"The Most Probable Song Title"

Programming Lab 2

Programming Language Fundamentals (CS 381)

Perl

For this lab you will use Perl. 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 the following file (it is in the Lab1 starter pack):

`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, the starter pack contains a subset of this dataset containing only song titles that begin with the letter "A" and another set for the letter "B". You will use these smaller datasets for debugging and testing purposes.

Getting Started

Lab2 builds on Lab1. Finish Lab1 and pass all tests before proceeding to Lab2. Make a copy of your `lab1.pl` script in a new directory `lab2`. You will expand the functions marked for Lab2.

```
$ cp lab1/lab1.pl lab2/lab2.pl
```

Bi-gram Counts

A 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 (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 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.

PROTIP: Double check that both words w_{i-1} and w_i are not empty before updating the count in your hashmap.

Complete the function `build_bigrams`. Have some pseudocode.

```
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
```

PROTIP: You can use the command `preprocess` which runs all the individual `filter` tasks (from Lab1) and then calls `build` to build the bi-gram model.

Finding the 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 will take 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. That way you will get the same results as the test cases.

Protip: When comparing counts and you find a tie, stick with the first word found. This will ensure you will pass the tests.

You can now use the command `mcw WORD`, where `WORD` is the argument passed to the function. You can now explore Tests `t30` – `t39`.

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. You can now explore Tests `t40` – `t59`. However, you will not pass these tests until you complete the next section.

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.

Hint: It is because these words occurred very often in either order in the corpus.

We can fix this by allowing each word to appear in the title only once. Add a data structure to track the word history. Edit `sequence` so that every time a word is used in your title it is added it to word history.

Now go back and edit `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. In other words, if the most common word has already been used, pick the next most common word, and so one.

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

Testing

The template and the test files are located on Canvas. Keep the test files in a subdirectory `./tests/` in order to use the run all test instructions below.

PROTIP: You do not need to run every test every time. Make a copy of all the tests. Keep only the tests relevant to your current task in the `./tests/` directory and copy tests in or out of the directory as needed.

Run Individual Tests

You can run any single test as follows:[1]

```
$ perl lab2.pl < ./tests/t30.in > t30.myout
diff ./tests/t30.out t30.myout
```

Run all Tests

You can also run all the tests at once. For this there is a Python3 script named `lab2-tester.py`. This is a Python script that executes `perl lab2.pl` for each test and performs a `diff` between the expected output and your output.

```
$ python3 lab2-tester.py
```

This script assumes:
- your perl file is named `lab2.pl` and in the same directory
- the subdirectory `./tests` is in the same directory
- it has permissions to make a subdirectory `./myouts`
- that your OS has the `diff` tool[2]

PROTIP: It may take about 10 minutes to run the full tests suite (n=33 tests).

Submission

Submit your program `lab2.pl` on Canvas. This project is worth 100 points. Grading will correspond to the number of tests passed, which roughly track your progress through the above tasks. You should comment your code (especially in `mcw` and `sequence`), but for the most part I have left ample instructions which suffice as the comments.

[1] If you know `vim`, consider using `vimdiff` instead.
[2] Windows users: http://gnuwin32.sourceforge.net/packages/diffutils.htm

Protip: Submit only your perl `.pl` script. Do not submit your executable program. Do not archive (e.g., `zip`) your file.

Each student will complete and submit this assignment individually. Submit your work before the deadline as late work is not accepted (except using your late days).

Protip: If you do not finish in time, turn in whatever work you have. If you turn nothing, you get a zero. If you turn in something, you receive partial credit.

command	argument	description
`build`		build the bi-gram 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 character in @tracks
`filter`	title	extract title from noisy input original
	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
	INT	prints only `INT` tracks
`random`		print sequence based on random word
`stopwords`	on	filter stopwords on
	off	filter stopwords off
`sequence`	`WORD`	find sequence to follow `WORD`