task2_31901611

January 28, 2021

1 FIT5196 Task 2 in Assessment 1

Student Name: Prashasti Garg

Student ID: 31901611 Date: 23/01/2021

Version: 1.0

Environment: Python 3.7.9 and Jupyter notebook

Libraries used: please include the main libraries you used in your assignment here, e.g.,: * pandas (for reading the excel file and sheets) * langid (for classifying the text language) * re (for regular expression) * nltk (for exploring features of raw data)

1.0.1 Imported libraries

- In this task, an excel file is given which consists of 26 sheets. These sheets include the id, date and text of tweets related to covid-19.
- In order to extract the data from excel file, pandas is used.
- The regex library is used to search a pttern in the texts of the sheets in excel file.
- the texts are then checked for english language using langid library.
- NLTK is used to work with human language data.

[368]:
```
import pandas as pd
import re
import langid
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.stem import PorterStemmer
from nltk.probability import *
from nltk.collocations import *
from nltk.util import ngrams
from sklearn.feature_extraction.text import CountVectorizer
```

1.0.2 Reading the Excel file

- The data is extracted from excel file using pd.ExcelFile.

[217]:
```
excel_data = pd.ExcelFile(r'D:/Jupyter Notebook/Wrangling/Dataset/tweet_dataset.
↪xlsx', engine='openpyxl',)
```

1.0.3 Names of the sheets in excel file is stored

- The provided sheets in excel file are segregated according to the date of tweets.

[218]:
```
# name of the sheets are extracted in a variable
sheet_names = excel_data.sheet_names
```

1.0.4 Text from all the sheets is stored in a dictionary

- The excel file is parsed, with NaN columns removed using dropna().

[219]:
```
# an empty dictionary is created, where dates/sheet name is the key and the␣
 ↪text in these respective sheets is the value
text_dict = {}
for i in sheet_names:
    sheet = excel_data.parse(i)
    sheet = sheet.dropna(how='all', axis=1)
    # removes all the columns with NaN
    sheet = sheet[sheet.columns[2]].dropna()
    t_list = sheet.values.tolist()
    text_dict[i] = t_list
```

1.0.5 Text is checked for english language via langid library

- The text collected from all the sheets are then checked for english language.

[220]:
```
# a new dictionary is created to collect all the texts which are in english␣
 ↪language
en_dict = {}
for date, text in text_dict.items():
    file = []
    for tweet in text:
        if langid.classify(tweet)[0] == 'en':
            file.append(tweet)
    en_dict[date] = file
```

1.0.6 Tokens are created from the text in sheets of excel file

- Python breaks each logical line into a sequence of elementary lexical components known as tokens. Each token corresponds to a substring of the logical line. The normal token types are identifiers, keywords, operators, delimiters, and literals, as covered in the following sections. (https://www.oreilly.com/library/view/python-in-a/0596100469/ch04s01.html#:~:text=Python%20breaks%20each%20logical%20line,covered%20in%20the%2
- A regex, r“[a-zA-Z]+(?:[-’][a-zA-Z]+)?” is used to search for the tokens with this pattern.

[221]:
```
# a dictionary is formed to collect all the tokens in the text.
tokens_dict = {}
```

```
# en_dict is iterated to create the tokens which are only in english language
for date, text in en_dict.items():
    tokens_list=[]
    for j in text:
        tokenizer = RegexpTokenizer(r"[a-zA-Z]+(?:[-'][a-zA-Z]+)?")
        tokens = tokenizer.tokenize(j)
        tokens_list += tokens
    tokens_dict[date] = tokens_list
```

1.0.7 Tokens are Normalised ie, converted to lower case

- All the collected tokens are then converted into lower case to enable the collecting of words easier.

[222]:
```
lower_tokens_dict = {}
for date, text in tokens_dict.items():
    lower_tokens_dict[date] = [tokens.lower() for tokens in text]
```

1.0.8 Created list for stop words

- Stop words are the words which occur very frequently in the text, which will create not much difference in the meaning of the text.

[223]:
```
# an empty list is created to store all the stop words which are extracted from␣
 ↪the text file provided
stop_words = []
# the provided text file of stop words is opened
text_file = open(r"D:/Jupyter Notebook/Wrangling/Dataset/stopwords_en.
 ↪txt",encoding="utf8")
# each word is iterated in the text_file
for word in text_file:
    words = word.strip('\n')
    stop_words.append(words)
#xtext file is closed
text_file.close()
```

1.0.9 Context Independent stop words removed

- All the stop words which are not bounded, are removed from the lower_tokens_dict

[224]:
```
# an independent tokens dictonary is created
ind_tokens_dict = {}
# date, text are iterared in lower_tokens_dict, where stop are not yet removed
for date, text in lower_tokens_dict.items():
    ind_tokens = []
    for i in text:
        if i not in stop_words:
            ind_tokens.append(i)
```

```
    ind_tokens_dict[date] = ind_tokens
```

1.0.10 Context Dependent stop words removed

- All the tokens with the threshold more than 24 days and the rare tokens with the threshold less than 2 days also the words whose length is less than 3 are all removed from the independent tokens list ie. ind_tokens_dict.

[225]:
```
# a set is created to collect the unique tokens from each date
set_text = []
for text in ind_tokens_dict.values():
    set_text += list(set(text))
```

[226]:
```
# the list of set, whose frequency distribution is done, then converted into a␣
↪dictionary
temp = dict(FreqDist(set_text))
```

[227]:
```
# a list is created to append all the conditions of dependent tokens
vals = []
for k, v in temp.items():
    if v < 2 or v > 24 or len(k) < 3:
        vals.append(k)
```

[ ]:
```
# a list is created to gather all the dependent tokens
collected_tokens = []
for text in ind_tokens_dict.values():
    # the tokens which fulfill the conditions are then removed from␣
↪collected_tokens
    if text not in vals:
        collected_tokens.append(text)
```

1.0.11 First 200 Meaningful Bigram using PMI

- Pointwise Mutual Information is used to collect all the 200 bigram tokens.
- Collocations are expressions of multiple words which commonly co-occur.(https://www.nltk.org/howto/collocations.html)

[260]:
```
tokens = []
for text in lower_tokens_dict.values():
    tokens = tokens + text
```

[261]:
```
bigram_measures = nltk.collocations.BigramAssocMeasures()
bigram_finder = nltk.collocations.BigramCollocationFinder.from_words(tokens)
bigram_200 = bigram_finder.nbest(bigram_measures.pmi, 200)
```

[262]:
```
bigram_200_list = []
for _0, _1 in bigram_200:
```

```
    res = _0 + "_" + _1
    bigram_200_list.append(res)
```

[299]:
```
bi_flat = [j for i in bigram_200 for j in i]
```

1.0.12 Porter Stemming of tokens

- Porter Stemming is done where the stems are removed from the tokens.

[287]:
```
stemmer = PorterStemmer()
stemmer_list = []
for i in collected_tokens:
    stem = ['{0}'.format(w, stemmer.stem(w)) for w in i]
    stemmer_list.append(stem)
```

1.0.13 Created a stemmed list

- The tokens are collected which have been stemmed as well as bigrams which are created using PMI.

[305]:
```
vocab_list = [j for i in stemmer_list for j in i if j not in bi_flat] +␣
↪bigram_200_list
```

[308]:
```
# the vocab_set is sorted
vocab_set = list(set(vocab_list))
vocab_set.sort()
```

[309]:
```
# index of each token which are sorted is found using enumerate
token_index = list(enumerate(vocab_set))
```

[311]:
```
# a final string is created where all the tokens are concatenated
final_vocab = ""
for i, w in token_index:
    final_vocab += "{}:{}\n".format(w, i)
```

[312]:
```
# a function is created to create the file for each tak in required format
def save_file(file_name, data):
    fout = open(file_name, 'w')
    fout.write(data)
    fout.close()
```

[313]:
```
# a .txt file is created to store all the data from final_vocab
save_file("./31901611_vocab.txt", final_vocab)
```

1.0.14 Top 100 Unigrams

- N-grams of texts are extensively used in text mining and natural language processing tasks. An n-gram is a contiguous sequence of n items from a given sample of text or speech. an

n-gram of size 1 is referred to as a “unigram”.

[316]:
```
unigram = {}
for i in range(len(stemmer_list)):
    uni_freq = list(dict(FreqDist(stemmer_list[i])).items())
    uni_freq.sort(key = lambda x: x[1], reverse = True)
    unigram[sheet_names[i]] =  uni_freq[:100]
```

[323]:
```
# a .txt file is created to store all the data from unigram dictionary which is␣
 ↪converted to string
save_file("./31901611_100uni.txt", str(unigram))
```

1.0.15 Top 100 Bigrams

- N-grams of texts are extensively used in text mining and natural language processing tasks. An n-gram is a contiguous sequence of n items from a given sample of text or speech. an n-gram of size 2 is a “bigram”.

[361]:
```
bigram = {}
for date, text in lower_tokens_dict.items():
    bi = ngrams(text, 2)
    bi_freq = list(dict(FreqDist(bi)).items())
    bi_freq.sort(key = lambda x: x[1], reverse = True)
    bigram[date] =  bi_freq[:100]
```

[363]:
```
# # a .txt file is created to store all the data from bigram dictionary which␣
 ↪is later converted to string
save_file("./31901611_100bi.txt", str(bigram))
```

1.0.16 Getting Count Vectors

- Create a vector that has as many dimensions as your corpora has unique words. Each unique word has a unique dimension and will be represented by a 1 in that dimension with 0s everywhere else.(https://towardsdatascience.com/introduction-to-word-embeddings-4cf857b12edc)

[417]:
```
vectorizer = CountVectorizer(analyzer = "word")
joined_data = [' '.join(text) for text in stemmer_list]
vectorizer.fit(joined_data)
```

[395]:
```
freq_dict = {}
for i in range(len(stemmer_list)):
    fd = dict(FreqDist(stemmer_list[i]))
    freq_dict[sheet_names[i]] = fd
```

[413]:
```
output = ""
for date, text in freq_dict.items():
    output += "{},".format(date)
```

```
    for k, v in text.items():
        if k in vectorizer.vocabulary_.keys():
            ix = vectorizer.vocabulary_[k]
            output += "{}:{},".format(ix, v)
```

[416]:
```
# a .txt file is created to store all the data from output which collectes the␣
↪vectors
save_file("./31901611_countVec.txt", output)
```