This code is for performing various machine learning classification algorithms on a dataset containing emotional speech recordings. The code imports various libraries such as numpy, pandas, tensorflow, seaborn, and matplotlib.pyplot. It also sets a warning filter to ignore future warnings.

The code then reads in a .csv file and saves it to a Pandas dataframe variable df. The dataframe is explored through various methods like head(), info(), corr(), unique(), and isnull().sum().sum(). The EMOTION column of the dataframe is the target variable and the other columns are the input features.

The code then performs various classification algorithms like logistic regression, random forest, gradient boosting, and support vector machines (SVM). It also performs some preprocessing steps like one-hot encoding and PCA (Principal Component Analysis). For each algorithm, it first performs a grid search to find the best hyperparameters, fits the model on the training data, predicts on the test data, and evaluates the performance using classification\_report and confusion\_matrix functions.

Finally, the code visualizes the confusion matrices using sns.heatmap() function for each algorithm's predictions.

import warnings

warnings.simplefilter(action='ignore', category=**FutureWarning**)

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

import tensorflow as tf

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

*# Input data files are available in the "../input/" directory.*

*# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory*

import os

print(os.listdir("D:\DNO\major"))

for df **in** ("D:\DNO\major"):

df=pd.read\_csv("D:\DNO\major/preprocessing.csv").fillna(0)

*# Any results you write to the current directory are saved as output.*

df.head()

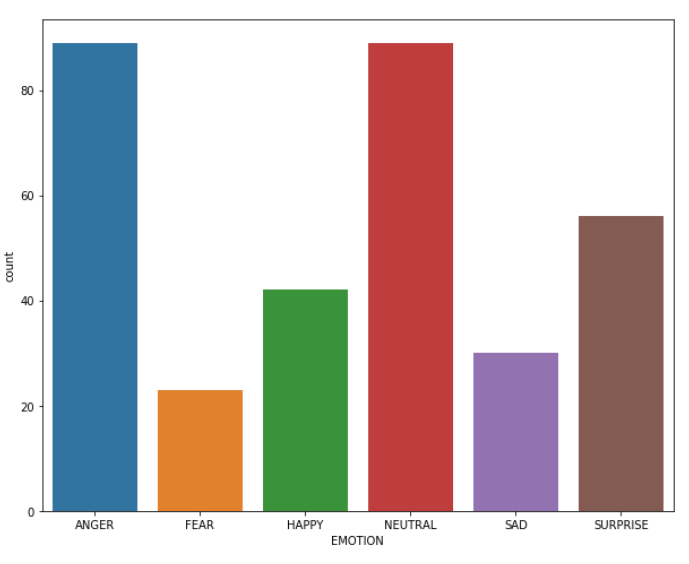
df.info()

df.corr()

df['EMOTION'].unique()

plt.figure(figsize = (10, 8))

sns.countplot(df['EMOTION'])

plt.show()

df['EMOTION'].value\_counts()

df.isnull().sum().sum() *#no missing values*

*#split into features and labels sets*

X = df.drop(['EMOTION','ID'], axis = 1) *#features*

y = df['EMOTION'] *#labels*

X.head()

X.info()

print("Total number of labels: **{}**".format(df.shape[0]))

target = df.ID

X.dtypes.sample(104)

one\_hot\_encoded\_training\_predictors = pd.get\_dummies(X)

one\_hot\_encoded\_test\_predictors = pd.get\_dummies(y)

final\_train, final\_test = one\_hot\_encoded\_training\_predictors.align(one\_hot\_encoded\_test\_predictors,join='left', axis=1)

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)

from sklearn.linear\_model import LogisticRegression

m1 = LogisticRegression()

m1.fit(X\_train, y\_train)

pred1 = m1.predict(X\_test)

from sklearn.metrics import classification\_report, confusion\_matrix

print(classification\_report(y\_test, pred1))

labels = ['ANGRY','FEAR','HAPPY','NEUTRAL','SAD','SURPRISE']

cm1 = pd.DataFrame(confusion\_matrix(y\_test, pred1), index = labels, columns = labels)

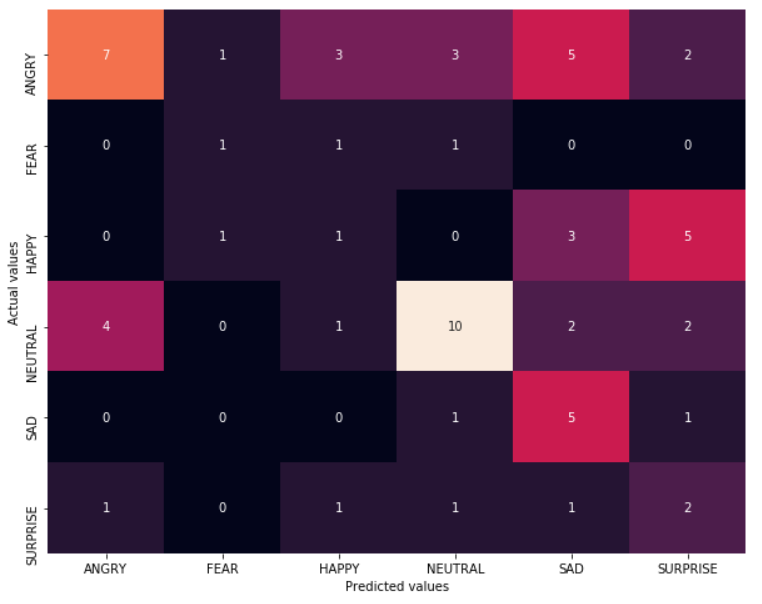
plt.figure(figsize = (10, 8))

sns.heatmap(cm1, annot = True, cbar = False, fmt = 'g')

plt.ylabel('Actual values')

plt.xlabel('Predicted values')

plt.show()



from sklearn.ensemble import RandomForestClassifier

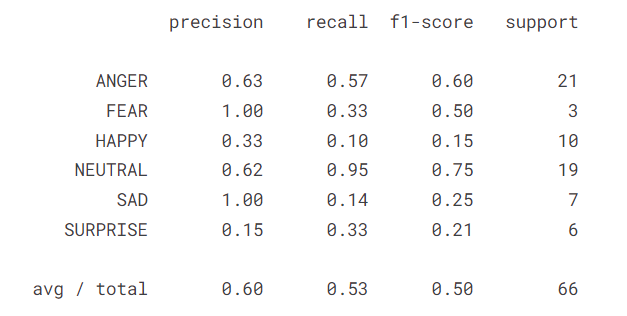
from sklearn.model\_selection import GridSearchCV

grid = {'n\_estimators': [10, 50, 100, 300]}

m2 = GridSearchCV(RandomForestClassifier(), grid)

m2.fit(X\_train, y\_train)

m2.best\_params\_ *#I got n\_estimators = 300*



cm2 = pd.DataFrame(confusion\_matrix(y\_test, pred2), index = labels, columns = labels)

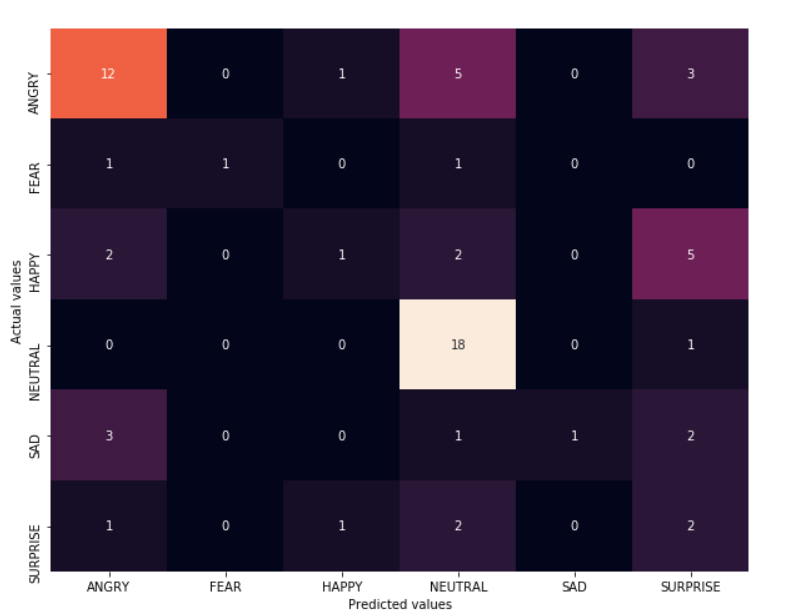
plt.figure(figsize = (10, 8))

sns.heatmap(cm2, annot = True, cbar = False, fmt = 'g')

plt.ylabel('Actual values')

plt.xlabel('Predicted values')

plt.show()



from sklearn.ensemble import GradientBoostingClassifier

grid = {

'learning\_rate': [0.03, 0.1, 0.5],

'n\_estimators': [100, 300],

'max\_depth': [1, 3, 9]

}

m3 = GridSearchCV(GradientBoostingClassifier(), grid, verbose = 2)

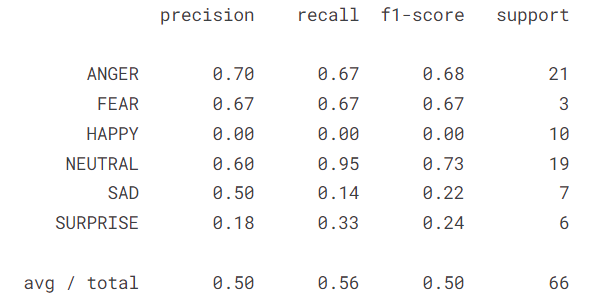
m3.fit(X\_train, y\_train)

m3.best\_params\_

{'learning\_rate': 0.03, 'max\_depth': 1, 'n\_estimators': 100}

pred3 = m3.predict(X\_test)

print(classification\_report(y\_test, pred3))



cm3 = pd.DataFrame(confusion\_matrix(y\_test, pred3), index = labels, columns = labels)

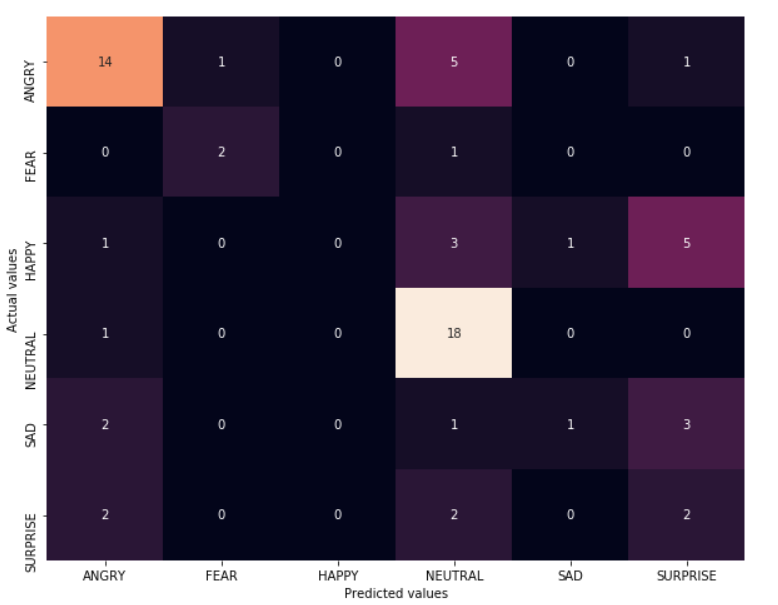
plt.figure(figsize = (10, 8))

sns.heatmap(cm3, annot = True, cbar = False, fmt = 'g')

plt.ylabel('Actual values')

plt.xlabel('Predicted values')

plt.show()



from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

scaler = StandardScaler()

scaler.fit(X\_train)

X\_sc\_train = scaler.transform(X\_train)

X\_sc\_test = scaler.transform(X\_test)

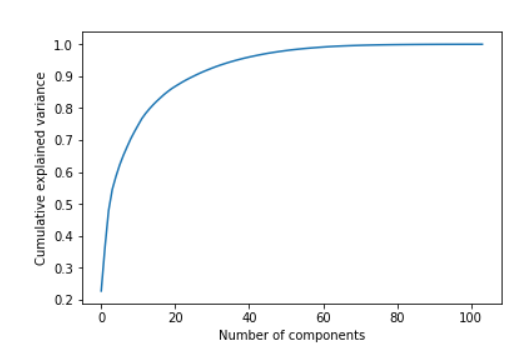
pca = PCA(n\_components=104)

pca.fit(X\_train)

plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))

plt.xlabel('Number of components')

plt.ylabel('Cumulative explained variance')



NCOMPONENTS = 104

pca = PCA(n\_components=NCOMPONENTS)

X\_pca\_train = pca.fit\_transform(X\_sc\_train)

X\_pca\_test = pca.transform(X\_sc\_test)

pca\_std = np.std(X\_pca\_train)

print(X\_sc\_train.shape)

print(X\_pca\_test.shape)

inv\_pca = pca.inverse\_transform(X\_pca\_train)

inv\_sc = scaler.inverse\_transform(inv\_pca)

grid = {

'C': [1,5,50],

'gamma': [0.05,0.1,0.5,1,5]

}

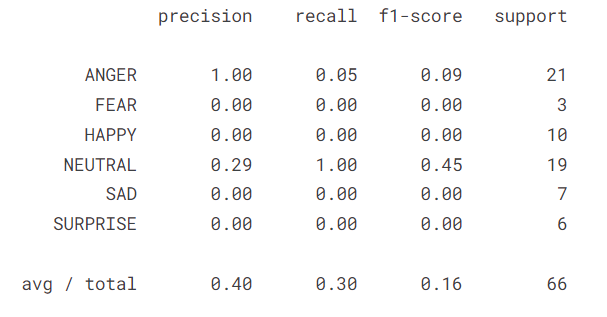
m5 = GridSearchCV(SVC(), grid)

m5.fit(X\_train, y\_train)

m5.best\_params\_ *#I got C = 1, gamma = 0.05*

pred5 = m5.predict(X\_test)

print(classification\_report(y\_test, pred5))



cm5 = pd.DataFrame(confusion\_matrix(y\_test, pred5), index = labels, columns = labels)

plt.figure(figsize = (10, 8))

sns.heatmap(cm5, annot = True, cbar = False, fmt = 'g')

plt.ylabel('Actual values')

plt.xlabel('Predicted values')

plt.show()

