Week4_10_ANN_sportsmodel

May 1, 2021

Artificial Neural Network (ANN): Classification

- In this model, the intention is a person is sports person or not.
- *This model is simple and predicted well.*

- **Import Libraries**

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

import tensorflow_hub as hub
import tensorflow_datasets as tfds
from tensorflow.keras.layers import Activation, Dense, Embedding
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization

import re
import nltk
import string
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords

import warnings
warnings.filterwarnings("ignore")
```

- **Downloading packages**

[2]:
```
nltk.download("stopwords")
stop_words = set(stopwords.words('english'))
wordnet_lemmatizer = WordNetLemmatizer()
```

```
nltk.download('wordnet')
nltk.download("punkt")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/preethamvignesh/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/preethamvignesh/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/preethamvignesh/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[2]: True

- **Cleaning the text, plotting, and prediction functions**

[3]:
```
#Cleans Text
def normalizer(tweet):
    no_urls = re.sub(r"http\S+", " " ,tweet)
    only_letters = re.sub("[^a-zA-Z]", " ",no_urls)
    tokens = nltk.word_tokenize(only_letters)[2 :]
    lower_case = [l.lower() for l in tokens]
    filtered_result = list(filter(lambda l: l not in stop_words, lower_case))
    #lemmas = [wordnet_lemmatizer.lemmatize(t) for t in filtered_result]
    return filtered_result

#Generate Plots for Model
def plot_graphs(history, metric):
  plt.figure( figsize=(16,12))
  plt.rcParams.update({'font.size': 22})
  plt.plot(history.history[metric])
  #plt.plot(history.history['val_'+metric], '')
  plt.xlabel("Epochs")
  plt.ylabel(metric)
  plt.legend([metric, 'val_'+metric])
  plt.show()

#make a prediction on input text
def predict_on_text(text):

  test_text = np.array([text])
  test_df =  pd.DataFrame(test_text, columns = ['text'])

  test_df['normalized_tweet'] = test_df.text.apply(normalizer)
  X = test_df["normalized_tweet"].astype(str)
```

```
  df = token.texts_to_sequences(X)

  df = tf.keras.preprocessing.sequence.pad_sequences(df, maxlen=max_length)

  prediction = np.round(model.predict(df)[0][0])

  if prediction:

    return "This is about Sports."

  else:

    return "This is not about Sports."

# compute accuracy
def accuracy(y, y_hat):

  acc = np.mean(y == y_hat)

  print('The accuracy is: ' + str(acc))

#generates confusion matrix
def confusionMatrix(ys,preds):

    N = len(ys)
    #Generate empty matrix
    confuse = np.zeros((2,2),dtype=int)

    #loop through both arrays
    for i in range(N):

      #increase count in entry of each label
      confuse[ys[i],int(preds[i])] = confuse[ys[i],int(preds[i])] + 1

    #return as dataframe
    return pd.DataFrame(confuse)
```

[4]:
```
import matplotlib.pyplot as plt
plt.style.use('ggplot')

def plot_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    x = range(1, len(acc) + 1)
```

```
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(x, acc, 'b', label='Training acc')
    plt.plot(x, val_acc, 'r', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(x, loss, 'b', label='Training loss')
    plt.plot(x, val_loss, 'r', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()
```

- **Read the datasets**

[5]:
```
#uploaded = files.upload()

df = pd.read_csv("~/Desktop/Work/ML_EIT/Data/sports_data.csv")

#map data labels to 0 or 1
df.label = df["label"].replace({"not_sports":0,"sports":1})

#add cleaned data to the DataFrame
df['normalized_tweet'] = df.title.apply(normalizer)

#seperate X and y out as DataFrames
X = df["normalized_tweet"].astype(str)
y = df["label"]
```

[6]:
```
df.head()
```

[6]:
```
   Unnamed: 0                                              title   subreddit  \
0           0  Fake News Report: Orient Daily Newspaper Tende…    religion
1           1         How to get signed/tryout for any ACB team?  Basketball
2           2                                        So calming      Forest
3           3  How violent is Rezero compared to Attack on ti…       anime
4           4                  Forest near a little vilage in Mk      Forest

   label                                    normalized_tweet
0      0  [report, orient, daily, newspaper, tenders, ap…
1      1                  [get, signed, tryout, acb, team]
2      0                                                []
3      0   [rezero, compared, attack, titan, vinland, saga]
4      0                               [little, vilage, mk]
```

[7]:
```
max_features = 50000 #we set maximum number of words to 5000
max_length = 100 #we set maximum sequence length to 400
embedding_dim = 50
```

```
token = tf.keras.preprocessing.text.Tokenizer(num_words=max_features)
token.fit_on_texts(X)

vocab_size = len(token.word_index) + 1

#conver X to a TensorFlow Type and add padding
df = token.texts_to_sequences(X)
df = tf.keras.preprocessing.sequence.pad_sequences(df, maxlen=max_length)
```

- **Split the datasets**

[8]:
```
df = df.astype('int')
X_train, X_test, y_train, y_test = train_test_split(df, y,test_size=0.
↪15,random_state=101)

X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
↪10, random_state=1)
```

[9]:
```
# determine the number of input features
n_features = X_train.shape[1]
n_features
```

[9]: 100

- **Model architecture**

[10]:
```
from keras.models import Sequential
from keras import layers

model = Sequential()
model.add(layers.Embedding(input_dim=vocab_size,
                           output_dim=embedding_dim,
                           input_length=max_length))
# model.add(layers.Flatten())
model.add(layers.GlobalMaxPool1D())
model.add(layers.Dense(16,activation='relu'))
model.add(layers.Dense(1,activation='sigmoid',kernel_initializer='uniform'))

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
embedding (Embedding)        (None, 100, 50)           662800
_________________________________________________________________
global_max_pooling1d (Global (None, 50)                0
_________________________________________________________________
dense (Dense)                (None, 16)                816
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 663,633
Trainable params: 663,633
Non-trainable params: 0
_________________________________________________________________
```

[11]:
```
history=model.fit(np.array(X_train), np.array(y_train), batch_size=50,epochs=10)
```

```
Epoch 1/10
185/185 [==============================] - 2s 9ms/step - loss: 0.6791 -
accuracy: 0.5770
Epoch 2/10
185/185 [==============================] - 2s 9ms/step - loss: 0.3456 -
accuracy: 0.9110
Epoch 3/10
185/185 [==============================] - 2s 8ms/step - loss: 0.1417 -
accuracy: 0.9541
Epoch 4/10
185/185 [==============================] - 2s 8ms/step - loss: 0.0813 -
accuracy: 0.9716
Epoch 5/10
185/185 [==============================] - 1s 8ms/step - loss: 0.0620 -
accuracy: 0.9768
Epoch 6/10
185/185 [==============================] - 2s 9ms/step - loss: 0.0609 -
accuracy: 0.9723
Epoch 7/10
185/185 [==============================] - 2s 9ms/step - loss: 0.0532 -
accuracy: 0.9774
Epoch 8/10
185/185 [==============================] - 1s 8ms/step - loss: 0.0518 -
accuracy: 0.9765
Epoch 9/10
185/185 [==============================] - 2s 8ms/step - loss: 0.0525 -
accuracy: 0.9761
Epoch 10/10
185/185 [==============================] - 1s 8ms/step - loss: 0.0509 -
accuracy: 0.9752
```

[12]:
```
model.evaluate(X_test,y_test)
```

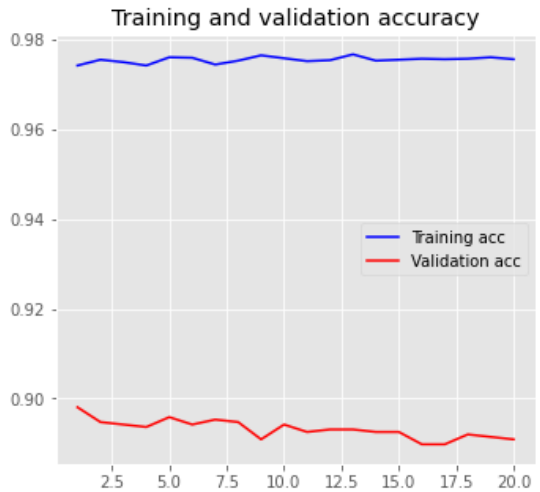
```
57/57 [==============================] - 0s 1ms/step - loss: 0.2884 - accuracy:
0.8942
```

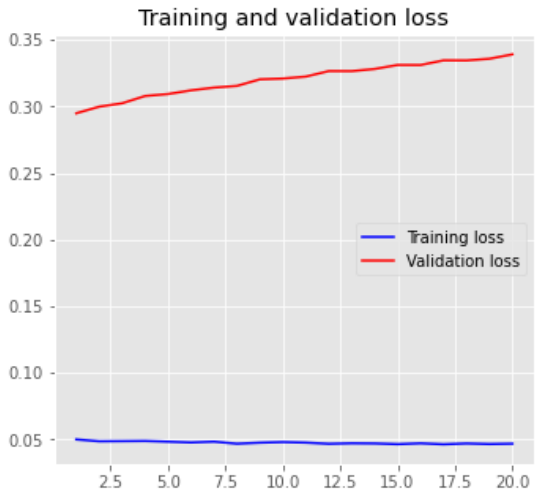
[12]:
```
[0.28838300704956055, 0.8941565752029419]
```

[13]:
```
history = model.fit(X_train, y_train,
                    epochs=20,
                    verbose=True,
                    validation_data=(X_test, y_test),
                    batch_size=52)

loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: {:.4f}".format(accuracy))
loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
print("Testing Accuracy:  {:.4f}".format(accuracy))
plot_history(history)
```

```
Epoch 1/20
178/178 [==============================] - 2s 13ms/step - loss: 0.0500 -
accuracy: 0.9743 - val_loss: 0.2950 - val_accuracy: 0.8980
Epoch 2/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0486 -
accuracy: 0.9756 - val_loss: 0.2999 - val_accuracy: 0.8947
Epoch 3/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0487 -
accuracy: 0.9750 - val_loss: 0.3025 - val_accuracy: 0.8942
Epoch 4/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0488 -
accuracy: 0.9743 - val_loss: 0.3080 - val_accuracy: 0.8936
Epoch 5/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0483 -
accuracy: 0.9761 - val_loss: 0.3094 - val_accuracy: 0.8958
Epoch 6/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0477 -
accuracy: 0.9760 - val_loss: 0.3122 - val_accuracy: 0.8942
Epoch 7/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0483 -
accuracy: 0.9745 - val_loss: 0.3143 - val_accuracy: 0.8953
Epoch 8/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0468 -
accuracy: 0.9753 - val_loss: 0.3155 - val_accuracy: 0.8947
Epoch 9/20
178/178 [==============================] - 2s 10ms/step - loss: 0.0475 -
accuracy: 0.9765 - val_loss: 0.3205 - val_accuracy: 0.8908
Epoch 10/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0480 -
accuracy: 0.9759 - val_loss: 0.3211 - val_accuracy: 0.8942
Epoch 11/20
```

```
178/178 [==============================] - 2s 9ms/step - loss: 0.0476 -
accuracy: 0.9752 - val_loss: 0.3225 - val_accuracy: 0.8925
Epoch 12/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0467 -
accuracy: 0.9754 - val_loss: 0.3267 - val_accuracy: 0.8931
Epoch 13/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0471 -
accuracy: 0.9767 - val_loss: 0.3266 - val_accuracy: 0.8931
Epoch 14/20
178/178 [==============================] - 1s 8ms/step - loss: 0.0470 -
accuracy: 0.9753 - val_loss: 0.3283 - val_accuracy: 0.8925
Epoch 15/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0464 -
accuracy: 0.9756 - val_loss: 0.3313 - val_accuracy: 0.8925
Epoch 16/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0471 -
accuracy: 0.9758 - val_loss: 0.3312 - val_accuracy: 0.8897
Epoch 17/20
178/178 [==============================] - 1s 8ms/step - loss: 0.0463 -
accuracy: 0.9757 - val_loss: 0.3347 - val_accuracy: 0.8897
Epoch 18/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0469 -
accuracy: 0.9758 - val_loss: 0.3347 - val_accuracy: 0.8920
Epoch 19/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0465 -
accuracy: 0.9761 - val_loss: 0.3359 - val_accuracy: 0.8914
Epoch 20/20
178/178 [==============================] - 2s 9ms/step - loss: 0.0468 -
accuracy: 0.9757 - val_loss: 0.3392 - val_accuracy: 0.8908
Training Accuracy: 0.9778
Testing Accuracy:  0.8908
```





- Predict the test data and evaluate the confusion matrix and accuracy

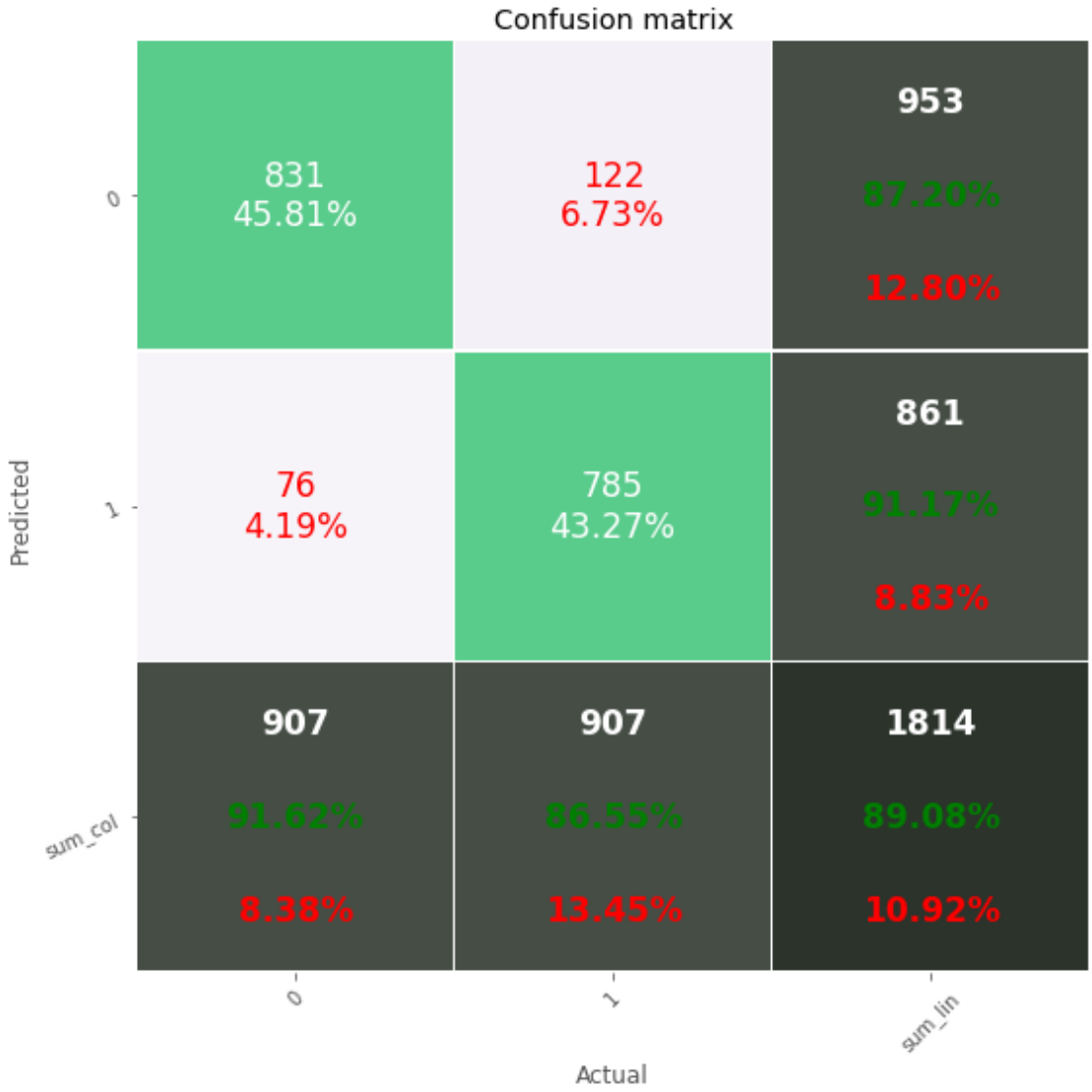
[14]:
```
y_pred = model.predict(X_test)
y_pred = y_pred > 0.5

con_res = confusion_matrix(y_test, y_pred)
print("Confusion matrix:")
print(confusion_matrix(y_test,y_pred))
print("Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred)*100))
```

```
Confusion matrix:
[[831  76]
 [122 785]]
Accuracy: 89.08%
```

Confusion matrix

- More information and different ways of confusion matrix: https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

[16]:
```
%run -i '/Users/preethamvignesh/Desktop/Work/ML_EIT/
 ↪confusion_matrix_different_ways.py'
df_confmatrx = pd.DataFrame(con_res, range(2),range(2))
df_confmatrx
cmap = 'PuRd'
confusion_matrix_dfrntway(df_confmatrx, cmap=cmap,fz=17)
```


Confusion matrix
831
45.81%
122
6.73%
953
87.20%
12.80%
76
4.19%
785
43.27%
861
91.17%
8.83%
907
91.62%
8.38%
907
86.55%
13.45%
1814
89.08%
10.92%
Predicted
0
1
sum_col
0
1
sum_lin
Actual

[ ]: