"Regular Expressions"

Programming Lab 1

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 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.

Protip: You will build on your code for Lab2.

Getting Started

Unpack the starter pack archive available on Canvas. First make a copy of the lab template (e.g., `cp lab1_template.pl lab1.pl`). You will modify `lab1.pl` for this lab.

Open the file in your favorite text editor. At the top, add your name and email as instructed by the comments. Set the variable `$name` to your name.

Begin by running the program.

```
$ perl lab1.pl
```

The template provides you with a working user menu (although not all features are enabled until you write them). It takes commands followed by an argument (optional). The command list is listed at the end of this document. Try it out and type commands, one per line. Afterwards, type the end-of-input character CTRL+D.

```
load tracks.txt
print
^D
```

This will read in the file `tracks.txt` and print out each line to standard out. You will want to kill the program or you will be waiting a long time.

> PROTIP: Remember you can use Ctrl+C to cancel execution of the program.

Next, run the program again and count the number of lines in the input file.

```
load tracks.txt
count tracks
^D
```

You can verify that there are indeed one million song titles. Now run your first test.

```
$ perl lab1.pl < ./tests/t01.in
```

```
54586
253658
4638630
```

Test `t01` loaded a smaller dataset `a_tracks` which contains the 54,586 tracks that begin with the letter 'A'. It counted the number of tracks, words, and characters in the file.

```
load a_tracks.txt
count tracks
count words
count characters
```

> PROTIP: You can save standard out into an output file, e.g.,
>
> `$ perl lab1.pl < ./tests/t01.in > t01.myout`

Pre-processing

As you can see, these songs titles are incredibly messy. We need to clean them up. Regular expressions are the perfect tool for that task. And Perl is the perfect language for regular expression wizardry.

Task 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 discards all other information.

PROTIP: You may find this site useful in debugging your regular expression. It allows you to test your regular expressions on a block of text that you provide.

http://regexpal.com/

In the code, find the function `extract_title`. In the `if` statement, edit the regular expression to capture only the song title. The `print` command is useful to print out the tracks as you develop your regular expressions.

PROTIP: The print command takes an optional integer argument which restricts printing to only the first n tracks. For example, `print 10`, prints only the first ten tracks.

Now you can run Test `t02`, which runs the `filter title` command.

```
$ perl lab1.pl < ./tests/t02.in
```

If you pass Test `t02`, try Test `t12` and Test `t22` which are the same but on a different set of tracks. There pattern is true for the other tests (e.g., Tests `t03`, `t13`, `t23`).

Task 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 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 Perl. Make sure you properly escape symbols when necessary.

> PROTIP: Failing to escape characters properly is a common mistake made in this lab (see: http://www.dispersiondesign.com/articles/perl/perl_escape_characters).

The last one listed above is an abbreviation `feat.` – short for featuring and followed by artist information you need not retain. Don't forget to match a literal period `.` after the `t`. 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.

In the code, find the incomplete function `comments`. Add a series of regex substitutions to remove the superfluous information. You can now use the command `filter comments` and try Test `t03`.

Task 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 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. Again, refer to list of escape characters.

For the symbols ¿ and ¡, use the unicode representation, which are `\x{00A1}` and `\x{00BF}`. In the code, find the incomplete function `punctuation`. Add a series of regex substitutions to remove the punctuation. You can now use the command `filter punctuation` and try Tests `t04` and `t05`. Note that colon is intentionally repeated from Task 2 (needed for `t05`).

***Step 4:* Trim whitespace**

All our replacements have left some blank titles or mangled titles.
First, identify any titles that begin or end with an apostrophe. Replace those apostrophes with nothing. Be careful not to disturb the apostrophes in the interior of the sentence. Next, trim leading and trailing whitespace from each title. Be careful not to replace all whitespace − we need whitespace between word boundaries. To match the tests, do not combine the above two operations into one, but handle apostrophes first and then handle whitespace as a second line of code.

In the code, find the incomplete function `clean` and complete the instructions to achieve the above goal.

***Step 5:* Filter out non-English characters**

We want to ignore all song titles that contain a non-English character (e.g., á, ì, ö, etc). Although this may eliminate some titles in languages like Spanish or Danish, we need to drop out the unicode garbage characters found in many of the titles.

If the titles contain only valid English alphanumeric characters ('a' to 'z' and 0 to 9) or the apostrophe, we keep it. Otherwise, we discard the song title entirely. Edit the function `clean` to achieve the above goals.
In the code, continuing editing the function `clean` to achieve the above goal.

***Step 6:* Skip blank titles**

After all the replacements, some titles are left with nothing. Continue to edit the function `clean` to throw out any empty titles. Use a regular expression to check if a title is empty (or contains only whitespace). If empty, do not retain it in the `@tracks` array. Likewise, if the title contains only an apostrophe, discard it. You may want to use the `count` command to check how many lines are left after you skip the blank ones. To match the tests, do not combine the above two operations into one, but handle apostrophes first and then handle whitespace as a second line of code.

In the code, continuing editing the function `clean` to achieve the above goal.

***Step 7:* Set to lowercase**

Convert all words in the sentence to lowercase. Perl has a function to do this for you. Go back and edit the `clean` function one more time to convert all titles to lower case.

In the code, continuing editing the function `clean`. Following tasks 4-7, try Test `t06`.

***Step 8:* Filtering out common words**

In the domain of Natural Language Processing (NLP), 'stop words' are common words that are often filtered out in preprocessing.

Go edit the function `stopwords` to 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`

Be careful to only replace entire words, not just any occurrence of these letters. Otherwise you will turn words like `outstanding` to `stg`.

PROTIP: The regex control character `\b` will be quite useful here.

We may not want to always filter these words, but we want to have the option. You can now use the command `filter stopwords` and try Test `t07`. You can toggle this mode using commands `stopwords on` and `stopwords off`.

Putting it all together

You can now use the command `preprocess` which runs all the individual `filter` tasks . Try Tests `t08`, `t09`, `t10` to test your system so far on `a_tracks` dataset.

PROTIP: Once you have debugged all the steps of `preprocess`, test on Tests `t11` − `t19` which tests on the `b_tracks` dataset. Tests `t21` − `t29` are similar but test on the full `tracks` dataset of 1M songs.

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.

Run Individual Tests

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

```
$ perl lab1.pl < ./tests/t01.in > t01.myout
diff ./tests/t01.out t01.myout
```

Run all Tests

You can also run all the tests at once. For this there is a Python3 script named `lab1-tester.py`. This is a Python script that executes `perl lab1.pl` for each test and performs a `diff` between the expected output and your output.

```
$ python3 lab1-tester.py
```

This script assumes:

- your perl file is named `lab1.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]

Submission

Submit your program `lab1.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.

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.

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

Name	Symbol	Words	Characters
none	none	3 369 967	18 511 941
left paren	(	2 905 463	15 440 220
left bracket	[	2 881 949	15 287 812
left brace	{	2 881 857	15 287 292
backslash	\	2 881 830	15 287 118
forwardslah	/	2 855 066	15 121 816
underscore	_	2 793 746	14 765 552
hyphen	-	2 751 305	14 49 6381
colon	:	2 728 469	14 359 868
double quote	”	2 725 946	14 344 805
left quote	`	2 725 946	14 344 805
plus	+	2 724 710	143 38 523
equals	=	2 724 548	14 337 640
featuring	feat.	2 720 691	14 315 552
Test t23	`t23.out`	2 720 691	14 315 552

Table 1: Counts for Task 2

Name	Symbol	Words	Characters
Test t23	none	2 72 0691	14 315 552
question	?	2 720 324	14 308 679
inverted question	¿	2 720 324	14 308 475
exclamation	!	2 720 100	14 303 582
inverted exclamation	¡	2 720 100	14 300 884
period	.	2 719 850	14 273 763
semi-colon	;	2 719 847	14 273 544
ampersand	&	2 714 526	14 267 735
dollar sign	$	2 714 500	14 267 572
asterisk	*	2 714 450	14 266 069
at sign	@	2 714 395	14 265 913
percent	%	2 714 390	14 265 760
pound sign	#	2 714 292	14 264 470
pipe	|	2 714 242	14 264 393
Test t24	all above	2 714 242	14 264 393

Table 2: Counts for Task 3

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`

Table 3: Menu system for Labs 1 and 2