We love Perl

Aims

This week you get to see how wonderful Perl can be for some tasks.

Assessment

Submission: `give cs2041 lab07 total_words.pl count_word.pl frequency.pl log_probability.pl identify_poet.pl [total_words.py count_word.py frequency.py log_probability.py identify_poet.py]]`
Deadline: either during the lab, or Monday 12 September 11:59pm (midnight)
Assessment: Make sure that you are familiar with the lab assessment criteria (lab/assessment.html).
We have covered only a small amount of Perl in lectures. In fact, to cover the whole language in detail would take a whole semester, so we're going to rely on you finding out about the language yourself in tutes, labs and assignments. A good place to start is the Perl documentation & tutorial links on the class home page For example you might find these useful:

- Perl language syntax (http://search.cpan.org/dist/perl/pod/perlsyn.pod)
- Perl functions (http://search.cpan.org/dist/perl/pod/perlsub.pod)
- Perl operators (http://search.cpan.org/dist/perl/pod/perlop.pod)

Introduction

In this lab exercise you will implement a Naive Bayes Document classifier (http://en.wikipedia.org/wiki/Naive_Bayes_classifier#Document_Classification) in Perl. Naive Bayes Document classifiers are used widely for applications such as spam filtering and language recognition.
You will use the approach to discover the author of a piece of poetry.

The directory /home/cs2041/public_html/lab/perl/poets/poems/ (lab/perl/poets/poems) contains text files containing poems written by 10 famous poets:

- Elizabeth Barrett Browning (lab/perl/poets/poems/Elizabeth_Barrett_Browning.txt)
- Emily Dickinson (lab/perl/poets/poems/Emily_Dickinson.txt)
- John Keats (lab/perl/poets/poems/John_Keats.txt)
- Percy Bysshe Shelley (lab/perl/poets/poems/Percy_Bysshe_Shelley.txt)
- Robert Frost (lab/perl/poets/poems/Robert_Frost.txt)
- Samuel Taylor Coleridge (lab/perl/poets/poems/Samuel_Taylor_Coleridge.txt)
- Walt Whitman (lab/perl/poets/poems/Walt_Whitman.txt)
- William Blake (lab/perl/poets/poems/William_Blake.txt)
- William Butler Yeats (lab/perl/poets/poems/William_Butler_Yeats.txt)
- William Wordsworth (lab/perl/poets/poems/William_Wordsworth.txt)

Your program will be given a poem and should determine which of these 10 famous poets is most likely to have written this poem. You program will based this on the probability that a particular poet will use a given word. This can be estimated from the frequency which the poet uses the word in the poems you are given. You should link the directory `/home/cs2041/public_html/lab/perl/poets/poems/` into your `lab07` directory. So to get started do something like this.

```
$ ln -s ~cs2041/public_html/lab/perl/poets/poems
$ ln -s ~cs2041/public_html/lab/perl/poets/poem?.txt
```

The `ln -s` command creates a symbolic link which saves disk space. If you are working on your own machine you may prefer to copy the files across and add them to your repo:

```
$ cp -r ~cs2041/public_html/lab/perl/poets/poems .
$ cp ~cs2041/public_html/lab/perl/poets/poem?.txt .
$ git add poems poem?.txt
```

Exercise: Total Words

Write a Perl script `total_words.pl` which counts the total number of words found on in its input (`STDIN`).
For the purposes of this program and the following programs we will define a word to be maximal non-empty contiguous sequences of alphabetic characters (`[a-zA-Z]`).

Any characters other than `[a-zA-Z]` separate words.

So for example the phrase " `The soul's desire` " contains 4 words: `("The", "soul", "s", "desire")`

For example:

```
$ ./total_words.pl <poems/Walt_Whitman.txt
116941 words
$ ./total_words.pl <poems/Emily_Dickinson.txt
112882 words
$ ./total_words.pl <poems/Robert_Frost.txt
21698 words
```

Hint: if your word counts are out a little you might be counting empty strings (split can return these).

Straight-forward solution using split for total_words.pl

```
#!/usr/bin/perl -w
$count = 0;
while ($line = <STDIN>) {
    my @words = split /[^a-zA-Z]+/, $line;
    foreach $word (@words) {
        $count++ if $word ne '';
    }
}
print "$count words\n"
```

More concise solution for total_words.pl

```
#!/usr/bin/perl -w
$count = 0;
while (<STDIN>) {
    foreach (/[a-z]+/ig) {
        $count++;
    }
}
print "$count words\n"
```

As usual:

```
$ ~cs2041/bin/autotest lab07 total_words.pl
$ git add total_words.pl
$ git commit -a -m "total_words.pl passes dryrun tests!"
```

Exercise: Count Word

Write a Perl script `count_word.pl` which counts the number of times a specified word is found on in its input (`STDIN`).

A word is as defined for the previous exercise.

The word you should count will be specified as a command line argument.

You program should ignore the case of words.

For example:

```
$ ./count_word.pl snow <poems/Elizabeth_Barrett_Browning.txt
snow occurred 11 times
$ ./count_word.pl path <poems/Robert_Frost.txt
path occurred 3 times
$ ./count_word.pl urn <poems/John_Keats.txt
urn occurred 7 times
$ ./count_word.pl England <poems/William_Blake.txt
england occurred 34 times
```

Hint: modify the code from the last exercise.

Hint: the Perl function `uc` & `lc` convert strings tolower & uppercase respectively.

Sample solution for count_word.pl

```
#!/usr/bin/perl -w
die "Usage: $0 <word>\n" if @ARGV != 1;
$specified_word = lc $ARGV[0];
$count = 0;
while ($line = <STDIN>) {
    $line = lc $line;
    my @words = split /[^a-z]+/i, $line;
    foreach $word (@words) {
        $count++ if $word eq $specified_word;
    }
}
print "$specified_word occurred $count times\n"
```

As usual:

```
$ ~cs2041/bin/autotest lab07 count_word.pl
$ git add count_word.pl
$ git commit -a -m "count_word.pl inital version - not working"
```

Exercise: Word Frequency

Write a Perl script `frequency.pl` which prints the frequency with each poet uses a word specified as argument. So if Robert Frost uses the word *"snow"* 30 times in the 21699 words of his poetry you are given, then its frequency is `30/21699 = 0.001382552`. For example:

```
$ ./frequency.pl snow
  11/ 50080 = 0.000219649 Elizabeth Barrett Browning
  56/112882 = 0.000496093 Emily Dickinson
  11/ 62844 = 0.000175037 John Keats
  24/ 53400 = 0.000449438 Percy Bysshe Shelley
  30/ 21698 = 0.001382616 Robert Frost
  19/ 38123 = 0.000498387 Samuel Taylor Coleridge
  14/116941 = 0.000119718 Walt Whitman
  18/ 40751 = 0.000441707 William Blake
  22/121198 = 0.000181521 William Butler Yeats
  25/117162 = 0.000213380 William Wordsworth
```

So of these poets Robert Frost uses the word *"snow"* most frequently. If you choose a word a randomly from Robert Frost the probability it will be "snow" is just over in 1 in a thousand (0.1%).

Make sure your Perl script produces exactly the output above (the printf format is `"%4d/%6d = %.9f %s\n"`). Note you should ignore case (change A-Z to a-z). You should treat as a word any sequence of alphabetic characters. You should treat non-alphabetic characters (characters other than a-z) as spaces.

Hint: use a hash table of hash tables indexed by poet and word to store the word counts.

Hint: this loop executes once for each `.txt` file in the directory `poems`.

```
    foreach $file (glob "poems/*.txt") {
        print "$file\n";
    }
```

Hint: reuse code from the last exercise.

Sample solution for frequency.pl

```
#!/usr/bin/perl -w

foreach $file (glob "poems/*") {
    my $poet = $file;
    $poet =~ s/.*\///;
    $poet =~ s/\.txt$//;
    $poet =~ s/_/ /g;
    open my $f, '<', $file or die "can not open $file: $!";
    while ($line = <$f>) {
        $line = lc $line;
        foreach $word ($line =~ /[a-z]+/g) {
            $frequency{$poet}{$word}++;
            $n_words{$poet}++;
        }
    }
    close $f;
}

foreach $word (@ARGV) {
    $word = lc $word;
    foreach $poet (sort keys %frequency) {
        my $f = $frequency{$poet}{$word}||0;
        my $n = $n_words{$poet};
        printf "%4d/%6d = %.9f %s\n", $f, $n, $f/$n, $poet;
    }
}
```

```
$ ~cs2041/bin/autotest lab07 frequency.pl
$ git add frequency.pl
$ git commit -a -m "what's your frequency Kenneth?"
```

Exercise: Word Log Probability

Now suppose we have the phrase *"Truth is beauty."* if John Keats uses the word *"truth"* with frequency 0.000333885 and the word "*is*" with frequency 0.004944671, the word *"beauty"* with frequency 0.000747265. We can estimate the probability of Keats writing the phrase *"Truth is beauty."* as:

```
0.000333885 * 0.004944671 * 0.000747265 == 1.23369825533711e-09
```

We could similarly estimate probabilities for each of the other 9 poets, and then determine which of the 10 poets is most likely to write *"Truth is beauty."* (it's Blake).

A sidenote: we are actually making a large simplifying assumption in calculating this probability. Its often called the *bag of words model* (http://en.wikipedia.org/wiki/Bag_of_words_model).

Multiplying probabilities like this quickly leads to very small numbers and may result in arithmetic underflow of our floating point representation. A common solution to this underflow is instead to work with the *log* of the numbers.

So instead we will calculate the the log of the probability of the phrase. You this by adding the log of the probablities of each word. For example, you calculate the log-probability of Keats like this:

```
log(0.000333885) + log(0.004944671) + log(0.000747265)  == -20.5132494670232 == log(1.23369825533711e-09)
```

Log-probabilities can be used directly to determine the most likely poet, as the poet with the highest log-probability will also have the highest probability.

Another problem is that we might be given a word that a poet has not used in the poems we have. For example:

```
$ ./frequency.pl mortality
   0/ 50080 = 0.000000000 Elizabeth Barrett Browning
   4/112882 = 0.000035435 Emily Dickinson
   5/ 62844 = 0.000079562 John Keats
   5/ 53400 = 0.000093633 Percy Bysshe Shelley
   0/ 21698 = 0.000000000 Robert Frost
   0/ 38123 = 0.000000000 Samuel Taylor Coleridge
   1/116941 = 0.000008551 Walt Whitman
   0/ 40751 = 0.000000000 William Blake
   0/121198 = 0.000000000 William Butler Yeats
   7/117162 = 0.000059746 William Wordsworth
```

It's not useful to assume there is zero probability that the poet would use the word, even though they haven't used it previously. You should avoid this when estimating probabilities by adding 1 to the count of occurrences of each word. So for example we'd estimate the probability of Robert Frost using the word *mortality* as (0+1)/21699 and the probability of John Keats using the word *mortality* as (5+1)/62896. This is a simple version of Additive smoothing (http://en.wikipedia.org/wiki/Additive_smoothing).

Write a perl script `log_probability.pl` which given an argument prints the estimate log of the probability that a poet would use this word. For example:

```
$ ./log_probability.pl mortality
log((0+1)/ 50080) = -10.8214 Elizabeth Barrett Browning
log((4+1)/112882) = -10.0247 Emily Dickinson
log((5+1)/ 62844) =  -9.2567 John Keats
log((5+1)/ 53400) =  -9.0938 Percy Bysshe Shelley
log((0+1)/ 21698) =  -9.9850 Robert Frost
log((0+1)/ 38123) = -10.5486 Samuel Taylor Coleridge
log((1+1)/116941) = -10.9763 Walt Whitman
log((0+1)/ 40751) = -10.6152 William Blake
log((0+1)/121198) = -11.7052 William Butler Yeats
log((7+1)/117162) =  -9.5919 William Wordsworth
```

You will only need to copy your `frequency.pl` and make a small modification. Make sure your output matches the above exactly (the printf format is `"log((%d+1)/%6d) = %8.4f %s\n"`)

Sample solution for log_probability.pl

```
#!/usr/bin/perl -w

foreach $file (glob "poems/*") {
    my $poet = $file;
    $poet =~ s/.*\///;
    $poet =~ s/\.txt$//;
    $poet =~ s/_/ /g;
    open my $f, '<', $file or die "can not open $file: $!";
    while ($line = <$f>) {
        $line = lc $line;
        foreach $word ($line =~ /[a-z]+/g) {
            $frequency{$poet}{$word}++;
            $n_words{$poet}++;
        }
    }
    close $f;
}

foreach $word (@ARGV) {
    $word = lc $word;
    foreach $poet (sort keys %frequency) {
        my $f = $frequency{$poet}{$word}||0;
        my $n = $n_words{$poet};
        printf "log((%d+1)/%6d) = %8.4f %s\n", $f, $n, log(($f+1)/$n), $poet;
    }
}
```

```
$ ~cs2041/bin/autotest lab07 log_probability.pl
$ git add log_probability.pl
$ git commit -a -m "logs are useful"
```

Exercise: Identifying the Poet

Write a Perl script `identify_poet.pl` that given 1 or more files, each containing a poem, prints the most likely poet for each poem.

In other words, for each file given as argument you should go through all (10) poets calculating the log-probability that the poet wrote that poem by summing the log-probability of that poet using each word in the file. You should print the poet with the highest log-probability.

The files poem1.txt (lab/perl/poets/poem1.txt), poem2.txt (lab/perl/poets/poem2.txt) and poem3.txt (lab/perl/poets/poem3.txt) contain famous poems by Keats, Shelley and Frost, which are not included in their poems in the poets directory.

Your program should produce exactly this output:

```
$ ./identify_poet.pl poem?.txt
poem1.txt most resembles the work of John Keats (log-probability=-2720.
poem2.txt most resembles the work of Percy Bysshe Shelley (log-probabil
poem3.txt most resembles the work of Robert Frost (log-probability=-720
```

You may find it helpful to add a '-d' flag which provides debugging information (this is optional), for example:

```
$ ./identify_poet.pl -d poem2.txt
poem2.txt: log_probability of -777.0 for Percy Bysshe Shelley
poem2.txt: log_probability of -789.0 for Robert Frost
poem2.txt: log_probability of -796.3 for Samuel Taylor Coleridge
poem2.txt: log_probability of -803.7 for William Wordsworth
poem2.txt: log_probability of -804.9 for John Keats
poem2.txt: log_probability of -805.4 for William Blake
poem2.txt: log_probability of -805.4 for Elizabeth Barrett Browning
poem2.txt: log_probability of -808.2 for William Butler Yeats
poem2.txt: log_probability of -818.1 for Walt Whitman
poem2.txt: log_probability of -832.0 for Emily Dickinson
poem2.txt most resembles the work of Percy Bysshe Shelley (log-probabil
```

Hints: you may like to create simpler input to use in debugging, for example:

```
$ echo Andrew Rocks >andrew_rocks.txt
$ ./log_probability.pl Andrew
log((0+1)/ 50080) = -10.8214 Elizabeth Barrett Browning
log((0+1)/112882) = -11.6341 Emily Dickinson
log((0+1)/ 62844) = -11.0484 John Keats
log((0+1)/ 53400) = -10.8856 Percy Bysshe Shelley
log((0+1)/ 21698) =  -9.9850 Robert Frost
log((0+1)/ 38123) = -10.5486 Samuel Taylor Coleridge
log((0+1)/116941) = -11.6694 Walt Whitman
log((0+1)/ 40751) = -10.6152 William Blake
log((0+1)/121198) = -11.7052 William Butler Yeats
log((10+1)/117162) =  -9.2734 William Wordsworth
$ ./log_probability.pl Rocks
log((0+1)/ 50080) = -10.8214 Elizabeth Barrett Browning
log((0+1)/112882) = -11.6341 Emily Dickinson
log((10+1)/ 62844) =  -8.6505 John Keats
log((20+1)/ 53400) =  -7.8410 Percy Bysshe Shelley
log((3+1)/ 21698) =  -8.5987 Robert Frost
log((4+1)/ 38123) =  -8.9391 Samuel Taylor Coleridge
log((19+1)/116941) =  -8.6737 Walt Whitman
log((8+1)/ 40751) =  -8.4180 William Blake
log((8+1)/121198) =  -9.5080 William Butler Yeats
log((49+1)/117162) =  -7.7593 William Wordsworth
$ ./identify_poet.pl -d andrew_rocks.txt
andrew_rocks.txt: log_probability of -17.0 for William Wordsworth
andrew_rocks.txt: log_probability of -18.6 for Robert Frost
andrew_rocks.txt: log_probability of -18.7 for Percy Bysshe Shelley
andrew_rocks.txt: log_probability of -19.0 for William Blake
andrew_rocks.txt: log_probability of -19.5 for Samuel Taylor Coleridge
andrew_rocks.txt: log_probability of -19.7 for John Keats
andrew_rocks.txt: log_probability of -20.3 for Walt Whitman
andrew_rocks.txt: log_probability of -21.2 for William Butler Yeats
andrew_rocks.txt: log_probability of -21.6 for Elizabeth Barrett Browni
andrew_rocks.txt: log_probability of -23.3 for Emily Dickinson
andrew_rocks.txt most resembles the work of William Wordsworth (log-pro
```

Sample solution for identify_poet.pl

```
#!/usr/bin/perl -w
$debug = 0;
foreach $file (glob "poems/*") {
    my $poet = $file;
    $poet =~ s/.*\///;
    $poet =~ s/\.txt$//;
    $poet =~ s/_/ /g;
    open my $f, '<', $file or die "can not open $file: $!";;
    while (<$f>) {
        tr/A-Z/a-z/;
        foreach (/[a-z]+/g) {
            $frequency{$poet}{$_}++;
            $n_words{$poet}++;
        }
    }
    close $f;
}
@poets = keys %frequency;
foreach $file (@ARGV) {
    my %log_probability;
    if ($file eq '-d') {
        $debug = 1;
        next;
    }
    open my $f, '<', $file or die "can not open $file: $!";
    while (<$f>) {
        tr/A-Z/a-z/;
        foreach $word (/[a-z]+/g) {
            foreach $poet (@poets) {
                $log_probability{$poet} += log((($frequency{$poet}{$word}||0) + 1)/$n_words{$poet});
            }
        }
    }
    close $f;
    @sorted_poets = sort {$log_probability{$b} <=> $log_probability{$a}} @poets;
    if ($debug) {
        foreach $poet (@sorted_poets) {
            printf "%s: log_probability of %.1f for %s\n", $file, $log_probability{$poet}, $poet;
        }
    }
    printf "%s most resembles the work of %s (log-probability=%.1f)\n", $file, $sorted_poets[0], $log_p
}
```

As usual:

```
$ ~cs2041/bin/autotest lab07 identify_poet.pl
$ git add identify_poet.pl
$ git commit -a -m "logs are useful"
```

Challenge Exercise: Poetic Python

Implement the above exercises in Python.
The example Python scripts (/~cs2041/code/python/code_examples.html) and links to external Python resources should help - but you will need more info - Google is your friend.

Hints for `total_words.py` & `count_word.py` :

This loop executes for each line of stdin:

```
import sys
for line in sys.stdin:
    print line
```

The function `re.split` or the function `re.findall` could be used to separate words.

Hints for `frequency.py` , `log_probability.py` & `identify_poet.py` :

Beware Python dicts need a slightly different approach to Perrl hashes, and also Perl & Python division have different semantics.

This loop executes once for each `.txt` file in the directory `poets` .

```
import glob
for file in glob.glob("poems/*.txt"):
    print file
```

You might find `math.log` , `sorted` , `re.sub` and `collections.defaultdict` useful.

Solution for total_words.py using re.split

```
#!/usr/bin/python

import re, sys

count = 0
for line in sys.stdin:
    words = re.split(r'[^a-zA-Z]+', line)
    for word in words:
        if word:
            count += 1

print("%d words" % count)
```

Solution for total_words.py using re.findall

```
#!/usr/bin/python

import re, sys

count = 0
for line in sys.stdin:
    for word in re.findall(r'[a-z]+', line, re.I):
        count += 1

print("%d words" % count)
```

More Pythonic solution for total_words.py using re.findall

```
#!/usr/bin/python

import sys, re

lines = "".join(l for l in sys.stdin).lower()
words = re.findall(r'[a-z]+', lines, flags=re.I)
print("%d words" % len(words))
```

Sample solution for count_word.py

```
#!/usr/bin/python

import re, sys

if len(sys.argv) != 2:
    sys.stderr.write("Usage %s: <word>\n" % sys.argv[0])
    sys.exit(1)

specified_word = sys.argv[1].lower()

count = 0
for line in sys.stdin:
    line = line.lower()
    words = re.split(r'[^a-z]+', line)
    for word in words:
        if word == specified_word:
            count += 1

print("%s occurred %d times" % (specified_word, count))
```

More Pythonic solution for count_word.py using re.findall

```
#!/usr/bin/python

import sys, re

word = sys.argv[1].lower()
lines = "".join(l for l in sys.stdin).lower()
words = re.findall(r'[a-z]+', lines, flags=re.I)
print("%s occurred %d times" % (word, words.count(word)))
```

Sample solution for frequency.py

```
#!/usr/bin/python
import sys, glob, re, collections

frequency = {}
n_words = collections.defaultdict(int)
for file in glob.glob("poems/*"):
    poet = re.sub(r'.*/', '', file)
    poet = re.sub(r'.txt$', '', poet)
    poet = re.sub(r'_', ' ', poet)
    frequency[poet] = collections.defaultdict(int)
    for line in open(file):
        line = line.lower()
        for word in re.findall(r'[a-z]+', line, re.I):
            frequency[poet][word] += 1
            n_words[poet] += 1

for word in sys.argv[1:]:
    word = word.lower()
    for poet in sorted(frequency.keys()):
        f = frequency[poet][word]
        n = n_words[poet]
        print("%4d/%6d = %.9f %s" % (f, n, f/float(n), poet))
```

More Pythonic solution for frequency.py

```
#!/usr/bin/python

import sys, glob, re, collections, os

frequency = {}
n_words = collections.Counter()
for f in glob.glob("poems/*.txt"):
    # remove directory, extension, and fix the underscores
    poet = os.path.splitext(os.path.basename(f))[0].replace('_', ' ')
    lines = open(f).read().lower()
    words = re.findall(r'[a-z]+', lines, flags=re.I)
    frequency[poet] = collections.Counter(words)
    n_words[poet] = len(words)

for word in sys.argv[1:]:
    for poet in sorted(frequency):
        f = frequency[poet][word.lower()]
        n = n_words[poet]
        print("%4d/%6d = %.9f %s" % (f, n, float(f) / n, poet))
```

Sample solution for log_probability.py

```
#!/usr/bin/python
import sys, glob, re, collections, math

frequency = {}
n_words = collections.defaultdict(int)
for file in glob.glob("poems/*"):
    poet = re.sub(r'.*/', '', file)
    poet = re.sub(r'.txt$', '', poet)
    poet = re.sub(r'_', ' ', poet)
    frequency[poet] = collections.defaultdict(int)
    for line in open(file):
        line = line.lower()
        for word in re.findall(r'[a-z]+', line, re.I):
            frequency[poet][word] += 1
            n_words[poet] += 1

for word in sys.argv[1:]:
    word = word.lower()
    for poet in sorted(frequency.keys()):
        f = frequency[poet][word]
        n = n_words[poet]
        print("log((%d+1)/%6d) = %8.4f %s" % (f, n, math.log((f+1)/float(n)), poet))
```

More Pythonic solution for log_probability.py

```
#!/usr/bin/python

import sys, glob, re, collections, os, math

frequency = {}
n_words = {}
for f in glob.glob("poems/*.txt"):
    # remove directory, extension, and fix the underscores
    poet = os.path.splitext(os.path.basename(f))[0].replace('_', ' ')
    lines = open(f).read().lower()
    words = re.findall(r'[a-z]+', lines, flags=re.I)
    frequency[poet] = collections.Counter(words)
    n_words[poet] = len(words)

for word in sys.argv[1:]:
    for poet in sorted(frequency):
        f = frequency[poet][word.lower()]
        n = n_words[poet]
        print("log((%d+1)/%6d) = %8.4f %s" % (f, n, math.log(float(f + 1) / n), poet))
```

Sample solution for identify_poet.py

```
#!/usr/bin/python
import sys, glob, re, collections, math

frequency = {}
n_words = collections.defaultdict(int)
for file in glob.glob("poems/*"):
    poet = re.sub(r'.*/', '', file)
    poet = re.sub(r'.txt$', '', poet)
    poet = re.sub(r'_', ' ', poet)
    frequency[poet] = collections.defaultdict(int)
    for line in open(file):
        line = line.lower()
        for word in re.findall(r'[a-z]+', line, re.I):
            frequency[poet][word] += 1
            n_words[poet] += 1

poets = list(frequency.keys())
debug = 0
for file in sys.argv[1:]:
    if file == '-d':
        debug = 1
        continue
    log_probability = collections.defaultdict(float)
    for line in open(file):
        line = line.lower()
        for word in re.findall(r'[a-z]+', line, re.I):
            for poet in poets:
                log_probability[poet] += math.log(((frequency[poet][word] + 1)/float(n_words[poet])))
    sorted_poets = sorted(poets, key=lambda p: -log_probability[p])
    if debug:
        for poet in sorted_poets:
            print("%s: log_probability of %.1f for %s" % (file, log_probability[poet], poet))
    print("%s most resembles the work of %s (log-probability=%.1f)" % (file, sorted_poets[0], log_proba
```

More Pythonic solution for identify_poet.py

```
#!/usr/bin/python

import sys, glob, re, collections, os, math

frequency = {}
for f in glob.glob("poems/*.txt"):
    lines = open(f).read().lower()
    author = os.path.splitext(os.path.basename(f))[0].replace('_', ' ')
    words = re.findall(r'[a-z]+', lines, flags=re.S | re.M | re.I)
    count = collections.Counter(words)
    # we need to use a default value to capture at creation time, not runtime
    # the following line just says the dictionary defaults to log(1/|words|)
    frequency[author] = collections.defaultdict(lambda x=math.log(float(1) / len(words)): x)
    for word, occurances in count.items():
        frequency[author][word] = math.log(float((occurances + 1)) / len(words))

for f in sys.argv[1:]:
    lines = open(f).read().lower()
    words = re.findall(r'[a-z]+', lines, flags=re.I)
    prob = {}
    for author, freq in frequency.items():
        prob[author] = sum(freq[word] for word in words)
    likely = max(prob, key=prob.get) # this is how you sort hashes by value in python
    print("%s most resembles the work of %s (log-probability=%.1f)" % (f, likely, prob[likely]))
```

As usual:

```
$ ~cs2041/bin/autotest lab07 total_words.py
$ ~cs2041/bin/autotest lab07 count_word.py
$ ~cs2041/bin/autotest lab07 frequency.py
$ ~cs2041/bin/autotest lab07 log_probability.py
$ ~cs2041/bin/autotest lab07 identify_poet.py
$ git add total_words.py count_word.py frequency.py log_probability.py identify_poet.py
$ git commit -a -m "python is poetic"
```

Testing

Rember to do your own testing as well as the autotest tests are available for this lab.

To run all tests:

```
$ ~cs2041/bin/autotest lab07
```

You can run a single test if you also pass the test label as the second argument to autotest. For example, to run just test `total_words_5` type:

```
$ ~cs2041/bin/autotest lab07 total_words_5
```

You can also tell autotest to the code you have commited to gitlab

```
$ ~cs2041/bin/autotest lab07 -gitlab
```

or a particular gitlab commit

```
$ ~cs2041/bin/autotest lab07 9bfa2c5a
```

Finalising

You must show your solutions to your tutor and be able to explain how they work. Once your tutor has discussed your answers with you, you should submit them using:

```
$ give cs2041 lab07 total_words.pl count_word.pl frequency.pl log_probability.pl identify_poet.pl [total_v
```

Whether you discuss your solutions with your tutor this week or next week, you must submit them before the above deadline.

Gitlab - More Information

I expect most students will just work in their CSE account and push work to gitlab.cse.unsw.edu.au from there, but you can try setting up a git repository on your home machine and pushing work to gitlab.cse.unsw.edu.au from there.

If you do so you'll want to use git's pull command to update the repository in your CSE account.

```
$ git pull
Unpacking objects: 100% (3/3), done.
From gitlab@gitlab.cse.unsw.EDU.AU/z5555555/16s2-comp2041-labs
   226cddf..e64fee9  master     -> origin/master
Updating 226cddf..e64fee9
Fast-forward
 total_words.pl |    1 +
 1 file changed, 1 insertion(+)
```

If ssh access doesn't work, you can also use https to access gitlab using a URL equivalent to `https://gitlab.cse.unsw.edu.au/z5555555/16s2-comp2041-labs.git` (replace 5555555 with your student number) and use your z-id & zPass.

```
$ git remote set-url origin https://gitlab.cse.unsw.edu.au/z5555555/16s2-comp2041-labs.git
$ git push
Username for 'https://gitlab.cse.unsw.EDU.AU':  z5555555
Password for 'https://z5555555@gitlab.cse.unsw.EDU.AU':  zPass
```