```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 22 02:58:20 2021

@author: HP
"""

"""fake news detection"""
import ftfy
import nltk
import re
import pandas as pd
import numpy as np
import warnings
from sklearn.base import BaseEstimator, TransformerMixin
from nltk.corpus import stopwords
from collections import defaultdict
from nltk.corpus import wordnet as wn
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
from tensorflow import keras
import numpy as np

from PIL import Image, ImageChops, ImageEnhance
#nltk.download('stopwords')
warnings.filterwarnings('ignore')
np.random.seed(0)

texttt="A Couple Did A Stunning Photo Shoot With Their Baby After Learning She Had An Inoperable Brain Tumo"
```

```
hashtag_re = re.compile(r"#\w+")
mention_re = re.compile(r"@\w+")
url_re = re.compile(r"(?:https?://)?(?:[-\w]+\.)+[a-zA-Z]{2,9}[-\w/#~:;.?+=&%@~]*")
extras_re = re.compile("[.;:!\'?,\"()\[\]]")
emoji_pattern = re.compile("["
                    u"\U0001F600-\U0001F64F"  # emoticons
                    u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                    u"\U0001F680-\U0001F6FF"  # transport & map symbols
                    u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                    u"\U00002702-\U000027B0"
                    u"\U000024C2-\U0001F251"
                    "]+", flags=re.UNICODE)


""" Preprocessing the text in the statements"""
def preprocess(text):

    p_text = hashtag_re.sub("",text)
    p_text = mention_re.sub("",p_text)
    p_text = extras_re.sub("",p_text)
    p_text = url_re.sub("",p_text)
    p_text = ftfy.fix_text(p_text)
    p_text = emoji_pattern.sub(" ", p_text)
    return p_text.lower()



def Tokenizer(str_input):
    words = re.sub(r"[^A-Za-z0-9\-]", " ", str_input).lower().split()
```

```
    porter_stemmer=nltk.PorterStemmer()
    words = [porter_stemmer.stem(word) for word in words]
    return words



# stop words list set to english
stopwords_list = stopwords.words('english') # stop word list



def print_cv_scores_summary(name, scores):
    print("{}: mean = {:.2f}%, sd = {:.2f}%, min = {:.2f}, max = {:.2f}".format(name, scores.mean()*100,
scores.std()*100, scores.min()*100, scores.max()*100))



data=[['nil','nil',texttt]];
df_raw_tfid = pd.DataFrame(data, columns = ['Lemmatised_words', 'TotalWords','text'])



 # creating new column to hold total number of words in the statements and calculating the total
words
df_raw_tfid['TotalWords'] = df_raw_tfid['text'].str.split().str.len()



from sklearn.base import BaseEstimator, TransformerMixin
class TextSelector(BaseEstimator, TransformerMixin):
    def __init__(self, field):
        self.field = field
    def fit(self, X, y=None):
```

```
        return self
    def transform(self, X):
        return X[self.field]
class NumberSelector(BaseEstimator, TransformerMixin):
    def __init__(self, field):
        self.field = field
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[[self.field]]



corpus=[]


for state in df_raw_tfid['text']:


    texts=preprocess(state)
    token=nltk.word_tokenize(texts)
    corpus.append(token)


tag_map = defaultdict(lambda : wn.NOUN)
tag_map['J'] = wn.ADJ
tag_map['V'] = wn.VERB
tag_map['R'] = wn.ADV


for index,entry in enumerate(corpus):
    # looping through the entries and saving in the corpus
    Final_words = []
        # fitting WordNetLemmatizer()
    word_Lemmatized = WordNetLemmatizer()
        # pos_tag will provide the 'tag' i.e if the word is Noun(N) or Verb(V) etc.
```

```
    for word, tag in pos_tag(entry):
        # condition is to check for Stop words and consider only alphabets
        if word not in stopwords.words('english') and word.isalpha():
            word_Final = word_Lemmatized.lemmatize(word,tag_map[tag[0]])
            Final_words.append(word_Final)
        # The processed words for each 'statement' will be store in column 'lemmatised_words in the dataframe'
    df_raw_tfid.loc[index,'Lemmatised_words'] = str(Final_words)




import joblib


loaded_model=joblib.load("F:/ml/spyder/fnd.pkl")


cnn_model_y_proba=loaded_model.predict(df_raw_tfid)
a=loaded_model.predict_proba(df_raw_tfid)







"""Tampered image detection"""


def second():


    def ErrorLevelAnalysis(path, quality):
        filename = path
        resaved_filename = filename.split('.')[0] + '.resaved.jpg'
        #ELA_filename = filename.split('.')[0] + '.ela.png'
```

```
    im = Image.open(filename).convert('RGB')
    im.save(resaved_filename, 'JPEG', quality=quality)
    resaved_im = Image.open(resaved_filename)

    ela_im = ImageChops.difference(im, resaved_im)

    extrema = ela_im.getextrema()
    max_diff = max([ex[1] for ex in extrema])
    if max_diff == 0:
      max_diff = 1
    scale = 255.0 / max_diff

    ela_im = ImageEnhance.Brightness(ela_im).enhance(scale)

    return ela_im

image_name="F:/ml/DESCRIPTION/csa/fke/Sp_D_CND_A_sec0056_sec0015_0282.jpg";


x=[]

x.append(np.array(ErrorLevelAnalysis(image_name,90).resize((192, 192))).flatten() / 255.0)

X = np.array(x)
X = X.reshape(-1, 192, 192, 3)


loaded_model=keras.models.load_model("F:/ml/spyder/my_model.h5")
```

```
    pred=loaded_model.predict(X)



    y_pred_cnn1 = loaded_model.predict(X)
    y_pred_cnn = np.argmax(y_pred_cnn1,axis = 1)
    return y_pred_cnn1


b= second()



true_percentage=((a[0,0]+b[0,0])/2)*100
fake_percentage=((a[0,1]+b[0,1])/2)*100
print("\n")
print("**************************TEXT DATA**************************")

print("\n")
if a[0,1] > a[0,0]:
  print("The given text news is fake with probability of ",("{:.2f}".format(a[0,1]*100)),"%")
else:
  print("The given text news is real with probability of ",("{:.2f}".format(a[0,0]*100)),"%")
print("\n")
print("**************************IMAGE DATA**************************")
print("\n")
if b[0,1] > b[0,0]:
  print("The given image is tampered with probability of ",("{:.2f}".format(b[0,1]*100)),"%")
else:
  print("The given image is real with probability of ",("{:.2f}".format(b[0,0]*100)),"%")
print("\n")
```

```
print("**************************AVERAGE*****************************")
print("\n")
if fake_percentage > true_percentage:
  print("The given news is fake with probability of ",("{:.2f}".format(fake_percentage)),"%")
else:
  print("The given news is real with probability of ",("{:.2f}".format(true_percentage)),"%")
```