```
# -*- coding: utf-8 -*-
"""charlie_graph_creation.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1KCgfmJTYLfCEKdx3neQ_B1e2gfejfdBy
"""

# For colab
#from google.colab import drive
#drive.mount('/content/drive')

import numpy as np
import nltk

# for colab
#pwd

import pandas as pd
df = pd.read_csv(r'F:\Stats\Estonia\New folder\phemernrdataset\pheme-rnr-dataset/dump_charlie

df.tail()

df = df.loc[:,[0,1,5,6,10]]
df.columns = ['tweet_id','src_text','reply_id','reply_text','labels']
y = df['labels']

df.tweet_id.nunique()

tweet_ids = list(df.tweet_id.unique())

reply_ids = list(df.reply_id.unique())

print(len(tweet_ids))
print(len(reply_ids))

total_ids = tweet_ids + reply_ids

len(total_ids)

all_ids = pd.DataFrame(total_ids)

len(all_ids)

def isOriginaltweetSimilarToReplyTweetID():
  count = 0
  for original in df.tweet_id:
    for reply in df.reply_id:
      if original == reply:
        count = count + 1
        print ("match: ", count, " ", original, " ", reply)
      else:
        continue

isOriginaltweetSimilarToReplyTweetID()

import numpy as np
import pandas as pd
s = (38191, 38191)
```

```
a = np.zeros(s, dtype = int)

indices = []
    for i in range (0, 38191)
        indices.append(i)

adjacency_matrix = pd.DataFrame( data = a, columns = total_ids)

adjacency_matrix.head()

import networkx as nx
G = nx.Graph()
G.add_nodes_from(total_ids)

def addEdgesInGraph():
    c = 0
    for original in df.tweet_id.unique():
        i = -1
        c = c+1
        print(c)
        for reply in df.reply_id:
            i = i + 1
            if ( original == df.iloc[i]['tweet_id'] ):
                G.add_edge(original, reply)

addEdgesInGraph()

nx.write_gpickle(G, "twitter_charlie_graph.gpickle")

nx.write_adjlist(G,"twitter_charlie_graph.adjlist")

nx.write_multiline_adjlist(G,"twitter_charlie_graph.adjlist")

nx.write_edgelist(G, "twitter_charlie_graph.edgelist.gz")

nx.write_gexf(G, "twitter_charlie_graph.gexf")

nx.write_graphml_lxml(G, "twitter_charlie_graph.graphml")

nx.write_yaml(G,'twitter_charlie_graph.yaml')

nx.info(G)

nx.density(G)

nx.degree(G)

d = nx.neighbors(G,552784600502915072)

l = nx.all_neighbors(G,552784600502915072)

i = 0
for neighbor in d:
    i = i+1
    print(i)
    print(neighbor)

a = nx.non_neighbors(G,552784600502915072)

i = 0
```

```
for nonneighbor in a:
    i = i+1
    print(i)
    print(nonneighbor)

d = nx.common_neighbors(G,552784600502915072, 552785249420447745)

i = 0
for common in d:
    i = i+1
    print(i)
    print(common)

"""## Sparse matrix"""

A = nx.adjacency_matrix(G)

import scipy.sparse

scipy.sparse.save_npz('adjacency_matrix_sparse_matrix.npz', A)

A.shape

"""## Features"""

l = []
for i in range(0, len(df)):
    l.append(i) #till 36188

l1 = []
for i in range(36189, 38191):
    l1.append(i) #till 36188

len(l1)

modified_data = pd.DataFrame(index = l, columns = ['tweet_ids', 'text', 'label'])

modified_data2 = pd.DataFrame(index = l1, columns = ['tweet_ids', 'text', 'label'])

modified_data = modified_data.append(modified_data2)

modified_data.shape

len(df)

def collatingData():
    j = -1
    for tweet_id in df.tweet_id.unique():
        j = j + 1
        for i in range(0, len(df)):

            if (tweet_id == df.iloc[i]['tweet_id']):
                modified_data.iloc[j]['tweet_ids'] = df.iloc[i]['tweet_id']
                modified_data.iloc[j]['text'] = df.iloc[i]['src_text']
                modified_data.iloc[j]['label'] = df.iloc[i]['labels']
                break
    print(j)

collatingData()
```

```
modified_data.tail()

def collatingReplyWithData():
    j = 2001
    for i in range(0, len(df)):
        j = j + 1
        modified_data.iloc[j]['tweet_ids'] = df.iloc[i]['reply_id']
        modified_data.iloc[j]['text'] = df.iloc[i]['reply_text']
        modified_data.iloc[j]['label'] = df.iloc[i]['labels']
        print(j)

collatingReplyWithData()

modified_data.tail()

"""## Saving combined data into csv file"""

modified_data.to_csv('Combined Data.csv')

modified_data = pd.read_csv('Combined Data.csv')

modified_data1 = modified_data.drop('Unnamed: 0', axis = 1)

modified_data1 = modified_data.drop(['Unnamed: 0.1', 'Unnamed: 0'], axis = 1)

modified_data1

"""## Feature Generate to label reply node - STANCE CLASSIFICATION(Support, Deny)"""

modified_data1['clean'] = modified_data1['clean'].fillna('na')

for i in range(0, len(modified_data1)):
    if (modified_data1.iloc[i]['clean'] == ' '):
        modified_data1.replace( ' ','no text')

modified_data1['clean'] = modified_data1['clean'].apply( lambda x: 'no text' if x == ' ' else

modified_data1.iloc[2049]['clean']

sum(modified_data1['clean'].isna())

modified_data1.to_csv('twitter_processed_data.csv')

import re
# function for cleaning data
def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.sub(i, 'no text', input_txt)
    return input_txt

modified_data['clean'] = np.vectorize(remove_pattern)(modified_data['text'], "@[\w]*")

modified_data['clean'] = modified_data['clean'].str.replace("[^a-zA-Z#]", "no text")

#modified_data['clean'] = modified_data.clean.apply(lambda x: ' '.join([w for w in x.split()

"""## To remove Emoticons if any"""

'''def removeEmoticons(string):
    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)
    return emoji_pattern.sub(r'', string)'''

'''for text in modified_data['clean']:
    print(removeEmoticons(text))'''

"""## Tokenize tweets"""

tweets = []
for tweet in modified_data1['clean']:
    tweets.append(tweet)

"""## Lemmatization"""

len(tweets)
tweets[2049]

import nltk
tokens = []
for t in range(0, len(nested_map)):
    print(t)
    for tweet in nested_map[t]:
        for f in tweet:
            print(f)
            tokens.append(nltk.word_tokenize(f))



len(tokens)

tokens

"""## Jaccard similarity"""

from numpy import argmax
import sklearn

def transformation(token_tweets):
    tokens = [w for s in token_tweets for w in s ]
    print()
    print('All Tokens:')
    print(tokens)

    results = []
    label = sklearn.preprocessing.LabelEncoder()
    onehot = sklearn.preprocessing.OneHotEncoder()

    encoded_all_tokens = label.fit_transform(list(set(tokens)))
    encoded_all_tokens = encoded_all_tokens.reshape(len(encoded_all_tokens), 1)

    onehot.fit(encoded_all_tokens)
    i = 0
    for tweet_tokens in token_tweets:
        i = i+1
        print(i)
        print('Original Input:', tweet_tokens)

        encoded_words = label.transform(tweet_tokens)
        print('Encoded by Label Encoder:', encoded_words)

        encoded_words = onehot.transform(encoded_words.reshape(len(encoded_words), 1))
        print('Encoded by OneHot Encoder:')
        print(encoded_words)

        results.append(np.sum(encoded_words.toarray(), axis=0))

    return results

transformed = transformation(tokens)

tokens_original = []
for i in range(0, len(df['tweet_id'].unique())):
    tokens_original.append(nltk.word_tokenize(modified_data1.iloc[i]['clean']))

transformed_original = []
for i in range(0, len(df['tweet_id'].unique())):
    transformed_original = transformation(tokens_original)

b = []
for v in range(0, len(df)):
    b.append(v)

"""
    Finding the posistion (from lookup table) of word instead of using 1 or 0
    to prevent misleading of the meaning of "common" word
"""
similarity_score = pd.DataFrame(index = v, columns = ['antonyms'])

def calculate_position(values):
    x = []
    for pos, matrix in enumerate(values):
        if matrix > 0:
            x.append(pos)
    return x

"""
    Since scikit-learn can only compare same number of dimension of input.
    Add padding to the shortest sentence.
"""
def padding(sentence1, sentence2):
    x1 = sentence1.copy()
    x2 = sentence2.copy()

    diff = len(x1) - len(x2)

    if diff > 0:
        for i in range(0, diff):
            x2.append(-1)
    elif diff < 0:
        for i in range(0, abs(diff)):
            x1.append(-1)

    return x1, x2
```

```
#y_actual = []
#for i in range(0, df['tweet_id'].unique()):
#    modified_data1.iloc[i]['clean']
#    y_actual.append([calculate_position(transformed_original[i])])

for j in range(0, df['tweet_id'].unique()):
    for i, text in enumerate(tokens):
        y_actual = calculate_position(transformed_original[j])
        y_compare = calculate_position(transformed[i])
        x1, x2 = padding(y_actual, y_compare)
        score = sklearn.metrics.jaccard_similarity_score(x1, x2)
        print('-----')
        print('Score: %.2f, Comparing Sentence: %s' % (score, text))
        similarity_score.iloc[i]['antonyms'] = score

"""## Antonyms"""

from nltk.corpus import wordnet
#antonyms = []
#for syn in wordnet.synsets("increase"):
 #   for lm in syn.lemmas():
  #      if lm.antonyms():
   #         antonyms.append(lm.antonyms()[0].name()) #adding into antonyms
#print(set(antonyms))

i = -1
antonyms = []
for source in df['src_text']:
    i = i + 1
    j = -1
    for reply in df['reply_text']:
        j = j + 1
        tokens_tweets[i]
for syn in wordnet.synsets("increase"):
    for lm in syn.lemmas():
        if lm.antonyms():
            antonyms.append(lm.antonyms()[0].name()) #adding into antonyms

source = tokens[0:2002]
len(source)

ids = df['tweet_id'].unique()
mapping_sentences = []
i = 0
for j in range(0, 2):
    if ids[i] == modified_data1.iloc[j]['tweet_ids']:
        reply_ids.append(df.iloc[j]['reply_id'])
        i = i + 1
        mapping_sentences.append([modified_data1.iloc[j]['clean']])

df.tweet_id.unique()[0:2]

def mapSentencePair():
    nested_map = []
    for original in df.tweet_id.unique()[0:2]:
        mapping_sentences = []
        i = -1
        for reply in df.reply_id:
            i = i + 1
            if ( original == df.iloc[i]['tweet_id'] ):
                reply_id = df.iloc[i]['reply_id']
                for j in range(0, len(modified_data1)):
                    if reply_id == modified_data1.iloc[j]['tweet_ids']:
                        mapping_sentences.append(modified_data1.iloc[j]['clean'])

        nested_map.append([mapping_sentences])

mapSentencePair()

nested_map[1]

mapping_sentences

"""## Antonym function"""

all_antonyms = []
def findAntonymsOfSourceText():
    for i in range(0, len(source)):
        antonyms_list = []
        for token in source[i]:
            for syn in wordnet.synsets(token):
                for lm in syn.lemmas():
                    if lm.antonyms():
                        antonyms_list.append(lm.antonyms()[0].name()) #adding into antonyms
        all_antonyms.append([antonyms_list])

findAntonymsOfSourceText()

for src_id in df['tweet_id'].unique():
    for original_text in nested_map:
        ans = check(all_antonyms[i], original_text)

def check(string, sub_str):
    if (string.find(sub_str) == -1):
        return 0
    else:
        return 1
```