Documentation

DataStream Script

By

Muhammad Ali Hashmi

&

Markus Luczak-Roesch

CONTENTS

1 Introduction

This script is to process the data from Wikipedia editing streams or Twitter hashtags to calculate its specificity, diversity, entropy, and pielou. It accepts the data as a CSV file which should be in the following format:

“ID”;”Time Stamp”;”Identifiers”

Ideally the “ID” is a number, “Time Stamp” is in "2017-12-01 00:19:02" format, and “Identifiers” can be empty string or multiple identifiers separated by a comma.

2 Running the script

Running the script is very simple. It takes the input file as first argument. The script may be called with python2.x or 3.x. For example on a terminal in mac or Linux:

```
python script.py filename.csv
```

2.1 Packages required to run the script

The following packages are essential for the script to run and must be installed on the system and accessible to the user running the script.

- csv
- sys
- time
- collections
- math
- multiprocessing (required for parallel processing if you want it)

3 Incremental Entropy Calculation

Calculating entropy from scratch at every step becomes impractical for a large data so the incremental entropy calculation approach has been adopted. For that the code has been kindly provided by Blaz Sovdat.

The basic structure of the script is that it reads the file line by line and only stores current line in the memory to make it efficient. It does the analysis on current line and stores the essential data required for next calculations in memory. It writes everything else directly onto the disk to output files.

4 DataProcessor Function

This function is the backbone of the script which is responsible for all the analysis. It takes the input file (*InFile*) as an argument. Here is a step by step breakdown of the function.

4.1 Output CSV Files

The function starts by opening three csv files to write the results into. One is for nodes, one for links, and one for writing the roots.

Then the csv format is defined as follows:

```
node_writer = csv.writer(nodes_csv, quotechar='"', delimiter=';', quoting=csv.QUOTE_ALL, skipinitialspace=True)
```

Here the Quotes or delimiters of the csv files can be changes as desired. In the above example, **;** is used as delimiter and **"** as quotes.

4.2 Defining Variables Values

With opening the input file provided, certain variables are defined to hold their current value. For example:

```
with open(InFile) as f:
      line = f.readline()
      diversity = -1
```

The variable "*line*" stores current line in the memory to work with the value of diversity is set to be -1 at start. It works by adding +1 to the current diversity value. So this initial value helps to start the first value of diversity with zero.

4.3 Formatting current line

```
while line:
      current_line = [item.strip(',\n\r') for item in line.split(';')] # For Twitter Data change to ('",\n\r').
      line = f.readline()
      if current_line[2] != "":
            current_line_sorted = [current_line[0], current_line[1],','.join(map(str,
      sorted((current_line[2]).split(","))))]
```

This "*while loop*" makes sure to work on the current line until all the lines in the file are finished. "*current_line*" splits the current line from the semicolon (;) and stores it as a list. The next "if statement" checks is the identifiers list is not empty then it proceeds forward, otherwise for nodes without identifiers "else statement" reads this line and makes its specificity =0 and diversity =0. They get the same entropy, evenness and richness value as the node before. The next step is to sort the identifiers alphabetically. So the variable "*current_line_sorted*" does that job. It converts the identifiers to a list, sorts them, & converts them back to a string.

4.4 Analysis of current line and entropy calculation

The next step is to do the analysis on "*current_line_sorted*". The "*for loop*" below converts the 3rd item of "*current_line_sorted*", i.e. the identifiers string to a list, loops through all the identifiers and checks if the identifier is present in the "*links_dictionary*". If yes, then it updates the dictionary ID with the current identifier, if not (*else statement*), it adds the identifier to the dictionary with its ID. It also updates its entropy at the start of the loop by using incremental entropy method described above.

```
      for identifier in current_line_sorted[2].split(','):
            entropy.update([CountChange(identifier, 1)])
            if identifier not in links_dict:
                  links_dict[identifier] = current_line_sorted[0]
                  roots_writer.writerow([current_line_sorted[0], identifier])
            else:
                  links_writer.writerow([links_dict[identifier], current_line_sorted[0], identifier])
                  links_dict[identifier] = current_line_sorted[0]  # Update identifier's ID in links_dict
```

4.5 Calculation of Pielou

Pielou (*J*) is calculated after the above loop finishes. Current value of the entropy (H′) is used for pielou calculation using the following formula:

$$J = \frac{\mathrm{H}'}{\ln(\mathrm{S})}$$

Where S is the number of total species (unique). The length of "*links_dictionary*" gives the value of S. So the script line for Pielou calculation reads as:

```
pielou = '{:.3f}'.format(float(entropy.entropy()) / log2(len(links_dict)))
```

4.6 Calculation of specificity and diversity

The next "*for loop*" shown below is for taking care of specificity and diversity. It adds all the identifiers from current line (as a string) to the "*identifiers_seen*" dictionary with default specificity 0 and the current value of diversity. * In next line, if the identifiers string is already present in the "*identifiers_seen*" dictionary, its value (specificity) is incremented by 1. When a new identifier string is observed, its diversity is also incremented by 1.

```
if current_line_sorted[2] not in identifiers_seen:
      diversity += 1  # Upon seeing a new identifier set, increment diversity by 1
      identifiers_seen[current_line_sorted[2]] = [0, diversity] # * Add identifier set to dict.
      div = diversity  # div value used so that 'diversity' value does not change again and again
else:  # If identifier already in identifiers_seen dictionary
      identifiers_seen[current_line_sorted[2]][0] += 1  # Increment its value (specificity) by 1
      div = identifiers_seen[current_line_sorted[2]][1]  # Pick diversity from identifiers_seen.
```

4.7 Writing Nodes

The last step in this function is to write the nodes. “*nodes_list*” is a list that contains the information in the following order:

Node ID; *TimeStamp*; *Identifiers*; *Specificity*; *Diversity*; *Entropy*; *Pielou*

```
nodes_list = [current_line_sorted[0], current_line_sorted[1], current_line_sorted[2],
              identifiers_seen[current_line_sorted[2]][0], div, '{:.3f}'.format(entropy.entropy()), pielou]
node_writer.writerow(nodes_list)  # Write the row to nodes csv file
```

5 Dictionaries and Variables Definition

As described above for using various dictionaries and variables, this is the place to define them. Following are the dictionaries and variables and their use:

- `entropy = EntropyHolder()` This is a class to hold the current entropy.
- `links_dict = {}` This is a dictionary to keep the identifiers and the ID of their last occurrence
- `identifiers_seen = {}` It is a dictionary having already seen identifiers with their [specificity, diversity] values as keys

6 Main Part of the Program

“*InFile*” is a variable that stores the link to the input file. It can be explicitly defined with the path to the input file or it can accept the file as first argument in command line. The variable “*FileName*” contains the name of input file without the extension to use elsewhere.

Now the function “*DataProcessor*” can be called with one processor or in parallel. Below line uses it with one processor. By default, this line is commented. If one processor is desirable, this line can be uncommented and the next lines for parallel processing can be commented.

```
DataProcessor(InFile)
```

Below three lines are for using the function in parallel for the analysis. If using multiprocessing, the above line must be commented out.

```
cpu_count = multiprocessing.cpu_count()  # Check the number of physical processors available
pool = multiprocessing.Pool(processes=cpu_count)  # Use all the available processors
pool.apply_async(DataProcessor(InFile))
```

The variable “*cpu_count*” counts the physical processors available on the machine. Then “*pool*” variable calls the multiprocessing module with the number of available processors on the machine running the script.

Below are the variables to record finish time and date to monitor the total run time of script

```
finish_time = time.strftime("%H:%M:%S")
finish_date = time.strftime("%d/%m/%Y")
time_file = sys.stdout
```

Finally, the below portion writes the log to an output file. This section is self-explanatory.

```
# ====================================================================================================== #
# Write the start and finish times and elapsed time to the file "Script-Report.csv"                      #
sys.stdout = open(FileName + '-Script-Report.csv', 'w')                                                  #
print 'The script started running at ', start_time, 'on', start_date                                     #
print 'The script finished processing at', finish_time, 'on', finish_date                                #
print 'The script used', cpu_count, 'CPU processors.'                                                    #
print 'Memory information printed below:'                                                                #
print 'The size of links_dict is: ', (sys.getsizeof(links_dict)) / 1000000, 'MB'                         #
print 'The size of identifiers_watched is: ', (sys.getsizeof(identifiers_seen)) / 1000000, 'MB'          #
print 'Total processing time was', "{0:.2f}".format(time.time() - start), 'seconds.'                     #
sys.stdout.close()                                                                                       #
sys.stdout = time_file                                                                                   #
# ====================================================================================================== #
```