Lab 2: Search Terms

Submitted By: Julian Singkham
Date: 12/18/2020

Abstract

The purpose of this lab is to familize ourselves with data cleaning, search term analytics, and spellchecking.

- The first objective was to derive search terms from a csv files and clean the data.
- The second objective was to create a frequency dictionary of the search terms.
- The final objective was to spellcheck the search terms using spellchecker and create a new spellchecked frequency dictionary

The data utilized in this lab is a search term csv file that contains about 1 million search terms used in the direc supply DSSI ecommerce platform

Parameters

In [1]:

```
from spellchecker import SpellChecker
import pattern.en
import csv

csv_freq_dict = {}
csv_freq_dict_spellchecked = []
```

Functions

Imports a CSV file and creates a list of the first item of each row.

Param csv: Name of the CSV file
Return:A list of the first item of each row of the csv

In [2]:

```
def import_csv_list_first_col(csv):
    temp = []
    csv_raw_data = []
    i = 0
    with open(csv, encoding='utf8') as file:
        for line in file:
            if i == 10000:
                break
            temp.append(line.rstrip('\n').split(','))
            i += 1
    csv_raw_data = [str(row[0]) for row in temp]
    file.closed
    return csv_raw_data
```

Given a list of strings, create a new list where each string is split by spaces.
EX: "Spicy Bacon" would be ["spicy", "bacon"]

Param original_list: List to split
Return: A list of single word strings

In [3]:

```
def split_tokens(original_list):
    new_list = []
    for item in original_list:
        new_list.extend(item.split(" "))
    return new_list
```

Replaces web spaces with a regular space from a string token

Param token: String token
Return: A string without web spaces

In [4]:

```
def remove_web_spaces(token):
    token.replace("%20", " ")
    return token
```

Removes non-alphabet characters from a string token

Param token: String token
Return: A string with only alphabet characters

In [5]:

```
def remove_non_alphabet(token):
    fixed_token = ""
    for char in token:
        if char.isalpha() or char == " ":
            fixed_token = fixed_token + char
    return fixed_token
```

Creates a frequency dictionary given a string list where the key is a string and the key-value is how many times the string appeared in the list.

Param input_list: String list
Return: A frequency dictionary

In [6]:

```
def list_to_freq_dict(input_list):
    freq_dict = {}
    for i in input_list:
        freq_dict[i] = input_list.count(i)
    return freq_dict
```

Creates a sorted frequency list given a frequency dictionary

Param freq_dict: Frequnecy dictionary
Return: A 2d list where the first row is frequency and the second row is the string

In [7]:

```
def sort_freq_dict(freq_dict):
    sorted_list = [(freq_dict[key], key) for key in freq_dict]
    sorted_list.sort()
    sorted_list.reverse()
    return sorted_list
```

Creates a spellchecker dictionary where the key is the misspelled word and the key-value is the most likely corrected word

Param input_list: List of misspelled words
Return: A spellecheck dictionary

In [8]:

```
def spellcheck_dict_init(input_list):
    spell = SpellChecker(distance=1)
    spellchecked_dict = {}
    for word in input_list:
        spellchecked_dict[word] = spell.correction(word)
    return spellchecked_dict
```

Given a misspelled string token, return the most likely corrected word

Param token: Misspelled token
Return: A correctly spelled word

In [9]:

```
def spellcheck_token(token):
    fixed_token = csv_spellcheck_dict[token]
    return fixed_token
```

Creates a csv file from a frequency list

Param input_list: Frequency list
Return: NONE

In [10]:

```python
def list_to_csv(input_list):
    fields = ["Frequency", "Word"]
    with open("Frequency of search terms.csv", "w", newline="") as file:
        write = csv.writer(file)
        write.writerow(fields)
        write.writerows(input_list)
```

Main

In [11]:

```python
# Import csv to search term list
%time csv_raw = import_csv_list_first_col("searchTerms.csv")
```

```
Wall time: 8.97 ms
```

In [12]:

```python
%%time
# This section of code filters the data from the csv file by removing non-alp
# replacing web spaces with spaces, and splitting search terms by word

csv_filtered = []
for i in range(len(csv_raw)):
    temp = remove_non_alphabet(remove_web_spaces(csv_raw[i]))
    csv_filtered.append(temp)
```

```
Wall time: 13 ms
```

In [13]:

```python
%time csv_filtered = split_tokens(csv_filtered)
```

```
Wall time: 2 ms
```

In [14]:

```python
%%time
# This section removes all blank search terms. Blank search terms are generat
# due to the removal of non-alphabet characters.
csv_fixed = []
for word in csv_filtered:
    if len(word) != 0:
        csv_fixed.append(word)
```

```
Wall time: 1.99 ms
```

In [15]:

```python
# This section of code creates a frequency dictionary of the filtered data.
# A list is also created to sort the frequency dictionary from most frequent
%time csv_freq_dict = list_to_freq_dict(csv_fixed)
%time csv_freq_list = sort_freq_dict(csv_freq_dict)
```

```
Wall time: 1.23 s
Wall time: 994 µs
```

In [16]:

```
%%time
# This section of code creates a spellchecked version of the fully filtered s
# Additionally it creates a frequency dictionary and a sorted list of the mos
csv_spellcheck_dict = spellcheck_dict_init(csv_fixed)
csv_spellchecked = []
for word in csv_fixed:
    csv_spellchecked.append(spellcheck_token(word))
```

```
Wall time: 795 ms
```

In [*]:

```
%time csv_spellcheck_dict = list_to_freq_dict(csv_spellchecked)
%time csv_spellcheck_freq_list = sort_freq_dict(csv_spellcheck_dict)
```

In [*]:

```
# Creates a csv file of the spellchecked search term frequency list from most
%time list_to_csv(csv_spellcheck_freq_list)
```

Conclusion

- The most frequent search tokens for the non-spellchecked data is food items like bacon, milk, chicken, beef. Bacon by far has the most hits with 459, followed by milk at 180 and chicken at 168. This is most likely due to the American breakfast having bacon as one of its components, additionally bacon is the most enjoyed bacon item so this makes sense. Food being the primarily searched items is logical as humans require a large amount of food on a daily basis.
- I hypothesis that numerical only entries are the UIC of the desired product. This would make sense as if I were to look up bacon there would be several different types one could buy, by using the specific UIC of the desired bacon, only one item pops up. Special characters could be search operators such as "XXX" being used to specify that the item must contain XXX in the name. "-XXX" could be used to specify that the item must not contain XXX.
- The spellchecked data in comparison to the original for the most part is the same. Some items saw a slight increase in frequency such as Juice increasing from 131 to 132 when using the spellchecked data. This isn't a surprise as people for the most part can correctly type the name of an item, I hypothesize that the more complex terms will see a much higher increase in hits due to the nature of the spelling.
- Overall I believe the spellchecked data is more accurate than the non-spellchecked version, but the magnitude is relatively small. The accuracy is in part due to most misspellings only containing one error and the spellchecker has relatively high accuracy in its spelling preditcion. As stated before most people should be able to spell words at a high level of accuracy so the only search terms that would benefit from spellchecker are edgecase complex ones. One issue with spellchecking is that the spellchecked version may be incorrect due to spelling properties such as spelling potatoe. The plural form of potato is potatoes so it can be assumed the user forgot an s rather than accidently added e to the word.
- The longest runing method is the list_to_freq_dict due to the count() method iterating through each item in the list. This is an O(n^2) operation due to each element in the list iterating through the entire list. If the list is 10x bigger it would take 100 times longer and at 100x it would take 10000 times longer.

References

References

(1) Used the information in this link for removing non alphabet characters. https://stackoverflow.com/questions/43023795/removing-all-numeric-characters-in-a-string-python (https://stackoverflow.com/questions/43023795/removing-all-numeric-characters-in-a-string-python)