CSCI 305 Concepts of Programming Languages

Programming Lab 1

Perl:

For this lab you will need to learn and use Perl. (You have seen a few examples in class, but you do need to learn more on Perl to finish this lab.) Perl is installed on `esus`, or most Linux systems. Type `perl -v` to check your version.

Windows users need to install Perl, for instance with Active Perl or Strawberry Perl, etc.

Dataset:

This lab will process a dataset containing 1 million song titles. The dataset was used at Columbia University by the Laboratory for the Recognition and Organization of Speech and Audio. For your convenience, you can download it directly from our course webpage:

```
http://www.cs.montana.edu/bhz/classes/spring-2016/csci305/LAB1/unique_tracks.txt
```

The file is big (82 MB), so don't try to keep multiple copies on your computer. For debugging and testing purpose, you can also use the dataset containing only song titles that begin with the letter 'A'. This file, which is much smaller, can be downloaded at:

```
http://www.cs.montana.edu/bhz/classes/spring-2016/csci305/LAB1/a_tracks.txt
```

File Template:

You should download the lab 1 perl program template at:

```
http://www.cs.montana.edu/bhz/classes/spring-2016/csci305/LAB1/template.lab1.pl
```

You should 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 $name` so the program will print your name whenever you execute it. Add your team member's name as variable `$partner`, if appropriate. (**You can finish this lab by yourself, or you can form a 2-person team.**)

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

```
esus.cs.montana.edu>perl zhu.binhai.lab1.pl unique_tracks.txt
```

The current template simply loops through each line of the file and prints the line. You need to do much more than that. Remember that you can use Ctrl-C to terminate the execution of the program.

Preprocessing:

Step 1: Extract song title

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

```
   TRMMPZM128F14A2184<SEP>SOVUYNF12AF72AAAD3<SEP>Blind Melon<SEP>All That I
Need
```

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 then discard all other information.

Step 2: Eliminate superfluous text

The song title, however, could be very noisy. It could contain additional information beyond the song title. Look at the following example:

```
   Adagio Cantabile [Solo Sonata for Bass-Vibes PWV 21] (Werner Pirchner)
```

You need to do some preprocessing to clean up the song titles. You need to write a series of regular expressions that match a block of text and replace it with nothing.

You could begin by writing a regular expression that matches a left bracket and any text that follows it. Replace the bracket and all the text that follow it with nothing. In this example, the modified title becomes `Adagio Cantabile`.

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

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

In the above list, left quote (on the tilde key above tab) is listed, but 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. The last one listed is `feat.` — short for featuring and followed by artist information you do not want to retain. For example, `Sunbeam feat. Vishal Vaid` should become `Sunbeam`.

Many of these characters have special meanings in perl. Make sure you properly escape symbols when necessary. Failing to escape characters properly will be the most common mistake in this lab (see:`http://www.dispersiondesign.com/articles/perl/perl_escape_characters`).

Step 3: Eliminate punctuation

Next, find and delete the following typical punctuation marks:

```
? ¿ ! ¡ . ; & $ @ % # |
```

Unlike Step 2, only delete the symbol itself and leave all of the text that follows. 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. (Refer to the list of escape characters.)

Step 4: Filter out non-English characters

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.) Here, 'English characters' 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: Convert to lowercase

Convert all words in the sentence to lower case. Perl has a function to do that.

Self-Check: in the `a_tracks` dataset, after all filtering, roughly 52,760 valid song titles can be found. If your number is way off (e.g., 100+/-), then 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 texts 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, split the title string into individual words. Next, use one or more data structures to keep track of these words pair counts. That is, for every word, you must keep track of the count for each word that follows it. Try to design your data structure for fast retrieval. (You can ask for a hint if necessary).

Self-Check: After you implement your data structure, you can check with 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**".

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, use the Perl function `rand()` to break it. For example, the line `print mcw("computer");` should give you your answer to Question 4.

Then 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 data set. 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 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. This is started 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 query word "**happy**", you might get the title:
  `happy now the world of the world of the world the world of the world of the world of the`
- Using the query word "**sad**", you might get the title:
  `sad love song for you ready for you ready for you ready for you ready for you ready for you`
- Using the query word "**computer**", you should get the title:
  `computer`
  because no song title in `a_tracks` contain the word "**computer**".

Step 6: Filter out stop words

Try to fix the problem you observed when trying different words, like `a, an, the, to`, etc. In natural language processing, '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 stop words from all the song titles:

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

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: Using the starting word "**amore**", what song title do you get?
Question 11: Using the starting word "**love**", what song title do you get?
Question 12: Even at this point, you might still see repeating phrase/word. Describe a way to (at least partially) solve this problem. Implement it.
Question 13: After Q-12, using the starting word "**montana**", what song title do you get?
Question 14: After Q-12, using the starting word "**bob**", what song title do you get?
Question 15: After Q-12, using the starting word "**moon**", what song title do you get?
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? What aspects did you like/dislike?
Question 19: Approximately how many hours did you spend on this lab (including learning Perl)?
Question 20: Do you think you will use Perl again? For what kind of projects?

Troubleshooting

This lab requires an independent study of the Perl language. Use any web tutorials and resources to learn Perl. Given the size of the class, we will not be able to debug your code for you. Please do not send panicked emails requesting us fix your bug for you — same as when you work for a company. Do 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 or in a 2-person team. Comment your program properly, intelligent comments and a clean, readable formatting of your code account for 20% of your grade for this lab.

Save your final version of your program as `lastname_firstname.lab1.pl`. 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. Do not submit the song file dataset. We will use the file `unique_tracks.txt` to evaluate your program.

Team members will submit identical code, but each student must submit individually. Submit your file to the Lab1 dropbox folder on D2L. Do not archive/compress your files but instead use two attachments. Note that late submissions will not be accepted.

DEADLINE: Feb, 16, 2016; 11:30pm.