loss = 0.01650761
Iteration 30, loss = 0.01638419
Iteration 31, loss = 0.01627396
Iteration 32, loss = 0.01616327
Iteration 33, loss = 0.01606678
Iteration 34, loss = 0.01596917
Iteration 35, loss = 0.01590270
Iteration 36, loss = 0.01584711
Iteration 37, loss = 0.01575222
Iteration 38, loss = 0.01570738
Iteration 39, loss = 0.01564091
Iteration 40, loss = 0.01562191
Iteration 41, loss = 0.01556880
Iteration 42, loss = 0.01552881
Iteration 43, loss = 0.01547941
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopp
```

```
# Mean Squared Error
mse = mean_squared_error(y_test_3, y_predmlp_3)
print("The mean squared error (MSE) on test set: {:.4f}".format(mse))
```

```
The mean squared error (MSE) on test set: 0.0318
```