FIT5196 Task 2 in Assignment 1

Name: Peiyu Liu

SID: 31153291

Date: 15/04/2021

Libraries used:

- re: process regular expression
- nltk(FreqDist): calculate words appearance
- nltk(PorterStemmer): extract stem of word
- nltk(ngrams): generate N-words
- nltk(sklearn.CountVectorizer): process word segmentation and matrix
- pandas: read file and reshape contents
- langid: fitler language

1. Introduction

Task 2 provides a tsv document that document contains 1000+ articles. I use the pandas package to read the tsv document and upload all contents in this document. I use pd. concat function to reshape this table and set a new index. I use langid to filter non-English contents. I need to use packages like the "re" package to compile the sample regular expression "\w+(?:[-']\w+)?" and find words that match this regular expression. I use the Porter stemming algorithm in the natural language processing method to extract the stem of words. Then use the FreqDist method to count the number of occurrences of words. To output bigram words, I use the N-grams package to pair word and word. I use the CountVectorizer library of sklearn in the natural language processing method for word segmentation and encode the words in the text and convert them into a sparse matrix.

Details will introduce in each sections.

2. Assignment Coding

Import libraries

- download package: pip install package_name

In [1]:

```
import re

import langid
import nltk
import pandas as pd
from nltk import FreqDist
from nltk import PorterStemmer
from nltk import ngrams
from sklearn.feature_extraction.text import CountVectorizer
```

Read tsv file

- use pandas to read table format
- use pd.concat() function to reshape tables as one table
- ignore_index should be true to ignore previous indexs in each tables

In [2]:

```
tsv_upload = pd.read_table('~/Desktop/5196/assignment 1/task2/31153291.tsv')
```

In [3]:

```
shape_redo = pd.concat([tsv_upload, tsv_upload], ignore_index=True)
```

Pre-processing contents

- use pd.iterrows() function to iterate dataframe, read each line.
- return value is [index, line_each]
- use langid to check all line, if line is English then store line to dic.

reference: iterrows() https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas (https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas)

In [4]:

```
shape_dic = {}
for index, line_each in shape_redo.iterrows():
    num_each = line_each['Unnamed: 0']
    if langid.classify(line_each.text)[0] != 'en':
        continue
    if num_each not in shape_dic.keys():
        shape_dic[num_each] = [line_each.text]
    elif line_each.text not in shape_dic[num_each]:
        shape_dic[num_each] += [line_each.text]
```

Read stop words

- read stop words document
- split each word

In [5]:

```
words_upload = open('stopwords_en.txt', 'r', encoding='utf-8')
stop_words = words_upload.read()
stop_words = stop_words.split('\n')
```

- use re to compile sample regular expression

In [6]:

```
sample_regx = re.compile(r"\w+(?:[-']\w+)?")
```

Filter words using stop words and sample regular expression

- iterator reads key and value in dictionary
- iterator reads each word in each value
- use re.findall() to match each word in lower format with sample regular expression
- store each capable word

In [7]:

```
num_value = {}
token_collection = []
for flag in shape_dic.keys():
    text_values = shape_dic[flag]
    initial_container = []
    for word in text_values:
        regx_token = re.findall(sample_regx, word)
        regx_token = [word.lower() for word in regx_token if word.lower() not in stop_words and len(word)
> 2]
        initial_container += regx_token
        regx_token = list(set(regx_token))
        token_collection += regx_token
    num_value[flag] = initial_container
```

Rule: word appearance 5%-95%

- expression to set maximum rate and minimum rate

In [8]:

```
appearance_word = 0
for flag in shape_dic.keys():
    appearance_word += len(shape_dic[flag])
min_rate = 0.05 * appearance_word
max_rate = 0.95 * appearance_word
```

Natural language processing

use PorterStemmer() function to remove the prefix and suffix of the word to get the root

- happy=>happiness, where happy is called the stem of happiness.
- Reference: PorterStemmer() trainning: https://pythonprogramming.net/stemming-nltk-tutorial/ (https://pythonprogramming.net/stemming-nltk-tutorial/)

use FreqDist to check word appearance

- FreqDist inherits from dict. The key in FreqDist is a word, and the value is the total number of occurrences of the word.
- Reference: FreqDist with python https://stackoverflow.com/questions/4634787/freqdist-with-nltk (https://stackoverflow.com/questions/4634787/freqdist-with-nltk)
- use maximum rate and minimum rate to check words' appearance to remove unavailable words

In [9]:

```
stemmer_import = PorterStemmer()
token_freq = FreqDist(token_collection)
collection_new = []
for flag in token_freq.keys():
    if token_freq[flag] > min_rate and token_freq[flag] < max_rate:
        collection_new.append(stemmer_import.stem(flag))
collection_new = list(set(collection_new))
collection_new = [word for word in collection_new if len(word) > 2 and word not in stop_words]
```

Process unigram word rating

- sorted all processed articles
- Each form of the word counts as the word has appeared, so use stem to extract the stem.
- after use stem() to extract words, then use FreqDist() to calculate times.
- use most_common() function to rate top 100 appearance words.
- Reference: most_common():https://www.geeksforgeeks.org/python-find-most-frequent-element-in-a-list/ (https://www.geeksforgeeks.org/python-find-most-frequent-element-in-a-list/)
- process word and time format like sample document

In [10]:

```
single_words = ''
for flag in sorted(num_value):
    word_token = num_value[flag]
    word_token = [stemmer_import.stem(word) for word in word_token]

    word_feq = FreqDist(word_token)
    feq_record = word_feq.most_common(100)
    single_words += str(flag) + ':' + str(feq_record) + '\n\n'
```

Output top 100 unigram words

In [11]:

```
with open('31153291_100uni.txt', 'w') as output:
    output.write(single_words)
```

Process 200 meaningful bigrams (PMI measure)

- PMI: In text processing, it is used to calculate the degree of association between two words.
- Reference: PMI: https://medium.com/dataseries/understanding-pointwise-mutual-information-in-nlp-e4ef75ecb57a (https://medium.com/dataseries/understanding-pointwise-mutual-information-in-nlp-e4ef75ecb57a)
- use nltk.collocations.BigramAssocMeasures() to get bigram words.
- use nltk.collocations.BigramCollocationFinder.from_words to find bigram words,extract the 2grams of any two words in the window, and realize the non-continuous frequent co-occurring word pairs.
- Reference: collocations:http://www.nltk.org/howto/collocations.html (http://www.nltk.org/howto/collocations.html)
- For binary phrases, the NBEST instance method based on PMI can be used.
- use collocation.nbest() to findg word pairs' frequency is top 200.
- Reference: collocation.nbest():https://stackoverflow.com/questions/21128689/how-to-get-pmi-scores-for-trigrams-with-nltk-collocations-python (https://stackoverflow.com/questions/21128689/how-to-get-pmi-scores-for-trigrams-with-nltk-collocations-python)
- reshape word pair like a_b and their frequencies.

In [12]:

```
vocab_bigram = nltk.collocations.BigramAssocMeasures()
```

In [13]:

```
initial_container = []
for flag in shape_dic.keys():
    text_value = shape_dic[flag]
    for word in text_value:
        initial_container += re.findall(sample_regx, word)
initial_container = [word.lower() for word in initial_container if word.lower() not in stop_words and len(w
ord) > 2]
collocation = nltk.collocations.BigramCollocationFinder.from_words(initial_container)
bigram_collection = collocation.nbest(vocab_bigram.pmi, 200)
bigram_collection = [str(word[0] + '_' + word[1]) for word in bigram_collection]
```

In [14]:

```
vocab_collection = collection_new + bigram_collection
vocab_collection.sort()
vocab_reshape = dict(zip(vocab_collection, [flag for flag in range(len(vocab_collection))]))
vocab_container = ''
for flag, values in vocab_reshape.items():
    vocab_container += flag + ":" + str(vocab_reshape[flag]) + '\n'
```

Output vocab document

In [15]:

```
with open('31153291_vocab.txt', 'w') as output:
    output.write(vocab_container)
```

Process bigram words

- use re.findall() to find words meet sample regular expression.
- compare words with lower format.
- use ngram to reshape word to word pairs(a,b). A sequence of 2 consecutive words in contents.
- Reference: Ngram: https://stackoverflow.com/questions/17531684/n-grams-in-python-four-five-six-grams (https://stackoverflow.com/questions/17531684/n-grams-in-python-four-five-six-grams)
- use FreqDist to check word appearance
- FreqDist inherits from dict. The key in FreqDist is a word, and the value is the total number of occurrences of the word.
- Reference: FreqDist with python https://stackoverflow.com/questions/4634787/freqdist-with-nltk (https://stackoverflow.com/questions/4634787/freqdist-with-nltk)
- use most_common() function to extract top 100 word pairs.
- Reference: most_common(): https://docs.python.org/3/library/collections.html (https://docs.python.org/3/library/collections.html)

In [16]:

```
bigram_container = ''
initial_container = []
for flag in shape_dic.keys():
    text_value = shape_dic[flag]
    for word in text_value:
        initial_container += re.findall(sample_regx, word)
    initial_container = [word.lower() for word in initial_container]
    pair_token = list(ngrams(initial_container, 2))
    pair_token = FreqDist(pair_token)
    pair_token = pair_token.most_common(100)
    bigram_container += str(flag) + ":" + str(pair_token) + '\n'
```

output top 100 bigram words

In [17]:

```
with open('31135291_100bi.txt', 'w') as output:
    output.write(bigram_container)
```

Process sparse representation

- CountVectorizer: It is a text feature extraction method. Only considers the frequency of each vocabulary in the text. CountVectorizer converts the text's words into a word frequency matrix, calculate the number of times each word appears.A model for text vectorization. This model does not consider the grammar and the order of words, but only considers the frequency of occurrence of all words.

Reference: https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

- The CountVectorizer （） class converts the words in the text into a word frequency matrix.
- Calculate the number of occurrences of each word through the fit_transform （） function.
- The keywords of all texts in the bag of words can be obtained through get_feature_names().
- See the result of the word frequency matrix through toarray().
- use dictionary zip()function to gather words and matrix.

Reference: https://scikit-learn.org/stable/modules/feature_extraction.html (https://scikit-learn.org/stable/modules/feature_extraction.html)

https://kavita-ganesan.com/how-to-use-countvectorizer/#.YHgEbkj7TkI (https://kavita-ganesan.com/how-to-use-countvectorizer/#.YHgEbkj7TkI)

- check word length is no less than 3.

In [18]:

```
vectorizer = CountVectorizer(stop_words=stop_words, token_pattern="\w+(?:[-']\w+)?", ngram_range=(1,
2))
```

In [19]:

```
matrix = ''
for flag in shape_dic.keys():
    reshape_dic = ' '.join(shape_dic[flag])
    try:
        matrix_data = vectorizer.fit_transform([reshape_dic])
    except ValueError:
        continue
    vocab_match = vectorizer.get_feature_names()
    matrix_data = matrix_data.toarray()[0]
    vocab_matdic = dict(zip(vocab_match, matrix_data))
    matrix_dic = {}
    for word, count in vocab_matdic.items():
        if count > 0:
            ind_word = word.split()
            if len(ind_word) == 2:
                word_shape = '_'.join(ind_word)
            else:
                word_shape = stemmer_import.stem(word)

            if word_shape in vocab_reshape.keys():
                word_indx = vocab_reshape[word_shape]
                if word_indx in matrix_dic.keys():
                    matrix_dic[word_indx] += vocab_matdic[word]
                else:
                    matrix_dic[word_indx] = vocab_matdic[word]
    regx_token = []
    for flag, value in matrix_dic.items():
        regx_token.append(str(flag) + ":" + str(value))
    matrix += str(flag) + ',' + ','.join(regx_token) + '\n'
```

output count vectorizition document

In [20]:

```
with open('31153291_countVec.txt', 'w', encoding='utf-8') as output:
    output.write(matrix)
```

3. Summary

Task 2 is a complex question for me. Because I have not master python very well at this stage. I have to read and find many materials to solve the facing problems. But luckily, I finished task 2, and I learned a lot of methods to handle a massive amount of texts.

Reference

https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas (https://stackoverflow.com/questions/16476924/how-to-iterate-over-rows-in-a-dataframe-in-pandas)

https://scikit-learn.org/stable/modules/feature_extraction.html (https://scikit-learn.org/stable/modules/feature_extraction.html)

https://kavita-ganesan.com/how-to-use-countvectorizer/#.YHgEbkj7TkI (https://kavita-ganesan.com/how-to-use-countvectorizer/#.YHgEbkj7TkI)

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html)

https://stackoverflow.com/questions/17531684/n-grams-in-python-four-five-six-grams (https://stackoverflow.com/questions/17531684/n-grams-in-python-four-five-six-grams)

https://stackoverflow.com/questions/4634787/freqdist-with-nltk (https://stackoverflow.com/questions/4634787/freqdist-with-nltk)

https://docs.python.org/3/library/collections.html (https://docs.python.org/3/library/collections.html)

https://medium.com/dataseries/understanding-pointwise-mutual-information-in-nlp-e4ef75ecb57a (https://medium.com/dataseries/understanding-pointwise-mutual-information-in-nlp-e4ef75ecb57a)

http://www.nltk.org/howto/collocations.html (http://www.nltk.org/howto/collocations.html)

https://stackoverflow.com/questions/21128689/how-to-get-pmi-scores-for-trigrams-with-nltk-collocations-python (https://stackoverflow.com/questions/21128689/how-to-get-pmi-scores-for-trigrams-with-nltk-collocations-python)