Megan O'Keefe
8 May 2016

Final Project Writeup:
"Bookbeats: Playlist-ify Your Text"

1. Description of the Motivating Problem

The original idea behind this project came from my love of mashing different media together, and also my love for books. I wanted to take the text of a book and figure out a way to reach into the vast Internet and apply NLP-related methodologies on that book. I also love to listen to music, so I thought- what if I could create a targeted playlist based on the text of a book? Going into this project, I wasn't sure how I might accomplish this, and I knew the process would require a lot of trial-and-error.

2. Citations, Acknowledgements

To my knowledge and extensive Google Searching, although there are lots of "playlist generators" out there (like Spotify's Playlist Miner), no one had developed a program to do the task of fetching songs based not on a search keyword or musical artist, but on input *data* or *corpora.* I set off without a clear idea of what I was doing, but I knew that the first step of the process would be figuring out what a text is "about." This is how I came across keyword extraction algorithms. After researching some existing implementations, I settled on RAKE, the well-researched Rapid Automatic Keyword Extraction algorithm. I was lucky to find released source code for a Python implementation of this algorithm. RAKE works by splitting the input text, removing stopwords, and computing word co-occurrence and frequency: keyphrase score = *deg(w) / freq(w).*

I also made use of Sravana's source code from early assignments, namely corpus.py and ngram.py. I also based my front-end Flask app on Sravana' demo. Thank you Sravana!

3. Explanation of My Approach

My algorithm went through several iterations, but the final version (as seen in fullSkeleton.py) works as follows:

1. Read in the input text. The Flask app allows for choices from my existing set of novels, or the user can paste in their own text.
2. Use RAKE to extract key phrases of at most three words from the text.
3. Extract the most common one- and two-word proper noun phrases from the text.
4. Combine these two sets of "keywords."
5. Iterate through each of these keywords, and get songs from the MusixMatch API
6. Given song metadata (title/artist), try to get Genius IDs for each of these candidate songs.
7. Use these song IDs to fetch song lyrics from Genius, using a web scraper I wrote.

8. Build a TFIDF matrix between the input text and all the candidate songs, and rank the songs based on cosine similarity with the novel (similarity.py).
9. Get the top ranked songs and use the Soundcloud API to embed the songs on the webpage.

Originally, I used cross-entropy rankings rather than cosine similarity rankings, but cross-entropy wasn't working very well in ranking the candidate songs, and after learning about vectors I felt like experimenting. I also added in the proper noun step at the end, with the hope of extracting place names (and thus getting songs that "take place" where the book takes place). I didn't quite reach that goal, but it was a step in the right direction.

4. Shortcomings of My Program and Ideas for Future Development

The most obvious shortcoming of my program is the "quality" of the final output songs. Do my output playlists "represent" the texts in question? No, not really— usually the case is that each song has some characteristic that clearly ties back to one of the extracted keyphrases. *Alice in Wonderland,* for instance, would often return songs by the rock band "White Rabbit," despite the lyrics not relating to the novel at all. Early iterations of my program returned songs that had very small sets of lyrics, assumedly because cross-entropy ranked them higher compared to other songs (*eg.* if a song only had a dozen lyrics and two of the phrases occurred heavily in the input novel). This is to say that as I tried to improve my algorithm to get better output songs, I realized the difficulty of my original problem— how, for instance, can we deduce what a text is "about?" How can we get a computer to see through the noise of a novel and extract its key themes? This is what I was trying to do with keyword extraction, and it gave me a lot of appreciation for researchers who work on these problems, because there is a lot of subjectivity at work.

Further, my program also does not discriminate based on genre, and the result is that the playlists are all over the place: rap, heavy metal, sometimes audiobooks that were somehow picked up in the MusixMatch query. It was never a goal to work with genre in this project (not enough time, and not necessarily related to NLP)— further, it would be a hard problem to figure out which song genres would best fit a text. I considered using book metadata (like publication date and book genre) to manually match a text to a set of genres, and this would be an idea for future development. Further, I could make it so that the user can choose a desired genre(s) from a drop-down menu in the Flask app.

Another shortcoming of my algorithm is how long it takes to run (at least 3-4 minutes for a full-length novel). But running it on a smaller text (like a news article) produces very mediocre song results— that is, the more data, the better. The majority of the running time is taken up by the API queries. I could make that step faster by figuring out a way to circumvent the three-step API fetch process, which admittedly was a band-aid sort of situation where directly querying Genius returned a ton of book excerpts from audiobooks (sometimes the text of the input novel itself!). So I had to go to MusixMatch which returned mostly just songs, then go back to Genius and scrape the lyrics because MusixMatch's free API would only give me the first few lines of the song lyrics. I'm sure that there is a better way to handle the Internet step of this project.

Hopefully, someday I can return to this project and make it better! I had a lot of fun trying to figure out how to solve this problem.

Link to original proposal: https://docs.google.com/document/d/1VvVzL0Maw9RTLnuq0u31vEvtl0JLowk6vhnuhUNbNEk/edit