Spotify Track Statistics and Lyrics Analysis
An Information Processing and Retrieval Study

André Santos, Edgar Torre, Miguel Amorim
up201907879@fe.up.pt, up201906573@fe.up.pt, up201907756@edu.fe.up.pt
Faculty of Engineering of the University of Porto

ABSTRACT

In the current days, we come across big amounts of data and so an increasing concern to index and search efficiently appears. In this paper, an approach to creating an information retrieval system for a music lyrics database is expounded. The proposed solution includes a description of the dataset preparation, enrichment, refinement, and exploration process, as well as the detailed exposition of the information retrieval stage, from the indexation of the documents to the evaluation of the resulting system. An overview of the used tools is also included. The final goal is to create a more complex search system, allowing users to efficiently find the exact music they really wanted to hear.

Keywords: Information Processing and Retrieval, Music, Lyrics, dataset, statistics, music data

1 INTRODUCTION

Music is essential in most people's daily lives. It is one of the oldest and most explored arts, and consequently exists in abundance and diversity. This on the one hand is positive, since there is always new music to discover, but on the other hand is negative, due to the difficulty in organizing, characterizing and grouping similar music. That said, this project aims to facilitate this process by using information systems. The main goal is to develop an information search system, which means we will work on multiple aspects, such as data collection, data preparation, information querying, information retrieval and retrieval evaluation.

There were three major constraints we had to take into account when choosing the initial dataset. We needed a dataset with:

- Large amount of data (at least some thousands of rows)
- Varied data types per row (at least ten columns, half of them with different types)
- Text based columns (at least one, two if of minor size)

Considering all of these conditions, we decided on a music dataset, since there are countless different musics, with multiple characteristics (name, artist, duration, type, date, album, ...) and some of which are lengthy text based (i.e. lyrics).

The database structure is then illustrated and the search tasks to perform in it are specified. Sub- sequently, the information retrieval stage is carefully discussed, clarifying each action taken towards achieving the designated goal. Lastly, the drawn conclusions are summarized and relevant future work possibilities are mentioned.

2 DATASET

We started looking for a dataset with a lot of different data, like release year, artist, album, popularity, and most importantly, with lyrics. There were some databsets which nearly met our needs, but lacked in some details. Some were way too focused on specific music genres, others were too small [1] and some were non-english monolingual [2], which meant a poor match for linking with the Spotify API (more about this later in the report). Later, in a deeper search, we found a good enough dataset.

2.1 Chosen Dataset

We found a dataset with more than 4 million rows with the lyrics of each one and some more data about each music. Its initial columns were: title, tag, artist, year, views, features, lyrics, id. Since it had a huge amount of data we could work with and it met our goals, it was a good fit. [3]

However, this dataset had some problems, including:

- Useless columns, such as id, which we removed.
- Inconsistent columns, such as features and views, which we removed some and revamped others (more about this later in the report)
- Excessive size of 9.21 GBs, of which we randomly chose 80000 rows.
- Lacked different data types after removing the non-useful columns.

With that problem in mind, the solution we found was finding some API that could give us the data we need. We found that the Web Spotify API [4] could add new information which complemented the one we already had, like popularity, albums, and duration of music.

In conclusion, we selected a dataset of music data collected by someone else and added to it a lot of information with the Spotify API.

2.2 Dataset Content

The chosen dataset contains 80000 musics with attributes as title, tag, artist, release year, lyrics. While from the API, for each music we got the data: track number, popularity, explicit, duration_ms, album's release date, album's number of tracks, album's name.

2.3 Data Quality and Source

Kaggle is a well-known platform for data searches, the post author states that all the data comes from genius.com, a well-known music platform with 5.5 million followers on social media. With this in mind, we can infer that this data is reliable. About Spotify, since it's an app that is used all around the world and already receives some awards, we can also state its reliability.

About the quality of the data, we can say that met our criteria, the data we needed was there and with very few unexpected values.

3 PIPELINE

Our Data Preparation Pipeline is done, mostly, on Python scripts, where pandas and matplotlib libraries were fundamental to cleaning and manipulating the data. With that, we manage to create a structured and clean SQL database system.

3.1 Pipeline Scheme

After all the analysis of the problem, the final version of the pipeline process was the following:

As we can see on figure 1, the initial data come from the Kaggle dataset "ds2.csv" and we refined this data, eliminating the duplicated ones and eliminating the columns that we did not need (year, features, id, views). Furthermore, we picked a smaller population of data from the bigger one we had (4 million to 80 thousand) and added more data that came from the API (track number, popularity, explicit, duration_ms, album's release date, album's number of tracks, album's name).

The join was made based on the title of the music and the name of the artist. After that, the process consists of plotting various graphs and turning the final version of the dataset on a structured SQL database system.

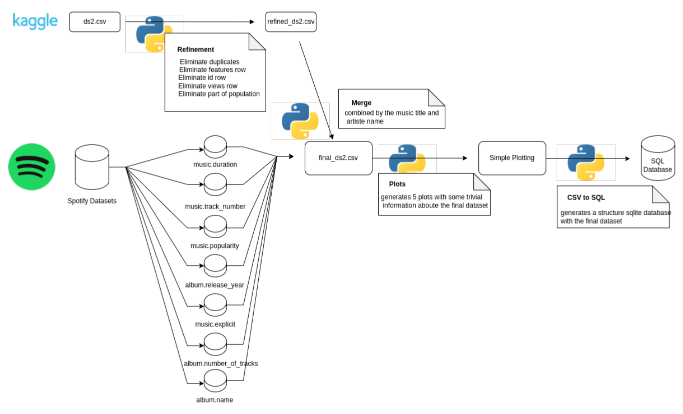


Figure 1: Pipeline Scheme.

3.2 Domain Conceptual Model

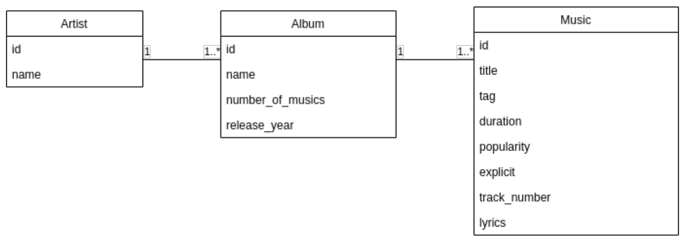


Figure 2: Domain Conceptual Model.

The Database consists of three tables. The table Artist stores the name of every artist. The table Album stores the general information about the albums. And a Music table, where all the information about the music is stored, that's the main class of our database. With this in mind, and using the final version of the dataset and another Python script. we created our SQL database. The Domain Conceptual Model is shown in figure 2.

3.3 Data characterization

We made a python script to see general information and plot them, to better understand the final version of this dataset.

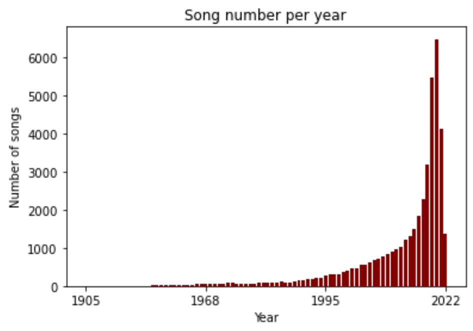


Figure 3: Musics per Year.

To start, as can be seen in figure 3, although a big part of the music of our dataset is recent, we have music from each year from before 1968, which means that the information about most recent years is much richer but we can see the evolution from years ago.

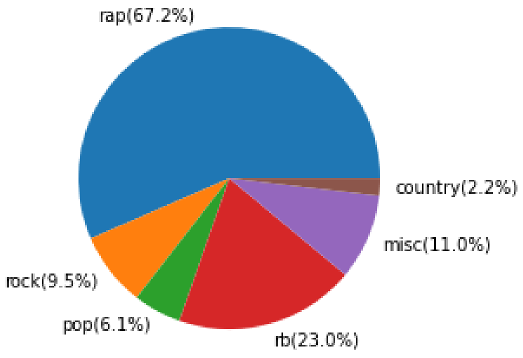


Figure 4: Musics Grouped by Genre.

The second plot, figure 4, shows us how many genres of music we have on our dataset. We can see the variety of genres, which makes our database richer and more reliable to search.

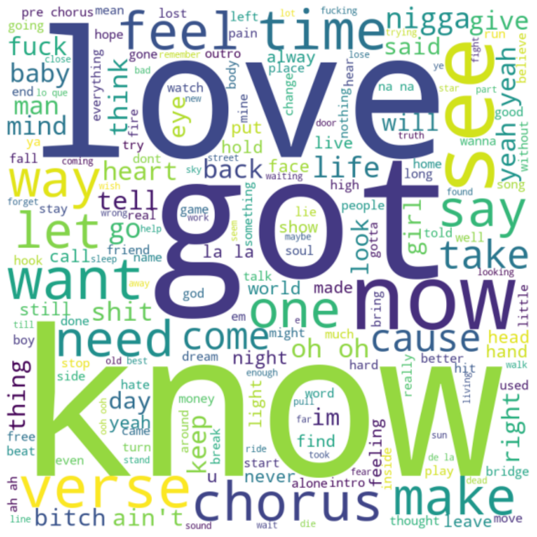


Figure 5: Lyrics Word Cloud.

We also found it interesting to search for the most popular words overall in the lyrics we had. For that, we made a word cloud (a type of graph to measure the frequency of words) present on figure 5. We can easily see that there are words that are used much more frequently than others, like ”love” or ”know”. It's no big surprise since we are used to hear these kind of expressions in many popular songs.

Furthermore, we thought we could see the duration range of the music (figure 6). We concluded that most songs have mostly 2 minutes in duration. However, we

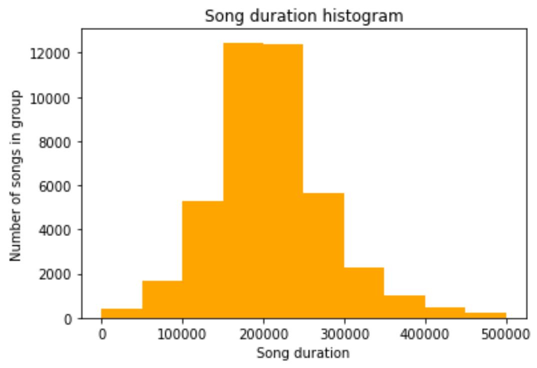


Figure 6: Music Duration Histogram.

have music that is in the 5 minutes range. We also noticed a fair number of outliers, surpassing the 10 minute mark, but it is also easily explained, since there are many extended or concert versions that tend to be bigger.

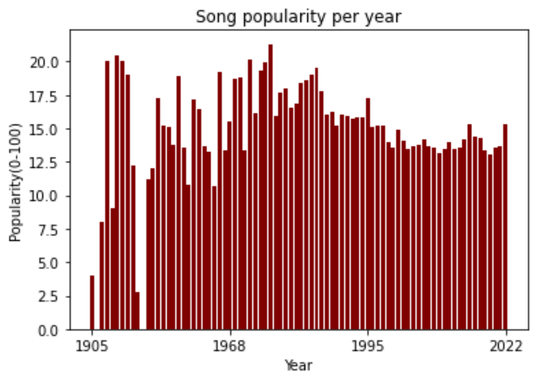


Figure 7: Music Duration Histogram.

To finish, as we can see on figure 7, we also studied the average popularity of the music every year. We were expecting to notice a really big difference between modern music and the last century songs, but what the graph shows is that it actually doesn't change much over the years. This is due to the popularity metric not being linear with number of Spotify streams during a certain amount of time, but a 0-100 scale solely based on the popularity of all the artist's tracks. Due to the random nature of our row selection, many of the final tracks weren't the artists' most popular ones, therefore justifying why the average popularity is never bigger than 20.

4 Search Tasks

With the work we have now, we can already search for a lot of information. In the future, we want to explore this, even more, the database with the objective of creating the best retrieval tool for this specific database. We are really interested to explore, so we can also answer the following questions:

- What are the most used words by genre?
- Did the music duration change with time?
- Do different genres have different percentage of explicit tracks?
- Which genres have the most disparity in popularity between explicit and non explicit tracks?
- Does the tracking number of music affect its popularity?

5 Collection + Indexing

The second part of this project was building the search system, which implied indexing the required fields and setting the collection. Since our system only has musics as records, we do not need more than the one collection.

5.1 Document definition

There are many available tools to perform Information Retrieval operations. The two most popular are Solr and Elasticsearch. Even though they are built around the same engine, there are substantial differences between both. In the end, the selected tool was Solr. [5]
The format of the file to be inserted was JSON, for both schema and the collection, since the conversion was already processed in the previous milestone as a preventive measure. This also grants consistency between the 2 files. However, Solr was actually pretty permissive in file types, so a 'CSV' format would perfectly fit as well. There was only need for the usage of one core/collection, since even though our data came from different sources, it was merged into a consistent file, that groups all of the information. Efficency-wise it was the better option. Besides, it is simpler. The document indexation itself was performed with the Solr default behaviour.
In order to turn on the solr interface we use docker. That allows the image to be set so that the whole system can be initiated with ease.

5.2 Indexing process

The collection is uploaded to the Solr core through a curl command, however, in order to index the fields as desired the schema must be previously inputted, indexing the coming collection.

Even though building a good search systems implies having a fair chunk of diversity in the indexed fields, some of the dataset columns were not relevant for the search.
In fields such as the popularity or the release year of a song, one can find utility aiding the search task, serving as a sorting or grouping method.

The text fields(title, lyrics, artist, album_name, tag) were all indexed as a Solr TextField type. This will be discussed further in the next section, since all filters and tokenizers used in the process will be thoroughly reviewed.

Table 1: Indexed parameters

Field	Indexed
title	true
tag	true
artist	true
lyrics	true
track_number	false
popularity	true
explicit	false
duration_ms	false
album_release_date	true
album_total_tracks	false
album_name	true

5.3 Schema

One of the main issues in the indexing operation was the fields language. Since one of the first requirements when searching for the dataset was the diversity of records in many dimensions(duration, language, genre), a big obstacle came up. Most of the filters and tokenizers in Solr work differently according to the language passed on the arguments. After some investigation, the conclusion came that interpreting any text from a certain field would not be too hoard, because python already provides a library that can do it. However, this did not prove to be useful. Since the titles and lyrics do not have a standardized and common language, there is no way to index every document according to its nature in different 'rows'. For example, the stop words filter would not be able to be used, because if we choose the language 'X' stop words, all the other ones will not change anything and can even malfunction. One solution could be splitting the multi-language fields into a group of them dedicated to each language. The problem was this would generate a huge amount of unnecessary data, and would overload the collection, so it was a dead-end. There are also fields specific to the English writing, but as explained there is no way to run it in a balanced way. In the end, even though many options were quickly discarded, there were still interesting alternatives to explore.

Firstly, after searching a few tokenizer alternatives, the standard one was kept in every field. [6] Its usage splits the text in tokens, using all white spaces and punctuation as edges. In other words, its perfectly useful since the

lyrics field, filled with commas and dots, turns into a clean token list and that is about what is needed.
Other options such as the N-Gram tokenizer could have been used, specially in the title field, since it allows the words to be interpreted with a certain degree of tolerance, however in a area like music, a single change in a query could have effect on the whole search.

Despite the earlier mentioned limitations, there were a fair amount of options to implement in the schema filters: [7]

- **ASCIIFoldingFilterFactory**
  The first filter is a converter from rare unicode characters(outside the first 127 ASCII values) to their correspondent code. This is useful because as previously mentioned, the dataset contains many different languages, which implies the existence of less used characters. With the conversion, all possible errors are prevented. For example, the letters Z and S with caron, or the oe connected latin character.

- **LowerCaseFilterFactory**
  This filter is pretty straightforward. Its function is converting every token to its lower case equivalent, so that no query fails because of an unmatching case.

- **ClassicFilterFactory**
  Similarly to the standard tokenizer, this filter's function is to clean excessive information in the tokens. Its usage is of help in this case, because besides the tokenizer parsing, this filter removes extra information, such as acronyms and some apostrophes. Despite the latter working differently in different languages, it works form some tokens and helps the overall scenario improving. For example, John Denver's Annie's Song would turn into Annie Song.

- **BeiderMorseFilterFactory**
  This filter is probably the most complex of all, however, its capabilities are really important given the language situation. One of its arguments allows the choice of the language, but it has an 'auto' value that sets the filter working for any language, as it detects the origin. Its usage only applies to names and it allows the user to miss-spell the artist name, because the Beider Morse is a phonetic filter and it can derive all similar names using its data. As it only works for names, it can only be used in the artist field. For example, if we type Stev, it will prompt similar options such as Steve, Steven or Stephen.

All filters are used across the full-text search fields, excepting the Beider Morse. It's worth noting that many other filters, such as the english specialized ones or other phonetic filters could have been used, which would seriously improve the search system, but it would require alterations to the collection and that is not what is intended.

6 Information Retrieval

As described in Section 4, there are some interesting search tasks we can explore using the Solr query tool, given our dataset domain, there are multiple possible and relevant queries, but many possible queries were left to explore since they didn't meet our information requirements or weren't going to have much impact on a regular person's search tasks. Given that, we have here some examples of interesting queries.

6.1 Query 1 - Phrase Match

Songs of Brian Adams since 2010.

Table 2: Parameters for Query 1.

Option	Values
q	artist: "Brian Adams"
fq	album_release_date: [2010 TO *]
sort	popularity desc
fl	lyrics, artist, title, album_name

In this query a phrase match is used. By closing the text with quotation marks, the search is processed with the two terms together. This allows us to search the artist name as a whole and skip the results of the artists with "Bruno" or "Adams" in their name.

6.2 Query 2 - Field Boost

Top 10 rock albums from the 90s.

Table 3: Parameters for Query 2.

Option	Values
q	tag: "rock" album_release_date: [1990 TO 2000]
q.op	AND
qf	tag^2 album_release_date
sort	popularity desc
fl	artist, album_name
rows	10

This query as a regular query using some delimiters and a range in the search parameters. However, a field boost is applied as well. This way, we elevate the importance of one field, in this example, the tag. A term boost could be used as well, in a query with at least two different terms/tokens and we could choose to give more relevance to one of them.

6.3 Query 3 - Wildcard

All songs about love, sorted by popularity.

Table 4: Parameters for Query 3.

Option	Values
q	title: lov * lyrics:lov * album_name:lov*
q.op	OR
sort	popularity desc
fl	lyrics, artist, title, album_name

In this query, wildcards are used to allow a search for love related songs, but including all derivations from it(loving, lovers, loved...). This is a enhancement that softens the restrictions of a possible search, allowing multiple results to appear.

6.4 Query 4 - Proximity Search

Rap songs from this century with the words "kill you" in the lyrics.

Table 5: Parameters for Query 4.

Option	Values
q	lyrics: "kill you"~10 album_release_date: [2000 TO *]
q.op	AND
sort	popularity desc
fl	lyrics, artist, title, album_name

Lastly, in this query a proximity search is used, so that a maximum limit of tokens is set between the terms in the phrase. This gives some freedom the simple phrase match wouldn't allow but isn't as permissive as a normal two term search.

6.5 Results

Our goal is to have a functional system capable of running the basic search tasks spotify provides while being able to execute queries across different sets of data, such as artist and songs altogether. We've proven our system can handle the basic queries, and one can go further and explore different parameters to refine the search.

7 Evaluation

Now it is time to evaluate and better adjust the system in order to obtain the best results. To evaluate the system we are going to test two different scenarios: schema without boosting and adjustments, and schema with boosting and some minor adjustments to the query. For the evaluation metrics we are going to use Precision and Recall. Precision measures retrieval specificity defined as the proportion of retrieved items that are judged by the user as being relevant; this measure penalizes system retrieval of irrelevant items (false positives) but does not penalize failures by the system to retrieve items that the user considers to be relevant (false negatives). Recall measures retrieval coverage defined as the proportion of the set of relevant items that is retrieved by the system, and therefore penalizes false negatives but not false positives. In the queries, EDisMax parser was later introduced to be used instead of DisMax since the first is an extension and improved version from the latter.

7.1 Scenario 1 - Remastered The Beatles Albums

We want to find the remastered albums of The Beatles, just by searching all albums for "Remastered" and artists for "The Beatles". We believe this is a rather precise query if the system is capable enough.

Table 6: Parameters used to boost the query.

Option	Values
q	artist: "The Beatles" album_name: Remastered
q.op	OR
fl	album_name, artist
qf	artist^2 album_name
mm	2
defType	dismax

After running the tests we came to the following results on Table

Table 7: Results for Scenario 1.

Metric	W/out boost	W/ boost
Average Precision	1	1
Precision at 5 (P@5)	0.60	–
Precision at 3 (P@3)	–	1

The following tables and P-R curves reveal the obtained results:

We can conclude the boosting techniques were helpful since the curve for the boosted system is a steady line at 1 for precision, and it skipped the wrong documents.

7.2 Scenario 2 - Funk Songs

Now we want to search for Brazilian funk songs. With that in mind, we searched all artists for "Mc*"(a common

Table 8: Albums retrieved with Schema without Boosting.

Artist - Album	Relevance
The Beatles - Revolver (Remastered)	R
The Beatles - 1 (Remastered)	R
The Beatles - Live At The BBC (Remastered)	R
The Beatles - The Beatles - Inspirations	N
The Ballet - I Blame Society	N

Table 9: Albums retrieved with Schema with Boosting.

Artist - Album	Relevance
The Beatles - Revolver (Remastered)	R
The Beatles - 1 (Remastered)	R
The Beatles - Live At The BBC (Remastered)	R

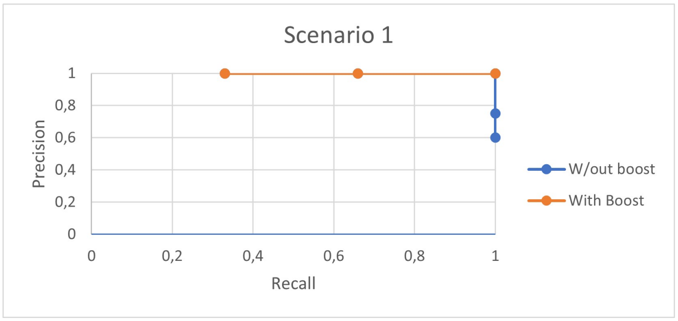


Figure 8: Experience 1.

prefix in many artists from the genre) and lyrics for words like ”amor”, also frequently used. This query shouldn’t have as many precision as the previous one, since it is less specific.

Table 10: Parameters used to boost the query.

Option	Values
q	artist: Mc* lyrics: amor
q.op	OR
fl	lyrics, artist, title, album_name
qf	artist^3 lyrics
mm	2
ps	3
defType	dismax

After running the tests we came to the following results on Table. Enhancing the artist field proved to be useful in the search.

Table 11: Results for Scenario 2.

Metric	W/out boost	W/ boost
Average Precision	0.34	0.64
Precision at 10 (P@5)	0.40	0.80

The following tables and P-R curves reveal the obtained results:

Table 12: Albums retrieved with Schema without Boosting.

Artist - Title	Relevance
Mr Jukes & Barney Artist - Autumn Leaves	N
Lyrics Born - Before and After	N
Lyrics Born - ir Racha	N
Heartist OTOD - Tango In Paris	N
MC Lars - Download This Song	R
MC Cabelinho - O Preço	R
Ana Carolina - Dias Roubados	N
Chocolate MC - Aveces	R
MC Nathan ZK - Maloka Firmeza	R
Mario Biondi - Bom de Doer	N

Table 13: Albums retrieved with Schema with Boosting.

Artist - Title	Relevance
Mr Jukes & Barney Artist - Autumn Leaves	N
Heartist OTOD - Tango In Paris	N
MC Lars - Download This Song	R
MC Cabelinho - O Preço	R
Chocolate MC - Aveces	R
MC Nathan ZK - Maloka Firmeza	R
MC Lars - Hurricane Fresh	R
Under MC - ¿Por qué Será?	R
Lis MC - Bad Bitch	R
Kid MC - Oração	R

We can conclude the boosting techniques were helpful since the curve for the boosted system reaches the 0.8 for precision, while the curve for the not boosted system only reaches 0.4.

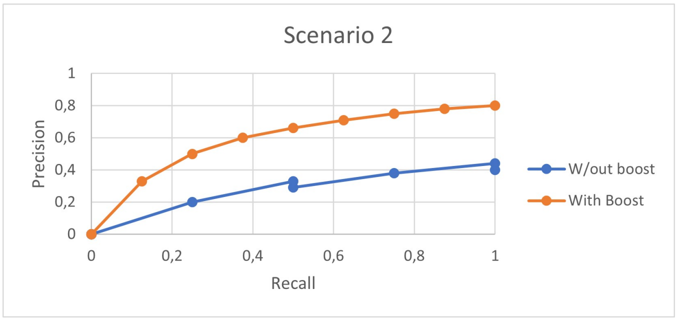


Figure 9: Experience 2.

7.3 Scenario 3 - Summertime

In this experience we want to find last years summer songs. With that in mind, we searched all lyrics and titles for words like ”summer” or ”beach”, ”hot” and ”sea”.

Table 14: Parameters used to boost the query.

Option	Values
q	lyrics: (summer beach hot sea) title: (summer beach hot sea)
q.op	OR
fq	album_release_date: 2022
fl	lyrics, artist, title, album_name
qf	lyrics^3 title
mm	2
ps	3
defType	dismax

After running the tests we came to the following results on Table

Table 15: Results for Scenario 3.

Metric	W/out boost	W/ boost
Average Precision	0.73	0.79
Precision at 10 (P@5)	0.60	0.70

The following tables and P-R curves reveal the obtained results:

Table 16: Albums retrieved with Schema without Boosting.

Artist - Title	Relevance
Flower Boy - Lemonade	R
Hewhocorrupts - Linguistic Violations	N
Louis Dunford - Summer in the Manor	R
James Bourne - Alone In Paradise	N
Frank Turner - Little Life	R
Diego Mar - Maloka Firmeza	R
burningforestboy - Days Before Summer	R
Spain - World Of Blue	R
Clutch - Red Alert Boss Metal Zone	N
punii+ - Battle	N

Table 17: Albums retrieved with Schema with Boosting.

Artist - Title	Relevance
Louis Dunford - Summer in the Manor	R
Flower Boy - Lemonade	R
Hewhocorrupts - Linguistic Violations	N
James Bourne - Alone In Paradise	N
Frank Turner - Little Life	R
Diego Mar - Second Chance	R
SWENDAL - Sunrise In Miami	R
Diego Mar - Second Chance	R
Spain - World Of Blue	R
punii+ - Battle	N

We can conclude the boosting techniques were helpful since the curve for the boosted system reaches a high level of precision than the system with no boost.

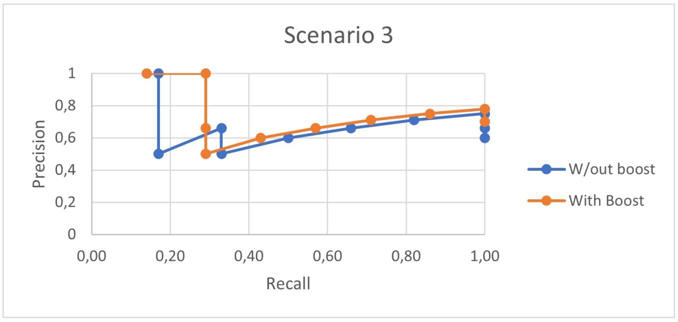


Figure 10: Experience 3.

7.4 Evaluation

Overall, the results were satisfactory. Average precision was most of the scenarios good in the boosted system as the Recall, meaning our top results were relevant and all relevant documents were retrieved. The boosted system improved metrics when compared to the regular one, meaning our optimizations had the desired effect. In all the graphs we can see that the boosted system outperforms the regular one in terms of precision for low recall values. This means that the first documents returned by the boosted system are more relevant than the ones returned by the regular one. Therefore, our boosts are having the desired effect of pushing more relevant results to the top.

8 Multi-Language Data-set Issue

As previously mentioned, the search system ended up in a stalemate, since the multi-language nature of the dataset was blocking the usage of many relevant filters. Because of that, the initial premise of having a diverse spread of songs was lifted, and only the english records were used in this stage.

8.1 Cleaning

In order to parse the data-set and to iteratively detect the languages in each song record, we used 'spaCy', which is a library for advanced natural language processing in python. [8] It contains a function that can identify with nearly 100% of confidence the origin of a text sequence, except for the musics with many languages in the lyrics(for example a french song with an english chorus). In these cases, the nlp language detector asserts the most present type, evidently with less confidence.

9 Schema upgrade

Considering the changes explained in the previous chapter. the set of available filters in the search system widened greatly. Every increment in the schema was thought according to each field necessities. [7]

- **EnglishMinimalStemFilterFactory & English-PossessiveFilterFactory** These filters are exclusive to the english language, however, they are an improvement comparing to the classic filter previously set to these variables, since they work on plurals and possessives, instead of the being just the latter. These suffixes are sliced from the word.

- **NGramFilterFactory** N-gram is a filter that generates tokens based on the original one, grouping the characters in all combinations possible withing a given range from minimum to maximum. This allows the search to be more tolerant, since word segments who could not be found previously if typed in the search, can now retrieve the pretended result. This was used in the title field, because it is a record frequently queried with errors or disparities to the pretended result(people normally do not know the exact name of a song, and they type what they heard). For example, the word love would turn into love + lov + ove, if the minimum defined was 3.

- **KStemFilterFactory** This filter is a stemmer who converts words derived from a common tense/verb(ex: jumping jumped $\longrightarrow$ jump) into the general token. This is useful in the lyrics field, since it's also very common for people to search similar words that do not match the reality, once again, the fault is in the lack of knowledge about the lyrics. This way, a broader set of songs will be matched when searched for a similar word sequence.

- **SynonymGraphFilterFactory** Lastly, still in the lyrics field, a synonym filter was added. This time, the main goal is not to improve the searches with identical morphology, but to provide results that are similar in meaning. This way, if a user needs to search about love, other synonyms will be added to the search(affection, care...), increasing immensely the number of relevant results, with no need to contain the original search word. To use the filter during indexing, another one needs to follow right after(Flatten Graph Filter) to squash the tokens in each other, since the original graph can't be parsed directly.

9.1 Synonym Usage

In order to use the synonym filter a file with the dictionary mapping must be provided in the pretended language, in this case english. The used data was retrieved using 'Wordnet', a database containing loads of lexical information about words from a large set of languages. The file containg the english synonyms was written in prolog, so a script was used to parse it and transform the words into the format used by Solr. [9]

10 Highlights

Besides the previous sections improvements, highlighting was also included in the search. This is a method that allows parts of the document matching the query to be referenced in the response. The parameters of the method are highly configurable allowing many different options.
This way, the highlighted boost is used with the main search query, on the attributes title and artist.
Highlighting is extremely configurable, perhaps more than any other part of Solr. There are many parameters for fragment sizing, formatting, ordering,

backup/alternate behavior, and more options that are hard to categorize. Nonetheless, highlighting is very simple to use. [10]

11 Schema Upgrade Evaluation

After all the updates in the schema, the queries were expected to improve the previously retrieved results. In order to evaluate the changes, the query used in the third scenario was repeated and compared to the data from the boosted query already instanced. The new results were as follows:

Table 18: Albums retrieved with Upgraded Schema and Boosting.

Artist - Title	Relevance
MOUNTAINKING - RED SEA DANCE	R
Ryan Robinette - Hello Summer	R
NerdOut - Summertime	R
ARY - The Sea	R
Hannah Hart -Summer Jam	R
Stephen Schwartz - Simple Joys	R
BZN - Hilee Hilay	R
Death Cab for Cutie - Coney Island	R
Danny B - AwhYeahOohWeE	R
Jaicko - Caribbean Girl	R

Table 19: Results for Scenario 4.

Metric	W/out boost	W/ boost
Average Precision	0.79	1
Precision at 10 (P@5)	0.70	1

From the retrieval operation, it is clear that the schema changes had impact in the query results and improved the overall outcome.

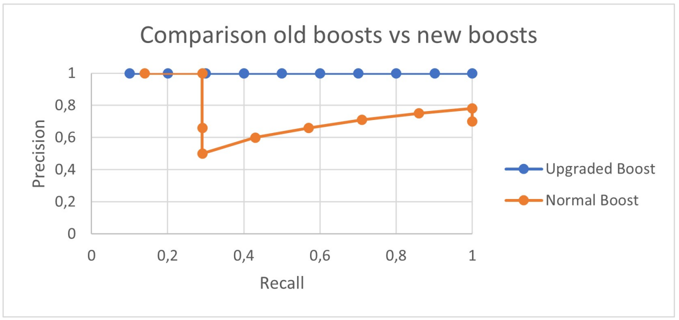
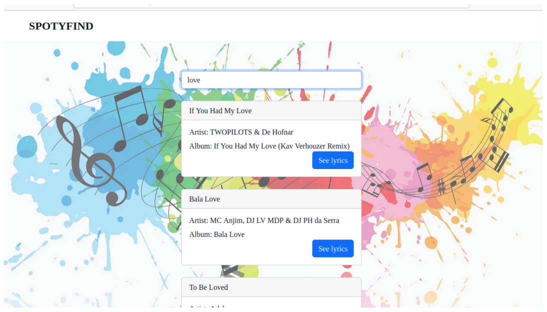


Figure 11: Experience 4.

12 Web Application

One of the improvements introduced to the search system was an interface. Solr API does not have a great accessibility, since its usage is confusing and difficult to understand for the general user.
To improve the search user accessibility, a simple interface was created. The application, SPOTYFIND, a NextJS app, is an application with a free text search bar with implemented suggestions(maximum of 20 suggestions) and using the id as dictionary. It shows real time results of the search according to what the user is typing. and it provides a simple interface in which the results may be visualised clearly and presented in an organized way. As we can see in figure 13, there is a button ”See lyrics” on each music card. If the user clicks it, will send the user to the music's lyrics page, visible on figure 14. In the figure 12, the main image from the app is shown. In the end, an appealing design was achieved. It is a very important part of the user experience as it motivates the person interacting with it to use the search system and facilitates its usage. The design must be thought for the user and its needs and to be the most straight-forward possible to reduce the entropy.

Figure 12: Spotyfind Main Page.



If You Had My Love
Artist: TWOPILOTS & De Hofnar
Album: If You Had My Love (Kav Verhouzer Remix)

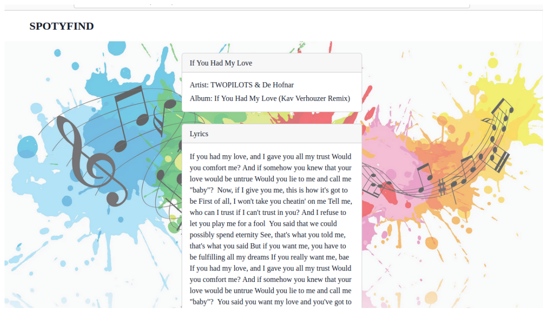
Bala Love
Artist: MC Anjim, DJ LV MDP & DJ PH da Serra
Album: Bala Love

To Be Loved

Figure 13: Spotyfind Search Page.

12.1 Tools

In the frontend, the used tools were NextJS[11] and React[12], adding up to tailwind for the CSS definition [13]. for the backend, the chosen technology was Express [14]. Regarding the backend of the search system,



If You Had My Love

Artist: TWOPILOTS & De Hofnar

Album: If You Had My Love (Kav Verhouzer Remix)

Lyrics

If you had my love, and I gave you all my trust Would you comfort me? And if somehow you knew that your love would be untrue Would you lie to me and call me "baby"? Now, if I give you me, this is how it's got to be First of all, I won't take you cheatin' on me Tell me, who can I trust if I can't trust in you? And I refuse to let you play me for a fool You said that we could possibly spend eternity See, that's what you told me, that's what you said But if you want me, you have to be fulfilling all my dreams If you really want me, bae If you had my love, and I gave you all my trust Would you comfort me? And if somehow you knew that your love would be untrue Would you lie to me and call me "baby"? You said you want my love and you've got to

Figure 14: Spotyfind Lyrics Page.

axios was used which is a promise-based HTTP Client and makes it possible to write async/await code to perform XHR requests very easily. Initially, to simplify the Solr requests, it was created a new axios instance where the desired configuration was defined (Solr URL and a timeout).

13 Improvements

The work from previous iterations was corrected and improved in several parts.
In the document header and beginning, the Abstract and introduction sections were mostly revamped, Keywords were relocated from Introduction to Abstract and personal academic emails were added.
The picture related changes were the Final dots addition to captions which were missing it, and every graphic caption was moved from bottom to top and the graph data was corrected with the updated values.
In the query parsing, the EDisMax specification was added. The correct recall and average precision values were also updated into the respective tables.
The schema was also filled with more practical examples. Document definiton is now a bit more detailed and the index fields are now specified in the indexing subsection.

14 Conclusions

This paper addressed a study with three distinct milestones in the construction of a system capable of answering complex queries about music data. The initial dataset included lyrics from various songs and was then complemented with information from the Spotify API. The retrieved data was cleaned, refined and conceptually described to represent the main classes and associations between them. From this point, it was possible to identify information retrieval tasks that were later answered, using Solr and our schema. Since the multi-language nature of the dataset was blocking further improvements, it was updated to include only english songs. Results were positive and consistent with reality for what would be expected and showed very significant differences when boosts and the latter upgrades were used. The results achieved matched the planned goals and the study can be considered a success.

15 Future Work

Having in consideration the entirety of this work, it is considered relevant to undertake the following steps to enhance the developed search system:

- improve and search for complementary relevant data
- implementation of the Rocchio Algorithm
- interface new features
- integrates artificial intelligence algorithms to analyze user relevance feedback, taking careful advantage of information like clicks or frequent searches to evaluate query results and continuously improve the system's quality
- understand and utilize each user's personal preferences to retrieve more relevant information and generate pertinent suggestions, tailoring a personalized experience for each use

References

[1] Saurabh Shahane. 2022. Short dataset, Last Access Date: 12/12/2022.

[2] Anderson Neisse. 2022. Dataset with few languages, Last Access Date: 12/12/2022.

[3] NikhilL Nayak. 2022. Used Dataset, Last Access Date: 12/12/2022.

[4] Spotify. SpotifyAPI, Last Access Date: 12/12/2022.

[5] Solr Apache. Solr Query Parser Documentation, Last Access Date: 12/12/2022.

[6] Solr Apache. Solr Tokenizers Documentation, Last Access Date: 12/12/2022.

[7] Solr Apache. Solr Filters Documentation, Last Access Date: 12/12/2022.

[8] Explosion.ai. spaCy library, Last Access Date: 12/12/2022.

[9] Princeton University. WordNet, Last Access Date: 12/12/2022.

[10] Solr Apache. Solr Highlights Documentation, Last Access Date: 12/12/2022.

[11] NextJS. NextJS documentation, Last Access Data: 12/12/2022.

[12] React. React documentation, Last Access Data: 12/12/2022.

[13] Tailwind. Tailwind documentation, Last Access Data: 12/12/2022.

[14] Express. Express documentation, Last Access Data: 12/12/2022.