Lab 2 - Raku

Due Oct 26 by 2:59am **Points** 100

The Most Probable Song Title

In this programming lab you will practice imperative programming in the language **Raku (https://raku.org/resources/)** . This lab builds on your regular expression tasks from **Lab 1**.

In this lab you will create an automatic song title generator. Given a seed (first) word, your program will produce the most probably song title, based on the statistical occurrences of word pairs across the One Million Song Dataset.

Language: Raku

For this lab, you will continue using Raku. Download Raku from here: **https://rakudo.org/ (https://download.racket-lang.org/)** if you have not already.

The **Comma IDE** **(https://commaide.com/)** is built for Raku and has syntax highlighting: **https://commaide.com/download** **(https://commaide.com/download)** .

Download the Lab 2 Starter Pack: **381_lab2_starter_pack.zip** **(https://canvas.oregonstate.edu/courses/1862887/files/89343739/download?download_frd=1)**

Dataset

▶ lab will make use of a subset of a dataset containing a million song titles. This dataset is used in various machine learning experiments and is provided by the **Laboratory for the Recognition and Organization of Speech and Audio at Columbia University** **(http://labrosa.ee.columbia.edu/)** .

This file is available to you in the starter pack. The original file is **available here (http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/unique_tracks.txt)** . This file contains one millions lines and weighs in at 82 MB. You should probably avoid viewing it in a web browser.

Lab Tasks

Lab 2 builds on Lab 1. Finish Lab 1 and pass all test before proceeding to Lab 2. Make a copy of your `lab1.raku` and save it as `lab2.raku`. You will complete the functions designated as part of Lab 2.

Bigram Counts

A **bigram** (http://en.wikipedia.org/wiki/Bigram) is a sequence of two adjacent words in a text. The frequency distribution of bigrams, or more generally n-grams, in text data is commonly used in statistical natural language processing. Across this corpus of one million song titles, you will count all the bigram words.

First, you need split the title string into individual words. Next, you should use an associative array to keep track of these word pair counts. That is, for every word, you must keep track of the count for each word that follows it. More specifically, use the variable `%counts` as a hash of hashes. The first key is w_{i-1}, the second key is w_i , and the value is the count of the number of times that words w_{i-1} and w_i appear sequentially in song titles across the entire corpus.

Complete the function build bigrams. Here is some pseudocode that describes the task.

```
foreach title in tracks array
  split title into individual words
    if there is more than one word
      while there are adjacent word pairs
        update count for word pair by 1
      end while
    end if
end foreach
```

Most Common Word

Now you are going to built a probabilistic song title. First begin by completing the function "most common word" `mcw`. This function takes in one argument, some word, and returns the word that most often followed that word in the dataset.

When you first extract all possible successive words to the given seed word, sort the set of keys. When comparing counts and you encounter a tie, stick with the first word found. This ensures that you get the same results as the grading test cases.

You can now use the command `mcw WORD`, where `WORD` is the argument passed to the function.

Building a Song Title

Now you are going to use this `mcw` function to string together a song title. Beginning with a given starting word, write an iterative structure that strings together words that most commonly follow each other in the dataset. Continue until a word does not have a successive word in the dataset (check for whitespace or newline) or the count of words in your title reaches the variable `$SEQUENCE_LENGTH`. Complete the function `sequence`.

You can now use the command `sequence WORD` where `WORD` is the seed word.

Unwanted Recursion

As you examine your new song titles, you probably notice a lot of repeated phrases such as "of the world" and "ready for you". Think about why that happens.

This repetition happens because these words occurred very often in either order of each other in the corpus. We can fix this repetition by allowing each word to appear in the title only once.

Add a data structure to track the word history. The empty hash `%word_history` is provided as a global variable. Edit `sequence` so that every time a word is used in your title it is added it to word history. Treat `%word_history` as map to Booleans values (0 or 1) in which `%word_history{"word"}=1` indicates that the word `word` has been seen before. If a word has not been seen before `%word_history{"word"}` conveniently returns the default value of zero.

Next go back and edit the `mcw` function. Presently, it picks the most common word to follow the seed word. Modify this so that it picks the most common word only if it has not yet been used in the sequence. If the most common word has already been used, pick the next most common word, and so on.

Make sure you clear your word history every time that `sequence` is called. The history should start over for each new sequence.

Menu System

Menu system for Labs 1 and 2

command	argument	description
`build`		build the bigram model
`debug`	on	turn on debug mode
	off	turn off debug mode
`count`	tracks	counts tracks in `@tracks`
	words	counts words in `@tracks`
	characters	counts characters in `@tracks`
`filter`	title	extract title from original input
	comments	removes extra phrases
	punctuation	removes punctuation
	unicode	removes non-Unicode and whitespace
`load`	`FILE`	loads the given `FILE`

`length`	`INT`	update `$SEQUENCE LENGTH=INT`
`mcw`	`WORD`	print most common word to follow `WORD`
`name`		print sequence based on your name
`preprocess`		run all filter tasks and build
`print`		prints all tracks to file `tracks.out`
	`INT`	prints only `INT` tracks to `stdout`
`random`		print sequence based on random word
`stopwords`	on	filter stopwords on
	off	filter stopwords off
`sequence`	`WORD`	find sequence to follow `WORD`

Testing

In the directory `tests` you will find a number of pairs of files representing the given input and expected output of the test.

```
raku lab2.raku < ./tests/t01.in
```

The above line will execute `lab2.raku` piping in the commands given in file `t01.in`. You can compare your result with the expected output given in file `t01.out`.

Test `t01` checks `mcw`.

Test `t02` checks `sequence`.

Resources

This lab requires an independent study of the Raku language. You are encouraged to use any web tutorials and resources to learn Raku.

- **Raku Guide** (https://raku.guide/)
- **Raku Resources** (https://raku.org/resources/)
- **Raku Tutorial** (https://www.i-programmer.info/news/222-perl/13267-the-raku-beginner-tutorial.html)
- **Raku Documentation** (https://docs.raku.org/language.html)

Allow yourself plenty of time, and use patience, perseverance, and the Internet to debug your code.

Submission

Each student will complete and submit this assignment individually. Do not consult with others. However, you are encouraged to use the Internet to learn Raku.

You will submit only your Raku code file.

- `lab2.raku`

Do not submit `tracks.txt`.

Grading criteria

This project is worth 100 points. Include your name in your file. Format your code neatly using expected conventions for indentations. Use descriptive variable names. Comment your program heavily. Thoughtful and unique comments help protect you against accusations of violations of academic integrity.

Your program will be evaluated against a series of automatic tests using Gradescope. The grading tests will be the same as the provided tests but will examine a different subset of the one million tracks. Your grade is determined by the number of public and hidden tests your program passes.

Lab 2 Rubric			
Criteria	**Ratings**		**Pts**
Most Common Word ▶ s of the mcw function	**50 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	50 pts
Sequence Tests of the sequence function	**50 to >0.0 pts** **Full Marks**	**0 pts** **No Marks**	50 pts
			Total Points: 100