“The Most Probable Song Title”

Programming Lab 1

Concepts of Programming Languages
CSCI305, Spring 2014

Released: January 22, 2014
Due: February 10, 2014 at 11:59 pm

Perl

For this lab you will use Perl. Perl is installed on all machines in the lab. Perl should already be installed on OSX and on most Linux systems. To check your version, type `perl -v` at the terminal.

Window users will need to install Perl and have their choice between Active Perl or Strawberry Perl.

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 millions 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 Template

You should download the lab 1 perl file template from `http://nisl.cs.montana.edu/~pdonnelly/CSCI305/data/template.lab1.pl`.

First, rename this file to `[LastName].[FirstName].lab1.pl` 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 in the file to reflect your name. Also edit the variable declaration `my $name` so the program will print your name whenever you execute it. Add your team member's name if appropriate.

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

```
>perl Donnelly.Patrick.lab1.pl unique_tracks.txt
```

The 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 can use Ctrl (or Cmd) +C to cancel 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. 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.

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 preform 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 follow 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 near the enter key). This is an important distinction. We do not want to omit the apostrophe as it allows contractions. Many of these characters have special meanings in perl. Make sure you properly escape symbols when necessary (see: `http://www.dispersiondesign.com/articles/perl/perl_escape_characters`). The last one is the abbreviation for `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 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 [UPDATED 01.24]**

Next, find and delete the following typical punctuation marks:

`?  ¿  !  ¡  .  ;  &  $  @  %  #  |`

Unlike before, delete only the symbol itself and leave the text the 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.

Step 4: **Filter out non-English characters [UPDATED 01.24]**

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 Perl’s word metacharacter definition (hint: lookup `\w` and `\s` in Perl) 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. Perl has a function to do this for you.

Self-Check [UPDATED 01.24]

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, that is sufficient. If you are way off, 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 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. You can compare your choice to mine (here).

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**” is 33 times.

Building a Song Title

Now you are going to built 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, use the Perl function `rand()` to break it (i.e., a coin toss). For example, the line `print 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 the following 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
```

  because 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 common 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 "**little**" again, what song title do you get now?

Question 12: Did this solve the problem from question 10? Explain.

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

Question 14: Did this solve the problem from question 10? If yes, give (here) your updated answer to Question 6. If no, propose a way to fix this problem and explain in detail.

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?

Troubleshooting

This lab requires an independent study of the Perl language. You are encouraged to use any web tutorials and resources to learn Perl. 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 16: Name something you like about Perl. Explain.

Question 17: Name something you dislike about Perl. Explain.

Question 18: Did you enjoy this lab? Which aspects did you like and/or dislike?

Question 19: Approximately how many hours did you spend on this lab?

Question 20: Do you think you will use Perl again? For which type(s) of project(s)?

Team Extension

This question is required by students working in teams. Individuals may complete this exercise, but it is not required.

Implement the “fix” you describe in Question 14. This should fix 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. If it goes 💣, you have not solved the problem.

Question 21: Using the starting word “**love**”, what song title do you get? Describe in one to two paragraphs your extension and how it fixed the problem.

Question 22: Share your favorite song title you have found.

Partner Evaluation

This response is required by students working in teams. This is your opportunity to assign your partner up to 10 points based on his or her contribution to the project. *This response will remain confidential.*

Question 23: In one paragraph, describe your working relationship with your partner and your division of work. Did you divide and conquer? Did you work together on all parts? Was this a productive collaboration? Did each team member pull his or her own weight?

Please evaluate your partner’s contribution on the lab with a score from 0 (no work) to 10 (equal partner).

Submission

Each student will complete and submit this assignment individually or in a two-person team. Do not consult with other teams. However, you are encouraged to use the internet to learn Perl.

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

Save the final version of your program as `[lastname]_[firstname].lab1.pl`. If you completed questions 21 and 22, submit the final version of your program. You may comment out and leave any code segments you used to answer the lab questions.

Type your lab questions in plain text as `[lastname]_[firstname].lab1.txt`. Include your name in the text file.

Your program must operate on the original dataset. Do not submit the song file dataset. I will use the file `unique_tracks.txt` from the source given above to evaluate your program.

Team members will submit identical code, but each student must submit individually. You will email these files to `msucsci305@gmail.com`. Use the subject line `[lastname]_[firstname] Lab1 Submission`. Do not archive your files but instead use two attachments. Email your two files before 11:59pm on the due date. Late submissions will not be accepted.