"The Most Probable Song Title"

Language Choice Programming Lab

Concepts of Programming Languages
CSCI 305, Spring 2018

Due: 3/25/2015 @ 12:30 PM

Choose A Language

For this lab you have a choice of one of the following languages:

- Ruby 2.3
- Go 1.9
- Python 3.6

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.

You will need to download the following file:

http://labrosa.ee.columbia.edu/millionsong/sites/default/files/AdditionalFiles/unique_tracks.txt

This file contains one million lines and weighs in at 82 MB. You should probably avoid viewing it in a web browser.

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.

http://nisl.cs.montana.edu/~pdonnelly/CSCI305/data/a_tracks.txt

File Templates

First, rename this file to `[LastName].[FirstName].open_lab.rb` for ruby, `[LastName].[FirstName].open_lab.go` for go, and `[LastName].[FirstName].open_lab.py` for python, where `[LastName]` and `[FirstName]` are your last and first names, respectively. Do not include the brackets `[ ]` in your file name. Secondly, edit the header comments int he file to reflect your name.

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

Ruby: `$ ruby Griffith.Isaac.open_lab.rb unique_tracks.txt`

Go: `$`

Python: `$ python36 Griffith.Isaac.open_lab.py 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 To Me`

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: http://regexpal.com/. It allows you to test your regular expressions on a block of text that your 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 no want to omit the apostrophe as it allows contractions.

Many of these characters have special meanings in the choosen languages. 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, paerntheses 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 vas majority of irregularities in the song titles.

Step 3: Eliminate punctuation

Next, find and delte the following typical punctuation marks:

`?  ¿  !  ¡  .  ;  &  @  %  #  |`

Unlike before, delte only the synmbol 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 metacharacter 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 teh 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.

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:

- 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 (in ruby), 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.

Question 1: Which word most often follows the word **"happy"**?

Question 2: Which word most often follows the word **"sad"**?

Question 3: How many different (unique) words follow the word **"computer"**?

Question 4: Which word most often follows the word **"computer"**?

Question 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:

- 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 the`
- 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`

  becasue no song titles in `a_tracks` contain the word **"computer"**

Lab Questions

Question 6: Using the starting word **"happy"**, what song title do you get?

Question 7: Using the starting word **"sad"**, what song title do you get?

Question 8: Using the starting word **"hey"**, what song title do you get?

Question 9: Using the starting word **"little"**, what song title do you get?

Question 10: 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

Question 11: Using the starting word **"amore"**, what song title do you get?

Question 12: Using the starting word **"love"**, what song title do you get?

Question 13: Using the starting word **"little"**, what song title do you get?

Question 14: Explain why so many of the titles devolve into repeating patterns.

Question 15: 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

Question 16: Describe in one or two paragraphs your extension and how it fixed the repeating phrase/word problem.

Question 17: Using the starting word **"montana"**, what song title do you get?

Question 18: Using the starting word **"bob"**, what song title do you get?

Question 19: 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.

Question 20: Share your favorite song title that you have found.

Troubleshooting

This lab requires an independent study of one of the three languages specified: Ruby, Go, or Python. 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

Question 21: Which language did you select and why did you select it?

Question 22: Name something you like about the language you selected. Explain.

Question 23: Name something you dislike about the language you selected. Explain.

Question 24: Did you enjoy this lab? Which aspects dis you like and/or dislike?

Question 25: Approximately how many hours did you spend on this lab?

Question 26: Do you think you would use the language you selected again? For whicy type(s) of project(s)?

Submission

Each student will complete and submit this assignment individually or in a two-person team. Do not consult with others. However, you are encouraged to use the internet to learn any aspect of Prolog 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 zip the source code into a file named `[lastname]_[firstname].prolog2.zip`. Type your lab questions in plain text as `[lastname]_[firstname].prolog2.txt`. Include your name in the text file.

We must be able to run your program from the command line with no arguments.

Submit your files to the Java Program 2 dropbox folder on BrightSpace. Submit your files before the due date as late submissions will not be accepted.