Credit Hours System

CMPN402

Cairo University

Faculty of Engineering

“Report”

Name: Mostafa Assem Anwar Kotb

ID: 1170223

Problem Number: Problem 2 (22.1)

Table of contents

Code

1- Main function is responsible for choosing the mode for the user as following:

*Modes of running:

a. User mode: Just taking the input from testcases.txt file and after running the program write the output in out.txt (no performance measuring)
b. Performance measurement: Runs the program on the input from testcases.txt and compute the average F1 score of the testcases, and the execution time of the program and write these numbers in performance.txt
c. Plotting of F1 score: Runs the program on the input from testcases.txt and compute only the average F1 score on different sizes of testcases (divide the input into 600 sample) and plot the output in output.png

2- User mode function "userMode(p, inFile, outFile)": it is responsible for iterating on the test cases in testcases.txt file and calling segment function which is doing the preprocessing function and the segmentation function on the input text.

3- Text preprocessing function is used to lower case the input text, remove the special characters, and remove white spaces in case of running the program but in case of preparing the input to measure the performance, it keeps the spaces for the expected output. e.g. "I am Mostafa" will be "iammostafa" for the input to the algorithm and will be "i am mostafa" for the true value in measuring the performance.

4- Veterbi algorithm function will be explained in algorithm explaining section

5- Performance measure function is used to run the above steps in addition to monitoring the execution time and calculating the average F1 score.

6- Performance plotting function is the same but calculates the average F1 score for incremental number of test cases by dividing the input to 600 samples and run the algorithm on them.

7- DataSetCreator file is used to convert any txt file to dataset json file which contains the word as a key and the probability of this word as a value.

8- TxtToJsonConv file is used to convert any dataset from .txt file to .json file because it is easier in the needed calculations.

9- For the dataset: 1/3 million most frequent words are included in count_1w.json which have each word with its count (frequency). All the words are lowercase and unigram only.

How to run

- Requirements:

```
matplotlib==3.2.2
seaborn==0.10.1
scikit_learn==0.24.2
```

- To install:
    1- pip install matplotlib
    2- pip install seaborn
    3- pip install -U scikit-learn

- Steps of running:
    1- Open cmd in file directory
    2- Run: python main.py
    3- Selected the desired mode (explained in the above section)

*If you want to change the input for the program, just write the input in testcases.txt

Textual answers to the problem

The main problem is to extract from a sentence with no spaces a meaning full sentence. So, at the beginning we need to prepare the input for the program by preprocessing this sentence. In preprocessing we need to remove the special characters like (@, #, $, etc.), we also need to lower case all the characters for easier matching. The second step is to apply Viterbi algorithm which is explained in the next section. The final step is to evaluate the output, this is done using F1 score which is depending on the precision and recall.

Precision: $\frac{\#\ of\ correct\ results}{\#\ of\ all\ returned\ results}$

Recall: $\frac{\#\ of\ correct\ results}{\#\ of\ all\ true\ results\ expected}$

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

In my code I used two F1 score functions the first one which is a modification on the real F1 score calculation, I calculate how much is the output is correct by matching the two lists even if they are not equal in size. For example, “this is a cat” and “this is acat” have a percentage of correctness so I take this matching into consideration. But the second F1 score is a library I used only to support my program as a second measurement (if the two lists are not equal in size it returns zero). The first one is preferred because it is less strict than the second one.

Algorithm explaining

The used algorithm is Viterbi algorithm implemented in the pseudo code below. The main concept for this algorithm is to get the best words with their probabilities by forming a path from the first character to the last character. By taking all sub strings from the left to right and get the best probability of sub right part. For example, “iammostafa” by taking “iam” the best probability of sub right word is “am”, same for the last iteration where the whole word will be taken to the test as following: “iammostafa” -> “ammostafa” -> …. -> “mostafa” (which is the best probability and will be the last word saved) -> “ostafa” -> ….

After constructing the list of the best probability’s words the last element in this list is the last correct output expected which is “mostafa”.

By backtracking the path from the last element by subtracting the current index by the length the current word till we reach the beginning of the list then by reversing the visited words.

e.g., the first loops finished at index 10 which is “mostafa” then this word is pushed in a new list. The new index = 10 – 7 = 3 which is “am” then the last index = 3 - 2 = 1 which is “i”

Finally the final list is -> [“mostafa”,”am”,”i”]. by reversing it the final answer will be returned from the function.

```
00: ''
01: 'i'
02: 'a'
03: 'am'
04: 'mm'
05: 'ammo'
06: 'mos'
07: 'most'
08: 'a'
09: 'af'
10: 'mostafa'
```

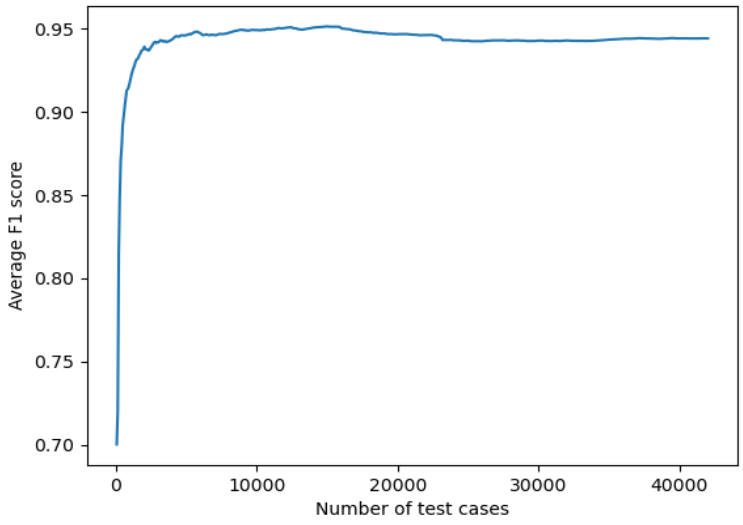
```
function VITERBI-SEGMENTATION(text, P) returns best words and their probabilities
  inputs: text, a string of characters with spaces removed
          P, a unigram probability distribution over words

  n ← LENGTH(text)
  words ← empty vector of length n + 1
  best ← vector of length n + 1, initially all 0.0
  best[0] ← 1.0
  /* Fill in the vectors best, words via dynamic programming */
  for i = 0 to n do
    for j = 0 to i − 1 do
      word ← text[j:i]
      w ← LENGTH(word)
      if P[word] x best[i- w] ≥ best[i] then
        best[i] ← P[word] × best[i − w]
        words[i] ← word
  /* Now recover the sequence of best words */
  sequence ← the empty list
  i ← n
  while i > 0 do
    push words[i] onto front of sequence
    i ← i − LENGTH(words[i])
  /* Return sequence of best words and overall probability of sequence */
  return sequence, best[i]
```

Figure **23.1** A Viterbi-based word segmentation algorithm. Given a string of words with spaces removed, it recovers the most probable segmentation into words.

Results and conclusions

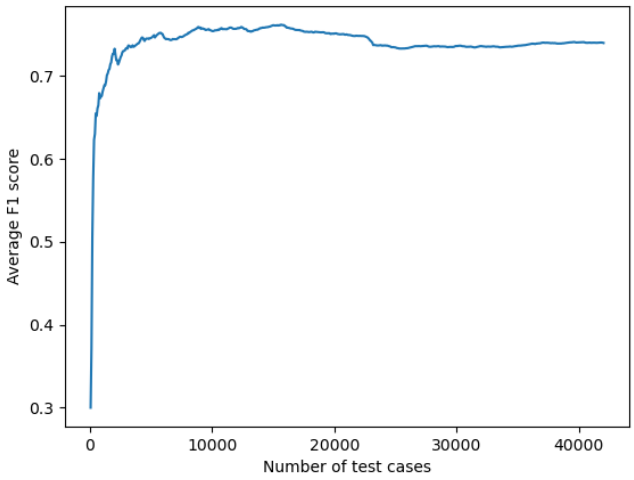
Note: The used F1 score is modified (explained before)

Number of test cases	Average F1 Score
1050	0.92
10010	0.9492
20020	0.9467
30030	0.9427
42000	0.9441



For 42000 test case input:

- Average F1 score modified = **0.9441**
- Average F1 score actual = **0.7386**
- Total execution time = **34.4114 seconds** in case of using only one F1 score calculation and **41.8905 seconds** in case of running both functions (to get actual time one of them must be commented)
- Memory usage = 3 MB (calculated by observing the task manager but there are many more professional ways to measure it)



*The above numbers are calculated in the program and the output is given in performance.txt

Note: these numbers are for three combined textbooks in which there are bigram words which are not handled and not included in the dataset. So, these numbers are results of completely random input not only unigram as the program expect.

Assumptions

- I assumed that the input will be only a unigram because bigram will not work perfectly for the chosen dataset and the implemented text preprocessing as not all the cases are handled.
- I assumed that the input sentence should be 90 words long or less. Because this type of inputs makes the algorithm not fully working as we don't give a probability (small one) to the non-existing words. The solution for this problem is to penalize these words with probability proportional to the word length. (Note: this solution is from an implemented code given in the references)

References

Dataset: http://norvig.com/ngrams/

Link for the solution of the problem stated in the assumptions: https://github.com/grantjenks/python-wordsegment/blob/master/wordsegment/__init__.py