Assignment 2 - NLP Stemming and Lemmatization

Matthew Dunne

The Data

Improve your analysis from Week-1 by identifying specific words / keywords / N-Grams that lead to failed food inspections in Chicago leveraging tokenization, stemming, lemmatization and N-Gramming in Python.

In [15]:

```
import pandas as pd
import nltk as nltk
import nltk.corpus
#use only the columns you need: whether the inspection failed and the specific violations
df = pd.read_csv('Food_Inspections.csv', usecols=['Results', 'Violations'])
#drop any rows with NA/NaN in either column
df=df.dropna()
df.head()
```

Out[15]:

	Results	Violations
1	Pass w/ Conditions	3. MANAGEMENT, FOOD EMPLOYEE AND CONDITIONAL E...
2	Pass w/ Conditions	5. PROCEDURES FOR RESPONDING TO VOMITING AND D...
3	Fail	3. MANAGEMENT, FOOD EMPLOYEE AND CONDITIONAL E...
4	Pass w/ Conditions	3. MANAGEMENT, FOOD EMPLOYEE AND CONDITIONAL E...
6	Fail	1. PERSON IN CHARGE PRESENT, DEMONSTRATES KNOW...

Create two dataframes, one of failed inspections and one of passed inspections.

In [16]:

```
failed=df.loc[df['Results']=='Fail']
passed=df.loc[(df['Results']=='Pass w/ Conditions') | (df['Results']=='Pass')]
violations=failed['Violations']
len(failed)
```

Out[16]:

```
31076
```

Each row in the dataset includes multiple violations. When broken up separately they look as follows.

In [17]:

```
string=violations.iloc[0]
splits=string.count('|')
offenses=string.split('|', maxsplit=splits)
offenses
```

Out[17]:

```
['3. MANAGEMENT, FOOD EMPLOYEE AND CONDITIONAL EMPLOYEE; KNOWLEDGE, RESPONSIBILITIES AND REPORTING - Comments: 2-102.14(N) OBSERVED NO EMPLOYEE HEALTH POLICY ON THE PREMISES. MANAGEMENT INSTRUCTED TO PROVIDE A SIGNED EMPLOYEE HEALTH POLICY FOR EACH EMPLOYEE. PRIORITY FOUNDATION 7-38-010. NO CITATION ISSUED. ',
 ' 5. PROCEDURES FOR RESPONDING TO VOMITING AND DIARRHEAL EVENTS - Comments: 2-501.11 OBSERVED NO CLEAN UP PROCEDURE ON THE PREMISES FOR VOMIT AND DIARRHEAL EVENTS. MANAGEMENT INSTRUCTED TO PROVIDE A CLEAN UP PROCEDURE AND SUPPLIES. PRIORITY FOUNDATION 7-38-005. NO CITATION ISSUED. ',
 ' 47. FOOD & NON-FOOD CONTACT SURFACES CLEANABLE, PROPERLY DESIGNED, CONSTRUCTED & USED - Comment
```

```
 ' 47. FOOD & NON-FOOD CONTACT SURFACES CLEANABLE, PROPERLY DESIGNED, CONSTRUCTED & USED - Comment
s: 4-501.11(A)  REPAIR THE LEAKING WATER FROM THE DISHWASHING MACHINE. ',
 ' 49. NON-FOOD/FOOD CONTACT SURFACES CLEAN - Comments: 4-602.13 CLEAN AND SANITIZE THE COFFEE AND
DRINK DISPENSER STATIONS BEHIND THE FRONT COUNTER AND THE EXTERIOR OF THE FROSTY MACHINE. CLEAN TH
E GREASE FROM THE VENTILATION HOOD AND FILTERS. CLEAN THE INTERIOR OF BOTH SIDES OF THE CONDIMENT
COOLER AT THE SANDWICH PREP STATION AND THE TOP OF THE DISHMACHINE. ',
 ' 55. PHYSICAL FACILITIES INSTALLED, MAINTAINED & CLEAN - Comments: 6-501.12 CLEAN THE FLOOR
BEHIND THE FRONT COUNTER AND BEVERAGE STATION. CLEAN THE GREASE ON THE WALL BEHIND THE DEEP FRYERS
. ',
 ' 58. ALLERGEN TRAINING AS REQUIRED - Comments: PA 100-0367 OBSERVED THE CERTIFIED FOOD MANAGER W
ITHOUT THE ALLERGEN TRAINING. MANAGEMENT INSTRUCTED TO SHOW PROOF OF TRAINING. ',
 ' 59. PREVIOUS PRIORITY FOUNDATION VIOLATION CORRECTED - Comments: 8-404.13  PREVIOUS PRIORITY FO
UNDATION VIOLATION #38 WAS NOT CORRECTED FROM REPORT #2222608 ON 9/20/18:  38 -  OBSERVED OVER 40
LIVE SMALL FLIES THROUGHOUT THE PREP AREA AND DINING AREA AND OUTSIDE OF THE WASHROOMS.
MANAGEMENT INSTRUCTED TO MINIMIZE OR ELIMINATE THE PEST ACTIVITY.  PRIORITY 7-42-090.  CITATION IS
SUED.']
```

We will focus on the languge after the comments. Looking at the first violation from the above list, we see which part this is:

In [18]:

```
import re
first=offenses[0].split(' - Comments: ')[1]
first
```

Out[18]:

```
'2-102.14(N) OBSERVED NO EMPLOYEE HEALTH POLICY ON THE PREMISES. MANAGEMENT INSTRUCTED TO PROVIDE
A SIGNED EMPLOYEE HEALTH POLICY FOR EACH EMPLOYEE. PRIORITY FOUNDATION 7-38-010. NO CITATION ISSUE
D. '
```

Collecting All of the Language in the Comments

We want to collect all of the relevant comments into one long list.

In [19]:

```
strings=violations
offenses=[]
details=[]
for i in range(len(strings.index)):
    #for every row in data set
    string=strings.iloc[i]
    #find the number of separate violations
    splits=string.count('|')
    #split for each violation to go through them separately
    new_offenses=string.split('|', maxsplit=splits)
    #for each separate offense in that observation
    for j in range(len(new_offenses)):
        #for j in range(len(new_offenses)): Had this in original submission. Don't know why throws
off numbers
        #you have run the whole notebook. Didn't really affect results.
        #not all separate violations have comments. Look only at those that do
        if re.search('Comments: ', new_offenses[j]):
            #split and take the narrative part
            detail=new_offenses[j].split(' - Comments: ')[1]
            #and put that into one long list
            details.append(detail)
        else:
            continue
```

How many Comments are there among all separate violations for failed inspections?

In [22]:

```
len(details)
```

Out[22]:

```
186643
```

Now we go through each comment, tokenize, remove single character tokens, remove numbers, render all to lowercase, and then take out stopwords.

NOTE: THIS CELL TIMES A LONG TIME TO RUN. DO NOT RUN UNLESS YOU HAVE TO!!!

In [23]:

```
words_collection=[]
#stopwords = stopwords.words('english')
stopwords = set(nltk.corpus.stopwords.words('english'))
for i in range(len(details)):
    #tokenize
    words = nltk.tokenize.word_tokenize(details[i])
    # Remove single-character tokens (mostly punctuation)
    words = [word for word in words if len(word) > 1]
    # Remove numbers
    words = [word for word in words if word.isalpha()]
    # Lowercase all words (default_stopwords are lowercase too)
    words = [word.lower() for word in words]
    # Remove stopwords
    words = [word for word in words if word not in stopwords]
    #and then put that into one long list
    #per Igor not a good idea for the nested loop to have the index named the same way (i). Might
create errors although probably not in this case
    for i in words:
        words_collection.append(i)
```

How many individual words are there in these comments?

In [24]:

```
len(words_collection)
```

Out[24]:

```
4047482
```

What is the length of the frequency distribution, i.e. how many distinct words?

In [25]:

```
fdist = nltk.FreqDist(words_collection)
```

In [26]:

```
word_counts=fdist
len(word_counts)
```

Out[26]:

```
19106
```

Take the frequency distribution and create a list of the respective words and numbers.

In [27]:

```
all_words=list(word_counts.keys())
all_numbers=list(word_counts.values())
```

Stemming

Instead of asking which words are the most common, we might ask which word stems are the most common.

First we try the Porter Stemmer.

First we stem.

In [28]:

```
porter = nltk.PorterStemmer()

#your count of words is a list of tuples. Create two separate lists. One for the stem of the word.
One for the count of that word.
p_stems=[porter.stem(t) for t in all_words]
numbers=[t for t in all_numbers]
```

Then we aggregate based on the stem.

In [29]:

```
p_dict={}
for i in range(len(numbers)):
    #HAVE TO DO IT THIS WAY. WHEN YOU STEM IT THE RESULTING LIST'S INDEX ISN'T SEQUENTIAL
    wor=p_stems[i]
    num=numbers[i]
    if wor not in p_dict.keys():
        p_dict[wor]=num
    else:
        p_dict[wor]+=num
```

Now the Lancaster Stemmer

In [30]:

```
lancaster = nltk.LancasterStemmer()
l_stems=list([lancaster.stem(t) for t in all_words])
numbers=[t for t in all_numbers]
```

We see that the Porter Stemming and the Lancaster Stemming have noticeably different results (stems). Here are the first ten stems under each method.

In [31]:

```
print(p_stems[0:10])
print(l_stems[0:10])
```

```
['observ', 'employe', 'health', 'polici', 'premis', 'manag', 'instruct', 'provid', 'sign',
'prioriti']
['observ', 'employ', 'heal', 'policy', 'prem', 'man', 'instruct', 'provid', 'sign', 'pri']
```

Then we aggregate based on the Lancaster stems.

In [32]:

```
l_dict={}
for i in range(len(numbers)):
    wor=l_stems[i]
    num=numbers[i]
    if wor not in l_dict.keys():
        l_dict[wor]=num
    else:
        l_dict[wor]+=num
```

Comparing Porter and Lancaster Stemmers

When we compare the Top 50 stems (after aggregation) from both Porter and Lancaster we see a great deal of overlap in stems (the form of which can vary between the two). There is some variation in order however.

What is noticeable, especially for the Porter stems, is that the first six tokens essentially form a human-readable sentence that could describe the reason for many failed inspections: "Must clean food area, floor, sink." Interestingly, for the Lancaster stems the 'stor' stem, probably meaning storage is more prominent than the 'store' stem in Porter.

In [33]:

```
from collections import OrderedDict
#sort the dictionary (porter and lancaster) by value in descending order
p_dict_sorted=OrderedDict(sorted(p_dict.items(), key=lambda x: x[1], reverse=True))
l_dict_sorted=OrderedDict(sorted(l_dict.items(), key=lambda x: x[1], reverse=True))
#and take the top 10
p_top50=list(p_dict_sorted.items())[0:50]
l_top50=list(l_dict_sorted.items())[0:50]
print('Top 50 Stems (Porter)\n')
print(p_top50)
print('\nTop 50 Stems (Lancaster)\n')
print(l_top50)
```

```
Top 50 Stems (Porter)

[('clean', 105242), ('must', 101994), ('area', 95054), ('food', 93438), ('floor', 66095), ('sink',
63074), ('instruct', 62072), ('prep', 54303), ('maintain', 52956), ('observ', 46797), ('wall', 451
35), ('repair', 43999), ('cooler', 43094), ('provid', 43078), ('storag', 42200), ('violat',
41808), ('shall', 38876), ('door', 36463), ('remov', 34827), ('rear', 33551), ('seriou', 29955), (
'equip', 29229), ('water', 29008), ('store', 26510), ('sanit', 25402), ('room', 24743), ('replac',
23817), ('insid', 22691), ('use', 22101), ('ceil', 22081), ('machin', 21508), ('hand', 21496), ('i
ce', 21363), ('shelv', 21302), ('pest', 21119), ('front', 20793), ('compart', 20708), ('manag', 19
640), ('detail', 18619), ('premis', 18304), ('issu', 18221), ('kitchen', 18169), ('found', 17990),
('basement', 17835), ('cook', 17716), ('light', 17659), ('control', 17633), ('need', 17551), ('cit
at', 17366), ('rodent', 16983)]

Top 50 Stems (Lancaster)

[('cle', 113861), ('must', 101996), ('food', 93438), ('stor', 68738), ('are', 68042), ('flo', 6658
7), ('prep', 65650), ('sink', 63076), ('instruct', 62099), ('maintain', 52969), ('observ', 46796),
('wal', 45145), ('repair', 43999), ('cool', 43583), ('provid', 43080), ('viol', 41847), ('shal', 3
8885), ('door', 36465), ('remov', 34828), ('rear', 33551), ('sery', 29952), ('equip', 29230), ('wa
t', 29045), ('area', 27013), ('sanit', 26541), ('room', 24745), ('bas', 24625), ('replac', 23820),
('us', 22877), ('insid', 22692), ('serv', 22688), ('ceil', 22088), ('machin', 21512), ('hand', 215
02), ('ic', 21365), ('shelv', 21302), ('pest', 21122), ('prop', 20801), ('front', 20793),
('compart', 20716), ('man', 20128), ('found', 19875), ('op', 18964), ('detail', 18619), ('prem', 1
8306), ('issu', 18221), ('kitch', 18186), ('cook', 17863), ('light', 17675), ('control', 17633)]
```

Lemmatization

The WordNet lemmatizer only removes affixes if the resulting word is in its dictionary. Therefore they will look different from the stems.

In [34]:

```
wnl = nltk.WordNetLemmatizer()
lemmas=[wnl.lemmatize(t) for t in all_words]
print('\nLemmas\n')
print(lemmas[0:20])
print('\nPorter Stems\n')
print(p_stems[0:20])
```

```
Lemmas

['observed', 'employee', 'health', 'policy', 'premise', 'management', 'instructed', 'provide',
'signed', 'priority', 'foundation', 'citation', 'issued', 'clean', 'procedure', 'vomit',
'diarrheal', 'event', 'supply', 'repair']

Porter Stems

['observ', 'employe', 'health', 'polici', 'premis', 'manag', 'instruct', 'provid', 'sign',
'prioriti', 'foundat', 'citat', 'issu', 'clean', 'procedur', 'vomit', 'diarrheal', 'event', 'suppl
i', 'repair']
```

In [35]:

```
len(lemmas)
```

Out[35]:

Out[35]:

```
19106
```

Now we aggregate based on the lemmas.

In [36]:

```
lemma_dict={}
for i in range(len(numbers)):
    wor=lemmas[i]
    num=numbers[i]
    if wor not in lemma_dict.keys():
        lemma_dict[wor]=num
    else:
        lemma_dict[wor]+=num
```

Then we look at what the top lemmas are by count.

In [37]:

```
lemma_dict_sorted=OrderedDict(sorted(lemma_dict.items(), key=lambda x: x[1], reverse=True))

#and take the top 50
lemma_top50=list(lemma_dict_sorted.items())[0:50]
print(lemma_top50)
```

```
[('must', 101994), ('area', 95055), ('food', 93437), ('clean', 81946), ('floor', 65871), ('sink',
63074), ('instructed', 61988), ('prep', 54062), ('observed', 46508), ('maintain', 45339), ('wall',
45135), ('cooler', 43094), ('repair', 42343), ('storage', 42193), ('violation', 41804), ('shall',
38875), ('door', 36462), ('provide', 34892), ('rear', 33547), ('remove', 31696), ('serious', 29947
), ('equipment', 29043), ('water', 29007), ('room', 24742), ('inside', 22689), ('replace', 22400),
('ceiling', 22080), ('machine', 21507), ('hand', 21480), ('ice', 21301), ('pest', 21119),
('front', 20793), ('compartment', 20695), ('premise', 18301), ('kitchen', 18169), ('found',
17990), ('issued', 17947), ('basement', 17834), ('control', 17598), ('citation', 17364),
('rodent', 16982), ('detail', 16899), ('hot', 16137), ('service', 16055), ('stored', 16005),
('manager', 15671), ('behind', 15513), ('interior', 15373), ('grease', 15227), ('droppings',
15196)]
```

We see the results are very similar to what we got with the stems for the very top words. There is more nuance in the lemmas, where we see 'storage' and 'stored' are kept separate, and so the words are even more readable by humans.