Ling 473 Project 2
Due 11:59 pm on Tuesday, August 17, 2021

For this project you will write a program to clean a corpus and tally the words in it. The resulting output is called a unigram language model. This is one of the most common tasks in computational linguistics text processing, as it is a prerequisite for many of the more advanced NLP tasks. For this task you will use part of the AQUAINT corpus of English newswire. You will process all of the files in the following directory, which contains 249 files containing New York Times news articles in SGML format.

`/corpora/LDC/LDC02T31/nyt/2000`

1. First, you will clean the files of formatting. The only appearance of the less-than (<) and greater-than (>) symbols in these files is for delimiting tags that we wish to discard. Accordingly, you will ignore all SGML tags, retaining only content that is outside of any SGML tag. In the following example (extracted from /corpora/LDC/LDC02T31/nyt/2000/20000101_NYT), the SGML tags we want to ignore are highlighted in red. In other words, we only want the content in black in this step.

   <DOC>
   <DOCNO> NYT20000101.0001 </DOCNO>
   <DOCTYPE> NEWS STORY </DOCTYPE>
   <DATE_TIME> 2000-01-01 00:18 </DATE_TIME>
   <HEADER>
   A3885 &Cx1f; ttc-z
   r s BC-BKW-CSUN-MISSISSIPPIS 01-01 0545
   </HEADER>
   </HEADLINE>
   (For use by NYTimes News Service clients)
   By LEE BARNATHAN
   c.2000 Los Angeles Daily News
   <TEXT>
   <P>
   LOS ANGELES _ School has been out at Cal State Northridge since the week before Christmas, but since you can learn something every day, Mississippi State's women's basketball team gave a lesson.
   </P>
   ……
   Hint: you can use regular expression to match the SGML tags and replace them by while spaces.

2. We are only interested in words, not punctuation, symbols, or digits. The next step in cleaning will be to gather all of the words. In real applications, you may want to run a tokenizer first to segment the sentence into a list of words (e.g., "*every data,*" will be segmented into a list of three words ["*every*", "*data*", ","]). However, in this project, we do not have a tokenizer. So, for this project, let us split the sentence/string by white spaces (so, "every day," will be segmented into a list of two words ["*every*", "*data,*"], where the comma is a part of the second word) and gather all the resulting words.

3. Keep the words that formed by one or more acceptable characters (i.e., filter out the words that contain any unacceptable characters). The acceptable characters include only the following:

   a. Capital letters `A` through `Z`

b. Lower-case letters a through z
c. The straight apostrophe, ' which is ASCII character 39, (\x27).

In the following example, words that are filtered out are highlighted in red.

NYT20000101.0001
NEWS STORY
000-01-01 00:18
A3885 &Cx1f; ttc-z
r s BC-BKW-CSUN-MISSISSIPPIS 01-01 0545
(For use by NYTimes News Service clients)
By LEE BARNATHAN
c.2000 Los Angeles Daily News
LOS ANGELES _ School has been out at Cal State Northridge since the week before Christmas, but since you can learn something every day, Mississippi State's women's basketball team gave a lesson.
……

4. After gathering these words, perform the following operations:
    a. Filter out the words that start or end with the straight apostrophe. For example, "*'words*" and "*words'*" will be filtered out. Note that this preserves internal occurrences of this character, as in the words "*Marvin's*" or "*O'Conner*".
    b. Convert every word to lower-case (e.g., "*NEWS*" will be lower-cased to "*news*")
5. Finally, you will tally (count) the number of instances of each word, and write this information, sorted according to descending order of frequency (the order of different words does not matter if their frequencies are the same), to the console (stdout). Each line will consist of the word, exactly one tab character, and then a number representing the total number of instances of that word in the entire corpus.

Suggestions

For tallying, you will most certainly want to use a hash table that maps a string (the word) to an integer (the current tally for that word). Depending on programming language, this data structure is called a *dictionary*, *map*, or perhaps *associative array*.

Because the results must be presented in sorted order, you must maintain the tallies in memory and process all the files, prior to sorting the results and writing them to the console.

Running the Program

The program should accept exactly one argument which is the absolute path to the directory containing the files to be processed. Do not hardcode the path in your code, which will disallow us from testing the code on new data. For other path, use the relative paths (paths that do not start with '/') to reference files you are including with your submission. Do not directly reference your home directory, since I may not have permissions for it when I'm running your program.

The program prints the required output to the standard output (e.g., use 'print' function is python). Each line is formatted by "word + a tab + the count". The results are written in descending order of their count. Here is an example of what the first few lines might look like:

```
the     4282861
of      1877767
to      1866867
a       1841045
and     1722331
in      1456567
that    779527
for     752122
is      693609
by      552998
on      545767
etc...
```

In other words, the output should match the format in `/dropbox/20-21/473/project2/example.output`.

Run your code on Patas with the following command line

```
./run.sh /corpora/LDC/LDC02T31/nyt/2000 >output
```

and obtain the "output" file. You can test your code on a small set of data which is offered to you at `/dropbox/20-21/473/project2/example_input`. Ideally, if you run your code with

```
./run.sh /dropbox/20-21/473/project2/example_input >ex.output
```

your output should match the one at `/dropbox/20-21/473/project2/example.output`.

Hint: it may take 5-10 minutes to run on patas. So it is recommended to use condor (but not required).

Submission

hw.tar.gz

Include the following files in your submission:

`run.sh`	The command(s) that run your program.
`output`	This is the captured console output (stdout) from running your program. This file should be produced by: `./run.sh /corpora/LDC/LDC02T31/nyt/2000 >output`
`(source code and binary files)`	All source code and binary files (jar, a.out, etc., if any) required to run and compile your program

Similar to project 1, gather together all the required files and use the following command to package your files for submission.

```
tar -czf hw.tar.gz ./*
```

Notice that this command packages all files in the current directory; do not include any top-level directories.

Check whether your submission contains all required file using

`/dropbox/20-21/473/project2/check_project2.sh`

`readme.{pdf, txt}`

Your write-up of the project. Describe your approach, any problems or special features, or anything else you'd like me to review. If you could not complete some or all of the project's goals, please explain what you were able to complete.

Submit **hw.tar.gz** and **readme.{pdf, txt}** to Canvas.

Corpus Citation

David Graff. 2002. *The AQUAINT Corpus of English News Text*. LDC, Philadelphia.