from django.shortcuts import render from django.template import RequestContext from django.contrib import messages

from django.http import HttpResponse from django.core.files.storage import FileSystemStorage import os

import pandas as pd import numpy as np

from string import punctuation from nltk.corpus import stopwords import nltk from nltk.stem import

WordNetLemmatizer import pickle from nltk.stem import PorterStemmer import os

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

from sklearn.feature\_extraction.text import TfidfVectorizer #loading tfidf vector from sklearn.metrics import accuracy\_score

from sklearn.metrics import f1\_score from sklearn.metrics import precision\_score from sklearn.metrics import recall\_score import matplotlib.pyplot as plt

from sklearn.naive\_bayes import

GaussianNB from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression import matplotlib.pyplot as plt

from keras.utils import to\_categorical

from keras.layers import MaxPooling2D

from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D, BatchNormalization,

AveragePooling2D from keras.layers import Convolution2D

from keras.models import Sequential, load\_model,

Model import pickle

from keras.callbacks import ModelCheckpoint import io import base64

from sklearn.ensemble import GradientBoostingClassifier

global username

#define object to remove stop words and other text processing stop\_words = set(stopwords.words('english'))

lemmatizer = WordNetLemmatizer() ps = PorterStemmer()

global accuracy, precision, recall, fscore, X, Y, sc, tfidf\_vectorizer accuracy = []

#define function to clean text by removing stop words and other special symbols def cleanText(doc):

tokens = doc.split()

table = s t r. maketrans('', ' ' , p u n c t u a t i o n ) t o k e n s = [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 = [ps.stem(token) for token in tokens]

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

return tokens

if os.path.exists("model/X.npy"): f = o p e n ( ' m o d e l /

t f i d f . p c k l ' , ' r b ' ) t f idf\_vectorizer = pickle.load(f) f.close() np.load("model/

X . n p y " ) Y = np.load("model/ Y.npy")

else:

dataset = pd.read\_csv("Dataset/Tweepfake.csv", sep=";") dataset = dataset.dropna() print(np.unique(dataset['account.type'], return\_counts=True)) print(dataset)

dataset = dataset.values

X = []

Y = []

for i in

range(len(dataset)

): tweet = dataset[i,1] tweet =

tweet.strip("\n").strip().lower(

) label = dataset[i,2]

tweet = cleanText(tweet)#clean description X.append(tweet)

if label els:

Y.append(0)

print(str(i)+" "+str(len(tweet))+" "+str(label)) X = np.asarray(X)

Y = np.asarray(Y)

tfidf\_vectorizer = TfidfVectorizer(stop\_words=stop\_words, use\_idf=True, smooth\_idf=False,

norm=None, decode\_error='replace', max\_features=900) X = tfidf\_vectorizer.fit\_transform(X).toarray() f = open('model/tfidf.pckl', 'wb')

pickle.dump(tfidf\_vectorizer, f) f.close()

np.save("mod el/X", X) np.save("mod el/Y", Y)

sc = StandardScale r() X =

sc.fit\_transfor m(X)

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=0.2) #split dataset into train and test

def calculateMetrics(algorithm, predict, y\_test): label = ['Normal Event', 'Disaster Event']

a = accuracy\_score(y\_test,predict)\*100 p = precision\_score(y\_test, predict,average='macro') \* 100 r =

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

f = f1\_score(y\_test, predict,average='macro')

\* 100 accuracy.append(a) precision.a

ppend(p) recall.appe nd(r) fscore.app end(f)

nb\_cls = GaussianNB() nb\_cls.fit(X\_train[0:1000], y\_train[0:1000]) predict = nb\_cls.predict(X\_test[0:200])

calculateMetrics("Naive Bayes", predict, y\_test[0:200])

lr\_cls = LogisticRegression(max\_iter=300) #create Logistic Regression object lr\_cls.fit(X\_train[0:1000], y\_train[0:1000])

predict = lr\_cls.predict(X\_test[0:200])

calculateMetrics("Logistic Regression", predict, y\_test[0:200]) dt\_cls = DecisionTreeClassifier() # c r e a t e D e c i s i o n T r e e o b j e c t dt\_cls.fit(X\_train[0:1000], y\_train[0:1000])

predict =

d t \_ c l s . p r e d i c t ( X \_ t e s t [ 0 : 2 0 0 ] ) calculateMetrics("Decision Tree", predict, y\_test[0:200])

rf\_cls = RandomForestClassifier() #create Random Forest object rf\_cls.fit(X\_train[0:1000], y\_train[0:1000])

predict = rf\_cls.predict(X\_test[0:200]) calculateMetrics("Random Forest", predict, y\_test[0:200])

gb\_cls = GradientBoostingClassifier() #create XGBOost object gb\_cls.fit(X\_train[0:1000], y\_train[0:1000])

predict = gb\_cls.predict(X\_test[0:200]) calculateMetrics("Gradient Boosting", predict,

y\_test[0:200])

X\_train1 = np.reshape(X\_train, (X\_train.shape[0], 30, 10, 3)) X\_test1 = np.reshape(X\_test, (X\_test.shape[0], 30,

10, 3)) y\_train1 = to\_categorical(y\_train) y\_test1 = to\_categorical(y\_test)

cnn\_model = Sequential()

cnn\_model.add(Convolution2D(32, (3 , 3), input\_shape = (X\_train1.shape[1], X\_train1.shape[2],

X\_train1.shape[3]), activation = 'relu')) cnn\_model.add(MaxPooling2D(pool\_size = (2, 2)))

cnn\_model.add(Convolution2D(32, (3, 3), activation = 'relu'))

cnn\_model.add(MaxPooling2D(pool\_size = (2,

2))) cnn\_model.add(Flatten()) cnn\_model.add(Dense(units = 256, activation = 'relu'))

cnn\_model.add(Dense(units = y\_train1.shape[1], activation = 'softmax'))

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

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

if os.path.exists("model/cnn\_weights.hdf5") == False:

model\_check\_point = ModelCheckpoint(filepath='model/cnn\_weights.hdf5', verbose = 1, save\_best\_only = True)

hist = cnn\_model.fit(X\_train1, y\_train1, batch\_size = 8, epochs = 50, validation\_data=(X\_test1, y\_test1), callbacks=[model\_check\_point], verbose=1)

f = open('model/cnn\_history.pckl', 'wb') pickle.dump(hist.history, f)

cnn\_model.load\_weights("model/cnn\_weights.hdf5")

p r e d i c t =

cnn\_model.predict(X\_test1)

predict = np.argmax(predict, a x i s = 1 ) y \_ t e s t 1 = np.argmax(y\_test1, axis=1)

calculateMetrics("CNN Algorithm", predict, y\_test1)

hybrid\_model = Model(cnn\_model.inputs, cnn\_model.layers[-2].output)#create mobilenet model hybrid\_features = hybrid\_model.predict(X\_test1) #extracting mobilenet features print(hybrid\_features.shape)

Y = y\_test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(hybrid\_features, Y, test\_size=0.2)

X\_train, X\_test1, y\_train, y\_test1 = train\_test\_split(hybrid\_features, Y, test\_size=0.1) rf = RandomForestClassifier()#create random forest object rf.fit(X\_train, y\_train)#train on mobileenet features

predict = rf.predict(X\_test)#perfrom prediction on test data calculateMetrics("Extension Hybrid CNN", predict, y\_test)#call function to calculate accuracy and other metrics

def LoadDataset(request):

Graph") #plt.show() plt.tight\_l ayout()

buf = io.BytesI O()

plt.savefig(buf, format='png', bbox\_inches='tight') plt.close() img\_b64 =

base64.b64encode(buf.getvalue()).decode()

context= {'data':output, 'img': img\_b64}

return render(request, 'ViewGraph.html', context)

def

DetectFakeAction(r equest): if request.method == 'POST':

global username, cnn\_model,

tfidf\_vectorizer, sc tweet = request.POST.get('t1', False) data = tweet.strip().lower()

data = cleanText(da ta) temp = [] temp.append (data)

temp = tfidf\_vectorizer.transform(temp).toarray() dl\_model = load\_model("model/ cnn\_weights.hdf5") temp = sc.transform(temp)

temp = np.reshape(temp, (temp.shape[0], 30, 10, 3)) predict = dl\_model.predict(temp) predict =

np.argmax(predict) print(predict) output =

"Normal " if predict

== 1:

output = "Bot Fake"

context= {'data': 'Given Tweet Detected as : '+output} return render(request, 'DetectFake.html', context)

def DetectFake(request):

if request.method == 'GET':

return render(request, 'DetectFake.html', {})

def index(request):

if request.method == 'GET':

return render(request, 'index.html', {})

def UserLogin(request):

if request.method == 'GET':

return render(request, 'UserLogin.html', {})

def Register(request):

if request.method == 'GET':

return render(request, 'Register.html', {})

defUserLoginAction(r e q u e s t ) : i f request.method == 'POST':

global username

u s e r n a m e =

request.POST.get('t1', False) p a s s w o r d =

request.POST.get('t2', False) status = "UserLogin.html"

context= {'data':'Invalid login details'}

if "admin" == username and "admin" == password:

context = {'data':"Welcome "+username} status = 'UserScreen.html'