Week 08 Laboratory Exercises

Objectives

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

Preparation

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

Getting Started

Set up for the lab by creating a new directory called `lab08` and changing to this directory.

```
$ mkdir lab08
$ cd lab08
```

There are some provided files for this lab which you can fetch with this command:

```
$ 2041 fetch lab08
```

If you're not working at CSE, you can download the provided files 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 dataset contains the lyrics of the songs of 10 well-known artists.

```
$ 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 dataset 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
```

Each is from one of the artists in the dataset but they are not from a song in the dataset.

As a first step in this analysis, write a Python script `total_words.py` which counts the total number of words in its stdin.

For the purposes of this program (and the following programs) we will define a word to be a maximal, non-empty, contiguous, sequence 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")`

```
$ ./total_words.py < lyrics/Justin_Bieber.txt
46589 words
$ ./total_words.py < lyrics/Metallica.txt
38096 words
$ ./total_words.py < lyrics/Rihanna.txt
53157 words
```

HINT:

If your word counts are a little too high, you might be counting empty strings.

NOTE:

A word is defined for these exercises to be a maximal, non-empty, contiguous, sequence of alphabetic characters (`[a-zA-Z]`).

You can assume your input is only ASCII.

Your answer must be Python only. You can not use other languages such as Shell, Perl or C.

You may not run external programs.

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 lab08_total_words total_words.py
```

before **Monday 24 July 12:00 (midday)** (2023-07-24 12:00:00) to obtain the marks for this lab exercise.

EXERCISE:

How many times does a word occur in standard input

Write a Python script `count_word.py` that counts the number of times a specified word is found in its stdin

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

Your program should ignore the case of words.

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

HINT:

Start with your code from the previous activity.

NOTE:

A word is defined for these exercises to be a maximal, non-empty, contiguous, sequence of alphabetic characters (`[a-zA-Z]`).

You can assume your input is only ASCII.

Your answer must be Python only. You can not use other languages such as Shell, Perl or C.

You may not run external programs.

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 lab08_count_word count_word.py
```

before **Monday 24 July 12:00 (midday)** (2023-07-24 12:00:00) to obtain the marks for this lab exercise.

EXERCISE:

Do you use that word often?

Write a Python script `frequency.py` thar 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` .

```
$ ./frequency.py 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
$ ./frequency.py death
   1/ 16359 = 0.000061128 Adele
   9/ 34080 = 0.000264085 David Bowie
   3/ 18207 = 0.000164772 Ed Sheeran
   0/ 46589 = 0.000000000 Justin Bieber
   1/ 27016 = 0.000037015 Keith Urban
  16/ 26192 = 0.000610874 Leonard Cohen
  69/ 38096 = 0.001811214 Metallica
  23/ 18985 = 0.001211483 Rage Against The Machine
   0/ 53157 = 0.000000000 Rihanna
```

Make sure your Python script produces exactly the output above.

HINT:

Start with your code from the previous activity.

A print like this will produce the correct output format:

```
print(f"{var1:4}/{var2:6} = {var3:.9f} {var4}")
```

Use a dict of dicts indexed by artist then word to store the word counts.

Use the `glob` module to find all the files that match a glob string.

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

```
for file in glob.glob("lyrics/*.txt"):
    print(file);
```

NOTE:

A word is defined for these exercises to be a maximal, non-empty, contiguous, sequence of alphabetic characters (`[a-zA-Z]`).

You can assume your input is only ASCII.

Your answer must be Python only. You can not use other languages such as Shell, Perl or C.

You may not run external programs.

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 lab08_frequency frequency.py
```

before **Monday 24 July 12:00 (midday)** (2023-07-24 12:00:00) 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 0.000146714
the word **"is"** with frequency 0.005897887
the word **"beauty"** with frequency 0.000264085
we can estimate the probability of Bowie writing the phrase **"truth is beauty"** as:

```
0.000146714 * 0.005897887 * 0.000264085 = 2.2851343535638401e-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 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(0.000146714) + log(0.005897887) + log(0.000264085) = -22.19942610926425
```

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.

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)/38096.
This is a simple version of [Additive smoothing](#).

Write a Python script `log_probability.py` which given a phrase (sequence of words) as arguments, prints the estimated log of the probability that each artist would use this phrase.

```
$ ./log_probability.py truth is beauty
 -23.11614 Adele
 -21.90679 David Bowie
 -23.10075 Ed Sheeran
 -21.70202 Justin Bieber
 -23.45248 Keith Urban
 -18.58417 Leonard Cohen
 -21.08903 Metallica
 -21.98171 Rage Against The Machine
 -22.51582 Rihanna
 -24.40992 Taylor Swift
$ ./log_probability.py death and taxes
 -22.64301 Adele
 -22.42756 David Bowie
 -21.66227 Ed Sheeran
 -25.56650 Justin Bieber
 -23.20281 Keith Urban
 -20.97467 Leonard Cohen
 -20.90589 Metallica
 -20.26248 Rage Against The Machine
 -25.84396 Rihanna
```

Make sure your output matches the above exactly

HINT:

Start with your code from the previous activity.

A print like this will produce the correct output format:

```
print(f"{var1:10.5f} {var2}")
```

NOTE:

Use the natural logarithm (base e) - `math.log` returns this by default, if you don't specify a base.
A word is defined for these exercises to be a maximal, non-empty, contiguous, sequence of alphabetic characters (`[a-zA-Z]`).

You can assume your input is only ASCII.

Your answer must be Python only. You can not use other languages such as Shell, Perl or C.

You may not run external programs.

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 lab08_log_probability log_probability.py
```

before **Monday 24 July 12:00 (midday)** (2023-07-24 12:00:00) to obtain the marks for this lab exercise.

EXERCISE:

Who sang those words?

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

For each file given as argument, you should go through all artists and for each calculate the log-probability that the artist sung those words.

You calculate the log-probability that the artist sung the words in theilfe, by for each word in the file calculating the log-probability of that artist using that word, and summing all the the log-probabilities.

You should print the artist with the highest log-probability.

Your program should produce exactly this output:

```
$ ./identify_artist.py 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:

If a word is used is used multiplied times in a file, its log-probablity should be added multiple times.

You do not need to use **glob**. **song?.txt** in the above example is expanded by the Shell. The filenames are passed as separate argument in **sys.argv**.

Start with your code from the previous activity.

NOTE:

A word is defined for these exercises to be a maximal, non-empty, contiguous, sequence of alphabetic characters (`[a-zA-Z]`).

You can assume your input is only ASCII.

Your answer must be Python only. You can not use other languages such as Shell, Perl or C.

You may not run external programs.

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 lab08_identify_artist identify_artist.py
```

before **Monday 24 July 12:00 (midday)** (2023-07-24 12:00:00) 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 **Week 9 Monday 12:00:00 (midday)** 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
```