# Week 07 Laboratory Exercises

## Objectives

- Proficiency at text processing in Perl.
- Understanding multi-dimensional hashes.
- Explore a simple machine learning algorithm.

---

## Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

---

## Getting Started

Create a new directory for this lab called `lab07`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab07
$ cd lab07
$ 2041 fetch lab07
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

---

EXERCISE:

## How many words in standard input?

In these exercises you will work with a dataset containing sing lyrics.

This zip file contains the lyrics of the songs of 10 well known artists.

```
wget -q https://cgi.cse.unsw.edu.au/~cs2041/20T2/activities/total_words/lyrics.zip
unzip lyrics.zip
Archive:  lyrics.zip
   creating: lyrics/
  inflating: lyrics/David_Bowie.txt
  inflating: lyrics/Adele.txt
  inflating: lyrics/Metallica.txt
  inflating: lyrics/Rage_Against_The_Machine.txt
  inflating: lyrics/Taylor_Swift.txt
  inflating: lyrics/Keith_Urban.txt
  inflating: lyrics/Ed_Sheeran.txt
  inflating: lyrics/Justin_Bieber.txt
  inflating: lyrics/Rihanna.txt
  inflating: lyrics/Leonard_Cohen.txt
  inflating: song0.txt
  inflating: song1.txt
  inflating: song2.txt
  inflating: song3.txt
  inflating: song4.txt
```

The lyrics for each song have been re-ordered to avoid copyright concerns.

The zip file also contains lyrics from 5 songs where we don't know the artists.

```
$ cat song0.txt
I've made up my mind,  Don't need to think it over,
If I'm wrong I am right,
Don't need to look no further,
This ain't lust,
I know this is love but,

If I tell the world,
I'll never say enough,
Cause it was not said to you,
And that's exactly what I need to do,
If I'm in love with you,
$ cat song1.txt
Come Mr. DJ song pon de replay
Come Mr. DJ won't you turn the music up
All the gal pon the dance floor wantin' some more what
Come Mr. DJ won't you turn the music up
$ cat song2.txt
And they say
She's in the class A team
Stuck in her daydream
```

They are each from one of the artists in the dataset but they are not from a song in the dataset.

To start on this analysis write a Perl script `total_words.pl` which counts the total number of words found 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 <lyrics/Justin_Bieber.txt
46589 words
$ ./total_words.pl <lyrics/Metallica.txt
38096 words
$ ./total_words.pl <lyrics/Rihanna.txt
53157 words
```

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

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest total_words
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab07_total_words total_words.pl
```

before **Tuesday 21 July 18:00** to obtain the marks for this lab exercise.

EXERCISE:

## How many times does a word occur in standard input

Write a Perl script `count_word.pl` which counts the number of times a specified word is found 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.

Your: program should ignore the case of words.

For example:

```
$ ./count_word.pl death <lyrics/Metallica.txt
death occurred 69 times
$ ./count_word.pl death <lyrics/Justin_Bieber.txt
death occurred 0 times
$ ./count_word.pl love <lyrics/Ed_Sheeran.txt
love occurred 218 times
$ ./count_word.pl love <lyrics/Rage_Against_The_Machine.txt
love occurred 4 times
```

**Hint**: modify the code from the last exercise.

**Hint**: the Perl functions `uc` & `lc` convert strings to lowercase & uppercase respectively.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest count_word
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab07_count_word count_word.pl
```

before **Tuesday 21 July 18:00** to obtain the marks for this lab exercise.

---

EXERCISE:

## Do you use that word often?

Write a Perl script `frequency.pl` which prints the frequency with which each artist uses a word specified as an argument.

So if Justin Bieber uses the word **"love"** 493 times in the 46583 words of his songs, then its frequency is `493/46583 = 0.0105832599875491`.

For example:

```
$ ./frequency.pl love
 165/ 16359 = 0.010086191 Adele
 189/ 34080 = 0.005545775 David Bowie
 218/ 18207 = 0.011973417 Ed Sheeran
 493/ 46589 = 0.010581897 Justin Bieber
 217/ 27016 = 0.008032277 Keith Urban
 212/ 26192 = 0.008094075 Leonard Cohen
  57/ 38096 = 0.001496220 Metallica
   4/ 18985 = 0.000210693 Rage Against The Machine
 494/ 53157 = 0.009293226 Rihanna
  89/ 26188 = 0.003398503 Taylor Swift
```

So of these artists, Ed Sheeran uses the word **"love"** most frequently. If you choose a word a randomly from an Ed Sheeran song the probability it will be "love" is just over in 1 in a hundred (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 artist and word to store the word counts.

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

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

**Hint**: reuse code from the last exercise.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest frequency
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab07_frequency frequency.pl
```

before **Tuesday 21 July 18:00** to obtain the marks for this lab exercise.

---

EXERCISE:

## When numbers get very small, logarithms are your friend

Now suppose we have the song line **"truth is beauty"**. Given that David Bowie uses the word **"truth"** with frequency 0.000146727 and the word **"is"** with frequency 0.005898407, the word **"beauty"** with frequency 0.000264108; we can estimate the probability of

Bowie writing the phrase **"truth is beauty"** as:

```
0.000146727 * 0.005898407 * 0.000264108 = 2.28573738067596e-10
```

We could similarly estimate probabilities for each of the other 9 artists, and then determine which of the 10 artists is most likely to sing **"truth is beauty"** (it's Leonard Cohen).

A sidenote: we are actually making a large simplifying assumption in calculating this probability. It is often called the **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 do this by adding the log of the probabilities of each word. For example, you calculate the log-probability of Bowie singing the phrase **"Truth is beauty."** like this:

```
log(0.000146727) + log(0.005898407) + log(0.000264108) = -22.1991622527613 = log(2.28573738067596e-10)
```

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

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

```
$ ./frequency.pl fear
   2/ 16359 = 0.000122257 Adele
  13/ 34080 = 0.000381455 David Bowie
   0/ 18207 = 0.000000000 Ed Sheeran
  10/ 46589 = 0.000214643 Justin Bieber
   0/ 27016 = 0.000000000 Keith Urban
   4/ 26192 = 0.000152718 Leonard Cohen
  39/ 38096 = 0.001023730 Metallica
  26/ 18985 = 0.001369502 Rage Against The Machine
   3/ 53157 = 0.000056437 Rihanna
   3/ 26188 = 0.000114556 Taylor Swift
```

It is not useful to assume there is zero probability that **Ed Sheeran** would use the word **fear** in a song even though he hasn'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 Ed Sheeran using the word **fear** as (0+1)/18205 and the probability of Metallica using the word **fear** as (39+1)/38082. This is a simple version of Additive smoothing.

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

```
$ ./log_probability.pl fear
log((2+1)/ 16359) =  -8.6039 Adele
log((13+1)/ 34080) =  -7.7974 David Bowie
log((0+1)/ 18207) =  -9.8096 Ed Sheeran
log((10+1)/ 46589) =  -8.3512 Justin Bieber
log((0+1)/ 27016) = -10.2042 Keith Urban
log((4+1)/ 26192) =  -8.5638 Leonard Cohen
log((39+1)/ 38096) =  -6.8590 Metallica
log((26+1)/ 18985) =  -6.5556 Rage Against The Machine
log((3+1)/ 53157) =  -9.4947 Rihanna
log((3+1)/ 26188) =  -8.7868 Taylor Swift
```

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"`)

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest log_probability
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab07_log_probability log_probability.pl
```

before **Tuesday 21 July 18:00** to obtain the marks for this lab exercise.

---

EXERCISE:

## Who sang those words?

Write a Perl script `identify_artist.pl` that given 1 or more files, each containing part of song), prints the most likely artist to have sung those words.

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

Your program should produce exactly this output:

```
$ ./identify_artist.pl song?.txt
song0.txt most resembles the work of Adele (log-probability=-352.4)
song1.txt most resembles the work of Rihanna (log-probability=-254.9)
song2.txt most resembles the work of Ed Sheeran (log-probability=-206.6)
song3.txt most resembles the work of Justin Bieber (log-probability=-1089.8)
song4.txt most resembles the work of Leonard Cohen (log-probability=-493.8)
```

**Hint:** only read each file once. Store the data in a (2-dimensional) hash. If you read the files many times your program will be very slow and exceed autotest time limits.

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

```
$ ./identify_artist.pl -d song2.txt
song2.txt: log_probability of -206.6 for Ed Sheeran
song2.txt: log_probability of -210.8 for Adele
song2.txt: log_probability of -211.5 for Taylor Swift
song2.txt: log_probability of -211.7 for Keith Urban
song2.txt: log_probability of -215.0 for Leonard Cohen
song2.txt: log_probability of -215.4 for Rage Against The Machine
song2.txt: log_probability of -215.7 for David Bowie
song2.txt: log_probability of -217.2 for Justin Bieber
song2.txt: log_probability of -222.2 for Metallica
song2.txt: log_probability of -223.4 for Rihanna
song2.txt most resembles the work of Ed Sheeran (log-probability=-206.6)
```

**Hint:** 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)/ 16359) =  -9.7025 Adele
log((0+1)/ 34080) = -10.4365 David Bowie
log((0+1)/ 18207) =  -9.8096 Ed Sheeran
log((0+1)/ 46589) = -10.7491 Justin Bieber
log((0+1)/ 27016) = -10.2042 Keith Urban
log((0+1)/ 26192) = -10.1732 Leonard Cohen
log((0+1)/ 38096) = -10.5479 Metallica
log((0+1)/ 18985) =  -9.8514 Rage Against The Machine
log((0+1)/ 53157) = -10.8810 Rihanna
log((0+1)/ 26188) = -10.1731 Taylor Swift
$ ./log_probability.pl Rocks
log((0+1)/ 16359) =  -9.7025 Adele
log((10+1)/ 34080) =  -8.0386 David Bowie
log((0+1)/ 18207) =  -9.8096 Ed Sheeran
log((1+1)/ 46589) = -10.0560 Justin Bieber
log((0+1)/ 27016) = -10.2042 Keith Urban
log((0+1)/ 26192) = -10.1732 Leonard Cohen
log((1+1)/ 38096) =  -9.8547 Metallica
log((0+1)/ 18985) =  -9.8514 Rage Against The Machine
log((2+1)/ 53157) =  -9.7824 Rihanna
```

**Hint:** if a word appears multiple times its log-probability needs to be summed multiple times.

```
echo echo echo >echo.txt
```

```
$ cat echo.txt
echo echo
$ ./log_probability.pl echo
log((0+1)/ 16359) =  -9.7025 Adele
log((0+1)/ 34080) = -10.4365 David Bowie
log((0+1)/ 18207) =  -9.8096 Ed Sheeran
log((0+1)/ 46589) = -10.7491 Justin Bieber
log((0+1)/ 27016) = -10.2042 Keith Urban
log((0+1)/ 26192) = -10.1732 Leonard Cohen
log((0+1)/ 38096) = -10.5479 Metallica
log((14+1)/ 18985) =  -7.1434 Rage Against The Machine
log((0+1)/ 53157) = -10.8810 Rihanna
log((1+1)/ 26188) =  -9.4799 Taylor Swift
$ ./identify_artist.pl -d echo.txt
echo.txt: log_probability of -14.3 for Rage Against The Machine
echo.txt: log_probability of -19.0 for Taylor Swift
echo.txt: log_probability of -19.4 for Adele
echo.txt: log_probability of -19.6 for Ed Sheeran
echo.txt: log_probability of -20.3 for Leonard Cohen
echo.txt: log_probability of -20.4 for Keith Urban
echo.txt: log_probability of -20.9 for David Bowie
```

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest identify_artist
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab07_identify_artist identify_artist.pl
```

before **Tuesday 21 July 18:00** to obtain the marks for this lab exercise.

---

CHALLENGE EXERCISE:

## A Perl Program that Prints Perl that Prints Perl that ...

Write a Perl program `perl_print_n.pl` which is given a two arguments, an integer **n** and a string.

If **n** is 1 it should output a Perl program which prints the string.

If **n** is 2 it should output a Perl program which prints a Perl program which prints the string.

If **n** is 2 it should output a Perl program which prints a Perl program which prints a Perl program which prints the string.

If **n** is 3 it should output a Perl program which prints a Perl program which prints a Perl program which prints a Perl program which prints the string.

And so on for any value of **n**.

For example:

```
$ ./perl_print_n.pl 1 'Perl that prints Perl'
print "Perl that prints Perl\n"
$ ./perl_print_n.pl 2 'Perl that prints Perl that Prints Perl'|perl|perl
Perl that prints Perl that Prints Perl
$ ./perl_print_n.pl 2 'Perl that ....'|perl|perl
Perl that ....
$ ./perl_print_n.pl 4 'Andrew Rocks!'|perl|perl|perl|perl
Andrew Rocks!
$ ./perl_print_n.pl 10 'I love COMP(2041|9044)!'|perl|perl|perl|perl|perl|perl|perl|perl|perl|perl
I love COMP(2041|9044)!
```

You can assume n is a positive integer.

You can assume the string contains only ASCII characters.

You can not make other assumptions about the characters in the string.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest perl_print_n
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab07_perl_print_n perl_print_n.pl
```

before **Tuesday 21 July 18:00** to obtain the marks for this lab exercise.

---

# Submission

When you are finished each exercises make sure you submit your work by running `give`.

You can run `give` multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Tuesday 21 July 18:00** to submit your work.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

You check the files you have submitted here.

Automarking will be run by the lecturer several days after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

After automarking is run by the lecturer you can view your results here. The resulting mark will also be available via give's web interface.

## Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

---