Introduction to my Mobivity Subway counter project

Kaiwen Sun (skw_kevin@126.com)

1. Overview

There are 286999 sandwiches, 137265 drinks. The total length of sandwiches are 222993.75 feet

The project is implemented using Apache Spark's Python API (PySpark). I chose Spark because receipt parsing is an application that can perfectly fit to parallel computation. One receipt doesn't not affect another. Spark is a distributed system handling this kind of application very well.

Python Regular Expression is heavily used in this program.

There are mainly 5 steps to achieve the final output:

a) Read from json file to get receipts' text.

b) Find areas of sold items in valid receipts.

c) Category items into 3 classes: sandwiches, drinks and others.

d) Determine sandwich's size and item's quantity.

e) Sum up number of sandwiches, drinks, and sandwich sizes.

There are two important source code files, countitems.ipynb and regexpfeatures.py . If the .ipynb file is not readable without software like Python Notebook / Jupyter, countitems.py is also provided. Countitems.ipynb has the main entrance of the program; regexpfeatures.py describes regular expressions used in the program.

The following three sections are going to describe how to decide valid area of items in a receipt, how to classify items, and how to decide sandwich's size and item's quantity.

2. Decide Valid Area of Items in a Receipt

I first printed out the first 100 receipts, the last 100 receipts and 100 randomly picked receipts. It is observed that items in a valid receipt always followed after a line in either one of the four:

a) "Qty\s*Size\s*Item\s*Price",

b) "---\s*ITEM\s*---\s*QTY\s*PRICE\s*MEMO\s*PLU",

c) "QTY\s*ITEM\s*TOTAL",

d) "Qty\s*Size\s*Item"

and always ended before a line containing "Total" or "!aEa- !a". These features are used to decide valid area of items in a receipt. Several

receipts, such as a “Cash Drop Report” and crashed receipts, are not valid receipts. There are very few invalid receipts, and their features are collected in order to ignore those receipts at run time.

Related Python Regular Expression objects are:

a) re_startItems = re.compile("|".join(possible_start))

b) re_endItems = re.compile("|".join(possible_end),re.IGNORECASE)

c) re_ignore = re.compile("|".join(possible_ignore),re.IGNORECASE)

3. **Classify Items into Categories**

a) **Sandwiches**

A line of sandwich item is a line containing “Sub”(not followed by “way”), “FlatBd” (flatbread sandwich) and such, “SOTD” (Sub of the day) and such, key words describing sandwich’s size (eg. “FtLong”), “6\””, “12\””, “BoxLn” (Boxed Lunch including a sandwich), and several special cases where the line is pruned due to limited width of receipt. Adding/extra/toppings to a sandwich should be excluded, even if it matches the feature above.

b) **Drinks**

A line of drink item is a line containing words like “Drink”, “Fnt” (fountain), “Coffee”, “Juice”, “Bottle”, “BtlWtr”, “Lemonade”, etc. Also, a line containing “oz” (unit of liquid) after a number is also considered as a drink. But soup is not a drink, even if “oz” is found.

c) **Others**

The regular expression of other items is used to collect known other items, in order to help me to find unconsidered sandwich / drink items. Items that are not categorized to any of sandwich, drink and other, are printed out to stderr. So unconsidered sandwich / drink items can be easily figured out.

Related Python Regular Expression objects are:

a) re_sandwich = re.compile("|".join(possible_sandwich),re.IGNORECASE)

b) re_drink = re.compile("|".join(possible_drink),re.IGNORECASE)

c) re_otherkind = re.compile("|".join(possible_otherkind),re.IGNORECASE)

4. **Decide Sandwich’s Size and Quantity**

a) **Sandwich’s size**

It is interesting that in addition to 6in and 12in sandwiches, there are also Mini sandwiches which are 3in long. Once an item is decided to be a sandwich, I use “Mini”, “6\””, “12\” and keywords indicating sandwich

length (eg. “FtLng”, “6 Inch”, “ FT “) to decide sandwich’s size. If the size if not specified, assume it is a foot-long sandwich.

b) **Quantity**

   i. Observing receipt samples, I found that the quantity is always an integer number surrounded by white-space at two sides.

   ii. If there are no such integer, use guessQty() function to choose a reasonable quantity from 1 to 10.

   iii. In rare cases, the item name also contains some integers surrounded by white-spaces (eg. “Any 6 Inch Sub”). Then there will be multiple such integers in a line. In these cases, use guessQty() function to choose the most reasonable quantity among those candidates.

c) **guessQty()**

   This function guesses the quantity of ambiguous item. The guess is based the category of the item, the total price of the item, and a reasonable range of unit price of this category of item. This function chooses one from the possible quantity from parameter qtylst such that the unit price of that category of item is reasonable.

Related Python Regular Expression objects are:

a) re_6insandwich = re.compile("|".join(possible_6insandwich))

b) re_12insandwich = re.compile("|".join(possible_12insandwich))

c) re_minisandwich = re.compile("|".join(possible_minisandwich))

d) re_itemQty = re.compile("|".join(possible_itemQty))

5. Result and Observations

There are 286999 sandwiches, 137265 drinks. The total length of sandwiches are 222993.75 feet

With 250000 receipts, it is hard to prove the result is correct. But the result is very reasonable. On average: each customer purchases a little more than one sandwich; about half of customers order a drink; the total length could be a interpenetrated as half of customers purchasing foot-long and the other half of customers purchasing half-long sandwiches (222993.75/286999≈3/4). All of these observations do not conflict with common sense.