

"The Most Probable Song Title"

Ruby Programming Lab

Concepts of Programming Languages

CSCI 305, Spring 2018

Due Date: February 9, 2018 at Midnight

Ruby

For this lab you will be using Ruby. Although Ruby is Hybrid language which combines Scripting, Imperative, Functional, and Object-Oriented concepts, we will be focusing simply on the Scripting and Imperative elements. Thus, solutions utilizing Functional or OO capabilities will be heavily penalized.

You can install ruby using instructions found at the following sites:

- [Windows](#)
- [Mac](#)
- [Linux](#)

Concepts we will explore as a part of this Lab:

- Regular Expression -> See this [tutorial](#) and this [too](#)
- Ruby -> See this [tutorial](#)
- Procedural Languages

Step -1: Fork and Clone this Repository

1. Using the Fork button above fork your own version of this repository.
2. On your system, use your installation of Git to clone **your fork** of this repo on your local machine.
3. Once you have the repository cloned, you can continue on.

Step 0: Getting everything ready

1. Install Bundler

```
gem install bundler
```

2. Install project dependencies (from the project root directory)

```
bundler install --binstubs
```

Dataset

This lab will make use 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. I have added this dataset to the repository under the name `unique_tracks.txt`

In addition, I have created a subset of this dataset containing only song titles that begin with the letter "A". We will use this file for debugging and testing purposes. This can be found in the file `a_tracks.txt`.

File Templates

There are four files of concern in the project directory:

- `ruby_lab.rb` - This is the ruby template in which you will add code to complete this assignment. You are given some code, do not remove this code unless you are sure of what you are doing. If the tests do not pass due to your modifications, you will lose credit.
- `questions.txt` - This is the file in which you will answer identified questions as part of the lab.
- `a_tracks.txt` - A small testing data set which is a subset (only tracks with titles starting with "A") derived from the next file.
- `unique_tracks.txt` - The million record data set which we will use for the lab.

There are some other files:

- `spec/*spec.rb` - These are test files used by RSpec to evaluate your code.
- `.rspec` - This indicates to RSpec that this project can be tested
- `Gemfile` - provides the specification of dependencies for this project
- `README.md` - this markdown file

This program takes as input the dataset file. For example, I execute the program at the command line as follows:

```
$ ruby ruby_lab.rb unique_tracks.txt
```

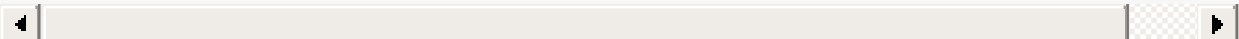
This initial template gives code to loop through each line of the file and prints out the line. You probably will not want to keep this line. Remember you use `ctrl+c` or `cmd+c` to cancel the execution of the program.

Pre-processing

Step 1: Extract song title

Each line contains a track id, song id, artist name, and the song title, such as:

```
TRWRJSX12903CD8446<SEP>SOBSFBU12AB018DBE1<SEP>Frank Sinatra<SEP>Everything Happens 1
```



You are only concerned with the last field, the song title. As your first task, you will write a regular expression that extracts the song title and stores it as the variable `title`. You will discard all other information.

You may find this site useful in debugging your regular expression: <https://regex101.com/>. It allows you to test your regular expressions on a block of text that you provide.

Step 2: Eliminate superfluous text

The song title, however, is quite noisy, often containing additional information beyond the song title. Consider this example:

```
Strangers in the Night (Remastered Album Version) [The Frank Sinatra Collection]
```

You need to perform some pre-processing in order to clean up the song titles. You will write a series of regular expressions that match a block of text and replace it with nothing.

Begin by writing a regular expression that matches a left parenthesis and any text that follows it. You need not match the right parenthesis explicitly. Replace the parenthesis and all text that follows it with nothing.

In the above Sinatra example, the modified title becomes `Strangers In The Night`.

Repeat this for patterns beginning with the left bracket, the left curly brace, and all the other characters listed below:

```
( [ { \ / _ - : " ` + = * feat.
```

Note that the above lists the left quote (on the tilde key above tab) and not the apostrophe (located left of the enter key). This is a very important distinction. We do not want to omit the apostrophe as it allows contractions.

Many of these characters have special meanings in Ruby. Make sure you properly escape symbols when necessary. Failing to escape characters properly will be the most common mistake made in this lab.

The last one listed above is an abbreviation `feat.` -- short for featuring and followed by artist information you do not need to retain. For example, `Sunbeam feat. Vishal Vaid` becomes `Sunbeam`.

In most cases, these symbols indicate additional information that need not concern us for this exercise. The above steps will very occasionally corrupt a valid song title that actually contains, for example, parentheses in the song title. Do not worry about these infrequent cases and uniformly carry out the procedure listed above. These steps will catch and fix the vast majority of irregularities in the song titles.

Step 3: Eliminate punctuation

Next, find and delete the following typical punctuation marks:

```
? , ! ; & @ % # |
```

Unlike before, delete only the symbol itself and leave all of the text that follows. Be sure to do a 'global' match in order to replace all instances of the punctuation mark. Be careful to match the period itself as the symbol "." has a special meaning in regular expressions. This is true for many of the symbols above. Again, refer to a list of escape characters specific to the language you selected.

Step 4: Filter out non-English characters

Lastly, ignore all song titles that contain a non-English character (e.g., á, ì, ö, etc.). (Hint: it may be easier to match titles that contain only English characters than to match titles that contain non-English characters). I define "English characters" to include the word meta-character definition (typically `\w` and `\s` in most languages) as well as the apostrophe character. This process will allow a few non-English song titles to creep through (e.g., *amore mio*), but will eliminate the majority of non-English titles.

Step 5: Set to lowercase

Convert all words in the sentence to lowercase. Each of these languages has a special function to do this for you.

Self-Check

In the `a_tracks` dataset, after all filtering steps, I find 52,760 valid song titles.

N.B.: If you are close to my number (within 10's), that is sufficient. If you are way off (i.e., 100+), you should double check your regular expressions.

This check can be performed using the following command:

On Mac or Linux:

```
rspec spec/self_check_1_spec.rb
```

On Windows:

```
rspec spec\self_check_1_spec.rb
```

Bi-gram Counts

A bigram is a sequence of two adjacent words in a text. The frequency distribution of bigrams in text(s) is commonly used in statistical natural language processing (see <http://en.wikipedia.org/wiki/Bigram>). Across this corpus of one million song titles, you will count all the bigram words.

First, you need to split the title string into individual words. Next, you should use one or more data structures 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. I strongly recommend you design your data structure for fast retrieval. Put some thought into which data structure to choose. Once you have decided, you can compare your choice to mine.

Self-Check

After you build and populate your bigram data structure, you can check yourself.

In the `a_tracks` dataset:

This check can be performed using the following command:

On Mac or Linux:

```
rspec spec/self_check_2_spec.rb
```

On Windows:

```
rspec spec\self_check_2_spec.rb
```

- The most common word to follow **"happy"** is **"now"**
- The most common word to follow **"sad"** is **"love"**
- The most common word to follow **"love"** is **"song"**
- There are 80 distinct words that follow the word **"love"**.
- The word **"song"** follows **"love"** 33 times.

Building a Song Title

Now you are going to build a probabilistic song title. First begin by creating a function "most common word" `mcw()`. This function will take in one argument, some word, and returns the word that most often followed that word in the dataset. If you find a tie, randomly select one value. For example, the line `puts mcw("computer")` should give you your answer to Question 4.

Now you are going to use this 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, or the count of words in your title reaches 20.

Lab Questions

Use your data structure(s) on the `unique_tracks` dataset to answer these and all subsequent Lab Questions.

To answer these questions execute the following command (in the terminal) from the root directory of your project:

On Mac or Linux

```
rspec spec/lab_quest_1_5_spec.rb -o lab_quest_1_5_output.txt
```

On Windows:

```
rspec spec\lab_quest_1_5_spec.rb -o lab_quest_1_5_output.txt
```

1. Which word most often follows the word **"happy"**?
2. Which word most often follows the word **"sad"**?
3. How many different (unique) words follow the word **"computer"**?
4. Which word most often follows the word **"computer"**?
5. How many times does this word follow **"computer"**?

User Control

Now add loop that repeatedly queries the user for a starting word until they choose to quit. I started it for you in the template. Your program will ask:

```
Enter a word [Enter 'q' to quit]:
```

For each word entered, use your code above to create a song title of 20 words (or less). Print out your newly designed song title. Repeat, querying the user for a new word.

Self-Check

For the `a_tracks` dataset:

This check can be performed using the following command:

On Mac or Linux:

```
rspec spec/self_check_3_spec.rb
```

On Windows:

```
rspec spec\self_check_3_spec.rb
```

- Using the seed word **"happy"**, you should get the title: happy now the world of the world of the world of the world of the world of the world of
- Using the seed word **"sad"**, you should get the title: sad love song for you ready for you ready for you ready for you ready for you ready for you ready
- Using the seed word **"computer"**, you should get the title: computer because no song titles in a_tracks contain the word **"computer"**

Lab Questions

To answer these questions execute the following command (in the terminal) from the root directory of your project, for questions 6-9. Question 10 should be answered in the questions.txt file:

On Mac or Linux

```
rspec spec/lab_quest_6_9_spec.rb -o lab_quest_6_9_output.txt
```

On Windows:

```
rspec spec\lab_quest_6_9_spec.rb -o lab_quest_6_9_output.txt
```

1. Using the starting word **"happy"**, what song title do you get?
2. Using the starting word **"sad"**, what song title do you get?
3. Using the starting word **"hey"**, what song title do you get?
4. Using the starting word **"little"**, what song title do you get?
5. Try a few other words. What problem(s) do you see? Which phrase do you most often find recurring in these titles?

Stop Words

Next try to fix the aforementioned problem(s) you observed in Question 10. In NLP, stop words are

common words that are often filtered out, such as common function words and articles. Before taking your bigram counts, filter out the following common stop words from the song title:

```
a, an, and, by, for, from, in, of, on, or, out, the, to, with
```

Lab Questions

To answer these questions execute the following command (in the terminal) from the root directory of your project, for questions 11-13. Questions 14 and 15 should be answered the questions.txt file.:

On Mac or Linux

```
rspec spec/lab_quest_11_13_spec.rb -o lab_quest_11_13_output.txt
```

On Windows:

```
rspec spec\lab_quest_11_13_spec.rb -o lab_quest_11_13_output.txt
```

1. Using the starting word "**amore**", what song title do you get?
2. Using the starting word "**love**", what song title do you get?
3. Using the starting word "**little**", what song title do you get?
4. Explain why so many of the titles devolve into repeating patterns.
5. Try several words. Find a song title that terminates in less than 20 words. Could you find one?
If so, which song title did you find? If not, why not?

Last Step

Implement a "fix" for the problematic phenomenon you observed in Question 6. If you have successfully solved these problems, you can remove the restriction of 20 words maximum in the song title. (*Hint: If it goes boom, then you have not solved the problem*)

Lab Questions

Answer the following questions Questions 16 through 20 in the questions.txt file.:

1. Describe in one or two paragraphs your extension and how it fixed the repeating phrase/word problem.
2. Using the starting word "**montana**", what song title do you get?
3. Using the starting word "**bob**", what song title do you get?
4. Using the starting word "**bob**" again, do you get the same title? If no, what do you get? Try it a third time. Explain why the title might differ each time.
5. Share your favorite song title that you have found.

Troubleshooting

This lab requires an independent study of the Ruby language. You are encouraged to use any web tutorials and resources to learn this language. Given the size of the class, I will not be able to debug your code for you. **Please do not send panicked emails requesting I fix your bug for you. Allow yourself plenty of time, and use patience, perseverance, and the internet to debug your code.**

Lab Questions

The following questions are for feedback and evaluation purposes. Points are awarded for any sincere answer.

These questions should be answered in the questions.txt file.

1. Name something you like about Ruby. Explain.
2. Name something you dislike about Ruby. Explain.
3. Did you enjoy this lab? Which aspects did you like and/or dislike?
4. Approximately how many hours did you spend on this lab?
5. Do you think you would use Ruby again? For which type(s) of project(s)?

Grading

With the exception of questions 10, 14, 15, 16, 19, and 20 questions 1-20 are automatically graded by passing the associated tests. This is all or nothing, either you passed the tests or not. The remaining questions will be graded based on your answers and may receive partial credit. Questions 21 - 25 will be awarded full credit as long as there is a sincere and appropriate answer.

- Questions 1-20 -> 2 points each

- Questions 21-25 -> 1 points each
- Comments and Code Style -> 5 points

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 any aspect of Ruby you need to complete the assignment, but not to answer the questions asked in this lab.

Comment your program heavily. Intelligent comments and a clean, readable formatting of your code accounts for 20% of your grade.

Save the final version of your program and construct a zip file containing **ONLY** the following items:

- ruby_lab.rb
- lab_quest_1_5_output.txt
- lab_quest_6_9_output.txt
- lab_quest_11_13_output.txt
- questions.txt

Note, I will only accept plain text files for your answers. The submission of PDF, Word, Images, or anything other format except plain text will receive zero credit.