Information Retrieval and Text Mining
Emotion Mining and Topic Extraction on Lyrics

Christoph Emunds (i6146758)
Richard Polzin (i6145946)

June 9, 2017

Contents

1 Introduction

Leaving melody aside, there is a lot of information that can be extracted from a song's lyrics. In this work, we aim to measure the similarity of artists with respect to emotions and topics of their songs based on their lyrics. A suitable application could be as an alternative to data driven recommender systems in music streaming services. The developed techniques can be used to recommend artists invoking similar emotions or singing about related topics.

In Section 2, we give an overview over our dataset and explain how it was gathered. Section 3 describes the preprocessing that has been done. We mention the tools we used for the visualization in Section 4. The actual emotion mining is described in Section 5 and the topic extraction in Section 6. Finally the results are visualized.

2 Dataset

In this section an overview over the datasets used and the acquisition thereof will be given. Furthermore some comment about lyrics quality will be given.

The *pylyrics3* package (Wenzel, 2017) is able to download the lyrics for all songs of a given artist if it exists on *lyrics.wikia.com.* This website, online since April 2006, contains more than 1.5 million songs based on crowd-sourcing. It is integrated into many different media players including Windows Media Player, iTunes, Winamp and Amarok. (Colombo, 2017). The lyrics are returned in a dictionary, where the keys are song names and the values are lyrics. The usage of this package is straightforward and removes the necessity to deal with raw HTML data. The retrieved songs are stored in JSON format, each song having a title, the artist's name and the actual lyrics.

Some of the top 100 Billboard artists could not be resolved when querying pylyrics3. This could be due to spelling variations or incomplete/currently maintained entries. Moreover, we scraped the names of 30 metal bands from *got-djent.com* and requested their lyrics via pylyrics3. Some of them could also not be resolved, either due to spelling variations or because they are not popular enough to be listed. Altogether, we ended up with 11.??? songs from 120 artists.

In addition to the dataset we gathered ourselves, we were given a database of more than 16.000 artists and 228.000 originally built by Sibirtsev and Schuster (Sibirtsev & Schuster, 2014). In the remainder of this document, we refer to it as the external dataset.

Some of the lyrics are not very clean. For example, they contain markers for the chorus and verse part or additional information like *INTRUMENTAL* or *SOLO*. In some lyrics contact information of the creator is included and the formatting often varies between different authors.

Eminem - 5 Stars General :

> (feat. Shabaam Shadeeg, Skam, Kwest & A.L.)
> [Intro:]
> [man 1:] I like this song...
> [man 2:] All right! All right! All right everybody set off down!
> [man 1:] Oh, please!
> [man 2:] Come on now, now! KHM!
> [Chorus: Sample]
> Rhymes it all! [scratches]
> And every rhymes what I've heard it all before.
> [Shabaam Shadeeq:]
> Of course - I rap till I'm hoarse, add the sauce
> ...

Meshuggah - Dehumanization :

> A new level reached, where the absence of air lets me breathe
> I'm inverted electrical impulses. A malfunctioning death-code incomplete
> All things before me, at first unliving glimpse undeciphered
> Its semantics rid of logic. Nothing is all. All is contradiction
> Grinding, churning - the sweetest ever noises
> Decode me into their non-communication
> A soundtrack to my failure, one syllable, one vowel
> A stagnant flow of endings. Un-time unbound. Merging to form the multi-none
> A sickly dance of matter, malignantly benign. Greeting the chasm - unbearable, sublime

3 Preprocessing

This section goes into detail on the preprocessing that the lyrics went through. The different aspects and the toolkits are described.

All of the programming is done in Python 3 using different modules to interface existing NLP toolkits. Preprocessing the lyrics consists of several parts. The lyrics are tokenized, stemmed and stopwords are removed. Furthermore a TF-IDF feature matrix is calculated and the language of songs is detected. Most of the preprocessing steps are independent and can be executed in parallel.

For tokenization and stemming, the *nltk* Python package (NLTK-Project, 2017) has been used. The lyrics are split into sentences and the sentences are split into words. For stemming nltk implements several algorithms. In our implementations, we use the Porter algorithm.

Nltk also provides a stopword list, but it did not yield satisfying results, so we sourced a list of stopwords that are more appropriate for lyrics. Moreover we tailored it to the specific lyric domain by iteratively adding stopwords that we found to obfuscate results.

In addition to tokenization, stemming and stopword removal we built TF-IDF features utilizing the *scikit-learn* package (Pedregosa et al., 2011). With the *TfIdfVectorizer* class a matrix of TF-IDF features can be created from a collection of raw documents.

For the topic extraction, all songs should be in the same language. Otherwise specific topics for the languages will be created, which was not the goal of the project. Therefor we needed to filter out lyrics that are not in English. In our dataset, all songs, except for one, are in English. For the external dataset, we implemented a rudimentary language detection using the *langdetect* Python package (Danilak, 2014). This package is a port of Google's language detection library.

To summarize the nltk toolkit is used for tokenization and stemming. A custom stopword list was developed and scikit-learn was utilized for TF-IDF features. Finally langdetect was used for language detection. After preprocessing we have available for every song the original text and estimated language and sentence and word tokens without stopwords and reduced to the word stems.

Various metrics, like flesch score, fog-score and flesch-kincaid levels, were implemented, but not used in the project. Therefore they won't be discussed in this report.

4 Visualization

In this section the tools used to visualize our results will be described and problems encountered using them will be discussed. Two different frameworks have been used for the visualizations which will be discussed in sequential order.

All the complex visualizations were created using the open-source platform *Gephi* (Bastian, Heymann, & Jacomy, 2009). Gephi is able to display a plethora of graphs and networks that can also be explored interactively. It provides a rich set of features, including filtering out edges with low weight, calculating modularity classes for the nodes in a graph, ...

To be able to import our results to Gephi, we made use of the *pygraphml* Python package. It provides a couple of methods to dump the extracted information into a *GraphML* file, that can be imported by Gephi. However, the pygraphml package is very rudimentary. An issue was that the weight attribute that has been added to edges is stored as a string, rather than a number, which makes it impossible for Gephi to recognize the edge weight properly. We circumvented this problem by simply exchanging a single line in the beginning of the GraphML file.

The second framework that was used is *word_cloud* (Mueller, 2017). It provides a Python interface to generate wordclouds from text. It was used in conjunction with *flask* (Ronacher, 2017) to spin up a local webserver that can be used to browse emotion and vocabulary wordclouds for the different artists. Wordclouds are created before through the main scripts and are just displayed in HTML, as creating them in real time added significant delay when exploring artists interactively.

To summarize pygraphml was used to export data to Gephi where visualizations involving the complete dataset were created. In addition wordclouds have been generated for the emotions and vocabularies of all individual artists. Those can be explored through the filebrowser ore more conveniently a local webserver created in flask.

5 Emotion Mining

An overview over the exact emotion mining workflow is given in this chapter. Furthermore first results are shown and discussed.

The emotion mining is based on the preprocessed data. The tokenized lyrics are used and every token is looked up in an emotion lexicon for its emotional value. For our experiments, we used the *NRC Emotion Lexicon* (EmoLex) (Mohammad & Turney, 2013).This lexicon specifies the emotional value for the eight emotions *anger*, *anticipation*, *disgust*, *fear*, *joy*, *sadness*, *surprise*, and *trust*.

The emotion mining we apply to the lyrics is only rudimentary. It does not handle negations, sarcasm or complex syntactical structure. We were not able to find either an emotion lexicon that specifies the opposite emotional value for every common emotion word, or suitable work that is related to the our specific need for negated emotion words.

This analysis of the tokenized songs resulted in a vector containing values for the eight emotions. These eight-dimensional emotion vector were then summed up and normalized for every artist in the dataset. For every artist the result was a vector containing information about how strong each of the eight emotions is in his songs.

To calculate similarity between artists we used the cosine distance in the eight-dimensional space.

5.1 Results

Figure 1 shows a graph of all 120 artists, where each artist is connected to every emotion. The edges themselves have weights depending on the strength of the emotions in an artist's song. The weights are normalized between 0 and 1.

The full graph is overwhelming and detailed. It does not yield any useful insight. Therefore, Figure **??** shows a graph that contains only those edges with sufficiently high weights. Our experiments showed that a threshold of 0.3 yielded a balanced mixture of complexity and conciseness.

Figure 3 shows the emotion similarity between individual artists. The modularity score separates the graph into three main regions, which...

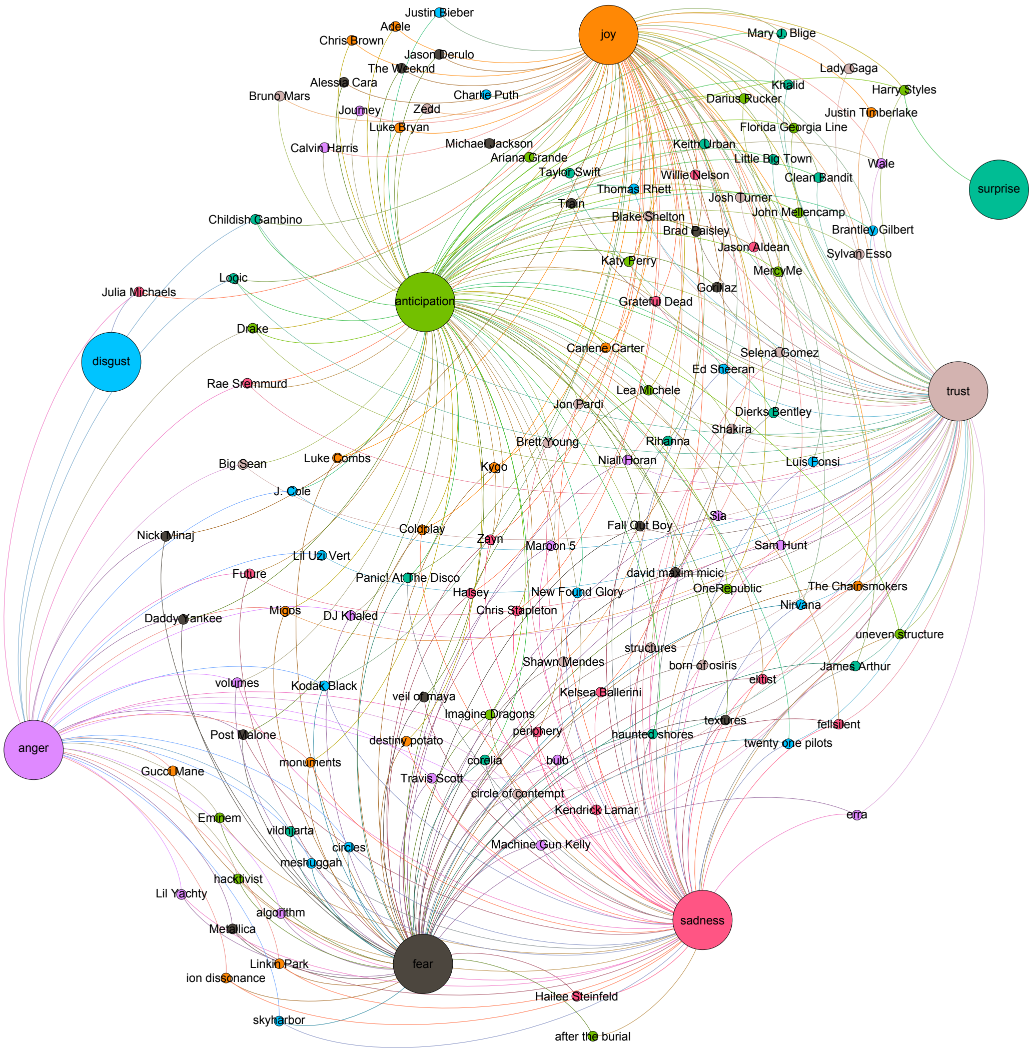

Figure 1: Emotion graph with edge weight greater than 0.37

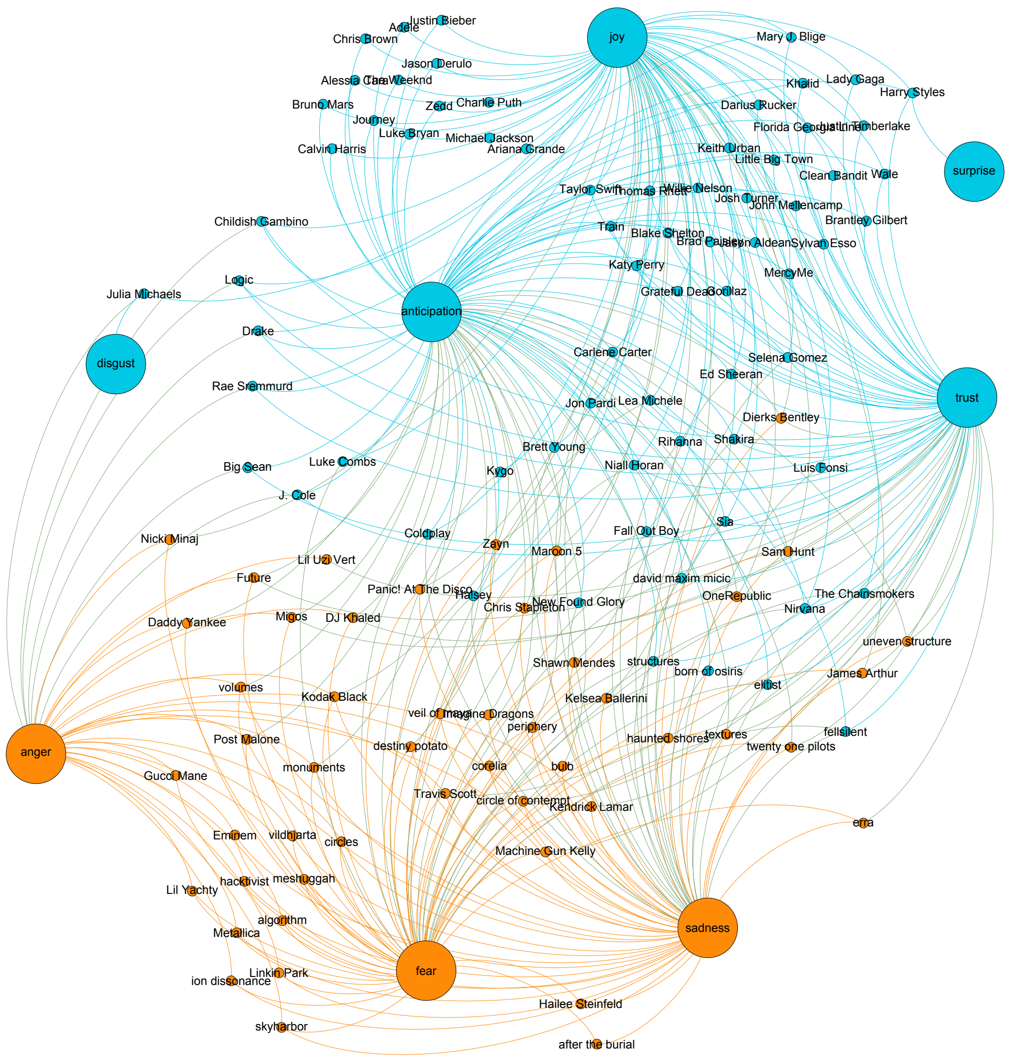

Figure 2: Emotion graph with recalculated modularity

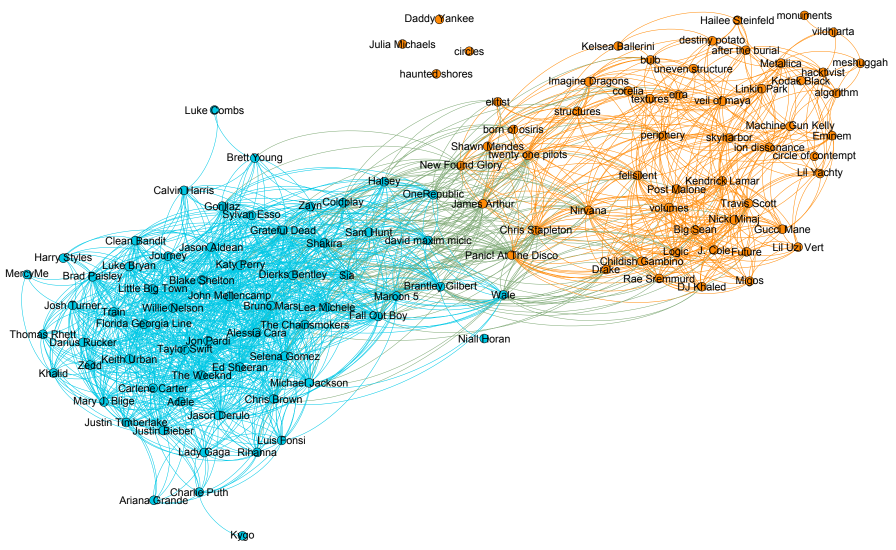

Figure 3: Emotion similarity between artists greater than 0.68

6 Topic Extraction

The Non-negative Matrix Factorization (NMF) algorithm has been used to extract potential topics from the lyrics. The *sklearn* Python package provides everything necessary to apply NMF to the data. First, a tf-idf feature vector for each song text is calculated. Afterwards, these are fed into the NMF algorithm, which returns two matrices that...

We found that a number of ten topics is appropriate for our dataset. When choosing more topics, they start to overlap too much. When choosing fewer topics, too much information is put into every single topic. This is of course subject to the experimentalist's taste.

6.1 Results

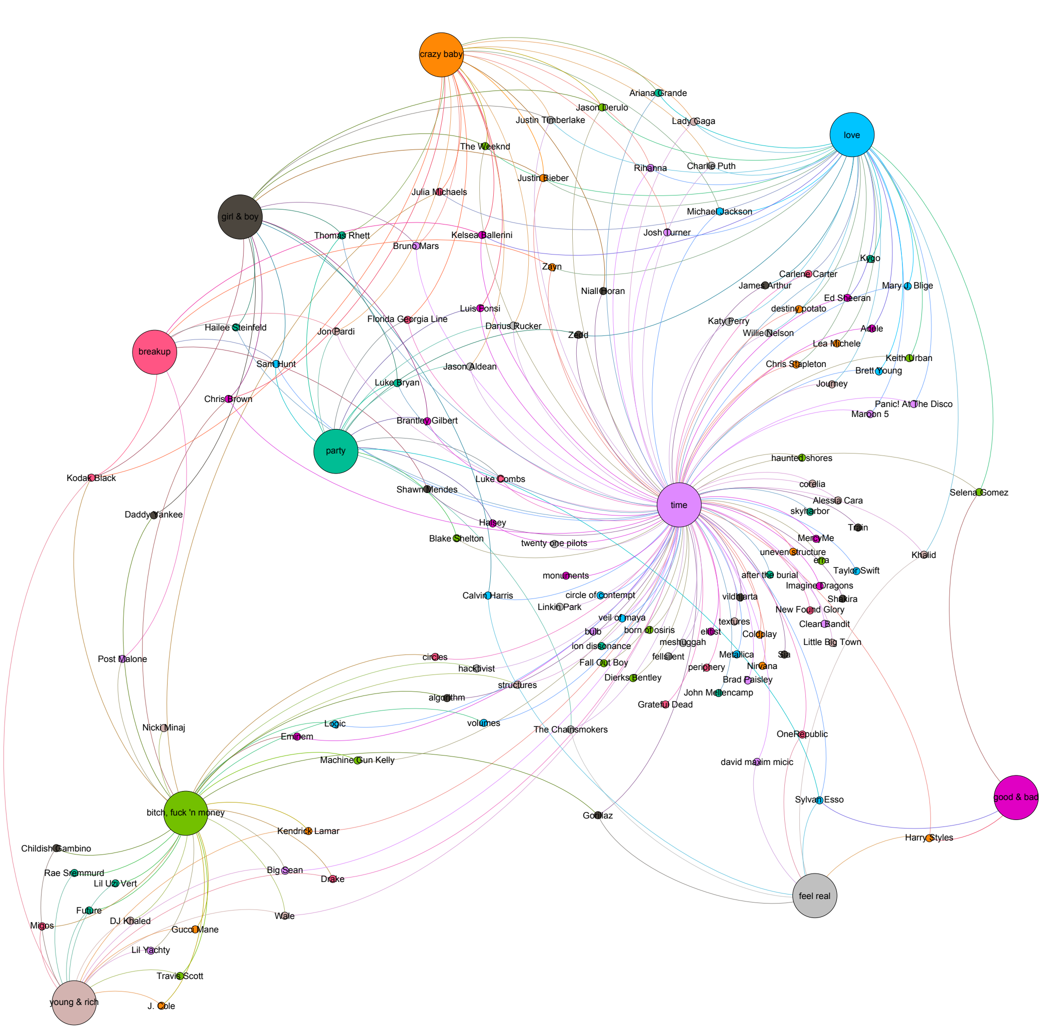

Figure 4: Topic graph with edge weights greater than 0.32

Figure 6 shows the similarity of topics between artists. It is interesting to see that a lot of the rappers form their own subgraph, that is completely isolated from the rest of the graph.

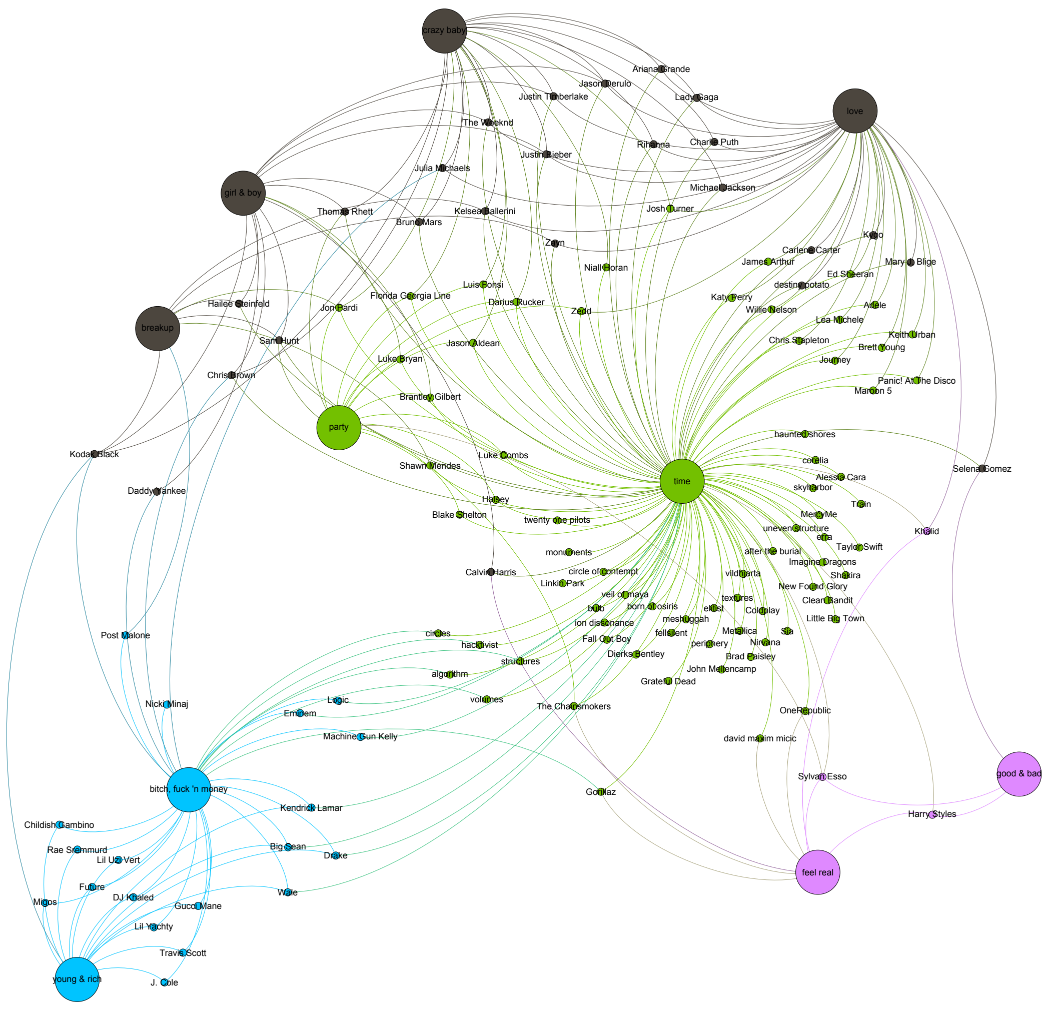

Figure 5: Topic graph with recalculated modularity

6.2 Evaluation

We tried to evaluate the resulting topics via held-out tests.

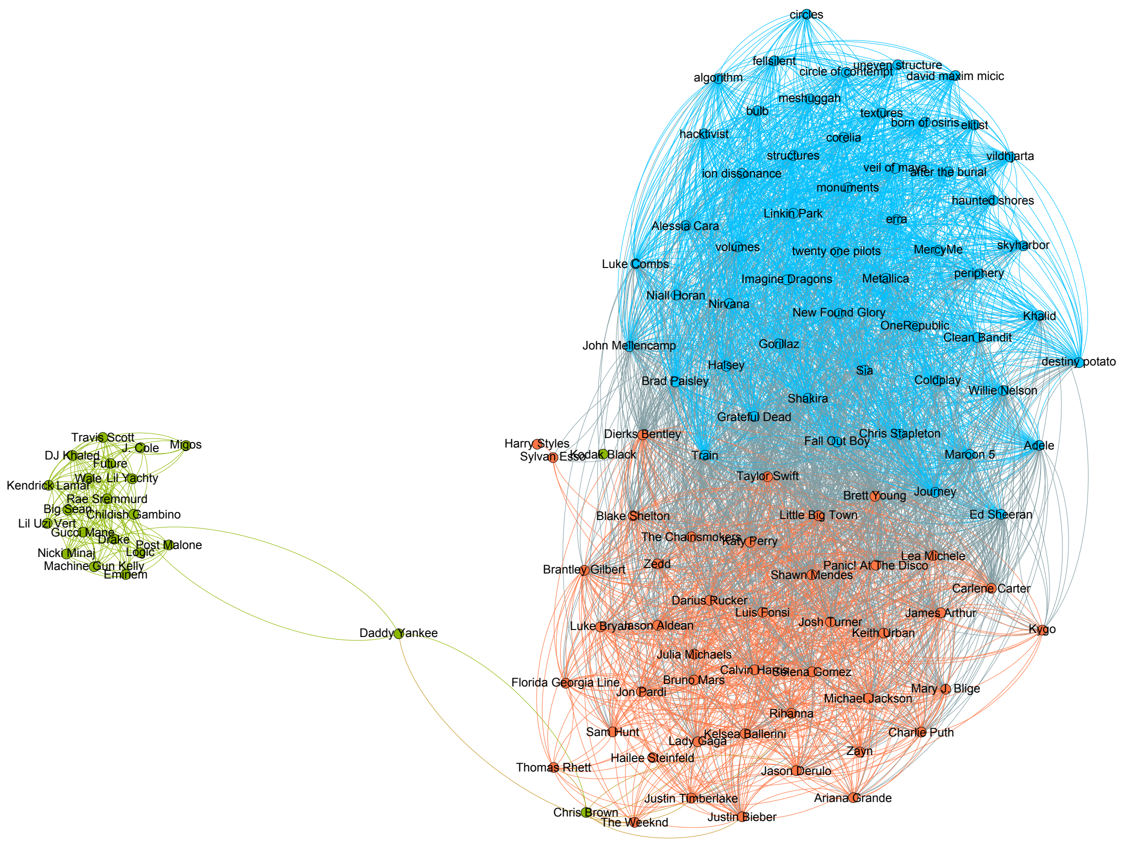

Figure 6: Topic similarity between artists greater than 0.82

7 Other techniques

In this section we want to mention some of the techniques we used that were neglected as we settled for the approach we took in the end. POS Tagging Named Entity Extraction RAKE Summarization Text statistics (Flesch Score, etc.)

8 Conclusion

References

Bastian, M., Heymann, S., & Jacomy, M. (2009). *Gephi: An open source software for exploring and manipulating networks.* Retrieved from `http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154`

Colombo, S. (2017). *Lyricwikia - fandom powered by wikia.* Retrieved from `http://lyrics.wikia.com/wiki/Lyrics_Wiki`

Danilak, M. (2014). *Language detection library.* Retrieved from `https://pypi.python.org/pypi/langdetect`

Mohammad, S. M., & Turney, P. D. (2013). Crowdsourcing a word-emotion association lexicon. , *29*(3), 436–465.

Mueller, A. (2017). *A little word cloud generator in python.* Retrieved from `https://github.com/amueller/word_cloud`

NLTK-Project. (2017). *Natural language toolkit.* Retrieved from `https://www.nltk.org/`

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Ronacher, A. (2017). *A microframework based on werkzeug, jinja2 and good intentions.* Retrieved from `http://flask.pocoo.org/`

Sibirtsev, & Schuster. (2014). *Information retrieval and text mining with lyrics.*

Wenzel, J. (2017). *Pylyrics3 - lyric scraper based on py-lyrics, updated for python 3.* Retrieved from `https://github.com/jameswenzel/pylyrics3`