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

André Santos, Edgar Torre, Miguel Amorim
Faculty of Engineering of the University of Porto

ABSTRACT

Information has never been so available as it is nowadays. That's why preparing and processing this information is more important than ever. This report aims to document the analysis and preparation of a database to extract correct, accurate and detailed information that will be useful to its owner.

1 INTRODUCTION

The project 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).

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

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, 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).

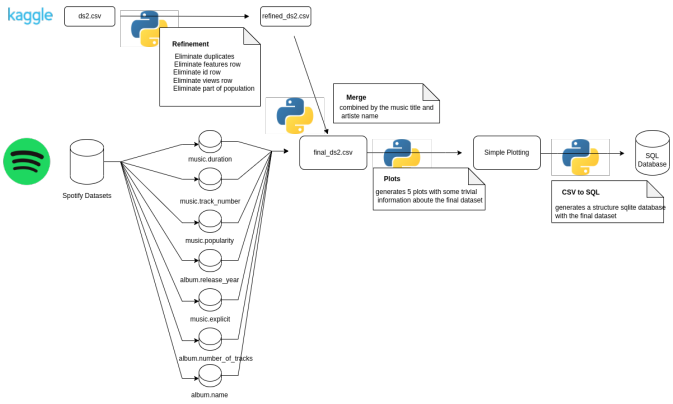

Figure 1: Pipeline Scheme

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.

3.2 Domain Conceptual Model

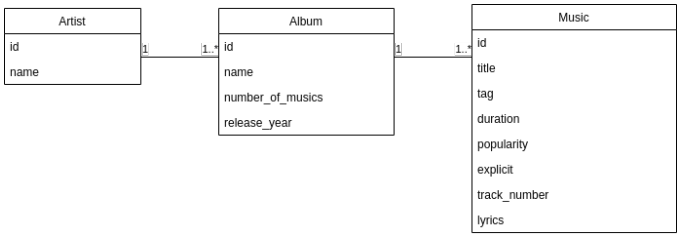

Figure 2: Domain Conceptual Model

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.

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.

To start, 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.

The second plot 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.

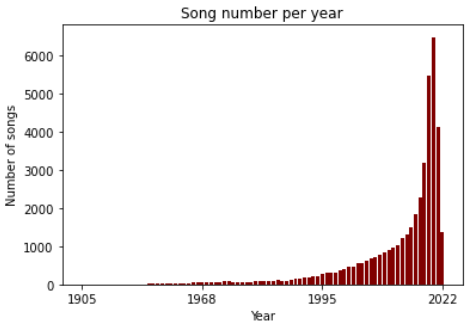

Figure 3: Musics per Year

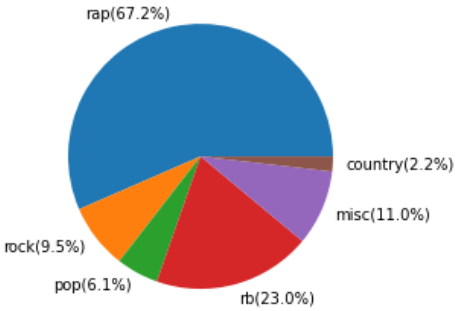

Figure 4: Musics Grouped by Genre

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). 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. We concluded that most songs have mostly 2 minutes in duration. However, we 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.

To finish, 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.

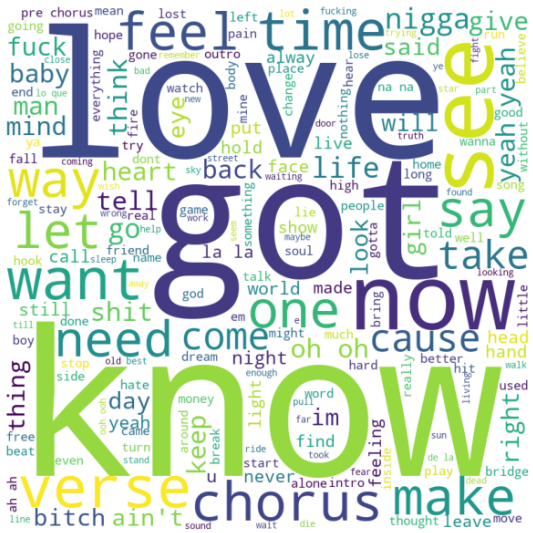

Figure 5: Lyrics Word Cloud

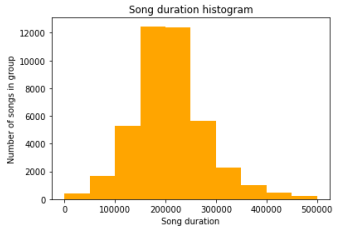

Figure 6: Music Duration Histogram

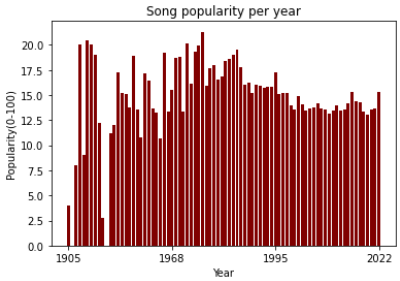

Figure 7: Music Duration Histogram

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.

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.

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:

- **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. [7]

- **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. [8]

- **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. [9]

- **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. [10]

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.

Option	Values
q	artist: "Brian Adams"
fq	album release date: [2010 TO *]
sort	popularity desc
fl	lyrics, artist, title, album name

Table 1: Parameters for Query 1.

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.

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

Table 2: Parameters for Query 2.

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.

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.

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

Table 3: Parameters for Query 3.

6.4 Query 4 - Proximity Search

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

Option	Values
q	lyrics: ”kill you”∼10 album release date: [2000 TO *]
q.op	AND
sort	popularity desc
fl	lyrics, artist, title, album name

Table 4: Parameters for Query 4.

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.

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.

Option	Values
q	artist: ”The Beatles” album_name: Remastered
q.op	OR
fl	album_name, artist
qf	artistˆ2 album_name
mm	2
defType	dismax

Table 5: Parameters used to boost the query.

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

Metric	W/out boost	W/ boost
Average Precision	0.88	1
Precision at 5 (P@5)	0.60	–
Precision at 3 (P@3)	–	1

Table 6: Results for Scenario 1.

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

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 7: 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

Table 8: Albums retrieved with Schema with Boosting.

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.

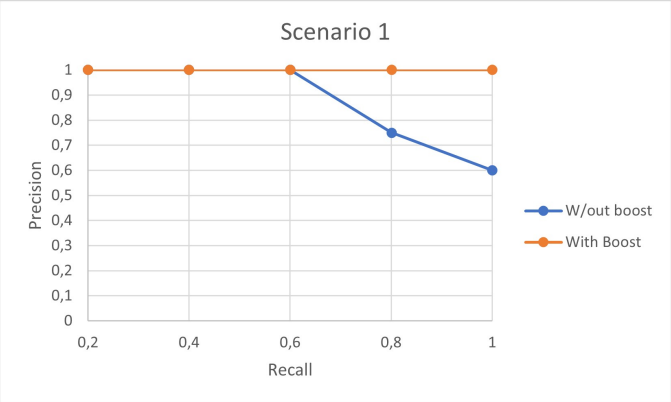

Figure 8: Experience 1

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 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.

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

Table 9: Parameters used to boost the query.

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

Metric	W/out boost	W/ boost
Average Precision	0.20	0.51
Precision at 10 (P@5)	0.40	0.80

Table 10: Results for Scenario 2.

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 reaches the 0.8 for precision, while the curve for the not boosted system only reaches 0.4.

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 11: Albums retrieved with Schema without 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

Table 12: Albums retrieved with Schema with Boosting.

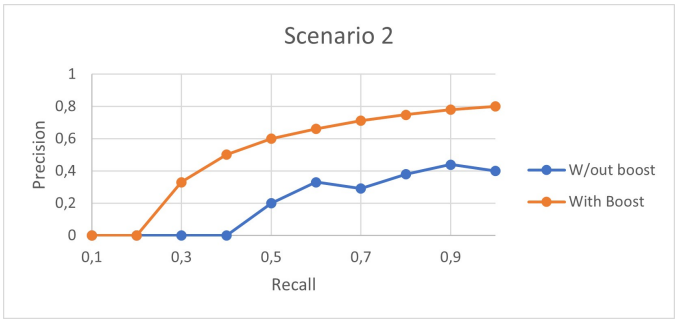

Figure 9: Experience 2

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".

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

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

Option	Values
q	lyrics: (summer beach hot sea) title: (summer beach hot sea)
q.op	OR
fq	$album_{r}elease_{d}ate$: 2022
fl	lyrics, artist, title, album_name
qf	lyrics^3 title
mm	2
ps	3
defType	dismax

Table 13: Parameters used to boost the query.

Metric	W/out boost	W/ boost
Average Precision	0.67	0.74
Precision at 10 (P@5)	0.60	0.70

Table 14: Results for Scenario 3.

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 15: Albums retrieved with Schema without 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

Table 16: Albums retrieved with Schema with Boosting.

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.

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.

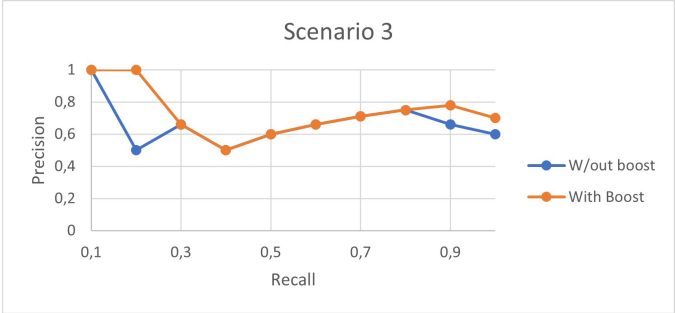

Figure 10: Experience 3

8 Improvements

Some parts of this document were revised in the latter stages of the project. Some of the changes were minor adjustments, like style and writing related. The introduction and abstract sections were the most updated since, not all information was correctly explained/presented. The references section was also added in this Milestone, and information from both deliveries was written.

9 Conclusions

This paper addressed two distinct milestones in the construction of a system capable of answering complex queries about spotify songs data. 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. Results were positive for what would be expected by a user and were consistent with existing literature. From now on, the future work will be working on a way where a user can interact with our retrieval system, like an application web.

References

[1] Saurabh Shahane. 2022. Short dataset.

[2] Anderson Neisse. 2022. Dataset with few languages.

[3] NikhilL Nayak. 2022. Used Dataset.

[4] Spotify. SpotifyAPI.

[5] Solr Apache. Solr Query Parser Documentation.

[6] Solr Apache. Standard Tokenizer Documentation.

[7] Solr Apache. ASCII Folding Filter Documentation.

[8] Solr Apache. Lower Case Filter Documentation.

[9] Solr Apache. Classic Filter Documentation.

[10] Solr Apache. Beider-Morse Filter Documentation.