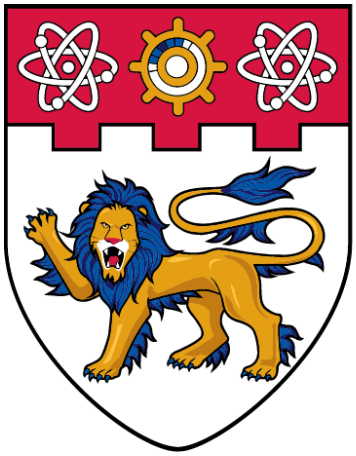


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ 4034 - Information Retrieval Project

🎵SongGenius🎵

Group 10

Mantri Raghav (U1822309B)
Nariani Mayank(U1722248K)
Dwivedee Lakshyajeet (U1822289L)
Tharakan Rohan Roy (U1822028E)
Lakshmanan Krithika Lakshmi (U1722088H)
Loh Han Yang (U1820738J)

1. Introduction

While streaming services like Spotify and Apple Music are able to provide a breadth of music and accompanying lyrics to users - they do little to present analysis on the underlying sentiment of songs. Song Genius is an application that allows users to search for songs based on artists, lyrics, genre, and sentiment. The application searches through a comprehensive database and presents the user with related songs which they might like. Our search results also include YouTube videos of the songs, links to the YouTube website of the song as well as details about the artist. If the user is trying to avoid explicit songs, an explicit rating out of 100 is also shown for them to listen at their own discretion.

Our dataset is created using the LyricsGenius API which fetches songs from genius.com, the most comprehensive and reliable dataset for songs and artists. Our data is created by collecting the most popular artists and songs in the past few years which ensures that users usually find relevant results.

Through our sentiment analysis models, we are also able to assign moods to songs (positive/happy, negative/sad, and neutral) in case the user would like to only listen to songs with a certain sentiment. We also use data analysis and visualisation to show trends in songs such as analysing the trajectory in sentiments over time, the relation between the genre and overall sentiment, the progression of artists sentiments in their lyrics over time.

Our project consists of data crawled using Python, which is then indexed using Solr, and a user interface which is created using Streamlit. Additionally, each song is assigned a sentiment and explicit rating using NLTK and Spacy.

GitHub: https://github.com/Blitzborg/Sentiment-Song-Search

Video: https://youtu.be/-md2pWfwLDU

Data:
https://drive.google.com/file/d/19yp4V89Xa04_ZNxqiWpWeunCTITszF7l/view?usp=sharing

Source Code*:
https://drive.google.com/file/d/1JMAkdEogNTUkL1I7HkLb7r5dMagUYRtK/view?usp=sharing

* Installation instructions can be found in the README.md file on GitHub as well as the code.zip folder

2. Crawling

a. Details

Data crawling refers to the process of collecting large amounts of data. In our project, the information of songs are collected through the LyricsGenius API that has the largest collection of song lyrics and musical knowledge.

b. Question 1

The LyricsGenius API is used to acquire the 50 most popular songs of the top 400 artists. The artists have been chosen from a wide range of years in order to gain a more thorough understanding of the evolution of songs and a more coherent dataset. Right from the classics of Elvis Presley and the Beatles to the modern songs of Pitbull and Justin Bieber, the dataset contains a wide range of artists. The songs are stored in json files. The code snippet is shown as below:

```
import lyricsgenius
import pandas as pd


genius = lyricsgenius.Genius('eMiXYROGF3D5dBmYyjqke6STCiR3MoWJ6sMZFwe4AG14vIwLvuZ8qRRzK157HKcd')

list_artist =['Adele', 'Joey + Rory', 'Draaco Aventura', 'Justin Bieber', 'Peer van Mladen', 'Chris Janson', 'One Direction', 'Drake
        'Zac Brown Band', 'Hearty2Raw', 'Future', 'Lady Antebellum', 'Adam Levine', 'Janet Jackson', 'Keith Urban', 'Sam Smith', 'Ja
        'Madelyn Victoria', 'Alicia Keys', 'Avicii', 'Bryan Adams','Faith Hill', 'MGK', 'Scarface', 'Austin Mahone', 'Disclosure', '
        'Luther Vandross', 'Major Lazer', 'Master P', 'Omi', 'Olivia Newton-John', 'Ricky Martin', 'Ronnie Dunn', 'The Rolling Stone
        'Jay Z', 'Lady Gaga', 'Lil Fizz', 'Iggy Azalea', 'Chase Rice', 'Danielle Bradbery', 'Dustin Lynch', 'Kacey Musgraves']

j=0
for i in list_artist:
  try:
      artist = genius.search_artist(i,max_songs=50,sort='popularity')
      artist.save_lyrics('artist'+str(j))
      j=j+1
  except:
       print('CANNOT')
```

Fig. 1: Code for retrieving data

The json files are read sequentially and only valuable information about the song and artist are captured. For example, the songs contained redundant information such as studio, filming director and images that are irrelevant to the users of our web application. Hence, the following attributes have been captured:

Artist	Song	Lyrics	Youtube Link	Date	Description	Featured Artist	Genre

Fig. 2: Fields present in the dataset

The dataset can be summarised as shown in the table below:

Number of records	10,002
Number of words	3,147,475

Number of artists	339
Number of genres	46

Table 1: Dataset statistics

3. Indexing and Querying

a. Details

The crawled data is indexed using Solr+Lucene+Jetty. The details of the database are as follows:

a. **Fields:** Our dataset contains a total of 9 fields. We use some of these fields for indexing while we don't use some since they are only relevant for the UI and we do not want search results to be dependent on them as well as increase our index size. A summary of the fields is shown in Table 2 and Fig. 3.

S. No.	Field Name	Stored	Indexed	Comments
1	_artist	✓	✓	Name of singer/band
2	artist_details	✓	✗	Short description of the artist
3	song_name	✓	✓	Name of song
4	lyrics	✓	✓	Lyrics of the song
5	link	✓	✗	YouTube link
6	date	✓	✗	Date of song release
7	sentiment	✓	✓	Positive/Negative/Neutral
8	genre	✓	✓	Genre of the song
9	explicit_rating	✓	✗	Explicit rating out of 100. Only used by UI

Table 2: Summary of fields stored by Solr database

```
<field name="_artist" type="text_general" uninvertible="false" indexed="true" stored="true"/>
<field name="_nest_path_" type="_nest_path_"/>
<field name="_root_" type="string" docValues="false" indexed="true" stored="false"/>
<field name="_text_" type="text_general" multiValued="true" indexed="true" stored="false"/>
<field name="_version_" type="plong" indexed="false" stored="false"/>
<field name="artist_details" type="text_general" uninvertible="false" indexed="false" stored="true"/>
<field name="date" type="string" uninvertible="false" indexed="false" stored="true"/>
<field name="explicit_rating" type="pint" uninvertible="false" indexed="false" stored="true"/>
<field name="genre" type="string" uninvertible="false" indexed="true" stored="true"/>
<field name="id" type="string" multiValued="false" indexed="true" required="true" stored="true"/>
<field name="link" type="string" uninvertible="false" indexed="false" stored="true"/>
<field name="lyrics" type="lyrics" uninvertible="false" indexed="true" stored="true"/>
<field name="sentiment" type="string" uninvertible="false" default="neutral" indexed="true" stored="true"/>
<field name="song_name" type="text_general" uninvertible="false" indexed="true" stored="true"/>
```

Fig. 3: Fields defined in the Solr schema

b. **Indexing:** When creating a text-based database, the input is indexed in a particular data structure to make searching faster. To reduce the size of this data structure and to normalize the corpus, certain processing steps are used.
In Solr, we use an *analyzer* to chain a series of processing steps to fields before indexing them. All indexed fields are first tokenized to obtain a sequence of tokens. Table 3 shows the further processing steps for each field.

S. No.	Field Type	Fields	Processing Steps
1	string	date, genre, link, sentiment	N/A (no tokenization either)
2	text_general	artist_details, song_name, _artist	Lower Case
3	lyrics	lyrics	ASCII Folding (á -> a), PatternReplaceFilter (e.g. [Verse 1], apostrophe, period, etc.), ClassicFilterFactory (U.S.A. -> USA), Lower Case
4	pint	explicit_rating	N/A (no tokenization either)

Table 3: Processing steps applied to different fields

c. **Searching:** When a query is sent to Solr, it must also be processed in a similar way as the respective field was during indexing. This ensures that the text being searched for in the index is in the same form.

During searching, the processing steps used on the queries are the same as those used in indexing except for the *lyrics* field where *ASCII Folding* is not used since the user is unlikely to enter a diacritic, *PatternReplaceFilter* is not used since user won't be entering [Verse 1], etc. and the *ClassicFilterFactory* is removed since the user is unlikely to use periods or apostrophes. Fig. _ shows the final processing steps used for querying and indexing each field.

```
<fieldType name="string" class="solr.StrField" sortMissingLast="true" docValues="true"/>
```

```
<fieldType name="text_general" class="solr.TextField" positionIncrementGap="100" multiValued="true">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" words="stopwords.txt" ignoreCase="true"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" words="stopwords.txt" ignoreCase="true"/>
    <filter class="solr.SynonymGraphFilterFactory" expand="true" ignoreCase="true" synonyms="synonyms.txt"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

```
<fieldType name="lyrics" class="solr.TextField">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.ASCIIFoldingFilterFactory"/>
    <filter class="solr.PatternReplaceFilterFactory" pattern="[(.*?)]" replacement=""/>
    <filter class="solr.ClassicFilterFactory"/>
    <filter class="solr.PatternReplaceFilterFactory" pattern="['',-.]" replacement=""/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.PatternReplaceFilterFactory" pattern="['',-.]" replacement=""/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>
```

```
<fieldType name="pint" class="solr.IntPointField" docValues="true"/>
```

Fig. 4: Processing steps used for different field types during indexing and querying

d. Ranking: Once documents which match a query are retrieved, they must be ranked in order to show the most relevant results at the top.

For ranking, we utilise the BM25 (Best Match v25) algorithm which is an improvement upon the TF-IDF algorithm. The IDF section of BM25 remains unchanged. The TF portion of BM25 is further dampened as compared to TF-IDF. The TF term follows equation (1) where *K*(=1.2) and *b* are constants. As TF increases, the TF-score asymptotically approaches K. In regular TF-IDF, the TF-score would consistently increase and never reach a saturation point.

The TF-score is also influenced by the length of the document with respect to the average document length of the corpus, represented by *L*. This means shorter documents can reach the asymptote faster while longer documents require more term instances to saturate.

$$TFScore = \frac{(K+1) \times TF}{K(1 - b + b \times L) + TF} \quad \text{- (1)}$$

b. Question 2

i. Build a simple web interface for the search engine

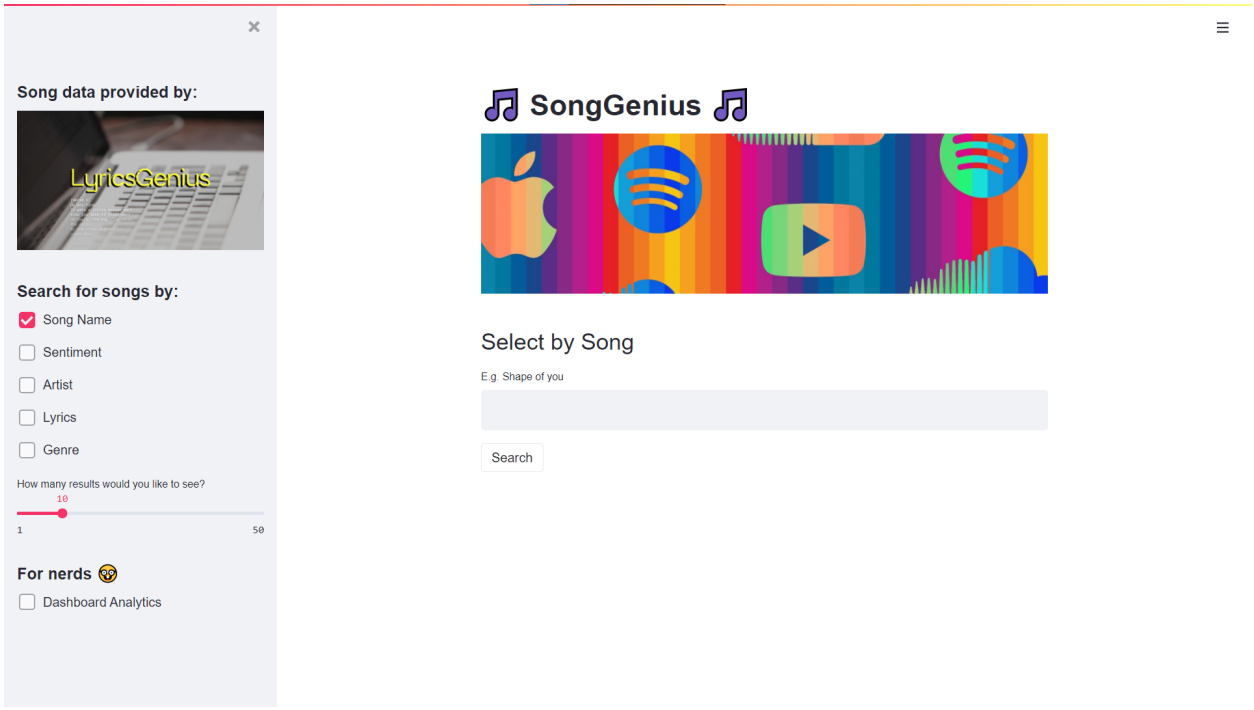


Fig 5: A full page screen grab of the UI

The user can search through our static database the songs that we have collected using the user interface.
The UI has been developed using the python framework known as Streamlit, and also using separate HTML and Bootstrap4 components within the Streamlit app.

Several options have been given for searching results, and they have been separated for a more user-friendly experience. A user can search using just one or several criteria at once. The criteria are -

- Searching by song name
- Searching by sentiment
- Searching by artist name
- Searching by lyrics
- Searching by genre

For limiting/increasing the number of search results to be visible, the user can use the slider on the sidebar to decide how many search results are to be rendered by the UI. By default, a maximum 10 are displayed, which can be pushed over to 50 upon the user's discretion.

Another option of seeing analytics about the data is given in the sidebar. More about this feature is explained in question 3.

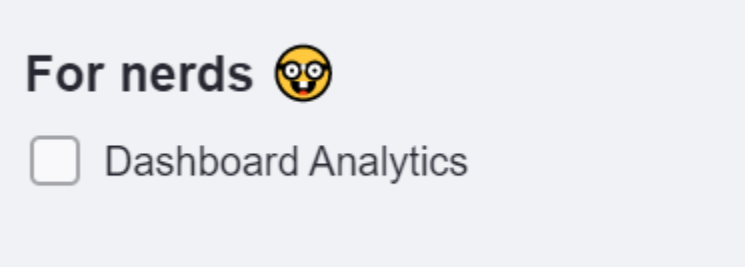


Fig 6 : Sidebar checkbox to see dashboard analytics

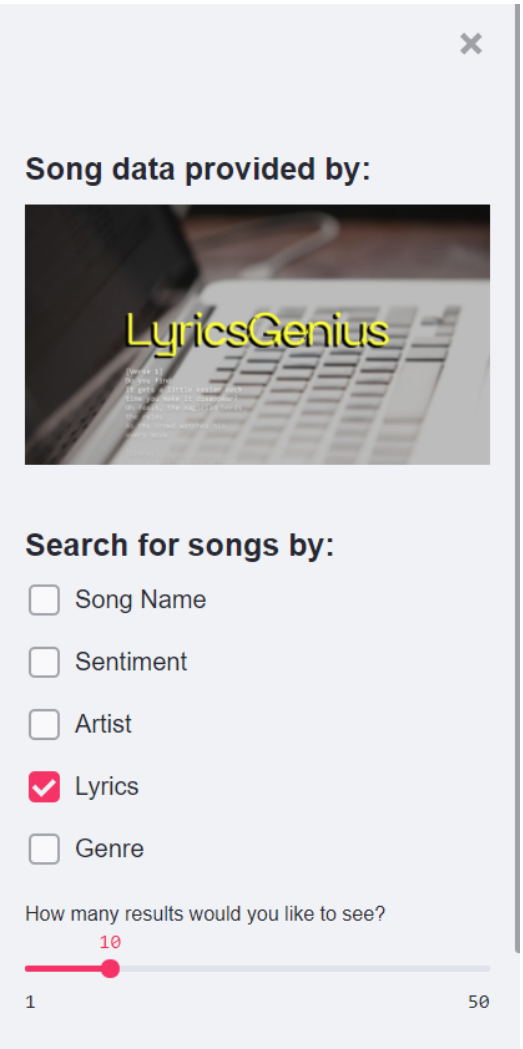


Fig 7: sidebar with options to filter results by

The user can enter any word or phrase relating to the criteria of search selected. This would then be converted to a URL which makes an HTTP GET request to the SOLR server. The SOLR server then returns the result in the form of a JSON object containing a dictionary of metadata about songs. The JSON object received as a response is then parsed and displayed in the UI. 5 sample queries with timings and screenshots are shown as follows -

ii. Write five queries, get their results, and measure the speed of the querying

S.No.	Query	Time (seconds)
1	_artist:”eminem”	0.020000457763671875
2	song_name:”hello”	0.020996809005737305
3	sentiment:"positive"	0.0059740543365478516
4	lyrics:"im in love with the shape of you"	0.007001399993896484
5	lyrics:"id catch a grenade for ya" AND genre:"Pop/Rock"	0.007002115249633789

Table 4: 5 queries and their retrieval time

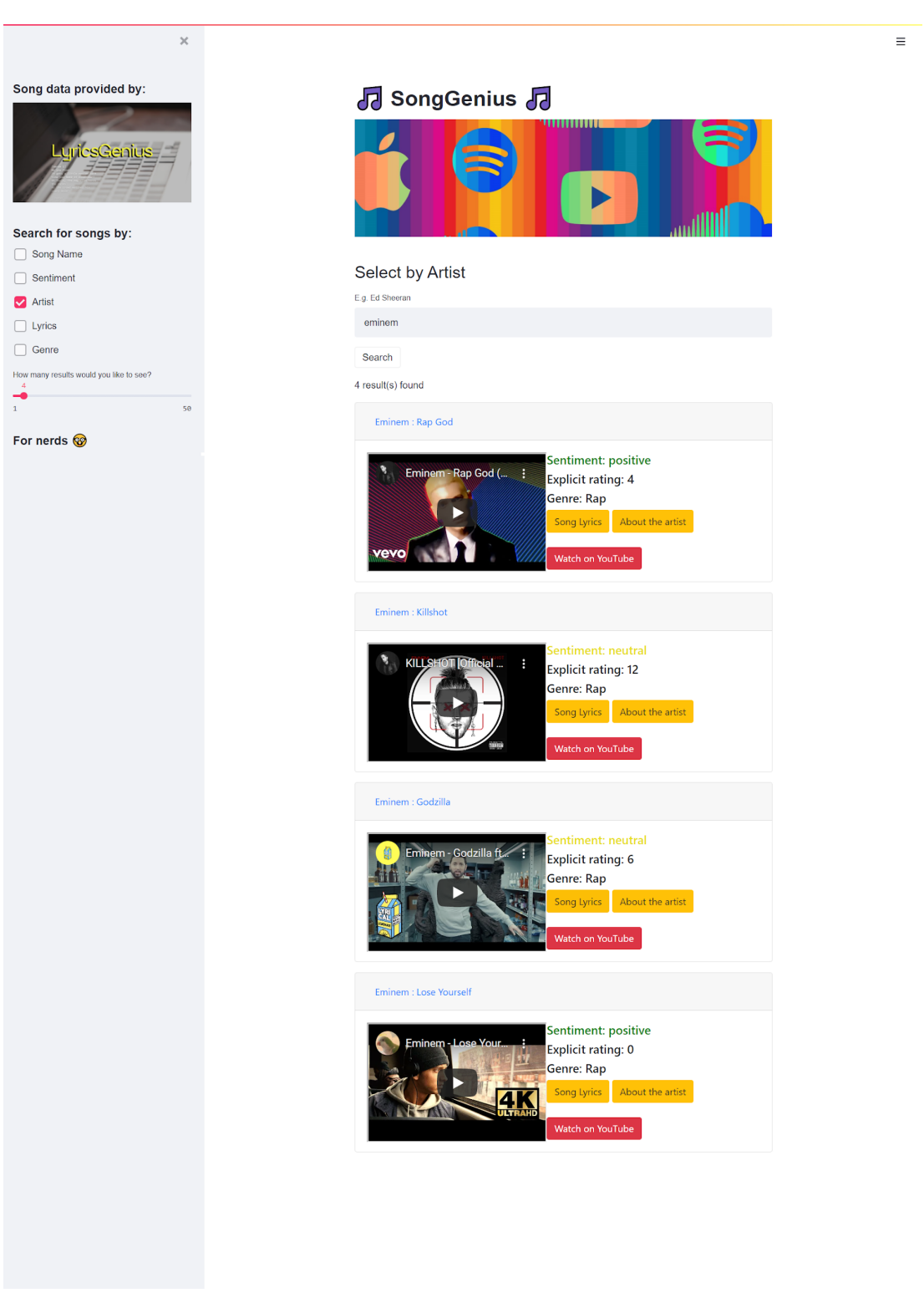


Fig. 8: Search results for query '_artist:"eminem"'

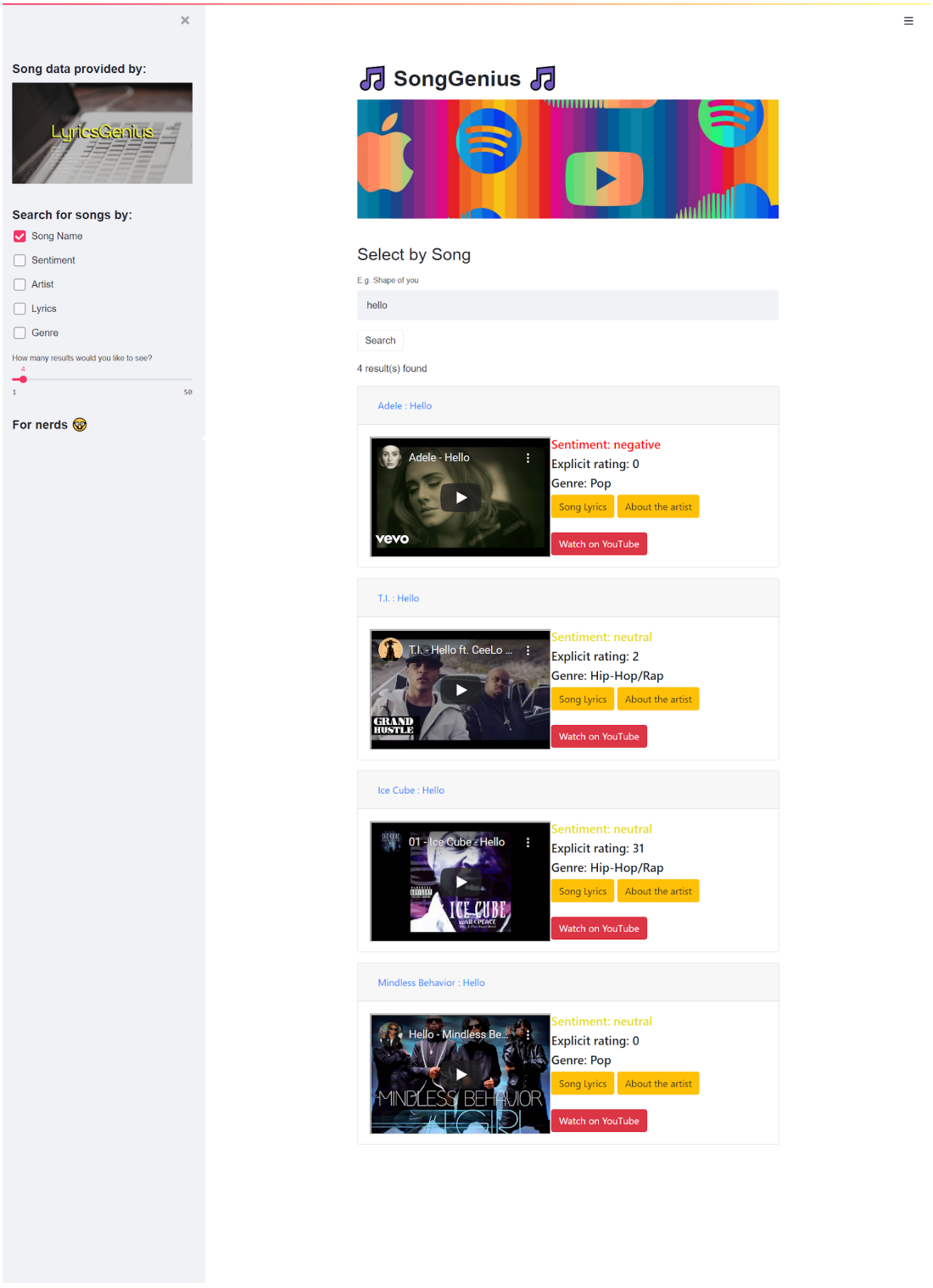


Fig. 9: Search results for query 'song_name:"Happy"'

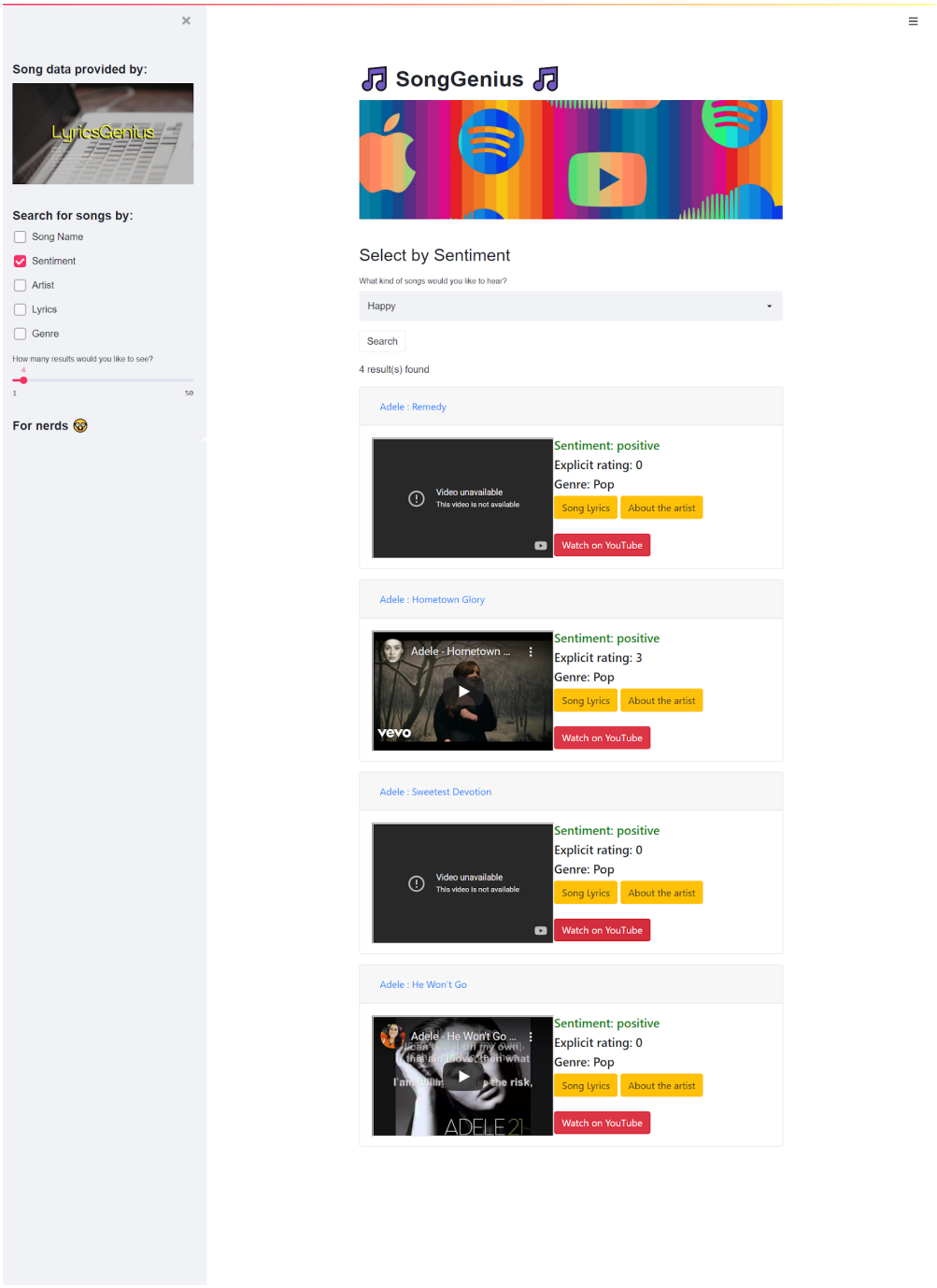


Fig. 10: Search results for query 'sentiment:"positive"'(shown as Happy in the UI but translated as positive before sending an HTTP request to the SOLR backend)
(some videos have been taken down by YouTube for unknown reasons so they are appearing as "unavailable" in the thumbnail)

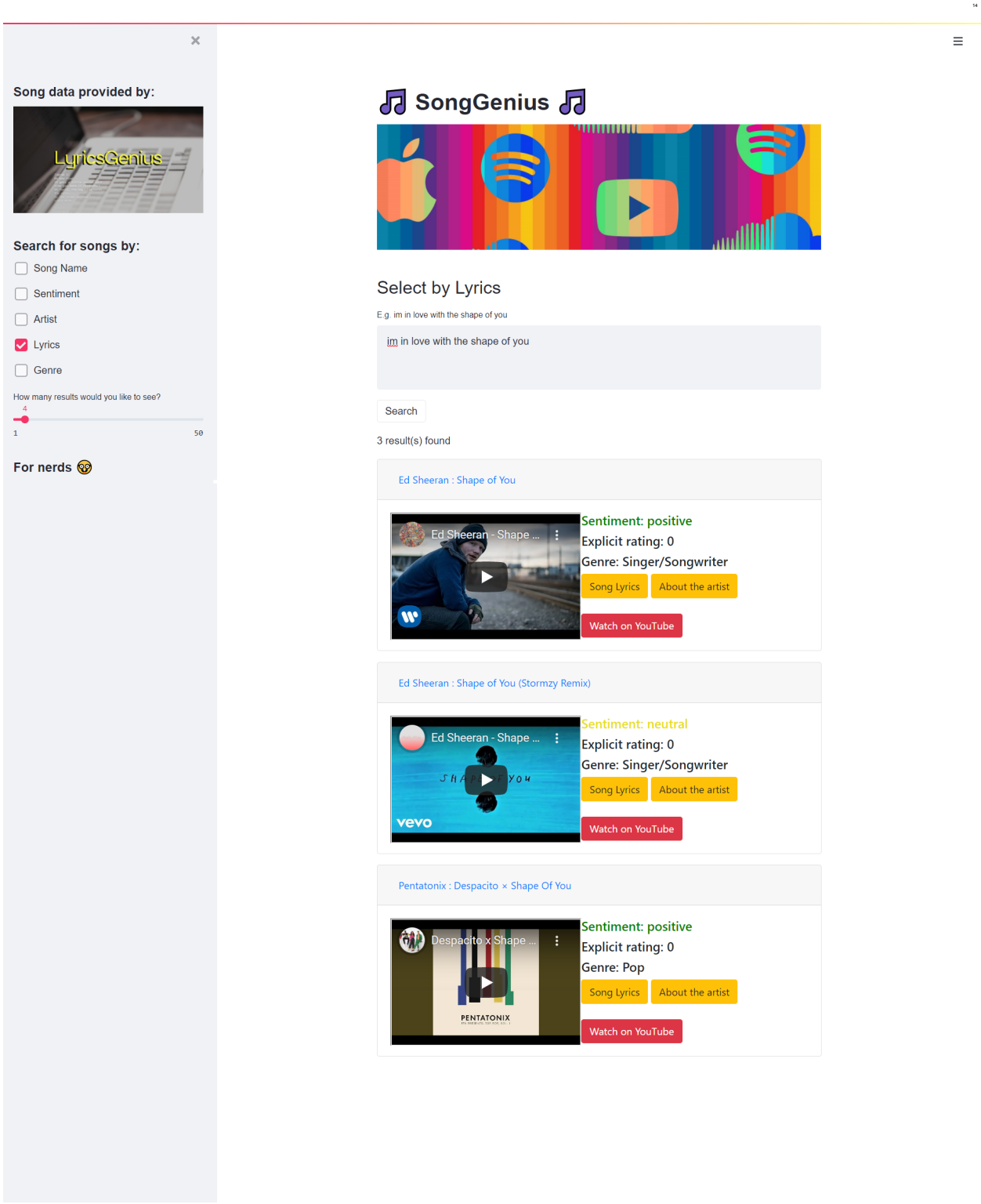


Fig. 11: Search results for query 'lyrics:"im in love with the shape of you"

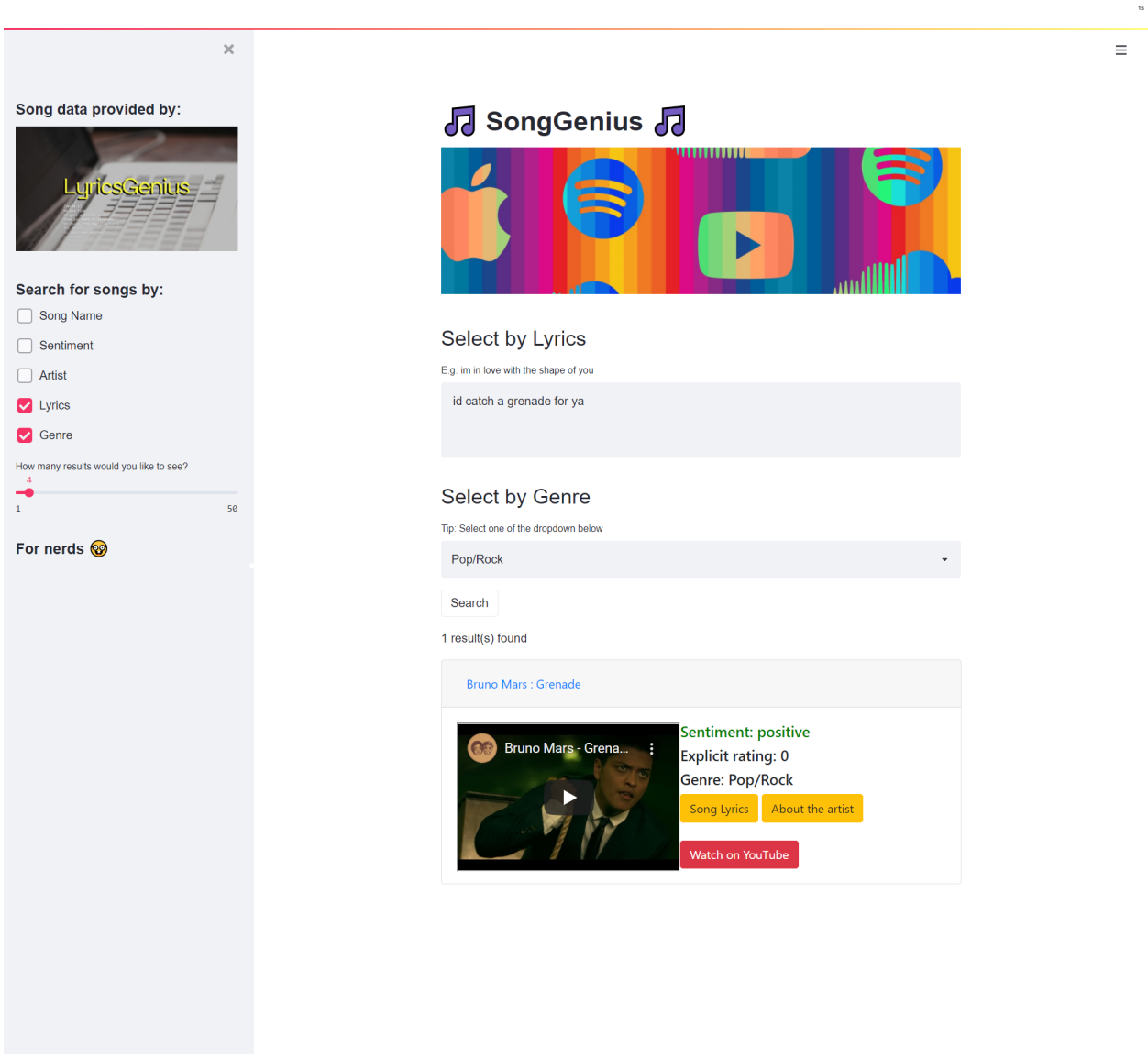


Fig. 12: Search results for query 'lyrics:"id catch a grenade for ya" AND genre:"Pop/Rock"'

c. Question 3

- Explore some innovations for enhancing the indexing and ranking. Explain why they are important to solve specific problems, illustrated with examples.

Multimodal search (video retrieval)

After making a search, each search result that is displayed is a card which consists of not just textual data about the song, but also the embedded YouTube video on the left side of each card. This way the user needn't browse on YouTube on a separate tab/window. The user can simply hear and see the music video within our application. This is done to enhance user experience as well.

This feature is made possible by getting YouTube links for each song using the LyricsGenius API. However the links that were fetched were not meant for embedding videos. Using string manipulation in python, the link to embed youtube videos is obtained from the standard YouTube music video link.

```
final_link = yt_link.replace("watch?v=", "embed/")
```

Fig. 13 String manipulation to enable video embedding

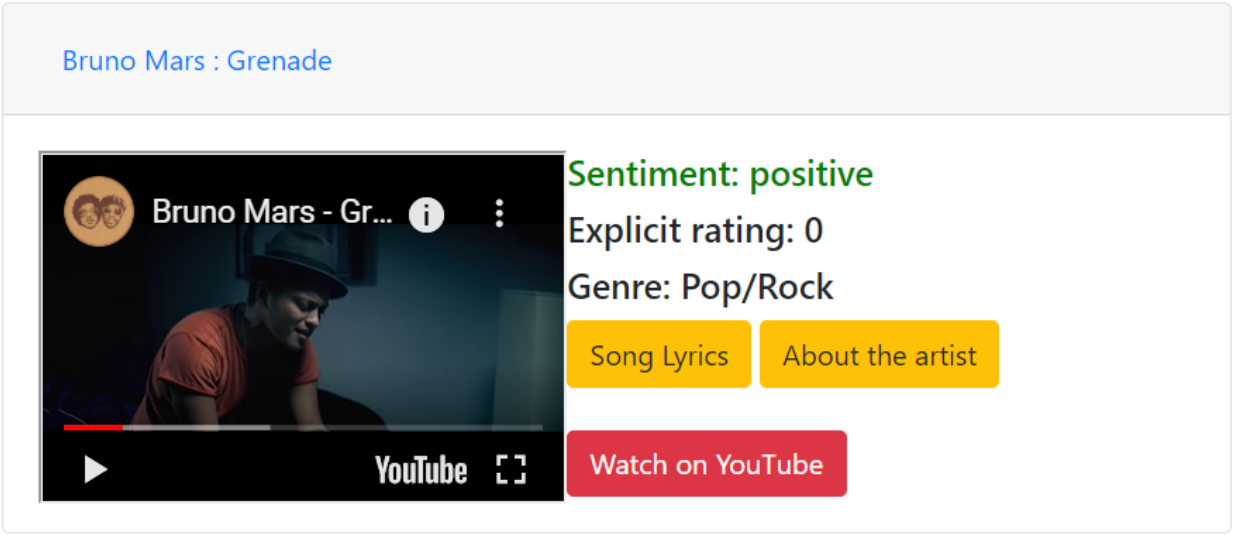


Fig. 14: A search result card from our web app with its YouTube video embed on the left

Multifaceted Search (visualizing information according to different categories)

The application also offers some interesting facts in the form of an analytics dashboard that shows trends of the sentiment of songs across artists, genres and over time.

i. Explicit Rating

Analysing the average explicit rating across genres - it is evident that the Hip Hop and Rap genres have a higher usage of profane language in comparison to more conservative genres such as Inspirational and Soul.

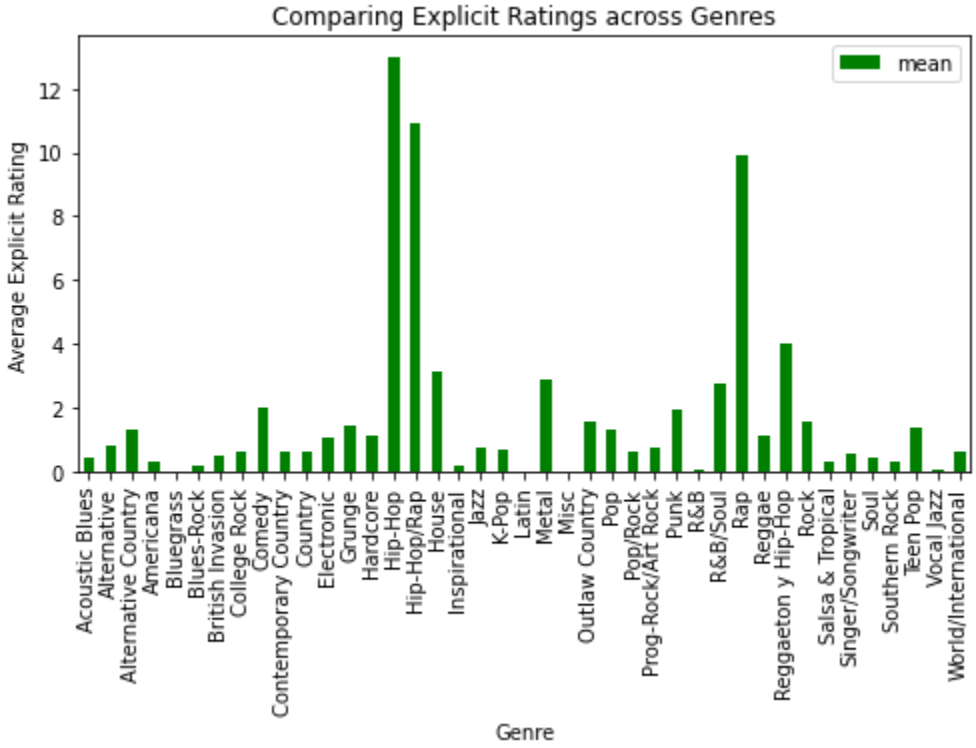


Fig. 15: Explicit Rating across Genres

In analysing profanity over time, it is found that on an average the use of profane has increased significantly since the 1930's. In particular, the music from the 1970's has the highest explicit scoring till date. This may be attributed to the uneven distribution of genres across decades in

the dataset with 10,000 songs. In general, however there are more Hip Hop/Rap songs in the 2010s with higher explicit ratings, however using the mean lowers the score.

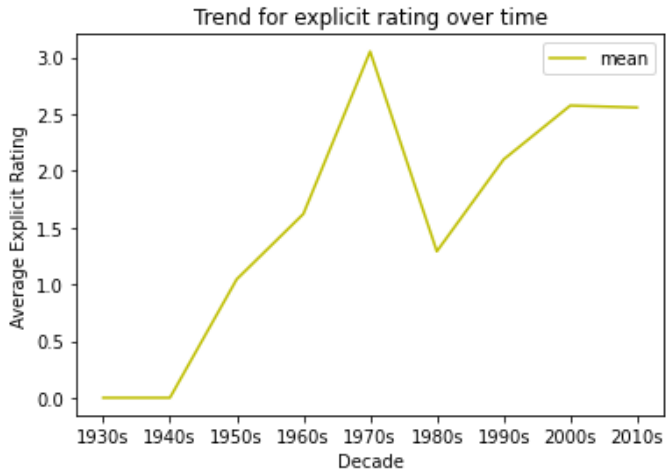


Fig. 16: Trend for explicit rating over time

In the chart below, it is seen that overall, there are more Country and Pop songs than those that are classified as Hip-Hop, Hip-Hop/Rap and Rap combined in the dataset.

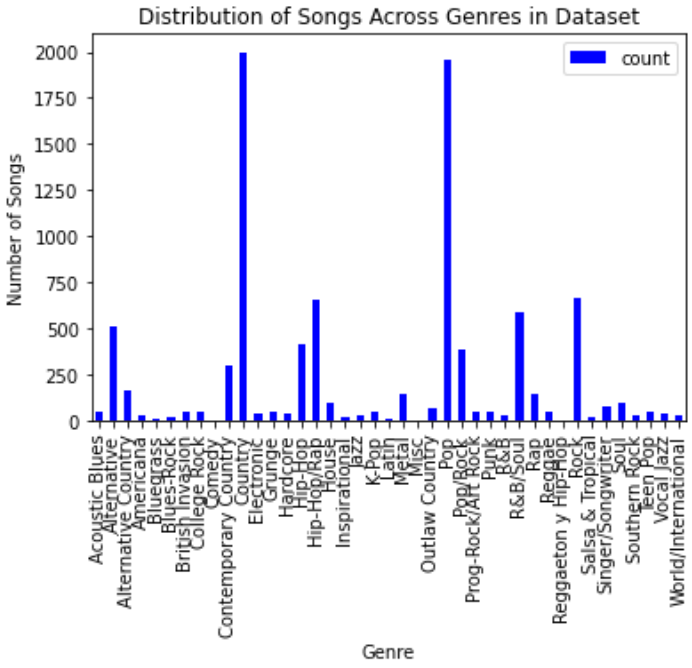


Fig. 17: Distribution of Songs across Genre

The table below shows how for some genres there are songs only from some decades.

genre	decade	
Acoustic Blues	2000s	17
	2010s	29
Alternative	1960s	3
	1970s	14
	1980s	26

Fig. 18: Distribution of Songs across Genre by decade

Ensuring even distribution of songs across years can be considered for future improvement in terms of analysis.

ii. Artist Analysis

Below is a chart that provides the average polarity of an artist's lyrics across their repertoire of songs. Those with negative polarity can be considered to be writing on average more "negatively" themed songs than others.

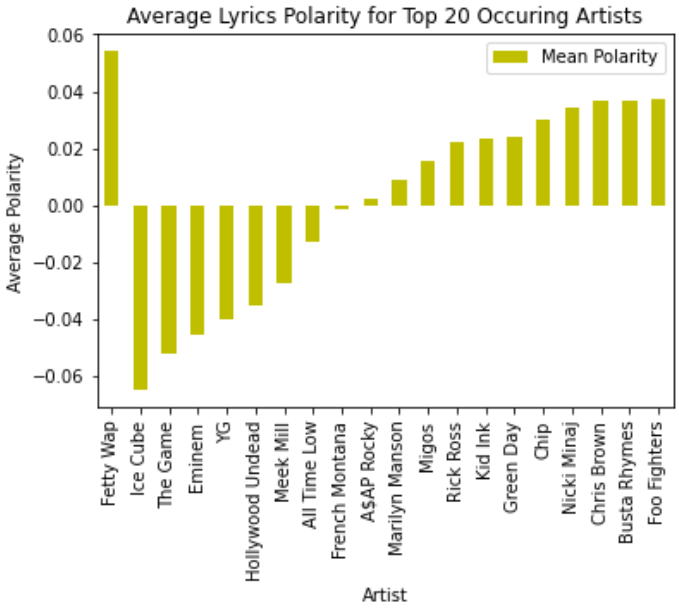


Fig. 19: Average lyrics polarity for top 20 artists

This is an interesting observation, because when analysing the average profanity of an artist, it is also found that the artist with lower polarity scores ('relatively negative' themes) are also using more profane language and are artists who perform genres like Rap and Hip-Hop.

artist	polarity	explicit_rating
Ke$hawn	-0.201528	12.500000
Ice Cube	-0.065259	13.100000
The Game	-0.052246	15.400000
Slipknot	-0.051294	5.224490
PnB Rock	-0.046471	10.906977
...	...	...
Michael Bolton	0.263604	0.058824
Andrea Bocelli	0.268828	0.750000
Alison Krauss & Union Station	0.293050	0.000000
Porter Wagoner	0.324394	0.000000
Priyanka Chopra	0.341343	0.000000

Fig. 20: Relation between average polarity and explicit rating for an Artist

iii. Polarity Analysis

We analysed the trend in polarity of songs over time to find that songs were most positively themed in the 1940s, and have since become relatively less positive over time.

The chart below shows results for analysis of polarity using the Textblob model. As shown in the graph, music has on average become less positive with time.

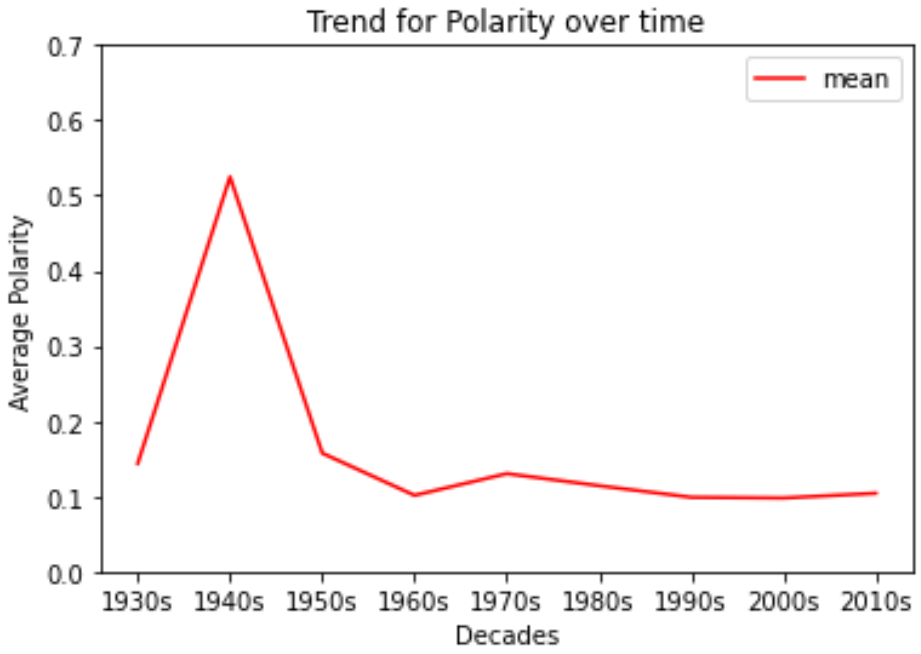


Fig. 21: Trend polarity over time

Using the NLTK Sentiment Intensity Analyzer, we analysed the overall neutrality of songs as well. Over the years, songs have become less positive, and more neutral. This could also be due to the increasing relevance of pop music with generic themes rather than highly emotion driven lyrics.

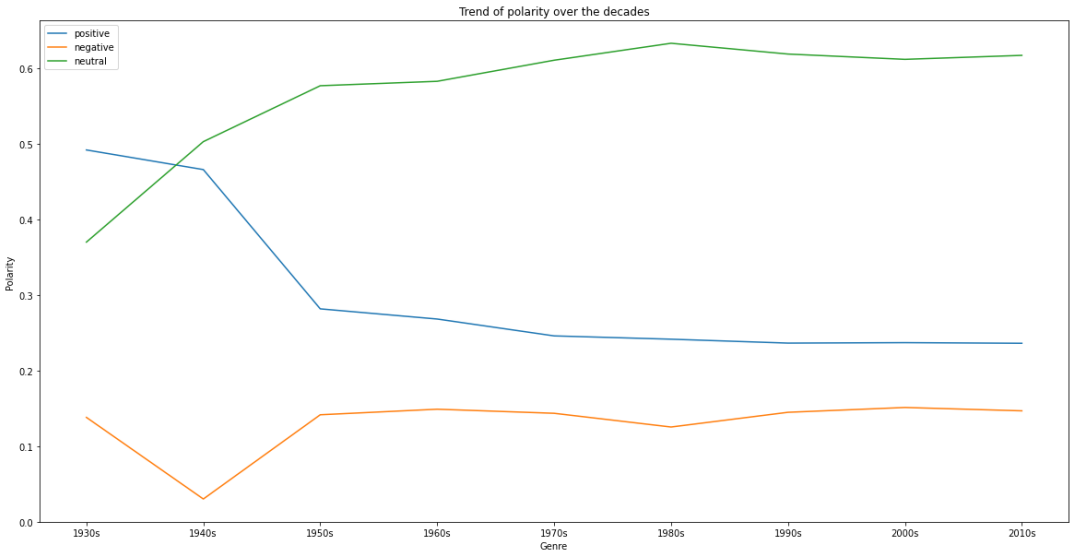


Fig. 22: Trend polarity over decades

The chart below compares the positive/negative/neutral valence of songs across genres. This analysis is done using the NLTK VADER Sentiment Intensity Analyser that offers a 3 part score that indicates the percentage of positive/negative/neutral sentiment. The scores range from 0 to 1 as they indicate percentages.

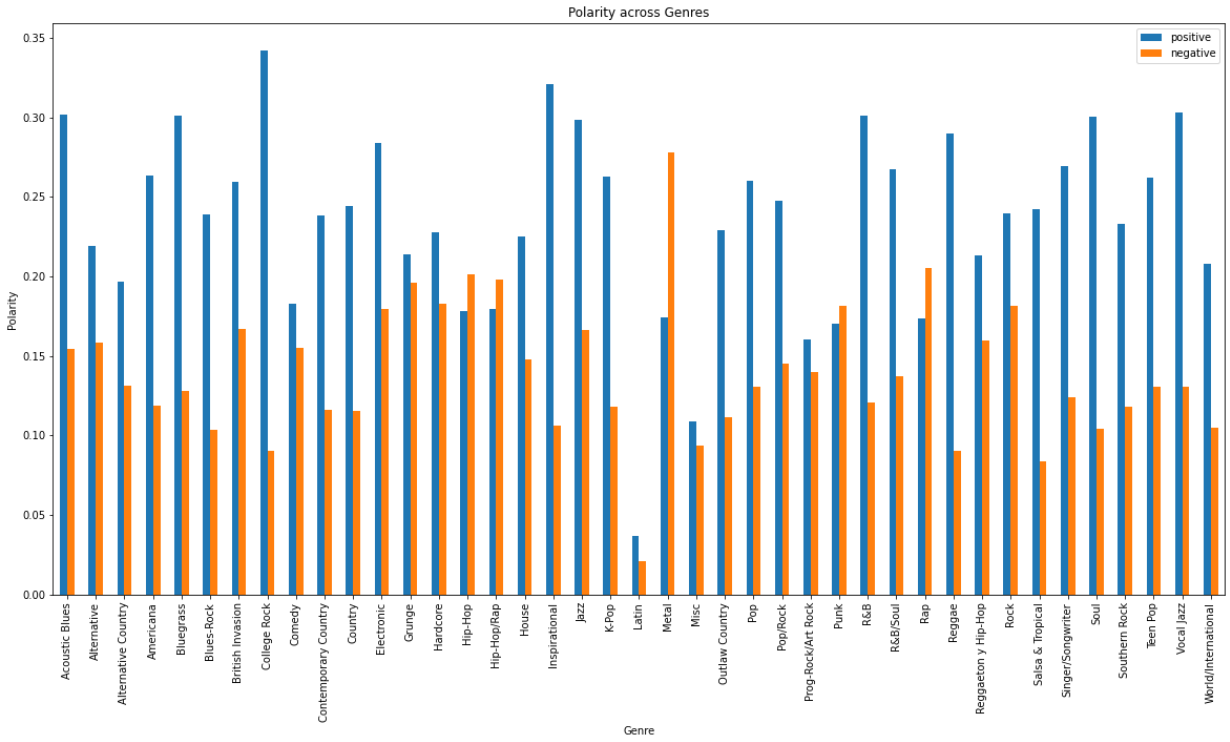


Fig. 23: Polarity using NLTK VADER Analyzer for non-neutral songs.

As is evident from the chart, some genres are much more positive than their are negatively themed like Inspirational music and R&B. Others such as Rap, Hip-Hop and Metal are more negative than they have positive emotions. This is likely due to the general themes discussed in different genres like Inspirational and R&B music are centered around love, devotion, motivation and other positive associations unlike Rap/Hip-Hop which oftentimes mention violence, drug-abuse and other negative associations.

iv. Polarity vs. Subjectivity Analysis

Music has always been known to be subjective to the listeners perception as the same song may appear more happy to one listener than the other based on how they perceive the lyrics.

Our analysis of the 10,000 songs on the dataset for subjectivity proves the same. Across decades from the 1930s till date - songs have remained about the same level of subjective

(approx 0.5 score on a scale from 0 to 1). However, the polarity of songs has changed over time. In general, the songs have remained the same level of subjective, but have become less positive in valence over time.

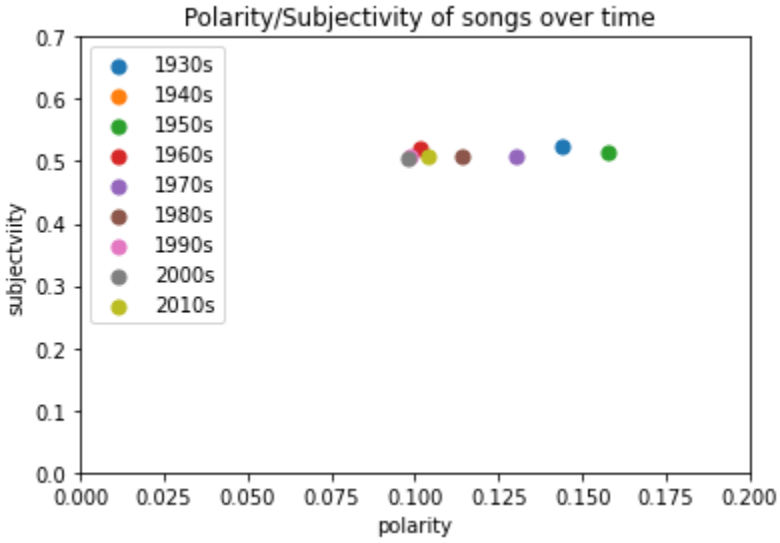
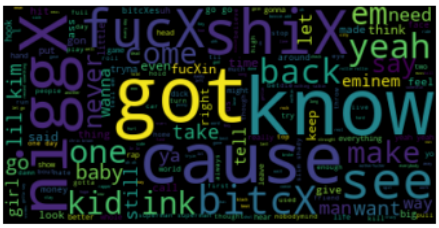
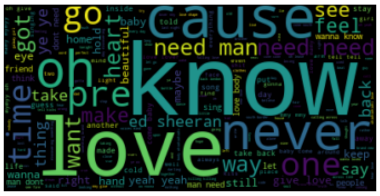
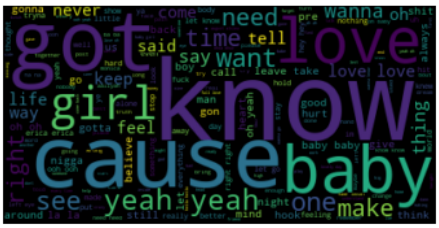


Fig. 24: Polarity of songs over time

v. Word Clouds

In section (iii) about polarity analysis, we made an observation that some genres are more positive than others and hypothesised that this may be because of the general language usage and themes addressed in these genres. To confirm our hypothesis, we performed a word cloud analysis on genres and some selected results in relation to the hypothesis are shown below:

Genre	Valence	Word Cloud**	Popular Themes
Hip-Hop/Rap	Negative		Money, Race, Abuse, Drugs, Women, Profanity, Sex

Hip-Hop	Negative		Money, Race, Women, Profanity
Rap	Negative		Money, Race, Profanity, Loneliness, Fame, Women
Country	Positive		Nostalgia, Family, Love, Relationships
Soul	Positive		Love, Healing, Faith
R&B	Positive		Love, Relationships, Sex

*** Profane language has been marked "X" for the word clouds.*

The multifaceted results have also been displayed on the Dashboard Analytics portion of the web application, shown by figure 25 below:

Dashboard Analytics

This portion does shows the evolution of trends of the songs over time

Fig. 25: Dashboard Analytics

Moreover, the positive/negative sentiment ratio can be viewed by year on the web app. A snippet is shown below:

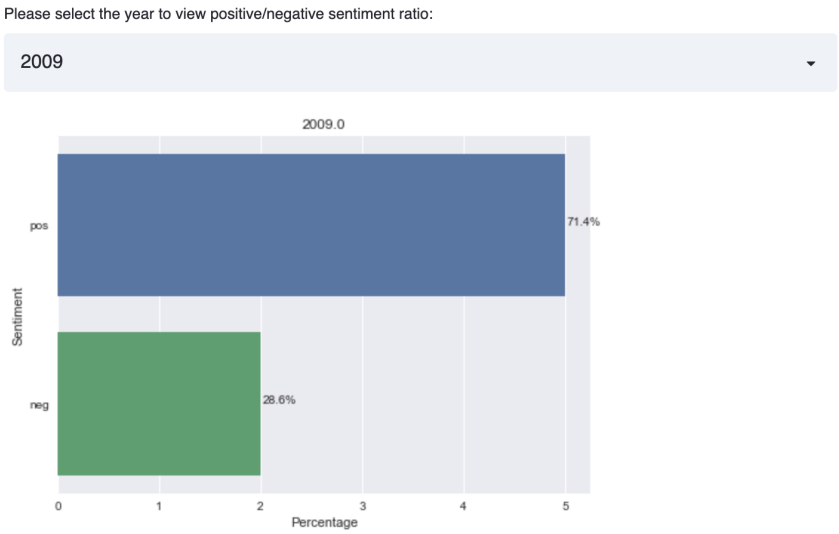


Fig. 26: Positive/Negative Sentiment ratio by year

4. Classification

a. Details

A challenge with analyzing the sentiment of lyrics especially when using a manually labelled training data set is that the perception of the lyrics is entirely up to the labeller. In particular:

1. The same song could be more positive to some people, but relatively neutral to others.
2. In some cases the sentiment of the song varies in different segments of the lyrics. For example, the first verse could be more negative, while the chorus and bridge of the song are more neutral - this results in the overall sentiment score being skewed.

This application aims to classify the lyrics of a song on the basis of its polarity - neutral, positive or negative. It will also offer to the user, an explicit rating for each song on a scale from 0-100.

b. Question 4

Discuss whether you had to pre-process the data and why?

The lyrics were processed using the *NLTK* library to remove stopwords and recurring, redundant terms etc. and tokenization of the lyrics into words. After some trials, we identified that without removing the standard stop words offered by the NLTK library and removing identifier words ([Verse],[Chorus] etc.) redundant to the sentiment of the song - the overall neutral valence of a song kept increasing. To offer a fair judgement of the sentiments, these words were removed before processing.

Before removing stop words, as seen from the graph below, the most common words were ‘i’, ‘you’, ‘the’, ‘and’, etc. These words do not add any significant value while performing sentiment analysis on the songs, hence we decided to remove them.

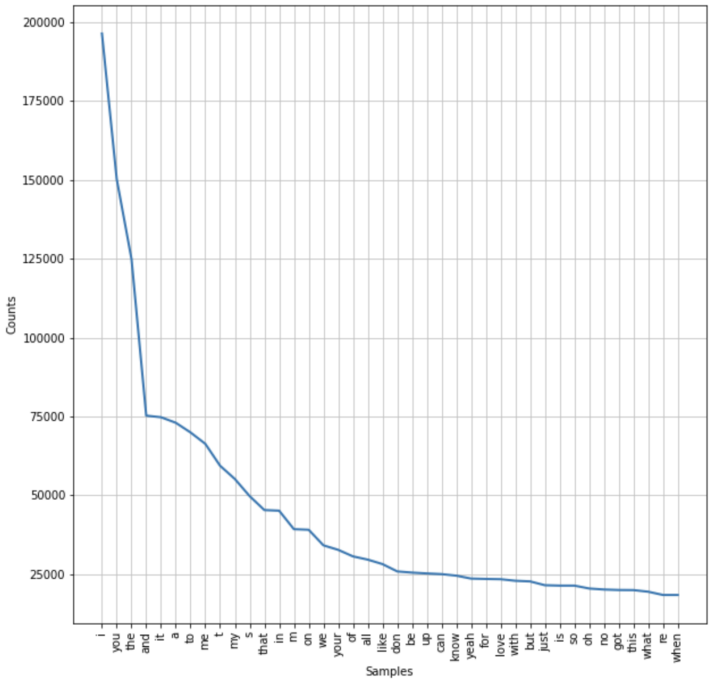


Fig 27 : Most frequently occurring words

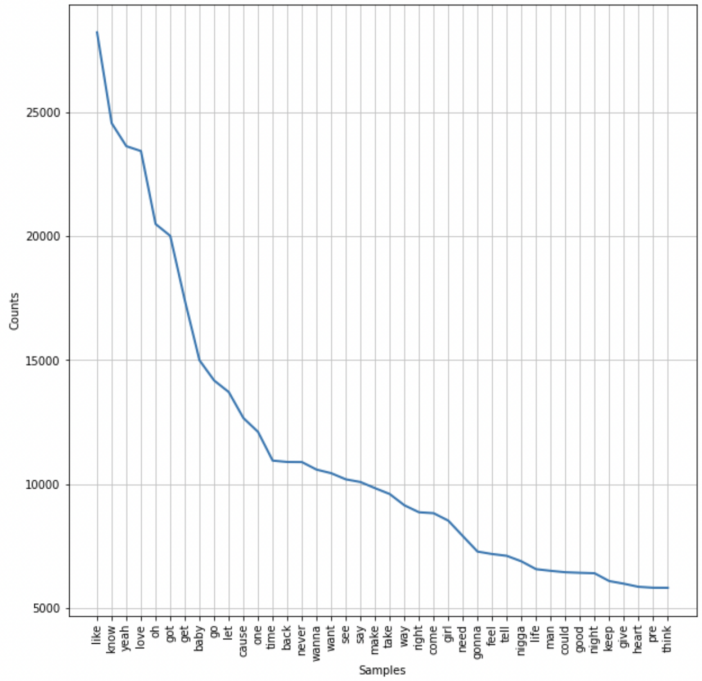


Fig 28 : Most frequently occurring words after removal of redundant terms

The cleaned lyrics were then tokenized to words to further analyse the themes of songs for analysis of trends across artists and time.

We chose not to stem and lemmatize lyrics as part of preprocessing because sentiments of lyrics are highly dependent on the polarity of the words. If the word ‘better’ was lemmatized to ‘good’ the polarity of the word also changes. With enough words that have been lemmatized this way, the overall score of the song could be compromised.

Evaluation Dataset

From the total of 10,000 songs that were crawled from the API - 1,000 songs were labelled with 250 positive, 250 negative and 500 neutral songs to represent 10% of the dataset being worked with. These 1,000 songs were used as a training set for our models and to validate its classification of the sentiments.

Classification Approach

We started off by building custom neural networks using tensorflow backend and apis form keras as well. However in spite of using word embeddings, Recurrent Neural Networks, Bi-directional LSTM and pooling layers, we were not able to get the required accuracy and the model training was not following the trend we expected it to follow.

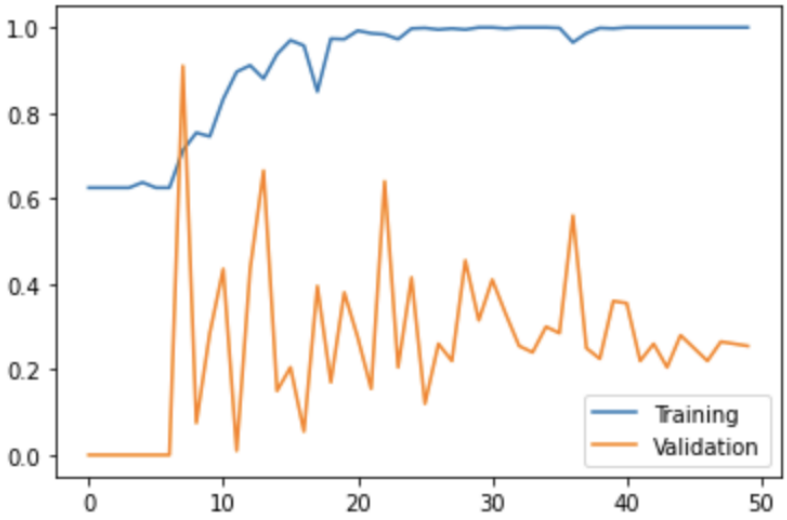


Fig 29 : Training and Validation Accuracy

Above graph shows the trend of the training and validation accuracy. We can observe that the validation accuracy does not continuously increase over time and does not follow a uniform pattern as expected.

Another issue we faced was that some songs were very long compared to the rest, this resulted in padding the shorter songs and could have been a result why the accuracy wasn't as high as expected. We used a vocabulary and converted the text to words for creating word embeddings and passing to the neural network.

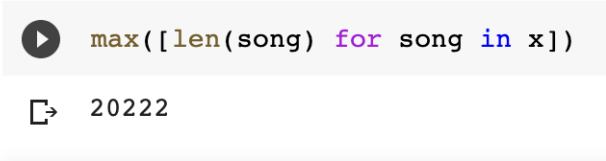


Fig 30 : Length of longest song after stop word removal

However, there were challenges in entirely relying on the custom model as with just 10,000 songs as a training set, the results were not sufficiently accurate. Since the training set was labelled using another python library called Textblob, the sentiment of a song could change from paragraph to paragraph and the human perception of lyrics is fairly subjective. Combined with the small size of the training set, this contributed to the performance of the models. Models were not able to learn the trends between the sentiment of the songs and the words of the song.

To overcome this issue, a hybrid classification approach was taken with three different classifiers - NLTK VADER *Sentiment Intensity Analyzer* (python library), *Textblob* (python library) and *spaCy* (python library).

First, the *Sentiment Intensity Analyzer* is used to classify songs as neutral or non-neutral in nature. Of all the non-neutral songs - *Textblob* is used to classify songs as having positive/negative valence. The *Spacy* model is a python library that was used to build the custom model trained on the labelled dataset and is used to classify songs as either positive, negative or neutral in nature.

Finally, an ensemble classification is performed to combine the outcomes of the three classifiers to produce a final result.

i. Using Sentiment Intensity Analyzer vs. Textblob

The VADER *Sentiment Intensity Analyzer* outputs a 4 component score to show the polarity of a song : positive, negative, neutral and compound. *Textblob* outputs the polarity and subjectivity of a song. The polarity ranges from -1 (most negative) to 1 (most positive) and the subjectivity ranges from 0 (objective) to 1 (subjective).

The *Sentiment Intensity Analyzer* was able to clearly score songs with “neutral” valence while with *Textblob* we would have had to decide the window of polarity scores that are considered as neutral sentiment.

We did not use the compound score for the Sentiment Intensity Analyzer as it is a combination of the prior 3 scores and there several songs being classified as neutral when using the sentiment Intensity Analyzer as the neutral scoring was fairly high - resulting in the compound score being skewed.

For example: “Today I am sad.” In this sentence, ¾ of the sentence comprises “neutral” words, and the only strong sentiment is the word ‘sad’, overall the sentence would have a lower negative score than it ideally should. To overcome this problem, we refined the pre-processing of lyrics to remove stopwords, and redundant words which are either neutral in nature or are not part of the VADER sentiment lexicon as they will not have a score.

Textblob however, only scores positive/negative polarity and accounts for the subjectivity of the song as well. In terms of more accurate non-neutral classification:

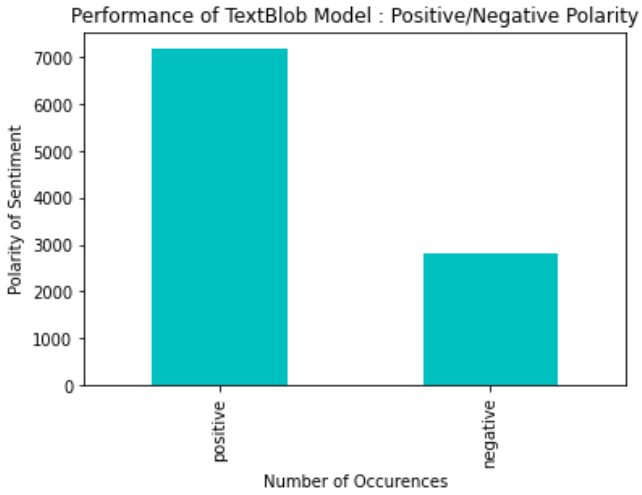


Fig 31 : Textblob Model - Positive/Negative Classification

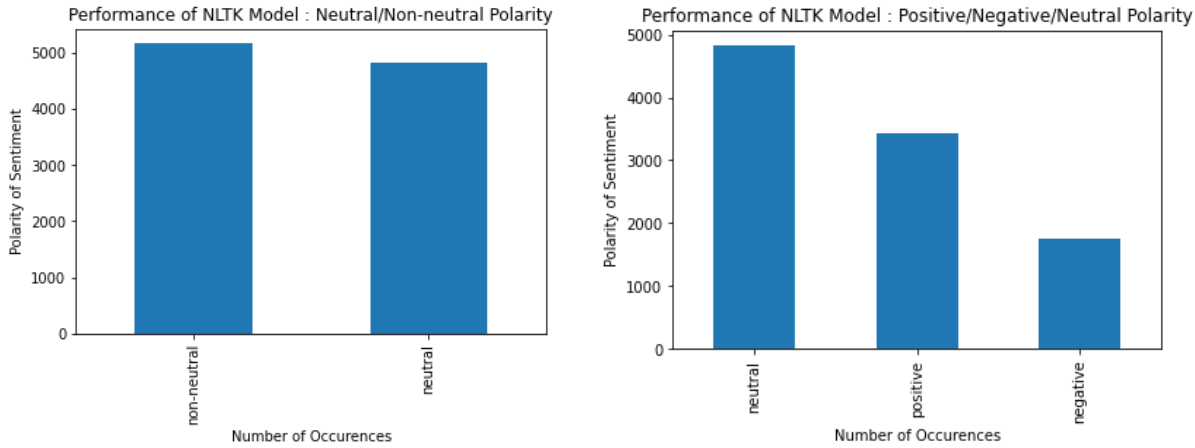


Fig 32 : NLTK VADER Sentiment Intensity Analyzer Classification

After experimenting with both approaches and comparing the outcomes with the initially labelled evaluation set, we identified that for our application Sentiment Intensity Analyzer would be better as a neutral/non-neutral classifier and that Textblob was better at scoring positive/negative valence.

ii. spaCy

We used a python library named spacy to build the sentiment analysis model, which we trained on all the songs labelled as non-neutral. This means that every song in the training set had the label positive or negative based on the sentiment of the lyrics of that song.

We built a training pipeline using spacy's implementation of pipelines for Natural Language processing models and performed training using that pipeline.
We used 20% of the training data as the validation set and the remaining as the training data.
We used the built in architecture best suited for sentiment analysis, called 'simple_cnn'. We even used a dropout ratio of 0.2.

Provide evaluation metrics such as precision, recall, and F-measure and discuss results

The confusion matrix, accuracy and F-score have been given below.
The precision was around 70% and the recall was around 98% while the accuracy was around 78%.

```
Confusion Matrix:
 [[ 81  43  37]
 [ 23 404  65]
 [  6  45 296]]
accuracy: 0.781
F1 Score: 0.7378516872417046
```

These results show that our model was more on the liberal side and not the conservative side. Our model would predict more songs as positive than compared to the

Discuss performance metrics, e.g., records classified per second, and scalability of the system.

```
100%|██████████| 5172/5172 [00:34<00:00, 148.59it/s]
```

c. Question 5

Innovations for enhancing classification [Ensemble Classification]

We used a voting ensemble to get the best results from the sentiment analysis classification models that we had built. A voting ensemble (or a “majority voting ensemble“) is an ensemble machine learning model that combines the predictions from multiple other models.
It is a technique used to improve model performance, by combining the predictions from multiple models, ideally to achieve better performance than any single model used in the ensemble. This technique is mainly used in classification and regression tasks and hence fit into our classification pipeline well.

```
final_preds = []
for i in tqdm(range(len(x_test))):
  song = x_test[i]
  # spacy1 = test_model(song)
  textblob = round((TextBlob(song).sentiment.polarity + 1)/2)
  scores = sid.polarity_scores(song)
  if scores['neg'] > scores['pos']:
    nltk_score = 0
  else:
    nltk_score = 1
  if textblob + spacy_final[i] + nltk_score == 3:
    final_preds.append("positive")
  elif textblob + spacy_final[i] + nltk_score == 2:
    final_preds.append("positive")
  else:
    final_preds.append("negative")
```

```
100%|██████████| 5172/5172 [00:34<00:00, 148.59it/s]
```

Fig 33 : Code for voting ensemble

We used results from NLTK VADER Sentiment Intensity Analyzer function, Textblob function and custom sentiment analysis model that was built using spacy. We had given each of the models equal weights as that is seen in Fig 33.

Enhanced classification (add a sentiment analysis subtask) : Explicit Rating

Streaming services like Spotify and Apple have filters for explicit music where it is automatically removed from all playlists. Song Genius allows the user to decide at their own discretion about listening to a song by offering a feature that attaches an explicit rating on the lyrics on a scale of 0-100.

This feature was built, getting inspiration from a python library called profanity-filter that compares a list of profane languages against a corpus. The corpus in this application is the processed lyrics with NLTK stopwords removed and the words of the lyrics are tokenized. After identifying the explicit language used in a song, a score is computed as a ratio of explicit words to total words in the song based on a formula we tried and tested to work the best.

```
print("Explicit Rating: "+ str(exp_tot/tot * 400 ) if  (exp_tot/tot * 400 < 100) else
                                str(100.00) + '%')
```

We observed that the songs with a high amount of profane content produce a higher explicit score. This could be used in the industry to enable a child mode and disable songs with the higher explicit score to be streamed.