Deep Learning-Demo

Capgemini

Lesson Objectives

On completion of this lesson, you will be able to understand :

- The key-steps of how to create an Artificial Neural Networks(ANN) :
- Train the ANN
- Test the ANN

- Our Dataset is in the folder "C:\BigData\MachineLearning\Machine Learning A-Z Template Folder\Part 8 - Deep Learning\Section 39 - Artificial Neural Networks (ANN)" in the file named as "Churn_Modelling.csv"

- The goal is to device a predictive customer-churn system

- Install "Tensorflow" for Anaconda by typing the following command at the Anaconda-terminal :

  conda install -c conda-forge keras

- Tensorflow is an open-source Machine-Learning Framework.

conda install –channel https://conda.anaconda.org/conda-forge keras

Importing the Libraries

- Import the basic libraries for any python application :
  - numpy
  - matplotlib
  - pandas

```
# Artificial Neural Network
# Part 1 - Data Preprocessing
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

- numpy : Contains a lot of mathematical tools or functionalities
- matplotlib.pyplot : To enable us to plot charts, graphs
- pandas : For importing and managing datasets into our code

- Change the directory to the path where our data-set lies “C:\BigData\MachineLearning\Machine Learning A-Z Template Folder\Part 8 - Deep Learning\Section 39 - Artificial Neural Networks (ANN)”

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

- Check the values of the matrix of Independent Variables(IV) and the vector of Dependent Variable(DV) at the IPyhton console

- Encoding of categorical data from the Matrix of IVs

```
# Encoding categorical data
##Label Encoding
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])
##OneHot Encoding
onehotencoder = OneHotEncoder(categorical_features = [1])
x = onehotencoder.fit_transform(x).toarray()
##Taking care of Dummy-Variable Trap
x = x[:, 1:]
```

- Ensure that you remove 1 column from every set of onehot-encoded column-list
- Check the values of the matrix of Independent Variables(IV) “x” at the IPyhton console after doing the label-encoding, after doing the onehot-encoding and after avoiding the dummy-variable trap

Splitting the Data into Training-set and Test-set

- Import the train_test_split class from the library sklearn.cross_validation :

```
# Splitting the Data into Training-set and Test-set
from sklearn.cross_validation import train_test_split
or
from sklearn.model_selection import train_test_split
```

- create 4 variables : x_train, x_test, y_train, y_test as follows :

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size = 0.2, random_state= 0)
```

- Our training-sets : x_train(as training data from the matrix of features of independent variables)and y_train(as training data from the vector of dependent variables associated with x_train)

- Our test-sets : x_test(as test-data from the matrix of features of independent variables) and y_test(as test-data from the vector of dependent variables associated with x_test)

Feature Scaling

- To prevent any IV from the matrix of IVs from dominating the entire ANN architecture, we need to implement feature-scaling

```
#Feature scaling the Data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

Importing the “keras” library

- Using the “keras” library/framework on top of the “tensorflow” platform

- The Sequential class for initializing the Neural-Network

- The Dense class for building the layers of our Neural Network.

```
# Part 2 - Creating the ANN
# Importing the appropriate classes from keras
from keras.models import Sequential
from keras.layers import Dense
```

Initializing the ANN and Adding the Input and Hidden Layer

- Create an object for the Deep-Learning Model

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

- Adding the Input Layer and Hidden Layer to our ANN

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

You may get a warning

or

```
classifier.add(Dense(activation="relu", input_dim=11, units=6, kernel_initializer="uniform"))
```

- The add function adds the hidden-layers

- The arguments to “Dense” will be the number of nodes for the hidden-layer, how the weights are updated, the activation function to be used and the number of input-nodes

Adding more Hidden Layer

- Adding on more Hidden-Layer

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

- NO input_dim parameter of the Dense() function because we already have a layer before this layer

- Adding the Output-Layer

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

- We will keep the activation-function as 'sigmoid' as we need an output from this output layer in a typical classifier-style(probabilities between 0 and 1)

Compiling the ANN

- Using the Stochastic Gradient Descent algorithm to optimize the selection of weights

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

- optimizer = 'adam' is used to select the Gradient Descent algorithm

- loss = 'binary_crossentropy’ is the ideal loss-function for classification type of problems

- The metric parameter or function is used to optionally judge the accuracy-performance of your model

Fitting the ANN to the Training-set

- "fit" method is used to fit or train the model on the training-set

```
# Fitting the ANN to the Training set
classifier.fit(x_train, y_train, batch_size = 10, nb_epoch = 100)
OR to avoid warnings
classifier.fit(x_train, y_train, batch_size = 10, epochs = 100)
```

- The training-set(Matrix of IVs and Vector DVs) is passed as parameter

- batch_size = 10 indicates the number of observations to be passed in each iteration

- nb_epoch = 100 is the number of times the entire dataset should be passed through the model

Prediction on the Test-set

- "predict" method is used to get the predictions on the test-set using our model

```
# Part 3 – Making the predictions to evaluate the model
# Predicting the Test set results
y_pred = classifier.predict(x_test)
y_pred = (y_pred > 0.5)
```

- We can use a "Confusion-Matrix" to test the performance of predictions done on the test-data

N=200	Predicted FALSE	Predicted TRUE
Actual FALSE	100	12
Actual TRUE	13	75
Percentage of Accuracy (100+75)/200		0.875

Evaluation of predictions using Confusion-Matrix

- The actual-data and the predicted data from the test-set are used to calculate the confusion-matrix

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

- Check the accuracy of our predictions against the actual-data from the test-set, at the i-python console just do the calculation : (1480+226)/2000 = 0.853

	0	1
0	1480	115
1	179	226

Thank You