Lab 3: Python Data Structures

Stuart Harley

Introduction: In this lab we create a list of the most popular search terms (tokens) for a given set of search queries. As people are imperfect, they often misspell search terms, so we are using a spell checking library to remove and interpret misspellings to find the actual most popular terms (and not just the most popular and consistently spelled terms). The search terms come from Direct Supply's DSSI eProcurement system.

Learning Outcomes:

- Using Python Lists, Sets, Dictionaries
- Basic natural language processing
- Plotting with matplotlib

Importing Libraries

In [1]:
```
from spellchecker import SpellChecker
import pattern.en
import csv
import time
import sys
import itertools
```

Creating a list of tokens: In this cell the csv file containing the seach terms is accessed. A list containing the search tokens that are all lowercase, have no surrounding paranthesis, and have no spaces is created.

In [2]:
```
%%time
token_list = []

with open('/home/harleys/searchTerms.csv') as file:
    next(file) # Gets rid of title line of csv file
    for line in file:
        split_line = line.split(",")
        if split_line[0].startswith('"'):
            split_line[0] = (split_line[0])[1] # removes possible preceeding "
        if split_line[0].endswith('"'):
            split_line[0] = (split_line[0][:-1]) # removes possible following
 "
        split_again = split_line[0].split("%20") # Splits token at any '%20'
        for token in split_again:
            split_token = token.split(" ")  # Splits token at any space
            for token2 in split_token:
                token_list.append(token2.lower())
print("bytes in token_list: " + str(sys.getsizeof(token_list)))
```

```
bytes in token_list: 12383856
CPU times: user 1.14 s, sys: 52.9 ms, total: 1.19 s
Wall time: 1.17 s
```

Creating a dictionary of tokens: In this cell a dictionary of tokens is created from the list of tokens previously generated. The key for the dictionary is the token and the value is the number of times that token occurs in the list. (The lab instructions state to create a dictionary from the set, but this would mean that every value of the dictionary would be 1, since a set contains no duplicates.)

In [3]:
```
%%time
token_dict = {}
for token in token_list:
    if token in token_dict:
        token_dict[token] += 1
    else:
        token_dict[token] = 1
del token_dict[""] # easier to remove the empty string than check for it
print("bytes in token_dict: " + str(sys.getsizeof(token_dict)))
```

```
bytes in token_dict: 2621544
CPU times: user 328 ms, sys: 0 ns, total: 328 ms
Wall time: 327 ms
```

Creating an output csv file from the dictionary of tokens

In [4]:
```
with open('/home/harleys/all_tokens.csv', 'w') as file:
    writer = csv.DictWriter(file, fieldnames=["SearchToken", "Occurances"])
    writer.writeheader()
    for key in token_dict.keys():
        file.write("%s,%s\n"%(key,token_dict[key]))
```

Example results of tokens and their number of occurances: In this cell several entries from the token dictionary were printed out, which is effectively the all_tokens.csv output file, just without the formatting.

In [5]:
```
print(dict(itertools.islice(token_dict.items(), 50)))
```

```
{'36969': 1, 'cmed': 4, '500100': 5, 'kend': 153, '5750': 61, '980228': 25,
'dync1815h': 4, 'dynd70642': 329, 'dees': 11, 'kc-21400': 1, 'link': 921, 'pc
1000': 27, '7081714': 30, '8507sa': 83, '8881-892910': 1, 'bacon': 6195, 'pin
eapple': 3118, '5065265': 2, 'enfit70550': 119, 'cheese': 13928, 'cheddar': 1
757, '68010': 18, '55507': 115, '8116055': 123, '2366607': 154, 'buttermilk':
457, '3009697': 19, '4185775': 131, '1953358': 18, '2157315': 19, '4782694':
63, '6653558': 3, '7062615': 2, 'milk': 4778, 'chicken': 19206, 'breast': 289
9, 'drit': 22, '0028': 1, '4944450': 41, 'romain': 135, 'banana': 3808, '16s
l': 43, 'uro51211ch': 2, 'huds': 11, '00640': 1, '7024755': 10, '6056105': 1
3, '6928832': 21, 'name': 121, 'tags': 38}
```

Analyzing the output file: The words with the most occurances in the output file words were basic foods like chicken, strawberry, and banana. The more uncommon or specific the term, the less occurances there were. Also, if a word was clearly misspelled, there were very few occurance.

There are also many "nonwords" in the data. For the most part, these are tokens that contain all numbers or a combination of words and numbers. I suspect that these are the serial codes or numbers for the product that is being ordered. These could be easier for a customer to use if they know exactly what they are looking to purchase.

Creating a new dictionary that contains only alphabetic tokens: In this cell a new dictionary is created from the previous dictionary that contains only tokens that are only made up of alphabetic characters

In [6]:
```
%%time
alpha_token_dict = {}
for key in token_dict:
    if key.isalpha():
        alpha_token_dict[key] = token_dict[key]
print("bytes in alpha_token_dict: " + str(sys.getsizeof(alpha_token_dict)))
```

```
bytes in alpha_token_dict: 295008
CPU times: user 20.9 ms, sys: 53 µs, total: 21 ms
Wall time: 20.3 ms
```

Creating a set from the dictionary of alphabetic tokens: In this cell a set is created from the dictionary of the alphabetic tokens

In [7]:
```
%%time
token_set = set()
for key in alpha_token_dict:
    token_set.add(key)
print("bytes in token_set: " + str(sys.getsizeof(token_set)))
```

```
bytes in token_set: 524512
CPU times: user 2.94 ms, sys: 0 ns, total: 2.94 ms
Wall time: 2.95 ms
```

Spellchecking the set and adding the correct spellings to a new dictionary: In this cell a dictionary is created with the alphabetic token as the key and the possile correct spelling as the value.

In [8]:
```
%%time
correct_spelling_dict = {}
spell = SpellChecker(distance=2)
for token in token_set:
    correct_spelling_dict[token] = spell.correction(token)
print("bytes in correct_spelling_dict: " + str(sys.getsizeof(correct_spelling_
dict)))
```

```
bytes in correct_spelling_dict: 295008
CPU times: user 24min 13s, sys: 39.3 s, total: 24min 52s
Wall time: 24min 52s
```

Creating an output csv file from the dictionary of tokens and their correct spellings

In [9]:
```
with open('/home/harleys/tokens_and_correct_spellings.csv', 'w') as file:
    writer = csv.DictWriter(file, fieldnames=["SearchToken", "CorrectSpelling"
])
    writer.writeheader()
    for key in correct_spelling_dict.keys():
        file.write("%s,%s\n"%(key,correct_spelling_dict[key]))
```

Example results of tokens and their possible correct spellings: In this cell the dictionary containing the tokens and their possible correct spellings was printed out which is effectively the all_tokens.csv output file, just without the formatting.

In [10]:
```
print(dict(itertools.islice(correct_spelling_dict.items(), 50)))
```

```
{'eakin': 'akin', 'miso': 'miso', 'watercolors': 'watercolours', 'biatain':
'britain', 'cookie': 'cookie', 'margerine': 'margarine', 'intermatic': 'inter
matic', 'metamuc': 'metamuc', 'renavite': 'renovate', 'remedy': 'remedy', 'el
iminating': 'eliminating', 'ezo': 'ego', 'preparation': 'preparation', 'sharp
s': 'sharps', 'champagne': 'champagne', 'shrebert': 'herbert', 'reddies': 'ed
dies', 'icybay': 'icybay', 'padfolio': 'padfolio', 'sombra': 'sombre', 'shipp
ing': 'shipping', 'squee': 'see', 'discharge': 'discharge', 'crayons': 'crayo
ns', 'mang': 'many', 'screens': 'screens', 'tilapis': 'tilapia', 'enmotion':
'emotion', 'after': 'after', 'pull': 'pull', 'musle': 'muscle', 'ties': 'tie
s', 'ointments': 'ointments', 'chimichanga': 'chimichanga', 'readycare': 'roa
dcare', 'blende': 'blonde', 'famatodine': 'famatodine', 'bismuth': 'bismuth',
'styrofoam': 'styrofoam', 'loradine': 'loraine', 'razo': 'razor', 'loperimad
e': 'loperamide', 'nebulize': 'nebulae', 'shasta': 'shasta', 'tasteeo': 'tast
e', 'towell': 'towel', 'eruption': 'eruption', 'hydrocorisone': 'hydrocortiso
ne', 'linguini': 'linguist', 'pillsbury': 'pillsbury'}
```

Creating a final set of tokens: In this cell, the final set of spell checked tokens is created.

In [11]:
```
%%time
final_spelled_set = set()
for key in correct_spelling_dict:
    final_spelled_set.add(correct_spelling_dict[key])
print("bytes in final_spelled_set: " + str(sys.getsizeof(final_spelled_set)))
```

```
bytes in final_spelled_set: 524512
CPU times: user 4.14 ms, sys: 4 µs, total: 4.14 ms
Wall time: 4.1 ms
```

Creating a final dictionary of tokens: In this cell, the final dictionary of spelled checked tokens is created. If a token was misspelled and a correct spelling was found, then the number of occurances of the misspelled word is added to the number of occurances of the correctly spelled word.

In [12]:
```
%%time
final_spelled_dict = {}
for token in final_spelled_set:
    final_spelled_dict[token] = 0
for key in correct_spelling_dict.keys():
    final_spelled_dict[correct_spelling_dict[key]] += token_dict[key]
print("bytes in final_spelled_dict: " + str(sys.getsizeof(final_spelled_dict
)))
```

```
bytes in final_spelled_dict: 295008
CPU times: user 5.44 ms, sys: 4.04 ms, total: 9.47 ms
Wall time: 9.32 ms
```

Creating an output csv file from the dictionary of correctly spelled tokens

In [13]:
```
with open('/home/harleys/correctly_spelled_tokens.csv', 'w') as file:
    writer = csv.DictWriter(file, fieldnames=["SearchToken", "Occurances"])
    writer.writeheader()
    for key in final_spelled_dict.keys():
        file.write("%s,%s\n"%(key,final_spelled_dict[key]))
```

Analyzing the output file: This file is very similar to the first csv file I created however it does not include non-alphabetic tokens. The frequency of some of the correctly spelled words is a bit higher than they were before. This is because identified misspelling occurances were added to the correctly spelled totals.

I think that this file is more accurate than the non-spelled checked version in general because misspellings are not intended and to effectively analyze the tokens, you need to account for them with spellcheck. However, words that were too badly mispelled could not be corrected. Also, it is likely that a misspelled word was corrected to the wrong word. Also it is likely that a missplelled word was intentially spelled that way. For these reasons, this list is most likely not perfect, however, it is still better than the non-spell checked version.

Conclusion

- **Wall time comparison**: The slowest cell, by a huge margin, is the cell in which the spellcheck takes place. This is because the correction method takes a long time to run. Other cells have to iterate through lists/sets/dictionaries a couple times, but this is done relatively quick.
- **Big-O analysis of the slowest cell**: When looking at the spellcheck method's code, it appears that this algorithm is O(n^distance), where n is the length of the word. (Distance is set to 2 in the above cell.) Distance is the amount of characters that the word can be off by. So it looks for a word within 'distance' changed letters.
- **Estimations**: Because the spellcheck method's n is based off of word size, (not the list size) increasing the size of the list to be iterated through should increase the time it takes by the same factor. So a csv file that is 10 times the length should take 10 times the time, or about 4 hrs. A csv file that is 100 times the size should take 100 times the time, or about 40 hrs.