task2_31282016

September 21, 2020

1 FIT5196 Task 2 in Assessment 1

Student Name: Nabilah Anuwar

Student ID: 31282016 Date: 20/09/2020

Environment: Python 3 and Jupyter notebook

Libraries used: please include the main libraries you used in your assignment here, e.g.,: * nltk (for making bigrams, unigrams, tokenization, and Porter Stemmer) * langid (for language detection) * xlrd (for importing excel) * sklearn (for countvetorizer) * warnings (for ignoring FutureWarnings since sklearn keep putting it)

1.1 1. Import libraries

```
[ ]: import langid
     import nltk
     import xlrd
     from nltk.stem.porter import *
     from nltk.collocations import *
     from sklearn.feature_extraction.text import CountVectorizer
     # import warnings filter
     from warnings import simplefilter
     # ignore all future warnings
     simplefilter(action='ignore', category=FutureWarning)
```

1.2 2. Coding

1.2.1 Set Variables

First we will set variables for use later

```
[ ]: # make variables for later
     ps = PorterStemmer()
     pattern = "[a-zA-Z]+(?:[-'][a-zA-Z]+)?"
     bigram_measures = nltk.collocations.BigramAssocMeasures()

     # get stopwords as a list
     with open("part2/stopwords_en.txt") as f:
             content = f.readlines()
             stopwords = [x.strip() for x in content]
```

Then we will set the function for opening the file

```
[ ]: def open_file(file):
             try:
                     # set up sheets dictionary to store values per sheet
                     data = {}
                     sheets = {}
                     ids = []
                     texts = []
                     # open workbook using xlrd
                     workbook = xlrd.open_workbook(file)
                     # get sheet as an object
                     for sheet in workbook.sheets():
                             sheets[sheet.name] = {"sheet": workbook.
      ↪sheet_by_name(sheet.name), "rows": []}
                     # go through sheets dictionary
                     for name in sheets.keys():
                             sheet = sheets[name]["sheet"]
                             # get row range in sheets for it to loop through to get␣
      ↪values in the row
                             for row in range(sheet.nrows):
                                     sheets[name]["rows"].append(sheet.row(row))
                             # loop through values in the row to differentiate␣
      ↪columns
                             for column in sheets[name]["rows"]:
                                     # get index to get info later
                                     index = sheets[name]["rows"].index(column)
                                     column_matched = False
                                     for item in column:
                                             # get item index
                                             col_index = column.index(item)
                                             # make sure its not empty cell
                                             # wont continue to loop when text is␣
      ↪found
                                             if item.value != "" and not␣
      ↪column_matched:
                                                     column_matched = True
                                                     text = item.value
                                                     # use index to get id and time
                                                     idc = column[col_index + 1].
      ↪value
                                                     timestamp = column[col_index +␣
      ↪2].value
                                                     if text in texts or idc in ids:
                                                             sheets[name]["rows"][index]␣
      ↪= ['empty']
                                                             continue
                                                     else:
                                                             pass
                                                     # append to compare if id or␣
      ↪text is duplicate
                                                     # if duplicate will not be␣
      ↪assigned
                                                     ids.append(idc)
                                                     texts.append(text)
                                                     sheets[name]["rows"][index] =␣
      ↪{"text": text, "id": idc, "timestamp": timestamp}
                             # append to finalize output file
                             data[name] = sheets[name]["rows"]
                     return data
             except Exception as e:
                     return e
```

Since we only need to only get tweets that are english we will sort them with langid

[ ]:
```
def lang_check(data):
        try:
                # for output later
                new_data = {}
                # make sure its a dictionary since this code is design for it
                if type(data) is not dict:
                        return "The array you provided is not a dictionary"
                # get keys in data file
                for key in data.keys():
                        new_data[key] = []
                        rows = data[key]
                        index = 0
                        # iterate through rows per date
                        for row in rows:
                                # remove empty row
                                if type(row) is not dict:
                                        continue
                                # remove titles
                                if row['text'] == 'text' and row['id'] == 'id'␣
↪and row['timestamp'] == 'created_at':
                                        continue
                                text = str(row['text'])
                                lang_test = langid.classify(text)
                                # check if its english
                                if lang_test[0] != 'en':
                                        continue
                                # add row to dictionary
                                new_data[key].append(row)
                                index += 1
                return new_data
        except Exception as e:
                return e
```

To answer Task 2.1 we create these functions

```
[ ]: # bigrams but with PorterStemmer and stopwords
def bigrams_vocab(data):
        # for output later
        # to get all the table tokens
        table_tokens = []
        # iterate through data keys
        for key in data.keys():
                rows = data[key]
                # iterate through each tweets
                for row in rows:
                        try:
                                tokenizer = nltk.RegexpTokenizer(pattern)
                                tokens = tokenizer.tokenize(row["text"])
                                new_tokens = []
                                for word in tokens:
                                        word = word.lower()
                                        # the bigrams doesnt seem to have be a␣
 ↪stem word so we don't use it
                                        # i got the wrong answer when i use␣
 ↪stopwords
                                        # make sure after transform is still in␣
 ↪limit
                                        if len(word) < 3:
                                                continue
                                        else:
                                                new_tokens.append(word)
                                table_tokens = table_tokens + new_tokens
                        except Exception as e:
                                pass
        finder = BigramCollocationFinder.from_words(table_tokens)
        a = finder.nbest(bigram_measures.pmi, 200)
        return a


def update_stopwords(data):
        to_stopword = []
        for key in data.keys():
                rows = data[key]
                sheet_tokens = []
                for row in rows:
                        try:
                                tokenizer = nltk.RegexpTokenizer(pattern)
                                tokens = tokenizer.tokenize(row["text"])
                                new_tokens = []
                                for word in tokens:
                                        word = word.lower()
                                        if word in stopwords:
                                                continue
                                        # stop words will not be stemmed
                                        # make sure transformed word doesn't␣
 ↪decrease len later
                                        elif len(word) < 3:
                                                continue
                                        else:
                                                new_tokens.append(word)
                                tokens_l = [x.lower() for x in new_tokens]
                                sheet_tokens = sheet_tokens + tokens_l
                        except Exception as e:
                                pass
                # make sure there are no repeating values
                a = set(sheet_tokens)
                to_stopword = to_stopword + list(a)
        # get frequency and add to stopwords those that appear less than 5 days
        to_stopword = nltk.FreqDist(to_stopword)
        for key in to_stopword.keys():
                if to_stopword[key] < 5:
                        stopwords.append(key)


def get_vocab(data):
        all_tokens = []
        # to append to list later
        bigrams = []
        # make sure its not repeating in single words
        bigram = []
        bgr = bigrams_vocab(data)
        for bg in bgr:
                # make text for vocab file
                text = bg[0]+"_"+bg[1]
                bigrams.append(text)
                bigram.append(bg[0])
                bigram.append(bg[1])
        update_stopwords(data)
        # get single vocabs
        for key in data.keys():
                rows = data[key]
                sheet_tokens = []
                for row in rows:
                        try:
                                tokenizer = nltk.RegexpTokenizer(pattern)
                                tokens = tokenizer.tokenize(row["text"])
                                new_tokens = []
                                for word in tokens:
                                        word = word.lower()
                                        if word in stopwords:
                                                continue
                                        elif word in bigram:
                                                continue
                                        else:
                                                word = ps.stem(word)
                                        # make sure after transform is still in␣
 ↪limit
                                        # language id is not used since already␣
 ↪since the start
                                        if len(word) < 3:
                                                continue
                                        else:
                                                new_tokens.append(word)
                                tokens_l = [x.lower() for x in new_tokens]
                                sheet_tokens = sheet_tokens + tokens_l
                        except Exception as e:
                                pass
                all_tokens = all_tokens + sheet_tokens
        all_tokens = all_tokens + bigrams
        # make sure all vocabs doesnt repeat
        all_tokens = set(all_tokens)
        vocab = list(all_tokens)
        vocab = sorted(vocab)
        return vocab


# make file for vocab
def make_vocab(data):
        vocab = get_vocab(data)
        f = open("31282016_vocab.txt", "w")
        n = 0
        for word in vocab:
                text = str(word) + ":" + str(n)
                f.write(text)
                f.write('\n')
                n += 1
        f.close()
```

To answer Task 2.2 we create these functions

```
[ ]: # make unigram
def uni_data(data):
        uni_arr = {}
        for key in data.keys():
                rows = data[key]
                sheet_tokens = []
                for row in rows:
                        try:
                                tokenizer = nltk.RegexpTokenizer(pattern)
                                tokens = tokenizer.tokenize(row["text"])
                                new_tokens = []
                                for word in tokens:
                                        word = word.lower()
                                        if not word.isalnum():
                                                continue
                                        elif word in stopwords:
                                                continue
                                        else:
                                                word = ps.stem(word)
                                        # make sure transformed word doesn't␣
 ↪decrease len later
                                        if len(word) < 3:
                                                continue
                                        else:
                                                new_tokens.append(word)
                                tokens_l = [x.lower() for x in new_tokens]
                                sheet_tokens = sheet_tokens + tokens_l
                        except Exception as e:
                                pass
                # get frequency for unigram
                freq = nltk.FreqDist(sheet_tokens)
                uni_arr[key] = freq
        return uni_arr


# make bigram
def bi_data(data):
        # create dictionary to use later
        bi_arr = {}
        # get data keys
        for key in data.keys():
                rows = data[key]
                sheet_tokens = []
                for row in rows:
                        try:
                                # tokenize the relevant words
                                tokenizer = nltk.RegexpTokenizer(pattern)
                                tokens = tokenizer.tokenize(row["text"])
                                new_tokens = [word.lower() for word in tokens␣
 ↪if word.isalnum()]

                                # use function to make bigrams
                                bigram = nltk.bigrams(new_tokens)
                                tokens_l = [x for x in bigram]
                                sheet_tokens = sheet_tokens + tokens_l
                        except Exception as e:
                                pass
                        # get frequency for bigram
                freq = nltk.FreqDist(sheet_tokens)
                bi_arr[key] = freq
        return bi_arr


# make file for unigram
def make_uni(data):
        # get data file
        unigram = uni_data(data)
        # make a file
        f = open("31282016_100uni.txt", "w")
        # iterate through keys
        for key in data.keys():
                # get top 100
                uni_100 = unigram[key].most_common(100)
                # make the line needed
                line = key + ":" + str(uni_100)
                # write line in file and add break for new line
                f.write(line)
                f.write('\n')
        # close file
        f.close()


# make file for bigrams
def make_bi(data):
        # get data file
        bigram = bi_data(data)
        # make file
        f = open("31282016_100bi.txt", "w")
        # iterate through keys
        for key in data.keys():
                # get top 100
                bi_100 = bigram[key].most_common(100)
                # write line in file and break for new line
```

```
            line = key + ":" + str(bi_100)
            f.write(line)
            f.write('\n')
    # close file
    f.close()
```

For Task 2.3 we created these functions

```
[ ]: def countvec(data):
         # update stopwords
         # update_stopwords(data)
         # create dictionary to use later
         freqy = {}
         row_l = {}
         cv_l = []
         for key in data.keys():
                 rows = data[key]
                 sheet_tokens = []
                 # tokenize at first
                 for row in rows:
                         try:
                                 tokenizer = nltk.RegexpTokenizer(pattern)
                                 tokens = tokenizer.tokenize(row["text"])
                                 new_tokens = []
                                 for word in tokens:
                                         word = word.lower()
                                         if not word.isalnum():
                                                 continue
                                         elif word in stopwords:
                                                 continue
                                         else:
                                                 word = ps.stem(word)
                                         # make sure transformed word doesn't␣
      ↪decrease len later
                                         if len(word) < 3:
                                                 continue
                                         else:
                                                 new_tokens.append(word)
                                 tokens_l = [x.lower() for x in new_tokens]
                                 sheet_tokens = sheet_tokens + tokens_l
                         except Exception as e:
                                 pass
                 # get tokenize  words
                 row_l[key] = sheet_tokens
                 # join all the keys to be used in countvectorization
                 row = " ".join(row_l[key])
                 cv_l.append(row)
```

```
                # get count from tokenize works
                freq = nltk.FreqDist(sheet_tokens)
                freqy[key] = dict(freq)
        # we can set min_df to ignore words that appear in less than 5 days
        # but we have the stopwords list already for that
        cv = CountVectorizer()
        cv.fit_transform(cv_l)
        # get index
        dicts = cv.vocabulary_
        for key in data.keys():
                rows = row_l[key]
                count = freqy[key]
                # set list for words to append in result
                words = []
                for row in rows:
                        # get values from the dictionaries using the word as key
                        index = dicts[row]
                        county = count[row]
                        text = "{}:{}".format(index, county)
                        words.append(text)
                        continue
                row_l[key] = words
        return row_l


def make_countvec(data):
        day = countvec(data)
        f = open("31282016_countVec.txt", "w")
        for key in day.keys():
                # write in date
                f.write(key)
                words = day[key]
                # get the values in dictionary
                for word in words:
                        # write accordingly
                        f.write(",")
                        f.write(word)
                f.write('\n')
        f.close()
```

1.3 Run Functions

[ ]:
```
data = open_file("part2/31282016.xlsx")
print("finished open file")
data = lang_check(data)
print("finished language check")
```

[ ]:
```
make_vocab(data)
print("vocabulary list is finished")
```

[ ]:
```
make_bi(data)
make_uni(data)
print("bigrams and unigrams are finished")
make_countvec(data)
print("finished all task, please check output files")
```

1.4 3. Summary

Give a short summary of your work done above, such as your findings.

- It was interesting because I was closer to the answer when bigrams aren't sorted through the stopwords yet unigrams have to be sorted through.
- the regex also sorted through random words such as link but we use langid to tackle that
- sklearn kept having FutureWarnings that i have to ignore it as the code continued even if it appeared
- vocabulary maker is the function that took the longest out of them all as it has to iterate through all the available words