Project 2A Specification

Project 2A Specification

This project is new, so please bear with me with any technical issues or data issues.

Remember, the goal of this project is to learn something timely and also to have some fun.

This specification is written by R^3, so, despite his reputation of writing on and on and on and on, the specification will likely not be as lengthy as those for Project 1.

There may be parts of the project that are vague. This usually means I am not particularly worried about certain nuances, or want you to struggle through it a bit, **but do feel free to ask me on Piazza and I will clarify if I do have expectations.**

Text Parsing

Most software engineers and data scientists focus a lot on working with numbers, structured data, and working with rather regular data (this is an understatement). Text mining (as well as other fields such as image processing, computer vision etc.) is becoming much more important as computer systems evolve and are able to process more and more data in more sophisticated ways. Knowing how to work with **unstructured** data is very important as a data scientist, or as a software engineer, particularly if you are interested in working in any kind of "data" space (i.e. data science, machine learning, whatever).

In Project 2A, you will write a function, in Python, that parses Reddit comments and takes them from raw messy text into a cleaner effluent that we can then use for analysis or training sentiment analysis model on.

This may sound daunting, but it isn't.

The Great Beyond

I might regret saying this, but if you are adventurous, you may want to use other computing resources, such as a personal server, some UCLA server, that provides more resources than a VM, but the VM should work. We cannot provide technical support, but there are some great tutorials on setting up Spark on Ubuntu. Spark works well in local mode, but the real power comes on a powerful server, or a cluster..

I will set up a Raspberry Pi in my office with a static IP so you can download the full data if you are interested, because my personal website is bandwidth limited, and the UCLA Google Drive is very slow.

The Data

On your VM, there is a `data` directory. The data comes from Reddit `/r/politics` and covers submissions and comments from November 2016 (right before the election) to the latest data dump covering to February 2018. It also includes all comments and submissions made to other subreddits made by users that post to /r/politics. This data can be used to detect highly partisan Reditors, bots (or possibly Russian interference as Reddit was used as a propaganda vehicle in addition to Facebook). I do not want to take over your entire hard drive, so I further limited the data to only comments between 50 and 100 words in length.

- `comments-minimal.json.bz2` contains comments in JSON format. These fields are not listed in any particular order:

Field Name	Data Type	Description
id	String	Unique identifier for this comment.
author	String	The username of the author of the comment. To see info about a particular user, use the URL http://www.reddit.com/user/USERNAME
subreddit_id	String	Remember youtube_video id? subreddit_id is an alphanumeric code that uniquely identifies a subreddit. It is **not** the subreddit's name.
subreddit	String	The case-sensitive name of the subreddit. It looks like /r/politics
stickied	Boolean	Is the comment stickied to the top of all comments? This is usually initiated by a moderator to keep discussions on track.
score	Integer	Represents some computation of upvotes and downvotes. This is the number that is seen as points on each comment.
retrieved_on	Timestamp	Ignore. The date/time that Michael scraped this comment.
permalink	URL	Useful for debugging. Copypasta this link to see the original comment in all its glory.
parent_id	String	Alphanumeric code denoting the parent comment, if this is a reply. **This key may be missing.**

is_submitter	Boolean	Is this comment posted by the OP (original poster)?
gilded	Boolean	This comment was so good, or so useful, that another user "gilded" them with Reddit Gold. Reddit Gold is the ad-free version of Reddit. Gilding only lasts a certain period of time.
edited	Boolean	Whether or not this comment was edited after being posted.
distinguished	Boolean	**I don't know what this is.**
created_utc	Timestamp	The date/time the comment was posted by the user.
controversiality	Integer: 0/1	Whether or not the comment is controversial based on number of upvotes and downvotes. This is some ratio that is Reddit proprietary.
can_gild	Boolean	Whether or not the user that made the comment has the power to gild other Redditors.
body	String	**The actual text.**
link_id	String	~~An ID that identifies the first URL included in a comment, **I think.**~~ **Links a comment back to the submission it appears on.**
author_flair_text	String	**This may be useful.** When a user writes a comment, or posts a submission, they have the option of adding a "flair" next to their username. Flair just represents something interesting about the user. In `/r/politics`, it can be political affiliation, which candidate they support(ed). **More importantly, many users add their STATE (i.e. California) as their flair.** In `/r/ucla`, it is typically the student's major and graduation year. For alumni, it is typically the degrees earned (year optional). For example: `"Mathematics B.S., Computer Science M.S., Statistics, Ph.D."` Or in /r/MTB it might the bike someone rides: `i.e. "Specialized Stumpjumper FSR Comp 6Fattie"`
author_flair_css_class	String	**Probably not useful.** A name of the CSS class used to display their flair. In /r/politics, it might be red or blue to denote party affiliation, but I have not seen it often.
author_cakeday	Boolean?	I believe this represents whether or not the post date was the user's "Reddit Birthday" sometimes called a "cake day" where a little icon of a cake may be displayed.

For advanced students, you will note that this data, if put into a table, is *not* normalized. It has redundant data, particularly user data.

- `submissions.json.bz2` contains submissions in JSON format. It has the following keys. Most of them are irrelevant and are related to, you guessed it, ad tracking. This list is not comprehensive since you will use submissions less than comments.

Field	Datatype	Description
id	String	Unique identifier for a submission.
title	String	The title submitted by the submitter.
thumbnail	URL	Raw URL pointing to the submission's thumbnail (if webpage or news article).
spoiler	Boolean	Used mainly for TV shows, but can also be used for election results.
subreddit_id	String	Same as with comments
subreddit	String	Same as with comments.
stickied	Boolean	The submission is stuck to the top of the subreddit page.
selftext	String	The text of a post, if it is not a link or article. For example, /r/ucla is mostly selftext... people asking questions or jokes about Gene Block.
score	Integer	Same as with comments, but displayed next to the story.
pinned	Boolean	Similar to stickied.
permalink	URL	A permalink to the submission.
over18	Boolean	If this post is NSFW (not safe for work) and is for 18+ audience.
author	String	Same as with comments.
locked		**Very common in /r/politics.** Once a thread becomes toxic, the entire discussion is usually locked so nobody can participate and it becomes read only.
num_comments	Integer	The number of comments.
user info	A bunch.	The same data that was provided for comments.

Investigating the Data

Do not decompress the `bz2` file in your VM. If you wish, copy the files to your shared directory (still `www`) and uncompress them there.

You can then use your favorite Eggert UNIX tools to investigate the data... but wait... there's more!

You can install a tool called **jq** that is basically grep for JSON files!!!

The Actual Project 2A

Your first goal is to write a **Python** function called `sanitize` in a file called `cleantext.py` that **takes a string as an argument,** parses and tokenizes messy text into something standardized **and returns a list containing four strings. These four strings are described later. You may not use any libraries that are not part of the standard libraries: `re` is a standard library, but `nltk` is not because it must be installed.**

Please use this template for your coding.

I wrote a function to do this at Facebook, and I now use it for everything including my dissertation. Your job is to reproduce this similar function.

Your function must do the following:

1. Replace new lines and tab characters with a single space.
2. Remove URLs. Replace them with the empty string ''. URLs typically look like `[some text](http://www.ucla.edu)` in the JSON.
3. Split text on a single space. If there are multiple contiguous spaces, you will need to remove empty tokens after doing the split.
4. Separate all *external* punctuation such as periods, commas, etc. into their own tokens (a token is a single piece of text with no spaces), but maintain punctuation *within* words (otherwise `he'll` gets parsed to `hell` and `thirty-two` gets parsed to `thirtytwo`). The phrase "`The lazy fox, jumps over the lazy dog.`" should parse to "`the lazy fox , jumps over the lazy dog .`"
5. Remove all punctuation (including **special characters** that are not technically punctuation) *except* punctuation that ends a phrase or sentence and *except* embedded punctuation (so thirty-two remains intact). Common punctuation for ending sentences are the period (.), exclamation point (!), question mark (?). Common punctuation for ending phrases are the comma (,), semicolon (;), colon (:). *While quotation marks and parentheses also start and end phrases, we will ignore them as it can get complicated. We can also RRR's favorite em-dash (--) as it varies (two hyphens, one hyphen, one dash, two dashes or an em-dash).*
6. Convert all text to lowercase.
7. The order of these operations **matters**, but you are free to experiment and you *may* get the same results.
8. Your function should return one data structure containing four collections: a string containing the parsed text, a string of all unigrams (single tokens) in any order separated by a space, a string of all bigrams (a pair of tokens **in sequence** contained within each phrase/sentence without bounding punctuation) separated by spaces with an underscore between words in the bigram, and a string of all trigrams (a triple of tokens **in sequence** contained within each phrase/sentence without bounding punctuation) separated by spaces with an underscore between words in the trigram.
9. This is not as simple as it seems and you will see why.

Example Comment:

```
I'm afraid I can't explain myself, sir. Because I am not myself, you see?
```

Parsed Comment (Returned String 1):

```
i'm afraid i can't explain myself , sir . because i am not myself , you see ?
```

Unigrams (Returned String 2):

```
i'm afraid i can't explain myself sir because i am not myself you see
```

Bigrams (Returned String 3):

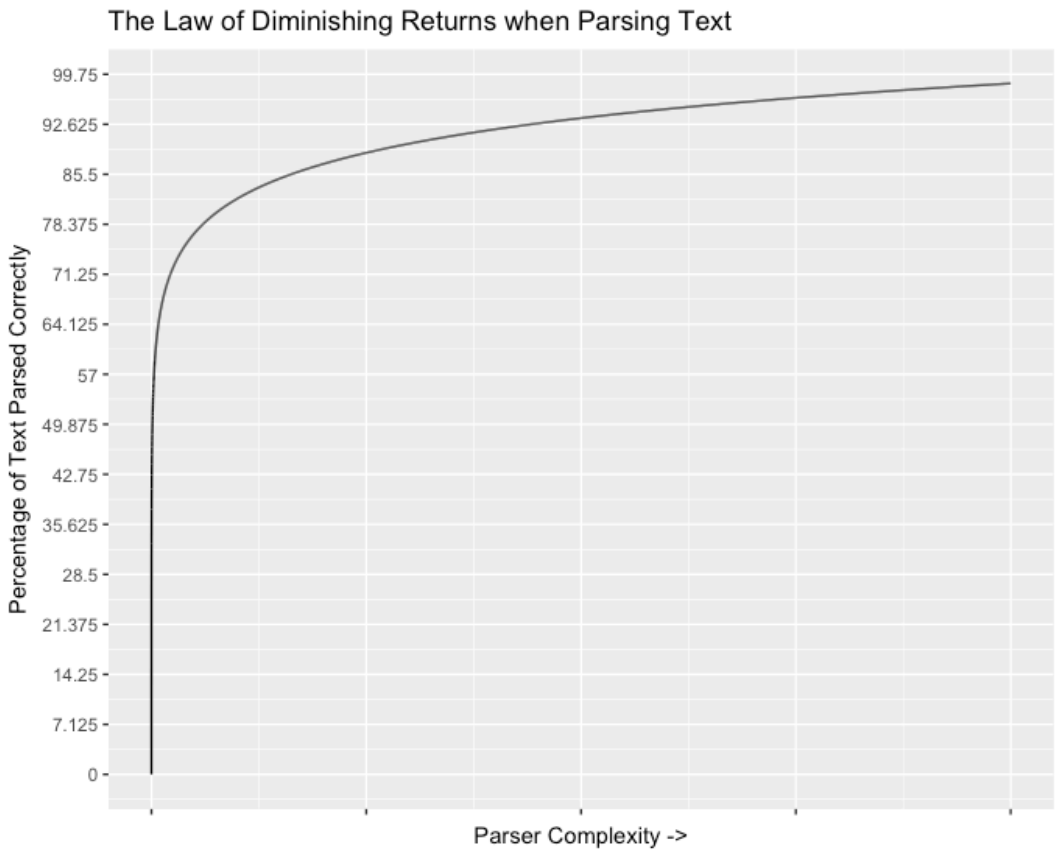
```
i'm_afraid afraid_i i_can't can't_explain explain_myself because_i i_am am_not not_myself you_see
```

Trigrams (Returned String 4):

```
i'm_afraid_i afraid_i_can't i_can't_explain can't_explain_myself because_i_am i_am_not am_not_myself
```

You will note that this function cannot be perfect. It takes a lot of finesse to write a good implementation. We are not expecting perfection, but we do expect code that meets the spec. Parsing (and optimization) is a never ending task, and learning *when to stop* is a very good skill for any software engineer to learn: the law of diminishing returns.

When I parse text, I do the main steps. Then I sample a couple hundred rows, identify unresolved problems, and determine how many records are affected by those problems. At a certain point, problems only affect less <1% of the data, and I stop (Project 2A will not be that thorough and that is OK).



You don't necessarily need to use Spark for part 2A, though it is likely helpful in case there is some weird translation issue moving from raw code to Spark-run code.

TODO: Testing Your Code

R^3 will create a Flask server somewhere, similar to project 1A, where you can copy/paste a comment from the file, press a button, and see what the text *should* parse to.

Submission Instruction

Preparing Your Submission

Please create **a folder named with your UID**, put all your files into the folder, then compress this folder into a single zip file called "P2A.zip". That is, the zip file should have the following structure.

```
P2A.zip
 |
 +- Folder named with Your UID, like "904200000" (without quotes)
     |
     +- readme.txt
     |
     +- team.txt
     |
     +- cleantext.py
```

Please note that the **file names are case sensitive**, so you should use the exact same cases for the file names. (For team work, only the submitter's UID is needed to name the folder.) Here is more detailed description of each file to be included in the zip file:

- `readme.txt`: Readme File
- **`team.txt`**: A plain-text file (no word or PDF, please) that contains the UID(s) of every member of your team. If you work alone, just write your UID (e.g. 904200000).
  If you work with a partner or a group, write all UIDs separated by a comma (e.g. 904200000, 904200001). **Do not include any other content in this file!**
- `cleantext.py`: The file containing your sanitize function.

Testing of Your Submission

Grading is a difficult and time-consuming process, and file naming and packaging convention is very important to test your submission without any error. In order to help you ensure the correct packaging of your submission, you can test your packaging by downloading this test script. In essence, this script unzips your submission to a temporary directory and tests whether or not you have included all your submission files. Once you download the test script, it can be executed like:

```
cs143@cs143:~$ ./p2a_test <Your UID>
```

(Put your P2A.zip file in the same directory with this test script, you may need to use "chmod +x p1b_test" if there is a permission error). You **MUST** test your submission using the script before your final submission to minimize the chance of an unexpected error during grading. Significant points may be deducted if the grader encounters an error during grading. When everything runs properly, you will see an output similar to the following from this script:

```
Check File Successfully. Please upload your P2A.zip file to CCLE.
```

Submitting Your Zip File

Visit the Project 2A submission page on CCLE to submit your zip file electronically by the deadline. **Submit only the "P2A.zip" file!** In order to accommodate the last minute snafu during submission, you will have 30-minute window after the deadline to finish your submission process. That is, as long as you start your submission before the deadline and complete within 30 minutes after the deadline, we won't deduct your grade period without any penalty.

Last modified: Thursday, 31 May 2018, 5:52 PM PDT