Lab 4: Search Terms 2.0 Pandas

Stuart Harley

Introduction: In lab 3 we created 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.

In this lab, we are doing the same thing but we are using numpy and pandas to manipulate the data. This specific notebook uses Pandas.

Learning Outcomes:

- Data importing with Numpy and Pandas
- Cleaning data

Importing Libraries

In [1]:
```
from spellchecker import SpellChecker
import pattern.en
import csv
import time
import sys
import pandas as pd
import re
import numpy as np
pd.options.mode.chained_assignment = None # hides a SettingWithCopy warning la
ter on
```

Function splits tokens at %20's and spaces

In [2]:
```
def split_token(string):
    return re.split('%20|\s|,', string)
```

Importing csv data into a pandas dataframe

In [3]:
```
%%time
original_tokens = pd.read_csv('/home/harleys/searchTerms.csv', usecols=["Searc
hTerm"], dtype=str)
print("bytes in original_tokens: " + str(sys.getsizeof(original_tokens)))
```

```
bytes in original_tokens: 68816387
CPU times: user 643 ms, sys: 28 ms, total: 671 ms
Wall time: 671 ms
```

Creates a new dataframe containing all tokens: (this splits the tokens at spaces, commas, or %20's.)

In [4]:
```
%%time
tokens = original_tokens["SearchTerm"].to_numpy()
all_tokens = pd.DataFrame([elem for singleList in list(map(split_token, tokens
)) for elem in singleList])
all_tokens.columns = ["SearchTerm"]
print("bytes in all_tokens: " + str(sys.getsizeof(all_tokens)))
```

```
bytes in all_tokens: 92425589
CPU times: user 2.54 s, sys: 96.2 ms, total: 2.64 s
Wall time: 2.64 s
```

Adds a column in the dataframe containing the tokens in lower case

In [5]:
```
%%time
all_tokens["Lowercase"] = all_tokens["SearchTerm"].map(lambda token: token.low
er())
print("bytes in lowercase column: " + str(sys.getsizeof(all_tokens["Lowercase"
])))
```

```
bytes in lowercase column: 92382509
CPU times: user 851 ms, sys: 16 ms, total: 867 ms
Wall time: 865 ms
```

Creates a new dataframe containing unique tokens and their number of occurances

In [6]:
```
%%time
unique_tokens = all_tokens["Lowercase"].value_counts().to_frame()
unique_tokens = unique_tokens.reset_index()
unique_tokens.columns = ["SearchTerm", "Occurances"]
print("bytes in unique_tokens dataframe: " + str(sys.getsizeof(unique_tokens
)))
```

```
bytes in unique_tokens dataframe: 4417601
CPU times: user 217 ms, sys: 8.49 ms, total: 226 ms
Wall time: 223 ms
```

Creating an output csv file from the database of the unique tokens and counts

In [7]:
```
unique_tokens.to_csv('/home/harleys/pandas_all_tokens.csv', index=False)
```

Example results of tokens and their number of occurances: In this cell several entries from the unique_tokens dataframe are printed out to illustrate what the csv file looks like.

In [8]:
```
unique_tokens.head()
```

Out[8]:

	SearchTerm	Occurances
0	chicken	19223
1	cream	16056
2	cheese	13955
3	beef	13566
4	pie	11475

Creates a new dataframe containing only the alphabetic tokens and their number of occurances

In [9]:
```
%%time
unique_alpha_tokens = unique_tokens[unique_tokens["SearchTerm"].str.isalpha()]
print("bytes in unique_alpha_tokens dataframe: " + str(sys.getsizeof(unique_al
pha_tokens)))
```

```
bytes in unique_alpha_tokens dataframe: 861331
CPU times: user 24.5 ms, sys: 31 µs, total: 24.6 ms
Wall time: 23.6 ms
```

Spellchecking the alphabetic tokens and adding the possible correct spelling as a new column: This code originally threw a SettingWithCopy warning, but I determined that the code acted as intended so I hid the warning.

In [10]:
```
%%time
spell = SpellChecker(distance=2)
unique_alpha_tokens["CorrectSpelling"] = unique_alpha_tokens["SearchTerm"].map
(lambda token: spell.correction(token))
unique_alpha_tokens = unique_alpha_tokens[["SearchTerm", "CorrectSpelling", "O
ccurances"]]
unique_alpha_tokens = unique_alpha_tokens.reset_index(drop=True)
print("bytes in dataframe columns with tokens and their spellchecked versions:
" + str(sys.getsizeof(unique_alpha_tokens["SearchTerm"]) + sys.getsizeof(uniqu
e_alpha_tokens["CorrectSpelling"])))
```

```
bytes in dataframe columns with tokens and their spellchecked versions: 13741
50
CPU times: user 23min 56s, sys: 27.5 s, total: 24min 23s
Wall time: 24min 24s
```

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

In [11]:
```
unique_alpha_tokens.to_csv('/home/harleys/pandas_tokens_and_correct_spellings.csv', index=False, columns=["SearchTerm", "CorrectSpelling"])
```

Example results of tokens and their possible correct spellings: In this cell several entries from the unique_alpha_tokens dataframe are printed out to illustrate what the csv file looks like.

In [12]:
```
unique_alpha_tokens.iloc[0:5, 0:2]
```

Out[12]:

	SearchTerm	CorrectSpelling
0	chicken	chicken
1	cream	cream
2	cheese	cheese
3	beef	beef
4	pie	pie

Creating a final dataframe of unique spell checked tokens

In [13]:
```
%%time
unique_correct_spelling_tokens = pd.DataFrame(unique_alpha_tokens["CorrectSpelling"].unique())
unique_correct_spelling_tokens.columns = ["SearchTerm"]
print("bytes in unique_correct_spelling_tokens: " + str(sys.getsizeof(unique_correct_spelling_tokens)))
```

```
bytes in unique_correct_spelling_tokens: 536352
CPU times: user 10.6 ms, sys: 50 µs, total: 10.7 ms
Wall time: 10.1 ms
```

Creating a final dictionary of tokens: In this cell, the final dictionary of spell 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 [14]:
```
%%time
temp_df = unique_alpha_tokens.drop(columns=["SearchTerm"])
final_spelled_dict = {}
row_index = 0
for token in unique_correct_spelling_tokens["SearchTerm"]:
    final_spelled_dict[token] = 0;
for token in temp_df["CorrectSpelling"]:
    final_spelled_dict[token] += temp_df.loc[row_index, "Occurances"]
    row_index += 1
print("bytes in final_spelled_dict: " + str(sys.getsizeof(final_spelled_dict
)))
```

```
bytes in final_spelled_dict: 295008
CPU times: user 76.1 ms, sys: 49 µs, total: 76.1 ms
Wall time: 74.7 ms
```

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

In [15]:
```
with open('/home/harleys/pandas_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]))
```

Conclusion

- In terms of space complexity, it appears that the numpy and pandas data structures use more bytes than simple lists and dictionaries. I believe this is because the numpy and pandas structures keep track of more information than a regular list. Also, numpy is not optimized for strings.
- In terms of time complexity, the numpy and pandas structures took longer than the regular structures. I believe this is because numpy is optimized for numbers, not strings. And they are both not really designed to change the lengths of data structures. In terms of the spellchecking cell, they all took about the same amount of time to run.
- In terms of performance, I think that it could be effective to use numpy for a set length of numbers. And to use pandas for a set length of strings or numbers where you want to operate on these datasets without really changing the length much.
- In terms of usability, it can be annoying to do things in pandas and numpy. They have very powerful methods which if you understand them well, can definitely help a lot. However, if you are new to them, it can be very difficult to get the output you are looking for especially because many of the methods have a large number of optional parameters which can be confusing to understand.