Lab 3: Search Terms with Pandas

Submitted By: Julian Singkham
Date: 01/07/2021

Abstract

The purpose of this lab is to familize ourselves with using panda data frames and functional programming for 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 compare the dictionary and list approach of frequency counts to the frequency counts generated by data frame value_counts().

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

Parameters

In [1]:

```
from spellchecker import SpellChecker
import pattern.en
import csv
import pandas as pd
import sys

freq_dict = {}
freq_dict_spellchecked = []
```

Functions

Imports a CSV file and creates a dataframe of the first item of each row. Additionally removes web spaces and splits search terms by space.
I.E "Spicy Bacon" would be "Spicy", "Bacon"

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

In [2]:

```
def import_csv_df_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
    file.closed
    remove_web_spaces_list = [str(row[0]).replace("%20", " ") for row in temp

    split_on_space_list = []
    for item in remove_web_spaces_list:
        split_on_space_list.extend(item.split(" "))

    df = pd.DataFrame({"Raw Data" : split_on_space_list})
    return df
```

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 [3]:

```
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 [4]:

```
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 [5]:

```python
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 [6]:

```python
def spellcheck_token(token):
    fixed_token = ''
    if(token != ''):
        fixed_token = spellcheck_dict[token]
    return fixed_token
```

Main

In [7]:

```python
# Import csv to search term data frame
df = import_csv_df_first_col("searchTerms.csv")

# This section of code filters the data from the dataset by removing non-alph
%time df["Removed Numbers"] = df["Raw Data"].str.replace('[0-9]', '')
%time df["Alphabet Only"] = df["Removed Numbers"].str.replace('[^a-zA-Z]', ''
```

```
Wall time: 8 ms
Wall time: 8 ms
```

In [8]:

```
%%time
# This section of code spellchecks the dataset
spellcheck_dict = spellcheck_dict_init(df["Alphabet Only"].tolist())

df["Spellchecked"] = df["Alphabet Only"].map(lambda token: spellcheck_token(t
df.head(10)
```

```
Wall time: 903 ms
```

Out[8]:

	Raw Data	Removed Numbers	Alphabet Only	Spellchecked
0	36969			
1	CMED	CMED	CMED	med
2	500100			
3	KEND	KEND	KEND	end
4	5750			
5	CMED	CMED	CMED	med
6	980228			
7	DYNC1815H	DYNCH	DYNCH	lynch
8	DYND70642	DYND	DYND	dyed
9	DEES	DEES	DEES	DEES

In [9]:

```
%%time
# Thia section of code benchmarks the time it takes for the dictionary and Li
# sorted frequency list of search terms
spellcheck_freq_dict = list_to_freq_dict(df["Spellchecked"].tolist())
spellcheck_freq_list = sort_freq_dict(spellcheck_freq_dict)
```

```
Wall time: 2.14 s
```

In [10]:

```
# Thia section of code benchmarks the time it takes for the data frame approa
# sorted frequency list of search terms
%time series_freq = df["Spellchecked"].value_counts(dropna=True)
series_freq.head(10)
```

```
Wall time: 3.52 ms
```

Out[10]:

```
           3436
bacon       459
milk        185
chicken     183
beef        145
juice       139
banana      119
sausage     119
creamer     113
cheese      110
Name: Spellchecked, dtype: int64
```

In [11]:

```
# This section of code benchmarks the size of the sorted frequency data frame
series_freq.memory_usage(deep=True)
```

Out[11]:

```
99242
```

In [12]:

```
# This section of code benchmarks the size of the sorted frequency list
sys.getsizeof(spellcheck_freq_list)
```

Out[12]:

```
11512
```

Conclusion

- The time it took to remove numbers from the dataset took 16ms and 8ms to remove special characters. The difference in times is attributed to the fact that the dataset used for the second operation is much smaller than the raw dataset. This difference in size allows for faster runtime. The two filtering functions can be done in one command which would take the same amount of time (16ms) as the first filtering function. When compared to the list method (13ms), the time difference between the two is marginal at 3ms which can be attributed to the random nature of time quantum given to threads.
- The time it took to spellcheck the dataset took 954ms while the list method took 795ms. This large difference in time is due to the fact that each column of the dataframe must have the same length, meaning empty values are kept. The dataset given to the list version does not contain empty values. I suspect a separate spellchecked data frame would compute faster than the list version due to the map function with lambda being vectorized.
- The largest difference in timing can be found in the approach to search term frequency. The list method took 1.9s while the data frame approach took 12.5ms. This dramatic difference in timing is due to the highly optimization nature of the pandas library.
- The data frame is about 9x larger at 99,242B when compared to teh 11,512B of the list method. This difference is reasonable as the dataframe data structure contains additional information related to Pandas features, while lists contain raw data.
- Overall the pandas approach used a lot less code due to using functional programming when compared to the list approach. Pandas performance degrades with empty cells and can be slower than a list approach due to the sheer difference in dataset size. One of pandas greatest disadvantage is that each series must be of the same data type, unlike a list which can have muliple data types in rows/cols.