PRELIMINARY TASK

Shaik Hashmath

4th B.Tech

JNTUK-UCEV

Visakhapatnam

27-11-2020

—

To build a classification model on the given data using Deep Learning approach

—

Credicxo Tech Private Limited

TASK: TO BUILD A CLASSIFICATION MODEL ON THE GIVEN DATA USING DEEP LEARNING APPROACH

The given dataset contains details about organic chemical compounds. The compounds are classified as either ‘Musk’ or ‘Non-Musk’ compounds. The task is to classify these compounds accordingly. I used an Artificial Neural Network (ANN) built using Keras.

This many-to-one relationship between feature vectors and molecules is called the "multiple instance problem". When learning a classifier for this data, the classifier should classify a molecule as "musk" if ANY of its conformations is classified as a musk. A molecule should be classified as "non-musk" if NONE of its conformations is classified as a musk.
A simple neural network can do a fine work to classify such data.

Step 1:

Importing the required libraries

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.decomposition import PCA
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import f1_score, precision_score, recall_score
```

Step 2: Reading the data and preprocessing

```
dataset = pd.read_csv('../input/credcxodataset/musk_csv.csv') #reading the dataset using pandas
dataset.head() #Displaying the dataset
```

Data preprocessing:

1. Checking for null values
2. Performing Feature Scaling

```
dataset.isna().sum() #Checking for any null values
```

```
X = dataset.iloc[:, 3:-1].values  #Extracting the important features from the dataset
y = dataset.iloc[:, -1].values
```

```
from sklearn.preprocessing import RobustScaler  #feature scaling
scaler = RobustScaler()

X = scaler.fit_transform(X)

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)
```

Step 3:

Building Deep Learining Model

Used an Artificial Neural Network (ANN)

```
model =tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(40, input_shape=(166,),activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(10,activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(1,activation=tf.nn.sigmoid))
```

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

```
history = model.fit(X_train, y_train, validation_split = 0.2, epochs = 40) #training the model
```

A simple neural network is built with three dense layers and the last layer should output only a single value hence a single neuron is used at the last layer.

Step 4:

Model Training

Trained the model for 40 epochs.

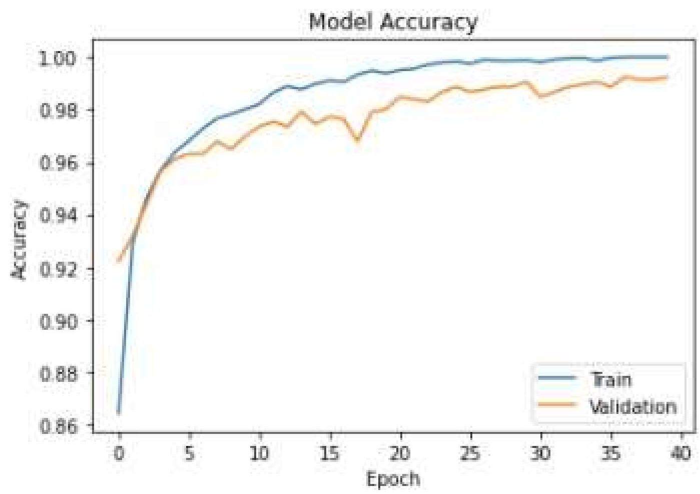
```
Epoch 1/40
132/132 [==============================] - 0s 3ms/step - loss: 0.4122 - accuracy: 0.8643 - val_loss: 0.2369 - val_accuracy: 0.9223
Epoch 2/40
132/132 [==============================] - 0s 3ms/step - loss: 0.1928 - accuracy: 0.9292 - val_loss: 0.1770 - val_accuracy: 0.9318
Epoch 3/40
132/132 [==============================] - 0s 3ms/step - loss: 0.1501 - accuracy: 0.9465 - val_loss: 0.1485 - val_accuracy: 0.9441
Epoch 4/40
132/132 [==============================] - 0s 3ms/step - loss: 0.1236 - accuracy: 0.9569 - val_loss: 0.1453 - val_accuracy: 0.9564
Epoch 5/40
132/132 [==============================] - 0s 3ms/step - loss: 0.1072 - accuracy: 0.9638 - val_loss: 0.1344 - val_accuracy: 0.9612
Epoch 6/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0962 - accuracy: 0.9680 - val_loss: 0.1119 - val_accuracy: 0.9631
Epoch 7/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0785 - accuracy: 0.9728 - val_loss: 0.1095 - val_accuracy: 0.9631
Epoch 8/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0716 - accuracy: 0.9766 - val_loss: 0.1038 - val_accuracy: 0.9678
Epoch 9/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0633 - accuracy: 0.9782 - val_loss: 0.1156 - val_accuracy: 0.9650
Epoch 10/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0637 - accuracy: 0.9801 - val_loss: 0.0900 - val_accuracy: 0.9697
Epoch 11/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0480 - accuracy: 0.9820 - val_loss: 0.0807 - val_accuracy: 0.9735
Epoch 12/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0420 - accuracy: 0.9865 - val_loss: 0.0803 - val_accuracy: 0.9754
Epoch 13/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0405 - accuracy: 0.9889 - val_loss: 0.1165 - val_accuracy: 0.9735
Epoch 14/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0382 - accuracy: 0.9877 - val_loss: 0.0794 - val_accuracy: 0.9792
Epoch 15/40
132/132 [==============================] - 0s 3ms/step - loss: 0.0354 - accuracy: 0.9898 - val_loss: 0.0990 - val_accuracy: 0.9744
```

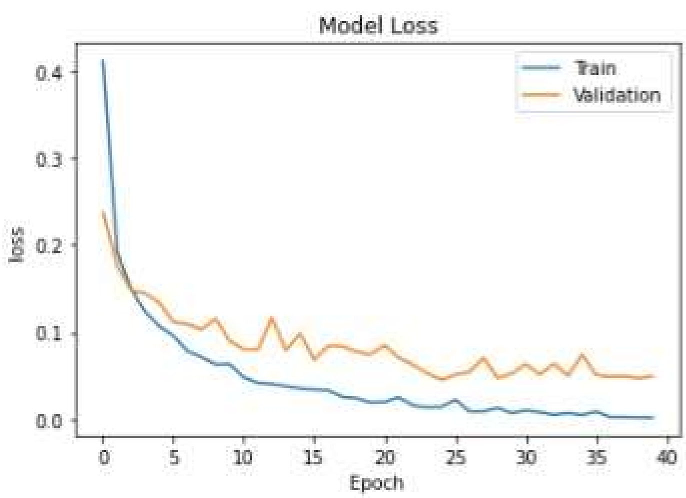
Step 5:

Post processing

Evaluated the model using the test set created at the beginning.

Plotting the graphs





Step 6:

Saving the model and its weights.

```
model.save("weights.h5") #saving weights
```

Final performance measures

```
print("Validation Accuracy:",val_score[1])
print("Validation Loss:",val_score[0])
print("f1_score:",f1_score(y_test,model.predict_classes(X_test)))
print("recall:",recall_score(y_test,model.predict_classes(X_test)))
print("precision_score:",precision_score(y_test,model.predict_classes(X_test)))
```

```
Validation Accuracy: 0.9931818246841431
Validation Loss: 0.014487503096461296
f1_score: 0.9772151898734178
recall: 0.965
precision_score: 0.9897435897435898
```

CONCLUSION:

The above model can be further improved by tweaking the hyperparameters like numbers of dense layer, number of neurons, adding dropouts, changing the epochs and optimizers.