Computer Science AP/X — CSCI-140/242
Zipf's Law — Project 1

9/13/2016

Due Date: Wednesday, October 19 @ 11:59pm
Version Info:

- **9/16/16:** Fixed help message for activity 4 and removed duplicate entries from data files (**please download again**).
- **9/15/16:** Initial version.

1 Overview

George Kingsley Zipf was a linguist who studied statistical occurrences in different languages. He theorized that given a sufficiently large body of language, the frequency of each word is close to inversely proportional to its rank in the frequency table. In other words, the most frequent word will occur approximately twice as often as the second most frequent word, three times more often than the third most frequent word, etc.

Google Books NGram Viewer provides us with a rich set of data of word occurrences in literature. An *n-gram* maps a contiguous sequence of `n` words. We will be looking only at *unigrams* (where `n=1`). This tracks occurrences of single words, without any dependency on previous words.

Using the studies of Zipf, along with the unigram data from Google, you will develop programs that study several features of language that are of interest to the field of linguistics.

2 Requirements Overview

2.1 Project Overview

In order, the programs you will develop are:

1. `word_count.py`: Counts the total number of occurrences of a word across all years.
2. `letter_freq.py`: Generates and plots the frequency distribution of individual letters in the words across all occurrences and all years.
3. `word_freq.py`: Generates and plots the relative popularity of each word, in relation to all others, across all occurrences and all years.
4. `word_length.py`: Generates and plots the average word length over a range of years.

2.2 Getting Started

2.2.1 matplotlib

Go through these installation instructions and get matplotlib installed on your machine.

Several of the programs you will be developing will be using this library to plot the data you extract from the unigrams. You can get a plethora of examples from the matplotlib website.

2.2.2 Setting Up PyCharm Project

1. Download and unzip the unigram data files into a `data` subdirectory.
2. Create a new PyCharm project and move the `data` directory into the top level of the project.
3. Create a `src` directory. Right click on the directory and select `Mark Directory as -> Sources Root`.
4. In your `src` directory, create the four main programs from section `2.1`.
5. Right click on the first program, `word_count.py` and select `Run 'word_count'`.
6. In the run configuration that appears next to the green run arrow, select `Edit Configurations...`.
7. In the new dialog, under `Working directory`, remove the trailing `/src` so the path points to the root directory of your project. Now when you run your project you can point to a unigram file, e.g. `short.csv` by specifying it as a script parameter in the run configuration, `data/short.csv`.
8. Create a Git repository in your `src` directory (Tools, Import into Version Control, Create Git repository...). All source files should be added to Git control. **Make certain you do not add the data files to source control! You do not have enough space on your CS accounts to store them.**

2.3 Unigram File Format

The unigram files are ASCII text formatted with *comma separated value* entries. Each line of the file represents one word, the year it appeared in, and the total number of occurrences of the word in all literature for the aforementioned year.

Take a look at the contents of `short.csv`:

```
airport, 2007, 175702
airport, 2008, 173294
request, 2005, 646179
request, 2006, 677820
request, 2007, 697645
request, 2008, 795265
wandered, 2005, 83769
wandered, 2006, 87688
wandered, 2007, 108634
wandered, 2008, 171015
```

For example, the word `request` appeared `646,179` times in the year `2005` or `2006`. If a word does not appear in a year it will not have an entry, e.g. `airport` did not appear in the year `2005`.

2.4 Command Line Processing with `argparse`

It is strongly suggested that you use the built-in `argparse` module for handling all command line processing for this project. All input will be fed to your programs through the command line (versus standard input).

Please read through the Argparse Tutorial and feel free to experiment with their examples so you understand how both positional and optional command line arguments are processed.

2.5 Design

This writeup is only providing you with the program requirements. You are free to design your programs however you wish. You may use whatever built-in tools exist in the default Python3 environment.

Because some tasks, like reading a unigram file, are common to all the programs, you should consider writing a separate helper module that can be re-used. It is considered bad design to make copies of your code.

When implementing your solution, you must be as efficient as possible when storing and retrieving data. All the program should be able to run with the large data set, `all.csv`, in under a minute.

To represent the entries in a unigram file, you should consider using either a `namedtuple` or a `class`.

You most likely will be using python's built-in dictionary as your main data structure. Make certain you understand why it is bad practice to look up a key by iterating over the dictionary, e.g.:

```
words = {}
...
for word in words:
    if word == 'special':
        return word['special']
```

This is an `O(n)` operation, where `n` is the number of unique entries in the dictionary. This should have been done more efficiently in `O(1)` time, e.g.:

```
if 'special' in words:
    return word['special']
```

3 Activity 1: Word Counting

The goal for this activity is to write a program, `word_count.py`, that is be able to count the total number of occurrences of a word across all years.

3.1 Command Line Arguments

The usage can be found by running the program, using `argparse`, with the help option, `-h`:

```
$ python3 -h word_count.py
usage: word_count.py [-h] word filename

positional arguments:
word        a word to display the total occurrences of
filename    a comma separated value unigram file

optional arguments:
-h, --help  show this help message and exit
```

3.2 Error Handling

If the file exists, it is guaranteed to be formatted correctly. Otherwise, display the following message to standard error and exit:

```
Error: {filename} does not exist!
```

Here, {`filename`} should be replaced with the file name from the command line.

If a word does not exist in the file, you should display the following message to *standard error* and exit:

```
Error: {word} does not appear!
```

Here, {`word`} should be replaced with the word from the command line.

To write to standard error, you must import the `sys` module and the message can be printed by sending a string to the `sys.stderr.write` method.

3.3 Output

If the word is found in the file, the word and its total number of occurrences across all years should be displayed to standard output in the format:

```
{word}: {count}
```

Here, {`word`} should be replaced with the word on the command line, and {`count`} should be replaced with the total count.

3.4 Examples

```
$ python3 word_count.py airport data/short.csv
airport: 348996

$ python3 word_count.py president data/all.csv
president: 47758594

$ python3 word_count.py garbage data/short.csv
Error: garbage does not appear!

$ python3 word_count.py airport garbage
Error: garbage does not exist!
```

4 Activity 2: Letter Frequency

The goal for this activity is to write a program, `letter_freq.py`, that can compute the frequency of each letter, `a-z`, across the total occurrences of all words over all years.

4.1 Command Line Arguments

The usage can be found by running the program with the help option, `-h`:

```
$ python3 letter_freq.py -h
usage: letter_freq.py [-h] [-o] [-p] filename

positional arguments:
  filename      a comma separated value unigram file

optional arguments:
  -h, --help    show this help message and exit
  -o, --output  display letter frequencies to standard output
  -p, --plot    plot letter frequencies using matplotlib
```

4.2 Error Handling

If the file exists, it is guaranteed to be formatted correctly. Otherwise, display the following message to standard error and exit:

```
Error: {filename} does not exist!
```

4.3 Output

If the output option is specified, the program should display the frequency distributions for each letter in alphabetical order, one letter per line, in the format:

```
{letter}: {frequency}
```

Here, {letter} is the letter, and {frequency} is that letter's frequency (a floating point value between `0.0-1.0` inclusively). The sum of all the letter's frequencies should be `1.0`.

4.4 Plotting

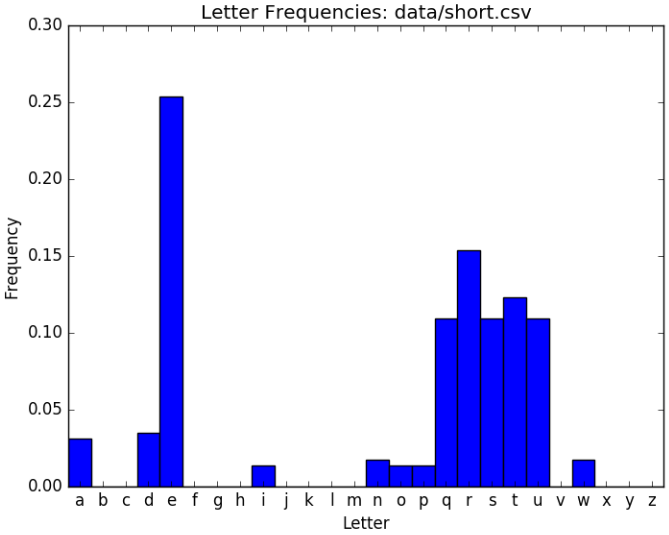
If the plotting option is specified, the program should display a histogram chart of the distributions.

- The title of the chart should be `"Letter Frequencies: {filename}"`. Here, {filename} is the name of the unigram file.
- The letters should go alphabetically left to right along the `x` axis with the label `"Letters"`.
- The frequencies should go in increasing value up the `y` axis with the label `"Frequency"`.

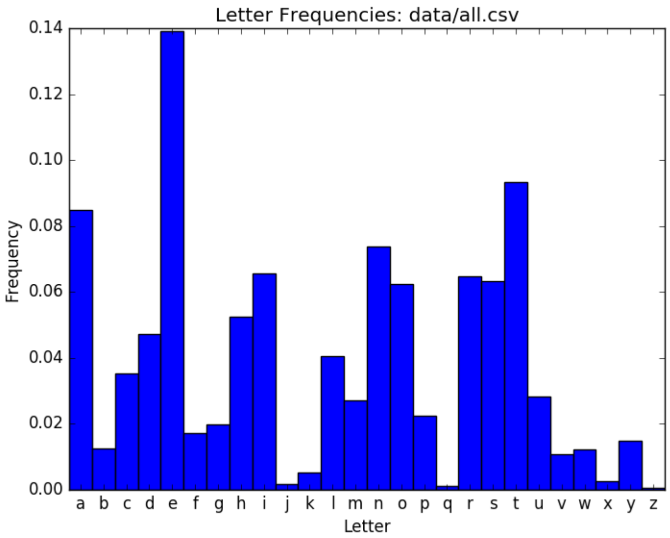
To do a histogram plot using `matplotlib`, refer to the bar method.

4.5 Examples

```
$ python3 letter_freq.py -o -p data/short.csv
a: 0.03104758705050717
b: 0.0
c: 0.0
d: 0.03500991824543893
e: 0.2536276129665047
f: 0.0
g: 0.0
h: 0.0
i: 0.013542627927787708
j: 0.0
k: 0.0
l: 0.0
m: 0.0
n: 0.017504959122719464
o: 0.013542627927787708
p: 0.013542627927787708
q: 0.10930884736053291
r: 0.15389906233882777
s: 0.10930884736053291
t: 0.12285147528832062
u: 0.10930884736053291
v: 0.0
w: 0.017504959122719464
x: 0.0
y: 0.0
z: 0.0
```



```
$ python3 letter_freq.py -o -p data/all.csv
a: 0.08485318771057006
b: 0.012667139569625722
c: 0.0352208769615757
d: 0.04735701027387059
e: 0.13924384449763727
f: 0.017187450073295318
g: 0.019864640528982708
h: 0.05236885693735097
i: 0.0657280703493182
j: 0.0018023874666368133
k: 0.005300854197729794
l: 0.040495094632853265
m: 0.027141880530028202
n: 0.07389032639136611
o: 0.062361268153893736
p: 0.022460236643795407
q: 0.0012377104962064355
r: 0.06475108655184325
s: 0.0633585740587229
t: 0.09320036015852061
u: 0.02814749270988877
v: 0.01085693113996992
w: 0.012198782896875544
x: 0.0026038365062501937
y: 0.015014078573617808
z: 0.000688021989574694
```



5 Activity 3: Word Frequency

The goal for this activity is to write a program, word_freq.py, that can compute the popularity of a word in terms of the number of occurrences of all words over all years. The relationship of the words, by rank, can be plotted as a log-log chart to reveal Zipf's law.

5.1 Command Line Arguments

The usage can be found by running the program with the help option, -h:

```
usage: word_freq.py [-h] [-o OUTPUT] [-p] word filename

positional arguments:
  word                  a word to display the overall ranking of
  filename              a comma separated value unigram file

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT, --output OUTPUT
                        display the top OUTPUT (#) ranked words by number of
                        occurrences
  -p, --plot            plot the word rankings from top to bottom based on
                        occurrences
```

5.2 Error Handling

If the file exists, it is guaranteed to be formatted correctly. Otherwise, display the following message to standard error and exit:

```
Error: {filename} does not exist!
```

Here, filename is the name of the file. If the word to search for is not found in the file, an error message should be displayed to standard error, and the program should exit:

```
Error: {word} does not appear in {filename}
```

Here, {word} is the word specified on the command line to search for and filename is the name of the file.

5.3 Output

The first line should display, to standard output, the overall rank of the word (where 1 is the top), e.g.:

```
{word} is ranked #{num}
```

Here, {word} is the word that was specified on the command line, and {num} is the rank of the word.

Next, the top ranked words from 1 up to and including the output number (specified on the command line) should be displayed, one line at a time, to standard output:

```
#{num}: {word} -> {count}
```

Here, {num} is the rank of the word, {word} is the word with that rank, and {count} is the overall number of occurrences of the word over all years.

If the output number is larger than the total number of words, your program should only print out the rankings for the total number of words.

5.4 Plotting

If the plotting option is specified, the program should display a *log-log* chart that ranks the words, in descending order of popularity, against each word's overall number of occurrences across all years.

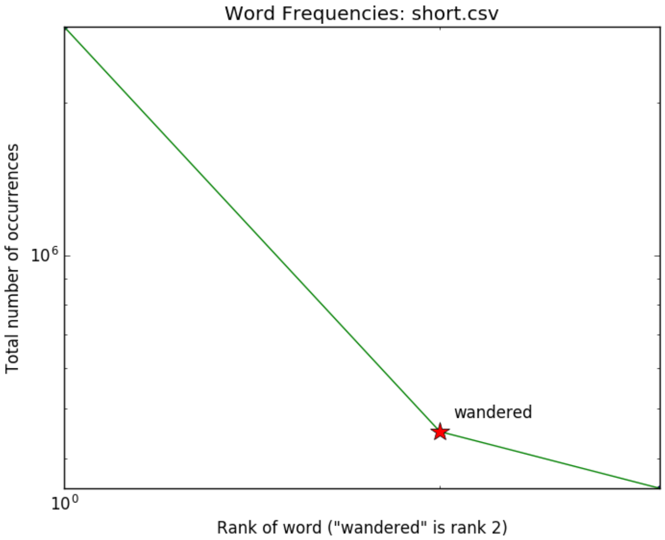
- The title of the chart should be `"Word Frequencies: {filename}"`. Here, {filename} is the name of the unigram file.
- The ranks should plot along the `x` axis from highest to lowest rank. The label should be `"Rank of word("{word}" is rank {rank})"`. Here, {word} is the word to find the ranking for, and {rank} is the rank of the word.
- The frequencies should go in increasing value up the `y` axis with the label `"Frequency"`.
- The word to find the rank for should be plotted on the graph as a star so that it stands out.

To do a log-log plot using `matplotlib`, refer to the loglog method.

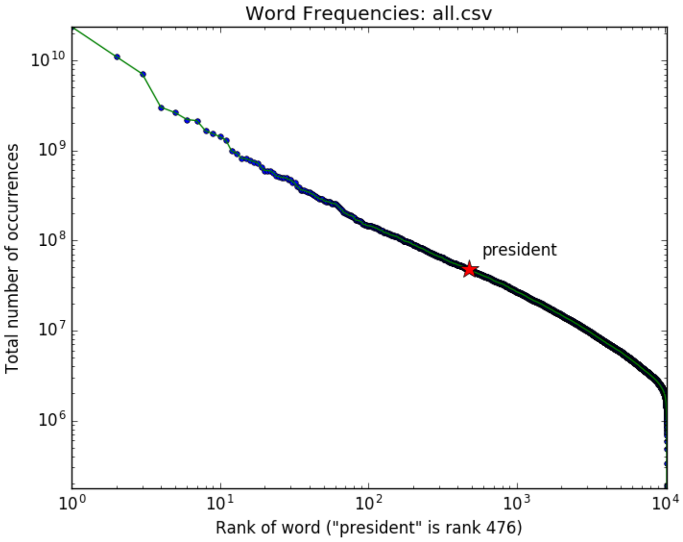
5.5 Examples

```
$ python3 word_freq.py -o 3 -p wandered data/short.csv
wandered is ranked #2
#1: request -> 2816909
#2: wandered -> 451106
#3: airport -> 348996
```

```
$ python3 word_freq.py president data/short.csv
Error: president does not appear in data/short.csv
```



```
$ python3 word_freq.py -o 20 -p president data/all.csv
president is ranked #476
#1: the -> 23688414489
#2: and -> 11021132912
#3: a -> 7083003595
#4: for -> 3021925527
#5: as -> 2626386982
#6: it -> 2204134503
#7: on -> 2143313303
#8: from -> 1651527506
#9: he -> 1529423270
#10: this -> 1423199366
#11: had -> 1298386780
#12: they -> 983804307
#13: all -> 930734157
#14: can -> 812711151
#15: been -> 808575455
#16: her -> 776381591
#17: more -> 746702714
#18: would -> 726248425
#19: other -> 651163319
#20: may -> 598572490
```


Word Frequencies: all.csv
Total number of occurrences
10^{10}
10^9
10^8
10^7
10^6
president
10^0
10^1
10^2
10^3
10^4
Rank of word ("president" is rank 476)

6 Activity 4: Average Word Length

The goal for this activity is to write a program, `word_length.py`, that can compute the average word length over a range of years. These results can be plotted as a line chart.

6.1 Command Line Arguments

The usage can be found by running the program with the help option, `-h`:

```
usage: word_length.py [-h] [-o] [-p] start end filename

positional arguments:
  start         the starting year range
  end           the ending year range
  filename      a comma separated value unigram file

 optional arguments:
  -h, --help    show this help message and exit
  -o, --output  display the average word lengths over years
  -p, --plot    plot the average word lengths over years
```

6.2 Error Handling

If the file exists, it is guaranteed to be formatted correctly. Otherwise, display the following message to standard error and exit:

```
Error: {filename} does not exist!
```

If the start year specified is greater than the end year, display the following message to standard error and exit:

```
Error: start year must be less than or equal to end year!
```

6.3 Output

Each year and the average word length should be displayed to standard output, one line per year, in the format:

```
{year}: {length}
```

Here, `{year}` is the year, and `{length}` is the average length for that year.

6.4 Plotting

If the plotting option is specified, the program should display a *line* chart that plots a given year against the average length of that word for the year.

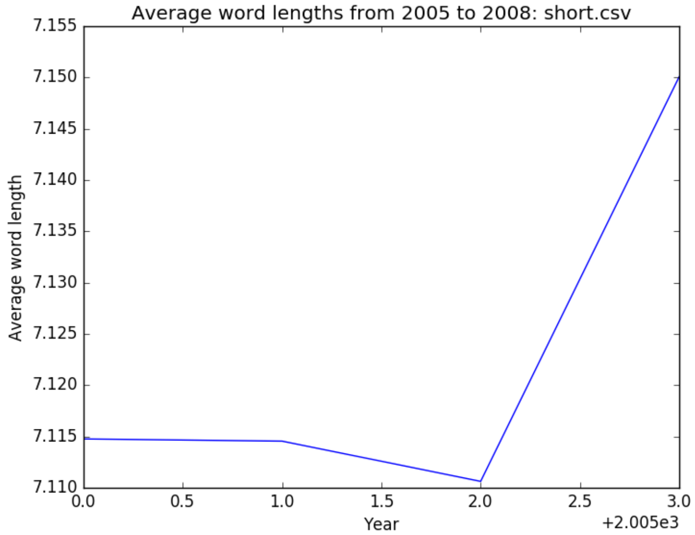
- The title of the chart should be `"Average word lengths from {start} to {end} : {filename}"`. Here, {start} is the start year, {end} is the end year, and {filename} is the name of the unigram file.
- The years should go in ascending order along the `x` axis with the label `"Year"`.

- The average word lengths should go in increasing value up the `y` axis with the label `"Average word length"`.

To do a line plot using `matplotlib`, refer to the plot method.

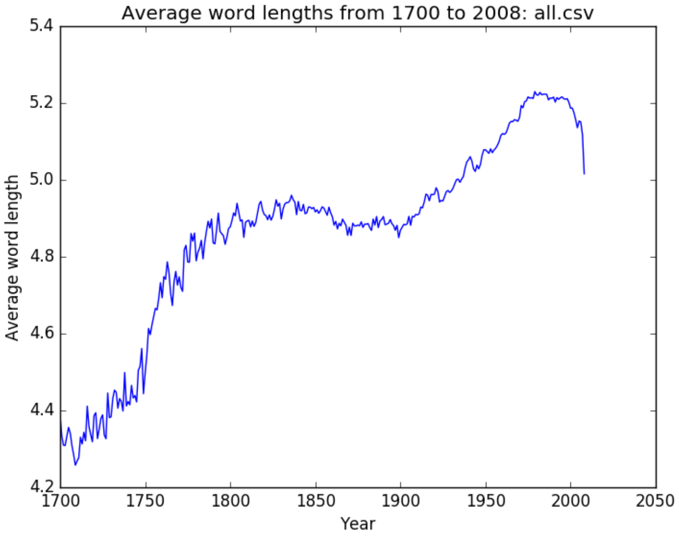
6.5 Examples

```
$ python3 word_length.py -o -p 2005 2008 data/short.csv
2005: 7.1147602294958
2006: 7.114548770228398
2007: 7.110627395031065
2008: 7.150069236398865
```



```
$ python3 word_length.py -o 2000 2008 data/all.csv
2000: 5.186467678279437
2001: 5.18675672147985
2002: 5.174571777742466
2003: 5.156411602922902
2004: 5.135465894450372
2005: 5.153216230810274
2006: 5.150410237817308
2007: 5.116866369875739
2008: 5.015915530598189
```

```
$ python3 word_length.py -p 1700 2008 data/all.csv
```



7 Grading

The grade breakdown for this assignment is as follows:

- Functionality: 70%
  - Word Counting: 10%
  - Letter Frequency: 20%
  - Word Frequency: 20%
  - Average Word Length: 20%
- Design: 20%. Your code supports code reuse and has ample reuse of common function/s. You've also chosen to store your word data in a structure that is easier for a programmer to work with (e.g. a dictionary containing `namedtuple's` or `classes`).
- Code Documentation and Version Control: 10%

8 Submission

The project is to be submitted on the CS systems using `try`. You need to do the same steps for creating a bare repository and a cloned repository as for labs.

Submit all of your source code, along with the Git log file. Assuming you are in the `src` directory of your cloned repository:

```
$ try csapx-grd project1-1 *.py log.txt
```