model

February 20, 2020

1 Classification Using ANN(Artifical Nural Network)

1.0.1 ———Install Packages———-

Tensorflow –> For Fast Numeric Computation

conda create -n tensorflow

Keras –> Wrap up of tensorflow/THeano which can reduce the size of code

pip install –upgrade keras

[1]:
```
#For Suppressing TensorFlow warnings of an older version
import warnings
warnings.simplefilter("ignore")
```

1.0.2 ———Pre Processing Data ———-

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix,classification_report
from keras.models import Sequential
from keras.layers import Dense
from keras.models import load_model
```

```
Using TensorFlow backend.
```

[3]:
```
dataset=pd.read_csv('musk_csv.csv')
dataset.head()
```

[3]:
```
   ID molecule_name conformation_name  f1   f2   f3  f4   f5  f6  f7  …  \
0   1      MUSK-211           211_1+1  46 -108  -60 -69 -117  49  38  …
1   2      MUSK-211          211_1+10  41 -188 -145  22 -117  -6  57  …
2   3      MUSK-211          211_1+11  46 -194 -145  28 -117  73  57  …
3   4      MUSK-211          211_1+12  41 -188 -145  22 -117  -7  57  …
4   5      MUSK-211          211_1+13  41 -188 -145  22 -117  -7  57  …

   f158  f159  f160  f161  f162  f163  f164  f165  f166  class
```

```
0  -308    52    -7    39   126   156   -50  -112    96       1
1   -59    -2    52   103   136   169   -61  -136    79       1
2  -134  -154    57   143   142   165   -67  -145    39       1
3   -60    -4    52   104   136   168   -60  -135    80       1
4   -60    -4    52   104   137   168   -60  -135    80       1

[5 rows x 170 columns]
```

Separating the Dependent , Independent , None Relative variables None Relavent Variables :-

ID,Molecule_Name,Conformation_Name

[4]:

```
#Features Variables(Independent Variables)
X=dataset.iloc[:,3:169].values
print(X)
```

```
[[  46 -108   -60 …  -50 -112   96]
 [  41 -188 -145 …  -61 -136   79]
 [  46 -194 -145 …  -67 -145   39]
 …
 [  44 -102   -19 …  -66 -144   -6]
 [  51 -121   -23 …  -44 -116  117]
 [  51 -122   -23 …  -44 -115  118]]
```

[5]:

```
#Target Variable(Dependent Variable)
Y=dataset.iloc[:,169].values
print(Y)
```

```
[1 1 1 … 0 0 0]
```

All data are in Numerical form (No categorical Data) We can directly process to splitting the data

Splitting into random 80:20 train test data

[6]:

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.
 ↪20,random_state=42)
print("X_Train")
print(X_train.shape)
print("Y_Train")
print(Y_train.shape)
print("X_Test")
print(X_test.shape)
print("Y_Test")
print(Y_test.shape)
```

```
X_Train
(5278, 166)
Y_Train
(5278,)
X_Test
(1320, 166)
Y_Test
(1320,)
```

Features Scalling Avoid one independent variable dominating another one

Avoid Biasing of Independent Variables

Make Computation Easy

[7]:
```
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
print("X_train")
print(X_train)
print("X_test")
print(X_test)
```

```
X_train
[[-0.29629308 -0.31524209  1.54819801 … -0.32936004  0.04844194
  -0.53219017]
 [-0.44773392 -0.84660709 -0.95528227 … -0.09734996 -0.04243066
   0.84155128]
 [-0.22057266 -0.83553699 -0.49876528 … -0.43644161 -0.36697565
  -0.34197981]
 …
 [-0.42880382 -0.04955959 -1.01418769 … -0.0616561  -0.09435786
   0.91552197]
 [-0.01234151  1.61095605  1.79854604 …  0.13466012  0.30807793
   1.02119439]
 [-0.33415329 -0.8687473  -1.26453572 … -0.22227846  0.30807793
   0.1652478 ]]
X_test
[[-0.29629308 -0.8576772  -1.13199853 … -0.45428854 -0.61362985
  -0.4899212 ]
 [-0.42880382 -0.03848948 -0.99946134 … -0.0616561  -0.15926686
   0.93665646]
 [-0.39094361  2.08697053  1.5187453  … -0.40074775 -1.21079264
  -0.66956431]
 …
 [-0.29629308 -0.63627511  0.53207954 … -0.3650539  -0.13330326
  -0.59559362]
 [-0.39094361 -0.14919052 -0.99946134 …  0.04542547 -0.35399385
   1.14800129]
```

```
[-0.40987371  0.13863219  1.20949185 …  0.02757854 -0.35399385
  1.13743405]]
```

1.0.3 ——— Create A Model ———-

[8]:
```
#Initialization the ANN
classifier=Sequential()

#Adding ip layer And first hidden layer
classifier.add(
            Dense(activation="relu", input_dim=166, units=83,
               kernel_initializer="uniform"
                    )
            )
#units=no of nodes in hidden layer --> Peremeter Tuning or avg(no of nodes in␣
 ↪ip layer,op layer)
#activation --> hidden layer (rectify fn) and op layer(sigmoid fn)
#input_dim=no of nodes in ip layer --> no of independent variables

#Adding 2nd Hidden Layer
classifier.add(
            Dense(activation="relu", units=83,
               kernel_initializer="uniform"
                    )
            )



#Adding Op Layer
classifier.add(
            Dense(activation="sigmoid", units=1,
               kernel_initializer="uniform"
                    )
            )
#output_dim=1--> Binary Classification
#activation --> sigmoid --> Probability of binary outcome




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

# loss--> logarithmic loss
# binary -->binary_crossentropy
# accuracy matrix is use to compute the optimal value of weight in next␣
 ↪itteration
```

```
classifier.summary()
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-
packages\tensorflow\python\ops\nn_impl.py:180:
add_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_ops) is
deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 83)                13861
_________________________________________________________________
dense_2 (Dense)              (None, 83)                6972
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 84
=================================================================
Total params: 20,917
Trainable params: 20,917
Non-trainable params: 0
_________________________________________________________________
```

Train a Model

[9]:

```
history=classifier.
 ↪fit(X_train,Y_train,batch_size=32,epochs=10,validation_split=.33)
#epoch --> number of time whole dataset is passed from model
#batch size--> number of observation after which you want to update the wight
#history will be used in graph plot
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-
packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables
is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 3536 samples, validate on 1742 samples
Epoch 1/10
3536/3536 [==============================] - 1s 219us/step - loss: 0.3340 -
accuracy: 0.8566 - val_loss: 0.1962 - val_accuracy: 0.9214
Epoch 2/10
3536/3536 [==============================] - 0s 86us/step - loss: 0.1503 -
accuracy: 0.9432 - val_loss: 0.1074 - val_accuracy: 0.9621
Epoch 3/10
```

```
3536/3536 [==============================] - 0s 95us/step - loss: 0.0955 -
accuracy: 0.9675 - val_loss: 0.0713 - val_accuracy: 0.9765
Epoch 4/10
3536/3536 [==============================] - 0s 93us/step - loss: 0.0701 -
accuracy: 0.9743 - val_loss: 0.0581 - val_accuracy: 0.9765
Epoch 5/10
3536/3536 [==============================] - 0s 90us/step - loss: 0.0556 -
accuracy: 0.9782 - val_loss: 0.0441 - val_accuracy: 0.9822
Epoch 6/10
3536/3536 [==============================] - 0s 99us/step - loss: 0.0385 -
accuracy: 0.9873 - val_loss: 0.0519 - val_accuracy: 0.9782
Epoch 7/10
3536/3536 [==============================] - 0s 86us/step - loss: 0.0299 -
accuracy: 0.9898 - val_loss: 0.0343 - val_accuracy: 0.9879
Epoch 8/10
3536/3536 [==============================] - 0s 89us/step - loss: 0.0204 -
accuracy: 0.9932 - val_loss: 0.0349 - val_accuracy: 0.9879
Epoch 9/10
3536/3536 [==============================] - 0s 91us/step - loss: 0.0170 -
accuracy: 0.9935 - val_loss: 0.0398 - val_accuracy: 0.9856
Epoch 10/10
3536/3536 [==============================] - 0s 104us/step - loss: 0.0151 -
accuracy: 0.9952 - val_loss: 0.0331 - val_accuracy: 0.9885
```

[10]:
```python
#Saving A model

classifier.save('my_model.h5')
```

1.0.4 ————————Post Processing Of Data————————

[11]:
```python
y_pred=classifier.predict(X_test)
#it returns probability but we need binary value 0/1
#0-->Non Musk
#1-->Musk
print(y_pred)
```

```
[[1.0848045e-05]
 [5.9604645e-08]
 [2.8362870e-04]
 …
 [7.7334046e-04]
 [0.0000000e+00]
 [0.0000000e+00]]
```

Thresholding value>0.5 –> 1

[12]:
```
y_pred=(y_pred>0.50 )
#It Will return boolean
y_pred=y_pred.astype('int64')
#Covert in int
print(y_pred)
```

```
[[0]
 [0]
 [0]
 …
 [0]
 [0]
 [0]]
```

1.0.5 ———Result Testing And Analysis————–

[13]:
```
cm=confusion_matrix(Y_test,y_pred)
print("Confusion Matrix")
print(cm)
print("Analysis OF Confusion Martix")
print("True Positive : "+str(cm[1][1]))
print("False Positive : "+str(cm[0][1]))
print("True Negative : "+str(cm[0][0]))
print("False Negative : "+str(cm[1][0]))
```

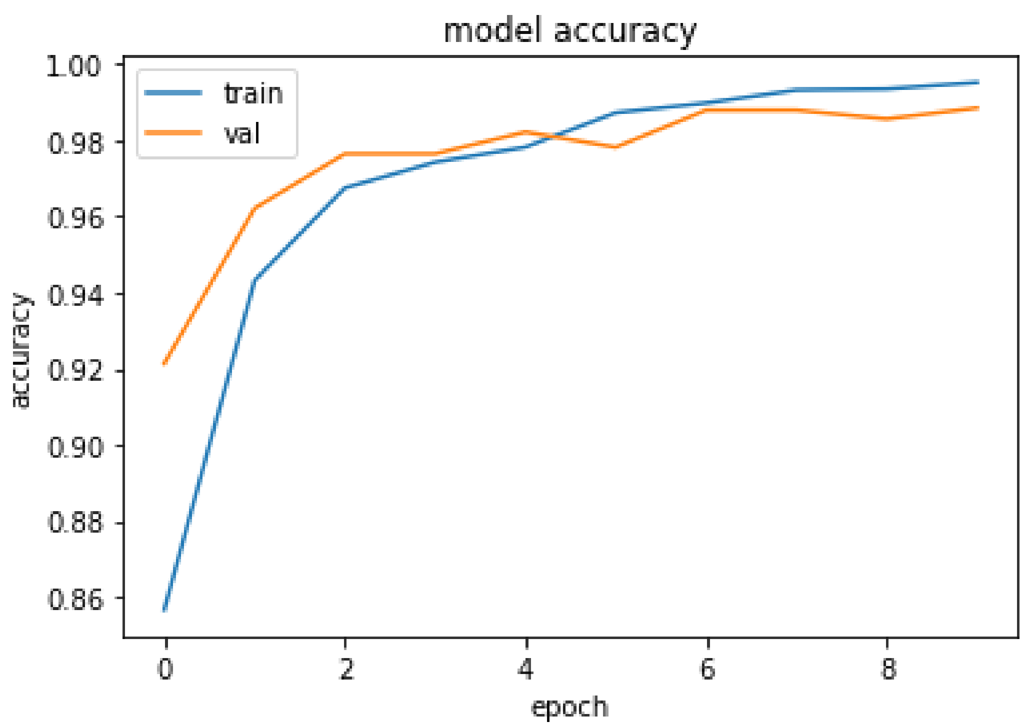
```
Confusion Matrix
[[1108    3]
 [  15  194]]
Analysis OF Confusion Martix
True Positive : 194
False Positive : 3
True Negative : 1108
False Negative : 15
```

[14]:
```
print(classification_report(Y_test,y_pred))
```

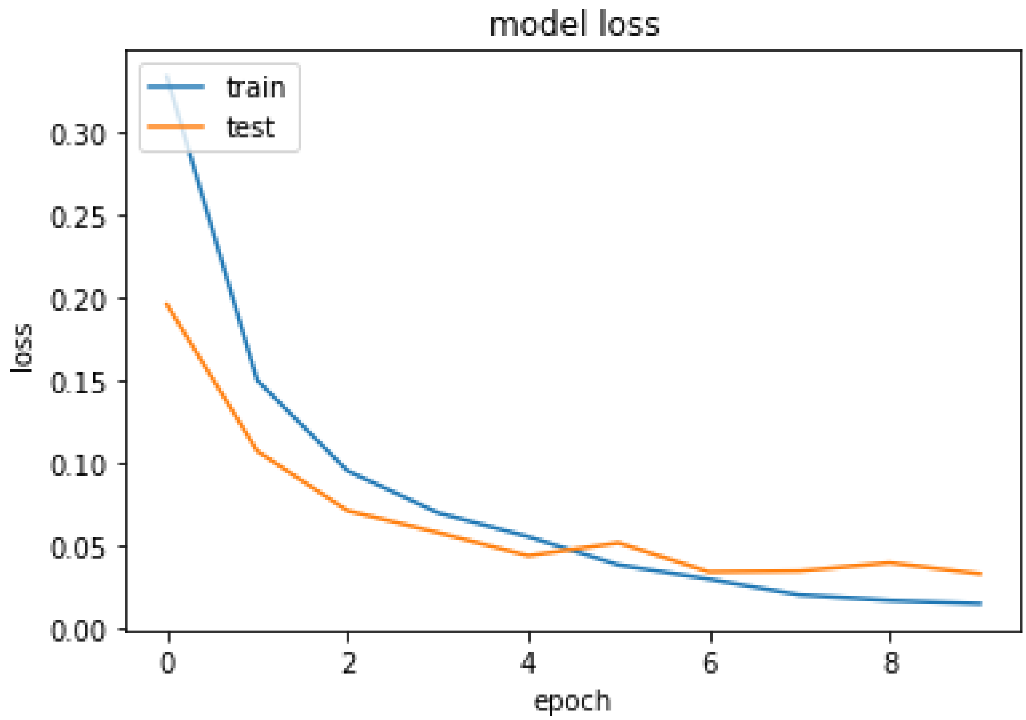
```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      1111
           1       0.98      0.93      0.96       209

    accuracy                           0.99      1320
   macro avg       0.99      0.96      0.97      1320
weighted avg       0.99      0.99      0.99      1320
```

1.0.6 ———Graph OF accuracy And loss———

[15]:
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



[16]:
```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



1.0.7 ——Conclusion ————

We are getting almost similar accuracy for both training and as well as test data set

Our Model Is Capable of solving given buissness Problem

[ ]:

```

```

[ ]:

```

```