10/12/2018

FIT 9133

Building a Child Language Analyser Semester 2 2018

Name: Nikhil Braganza

Student ID: 29977770

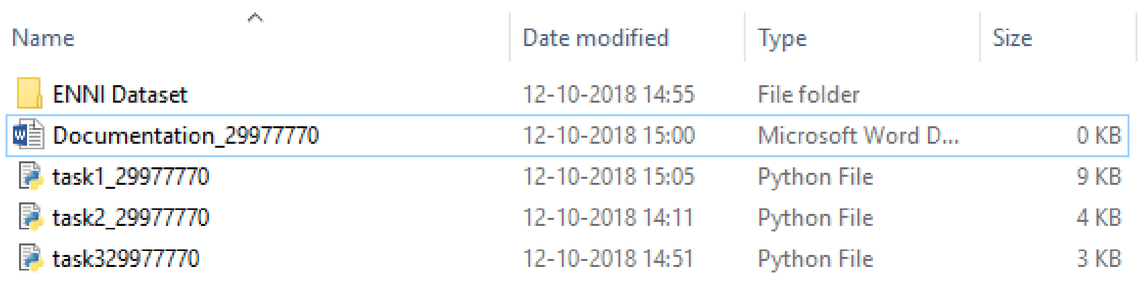
MONASH UNIVERSITY

Table of Contents

Requirements

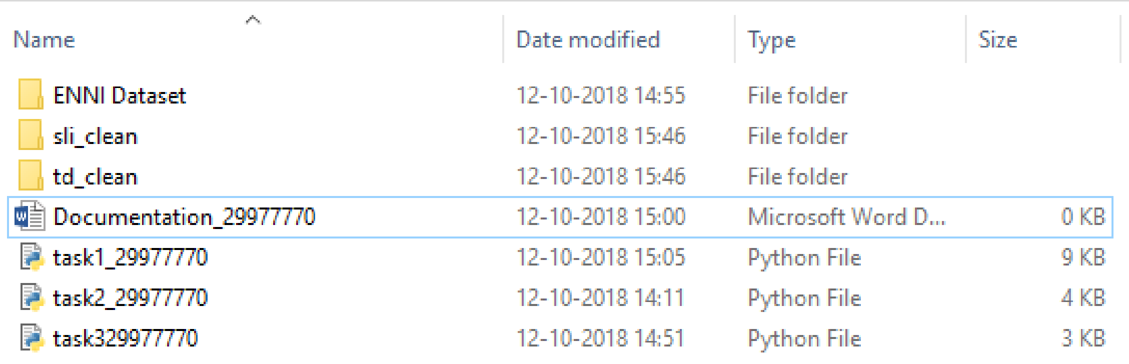
The requirements for running the files in this assignment are as follows:

Each of the three files namely task1_29977770.py, task2_29977770.py, task3_29977770.py should be in the same directory. Also the ENNI Dataset that is to be filtered and analysed should be present in the same directory. It should be named as "ENNI Dataset", and it should further contain two folders namely "SLI" and "TD" which contains all the respective files. To avoid confusion a picture showing the arrangement has been attached below.



Name	Date modified	Type	Size
ENNI Dataset	12-10-2018 14:55	File folder	
Documentation_29977770	12-10-2018 15:00	Microsoft Word D...	0 KB
task1_29977770	12-10-2018 15:05	Python File	9 KB
task2_29977770	12-10-2018 14:11	Python File	4 KB
task329977770	12-10-2018 14:51	Python File	3 KB

After the execution the same folder will have two additional folders with the clean files in them as follows.

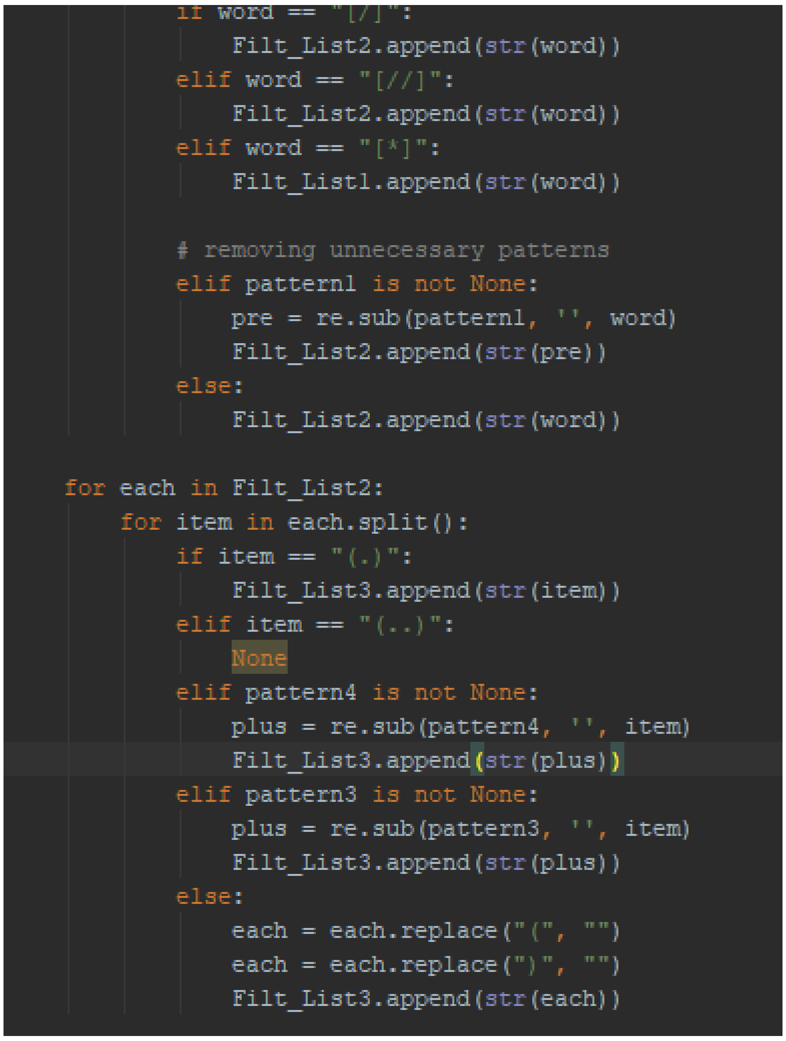


Name	Date modified	Type	Size
ENNI Dataset	12-10-2018 14:55	File folder	
sli_clean	12-10-2018 15:46	File folder	
td_clean	12-10-2018 15:46	File folder	
Documentation_29977770	12-10-2018 15:00	Microsoft Word D...	0 KB
task1_29977770	12-10-2018 15:05	Python File	9 KB
task2_29977770	12-10-2018 14:11	Python File	4 KB
task329977770	12-10-2018 14:51	Python File	3 KB

Algorithm

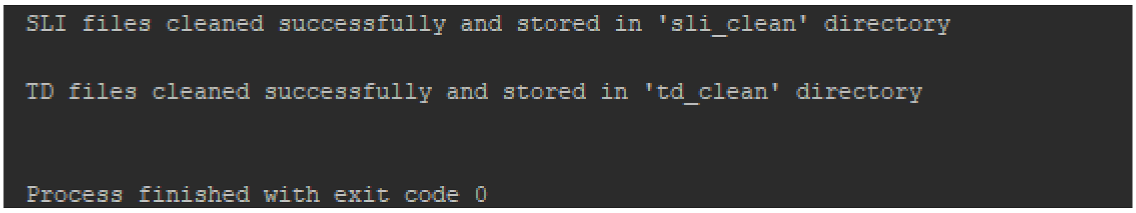
Task 1:

All the required and variables are initialized. All the required lines are extracted by the use of flag and the if-else loop. The use of regular expressions has been done in this assignment. Regular expressions helps to find a pattern of characters or digits in a long file. We have identified some of the patterns that are not required and then removed it from the file. And the required words in each line are appended in a list. Since there are multiple unwanted patterns, use of multiple lists and loops are required, some of them are nested within one another. A snippet of the code is shown below.



```
            if word == "[/]":
                Filt_List2.append(str(word))
            elif word == "[//]":
                Filt_List2.append(str(word))
            elif word == "[*]":
                Filt_List1.append(str(word))

            # removing unnecessary patterns
            elif pattern1 is not None:
                pre = re.sub(pattern1, '', word)
                Filt_List2.append(str(pre))
            else:
                Filt_List2.append(str(word))

    for each in Filt_List2:
        for item in each.split():
            if item == "(.)":
                Filt_List3.append(str(item))
            elif item == "(..)":
                None
            elif pattern4 is not None:
                plus = re.sub(pattern4, '', item)
                Filt_List3.append(str(plus))
            elif pattern3 is not None:
                plus = re.sub(pattern3, '', item)
                Filt_List3.append(str(plus))
            else:
                each = each.replace("(", "")
                each = each.replace(")", "")
                Filt_List3.append(str(each))
```

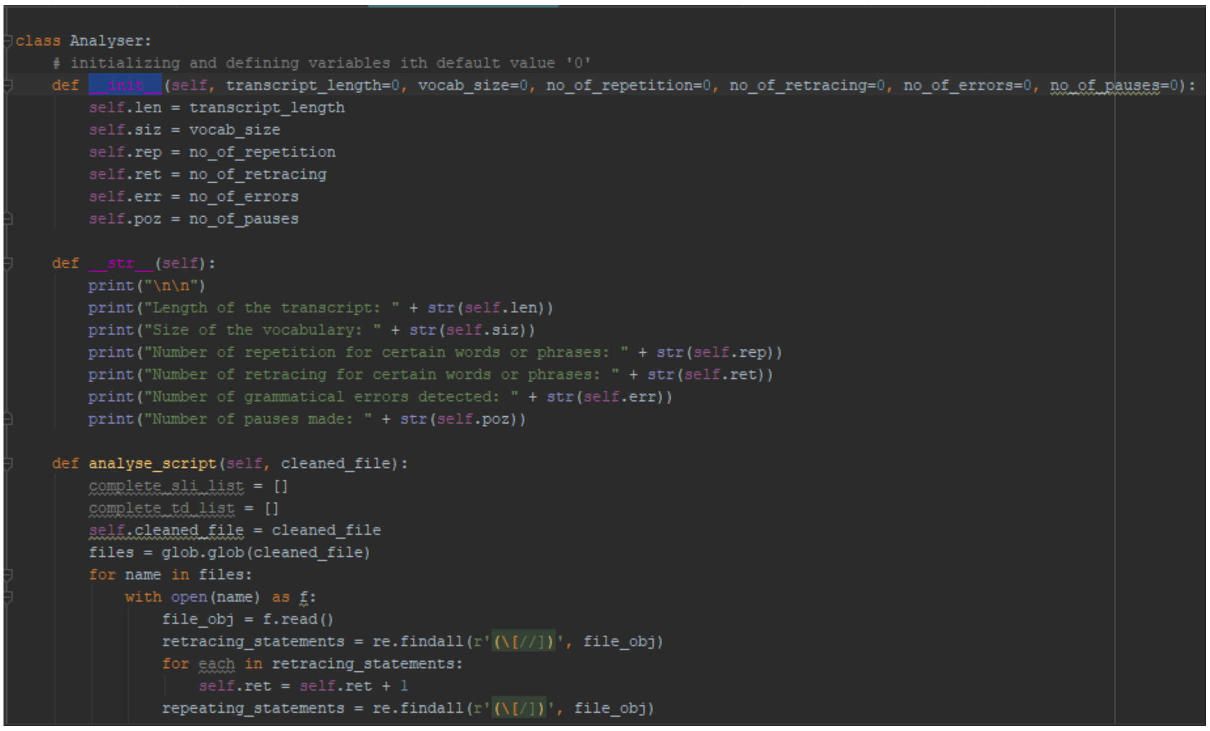
The whole code for filtering is then defined as two functions, one for the SLI files and the other for the TD files. The filtered files are then created and stored in the ‘sli_cleaned’ and ‘td_cleaned’ directories respectively. After the cleaning of each sets of files, we get the following output in the console.



```
SLI files cleaned successfully and stored in 'sli_clean' directory

TD files cleaned successfully and stored in 'td_clean' directory


Process finished with exit code 0
```

Task 2

Task 2 can only be run after completion of task 1. A class named Analyser has been defined with the methods to initialize, overload and count the number of each statistics. A snippet of the code is given below.



```python
class Analyser:
    # initializing and defining variables ith default value '0'
    def __init__(self, transcript_length=0, vocab_size=0, no_of_repetition=0, no_of_retracing=0, no_of_errors=0, no_of_pauses=0):
        self.len = transcript_length
        self.siz = vocab_size
        self.rep = no_of_repetition
        self.ret = no_of_retracing
        self.err = no_of_errors
        self.poz = no_of_pauses

    def __str__(self):
        print("\n\n")
        print("Length of the transcript: " + str(self.len))
        print("Size of the vocabulary: " + str(self.siz))
        print("Number of repetition for certain words or phrases: " + str(self.rep))
        print("Number of retracing for certain words or phrases: " + str(self.ret))
        print("Number of grammatical errors detected: " + str(self.err))
        print("Number of pauses made: " + str(self.poz))

    def analyse_script(self, cleaned_file):
        complete_sli_list = []
        complete_td_list = []
        self.cleaned_file = cleaned_file
        files = glob.glob(cleaned_file)
        for name in files:
            with open(name) as f:
                file_obj = f.read()
                retracing_statements = re.findall(r'(\[//])', file_obj)
                for each in retracing_statements:
                    self.ret = self.ret + 1
                repeating_statements = re.findall(r'(\[/])', file_obj)
```

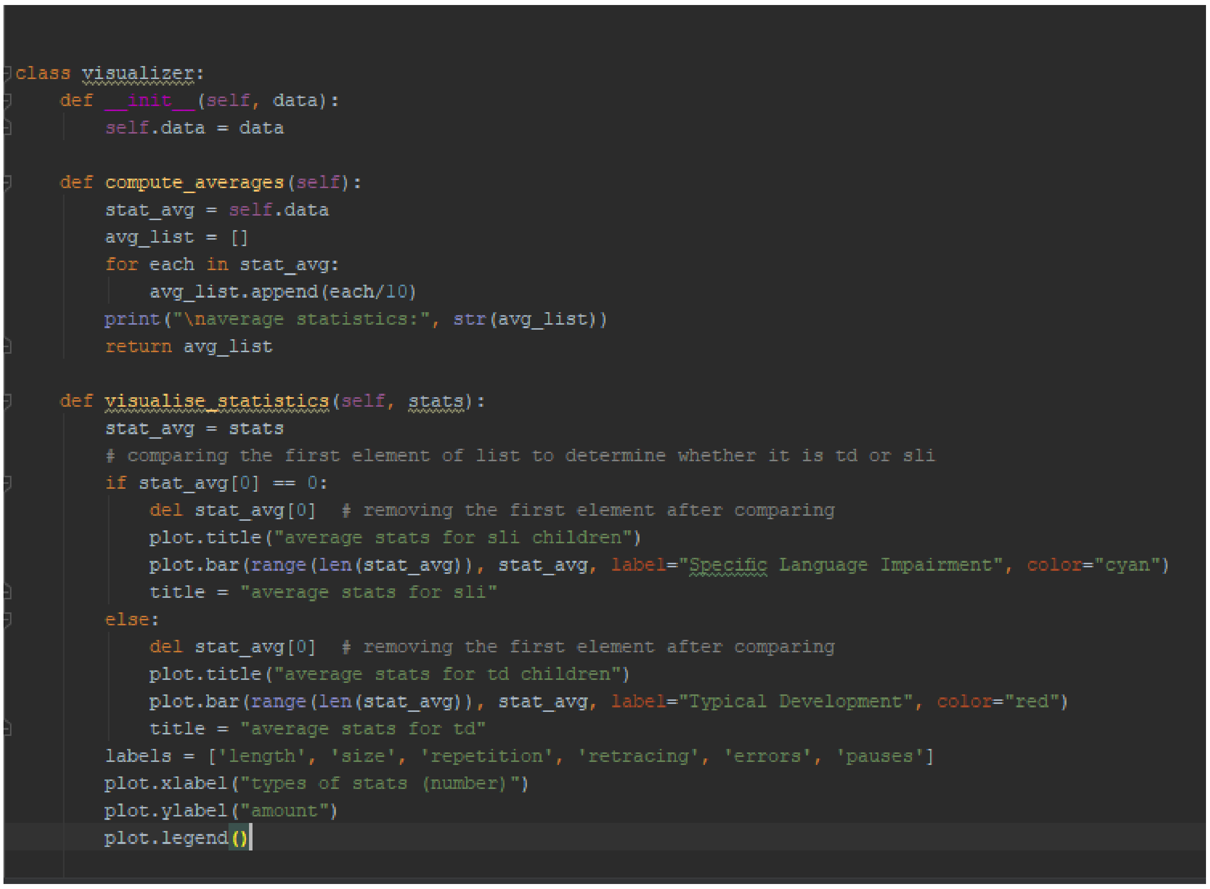
Two objects are created for each group SLI and TD respectively for calculating the various statistics.

(Note: an extra element has been added in the final statistics list to identify the group of the stats)

Task 3

The use of this file is mainly to plot the bar graph according to the statistics that is calculated in the task 2 by using the matplotlib. This file has visualiser class with two functions, one is for calculating the average stats for a group of data. And the other is to plot the graph.

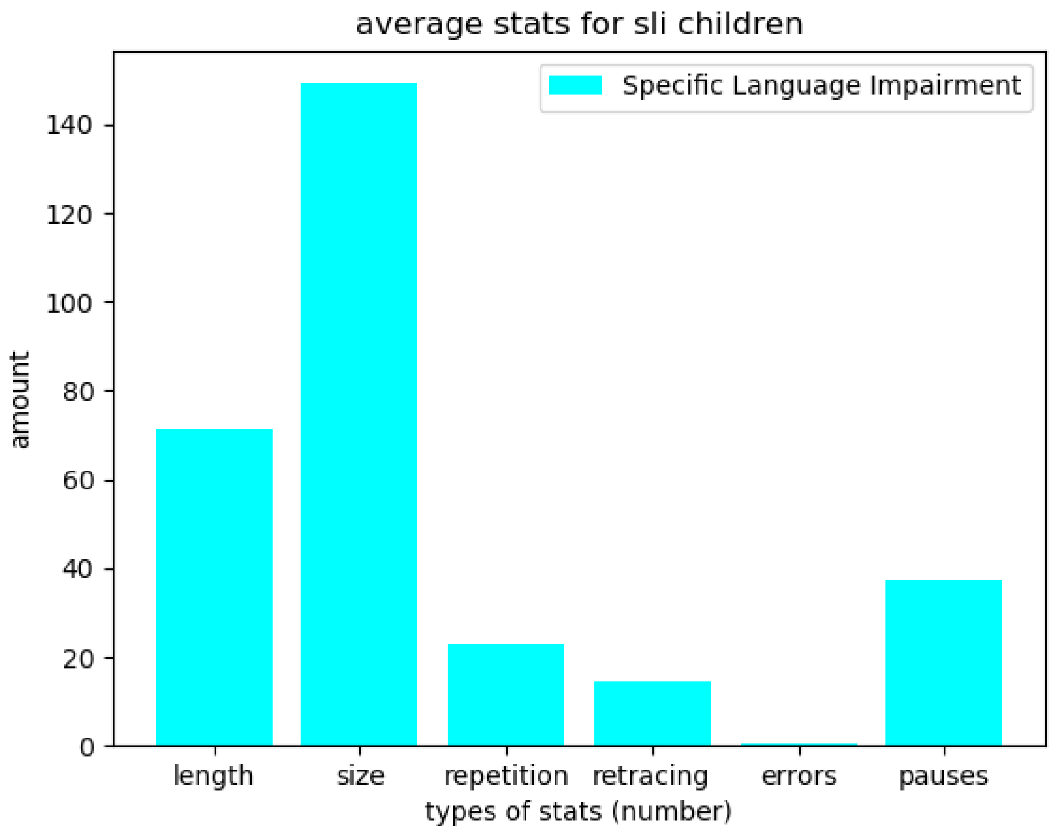
We can see the classes and the defined functions in the code below :

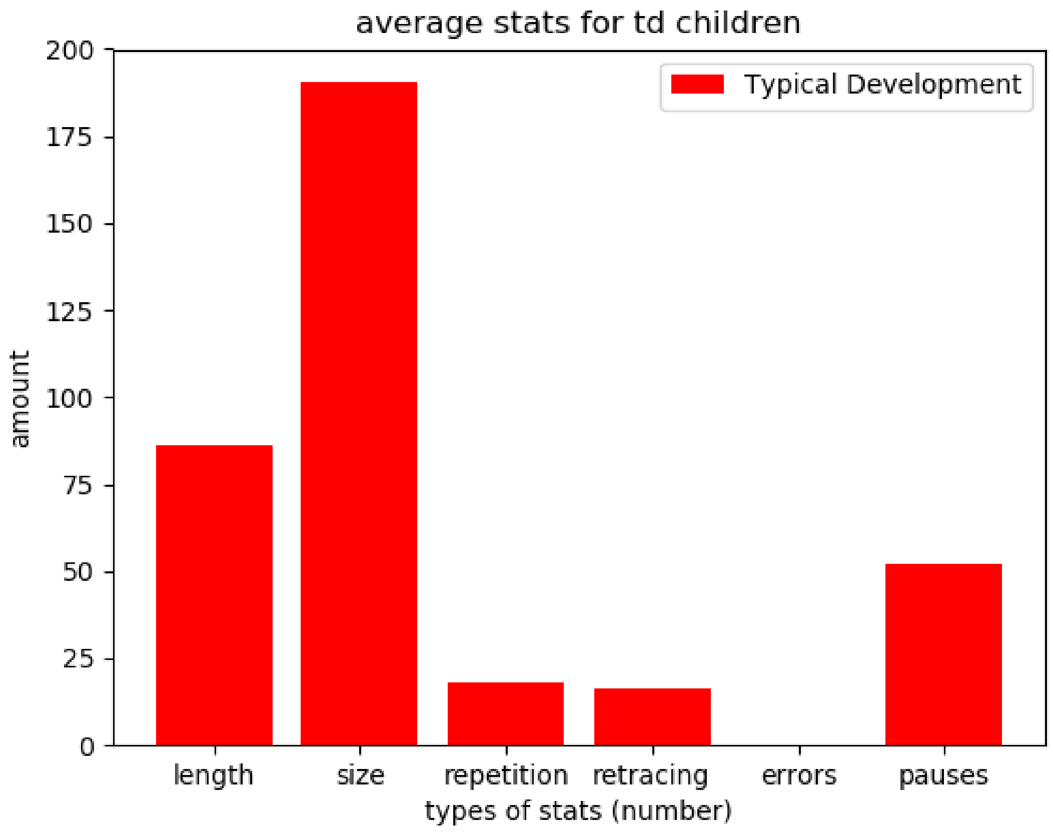


```python
class visualizer:
    def __init__(self, data):
        self.data = data

    def compute_averages(self):
        stat_avg = self.data
        avg_list = []
        for each in stat_avg:
            avg_list.append(each/10)
        print("\naverage statistics:", str(avg_list))
        return avg_list

    def visualise_statistics(self, stats):
        stat_avg = stats
        # comparing the first element of list to determine whether it is td or sli
        if stat_avg[0] == 0:
            del stat_avg[0]  # removing the first element after comparing
            plot.title("average stats for sli children")
            plot.bar(range(len(stat_avg)), stat_avg, label="Specific Language Impairment", color="cyan")
            title = "average stats for sli"
        else:
            del stat_avg[0]  # removing the first element after comparing
            plot.title("average stats for td children")
            plot.bar(range(len(stat_avg)), stat_avg, label="Typical Development", color="red")
            title = "average stats for td"
        labels = ['length', 'size', 'repetition', 'retracing', 'errors', 'pauses']
        plot.xlabel("types of stats (number)")
        plot.ylabel("amount")
        plot.legend()
```

Output

After running of the task three, two graphs will pop up showing the average statistics of the two children groups as follows:


average stats for sli children
Specific Language Impairment
amount
0
20
40
60
80
100
120
140
length
size
repetition
retracing
errors
pauses
types of stats (number)


average stats for td children
Typical Development
amount
0
25
50
75
100
125
150
175
200
length
size
repetition
retracing
errors
pauses
types of stats (number)