Bewy, a Music Group Recommender System

LEOPAUL BOESINGER, MARC EGLI, and YASSINE KHALFI

In this report we present an innovative way of building a recommender system that generates a playlist for a group of users. Using the last.fm dataset, we build three recommenders. The first one uses collaborative recommendation technique to generate a playlist based on other users listening habits. The second recommender is content-based, and uses track attributes to recommend tracks similar to the users' tastes. The last recommender uses heuristics and looks for common artists,track, and genres among the users in the group to build a list of tracks fitting most users. We then provide some useful insight for building group recommenders.

1 INTRODUCTION AND RESEARCH QUESTIONS

Did you listen to music today ? The answer is probably yes. On average we listen to 2.5 hours of music a day, there are few things you spend more time on in your day. Music can be enjoyed alone, but it can also be enjoyed in a group, with friends or family at a party or on a trip.

If when you listen to music on your own, it is quite easy for you to know what to listen to because it depends on your taste, when you have to decide what music to listen to for a group of people, the task becomes more difficult because it is not easy to find the songs that are likely to please the whole group, especially when this group is heterogeneous. This problem is the one we are going to try to solve with Bewy.

Based on the listening history of each user, Bewy builds a playlist that seeks to appeal to as many people as possible.In the following sections, we will explain the different stages of this process and how different techniques allow us to create this playlist brick by brick.

In this paper, we will attempt to answer the following Research Questions :

- Can we determine the propensity for a user to like a track, given the user's history of listening ?
- Given track ratings for users, how to compute track ratings for the whole group ?
- How to maximize the robustness of a recommender system, even when users don't have much listening history ?

2 DATA DESCRIPTION AND EXPLORATORY ANALYSIS

In our study, we model the group, as a sample of users from the "Last.fm Dataset - 1K users", which tracks the full listening history of 1'000 users. This dataset contains a row for each track that a specific user listened to, alongside the timestamp of the listening. We also queried Spotify's API using the author and the track name from the last.fm dataset, to obtain information about the tracks themselves. Some of these are : the Danceability, Valence, Energy or Tempo.

2.1 Last.fm dataset

We cleaned the provided dataset by using a regex to make sure that only rows with a valid *user-id*, *artist-id* and *track-id* are kept. All the user ids are of the form "user_" followed by six unique digits for an unique user whereas the other ids have the form of a SHA-1 hash. With this we went from nearly 20 Million rows to 17 Million. When matching the regex we also made sure that if there was a NaN entry we also removed the row.

By performing a deeper analysis of data, more precisely of the timestamps, we concluded that it was better to only keep to most recent data entries as we want to recommend tracks that are relevant in the near future for every user. This assumption comes from the fact that users are more likely to prefer the music they listen to most recently than music they listened to, a few years ago.

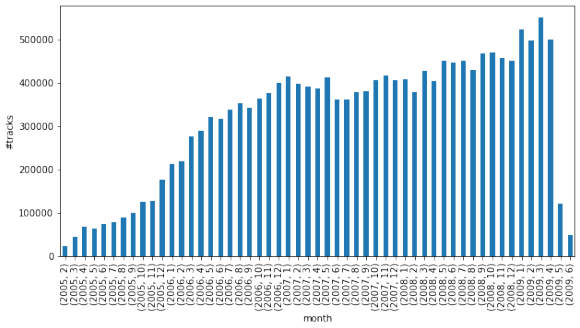

Fig. 1. Number of tracks per month

We therefore filter out a big part of our data and only keep the last 6 months where we had enough data (the last months were not carefully scrapped), from November 2008 to April 2009.

Another important point is to keep only tracks that have enough listens per users as we will query Spotify's API for each track. This will ensure that we will find most of them in their API. Furthermore, it allows us to reduce the computations that are very long, and also to avoid recommending unpopular music.

2.2 Spotify's features

The Last.fm dataset provides us with data about users/track pairs but doesn't give us any additional information about the track except its name. We decided to use Spotify's API to enrich our data with more information about each track. The additional data obtained for each track are attributes describing the tracks. For example, we get the danceability score for each track which indicates the potential to dance to it or even a set of genres corresponding to the author of the track (because Spotify doesn't provide a genre per track but only genres per artist).

We then combine the cleaned and filtered 1k dataset with the result of the query to form our final dataset that will be used to train

a recommender system in order to compute a rating for each user, for the songs.

3 THE PROPOSED APPROACH

There are several types of recommendation systems for music. There are recommendation systems that rely on a user's history to suggest what they can listen to next, systems that use users' histories to see what similar people are listening to and make suggestions based on this information, and recommendation systems that analyse the features of different music to find similar songs.

In our case, we have chosen to use several approaches. Each recommender system has its advantages and disadvantages and by combining them we can get the best out of each system, knowing that depending on the users and the data available to us for a group of users, one type of recommender may be more interesting than another.

Our approach for creating the playlist has been divided in three parts, the collaborative, content-based and heuristic recommender.

The group of users can then select the flavour of recommending which they prefer. We have chosen to use multiple recommenders for several reasons :

- robustness of results if we have different recommenders (ex : one may perform worse if we don't have much information about users, like collaborative)
- incorporate more stochasticity, so even with the same amount of data about users, can generate many different playlists
- allow people to discover music, through content based and heuristic recommenders

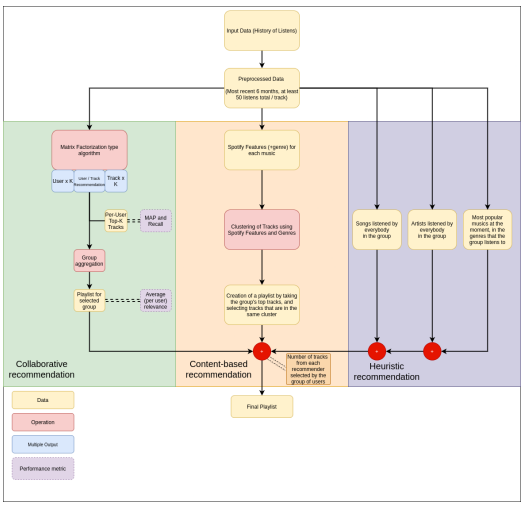

Fig. 2. Diagram of our approach

3.1 Collaborative Recommender

For the collaborative part, the aim is to use the user community to find music that each user is likely to like, based on their listening experience.

To do this, several steps are necessary, the first is to switch from a number of listens to a rating. One of the challenges with a music recommender as in this case is to go from implicit feedback (has the user listened to the music, if so how many times) to a score to be able to have a preference system. In our case we calculate a score based on the number of times the user listens to the music. We have experimented different methods to calculate the score of the songs:

- Linear binning : We take the standardized ratings per user, and linearly bin them from 1 to 10.
- Quartile binning : We take the standardized ratings per user, and bin them such that we have about as many tracks in each bin from 1 to 10. For example we have as many songs that have a score of 2 as songs that have a score of 8 for example.
- Quartile binning with item mean removed : We start from the Quartile binning, remove for each track, the average rating it had, and linearly bin again from 1 to 10. This rating takes into account the number of plays of the user for whom the rating is computed, but also the other users in comparison.

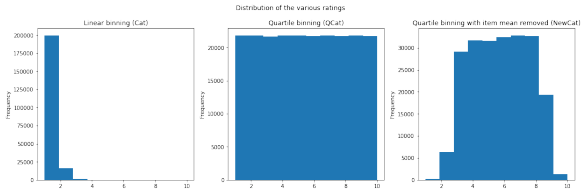

Fig. 3. Distribution of ratings using the different binnings

From Figure [3] we see that Linear Binning is very biased towards a rating of 1, whereas the Quartile Binning is evenly distributed, and the Quartile binning with item mean removed seems to follow a normal distribution.

We used these rating techniques for a very long time before moving to another technique for calculating the rating at the end of the project, BM25 [Amati 2009], which we will discuss in the next section.

Once we have the ratings, the second step is to find models to make recommendations to our users. Several machine learning algorithms can be used for our collaborative systems. We can use matrix factorization, clustering, or even KNN-based algorithms. In our quest to find the best result for our recommendation system, we have tried these different algorithms (the results are available in the next section). Coming back to matrix factorization, which is the method we ended up using after our tests, matrix factorization allows us to represent music and users in the same lower dimensional space. This model allows us to move from the user-song matrix which is very sparse (because we only listen to a few songs out of the total songs available) to two dense matrices with reduced dimensions with latent features. While those features are not human interpretable, the embedding gives very interesting information, on how a user is defined, and how a track is defined in this lower dimension space. This allows us to have a more scalable and compact model, but also to detect which of our items, as well as which of our users, are similar.

Once our model has been created, we can then use it to find out according to the model what rating would be given to each item (music in this case), by doing this for all the users in our group we can then determine which music add to our playlist. When we try to add a song to the playlist, we take into account two parameters:

- Average rating, which allows us to see the average score of the track among the users in the group.

$$rat(\mathcal{G}, i) = \frac{1}{|\mathcal{G}|} \sum rating(u, i)$$

- Average Pair-wise Disagreement, which allows us to see if the song has a similar rating among all the users or if the users have different ratings.

$$dis(\mathcal{G}, i) = \frac{2}{|\mathcal{G}|(|\mathcal{G}| - 1)} \sum_{u \neq v} |rating(u, i) - rating(v, i)|$$

These two parameters allow us to refine the choice of music we add. If we want to maximise the average rating, we will have a system that adds the music with the best average score even if the latter may have a very low score among some of the users. Such a scoring function has been used widely in the group recommender systems literature, as in [Amer-Yahia et al. 2009].If we want to maximise the average pairwise agreement (1 minus the disagreement), we have the opposite case where we will privilege the music where the similarity score between users is the highest but where the average score is not necessarily the highest. To take both criteria into account, we have a γ parameter that allows the user to select the importance to be given to each of these criteria. Using a γ close to zero will give more weight to minimize the disagreement, while a γ close to 1 will try to maximize the average rating.

$$rating(|\mathcal{G}|, i) = \gamma * rat(|\mathcal{G}|, i) + (1 - \gamma) * (1 - dis(|\mathcal{G}|, i)), \gamma \in [0, 1]$$

Using this scoring function for the group, we can now simply select the tracks that maximize it to get a playlist !

3.2 Content-based Recommender

Each music you listen to can be represented by a list of features. By features, we are talking about liveness, speechiness, danceability, tempo, valence, etc. Each musical genre can also be defined by these features, each genre will have specific values for these different features. For example, rap and electro will not have the same tempo nor the same speechiness or the same liveness. Our content based recommender's role will be to find the most similar music to the one you like the most, based on the principle that if you love a song, you are more likely to like the music that is "close" to it.

As explained in the previous section, the last.fm dataset does not contain any information about the music itself apart from the artist and title, so once the information about the features is retrieved from Spotify, we can move on to creating a model.

The first step in creating our model was to transform our features so that they could be used correctly. Indeed, since we are working with mixed features and the final goal is to compute distances between songs, we started by transforming non-numerical variables into numerical variables and scaling our different features. Once all this work is done, we can very simply calculate for a given song its "distance" from the others using a Euclidean distance. For example, this is what we get for the song "Not if You Were The Last Junkie On Earth".

	artist-name	track-name	genres
0	The Dandy Warhols	Not If You Were The Last Junkie On Earth	{'alternative rock', 'dance-punk', 'indie rock...
1719	The Dandy Warhols	Get Off	{'alternative rock', 'dance-punk', 'indie rock...
5766	The Dandy Warhols	We Used To Be Friends	{'alternative rock', 'dance-punk', 'indie rock...
432	The Dandy Warhols	Minnesoter	{'alternative rock', 'dance-punk', 'indie rock...
5859	The Dandy Warhols	Bohemian Like You	{'alternative rock', 'dance-punk', 'indie rock...
5926	Kaiser Chiefs	Always Happens Like That	{'modern rock', 'alternative rock', 'dance-pun...
4910	Kaiser Chiefs	Love'S Not A Competition (But I'M Winning)	{'modern rock', 'alternative rock', 'dance-pun...
510	Kaiser Chiefs	Never Miss A Beat	{'modern rock', 'alternative rock', 'dance-pun...
2826	Kaiser Chiefs	Ruby	{'modern rock', 'alternative rock', 'dance-pun...
6824	Kaiser Chiefs	Na Na Na Na Naa	{'modern rock', 'alternative rock', 'dance-pun...
7242	Kaiser Chiefs	Like It Too Much	{'modern rock', 'alternative rock', 'dance-pun...

Fig. 4. Nearest tracks for "Not if You Were The Last Junkie On Earth"

Then, once we had created our model allowing us to have distances between the different songs, we performed clustering on our data. Clustering allows us to check our results to see if the representation of songs through features makes sense and also to have variety in the proposals by recommending other music from the same cluster.

3.3 Heuristic Recommender

When we want to create a common playlist for a group of people, a basic idea would be to see what artists, music or genres we have in common. To know which artist, track or genre we should consider, we use a voting system. A song gets a vote if a user has listened to it. An artist gets a vote from a user, if the user has listened to one of his songs. A genre gets a vote from a user, if the user has listened to a song whose genre is the one in question. So for genres, artists and tracks, the maximum number of votes they can have is the number of users in the group. This voting system is used in each part to find when needed the most popular artists, tracks or genres in the group.

3.3.1 Recommendation based on common songs/artists.

In this heuristic part, we try to find the most popular artists in the group through our voting system. Once we have these artists with the voting system, we can take songs from these artists to add songs to the playlist of the group.
To illustrate what this can give us in terms of results, we can take as an example a group of 4 users for whom we would like to create a playlist. If we run our algorithm and voting system to find the most popular artists in this group of 4 users, we end up with the following results.

```
Camera Obscura, Amy Winehouse, Mgmt, Beck, Bloc Party, Sufjan Stevens, Michael Jackson, Iron & Wine, Arctic Monkeys, Yeah Yeah Yeahs,
The Shins, Daft Punk, Damien Rice, Muse, Abba, Radiohead, Hot Chip, Belle And Sebastian,
```

Fig. 5. Common artists

We find a list of artists that most users listened to, it is not necessarily the same type of music but it is a base that will allow us to put in the playlist songs from artists that are popular in the group.

If instead of considering the artists we want to directly consider the songs, to directly have items to add to the playlist we can also do it. If we do it for the same 4 users as in the previous step, we end up with the following results:

Fig. 6. Common songs

This part of our recommender allows us to add some songs to our playlist which are relevant for the users.

3.3.2 Recommendation based on popular songs.

Another way to grow our playlist is to add popular music. Here we start from the principle that a song is considered popular if it appeals to a large number of people. So we suggest popular music to users in their playlist because we think they will like it. Moreover we bring a nuance, we add popular music in the genres that our users listen to, so if our users only listen to classical music it is useless to propose them pop music. A first step is to find the most popular genres among all the users of our group through a voting system, if we look for theses genres for the same group as in the previous part, we find the following result:

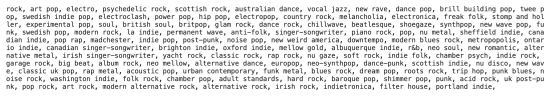

Fig. 7. Common genres

So we find a large number of genres with maximum popularity, because in our data each song can have up to ten genres. Once we have found the genres listened to by all the users, we can look at the most popular music of these genres, and we find the following results:

	musicbrainz-track-id	artist-name	track-name
1392	2b9241b3-a7c4-4866-a15d-17344cf44541	Britney Spears	Circus
1196	258bc802-8111-4d16-b081-fae0ea6aecba	Britney Spears	If U Seek Amy
173	051caac1-1f67-4733-80b5-62cb32660daa	Kings Of Leon	Sex On Fire
6188	c697b759-2ef6-43bb-a97a-2c56409abade	Franz Ferdinand	Ulysses
2853	5a536521-64ae-4a57-8afa-2a4235bc8841	Britney Spears	Womanizer

Fig. 8. Popular songs among common genres

We end up with very popular music that has a great chance of being loved by our users.

4 EXPERIMENTAL EVALUATION

Our three flavour approach has the advantage that we are able to obtain robust recommendations based on what the user is interested in. However, in order to prove this robustness, we have to evaluate each recommender system on its own, since they are all very different. In some instances, it doesn't make much sense to evaluate the system using offline experiments, so we decided to favour manual verification where we couldn't find satisfactory metrics.

4.1 Choosing Relevant metrics

Selecting a Metric is crucial to determine the efficiency of a Recommender System. While one could think that a simple RMSE could do the trick, it is often considered as a lackluster metric, which doesn't truly capture how well a Recommender System functions. As described in [Bellogín and Said 2018], there are many metrics other than RMSE which can be used to determine the accuracy of a Recommender System. One issue with RMSE is that it computes the overall error, but doesn't make any difference between the predicted-top items and the rest of the items, where recommender systems should be able to have a very precise recommendation for what should be top items, and maybe less performing otherwise.

The notion of relevance is very important in Recommender systems, which is one of the reasons why we decided to use decision-support metrics when evaluating our systems : We want the model to return relevant items for a user, and to avoid irrelevant ones. The metrics we decided to keep are : Precision, Recall and Mean Average Precision (MAP). The MAP metric is another important metric here : not only do we want items to be relevant (captured by precision), we want the model to be able to decide which item is the most relevant inside its top propositions, which is obtained from the MAP.

After having selected our metrics, we need to define a notion of relevance. We decided to use two different definitions, which are used at two different granularities : The fine-grained, track based relevance, where we consider a track relevant if it was listened at some point by the user. The coarse-grained, artist based relevance, where we consider a track relevant if the author of the track was listened at some point by the user.

These two relevance definitions are very similar, and in fact, we know that the artist-based relevance will always be higher than the track-based relevance (since if the user listened to the track, it listened to some track made by the author), but we argue that both definitions are interesting in our case (and we will show it later). Of course, both relevance definitions rely on assumptions. The track-based definition assumes that if some user listens to a song, then it must like it, which is not always true. Furthermore, the artist-based definition makes an even stronger assumption that if some user listens to a song, then it must like any track made by the author of the song. These assumptions are necessary, given the current dataset, if we want to provide a meaningful offline evaluation, and is something that is common among other recommender systems.

Since our decision based metrics need a cut-off (we consider Precision@k), we chose the value of 10. It is simply an educated guess : we consider that if the model is able to obtain a good recommendation over the top-10 tracks, then it should be good enough to generate a playlist for the group. With respect to the track relevance, we also needed the cutoff value to not be too big, otherwise the number of users for which we could remove k tracks (which are the ones we consider relevant), in order to insert them in the test set, would become lower and lower.

In order to use these metrics to tune our models, we performed train / validation / test splitting. We performed a splitting method which is very close to a well-known method described in [Meng et al. 2020] as Temporal User Splitting. We start from the original dataset of (user, track) pairs. For all users who have listened to 50 or more tracks, we take the 20% of the tracks that were listened the most recently and put them in the test set. The number 50 here is selected such that, we have at least 10 tracks which we consider relevant for each user, consistently with the cut-off value of 10 selected before. We use exactly the same method, once again on the initial train set obtained when doing a first split. This makes us able to obtain a train / validation / test split, from this initial train / test split.

4.2 Evaluating the Collaborative Recommender

The collaborative recommender is the most logical one to evaluate, it is the one for which we previously defined the train / test split and the different relevance metrics.

4.2.1 Selection of rating.

We have previously presented different kinds of ratings which were created from the (user, track) pairs (and number of listens). In order to determine which one is the most successful, we chose to compute all the various metrics, using a single model (SVD).

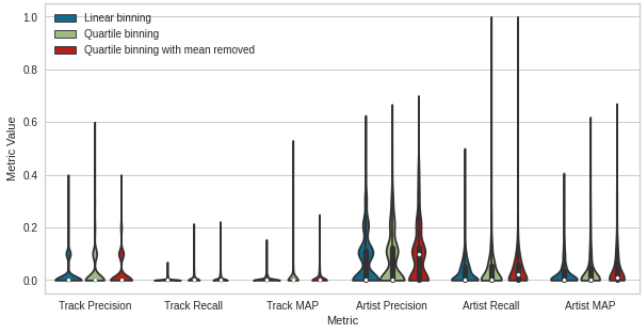

Fig. 9. Decision support metric results using the various ratings

It is hard to say exactly which kind of rating wins here, because the standard deviation is very high. In fact, it is even higher than the mean value for all those metrics. This means that the results vary very much from user to user, and while for some users, we get decent values for the metrics, for most users, we get very poor values. We decided to use a violin plot to show that for most users, it performs badly, but for some other users, the model is able to obtain decent result.

While we acknowledge that some metrics may perform better with a different model, and that it would thus be better to try for every metric and every kind of model, we argue that the difference it makes is actually not significant, since later on, we will select a completely different kind of model, as well as a different kind of metric (see below).

4.2.2 Selection of model.

Our first choice of library was the **scikit-surprise** library, as it features many different kinds of models which we could try. We made the choice of comparing the SVD, Probabilistic Matrix Factorization (PMF) [Salakhutdinov and Mnih 2008], Non-negative Matrix Factorization (NMF) [Luo et al. 2014], User-based KNN with Baseline [Koren 2010], Item-Based KNN with Baseline and CoClustering [George and Merugu 2005]. Results on the validation set are shown in Figure [10]

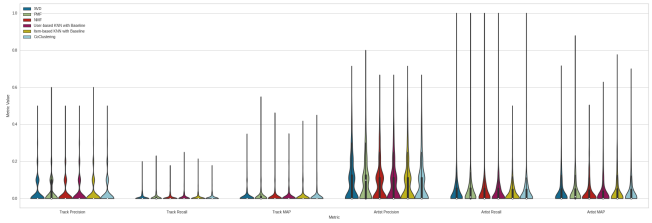

Fig. 10. Decision support metric results using the various models

Once again, we have very poorly performing models, and what was the most peculiar to us was the fact that all models have very similar results in the end (with respect to our metrics). This lead us to try another kind of model, the Implicit Alternating Least Squares. Furthermore, since the creator of the implicit python library had promising results on the last.fm 360K dataset (which is a similar dataset, that features artist listens instead of track listens) using a different kind of rating [Frederickson [n.d.]], we decided to also try it out on the 1K dataset. The rating we now use is to simply keep the number of plays per (user, track) pair, and to consider it like a TF-IDF matrix. We then apply the BM25 weighing scheme and use the final matrix as our rating.

This model got us very promising results, we thus decided to compute a grid-search using as parameters the number of factors of the embedding, and both hyper-parameters of the BM25 rating (K1,b), as we empirically found that these parameters had the most impact on the model.

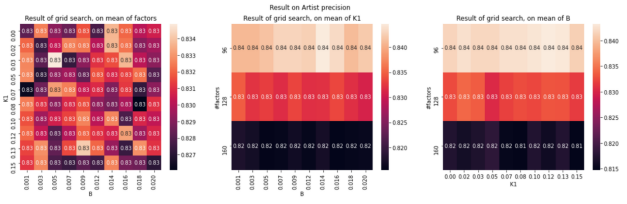

Fig. 11. Result of the gridsearch on ALS

Figure [12] then shows the final results obtained using the parameters selected using the grid search.

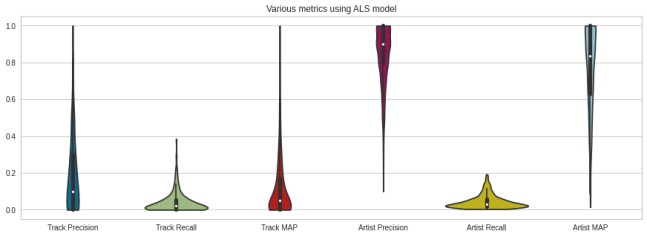

Fig. 12. Decision support metric results using the Implicit ALS model

These results are very convincing : be it with respect to Track or Artist definitions, we get much better results than with the previous models. One thing to note is that the track precision is still not very high, but it is compensated by the artist precision. It may simply be that the model understands well which artists are liked by the user, so it recommends many tracks from those artists, but of course, some of these tracks are not in the test set, so we consider them irrelevant with respect to track precision even if they may in fact be liked by the user. This is why we think it is interesting to look at both metrics.

4.2.3 *User clustering.*

From now on, we will call artist relevance as simply relevance, as we examined only the artist relevance, because we consider that it is easier to compare average artist relevance ranging from 0 to 1 than track relevance which will only be between 0 to 0.1, but both concepts could be used in the same way. In order to get an idea of the impact of the group similarity on the average group relevance (the average of the user relevance for each user in the group), we used dimensionality reduction (using Sklearn's TruncatedSVD) to obtain a lower dimensionality embedding from our model's User matrix, and performed Kmeans clustering on it, so as to obtain a user clustering. The number of clusters was selected using the elbow method on the sum of square errors, as well as the average silhouette score. Figure [13] shows the resulting per cluster silhouette score.

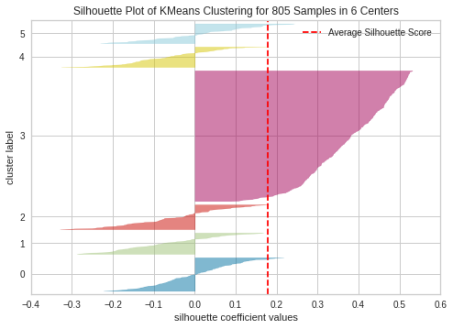

Fig. 13. Decision support metric results using the Implicit ALS model

This clustering is not perfect : one cluster has the majority of samples, and all the other clusters are not very similar, but that was to be expected, the user matrix from the model was not meant to be used for KMeans clustering by default. However, we should still be able to get interesting results using this clustering.

4.2.4 *Impact of the number of users in the group on average relevance.*

In order to understand both the effects of the inter-group similarity between users and the size of the group on the average relevance we devised an experiment. Using three types of sampling (uniform sampling, stratified sampling (as many users from each cluster), single cluster sampling), and various numbers of users, we use the sampler to obtain a group, and compute the average per-user relevance inside the group, for a playlist of 10 songs generated by the recommender. The experiment is repeated 20 times for each (sampler, #users) pair to obtain consistent results. The results are shown in Figure [14].

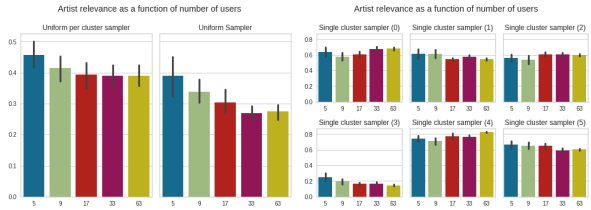

Fig. 14. Average group relevance as a function of the sampling method, and the number of users

These results are interesting : depending on the cluster we may get better or worse results, but what is the most important is that no matter the group size, the average relevance doesn't get much worse as the number of users grows. What can be even more intriguing is that for some clusters it may even look like it grows : This is simply because we use the Average User Relevance in the group, and that with lower group sizes, we could sample users which are very different from the rest of the cluster (as proven by the low similarity score in Figure [13]). Another interesting thing to note is that we only sample a very low number of songs, and yet the relevance is quite high, which makes us think that our relevance metric maybe a bit too easy to satisfy : for example, most people listened to The Beatles, and thus, the average relevance would already be quite high if we only sampled tracks from The Beatles. However, this is a bit counteracted from the fact that all clusters seem to get a decent score, which means that the model still is able to understand what kind of artist to recommend to each group.

In general, when sampling all users from the same cluster, we get the best results (this is expected, similar users can have good recommendations). However, what is a bit more peculiar is that the uniform sampler performs worse than the Uniform per cluster sampler (stratified), which could be explained by the fact that the uniform sampler is fully random, while the stratified one is able to have similarities between users in the same cluster, and sometimes users from different clusters may actually be more similar than users in other clusters. With a better clustering, this would probably have not been the case.

4.2.5 *Impact of γ in average relevance.*

In order to select the γ parameter, we propose to use the single γ which gives us the best relevance on a test group. Recall that a γ closer to 0 acts to Minimize Disagreement, while γ closer to 1 acts to Maximize Average relevance. In a sense, we may expect that this experiment will favor γ closer to 1, since the evaluation metric is the average user relevance, but it is better to test this hypothesis.

For each gamma value, we use the uniform sampler to create 50 groups (groups are always the same) of 5 users.

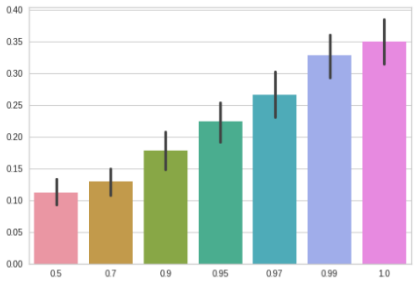

Fig. 15. Average group relevance as a function of gamma

This result makes sense, because we get the maximal average relevance, when we take $\gamma = 1$.

We hoped that some γ value slightly lower than 1 would be better on average, since it would make for a higher artist relevance for the user that is the most far away in taste compared to the rest of the group but it simply doesn't look to be the case.

However, since the Confidence Intervals overlap, it should actually be safe to pick any value of γ between 0.9 and 1, and get meaningful results with respect to the average group relevance. Thus, we conclude that for real-world applications, selecting a γ parameter close to 1 but still slightly lower can be better, depending on the similarity in the music taste of the people in group.

4.3 Evaluating the Content-based Recommender

While we had many ways to evaluate the Collaborative Recommender, the goal of the Content based Recommender is completely different. We aim to provide the group with tracks that they may enjoy, by giving close tracks with respect to the Spotify features.

Our original idea was to simply recommend the closest tracks with respect to a distance computed from the Spotify features, however, since we couldn't find a reasonable evaluation of this method, we instead decided to use K-Means clustering instead. This is great because we can at least compute the inter-cluster silhouette score, and can proudly say that we are confident in our clustering. Figure [16] shows the clusters silhouette scores.

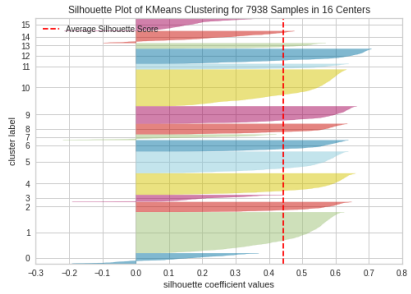

Fig. 16. Track silhouettes

It looks promising ! All clusters are reasonably well-balanced, and have a good similarity score. In order to show the labels of the tracks, we used TSNE to create a 2D embedding of the tracks, and made a scatter plot with the cluster labels, shown in Figure [17].

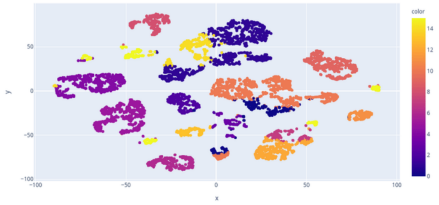

Fig. 17. Track TSNE scatter plot

Once again, this result is convincing.

4.4 Evaluating the Heuristic Recommender

It is even harder to provide evaluation for the heuristic recommender than for the content based recommender. If we use the voting system on artists or tracks, then the selected tracks by the recommender are already relevant, since that's how they are retrieved by the system. If we however use the per-genre recommender, it will provide popular artists in the genre. This means that they will not necessarily be relevant for the group, but we assume that if the group decided to ask for the heuristic recommender, it is because they actually wanted this to happen.

4.5 Evaluating a complete playlist

Since the final playlist is simply the concatenation of all other playlists, and all the recommenders are evaluated on their own, the final playlist could in itself be considered evaluated as well.

Furthermore, we were not able to find a single metric which could be applied to the complete playlist. Instead, we decided to build an interactive demo, for which we could select a group of users, and obtain the complete playlist for that particular group, in order to at least be able to manually check that our results make sense. This does not replace proper online evaluation, but it is the closest we could get to it in this timeframe.

5 DISCUSSION AND IMPLICATIONS

In the collaborative part, the selection of ratings and model concurs with the blog posts of the author of the implicit library : The Implicit Alternating least squares (ALS) with BM25 weighting is a good fit for the last.fm dataset. This seems to be true both for the 360K and 1K dataset. This method performed much better than our first handmade metrics, both with respect to the track relevance and the artist relevance. It is not completely clear if the weighting from BM25 can be interpreted as a propensity for a user to like a track, but it is at least the method that created the most robust collaborative recommender, so we think that it indeed captures the relevant information to obtain this propensity. More generally for recommender systems, this experiment suggests that the BM25 can be used more broadly than in the usual Document Retrieval sense.

The impact of the number of users in a group on the average relevance makes us think that it is indeed possible to build a playlist recommender for a group, even of fairly large size, as long as the users in the group are similar enough. It is even possible, if the group is made of completely different users to find some relevant songs, but the full playlist may not be liked by everyone.

While in our experiments the γ parameter may not shine, because offline evaluation with average user relevance couldn't capture the importance of fairness, manual evaluation using our demo suggested that having this parameter was useful to generate a more balanced playlist. In general for group recommender systems, we think that having a scoring function which is a mix of an average scoring function (such as Least Misery, or simply the Average user rating), and an agreement scoring function is the way to go. [Masthoff 2011] collects the aggregation strategies of some of the most well-known group recommender systems, and they all seem to use some reasonably similar technique. This is also motivated by giving access to the γ parameter to the user, so the group can choose whether they want to maximize the user average, or if they want to make no singular sacrifice.

In the content based part, we saw that creating a clustering based on the tracks is very reasonable, given that the clusters are well balanced, and have decent silhouette scores. This is interesting, but with no online evaluation, it is still hard to determine if the clustering is precise enough to make recommendations that would be enjoyed by users, and we have a similar issue for the heuristic part. However, from the manual inspection performed on our prototype, we think that the combination of the three flavours of recommenders help to give a more balanced playlist. When the collaborative recommender performs worse due to having not much listening history for some users of the group, we can fallback to the two other recommenders, which can be very helpful. In general for group recommender systems, we can therefore suggest that having multiple providers with each their own specificity, and letting the user choose the percentage of content they want from each provider is a good idea.

6 CONCLUSIONS AND FUTURE WORKS

6.1 Conclusions

In this report we presented the pipeline of our recommender system and the design choices we made along it. We first implemented a collaborative recommender system by training a matrix factorization model with the filtered dataset. A scoring aggregator was then presented to convert user-track ratings into group-track ratings. We then created a content-based recommender by taking advantage of the Spotify API, where we could retrieve musical features for the tracks in our dataset, which were then used to build a clustering of tracks. Finally, we created a heuristic recommender that implements some real-world recommendation rules. In that part, we either suggest :

- Tracks that are commonly listened to by many members of the group
- Tracks from artists that are commonly listened to by many members of the group
- Tracks from genres that are commonly listened to by many members of the group

By combining those three recommending systems we were able to propose a new and innovative way of creating custom playlists to a group of users. Even more, providing users with the ability to select the number of songs coming from each recommender system makes our system more versatile, as in real-world situations, the group of users may be sometimes more inclined to listen to tracks that they never listened to before, while sometimes they may just want to listen to their favourite tracks. It also makes it more robust, because in cases where the collaborative recommender may perform worse (if the users in the group don't have many listens), the group can then choose to listen more to the heuristic part, which guarantees that they will be provided with a playlist with common songs, artists or genres.

6.2 Future Works

The above work can be extended by, first of all, reviewing our performance metrics. Indeed, we are right now performing offline evaluation of the collaborative recommender system using precision, recall and MAP, but for the two other recommender systems, we couldn't find interesting relevance metrics, because it is not possible to evaluate if we are making a good never-listened-before recommendation. Therefore, having direct user feedback would be very help full to build such a performance metric. Our model would have to receive a direct feedback from the user. For example a user could rate the proposed playlist, and evolve its recommendation according to the user's incoming feedback.

For the collaborative part, it would have been interesting to try out more designs from the implicit library such as Logistic Matrix Factorization [Johnson 2014], used by Spotify. This is also justified by the fact that both Logistic Matrix Factorization and Implicit ALS employ a confidence metric that measures the propensity for the user to like a particular item, and often perform well in similar cases.

Another improvement that could be made is to do a better feature selection to improve the songs clustering. Right now the clustering uses all the Spotify features, and gives equal weight to each normalized feature, which can be a problem. We could also find a way to incorporate the genres in the clustering of the tracks.

Also, we could evaluate other scoring methods to calculate scores per song/user. In the above presented work we first implemented our own scoring methods that did not perform very well. We then

did some research and decided to use Okapi BM25 which is a well-known, state of the art ranking function. One thing which was very nice in our case is its versatility since we could tune its hyper-parameters to maximize our metrics. However, this is not the only weighing scheme that exists and therefore evaluating a bigger amount of such scoring functions could help us find an even better suited one.

Finally, we were also provided a dataset only containing artists and users which we choose not to use for this project, as we wanted to find a playlist, and not a set of artists to recommend. Nevertheless, we tried to use a very similar implicit ALS model on this dataset and got promising results. One such experiment was that, if we gave it a user that listened mostly to United States rap artists, and to French pop artists, one of the top 10 recommended artists was MC Solaar, which is a French rap artist, which was quite a surprising but nevertheless encouraging result. In the end we could use both datasets, for example by first doing a coarse-grained recommendation of artists and then a fine-grained recommendation for tracks.

ACKNOWLEDGMENTS

We would like to thank all the teaching team for the very profitable feedback provided during the whole semester.

REFERENCES

Giambattista Amati. 2009. *BM25.* Springer US, Boston, MA, 257–260. https://doi.org/10.1007/978-0-387-39940-9_921

Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. 2009. Group Recommendation: Semantics and Efficiency. *Proc. VLDB Endow.* 2, 1 (Aug. 2009), 754–765. https://doi.org/10.14778/1687627.1687713

Alejandro Bellogín and Alan Said. 2018. *Recommender Systems Evaluation.* Springer New York, New York, NY, 2095–2112. https://doi.org/10.1007/978-1-4939-7131-2_110162

Ben Frederickson. [n.d.]. Distance Metrics for Fun and Profit. https://www.benfrederickson.com/distance-metrics/

Thomas George and Srujana Merugu. 2005. A Scalable Collaborative Filtering Framework Based on Co-Clustering. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM '05).* IEEE Computer Society, USA, 625–628. https://doi.org/10.1109/ICDM.2005.14

Christopher C. Johnson. 2014. Logistic Matrix Factorization for Implicit Feedback Data.

Yehuda Koren. 2010. Factor in the Neighbors: Scalable and Accurate Collaborative Filtering. *ACM Trans. Knowl. Discov. Data* 4, 1, Article 1 (Jan. 2010), 24 pages. https://doi.org/10.1145/1644873.1644874

Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An Efficient Non-Negative Matrix-Factorization-Based Approach to Collaborative Filtering for Recommender Systems. *IEEE Transactions on Industrial Informatics* 10, 2 (2014), 1273–1284. https://doi.org/10.1109/TII.2014.2308433

Judith Masthoff. 2011. *Group Recommender Systems: Combining Individual Models.* Springer US, Boston, MA, 677–702. https://doi.org/10.1007/978-0-387-85820-3_21

Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2020. Exploring Data Splitting Strategies for the Evaluation of Recommendation Models. arXiv:2007.13237 [cs.IR]

Ruslan Salakhutdinov and Andriy Mnih. 2008. Probabilistic Matrix Factorization. In *Advances in Neural Information Processing Systems*, Vol. 20.