NDB Music Recommender

TFM Memoria

Ana Eguía López
TFM – Master Data Science, ed. 23
Madrid, July 2021
https://github.com/aeguia/ndb-music-recommender.git

Index

1. Introduction

New generations do not know what it means to buy a cd, maybe a vinyl that has brought back into fashion, much less a cassette tape, not even buying a digital download. Throughout the last decade, the consumption of streaming music has been increasing in such a way that it has caused the drop down in physical sales and with it, new consumer platforms have emerged. Among the most prominent: Spotify, Apple Music, Tidal, Amazon Music Unlimited and YouTube Music.

Millions songs at a click button, what´s next? Nothing, relax... open your ears, just let yourself go, Recommender Systems came here to stay.

The project that I propose seeks to generate a Music Recommender whose target audience would be mainly attendees of Live Music Festival, Noches del Botánico (NDB).

NDB is a relatively young Festival with only 5 editions until date that takes place throughout the months of June and July in Madrid where almost every evening a live concert from a renowned artist and an opening artist are programmed.

NDB festival is characterized by the eclecticism of its musical commitment where in each edition all musical genres coexist from flamenco to jazz, rock to blues through urban rhythms, electronics or any avant-garde music.

To collect the data to build the model I have used Spotify API and the public Million Playlist Dataset which includes 1 million playlists with a total of more than 2.2 million songs, created by US users on the Spotify platform between January 2010 and October 2017.

Taking advantage of the existence of the MPD dataset with more than 65 million rates of 2,2 million songs collected from user´s feedback implicitly in the form of their playlists, I have used it to build a machine learning model of type Collaborative Filtering (CF) item based.

2. Datasets Description

The start point of my project is an excel file (*NDB_artist_2021_2016.csv*) containing a list of concerts taking place at NDB Festival (Noches del Botánico) 2016 – 2021.

Editions	Concerts	Artists
5	> 211	> 194

Table 1: NDB Concerts

File is processed using Spotify API in order to get artists features (*artists_ndb.csv*) that will be use to filter MPD dataset input, as well to build random playlists of tracks from artists playing at NDB Festival editions that will be use as input in our final Model.

Field	Type	Description
artist_name	str	The name of the artist
artist_spid	str	Spotify ID for the artist
artist_uri	int	Spotify URI for the artist
artist_followers	int	Number of followers of the artist
artist_popularity	int	The popularity of the artist
artist_genres	list of str	list of the genres the artist is associated with
2016	int	Artist plays at edition 2016 [1-0]
2017	int	Artist plays at edition 2017 [1-0]
2018	int	Artist plays at edition 2018 [1-0]
2019	int	Artist plays at edition 2019 [1-0]
2021	int	Artist plays at edition 2021 [1-0]

Table 2: NDB Artist features fields

	artist_name	artist_spid	artist_uri	artist_followers	artist_popularity	artist_genres	2016	2017	2018	2019	2021
45	Cécile McLorin Salvant	6PkSULcbxFKkxdgrmPGAvn	spotify:artist:6PkSULcbxFKkxdgrmPGAvn	63909	52	[contemporary vocal jazz, jazz pop, vocal jazz]	1	0	1	0	1

Figure 1: df_artist_features row sample

In order to simulate ratings of tracks played by artists taking place at any of the different editions of NDB Festival (Noches del Botánico) 2016 – 2021, I have used public dataset (MPD), *The Spotify Million Playlist Dataset*. MPD dataset contains 1 million playlists consisting of over 2 million unique tracks by nearly 300.000 artists created by users on the Spotify platform.

I have used the list of NDB artist for filtering MPD playlists read in, keeping only those containing tracks from at least one of the artists playing at the Live Festival.

Please, check README file for further information regarding data collection.

The Million Playlist Dataset consists of 1.000 slice files, following naming convention:

*mpd.slice.*STARTING_PLAYLIST_ID_-_ENDING_PLAYLIST_ID*.json*

From *mpd.slice.0-999.json* To *mpd.slice.999000-999999.json*

Each .json slice subfile contains 1.000 playlists containing the following attributes:

```
{
  "info": {
    "generated_on": "2017-12-03 08:41:42.057563",
    "slice": "0-999",
    "version": "v1"
  },
  "playlists": [
    {
      "name": "Throwbacks",
      "collaborative": "false",
      "pid": 0,
      "modified_at": 1493424000,
      "num_tracks": 52,
      "num_albums": 47,
      "num_followers": 1,
      "tracks": [
        {
          "pos": 0,
          "artist_name": "Missy Elliott",
          "track_uri": "spotify:track:0UaMYEvWZi0ZqiDOoHU3YI",
          "artist_uri": "spotify:artist:2wIVse2owClT7go1WT98tk",
          "track_name": "Lose Control (feat. Ciara & Fat Man Scoop)",
          "album_uri": "spotify:album:6vV5UrXcfyQD1wu4Qo2I9K",
          "duration_ms": 226863,
          "album_name": "The Cookbook"
        },
        {
          "pos": 1,
          "artist_name": "Britney Spears",
          "track_uri": "spotify:track:6I9VzXrHxO9rA9A5euc8Ak",
          "artist_uri": "spotify:artist:26dSoYclwsYLMAKD3tpOr4",
          "track_name": "Toxic",
          "album_uri": "spotify:album:0z7pVBGOD7HCIB7S8eLkLI",
          "duration_ms": 198800,
          "album_name": "In The Zone"
        },
```


Figure 2: input *json slice file format*. Playlists and Tracks attributes

One of the first problems I had to overcome at the beginning was being able to read such a huge volume of data from json files. If I used json.read to load content all at once I run out of memory, that´s why I finally used ijson that only kept in memory current file.

I also had to divide initial read in data phase into two to avoid memory overflow errors. First, parse json files to panda's dataframe to get playlists_df and tracks_df and a second step build co-occurrence matrix with tracks ratings per playlist.

MPD json to NDB files are read and processed into 2 pandas dataframes, *playlists_df* and *tracks_df* following detailed description.

- Playlists

Playlists metadata. The list of tracks keeps original order from json file.

Field	Type	Description
pid	int	Playlist ID number [0:279.032]
name	str	Playlist name
num_tracks	int	Number of unique tracks
num_artists	int	Number of unique artist
num_albums	int	Number of unique albums
num_followers	int	Number of followers
duration_ms	int	Duration of the playlist in milliseconds
tracks	list of str	List of Spotify IDs of tracks in the playlist

Table 3: Playlists fields

	pid	name	num_tracks	num_artists	num_albums	num_followers	duration_ms	tracks
94265	94265	alt rock	27	26	27	1	6881096	[55C8Zut2Ug8wPNrU4OJYPd, 6O8EJ1tfMWYnUK73bqeNE...

Figure 3: playlists_df row sample

- Tracks

Tracks metadata. Added column *spid* splitting track Spotify ID from track Spotify URI. Also added track number id, *tid*, that will be use to build co-occurrence matrix.

Field	Type	Description
track_uri	str	Spotify URI for the track
track_name	str	Track name
duration_ms	int	Track length in milliseconds
artist_uri	str	Spotify URI for the artist
artist_name	str	Artists who performed the track
album_uri	str	Spotify URI for the album
album_name	str	Album on which the track appears
spid	str	Spotify ID for the track
tid	int	Track ID number [0:1124914]

Table 4: Tracks fields

	track_uri	track_name	duration_ms	artist_uri	artist_name	album_uri	album_name	spid	tid
543773	spotify:track:2wXP4vskHaiPz0S0mBEF89	Worms	63026	spotify:artist:2wzMOQwNT6ZvVB4amvhFAH	The Pogues	spotify:album:4V92Puney9WxGPeckt1LG4L	If I Should Fall From Grace With God	2wXP4vskHaiPz0S0mBEF89	543773

Figure 4: tracks_df row sample

Playlists and Tracks file are read as input parameters to build the co-occurrence matrix.

- Co-occurrence Matrix

Sparse matrices are memory efficient data structures that enable us store large matrices with very few non-zero elements as well perform complex matrix computations.

In order to create our matrix, first it´s needed to build a dictionary mapping Spotify Ids from Playlist´s Tracks list to numeric track ID.

Each Playlist represents a user and each track represents an item. Playlist Ids will be matrix rows and Tracks Ids will be matrix columns. To implement ratings it is consider a value of 1 if Track is included in the Playlist and 0 if not.

For matrix definition, I have used Dictionary of Keys based sparse matrix, dok format.

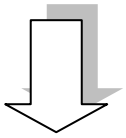
dok_matrix((M,N), [dtype]), where M: length(playlistsIDs) and N: length(tracksIDs) and dtype as int8 to save memory considering ratings are always binary data, 1 or 0.

To load data as 1, Playlist´s (pid) Tracks list are mapped from track Spotify ID (spid) to track ID number (tid).

Finally, matrix is saved converted to Compressed Sparse Row (csr) format.

	Track 0	Track 1	Track 2	...	Track N
Playlists 0	1	1	0	...	1
Playlists 1	0	1	1	...	0
...	0	0	1	...	0
Playlists N	0	0	0	...	1

Table 5: Sparse Matrix format



Row	0	0	0	1	1	⋯	N
Column	0	1	N	1	2	⋯	N
Value	1	1	1	1	1	⋯	1

Table 6: Compressed Sparse Row (csr) format.

3. Exploratory Data Analysis

3.1. Playlists

After applying NDB artists filter to read input MPD json datafiles, the total numbers are the following:

279.032 Playlists	1.124.914 Tracks	172.112 Artists	426.060 Albums	23.545.215 Ratings

Table 7: NDB_MPD dataset Totals

The presence of artists from NDB Festival in the dataset (Figure 5) is very small, representing 1.4% of the total number of tracks and 3.2% from total ratings, but enough to use this subgroup as seed to generate random playlists and use it to test my final model.

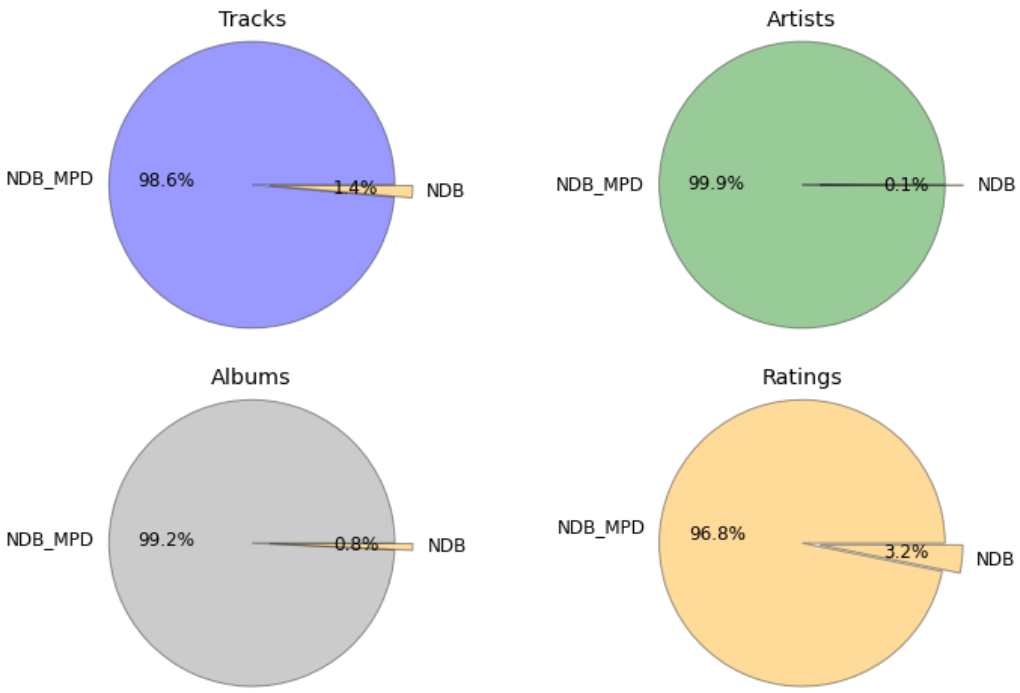


Figure 5: Pie plots with Representation of NDB Artists within NDB_MPD dataset

If we start by having a look to the number of tracks, artists and albums per playlist (Figure 6) we can see that follows a right or positively skewed distribution, it´s better to use median instead of mean to check average values.

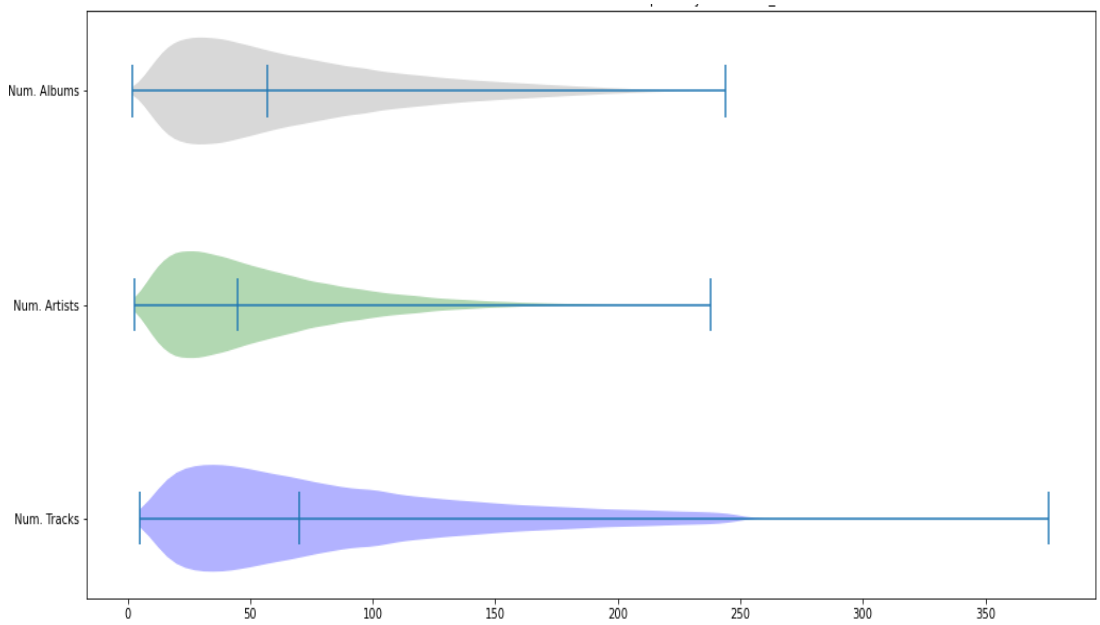


Figure 6: Violin plot with Distribution of Tracks, Artists and Albums per Playlist in NDB_MPD dataset

Median of tracks per playlist: 70
Median of artists per playlist: 45
Median of albums per playlist: 57

3.2. Tracks

Let´s continue with most popular tracks in our Dataset (Figure 7), this is calculated considering the frequency in which a specific track appears in a different playlist. As rates are binary data, get same result by count frequency or summarizing track rates.

Coincidentally, the most frequent song is “Ophelia” by The Lumineers which is the band that inaugurated NDB Festival in its first edition of 2016, by doing sold out and has another 2 tracks in Top 50 tracks. There are other tracks from NDB Artists in Top 50, contemporary electronic music artists like M83 and Jessie J but also oldies but goodies like 2 tracks from Daryl Hall & John Oates.

Many of the songs from this set (“Brown Eyed Girl” by Van Morrison - 1967; “Bohemian Rhapsody” by Queen - 1975; “Hotel California” by The Eagles – 1976; “My Girl” by The Temptations – 1965) are classics of modern music that have passed from generation to generation regardless their publication date and everyone knows them from the Radio or Television when not first-hand. The songs played by current bands are the kind of music that a wide audience likes.

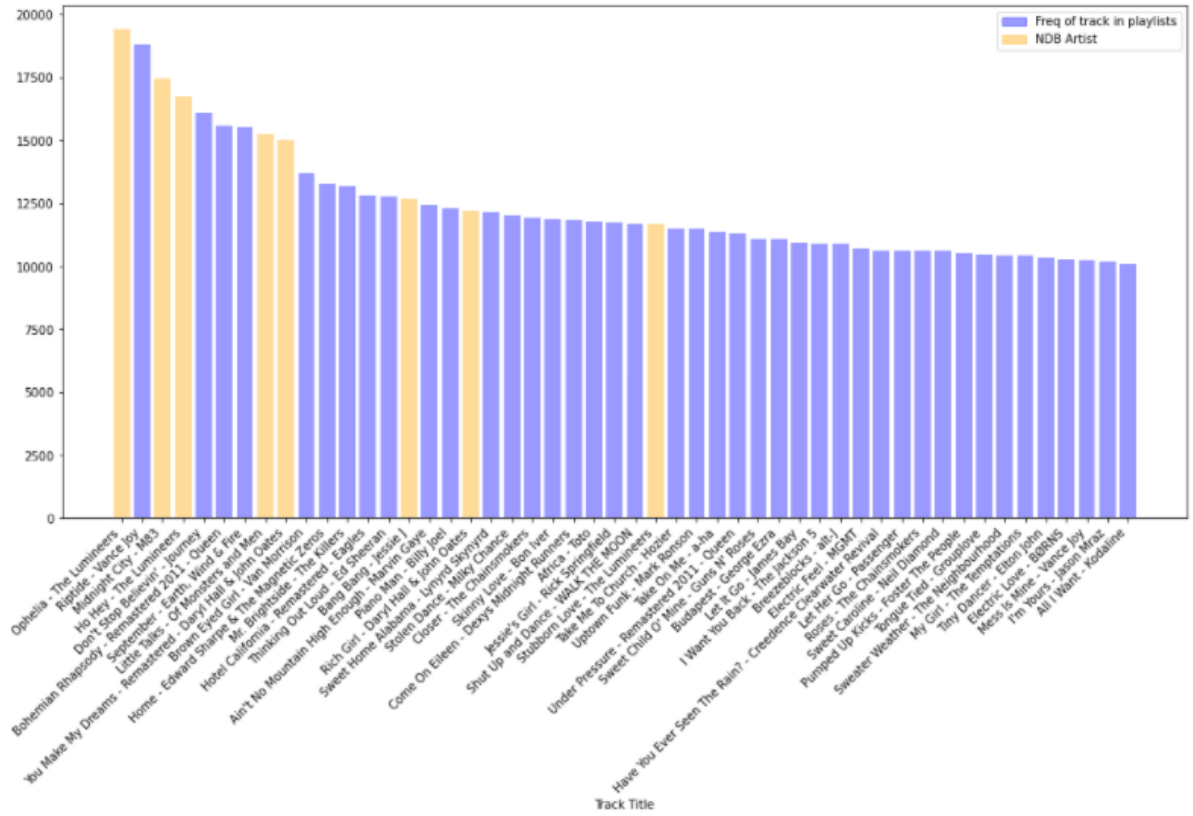


Figure 7: Bar plot with Top 50 Tracks in NDB_MPD

Analyzing results for the set of Top artist in NDB_MPD dataset (Figure 8) calculated summarizing track rates grouped by artist, we can see that some of the Top artists were also included in the Top tracks set, like The Lumineers, Ed Sheeran, The Killers or The Chainsmokers as well classic artists like Queen, Elton John and Billy Joel.

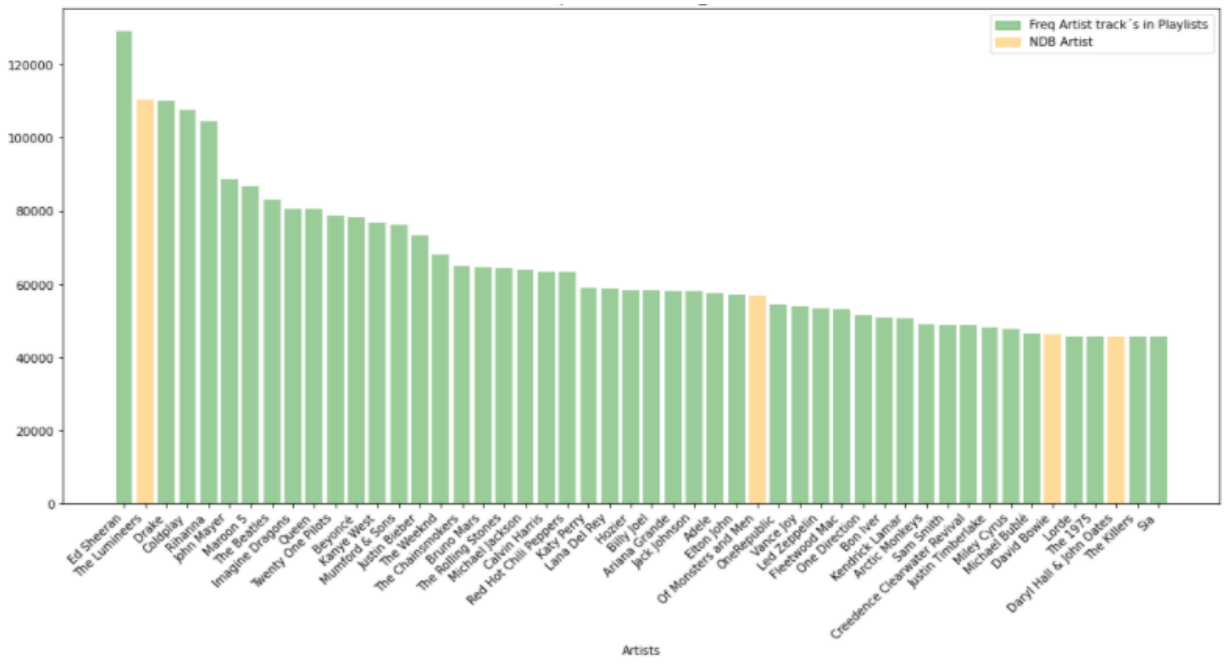


Figure 8: Bar plot with Top 50 Artists in NDB_MPD

All the current artists included in the set are renowned artists in different musical genres: Hip-Hop – Drake, Kanye West, Kendrick Lamar; R&B – Rihanna, Beyoncé; Pop – Justin Bieber, Katy Perry, Sam Smith and if we focus on immortal artists, there we can find names as The Beatles, The Rolling Stones or David Bowie.

Figure 9 shows total number of unique tracks from Top 50 Artists in the dataset, the graph shows how the longer bars coincide with artists with very long careers and an extensive discography like David Bowie, The Rolling Stones, The Beatles, Queen and so on. But it is worth highlighting younger artists with a much smaller repertoire sneak into the group because their songs are included in a large volume of playlists, between them once more The Lumineers, Of Monsters and Men or Lorde.

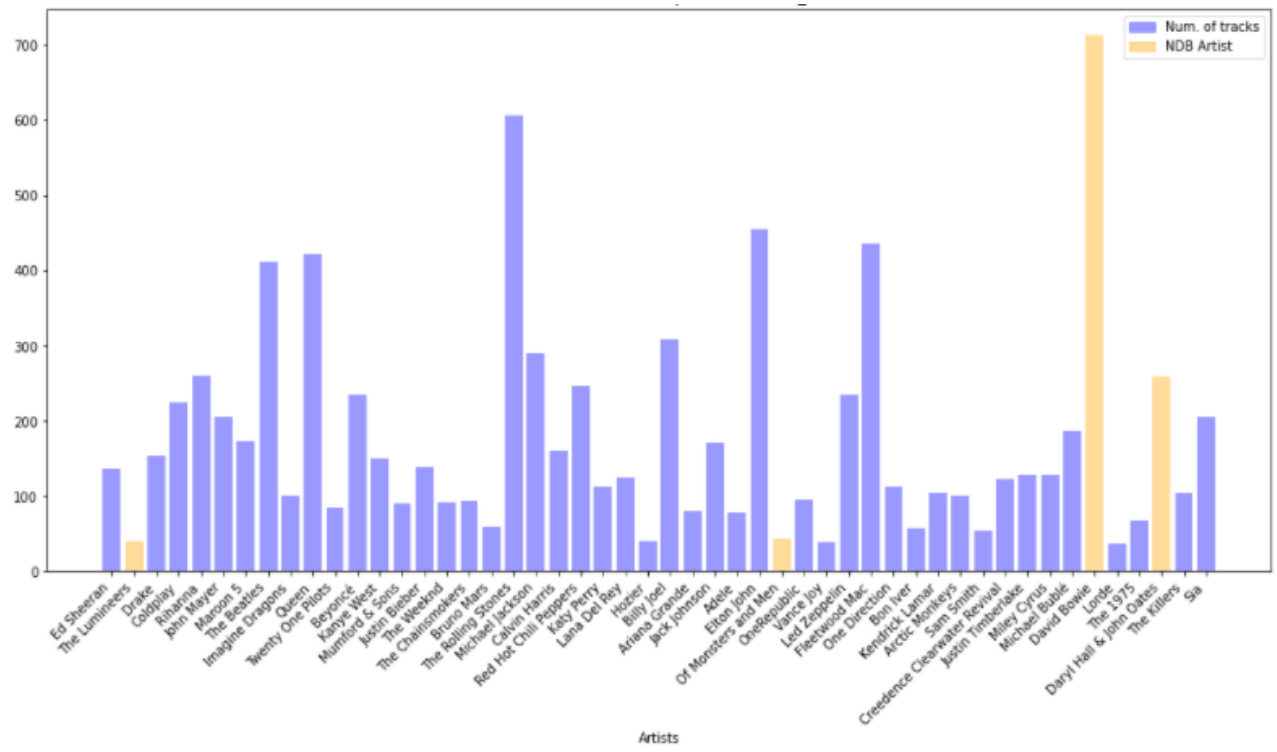


Figure 9: Bar plot with Num of tracks from 50 Top Artists in NDB_MPD

Finally, in the Figure 10 we can see that most of the current artists have disappeared from the list by removing the filter of Top 50 Artists and remain only those ones with higher volume of tracks in the dataset. And obvious is also shown, classical music composers (Bach, Chopin...), deceased artists (Sinatra, Elvis...) or music legends who are still active today (Dylan, Neil Young...) all sharing the fact of having very long careers behind them, extensive repertoires as well as a vast discography.

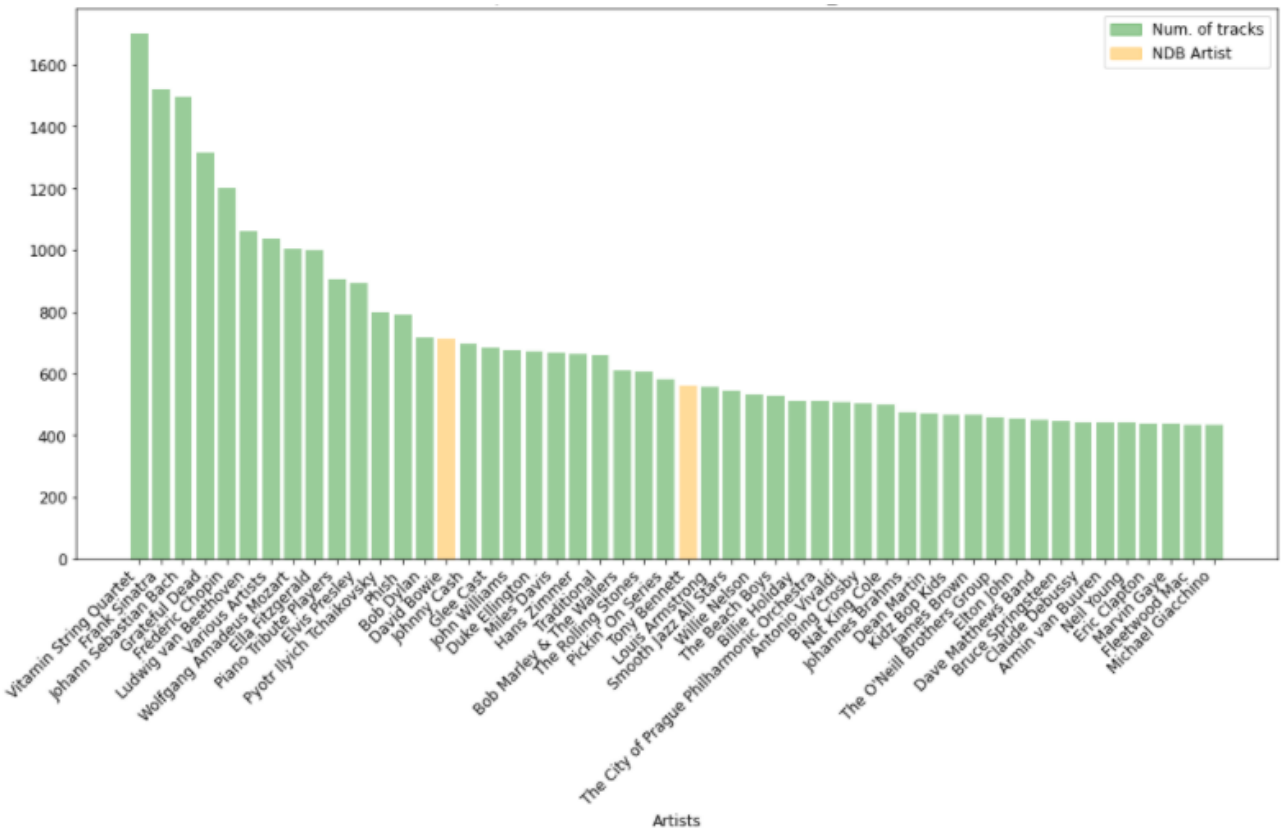


Figure 10: Bar plot Top 50 Artists with more tracks in NDB_MPD

3.3. Co-Occurrence Matrix

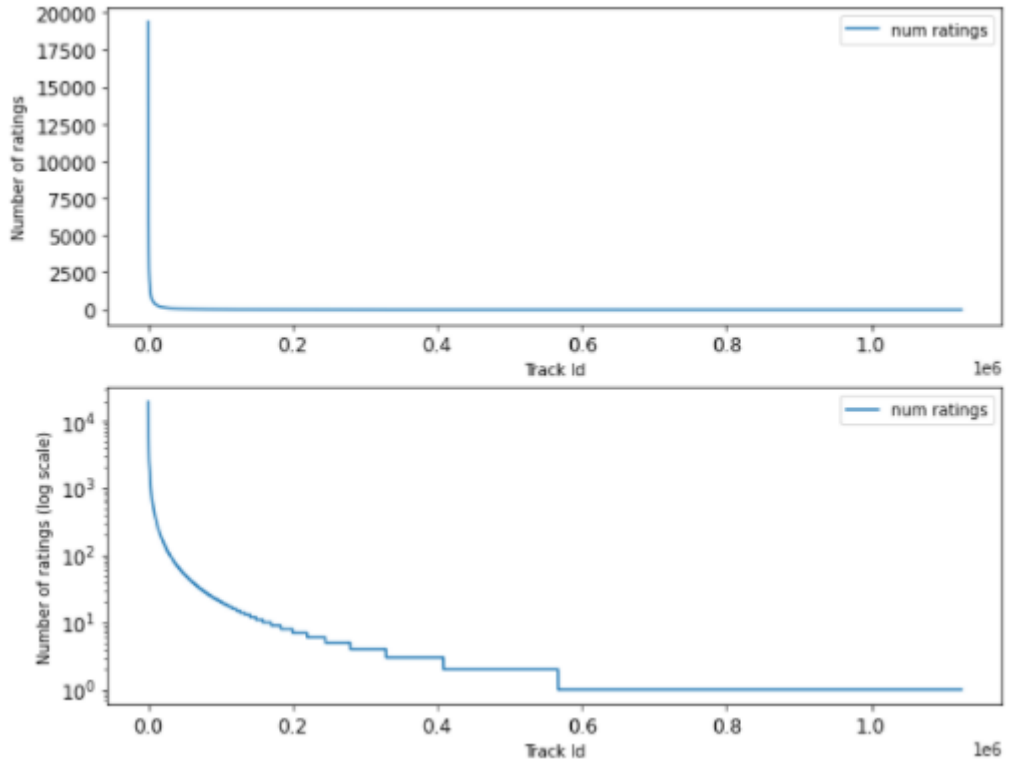


Figure 11: Long Tail: Rating Frequency of Tracks (by Total Ratings and in Log scale) - MPD_NDB

If we plot the Rating Frequency for tracks in our dataset we can see that follows a called “Long Tail” distribution. Typically only a small fraction of items are rated frequently, this is referred as the ”head” while the vast majority of items are placed in the “long tail” and represents a small percentage of the whole ratings.

Rating frequency in log scale reveals that just a 15% of the tracks (168.737 out of 1.124.914) have been rated 10 times or more while close to half of the dataset only have 1 rate (506.211). This results in a highly skewed distribution of ratings.

All this facts confirms what is usual in this kind of Systems, suffers from data sparsity.

The sparsity level in our rating matrix is almost 1, 0.999925.

Sparsity = 1 — |R|/|I|*|U|

Where,

- R = Rating
- I = Items
- U = Users

4. NDB Recommender System

A **recommender** (or recommendation) **system** (or engine) is a filtering system which aim is to predict a rating or preference a user would give to an item, for example; a film, a product, a song, etc. It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user. It looks at the items they like and combines them to create a ranked list of suggestions.

The data needed to build it contains a set of items and a set of users who have reacted to some of the items. The reaction can be explicit (rating on a scale of 1 to 5, likes or dislikes) or implicit (viewing an item, adding it to a wish list, the time spent listening a song).

Recommender systems are generally divided into two main categories: **collaborative filtering** and **content-based** systems. We could also consider a third category, **hybrid** systems which use a combination of the other two.

- **Simple recommenders**: offer generalized recommendations to every user, are based on the popularity of the item. The basic idea behind this system is that items that are more popular and critically acclaimed will have a higher probability of being liked by the average audience.
- **Content-based recommenders**: suggest similar items based on a particular item. This system uses item metadata to make these recommendations. The general idea behind these recommender systems is that if a person likes a particular item, he or she will also like an item that is similar to it. And to recommend that, it will make use of the user's past item metadata.
- **Collaborative filtering engines:** these systems are widely used, and they try to predict the rating or preference that a user would give an item-based on past ratings and preferences of other users. Collaborative filters do not require item metadata like its content-based counterparts.
- **Hybrid systems**: hybrid approaches can be implemented in several ways: by making content-based and collaborative-based predictions separately and then combining them; by adding content-based capabilities to a collaborative-based approach (and vice versa); or by unifying the approaches into one model.

My project focuses on *collaborative filtering* (CF) systems and use item-based collaborative filtering to perform track recommendations. CF is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person B on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person.

With the project, I´ll build a model that will provide a playlist with 50 recommended tracks giving as input a simulated user-playlist randomly generated with tracks from NDB Artists filter by Edition year.

4.1. Baseline Model – Popular Tracks

To recommend popular items is usually taken as a baseline to measure the effectiveness of a recommender system.

As the rating matrix has only binary data [0, 1] meaning a track is present or not at a specific playlist, ratings are implicit and the rating media is equal for every track so popular baseline takes into account most rated tracks, tracks appearing with highest frequency inside playlists.

This model does not take into account which songs are inside the playlist. All users (playlists) receive same recommendations, same playlist with most popular tracks. Later we will see the evaluation but as expected the results are very low because it is only effective with playlists that contain very famous songs.

rank	title	tid
1	Riptide - Vance Joy	361
2	Ophelia - The Lumineers	1755
3	Don't Stop Believin' - Journey	1579
4	Bohemian Rhapsody - Remastered 2011 - Queen	586
5	Midnight City - M83	7278
6	Ho Hey - The Lumineers	55
7	September - Earth, Wind & Fire	1078
8	Little Talks - Of Monsters and Men	1889
9	Mr. Brightside - The Killers	4055
10	Brown Eyed Girl - Van Morrison	621
11	You Make My Dreams - Remastered - Daryl Hall &...	2358
12	Hotel California - Remastered - Eagles	1077
13	Sweet Home Alabama - Lynyrd Skynyrd	2270
14	Closer - The Chainsmokers	856
15	Thinking Out Loud - Ed Sheeran	33
16	Piano Man - Billy Joel	1744
17	Home - Edward Sharpe & The Magnetic Zeros	871
18	Stolen Dance - Milky Chance	1044
19	Shut Up and Dance - WALK THE MOON	816
20	Africa - Toto	1588
21	Take Me To Church - Hozier	2129
22	Come On Eileen - Dexys Midnight Runners	2583
23	Sweet Child O' Mine - Guns N' Roses	1562
24	Jessie's Girl - Rick Springfield	3792
25	Under Pressure - Remastered 2011 - Queen	2593
26	Uptown Funk - Mark Ronson	2299
27	Ain't No Mountain High Enough - Marvin Gaye	594
28	Rich Girl - Daryl Hall & John Oates	584
29	Roses - The Chainsmokers	451
30	Have You Ever Seen The Rain? - Creedence Clear...	3934
31	Take On Me - a-ha	1062
32	Bang Bang - Jessie J	2458
33	Breezeblocks - alt-J	3944
34	Budapest - George Ezra	357
35	Skinny Love - Bon Iver	3137
36	Let It Go - James Bay	1672
37	Electric Feel - MGMT	2633
38	Pumped Up Kicks - Foster The People	1005
39	Tiny Dancer - Elton John	2833
40	Stubborn Love - The Lumineers	1761
41	Tongue Tied - Grouplove	3019
42	Sweater Weather - The Neighbourhood	4128
43	Sweet Caroline - Neil Diamond	13
44	Let Her Go - Passenger	367
45	More Than a Feeling - Boston	3904
46	Shape of You - Ed Sheeran	878
47	Rocket Man (I Think It's Going To Be A Long Lo...	3961
48	Wonderwall - Remastered - Oasis	1667
49	Electric Love - BØRNS	2168
50	Santeria - Sublime	1500

Figure 12: Popularity Model – Top 50 tracks – subset_mpdNDB (pid= 157638)

If we analyze the list of the most popular tracks we find a mixture of themes from very different decades, genres and artists. It is a playlist that apart from having in common the popularity we cannot glimpse any other types of characteristics to relate some songs to others.

4.2. Collaborative Filtering - K-Nearest Neighbors

kNN is a machine learning algorithm to find clusters of similar users (playlists) based on common item ratings (tracks), and make predictions using the average rating of top-k nearest neighbors.

The data structure to implement kNN algorithm uses a co-occurrence matrix, where each user (playlist) is a row and items (tracks) are columns of the matrix and cell value is the rate the user gives to a specific item.

In chapter 2 of the Memoria, Dataset Description, there is a section called Sparse Matrix where it is explained how the matrix has been constructed from the input data obtained from MPD dataset.

Preparing Data:

- **Dimension Reduction**

As we have seen in the Co-occurrence Matrix analysis, our data is extremely sparse. In order to save memory and not run into “MemoryError” during model training and save some resources, I´m going to apply some filtering to our dataset to reduce dimension.

The first filter applied is to reduce the number of tracks by keeping only those which have at least 10 ratings. By dropping out unpopular tracks I´ll try to improve recommendation quality when training the model.

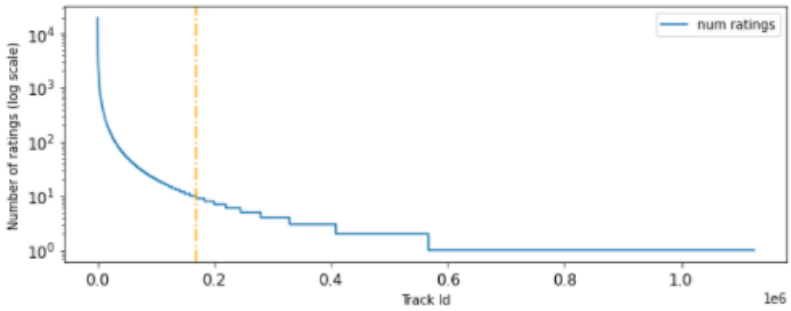


Figure 13: Filter by track rating: threshold 10 - MPD_NDB

The second filter applies a threshold to the length of a playlist, keeping those having a minimum of 50 tracks on it. In this way, I try to ensure that when I go to execute the Train-Test split of the dataset I have enough tracks in the Test set that I will use to evaluate the models.

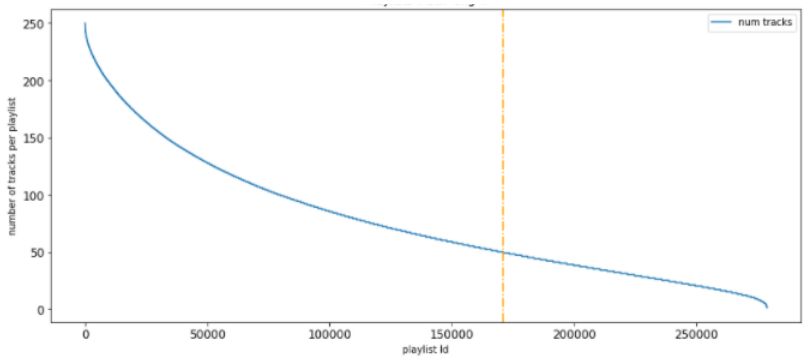


Figure 14: Filter by playlist length: threshold 50 – MPD_NDB

After applying both filters, new data dimensions are:

Dataset	Playlists	Tracks	Ratings
NDB_MPD	279.032	1.124.914	23.545.215
subset_mpdNDB	171.069	170.117	18.378.358

Table 7: Dataset dimension reduction

Another step is needed before starting to build a model. As some playlists and tracks index has been deleted, before reset indexes and save new sparse matrix containing subset_mpdNDB dataset in csr format, a dictionary to map old index of playlists and another one for old track indexes is also saved.

- ### Train-Test Split

To prepare the Train and Test sets, I convert csr sparse matrix from subset_mpdNDB back into dataframe format because I want the split to be done taking into account a fixed percentage of each playlist, train set having a 70% and test set 30% and stratify parameter is not available when data input is in sparse matrix format.

Another important thing to consider before resetting the indexes and save new train and test sets (dataframe and sparse matrix format) is to order rows according to their original index so as not to lose original rank of tracks within its playlist that will be needed to evaluate metrics when building the model.

- ### Fitting the Model

I initialize the Nearest Neighbors model with 25 *neighbors* and specifying distance metric *cosine* so the model will measure similarity between 2 playlists vectors by using cosine similarity.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

where A_i and B_i are components of vector A and B respectively.

As I´m fitting the model with a sparse matrix by default the algorithm used to compute nearest neighbors is *brute* (use unsupervised algorithms with sklearn.neighbors).

- ## Making Recommendation

One I have found nearest neighbors playlists from our input playlist, I still need to refine the prediction function because what we are looking for is to return a playlist of topk where k is the number of recommended tracks to return by the model and now I have as many playlists as n neighbors with whom I have initialized the model. I need to order playlists tracks subset (25 x num. tracks in neighbour playlist).

I review each of the neighbors playlists following the rank of each neighbor and create a dictionary of tracks assigning a weight to each of them according to their playlist rank and excluding input playlist tracks. This way, songs from a closer neighbour will have a higher score but if a song appears in successive playlists it will also gain more weight.

rank	title	tid
1	Sleep Apnea - Beach Fossils	20476
2	Ode To Viceroy - Mac Demarco	27701
3	One More Love Song - Mac Demarco	7269
4	No Woman - Whitney	255
5	Baby (Bonus Track) - Ariel Pink	5878
6	Places - Shlohmo	23302
7	Teen Sex - infinite bisous	76382
8	For Emma - Bon Iver	6842
9	First Day Of My Life - Bright Eyes	393
10	Call Me Up - HOMESHAKE	29994
11	Still Beating - Mac Demarco	7267
12	Nothing Lasts - Bedroom	81792
13	17 - Youth Lagoon	2874
14	Time Moves Slow - BadBadNotGood	4694
15	Rivers And Roads - The Head and the Heart	1757
16	Eugene - Sufjan Stevens	10449
17	I Love You So - The Walters	18642
18	Moonlight On The River - Mac Demarco	8577
19	Stuck on the puzzle - Alex Turner	286
20	Don't Know Why - Norah Jones	383
21	363N63 - King Krule	1277
22	Transatlanticism - Death Cab for Cutie	8157
23	Watching Him Fade Away - Mac Demarco	8580
24	I Think Ur A Contra - Vampire Weekend	11644
25	The Suburbs - Mr Little Jeans	14615
26	Imagine - 2010 - Remaster - John Lennon	15430
27	Bluish - Animal Collective	16081
28	Beach Monster - STRFKR	22264
29	The Greatest - Cat Power	29288
30	In Your Eyes - BadBadNotGood	30914
31	Ceilings - Local Natives	31862
32	BUS RIDE - KAYTRANADA	33732
33	Still Together - Mac Demarco	37259
34	The Flight - Toro y Moi	37262
35	The Ice Is Getting Thinner - Death Cab for Cutie	43297
36	Worms - Youth Lagoon	45449
37	Organ Ronald Donkey Water - Mac Demarco	45749
38	The Time Has Come Again - The Last Shadow Puppets	49034
39	Raspberry Cane - Youth Lagoon	59127
40	Choking on Flowers - Fox Academy	60876
41	Never Again - Bahamas	62432
42	The Way I Feel Inside - The Zombies	65395
43	Sorry Bro - SALES	79028
44	Pool Party - Julia Jacklin	90243
45	We All Need Something - Bedroom	108439
46	Holy Dances - Beach House	116224
47	Flowers Bloom - High Highs	116260
48	10,000 Claps - Phantogram	134305
49	TV Volume - HOMESHAKE	151792
50	Brother (Live) - Mac Demarco	155494

Figure 15: kNN Model – Top 50 tracks – subset_mpdNDB (pid= 157638)

If we compare this list with the one from popularity model although the input playlist is the same, the recommendations are very different. In this case it does not even include a song from the popular list. This playlist has been built looking for the tracks most similar to its original playlist

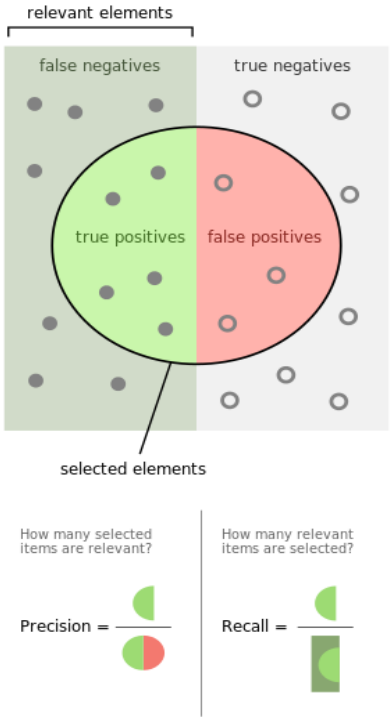
based on the cosine distance between 2 vectors (in our case user-playlist) but at the same time it has a second level ranking tracks from playlist neighbors and reorder, assigning them a specific weight according to their rank playlist.

The main contribution of this model compared to popularity is to include a degree of customization when recommending based on similar songs to those from the input playlist.

- ## Evaluating the Model

A recommender system typically produces an ordered list of recommendations for each user. The goal of an evaluation metric is to measure the relevance or accuracy of a recommendation, quality of a particular ranking of known relevant/non-relevant items.

Not ranked metrics: **precision (P)** (also called positive predictive value) is the fraction of retrieved instances that are relevant and **recall (R)** (also known as sensitivity) is the fraction of relevant instances that are retrieved.



$$\text{recall@}\,N = \frac{\sum_{k=1}^{N} rel(k)}{\sum_{i \in I_u} 1}$$

$$\text{precision@}\,N = \frac{\sum_{k=1}^{N} rel(k)}{N}$$

Figure 16: Precision and Recall. Source: Wikipedia

Relevant recommendations are defined as recommendations of items that the user has rated positively in the test data.

Mean Averaged Precision (map@k) and **Mean Averaged Recall (mar@k or recall@k)** measure **precision** and **recall** calculated by considering only the subset of recommendations from rank 1 through k number. These are the metrics I´ve used to evaluate and compare Popularity and k Nearest Neighbors models.

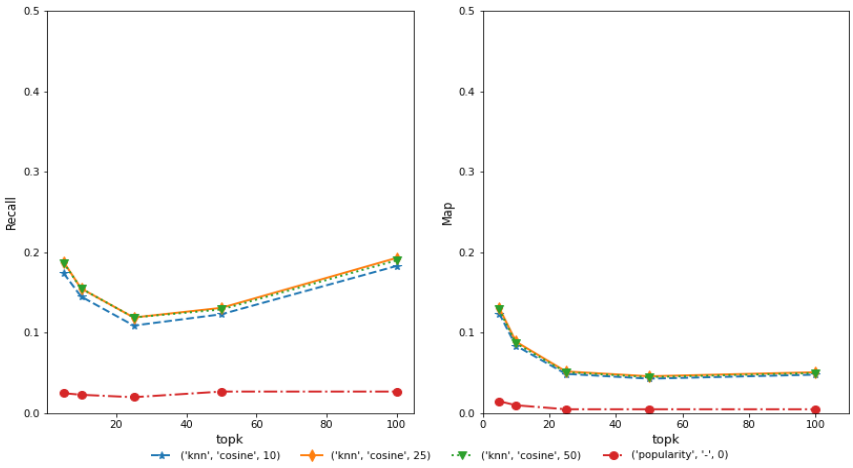


Figure 17: Model Popularity and kNN (10, 25, 50). Precision and Recall at k (5, 10, 20, 50, 100) - subset_mpdNDB

The first aspect to highlight when comparing kNN model with popularity model is that the result of both metrics improves. Recall at k close to 20% and 10% for Precision at k when for popularity model it was practically 0. Although absolute value is low, the relative of one model against the other is significant. The reason why metrics are quite small is probably because there is too many zero values in the rating matrix and many tracks had been rated only once.

Regarding the hyper parameters, number of neighbors when initialize the model, note that increasing number of neighbors reach a point where it stagnates. kNN for 25 neighbors and for 50 neighbors are practically parallel lines, so the final model I will be limited to 25 neighbors.

With respect to topk (number of recommendations), the trend of the result of the metrics is to decrease as the number of predictions increases: 5, 10, 25. It only starts to recover by doubling topk to 50 even 100 recommendations but only recovering returning to initial level, it does not improve metrics any higher.

Comparing Recall at k vs Precision at k, note Mean Averaged Recall is always better than Mean Averaged Precision but it is logical because precision takes into account not only relevant items retrieved but also their rank within the list of recommendations which makes its score lower.

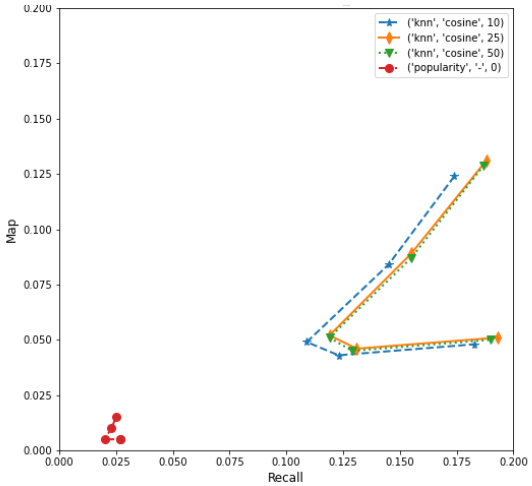


Figure 18: Precision vs Recall curve at k (5, 10, 20, 50, 100) - subset_mpdNDB

5. Conclusions

As a first approximation to Recommender Systems**, K Nearest Neighbors** model offers a degree of personalization in the recommendations returned, especially when it is compared with **Popularity** as base model.

Another point in its favour is that it allows you to use complex data structure such as co-ocurrence matrix and to use different forms of representation of sparse data (for matrix creation I´ve used a Dictionary of Keys – DOK matrix and for training the model, I´ve used Compresses Sparse Row - CSR) to be able to apply some of the computational advantages of linear algebra, deal with a big volume of data and memory management.

Another fact that has caught my attention is that k Nearest Neighbors Model takes more time in inferring predictions than training the model. The model is trained immediately after initialize the model with chosen hyper parameters (metric distance="cosine", n_neighbors=25 and algorithm="brute") and it´s ready to predict.

Regarding the final test, when K Nearest Neighbors model has been executed with random playlists uniquely build with NDB artists tracks, we can observe an uncertain result. This is not only explained by the sparsity data in the matrix but also because those playlist are completely new for the model and the mixture of artists, music genres and track release year are really dissimilar, which means that the elements of the playlist are very dispersed among themselves, so when applying the model, it is difficult to find nearby vectors and the final playlist can be highly influenced by the most famous artists and those artists from minority genres or less well-known are made invisible.

If we analyze the prediction shown at figure 19 in chapter 6, input playlist was 2019 Edition. That year NDB artists playing with bigger presence in MPD_NDB dataset were: Juanes, Daryl Hall & John Oates, The Roots, Keane and Rodger Hodgson from Supertramp and artists with lower presence were Zahara, Rufus T Firefly, Twanguero, Tomatito, Los Planetas, Andrea Motis ... all of them national artists. The predictions we are getting are a mixture of latin singer-songwriters; Juanes, Julieta Venegas, El Chombo – Panamanian DJ, Vicente Fernández – rancheras singer and classic bands/songwriters from the 70s-80s; Paul Simon, Billy Joel, Elton John, Talking Heads that fit to a certain extent with the playlist received but the challenge would be get a better balance representation of the artists and songs received because the popular ones ends up making the rest invisible.

NDB is characterized by programming musical genres that do not usually coexist. This has resulted in a really challenging playlist to test the model and have led me to experience first-hand all the problems that a recommendation system has to deal with.

The main challenges faced by a collaborative filtering based on **K Nearest Neighbors** model:

- Cold start, it occurs when the new user is logged into the system. Due to lack of ratings from new users in the CF, it is impossible to calculate the similarity between new users and other users and thus the system cannot make accurate recommendations. This problem also affects when new items were added to the catalogue and have either none or few ratings.
- Data sparsity, given the high number of zero entries in the matrix (we have already seen that sparsity level in our matrix was almost 1), this affects the calculation of the distance between vectors, similar or opposite, are both pretty large.
- Popular bias, as we have seen in the figure 11, Rating Frequency of Tracks, it follows a "Long Tail" distribution. Few items account for the vast majority of ratings in the "head" while vast majority of items are placed in the "tail".
- Computational bottleneck in the distance calculation. For each pending recommendation, a distance calculation between all training-testing user pairs is performed.
- Scalability, when data grows with new users and new items handling training data required is too large to fit in memory.

To be continued…

The work I have done builds the foundations to be able to continue deepening in the field of Recommender Systems.

- The next step that I would take would be redefining prediction function. I would try to extend k Nearest Neighbors analysis in addition to playlists among their total group of songs trying to find some kind of compensation for more unknown songs in case there is a significant weight within source playlist.
- Another aspect to take into account is the quality of input data since the representation of NDB artists within the original dataset is very scarce. MPD dataset includes public playlists created by US Spotify users, many of the artists playing at NDB Festival are local artist's not so well-known for US audience.
  It would be appropriate to look for an alternative to include playlists that follow these artists or are representative of the musical genres programmed at the Festival.
- To escape the popularity bias as well to improve the efficiency of the model as it scales with new playlists and songs, the most appropriate thing would be to switch to a Matrix Factorization based model like Singular Value Decomposition or SVD. Matrix factorization can be used to discover a set of underlying factors that seem to be driving the ratings that are responsible for the patterns that you can see within the matrix. This method is known as latent factor analysis; features associated with the playlist should match with features associated with their tracks. If we can discover those latent features within a playlists (like artists, album, genre, realise year, audio features...), it would be possible to get accurate recommendations as we will provide a more personalized experience.

- To facilitate the implementation of a new model based on Matrix Factorization it would be better to adopt the use of the Surprise library based on Python Scikit designed for building and analyzing recommender systems.

6. NDB Recommender App

README file include the instructions to run and test NDB Recommender App.

There is also a notebook (NDB_Recommend.ipynb) to test K Nearest Neighbor model but new playlist cannot be listened from there.

Take into account that now kNN model is feed with new playlists and the model is initialized using complete rating matrix, *matrix_playlistTrackRating_ndb.npz* instead of the subset I´ve been using for testing.

Another thing to keep in mind is that in order to play the playlist generated by the model with the Spotify web player, I am using my spotify user to login and replace one of my playlists called 'NDB_Recommends' whose playlistID I have added as an environment parameter.

Environment variables needed [SPOTIPY_CLIENT_ID; SPOTIPY_CLIENT_SECRET; SPOTIPY_REDIRECT_URI; SPOTIPY_USER; PLAYLIST_ID].

User Manual

The front-end is divided into 2 sections, a sidebar that includes input variables and the content of the app where the playlist will be displayed.

Figure 19: NDB Recommender App - Sidebar

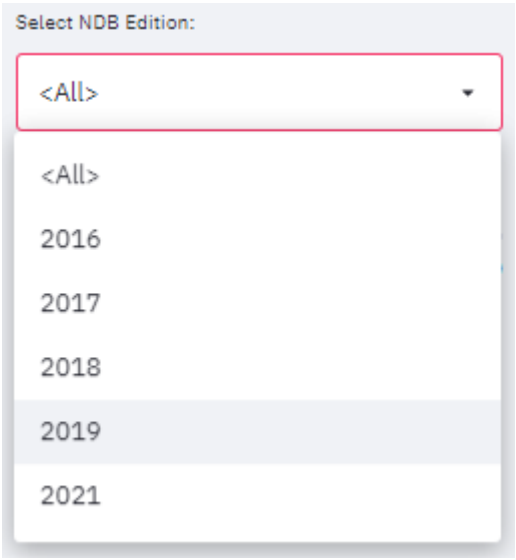


Figure 20: NDB Recommender App – Selection NDB Edition

The user selects one edition year (2016-2021 or All) and click to choose the number of recommendations to include in the new playlist, from 0 to 50 in steps of 10 or 20 by default.

When pressing Recommend button, year edition suggested playlist will be shown below Recommend button and the content of the App will show the playlist with the recommendations generated by the model.

New playlist can be listen to at Spotify player on top by clicking Play and move Next or Previous Track. And recommended titles of the tracks can be read below Spotify player.

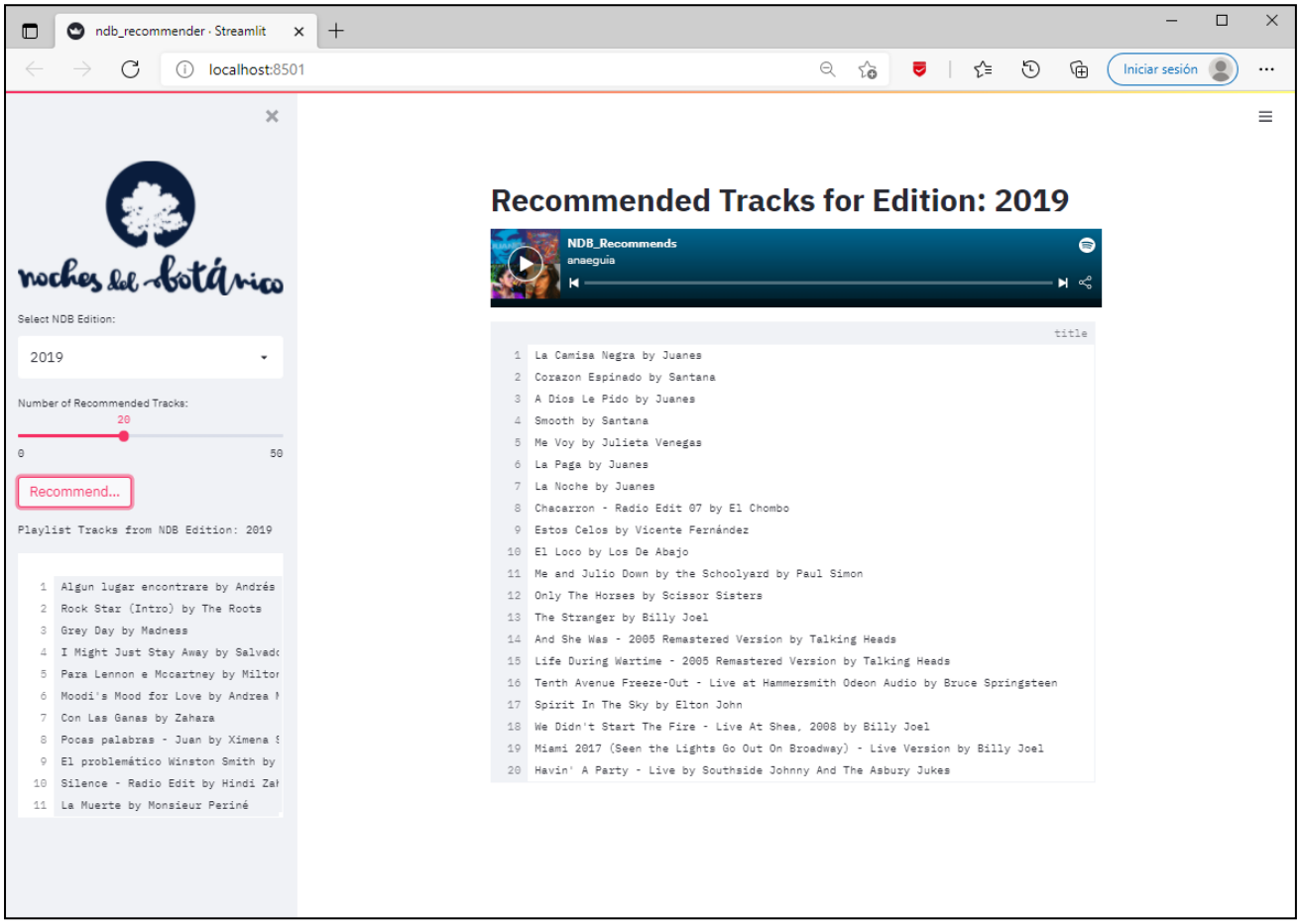


Figure 21: NDB Recommender App Front-end