HW 6 - Sam Farren

Text Mining on Lyrics of the Billboard Top 100 songs from 1964-2015

Overview:

This lab gave us room for creativity in an attempt to allow the student to pick a particular area of interest to them and perform analysis on a dataset. My application domain was centered around that of text mining. In particular, I wanted to focus on song lyrics and any information I could gain related to what the lyrics of a song can tell us. For the programming portion of the assignment, I decided to go with Python to help me with certain advanced Natural Language Processing aspects. I also have found Python to be much faster in certain aspects, especially implementing customized programming applications to specific data, so I decided to use it instead of R due to the rather large amount of lyrics I would be going through. The specific algorithm I chose to work with was related to text mining and was calculating TF x IDF and computing similarities between documents. In addition I pretty much wrote all custom software in forming all data structures and computing graphs and statistical measures.

The Dataset:

Source: (https://raw.githubusercontent.com/walkerkq/musiclyrics/master/billboard_lyrics_1964-2015.csv)

I found it quite a challenge to find a good dataset to enable me to get creative. I found a couple that were interesting, however they weren't big enough to really capture finite conclusions. I happened to stumble across one that had the Top 100 Billboard Songs from 1964-2015. I found this to be a really cool one, not just in the sense that there were over 5000 songs, but it also gave a timeline that held valuable information related to these songs. It was also a pretty complete dataset, with about 4920 lyrics to songs out of 5100 being there. The structure of the data was as follows:

("Rank","Song","Artist","Year","Lyrics","Source")

The attributes that I planned to be of most importance were the Lyrics, Year, and Song. The song title was essentially the unique identifier to all the lyrics of each song, while the year gave the timeline of the songs which allowed me to analyze how songs have changed over time. Many questions were raised throughout the process of this lab and I plan to answer those questions I chose to pursue.

Data Structure Organization and Preprocessing:

The approach to organizing the lyrics and keeping track of which song had what lyrics was as follows. I would have a map where the key is the title of a song, and the value is a sorted list of tuples where the first value in the tuple is a word in that song, and the second is the number of occurrences of that word in the song. For example, it looks something like:

"song title" : [("lyric1",5), ("lyric2",4), ("lyric3",3)…]

I figured having them sorted in descending order would help with analyzing and finding more significant lyrics since the max occurrences could be easily accessed with this method. When first running this on all 5100 songs, there were over 41300 unique words used between all the songs when combined. Even after removing stop words, which I will discuss more in detail later, there were still 41066 unique words from all the lyrics combined. With so many words I found Python Dictionaries to be of most useful as words used as keys could be accessed very quickly. The above value of the song title ended up being a tuple value after being sorted, so finding words from the tuples ended up being much less efficient since it had to perform a search rather than a hash.

After all of the lyrics had been parsed and dictionaries of each song had been created, I wanted to combine all of the lyrics to get a global count of all of the lyrics from all the songs. This would represent all of the unique words present throughout all the documents combined together. However, Another problem I ran into with organizing all the data was that since dictionaries are unordered, I needed a way to map each word and its occurrence for calculations related to Tf-Idf. In my specific instance, the matrices used for those calculations were 100 rows for the 100 different songs of 2015, and 3465 columns for each unique word from all the songs combined. Therefore when iterating over songs and their lyrics, it wasn't sure to maintain the same order when iterating over them since maps are unordered. I had to then create a separate data structure to map each word to a position in the Tf-Idf matrices. After all of this was handled, the computations went relatively smooth.

It did turn out that some Lyrics weren't present in the data set for certain songs. These appeared as "NA" so if lyrics weren't incorporated, that song was just left out of the analysis.

Bag of Tokens Approach:

The bag of tokens approach was used to gather as much meaning out of the lyrics without worrying about context and natural language processing. This was used for the entirety of the programming application. An example of this comes within the top 100 songs of 2015 when looking at the top 5 words used by some of the songs in this era (sorry for explicitness in some songs).

```
the heart wants what i
('wants', 17)
('what', 14)
('heart', 13)
('and', 10)
('me', 8)
```

```
truffle butter:
('the', 19)
('a', 14)
('aint', 8)
('me', 8)
('right', 7)
```

```
one last time:
('you', 27)
('know', 17)
('to', 10)
('it', 10)
('time', 10)
```

```
what do you mean:
('what', 29)
('mean', 27)
('oh', 18)
('wanna', 10)
('but', 9)
```

```
the hanging tree:
('the', 12)
('are', 8)
('tree', 8)
('hanging', 6)
('midnight', 6)
```

```
flex ooh ooh ooh:
('it', 17)
('like', 14)
('a', 13)
('ooohh', 11)
('nigga', 11)
```

```
fight song:
('i', 13)
('song', 12)
('fight', 11)
('in', 9)
('ill', 7)
```

```
habits stay high:
('my', 20)
('the', 14)
('ooh', 12)
('you', 10)
('high', 9)
```

```
dont:
('my', 17)
('and', 16)
('dont', 15)
('she', 11)
('you', 11)
```

```
no type:
('the', 19)
('aint', 15)
('like', 14)
('you', 13)
('living', 9)
```

```
lean on:
('we', 22)
('lean', 15)
('need', 14)
('kiss', 12)
('gun', 12)
```

```
sugar:
('i', 19)
('and', 16)
('little', 14)
('on', 13)
('im', 11)
```

```
gdfr:
('for', 18)
('going', 17)
('it', 17)
('real', 14)
('i', 11)
```

```
chandelier:
('im', 13)
('123', 12)
('chandelier', 8)
('gonna', 8)
('like', 6)
```

```
earned it:
('it', 34)
('girl', 13)
('youre', 12)
('earned', 10)
('cause', 8)
```

```
only:
('the', 30)
('and', 23)
('in', 21)
('with', 16)
('bitches', 16)
```

```
shut up and dance:
('oh', 22)
('said', 15)
('me', 13)
('up', 13)
('shut', 12)
```

```
uptown funk:
('funk', 31)
('uptown', 29)
('just', 18)
('watch', 18)
('believe', 18)
```

```
blank space:
('a', 18)
('i', 15)
('its', 11)
('tell', 9)
('love', 8)
```

```
el perdon:
('que', 16)
('sin', 13)
('yo', 12)
('gusta', 12)
('te', 11)
```

```
house party:
('a', 19)
('to', 11)
('house', 11)
('gonna', 9)
('up', 7)
```

```
you know you like it:
('it', 18)
('know', 12)
('like', 11)
('what', 10)
('insane', 8)
```

```
ayo:
('like', 26)
('ayo', 22)
('her', 13)
('need', 13)
('she', 12)
```

```
watch me:
('watch', 77)
('ooh', 31)
('bop', 30)
('nae', 24)
('whip', 18)
```

Essentially it's extracting the occurrences of songs regardless of its context to draw meaning from which document it occurred in, how unique it was, and how often it occurred in the document.

Programming Application:

For the Programming portion of the lab I decided to go with a Proof of Concept with calculating the TF-IDF weighting for the top 100 songs just from 2015. I decided to do a POC because, after combining all 5100 songs from the past 52 years, there were 41061 unique words from all of the songs combined. That would mean the feature vector for the TF-IDF would have a dimension of 41061. Also to mention that at 4 bytes per integer value in a (41061 X 5100) dimension matrix, that is around 1 GB of memory just for the matrix, not to mention an immense amount of computation for calculating everything and creating other matrices so data doesn't get overwritten. Therefore, I reduced the matrix to (3631 X 100) for a much more manageable computation. And after removing stop words this further reduced to around
(3473 X 100). The calculation of the TF-IDF would then allow me to compare the similarities of different songs in extensions of the application.

The algorithm used was as follows:

1. Compute Term Frequency of Lyrics in each document
2. Perform normalization or word stemming processing on Unique words and documents
3. Compute IDF matrix
4. Compute TF * IDF matrix
5. Use TF * IDF matrix to help in other data analysis techniques such as computing similarity

TF(Term Frequency):

The term frequency of each word was discussed a little bit earlier, and was just a count of words in each of the document. Adjusting for variations in document length I attempted to use a double normalization. In order to do this I first saw that I already had all of the term frequencies for each of the songs (documents) I was analyzing, and they were in sorted order from the preprocessing stage described above. This game me easy access to the max for the term frequency normalization which was computed as follows:

$$\text{double normalization}(t,d) = 0.5 + \frac{0.5 \times f(t,d)}{MaxFreq(d)}$$

However, later on I decided not to stick with it and just see how the results would pan out with mainly raw data.

Sparsity of song lyrics:

After this I decided to look at how sparse the corpus was at different amounts of songs. These are described in the table below:

# of songs	% Sparseness with stop words	% Sparseness without stop words
2	44.2%	48.1%
5	80%	79.2%
10	85.7%	89.5%
25	91.8%	92.3%
50	94.7%	95.8%
100	96.5	97.6%

Sparseness of matrices by number of songs:

You will notice that sparsity actually increases here when the stop words are removed, because most of the songs have stop words in them, therefore the number of non zero entries will decrease, even though many columns of the matrix are getting deleted. For example, in the list of 2015 songs, there are 347300 entries in the matrix, and 338,992 entries that are zero. There were 215 stop words present that got deleted, so with one stop word having a column of 100 rows, that is 215*100 possible entries that got deleted or 21500 entries from the matrix. Of these 215 unique stop words, there were 4675 total occurrences of these words in the song lyrics.

Word Stemming:

Word Stemming was something I wasn't sure if I was going to implement or not, but I did decide to look into it a little bit. After some additional computations, the following Map determined the top words that were substrings of other words:

```
[('ear', 43), ('ill', 35), ('ive', 35), ('end', 34), ('sin', 34), ('eat', 32
), ('use', 28), ('man', 25), ('ate', 25), ('low', 21), ('sing', 20), ('cause
', 20), ('king', 20), ('win', 19), ('ass', 19), ('gin', 19), ('baby', 18), (
'son', 17), ('let', 16), ('ooh', 16), ('car', 16), ('hol', 16), ('yeah', 16)
, ('ring', 16), ('lie', 16), ('men', 15), ('day', 15), ('love', 14), ('don',
 14), ('ink', 14), ('hes', 14), ('time', 14), ('ari', 13), ('night', 13), ('
ice', 13), ('hey', 13), ('way', 13), ('row', 12), ('mea', 12), ('iti', 12),
('lay', 12), ('las', 12), ('chorus', 12), ('old', 12), ('body', 11), ('chin'
, 11), ('hit', 11), ('rob', 10), ('die', 10), ('run', 10), ('hot', 10), ('he
ll', 10), ('red', 10), ('que', 10), ('hand', 10), ('know', 10), ('hear', 10)
, ('home', 10), ('ima', 9), ('dan', 9), ('long', 9), ('head', 9), ('pos', 9)
, ('say', 9), ('mean', 8), ('got', 8), ('rap', 8), ('sec', 8), ('sea', 8), (
'walk', 8), ('need', 8), ('air', 8), ('rid', 8), ('real', 8), ('bit', 8), ('
play', 8), ('star', 8), ('try', 7), ('boy', 7), ('aye', 7), ('ooo', 7), ('he
art', 7), ('fuck', 7), ('dream', 7), ('gon', 7), ('dont', 7), ('able', 7), (
'ese', 7), ('town', 7), ('lets', 7), ('right', 7), ('stand', 7), ('act', 7),
 ('owe', 7), ('babe', 7), ('tie', 7), ('round', 6), ('tee', 6), ('leg', 6),
('law', 6), ('band', 6), ('care', 6), ('light', 6), ('sam', 6), ('rent', 6),
 ('lit', 6), ('girl', 6), ('ser', 6), ('arm', 6), ('tend', 6), ('ends', 6),
('por', 6), ('life', 6), ('look', 6), ('ski', 6), ('sit', 6), ('fore', 6), (
'count', 6), ('tip', 6), ('land', 6), ('did', 5), ('list', 5), ('oosh', 5),
```

So for example the string 'ear' was found in 43 other words in the list of unique words. After looking more into these, I found that most of the occurrences in the words they were found in, weren't plural, past, or future tenses. They were meaningless substrings such as the word "men" appearing in the word "amen." It didn't make sense to combine them, therefore, I decided to not take out the stems I found in the map above.

IDF:

The IDF was a The IDF was calculated as follows:

$$IDF(t) = \quad log(\frac{n}{k})$$

n — total number of docs

k — # docs with term t appearing
(the DF document frequency)

The top 10 IDF's associated with the top 100 songs for the Billboard Charts of 2015 were:

```
('im', 0.125)
('like', 0.149)
('just', 0.161)
('know', 0.161)
('got', 0.187)
('dont', 0.208)
('cause', 0.26)
('love', 0.26)
('baby', 0.319)
('make', 0.328)
```

```
 Number of Unique documents each word appeared in:
im: 75
like: 71
just: 69
know: 69
got: 65
dont: 62
cause: 55
love: 55
baby: 48
make: 47
```

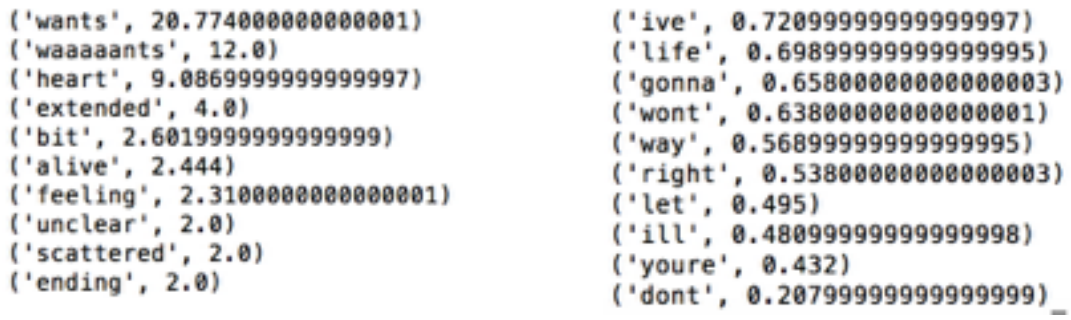
Once these words were determined, I decided to add more stop words to the “stop-word-list.txt” file since the words: “im", “just”, “got”, and “cause” didn’t really hold any value or meaning. After doing this I computed other results with a hope of obtaining more meaningful words. The new list is below:

```
('like', 0.149)
('know', 0.161)
('dont', 0.208)
('love', 0.26)
('baby', 0.319)
('make', 0.328)
('want', 0.357)
('tell', 0.367)
('oh', 0.387)
('wanna', 0.409)
```

```
 Number of Unique documents each word appeared in:
like: 71
know: 69
dont: 62
love: 55
baby: 48
make: 47
want: 44
tell: 43
oh: 41
wanna: 39
```

I found some of these values to be quite interesting, especially since these occurrences were only out of 100 songs. The words “like” and “love” hold a pretty significant meaning in the top songs, as this seems to be a common topic of interest for artists to sing or talk about. Since these occurrences are separate from their context it was difficult to get the exact meaning of what the song was actually talking about. This would be where the Natural Language Processing ToolKit could come in handy to analyze semantics, word order, and context to get a deeper understanding of the lyrics. Others words such as “want” and “wanna” are also interesting, because the use of these are associated with desires that one has, such as a commodity or object or an action they long for. These computations provided much more information to the lyrics and made me think of many more questions.

The final calculations made for this was obtaining the TF * IDF matrix. This was done by multiplying each entry in my IDF matrix by the count of each word in the song(document). The top 10 and bottom 10 TF * IDF words are displayed below:



Tf-Idf weights for song "the heart wants what it wants"

The words on the left are the higher weighted words computed by the algorithm for the song "The heart wants what it wants". This is just one example for the 100 that were in the matrix since it's somewhat difficult to visualize a matrix that is 100 X 3465. One thing to mention is that higher weighted words tended to appear in the title of the song just as it is seen above. A general idea can be extracted simply from the title of a song, however deeper details can be analyzed from all the lyrics associated with that song.

Other Questions

Explicit Lyrics Analysis:

One of the questions I wanted to try and answer was if I could find a specific year in which explicit lyrics began to significantly increase. This was done through comparing my lyrics dataset with a google text document with a list of around 700 flagged explicit words. This document was called "Terms-To-Block.csv" in the submission. I deleted around 100 or so words from the doc that were quite outlandish and wouldn't be in song lyrics that were on the billboard chart. And based on the graph below, this was much easier than I thought it was going to be. You will notice that after 1991, 1992 almost doubled the occurrence of explicit words in 1991, and then 1993 doubled 1992. I find it quite interesting that from 1965-1991 the amount of explicit words remained relatively steady at around 15-20, but then there was a jump to around 100 for years 1994-2000. Then from 2001-2015, the mean was around 120 with a max of 2004 being at 178. I would assume that in the range of 1991-1993 there was a new reformed law for censorship in music or media, or maybe this was the adoption of new hip hop that could have been more explicit than what was currently being played on radios. I'm sure there is an explanation, but since this isn't a history paper, I will leave the analysis at what it is.

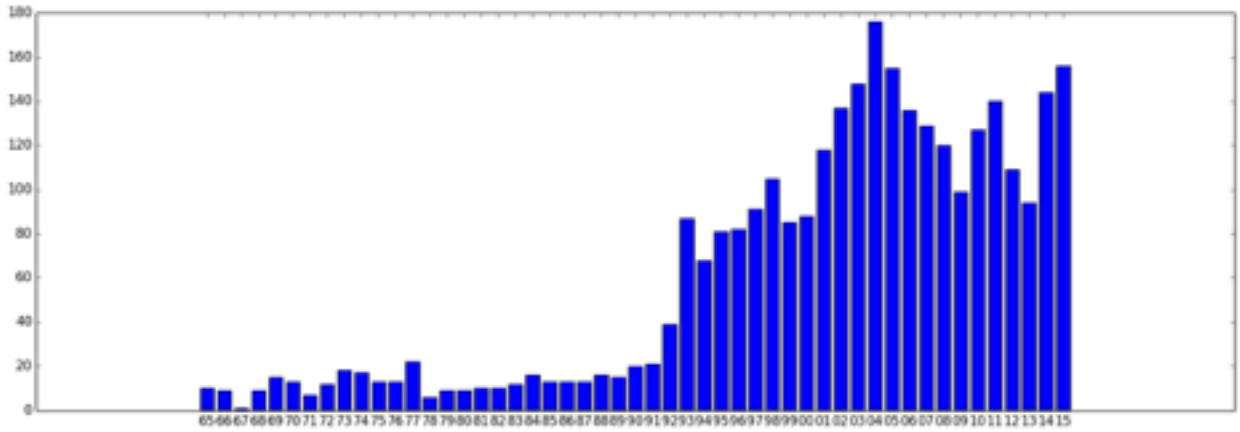


Figure A: Occurrence of Explicit Lyrics in Songs By Year

Extending the Application:

There are many tools available to further advance applications for specific analysis. In my case, I think it would be cool to use Python's NLPTK (Natural Language Processing ToolKit) package for more advanced processing and maintaining context of the song lyrics for more in depth analysis. A lot of information is lost when just doing a bag of words approach. Context can be very useful in the text mining domain, especially if you want to find correlations between songs with political ideas, or events that may have sparked popular songs to come to rise.

I also think it would be cool to obtain the lyrics to all of an artists songs. Then you can run text analysis on a specific artists lyrics and see what they use the most and tend to have songs about. You then could then run the Tf-Idf algorithm to compute similarities between artists. Obtaining the dataset for this would probably have a lot of web scraping involved as most artists lyrics aren't readily available in a data set.

Another thought was regards to the weighting of Tf-Idf and how it could help with creating an ANN model to predict and classify either the popularity of a song based off its content, or predict what year a song was written in.

Conclusion and Thoughts:

I have never worked with a dataset as big as the one in this project, and I can't stress enough how important code organization is for this, as that caused a lot of debugging time within writing this application. Another thing worth mentioning is the differences

between R and Python. I find R to be fantastic with regards to data visualization in comparison with Python. I found it much harder to read and understand my results with Python, however I thought it was easier writing the application to analyze the dataset in Python. Each has their pros and cons but Python was my go to on this one due to its performance advantage in my opinion. I find that this application, especially since its related to music, could be expanded in many ways and I may do look into it a little more after finals.