Word Counting Project

Bin Han

Table of contents

Project Introduction

With this lab we practice basic operations on lists, string and dictionaries, as well as fundamental programming concepts such as loops, if statements and functions. The lab is designed to be simple but despite its simplicity it also demonstrates a technique which is in the foundations of many algorithms which allow machines to understand human language.

Note that machines do not understand language the way we do. Instead they collect statistics which allow them to make inferences which are right most of the time, even when the machine lacks real understanding. The most basic statistics is plain word counting, i.e. to collect which words appear in a document and how often. There are four parts in this lab:

Tokenization - how to split a running text into a sequence of words (tokens);

Counting - aggregate statistics for the frequencies of the different words; Printing - print the final results; Completed Program - assembling a complete program by using the pieces above.**

**Tokenization(Processing Lines and Recognizing Words) In most languages, it is easy to separate words by just looking for the spaces in between. There are exceptions of course but as longs as we stick with English (or Swedish), this is mostly true. Try this two lines in the Python shell:

```
text="Simple is better than complex"
print(text.split())
```

```
['Simple', 'is', 'better', 'than', 'complex']
```

It looks as if Python already knows how to separate into words. The method split simply splits a string into a list of words if they are separated by one or more spaces. Unfortunately this solution wouldn't go very far:

```
text = 'In the face of ambiguity, refuse the temptation to guess.'
print(text.split())
```

```
['In', 'the', 'face', 'of', 'ambiguity,', 'refuse', 'the', 'temptation', 'to', 'guess.']
```

Do you see the problem? Punctuation symbols such as comma and dot are usually glued to the last token without space, despite that we don't consider them part of the word. The method split does not know anything about punctuation, and that is the problem. We can do something better, but lets first clarify what we count as a word:

a word cannot contain white spaces. White spaces are used to detect word boundaries but are otherwise ignored; a word is any sequence of one or more letters, i.e. symbols from the alphabet; a word is any sequence of digits, e.g. the symbols "0"… "9"; any other symbol not mentioned above is counted as a single word containing only the symbol alone. Your first task is to define a function called tokenize which should take a complete document as a list of text lines and produce a list of tokens. For example:

```
def tokenize(lines):
    words=[]
    for line in lines:
        start =0
        while start < len(line):
            while start < len(line) and line[start].isspace():#skip the whitespace
                start=start+1
            #print(line[start])
            if start < len(line) and line[start].isalpha():#indentity the character type
                #print(f"{line[start]} is a letter")
                end=start
                while end<len(line) and line[end].isalpha():
                    end=end+1
                words.append(line[start:end].lower())
                start=end
```

```
            elif start <len(line) and line[start].isdigit():
                #print(f"{line[start]} is a digit")
                end=start
                while end <len(line) and line[end].isdigit():
                    end=end+1
                words.append(line[start:end])
                start=end
            elif start< len(line):
                #print(f"{line[start]} is a symbol")
                words.append(line[start])
                start=start+1
    return words
document = ['"They had 16 rolls of duct tape, 2 bags of clothes pins,',
            '130 hampsters from the cancer labs down the hall, and',
            'at least 500 pounds of grape jello and unknown amounts of chopped liver"',
            'said the source on a recent Geraldo interview.']
tokenize(document)
```

```
['"',
 'they',
 'had',
 '16',
 'rolls',
 'of',
 'duct',
 'tape',
 ',',
 '2',
 'bags',
 'of',
 'clothes',
 'pins',
 ',',
 '130',
 'hampsters',
 'from',
 'the',
 'cancer',
 'labs',
 'down',
 'the',
 'hall',
```

```
 ',',
 'and',
 'at',
 'least',
 '500',
 'pounds',
 'of',
 'grape',
 'jello',
 'and',
 'unknown',
 'amounts',
 'of',
 'chopped',
 'liver',
 '"',
 'said',
 'the',
 'source',
 'on',
 'a',
 'recent',
 'geraldo',
 'interview',
 '.']
```

Counting

The second part of the lab is to implement a function which takes a list of words and counts how often each word appears. In addition, the function takes a list of stop words. These are words which are not interesting and should be ignored while counting.

```
def countWords(words, stopwords):
    frequencies={}
    for word in words:
        if word in stopwords:
            continue
        if word not in frequencies:
           frequencies[word]=1
        else:
           frequencies[word]+=1
```

```
    return frequencies
countWords(['it','is','a','book'], ['a','is','it'])
```

```
{'book': 1}
```

Printing

The last missing piece is to be able to print the collected statistics. We already have a way to construct a dictionary where the keys are the words and the values are the counts. We just have to iterate through the entries and print the data. Precisely for that the dictionary type has a method called items.

```
def printTopMost(frequencies, n):
    sorted_items=sorted(frequencies.items(), key=lambda x:-x[1])#descreasing order to sort
    for i, (word, freq) in enumerate(sorted_items):
        if i >=n:
            break
        print(f"{word.ljust(20)}{str(freq).rjust(5)}")
printTopMost({'text': 9, 'word': 30, 'fiction': 6, 'count': 11, 'counting': 7, 'novel': 6},3)
```

```
word                   30
count                  11
text                    9
```

Completed Program

It is time to piece the different parts into a complete working program. Start a new module called topmost (file topmost.py) and import the module wordfreq from it.

```
import wordfreq
import sys
import urllib.request
def main():
    if len(sys.argv) != 4:
        print("Usage: python3 topmost.py <stopwords_file> <input_file> <top_n>")
        sys.exit(1)

    stopwords_file = sys.argv[1]
    input_file = sys.argv[2]
```

```
    top_n = int(float(sys.argv[3]))  # handles if passed as 20. instead of 20

    # Read stopwords
    f1=open(stopwords_file, "r")
    stopwords=[line.strip() for line in f1]
    f1.close()

    # Read input file
    if input_file.startswith("http://") or input_file.startswith("https://"):
        response=urllib.request.urlopen(input_file)
        lines=response.read().decode("utf8").splitlines()
    else:
        with open(input_file, "r", encoding="utf-8") as f2:
            lines = f2.readlines()

    tokens = wordfreq.tokenize(lines)
    word_counts = wordfreq.countWords(tokens, stopwords)
    wordfreq.printTopMost(word_counts, top_n)

# Call main
main()
```

```
ValueError: could not convert string to float: '--HistoryManager.hist_file=:memory:'
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[13], line 31
     28     wordfreq.printTopMost(word_counts, top_n)
     30 # Call main
---> 31 main()
Cell In[13], line 11, in main()
      9 stopwords_file = sys.argv[1]
     10 input_file = sys.argv[2]
---> 11 top_n = int(float(sys.argv[3]))  # handles if passed as 20. instead of 20
     13 # Read stopwords
     14 f1=open(stopwords_file, "r")
ValueError: could not convert string to float: '--HistoryManager.hist_file=:memory:'
```

Test the program:

```
import subprocess

result = subprocess.run(
    ["python", "topmost.py", "eng_stopwords.txt", "examples/article1.txt", "20"],
```

```
    capture_output=True,
    text=True
)

# Print output
print("Output:")
print(result.stdout)

# Print any errors
if result.stderr:
    print("Error output:")
    print(result.stderr)
```

```
Output:
word                    30
words                   21
count                   11
text                     9
000                      9
counting                 7
fiction                  6
novel                    6
rules                    5
length                   5
used                     4
usually                  4
details                  4
software                 4
sources                  4
.                        4
processing               4
segmentation             4
rule                     4
novels                   4
```