**PROJECT CODE**

**Main.py**

import os  
from uuid import uuid4  
from flask import Flask, request, render\_template, send\_from\_directory, jsonify  
from flask\_restful import Resource, Api    
import numpy as np  
import cv2  
import detection  
app = Flask(\_\_name\_\_)  
APP\_ROOT = os.path.dirname(os.path.abspath(\_\_file\_\_))  
  
  
# api= API(app)  
  
# class HelloWorld(Resource):  
#     def get(self):  
#         return {'hello': 'world'}  
  
# api.add\_resource(HelloWorld, '/')  
  
  
model = r'detection.py'  
model = os.path.join(os.path.dirname(\_\_file\_\_),model)  
  
if not os.path.isfile(model):  
    print("model is missing from the model folder")  
    exit()  
  
  
@app.route("/")  
def index():  
    return render\_template("upload.html")  
  
@app.route("/upload", methods=["POST"])  
def upload():  
    target = os.path.join(APP\_ROOT, 'images/')  
  
    if not os.path.isdir(target):  
            os.mkdir(target)  
    else:  
        print("Couldn't create upload directory: {}".format(target))  
     
  
    for upload in request.files.getlist("file"):  
        filename = upload.filename  
        destination = "/".join([target, filename])  
        upload.save(destination)  
  
    if request.method=='POST':  
        input = cv2.imread(destination)  
        gender, age = detection.age\_gender\_detector(input)  
        # result= people = [{'gender': gender},{'age': age}]  
        # return jsonify(result)  
        # return send\_from\_directory('','frameface.jpg')  
        return render\_template("result.html", gender=gender, age=age)  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    app.run(debug=True)

**Detection.py**

import cv2 as cv  
import math  
import time  
# import cv2\_imshow  
  
def getFaceBox(net, frame, conf\_threshold=0.7):  
    frameOpencvDnn = frame.copy()  
    frameHeight = frameOpencvDnn.shape[0]  
    frameWidth = frameOpencvDnn.shape[1]  
    blob = cv.dnn.blobFromImage(frameOpencvDnn, 1.0, (300, 300), [104, 117, 123], True, False)  
  
    net.setInput(blob)  
    detections = net.forward()  
    bboxes = []  
    for i in range(detections.shape[2]):  
        confidence = detections[0, 0, i, 2]  
        if confidence > conf\_threshold:  
            x1 = int(detections[0, 0, i, 3] \* frameWidth)  
            y1 = int(detections[0, 0, i, 4] \* frameHeight)  
            x2 = int(detections[0, 0, i, 5] \* frameWidth)  
            y2 = int(detections[0, 0, i, 6] \* frameHeight)  
            bboxes.append([x1, y1, x2, y2])  
            cv.rectangle(frameOpencvDnn, (x1, y1), (x2, y2), (0, 255, 0), int(round(frameHeight/150)), 8)  
    return frameOpencvDnn, bboxes  
  
faceProto = r"modelNweight\modelNweight\opencv\_face\_detector.pbtxt"  
faceModel = r"modelNweight\modelNweight\opencv\_face\_detector\_uint8.pb"  
  
ageProto = r"modelNweight\modelNweight\age\_deploy.prototxt"  
ageModel = r"modelNweight\modelNweight\age\_net.caffemodel"  
  
genderProto = r"modelNweight\modelNweight\gender\_deploy.prototxt"  
genderModel = r"modelNweight\modelNweight\gender\_net.caffemodel"  
  
MODEL\_MEAN\_VALUES = (78.4263377603, 87.7689143744, 114.895847746)  
ageList = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)', '(48-53)', '(60-100)']  
genderList = ['Male', 'Female']  
  
# Load network  
ageNet = cv.dnn.readNet(ageModel, ageProto)  
genderNet = cv.dnn.readNet(genderModel, genderProto)  
faceNet = cv.dnn.readNet(faceModel, faceProto)  
  
padding = 20  
  
def age\_gender\_detector(frame):  
    # Read frame  
    t = time.time()  
    frameFace, bboxes = getFaceBox(faceNet, frame)  
    for bbox in bboxes:  
        # print(bbox)  
        face = frame[max(0,bbox[1]-padding):min(bbox[3]+padding,frame.shape[0]-1),max(0,bbox[0]-padding):min(bbox[2]+padding, frame.shape[1]-1)]  
  
        blob = cv.dnn.blobFromImage(face, 1.0, (227, 227), MODEL\_MEAN\_VALUES, swapRB=False)  
        genderNet.setInput(blob)  
        genderPreds = genderNet.forward()  
        gender = genderList[genderPreds[0].argmax()]  
        ageNet.setInput(blob)  
        agePreds = ageNet.forward()  
        age = ageList[agePreds[0].argmax()]  
  
        label = "{},{}".format(gender, age)  
        cv.putText(frameFace, label, (bbox[0], bbox[1]-10), cv.FONT\_HERSHEY\_SIMPLEX, 3.5, (0,255,255), 2, cv.LINE\_AA)  
        cv.imshow("Age Gender Demo", frameFace)  
        cv.imwrite("static/frameface.jpg", frameFace)  
  
    return gender, age

**Classification.py**

import numpy as np # linear algebra  
import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)  
import matplotlib.pyplot as plt  
import seaborn as sns  
#from wordcloud import WordCloud  
  
#to data preprocessing  
from sklearn.compose import ColumnTransformer  
from sklearn.preprocessing import OneHotEncoder  
  
#NLP tools  
import re  
import string  
from nltk.corpus import stopwords  
import nltk  
from nltk import corpus  
import nltk.corpus as Corpus  
# from nltk.corpus import stopwords  
from nltk.corpus import stopwords  
nltk.download("stopwords")  
STOPWORDS = nltk.corpus.stopwords.words('english')  
from nltk.stem.porter import PorterStemmer  
from sklearn.feature\_extraction.text import CountVectorizer  
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split, GridSearchCV  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.metrics import f1\_score, accuracy\_score, precision\_score, recall\_score, make\_scorer  
from time import time  
  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.svm import LinearSVC  
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier, BaggingClassifier  
from sklearn.linear\_model import LogisticRegression, SGDClassifier  
from sklearn.neighbors import KNeighborsClassifier  
import joblib  
  
  
  
#train split and fit models  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn import svm  
#from xgboost import XGBClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.datasets import make\_classification  
  
#model selection  
from sklearn.metrics import confusion\_matrix, accuracy\_score  
  
df = pd.read\_csv('public\_data.csv',index\_col=[0])  
df.head(6)  
  
[df.info](http://df.info/)()  
  
df.describe().T  
  
dt\_trasformed = df[['class', 'tweet']]  
y = dt\_trasformed.iloc[:, :-1].values  
  
sns.countplot(df['class'])  
plt.show()  
  
sns.countplot(df['hate\_speech'])  
plt.title('Number of people felt text as hate speech')  
plt.show()  
  
sns.countplot(df['offensive\_language'])  
plt.title('Number of people felt text as offensive language')  
plt.show()  
  
sns.countplot(df['neither'])  
plt.title('Number of people felt text as neither of them')  
plt.show()  
  
def plot\_dist(data):  
     
    fig,ax = plt.subplots(1,2,figsize=(15,5))  
    sns.distplot(data,ax=ax[0])  
    sns.boxplot(data,ax=ax[1])  
    plt.show()  
     
def plot\_dist(data):  
     
    fig,ax = plt.subplots(1,2,figsize=(15,5))  
    sns.distplot(data,ax=ax[0])  
    sns.boxplot(data,ax=ax[1])  
    plt.show()  
     
# average word length  
df['avg\_word\_len'] = df['tweet'].apply(lambda x: np.mean([len(c) for c in  x.split(' ')]))  
plot\_dist(df['avg\_word\_len']);  
  
# number of stopwords  
df['Number of stop words'] = df['tweet'].apply(lambda x: len([c for c in x.split(' ') if c in STOPWORDS]))  
sns.countplot(df['Number of stop words'])  
plt.show()  
  
# number of character count  
df['Number of char'] = df['tweet'].apply(lambda x: len(x))  
plot\_dist(df['Number of char']);  
  
# number of unique  
df['Number of unique words'] = df['tweet'].apply(lambda x: len(set(x.split(' '))))  
plot\_dist(df['Number of unique words']);  
  
# number of urls  
df['Number of urls'] = df['tweet'].apply(lambda x: len(re.findall('https?',x)))  
sns.countplot(df['Number of urls'])  
plt.show()  
  
# number of emoticons  
df['Number of emoticons'] = df['tweet'].apply(lambda x: len(re.findall('&#[0-9]+;',x)))  
sns.countplot(df['Number of emoticons'])  
plt.show()  
  
# number of punctuations  
df['Number of punctuations'] = df['tweet'].apply(lambda x: len([c for c in str(x) if c in string.punctuation]))  
plot\_dist(df['Number of punctuations']);  
  
# number of mentions  
df['Number of mentions'] = df['tweet'].apply(lambda x: len(re.findall('@',x)))  
sns.countplot(df['Number of mentions'])  
plt.show()  
  
# number of hastags  
df['Number of hashtags'] = df['tweet'].apply(lambda x: len(re.findall('#',x)))  
sns.countplot(df['Number of hashtags'])  
  
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])], remainder='passthrough')  
y = np.array(ct.fit\_transform(y))  
print(y)  
  
y\_df = pd.DataFrame(y)  
y\_hate = np.array(y\_df[0])  
y\_offensive = np.array(y\_df[1])  
print(y\_hate)  
print(y\_offensive)  
  
y\_df = pd.DataFrame(y)  
y\_hate = np.array(y\_df[0])  
y\_offensive = np.array(y\_df[1])  
print(y\_hate)  
print(y\_offensive)  
  
corpus=df['tweet']  
from sklearn.feature\_extraction.text import CountVectorizer  
cv = CountVectorizer()  
X = cv.fit\_transform(corpus).toarray()  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_hate, test\_size = 0.33, random\_state = 0)  
  
classifier\_np = GaussianNB()  
classifier\_np.fit(X\_train, y\_train)  
  
classifier\_dt = DecisionTreeClassifier(criterion = 'entropy', random\_state = 0)  
classifier\_dt.fit(X\_train, y\_train)  
  
classifier\_knn = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)  
classifier\_knn.fit(X\_train, y\_train)  
  
classifier\_lr = LogisticRegression(random\_state = 0)  
classifier\_lr.fit(X\_train, y\_train)  
  
classifier\_rf = RandomForestClassifier(n\_estimators = 10, criterion = 'entropy', random\_state = 0)  
classifier\_rf.fit(X\_train, y\_train)  
  
classifier\_svm = svm.SVC()  
classifier\_svm.fit(X\_train, y\_train)  
  
classifier\_xgb= XGBClassifier()  
classifier\_xgb.fit(X\_train, y\_train)  
  
classifier\_ab= AdaBoostClassifier()  
classifier\_ab.fit(X\_train, y\_train)  
  
#Naive Bayes  
y\_pred\_np = classifier\_np.predict(X\_test)  
cm = confusion\_matrix(y\_test, y\_pred\_np)  
print("Naive Bayes")  
print(cm)  
  
#KNN  
y\_pred\_knn = classifier\_knn.predict(X\_test)  
cm = confusion\_matrix(y\_test, y\_pred\_knn)  
print("KNN")  
print(cm)  
  
#XGBoost Classifier  
y\_pred\_xgb = classifier\_xgb.predict(X\_test)  
cm = confusion\_matrix(y\_test, y\_pred\_xgb)  
print("XGBoost Classifier")  
print(cm)  
  
#SVM  
y\_pred\_svm = classifier\_svm.predict(X\_test)  
cm = confusion\_matrix(y\_test, y\_pred\_svm)  
print("#SVM")  
print(cm)