Johannes Gutenberg-Universität Mainz
Gutenberg-Institut für Weltliteratur und schriftorientierte Medien
Seminar: Natural Language Processing
Winter semester 2021/22
Professor: Dr. Demival Vasques Filho

The Most Common Themes in Chart Songs of 2010–2020

Project of Nádia dos Santos Ossenkop and Lea Justine Rußwurm

Introduction

Most people listen to music at one point or another during their daily routine. Some listen to it in their free time, others listen to the radio during their commute to work, others again while doing household chores or while working. Songs that are listened to and downloaded more often than others end up in the charts, where they stay until a new song takes their place.

But what compels people to listen to this song more often than to other songs? Do they have a common theme? In this project we looked for prevalent theme in songs that made it into the top 10 charts in Germany in the past decade. Due to the many dimensions of music – its rhythm, the instruments used, its genre, etc – we decided to focus on the songs' lyrics for this project.

Analysing the Lyrics

First, we needed a corpus with the lyrics we wanted to analyse. Since there was none that suited our needs, we had to create our own corpus. To do this, we made a Spotify playlist[1] containing the top ten songs in the German charts from 2010 to 2020[2]. Excluding duplicates, we ended up with a playlist of around ninety songs in total.

To extract the lyrics from our playlist, we wanted to use the Python library *Spotipy*[3] that works with a Spotify web API. To do this, we had to register our application on Spotify for Developers to receive a Client ID and a Client Secret that would allow us to use Spotify's API. However, Spotipy alone could not extract the lyrics from our playlist, so we added the Python client *LyricsGenius*[4] to access the lyrics and annotations stored on Genius.com. We wrote the code with the help of a YouTube guide,[5] but since that code was only for showing the lyrics in the terminal while the song was playing, it had to be modified.

The biggest difficulty was finding the right library and code to automate the lyric extraction, since searching for each song individually to extract the lyrics would have taken too long and been too much

[1] The playlist in question can be found here: https://open.spotify.com/playlist/4tAECi1mLIUg4qh419jnGA?si=c252b9e0ddf44ca5. [06.02.2022]
[2] Offizielle Deutsche Charts. Top 100 Single-Jahrescharts, 2010–2020: https://www.offiziellecharts.de/charts/single-jahr/for-date-2010. [06.02.2022]
[3] Spotipy Doc: https://spotipy.readthedocs.io/en/2.19.0/. [06.02.2022]
[4] LyricsGenius: a Python client for the Genius.com API: https://pypi.org/project/lyricsgenius/; for more information see the LyricsGenius Doc: https://lyricsgenius.readthedocs.io/en/master/ [both last accessed 06.02.2022].
[5] Elbert: How to automate with Python, Spotipy, and LyricsGenius, in: YouTube, https://www.youtube.com/watch?v=cU8YH2rhN6A&t=1662s. The GitHub repository can be found here: https://github.com/rappercodes/lyrics-app [both last accessed 06.02.2022].

work. Using the Client ID of both the Spotify API and the Genius API, we could automate the search through the Genius site to look for the song / name of the artist of the song currently playing on Spotify (that is the scope from Spotify "user-read-currently-playing"). The search criteria included songs featuring other artists. While the song is playing on Spotify, the Genius access token searches for it (*genius.search_song(title=song_title, artist=artist_name)*), and when it is found, it is printed in the terminal (*song.lyrics*) and then saved as a .json file (*song.save_lyrics()*) with all the information given by the Spotify-token as a dict. We saved the data as .json files instead of .txt files so we could access other key values later if we wanted to; however, when saving all the lyrics onto one list variable, we ended up saving them as a .txt file anyway because that was easier to work with. Four of the songs on our playlist didn't come with lyrics so for the purpose of this project, we decided to exclude them. We wanted to automate the code so that the lyrics would be recorded without needing to play the song, but this improvement in the code was not possible. As a result, we had to listen to the songs while the code was being executed.

Next, we needed to clean the lyrics but did not want to go through each .json file manually. We went looking for a solution and finally settled on using the *os.listdir()* function of Python. Using this, we could get just the lyrics from each .json file and save them into one .txt file. This is the file we used for any further analysis. Sadly, it is possible that some data was lost in the process, such as the ability to distinguish between songs or to call up the title or artist of a song.

The .txt file still contained elements we did not want, such as "[Chorus]" or "[Verse]" annotations, so we removed them and any other unwanted element as best as possible using the *.replace()* function.

To figure out where to start with our analysis, we wrote two pieces of code: One would take a user input, go over the .txt file and count how often the input word appeared in the text. This was useful, but slow and inaccurate since it depended on the user guessing the most common words and writing them the same way they appeared in the file. So we looked for an alternative that would go over the text and give us a count of the most used words and with the help of PythonGuides[6] and Stackoverflow, we found some code that could do just that by using the *counter* of the *collection* library in Python. We could then print the most common nouns and tokens sorted by frequency with the corresponding frequency beside it.

Using this information, we tried working with the *matcher* function of *spacy,* but didn't get great results. We tried different patterns such as *pattern = [{"LEMMA": "love"}, {"POS": "VERB"}]* or *pattern = [{"LEMMA": {"IN": ["love", "you"]}}, {"POS": "NOUN"}],* but the results weren't useful for analyzing the songs since they gave no context and could thus not be used to determine the theme of the song. Using a pattern with empty dictionaries (such as *patternTwo = [{}, {"LEMMA": "tonight"}, {}]*) proved marginally more successful, but still didn't give back viable results.

[6] Singh, Vineet: Python Count Words in File, in: PythonGuides, 12.01.2021. https://pythonguides.com/python-count-words-in-file/ [06.02.2022]

Spacy also provides a *Dependency Matcher* that allows one to match words based on their dependency and not just their proximity.[7] Using the common word “love” as an anchor, we tried to find out in which contexts it is used to determine the theme. But while our dependency matcher worked for matching two dependent words, we failed when trying to implement a third chain to the matcher, so the results are again not conclusive.

Results and Conclusion

The automated search of the most frequent words showed that “baby”, “love”, “boy”, “time” and “heart” are the most frequently used words in our data sample, each of them being used between a hundred to up to two hundred times. The numbers decline after this; while “night”, “girl” and “boom” are still counted around eighty times, “way”, “world”, “life” and “tonight” range from around sixty to seventy times. It’s noticeable that words associated with love and relationships are the most prevalent words in our song lyrics and that most refer to other people.

Neither the matcher nor the dependency matcher gave back more conclusive results. This is probably due to a combination of multiple factors, such as possible the sometimes creative syntax of song lyrics throwing the matchers off and to vague patterns that weren’t able to do what we wanted them to. If we were to repeat this project or continue it, we would have to refine those patterns and also look for other, more efficient means of determining the theme of the songs.

Since both matchers more or less failed, we have to rely most on our counter and thus conclude that the prevalent themes of songs in the charts of the last ten years are love and relationships.

[7] Finding linguistic patterns using spaCy, in: Applied Language Technology. https://applied-language-technology.mooc.fi/html/notebooks/part_iii/03_pattern_matching.html#matching-syntactic-dependencies [06.02.2022]

Sources

Elbert: How to automate with Python, Spotipy, and LyricsGenius, in: YouTube, https://www.youtube.com/watch?v=cU8YH2rhN6A&t=1662s. The GitHub repository can be found here: https://github.com/rappercodes/lyrics-app [both last accessed 06.02.2022].

Finding linguistic patterns using spaCy, in: Applied Language Technology. https://applied-language-technology.mooc.fi/html/notebooks/part_iii/03_pattern_matching.html#matching-syntactic-dependencies [06.02.2022]

LyricsGenius: a Python client for the Genius.com API: https://pypi.org/project/lyricsgenius/; for more information see the LyricsGenius Doc: https://lyricsgenius.readthedocs.io/en/master/ [both last accessed 06.02.2022].

Offizielle Deutsche Charts. Top 100 Single-Jahrescharts, 2010–2020: https://www.offiziellecharts.de/charts/single-jahr/for-date-2010. [06.02.2022]

Singh, Vineet: Python Count Words in File, in: PythonGuides, 12.01.2021. https://pythonguides.com/python-count-words-in-file/ [06.02.2022]

Spotipy Doc: https://spotipy.readthedocs.io/en/2.19.0/. [06.02.2022]