CIS 668 Assignment #1

Analysis of Review Contents

***Instructor**: Professor Lu Xiao*

***Student**: Leah Luo*

***Date**: 02/12/2020*

CIS 668
Leah Luo
Assignment #1
02/12/2020

Data Pre-processing

1. **Retrieve the data needed.**
   a. Extract the text content from the text file. (rawtextSplit)
      The second figure shows the first 20 words in the original file.

```
import nltk
import re
from nltk import FreqDist
from nltk import pos_tag
from nltk.corpus import stopwords
from nltk.corpus import wordnet as wn
from nltk.corpus import PlaintextCorpusReader
from nltk.tokenize import word_tokenize
from collections import defaultdict
from nltk.collocations import *

#Open the original file
path = "/Users/LXIN/Desktop/T/clothing_shoes_jewelry.txt"
textfile = open(path,"r")
rawtext = textfile.read()
textfile.close()
rawtextSplit = rawtext.splitlines()
#Original text content
print(rawtextSplit[:20])
```

```
['reviewerID:A1KLRMWW2FWPL4', 'asin:0000031887', 'reviewerName:Amazon Customer "cameramom"', 'helpful:[0, 0]', "rev
iewText:This is a great tutu and at a really great price. It doesn't look cheap at all. I'm so glad I looked on Ama
zon and found such an affordable tutu that isn't made poorly. A++", 'overall:5.0', 'summary:Great tutu-  not cheapl
y made', 'unixReviewTime:1297468800', 'reviewTime:02 12, 2011', '', 'reviewerID:A2G5TCU2WDFZ65', 'asin:0000031887',
'reviewerName:Amazon Customer', 'helpful:[0, 0]', 'reviewText:I bought this for my 4 yr old daughter for dance clas
s, she wore it today for the first time and the teacher thought it was adorable. I bought this to go with a light b
lue long sleeve leotard and was happy the colors matched up great. Price was very good too since some of these go f
or over $15.00 dollars.', 'overall:5.0', 'summary:Very Cute!!', 'unixReviewTime:1358553600', 'reviewTime:01 19, 201
3', '']
```

   b. Create a document to save the extracted text, which contains only the reviews contents (reviews.txt)
      The second figure shows the first 20 words in the new file.

```
#Extract only the reviewText from the file
def extract(files):
    f = open("/Users/LXIN/Desktop/T/reviews.txt",'w+')
    for var in files:
        if "reviewText" in var:
            varwrite = re.sub("reviewText:", "", var)
            f.write(varwrite+"\n")
    f.close()
extract(rawtextSplit)
```

```
textfile2 = open("/Users/LXIN/Desktop/T/reviews.txt")
reviewText = textfile2.read()
print(reviewText[:20])
```

```
This is a great tutu
```

2. Pre-processing the data

a. Tokenization. Open the new text file *reviews.* Separate the file content into tokens with word tokenizer.
The second figure shows the first 20 tokenized words in the file.

```
#Tokenize
tokens = nltk.word_tokenize(reviewText)
reviewToken = nltk.Text(tokens)
print(reviewToken[:20])
```

```
['This', 'is', 'a', 'great', 'tutu', 'and', 'at', 'a', 'really', 'great', 'price', '.', 'It', 'does', "n't", 'look'
, 'cheap', 'at', 'all', '.']
```

b. Lowercase, isalpha()
Convert all the alphabetical characters to lower case.
I added one more command to exclude words whose length is less than 1, in order to exclude the words with no meaning like “k”, “u” etc. *[len(w)>1]*
The second figure shows the first 20 words from the result.

```
#Fileters: stopwords, isalpha() and length
reviewLower = [w.lower() for w in reviewToken if w.isalpha() and len(w)>1]
print("reviewLower")
print(reviewLower[:20])
```

```
['this', 'is', 'great', 'tutu', 'and', 'at', 'really', 'great', 'price', 'it', 'does', 'look', 'cheap', 'at', 'all'
, 'so', 'glad', 'looked', 'on', 'amazon']
```

c. Stop words
Remove all the stop words, which are words commonly used such as “this” and “it”.
Note: We need to apply the lower() function first because the machine will not recognize the stop words in upper case.
The stop word list I used is the one given by NLTK. *[stopwords.words(‘english’)*

```
stopwords = nltk.corpus.stopwords.words('english')
reviewStop = [w for w in reviewLower if not w in stopwords]
print("stop")
print(reviewStop[:20])
```

```
['great', 'tutu', 'really', 'great', 'price', 'look', 'cheap', 'glad', 'looked', 'amazon', 'found', 'affordable', '
tutu', 'made', 'poorly', 'bought', 'yr', 'old', 'daughter', 'dance']
```

d. Lemmatization.
For the data pre-processing, I chose lemmatization instead of stemming because I found lemmatization performs more accurate, that it does not cut either the beginning or end of the word. Lemmatization also considers morphological analysis of the words, returns the lemma which is the base form.
I used Wordnet lemmatization and POS tagging to lemmatize the words. Note wordnet is a large lexical database for English.
A tag_map is created first, where the first letter of the pos_tag will be matched to the value from wordnet dictionary.
A new empty list is created to store the words after lemmatization. Use loop to lemmatize all the words, which takes two arguments, one is the word and the other is a mapping of pos_tag with wordnet value.
The second figure is the first 20 words after lemmatization.

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

wnl = nltk.WordNetLemmatizer()

reviewLem = []
for reviewStop, tag in pos_tag(reviewStop):
    lemma = wnl.lemmatize(reviewStop, tag_map[tag[0]])
    #print(reviewStop, "=>", lemma)
    reviewLem.append(lemma)
print(reviewLem[:20])
```

```
['great', 'tutu', 'really', 'great', 'price', 'look', 'cheap', 'glad', 'look', 'amazon', 'find', 'affordable', 'tu
tu', 'make', 'poorly', 'buy', 'yr', 'old', 'daughter', 'dance']
```

Data Analysis

1. **Frequency distribution.**

   Create a frequency distribution of the words using the NLTK FreqDist module.

```
#Get the top 50 frequency
fdist = FreqDist(reviewLem)
fdistKey = list(fdist.keys())
topkeys = fdist.most_common(50)
for pair in topkeys:
    print(pair)
```

```
('wear', 115219)
('fit', 109608)
('size', 99773)
('like', 98903)
('look', 98124)
('shoe', 81244)
('love', 79430)
('get', 78958)
('great', 78809)
('buy', 73317)
('well', 68160)
('would', 67419)
('good', 67118)
('one', 64270)
('comfortable', 57492)
('make', 55817)
('color', 52190)
('order', 50962)
('nice', 46638)
('really', 46108)
('go', 44047)
('small', 43320)
('little', 42764)
('foot', 40181)
('pair', 40127)
('time', 38954)
('price', 37498)
('work', 34984)
('use', 34509)
('quality', 33403)
('watch', 32702)
('perfect', 31926)
('large', 31584)
('also', 30148)
('big', 29996)
('much', 29976)
('purchase', 28577)
('think', 28250)
('feel', 28194)
('long', 28000)
('want', 27476)
('need', 27252)
('even', 26785)
('day', 26716)
('find', 26450)
('shirt', 25976)
('recommend', 25828)
('say', 25229)
('bra', 24803)
('come', 24157)
```

2. Bigram Frequency Distribution

a. Import the collection finder module. *from nltk.collocations import **

```
from nltk.collocations import *
```

b. Define a new function named *alpha_filter* which returns True if the character is non-alphabetical and False otherwise.

```
#Bigram Frequency Distributions
def alpha_filter(w):
    # pattern to match a word of non-alphabetical characters
    pattern = re.compile('^[^a-z]+$')
    if (pattern.match(w)):
        return True
    else:
        return False
```

c. Define a variable named BigramCollocationFinder and import the collocation finder module.

To include only the bigrams make sense, I apply two filter functions to exclude the non-alphabetical characters and the characters in stop word list.

```
#Get the top 50 Bigram frequency
bigram_measures = nltk.collocations.BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(reviewToken)
#finder = BigramCollocationFinder.from_words(reviewStop)
finder.apply_word_filter(alpha_filter)
finder.apply_word_filter(lambda w: w in stopwords)
scored = finder.score_ngrams(bigram_measures.raw_freq)
print("Bigram")
for bscore in scored[:50]:
    print(bscore)
```

```
Bigram
(('well', 'made'), 0.0005933578290056148)
(('would', 'recommend'), 0.0004463071306833277)
(('good', 'quality'), 0.0004102218806680609)
(('highly', 'recommend'), 0.0003568870880630786)
(('really', 'like'), 0.0003356060431822802)
(('fit', 'perfectly'), 0.0002956214712789167 5)
(('fit', 'well'), 0.0002921847808012723)
(('look', 'like'), 0.0002870297450848056)
(('looks', 'great'), 0.0002711680967264466)
(('another', 'pair'), 0.0002554386287710738 7)
(('look', 'great'), 0.0002527289305098542)
(('wear', 'size'), 0.0002432780316963319)
(('looks', 'like'), 0.0002398413412186874 4)
(('feel', 'like'), 0.0002334305916738506 7)
(('year', 'old'), 0.0002309191640171105)
(('fit', 'great'), 0.0002270859323305070 4)
(('great', 'price'), 0.0002237153320543557 5)
(('even', 'though'), 0.0002157184176736830 7)
```

```
(('usually', 'wear'), 0.0002081841347034625)
(('would', 'buy'), 0.00020725887188255824)
(('normally', 'wear'), 0.00019622180823320007)
(('light', 'weight'), 0.00019476678238003505)
(('one', 'size'), 0.00019185985493465132)
(('half', 'size'), 0.00018967887828537697)
(('long', 'time'), 0.0001862421878077325)
(('fits', 'well'), 0.00018353248954651284)
(('every', 'day'), 0.000177914822419594)
(('arch', 'support'), 0.00017639474778525125)
(('look', 'good'), 0.00017269369650163414)
(('size', 'larger'), 0.0001724293356956615)
(('little', 'bit'), 0.00016456460171797513)
(('really', 'nice'), 0.00016416806050901616)
(('first', 'time'), 0.00016271407607616658)
(('fits', 'perfectly'), 0.00016040091902390588)
(('much', 'better'), 0.00015802167177015203)
(('great', 'quality'), 0.00015537918988745896)
(('ordered', 'size'), 0.00015352753806861696)
(('looks', 'good'), 0.00015095002021038362)
(('high', 'quality'), 0.00015015693779246568)
(('perfect', 'fit'), 0.0001497639658350668)
(('different', 'colors'), 0.00014414272945658787)
(('long', 'enough'), 0.00014216002341179297)
(('fits', 'great'), 0.00014083821938192973)
(('love', 'love'), 0.00013859115253116218)
(('would', 'definitely'), 0.00013845897212817586)
(('flip', 'flops'), 0.00013746761910577842)
(('definitely', 'recommend'), 0.00012841326150121512)
(('second', 'pair'), 0.00012821499089673565)
(('really', 'cute'), 0.00012735581827732454)
(('fit', 'perfect'), 0.00012550529263551598)
```

3. **Bigrams by Mutual Information scores (min frequency 5)**
   Note that the PMI measure should be applied to the finder that removed non-alphabetic words and stopwords.

```
#Bigrams by their Mutual Information scores (using min frequency 5)
finder.apply_freq_filter(5)
scored2 = finder.score_ngrams(bigram_measures.pmi)
print("PMI")
for score in scored2[:50]:
    print (score)
```

```
(('badgley', 'mischka'), 21.52906027001395)
(('salvatore', 'exte'), 21.52906027001395)
(('showviewpoints', 'sortby'), 21.52906027001395)
(('spatestruck', 'lenders'), 21.52906027001395)
(('tessuto', 'vela'), 21.52906027001395)
(('krav', 'maga'), 21.266025864180154)
(('pepto', 'bismol'), 21.266025864180154)
(('herman', 'munster'), 21.043633442843706)
(('hypo', 'allergenic'), 21.043633442843706)
(('tku', 'trish'), 21.043633442843706)
(('myia', 'passiello'), 20.850988364901312)
(('birko', 'flor'), 20.85098836490131)
(('estado', 'execelente'), 20.85098836490131)
(('norman', 'reedus'), 20.85098836490131)
(('hola', 'gente'), 20.82124102150726)
(('saudi', 'arabia'), 20.82124102150726)
(('charlotte', 'russe'), 20.52906027001395)
(('mikels', 'esq'), 20.52906027001395)
(('giorgio', 'brutini'), 20.529060270013947)
```

```
(('grady', 'harp'), 20.391556746264015)
(('sherpani', 'soleil'), 20.365561537731068)
(('laurel', 'burch'), 20.239553652818962)
(('lbssize', 'lbssize'), 20.172916459788674)
(('fecha', 'indicada'), 20.169164324927564)
(('caslynn', 'lizzie'), 20.16649019062924)
(('gramado', 'rio'), 20.150548646760218)
(('carolyn', 'pollack'), 20.096100862737842)
(('vince', 'camuto'), 20.096100862737842)
(('buenas', 'tardes'), 20.043633442843706)
(('uacute', 'nico'), 20.043633442843706)
(('muk', 'luks'), 20.02501776467636)
(('liz', 'claiborne'), 19.97651924698517)
(('juanita', 'wilson'), 19.944097769292792)
(('hanky', 'panky'), 19.850988364901312)
(('strawberry', 'shortcake'), 19.850988364901312)
(('yak', 'trax'), 19.83306645690405)
(('bon', 'bebe'), 19.82862055187286)
(('audrey', 'hepburn'), 19.802625343339912)
(('muay', 'thai'), 19.681063363459)
(('farrell', 'manuel'), 19.670416119259485)
(('darth', 'vader'), 19.665121819589974)
(('nether', 'regions'), 19.654591152097808)
(('hallux', 'limitus'), 19.62859594356486)
(('alt', 'alt'), 19.62003393006144)
(('gloria', 'vanderbilt'), 19.603060851457727)
(('pom', 'poms'), 19.603060851457723)
(('puerto', 'rico'), 19.52906027001395)
(('aurora', 'borealis'), 19.529060270013947)
(('tai', 'chi'), 19.529060270013947)
(('buzz', 'lightyear'), 19.45867094212255)
```

Interpretation

1. **What I have learned about the reviews**

   a. Top 50 Frequency distribution

   Note that several words with high frequency are about size, such as “wear”, “fit”, and “size”. And several Noun words are under the clothing category such as “shoe” and “shirt”. This implies that most reviews customers wrote are about clothes they bought, and what they might concern most is if the clothes fit.

   It is reasonable to say that what customers bought most from Amazon are clothes and other stuffs wear.

   Price is another thing most of them mentioned, as “price” has a frequency of 37498.

   b. Top 50 Bigram Frequency distribution

   Note that most of the bigrams are positive comments towards products, such as “well made”, “good quality” and “highly recommend”, which implies that most customers are satisfied with what they purchased.

   Several bigrams with high frequency like “fit well” and “fit perfectly” indicates that conclusion from part(a) seems reasonable for part(b) also. Products purchased most on Amazon are under clothing category.

   Since the bigrams related to quality have high frequency, we could conclude that customers care the quality the most.

   c. Top 50 Bigrams by Mutual Information scores

   The output seems strange at first glance, that the bigrams do not look like those we see in daily life. However, once we search them in the text file, we could conclude

that they might be the name of brands, name of people, and some other possible word groups.
We could summarize some interesting conclusions from this output.
For example, popular brands among customers on Amazon. It is reasonable to conclude that Badgley Mischka is the most popular brand since it has the highest frequency.

d.
Though it is easy for people to catch useful information while reading, it needs various tasks for machine to perform the same job, like tokenization, lemmatization and removing stop words. However, with necessary training, machine could process and analyze abundant data fast and accurate in short time.

2. Additional analysis tasks needed

a. There are some methods I used that are not 100% accurate. For example, when I tried to exclude the words which are non-alphabetic and in stop word list, the output indicates that there are some words being cut roughly. Therefore, additional analysis tasks to prevent situations like that are necessary.

```
['this', 'great', 'tutu', 'really', 'great', 'price', 'it', 'look', 'cheap', 'glad', 'looked', 'amazon', 'found', '
affordable', 'tutu', 'made', 'poorly', 'bought', 'yr', 'old']
```

b. Additional data pre-processing tasks might be important if we want to analyze the reviews in detailed. Though we have the frequency distribution, we do not know what most customers think about the products they purchased in each category. For example, are most the reviews positive or negative? What is the main concern customers have? We could perform the tasks to extract the reviews and tag them with “positive” and “negative” and analyze them.

c. One thing needed to be improved in the future is that though we extracted stop words and non-alphabetic characters, there are still many words that are not that useful for analyzing the reviews. For example, many of the Verb words with high frequency like “get” and “buy” do not provide too much information. However, we could not simply exclude the Verb words.