**Main.py**

import matplotlib

matplotlib.use('TkAgg')

from matplotlib import pyplot as plt

import os

from flask import Flask, send\_file,render\_template, request, redirect, Response

import pymysql

from datetime import date

import json

import os

from string import punctuation

from nltk.corpus import stopwords

import nltk

from nltk.stem import WordNetLemmatizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import accuracy\_score

from sklearn import svm

from sklearn.model\_selection import train\_test\_split

import pandas as pd

import numpy as np

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

import pickle

from datetime import date

import random

from keras.utils.np\_utils import to\_categorical

from keras.layers import MaxPooling2D

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Convolution2D

from keras.models import Sequential

from keras.models import model\_from\_json

import pickle

from keras.layers import LSTM, Bidirectional

import keras

app = Flask(\_\_name\_\_)

app.secret\_key = 'amazon'

app.config['SEND\_FILE\_MAX\_AGE\_DEFAULT'] = 0

global cnn\_classifier, lstm\_classifier

global vectorizer

stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer()

reviews = []

sentiments = []

fake = []

accuracy = []

precision = []

recall = []

fscore = []

def cleanPost(doc):

tokens = doc.split()

table = str.maketrans('', '', punctuation)

tokens = [w.translate(table) for w in tokens]

tokens = [word for word in tokens if word.isalpha()]

tokens = [w for w in tokens if not w in stop\_words]

tokens = [word for word in tokens if len(word) > 1]

tokens = [lemmatizer.lemmatize(token) for token in tokens]

tokens = ' '.join(tokens)

return tokens

@app.route("/Train")

def Train():

global reviews

global sentiments

global fake

global accuracy

global precision

global recall

global fscore

global cnn\_classifier, lstm\_classifier

global vectorizer

reviews.clear()

sentiments.clear()

fake.clear()

accuracy.clear()

precision.clear()

recall.clear()

fscore.clear()

if os.path.exists('model/reviews.txt.npy'):

reviews = np.load("model/reviews.txt.npy")

sentiments = np.load("model/rating.txt.npy")

fake = np.load("model/fake.txt.npy")

else:

dataset = pd.read\_csv("Dataset/Reviews.csv",nrows=10000)

dataset = dataset.values

for i in range(len(dataset)):

numerator = dataset[i,4]

denomerator = dataset[i,5]

rating = dataset[i,6]

text\_review = dataset[i,9]

text\_review = text\_review.strip('\n')

text\_review = text\_review.strip()

reviews.append(cleanPost(text\_review.strip().lower()))

sentiments.append(rating-1)

if numerator >= denomerator:

fake.append(0)

else:

fake.append(1)

reviews = np.asarray(reviews)

sentiments = np.asarray(sentiments)

fake = np.asarray(fake)

np.save("model/reviews.txt",reviews)

np.save("model/rating.txt",sentiments)

np.save("model/fake.txt",fake)

vectorizer = TfidfVectorizer(stop\_words=stop\_words, use\_idf=True, smooth\_idf=False, norm=None, decode\_error='replace', max\_features=500)

tfidf = vectorizer.fit\_transform(reviews).toarray()

df = pd.DataFrame(tfidf, columns=vectorizer.get\_feature\_names())

print(str(df))

print(df.shape)

df = df.values

X = df[:, 0:500]

X = X.reshape((X.shape[0],X.shape[1],1,1))

X1 = X.reshape((X.shape[0],X.shape[1],1,1))

indices = np.arange(X.shape[0])

np.random.shuffle(indices)

X = X[indices]

sentiments = sentiments[indices]

sentiments = to\_categorical(sentiments)

X1 = X1[indices]

fake = fake[indices]

fake = to\_categorical(fake)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, sentiments, test\_size=0.2)

X\_train1, X\_test1, y\_train1, y\_test1 = train\_test\_split(X1, fake, test\_size=0.2)

if os.path.exists('model/cnn.json'):

with open('model/cnn.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

cnn\_classifier = model\_from\_json(loaded\_model\_json)

json\_file.close()

cnn\_classifier.load\_weights("model/cnn\_weights.h5")

cnn\_classifier.\_make\_predict\_function()

else:

cnn\_classifier = Sequential()

#defining convolution CNN layer with 32 filters and giving X as input details

cnn\_classifier.add(Convolution2D(32, 1, 1, input\_shape = (X.shape[1], X.shape[2], X.shape[3]), activation = 'relu'))

cnn\_classifier.add(MaxPooling2D(pool\_size = (1, 1)))

#defining another CNN layer with 32 filters

cnn\_classifier.add(Convolution2D(32, 1, 1, activation = 'relu'))

cnn\_classifier.add(MaxPooling2D(pool\_size = (1, 1)))

#converting dataset into single dimensional from multi dimensional array

cnn\_classifier.add(Flatten())

#defining output layer

cnn\_classifier.add(Dense(output\_dim = 256, activation = 'relu'))

#defining prediction output layet as sentiment values prediction

cnn\_classifier.add(Dense(output\_dim = sentiments.shape[1], activation = 'softmax'))

print(cnn\_classifier.summary())

#compiling CNN model

cnn\_classifier.compile(optimizer = 'adam', loss = 'categorical\_crossentropy', metrics = ['accuracy'])

#now training CNN with X as dataset reviews and sentiments values as Y. Here X are reviews and sentiments are sentiment values

hist = cnn\_classifier.fit(X, sentiments, batch\_size=16, epochs=10, shuffle=True, verbose=2)

cnn\_classifier.save\_weights('model/cnn\_weights.h5')

model\_json = cnn\_classifier.to\_json()

with open("model/cnn.json", "w") as json\_file:

json\_file.write(model\_json)

json\_file.close()

if os.path.exists('model/lstm.json'):

with open('model/lstm.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

lstm\_classifier = model\_from\_json(loaded\_model\_json)

json\_file.close()

lstm\_classifier.load\_weights("model/lstm\_weights.h5")

lstm\_classifier.\_make\_predict\_function()

else:

lstm\_classifier = Sequential()

#defining LSTM layer with 32 filetsr and X reviews as input

lstm\_classifier.add(LSTM(32, return\_sequences=True, input\_shape=(X1.shape[1],1)))

lstm\_classifier.add(Dropout(0.3))

#defining another LSTM layer

lstm\_classifier.add(LSTM(16, return\_sequences=True, input\_shape=(X1.shape[1],1)))

lstm\_classifier.add(Dropout(0.3))

#defining another LSTM layer to further filter dataset

lstm\_classifier.add(LSTM(8,input\_shape=(X1.shape[1],1)))

#Remove or drop irrelevant fetaures or data

lstm\_classifier.add(Dropout(0.3))

#input output values as FAKE

lstm\_classifier.add(Dense(fake.shape[1], activation='softmax'))

#compile LSTM model

lstm\_classifier.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

#now start training LSTM model with X reviews and fake class labels

lstm\_acc = lstm\_classifier.fit(X1, fake, epochs=10, batch\_size=256, validation\_data=[X\_test1, y\_test1])

lstm\_classifier.save\_weights('model/lstm\_weights.h5')

model\_json = lstm\_classifier.to\_json()

with open("model/lstm.json", "w") as json\_file:

json\_file.write(model\_json)

json\_file.close()

predict = cnn\_classifier.predict(X\_test)

predict = np.argmax(predict, axis=1)

y1\_test1 = np.argmax(y\_test, axis=1)

output = '<table border=1 align=center>'

output+='<tr><th><font size=3 color=black>Algorithm Name</font></th>'

output+='<th><font size=3 color=black>Precision</font></th>'

output+='<th><font size=3 color=black>Recall</font></th>'

output+='<th><font size=3 color=black>FSCORE</font></th>'

output+='<th><font size=3 color=black>Accuracy</font></th></tr>'

a = accuracy\_score(y1\_test1,predict)\*100

p = precision\_score(y1\_test1,predict,average='macro') \* 100

r = recall\_score(y1\_test1,predict,average='macro') \* 100

f = f1\_score(y1\_test1,predict,average='macro') \* 100

output+='<tr><td>CNN Algorithm</td><td>'+str(p)+'</td><td>'+str(r)+'</td><td>'+str(f)+'</td><td>'+str(a)+'</td></tr>'

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

predict = lstm\_classifier.predict(X\_test1)

predict = np.argmax(predict, axis=1)

y\_test1 = np.argmax(y\_test1, axis=1)

a = accuracy\_score(y\_test1,predict)\*100

p = precision\_score(y\_test1,predict,average='macro') \* 100

r = recall\_score(y\_test1,predict,average='macro') \* 100

f = f1\_score(y\_test1,predict,average='macro') \* 100

output+='<tr><td>LSTM Algorithm</td><td>'+str(p)+'</td><td>'+str(r)+'</td><td>'+str(f)+'</td><td>'+str(a)+'</td></tr>'

accuracy.append(a)

precision.append(p)

recall.append(r)

fscore.append(f)

df = pd.DataFrame([['CNN','Accuracy',accuracy[0]],['CNN','Precision',precision[0]],['CNN','Recall',recall[0]],['CNN','FScore',fscore[0]],

['LSTM','Accuracy',accuracy[1]],['LSTM','Precision',precision[1]],['LSTM','Recall',recall[1]],['LSTM','FScore',fscore[1]],

],columns=['Parameters','Algorithms','Value'])

df.pivot("Parameters", "Algorithms", "Value").plot(kind='bar')

#plt.show()

plt.savefig("static/result/cls.png")

#plt.close()

output+="</table>"

output+="<br/>"

output+='<img src="static/result/cls.png" height="600" width="600"/>'

output+='<br/><br/><br/><br/><br/><br/>'

return render\_template("Train.html",output=output)

@app.route('/ViewClassification')

def ViewClassification():

output = '<table border=1 align=center>'

output+='<tr><th><font size=3 color=black>Reviewer Name</font></th>'

output+='<th><font size=3 color=black>Review Text</font></th>'

output+='<th><font size=3 color=black>Predicted Ratings</font></th>'

output+='<th><font size=3 color=black>Predicted Sentiment</font></th>'

output+='<th><font size=3 color=black>Review Date</font></th></tr>'

con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'amazonreviews',charset='utf8')

pos = 0

neg = 0

neu = 0

myresult = "none"

with con:

cur = con.cursor()

cur.execute("select \* from submit\_reviews")

rows = cur.fetchall()

for row in rows:

if row[2] == 4 or row[2] == 5:

pos = pos + 1

myresult = "Positive"

if row[2] == 3:

neu = neu + 1

myresult = "Neutral"

if row[2] == 1 or row[2] == 2:

neg = neg + 1

myresult = "Negative"

output+='<tr><td>'+row[0]+'</td><td>'+str(row[1])+'</td><td>'+str(row[2])+'</td><td>'+myresult+'</td><td>'+str(row[3])+'</td></tr>'

height = [pos,neg,neu]

bars = ('Positive', 'Negative','Neutral')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.xticks(y\_pos, bars)

plt.title("Reviews Predicted Ratings Graph")

plt.savefig("static/result/reviews.png")

#plt.close()

output+="</table>"

output+="<br/>"

output+='<img src="http://localhost:9999/static/result/reviews.png?cache="'+str(random.randint(10,10000))+' height="600" width="600"/>'

output+='<br/><br/><br/><br/><br/><br/>'

return render\_template("ViewClassification.html",output=output)

@app.after\_request

def add\_header(r):

"""

Add headers to both force latest IE rendering engine or Chrome Frame,

and also to cache the rendered page for 10 minutes.

"""

r.headers["Cache-Control"] = "no-cache, no-store, must-revalidate"

r.headers["Pragma"] = "no-cache"

r.headers["Expires"] = "0"

r.headers['Cache-Control'] = 'public, max-age=0'

return r

@app.route('/SubmitReviewAction', methods =['GET', 'POST'])

def SubmitReviewAction():

if request.method == 'POST':

global cnn\_classifier, lstm\_classifier

global vectorizer

if os.path.exists('model/cnn.json'):

with open('model/cnn.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

cnn\_classifier = model\_from\_json(loaded\_model\_json)

json\_file.close()

cnn\_classifier.load\_weights("model/cnn\_weights.h5")

cnn\_classifier.\_make\_predict\_function()

if os.path.exists('model/lstm.json'):

with open('model/lstm.json', "r") as json\_file:

loaded\_model\_json = json\_file.read()

lstm\_classifier = model\_from\_json(loaded\_model\_json)

json\_file.close()

lstm\_classifier.load\_weights("model/lstm\_weights.h5")

lstm\_classifier.\_make\_predict\_function()

name = request.form['t1']

review = request.form['t2']

result1 = 'none'

result2 = 'none'

data = cleanPost(review.strip().lower())

temp = []

temp.append(data)

temp = np.asarray(temp)

temp = vectorizer.transform(temp).toarray()

temp = temp.reshape((temp.shape[0],temp.shape[1],1,1))

predict1 = cnn\_classifier.predict(temp)

predict1 = np.argmax(predict1) + 1

if predict1 == 1 or predict1 == 2:

result1 = 'Negative';

if predict1 == 3:

result1 = 'Neutral';

if predict1 == 4 or predict1 == 5:

result1 = "Positive"

predict2 = lstm\_classifier.predict(temp)

predict2 = np.argmax(predict2)

if predict2 == 0:

result2 = "Genuine";

if predict2 == 1:

result2 = "Fake";

today = date.today()

status = 'Error in submitting your review'

db\_connection = pymysql.connect(host='127.0.0.1',port = 3308,user = 'root', password = 'root', database = 'amazonreviews',charset='utf8')

db\_cursor = db\_connection.cursor()

query = "INSERT INTO submit\_reviews(reviewer\_name,review\_text,ratings,review\_date) VALUES('"+name+"','"+review+"','"+str(predict1)+"','"+str(today)+"')"

db\_cursor.execute(query)

db\_connection.commit()

print(db\_cursor.rowcount, "Record Inserted")

if db\_cursor.rowcount == 1:

status = "Hybrid Fuzzy Prediction using CNN and LSTM<br/>Your review sentiments predicted as : "+result1+"<br/> Review predicted as : "+result2

return render\_template("SubmitReview.html",error=status)

@app.route("/SubmitReview")

def SubmitReview():

return render\_template("SubmitReview.html")

@app.route("/Logout")

def Logout():

return render\_template("Logout.html")

@app.route("/index")

def index():

return render\_template("index.html")

@app.route("/Login")

def Login():

return render\_template("Login.html")

@app.route('/LoginAction', methods =['GET', 'POST'])

def LoginAction():

if request.method == 'POST':

global uid

username = request.form['t1']

password = request.form['t2']

status = 'none'

con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'amazonreviews',charset='utf8')

with con:

cur = con.cursor()

cur.execute("select username FROM register where username='"+username+"' and password='"+password+"'")

rows = cur.fetchall()

for row in rows:

if row[0] == username:

uid = username

status = 'success'

break

if status == 'success':

return render\_template("RetailerScreen.html",error='Welcome '+username)

else:

return render\_template("Login.html",error='Invalid Login')

@app.route("/Signup")

def Signup():

return render\_template("Signup.html")

@app.route('/SignupAction', methods =['GET', 'POST'])

def SignupAction():

if request.method == 'POST':

name = request.form['t1']

gender = request.form['t2']

contact = request.form['t3']

address = request.form['t4']

email = request.form['t5']

username = request.form['t6']

password = request.form['t7']

status = 'none'

con = pymysql.connect(host='127.0.0.1',port = 3306,user = 'root', password = 'root', database = 'amazonreviews',charset='utf8')

with con:

cur = con.cursor()

cur.execute("select \* FROM register")

rows = cur.fetchall()

for row in rows:

if row[6] == username:

status = 'Given username already exists'

break

if status == 'none':

db\_connection = pymysql.connect(host='127.0.0.1',port = 3308,user = 'root', password = 'root', database = 'amazonreviews',charset='utf8')

db\_cursor = db\_connection.cursor()

query = "INSERT INTO register(retailer\_name,gender,contact\_no,address,email,username,password) VALUES('"+name+"','"+gender+"','"+contact+"','"+address+"','"+email+"','"+username+"','"+password+"')"

db\_cursor.execute(query)

db\_connection.commit()

print(db\_cursor.rowcount, "Record Inserted")

if db\_cursor.rowcount == 1:

status = "New Retailer Signup process completed"

return render\_template("Signup.html",error=status)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='127.0.0.1', port=9999, debug=True)