AUTIFY: **AUT**omatic playlist continuation for Spot**IFY**

Ioanna-Maria Panagou[1] and Ioannis Matskidis[2]

[1]ipanagou@uth.gr
[2]imatskidis@uth.gr

ABSTRACT Nowadays, music streaming services are gaining more and more appeal, since they provide the user with the ability to organize and categorize their music library easily. One of the most compelling features that enhance the user experience is Automatic Playlist Continuation (APC). Dedicated machine and deep learning algorithms can aid the user in curating personalized playlists that align with their music preferences and/or the occasions those playlists were created for, such as studying, road trips etc. Those algorithms provide track recommendations for playlists either relying on the fact that similar playlists can contain the same tracks, a technique known as Collaborative Filtering (CF), or by creating a profile for each playlist based on their content and recommending tracks that are closer to this profile (Content-Based Filtering or CB).

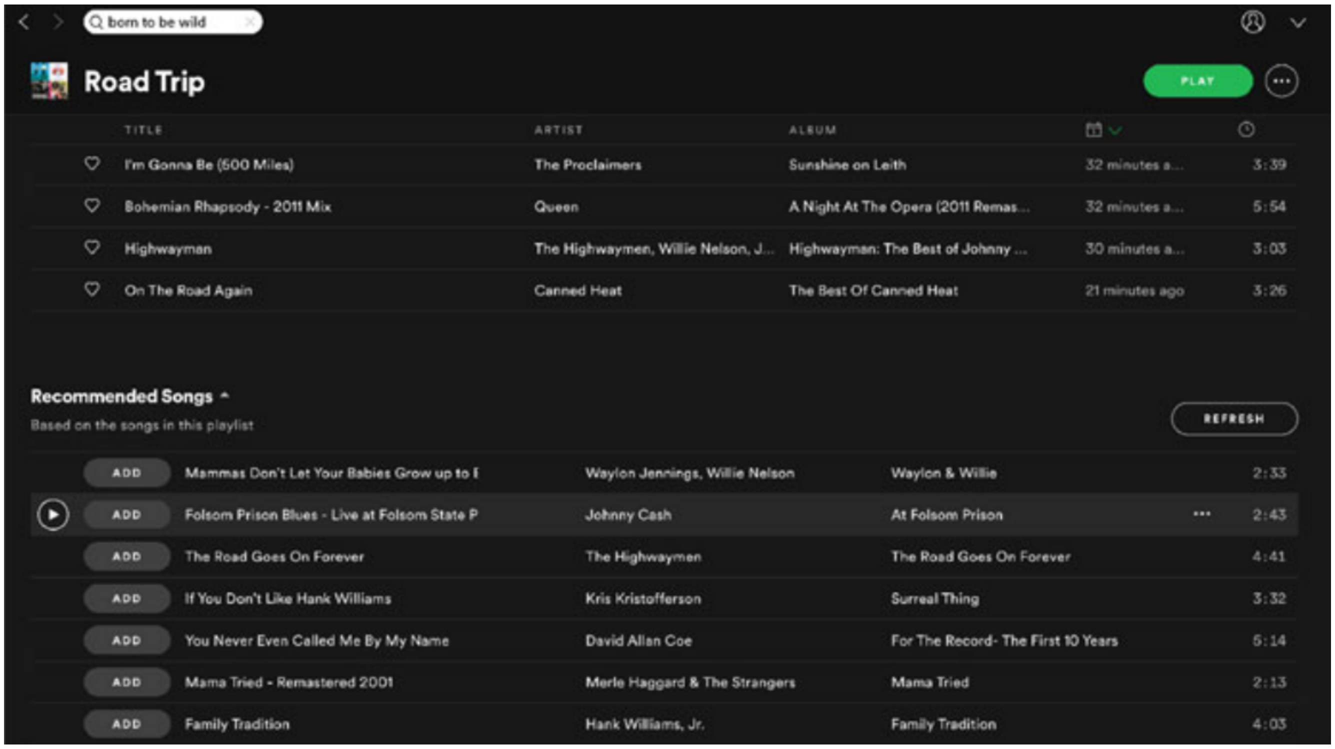
In this project, we explore those 2 types of algorithms, as well as their combination and evaluate them on a real life dataset, the Spotify Million Playlist dataset. We managed to obtain results that align with the state-of-the-art, using very simple models that rely on the principals described above. We also compared those two principals for the task of APC and verified the results provided in the existing literature.

1 INTRODUCTION

The goal of automatic playlist continuation (APC) consists of adding one or more tracks to a playlist in a way that fits the same target characteristics of the original playlist. To state the task more formally, let M be the universe of tracks in the underlying music catalog. Given a playlist P created by a user u, that contains k music tracks $M_P = \{m_{P1}, m_{P2}, \ldots, m_{Pk}\}$, the task is to recommend music tracks from $M - M_P$ to the user for completing the playlist. In addition, each playlist includes some meta-data information, such as title. It should be noted that k can be equal to zero for some playlists, meaning that the user has created the playlist but no music track has yet been added to the playlist, known as the cold-start problem.

An example of this is Spotify's playlist creation application as pictured in 1. Based on tracks that the user has already added to their playlist, further songs are recommended at the bottom for consideration that match the overall tone of the playlist thus far. The songs are sorted according to their relevance; the songs that are predicted by the recommendation algorithm to be the most relevant to the playlist are listed first. Initially, only the 10 most relevant songs are dislayed. If the user wants to explore more options, they can hit the 'Refresh' button and the next 10 most relevant tracks will appear repeatedly until all recommendations are displayed.

For the task of automatic playlist continuation, the input to our algorithm is a set of playlists, which are identified by their `playlist_id` and the output our model has to produce is a set of recommended tracks in descending order of relevance, which are identified by their `track_id` or `track_uri`. Those 2 have a



Road Trip

TITLE	ARTIST	ALBUM		
I'm Gonna Be (500 Miles)	The Proclaimers	Sunshine on Leith	32 minutes a...	3:39
Bohemian Rhapsody - 2011 Mix	Queen	A Night At The Opera (2011 Remas...	32 minutes a...	5:54
Highwayman	The Highwaymen, Willie Nelson, J...	Highwayman: The Best of Johnny ...	30 minutes a...	3:03
On The Road Again	Canned Heat	The Best Of Canned Heat	21 minutes ago	3:26

Recommended Songs

Based on the songs in this playlist

ADD	Mammas Don't Let Your Babies Grow up to I	Waylon Jennings, Willie Nelson	Waylon & Willie	2:33
ADD	Folsom Prison Blues - Live at Folsom State P	Johnny Cash	At Folsom Prison	2:43
ADD	The Road Goes On Forever	The Highwaymen	The Road Goes On Forever	4:41
ADD	If You Don't Like Hank Williams	Kris Kristofferson	Surreal Thing	3:32
ADD	You Never Even Called Me By My Name	David Allan Coe	For The Record- The First 10 Years	5:14
ADD	Mama Tried - Remastered 2001	Merle Haggard & The Strangers	Mama Tried	2:13
ADD	Family Tradition	Hank Williams, Jr.	Family Tradition	4:03

Figure 1: Spotify's playlist creation feature

1-1 correspondence and can both be used as index. It should be noted that we are responsible for assigning a unique `track_id` during preprocessing the dataset, while the `track_uri` of each track is given to us in the original dataset. Each playlist, apart from its `playlist_ id`, is accompanied by some metadata, such as its title and a short description and the uris of the tracks that it contains. For each track in the playlist, we know its position in the playlist, its title, the artist the track belongs to etc (see section 3). This type of feedback can be classified as implicit feedback. In recommender systems, there are 2 types of feedback, explicit feedback, such as user reviews and ratings, and implicit feedback, such as user clicks and views. In our case, we don't possess explicit information about how fitting a track is for a particular playlist; we can only assume that the track is fitting, because the user chose to include it in their playlist. Although implicit feedback is easier to collect than explicit feedback, it is characterized by a natural sparsity of negative feedback; we cannot possibly know if a user considers a track completely irrelevant for a particular playlist. The fact that the track is not included in the playlist can also indicate that the user is not aware of its existence.

We experimented with different approaches for the task of automatic playlist continuation. Our main goal was to experiment with both collaborative filtering and content-based filtering as well as a combination of the two. For collaborative filtering, we employed the matrix factorization technique from (1) that utilizes alternating least squares in order to compute user and item (or, in our case, playlist and track) latent factors. In addition, we also used a variation of the K-Nearest Neighbor algorithm described in (2) that recommends tracks based on the tracks that exist in the K most similar playlists to the query playlist. To incorporate content-based features, we proposed a modification on the aforementioned KNN algorithm inspired by (3) that qualifies similarity between two playlists as a weighted average of the collaborative filtering similarity and content based similarity.

2 LITERATURE REVIEW

Great interest has been shown around the task of automatic playlist continuation. The most notable example is the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation. In this open competition, each participating team had to submit their solution for the task of APC on the Spotify Million Playlist (MDP) dataset (see section 3) in one or both of the following tracks: main and creative. For the former track, the teams were constrained to only using the given dataset for development and evaluation, whereas in the creative track, the teams had the chance to use other publicly available datasets or create their own features. An overview of the best performing approaches is presented in (4).

For the main track, most teams employed a 2-stage approach; the first stage initially retrieves a small set of tracks compared to the entire track library and the second stage re-ranks those tracks with the aim of improving accuracy.

A popular approach used in the main track predominantly was matrix factorization (MF) for collaborative filtering (CF). These approaches mostly create an incomplete playlist-track matrix and use matrix factorization to learn a low-dimensional dense representation for each playlist and track. They learn similar representations for the tracks that often occur together in user-created playlists. The matrix factorization algorithms used by the top teams include weighted regularized matrix factorization (WRMF) (1) and LightFM with a weighted approximate-rank pairwise (WARP) loss (5). Remarkably, other teams achieved very promising results using simple neighborhood-based CF methods employing playlist-playlist similarity to recommend tracks, for example (2).

Other teams utilized neural networks. These neural approaches include: (1) simple feed-forward networks for predicting tracks given each playlist or for neural collaborative filtering (6), (2) convolutional models for playlist embedding or extracting useful information from playlist titles, (3) recurrent neural networks and in particular long short-term memory networks for modeling the sequence of tracks in the playlists, and (4) autoencoders for learning playlist representations.

As for the creative tracks, most teams that submitted to the main tracks, submitted a very similar model with minimal changes. They used the Spotify API (7) to retrieve audio features and incorporated them to their model either by re-re-ranking collaborative filtering approaches with scores from audio features or by enriching their content-based approaches with audio metadata.

3 DATASET AND FEATURES

For algorithm training and development we used the Million Playlist Dataset (MDP) (8), which contains one million user-created playlists from the Spotify Platform. The dataset includes, for each playlist, its title as well as the list of tracks (including album and artist names), and some additional meta-data such as Spotify URIs and the number of followers of the playlist. An example of a typical playlist entry is provided in listing 1.

Listing 1: Example of a playlist entry in MPD

```
{
        "name": "musical",
        "collaborative": "false",
        "pid": 5,
        "modified_at": 1493424000,
        "num_albums": 7,
```

```json
        "num_tracks": 12,
        "num_followers": 1,
        "num_edits": 2,
        "duration_ms": 2657366,
        "num_artists": 6,
        "tracks": [
            {
                "pos": 0,
                "artist_name": "Degiheugi",
                "track_uri": "spotify:track:7vqa3sDmtEaVJ2gcvxtRID",
                "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
                "track_name": "Finalement",
                "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az4O6UML",
                "duration_ms": 166264,
                "album_name": "Dancing Chords and Fireflies"
            },
            {
                "pos": 1,
                "artist_name": "Degiheugi",
                "track_uri": "spotify:track:23EOmJivOZ88WJPUbIPjh6",
                "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
                "track_name": "Betty",
                "album_uri": "spotify:album:3lUSlvjUoHNA8IkNTqURqd",
                "duration_ms": 235534,
                "album_name": "Endless Smile"
            },
            {
                "pos": 2,
                "artist_name": "Degiheugi",
                "track_uri": "spotify:track:1vaffTCJxkyqeJY7zF9a55",
                "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
                "track_name": "Some Beat in My Head",
                "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az4O6UML",
                "duration_ms": 268050,
                "album_name": "Dancing Chords and Fireflies"
            },
            // 8 tracks omitted
            {
                "pos": 11,
                "artist_name": "Mo' Horizons",
                "track_uri": "spotify:track:7iwx00eBzeSSSy6xfESyWN",
                "artist_uri": "spotify:artist:3tuX54dqgS8LsGUvNzgrpP"
```

```
                         ,
                    "track_name": "Fever 99\u00b0",
                    "album_uri": "spotify:album:2Fg1t2tyOSGWkVYHlFfXVf",
                    "duration_ms": 364320,
                    "album_name": "Come Touch The Sun"
                }
            ],

        }
```

There is also a separate challenge dataset, which, unfortunately, does not include the groundtruth values, so the only way to evaluate our algorithm on the challenge dataset is to submit our solution to the ongoing online competition (9). For that reason, we decided to split the training dataset into trainining, validation and testing dataset ourselves with an 80/10/10 split, which results in 800000 playlists in the training dataset, 100000 playlists in the validation and testing dataset each. To simulate missing track entries in the validation and testing dataset, we only kept the 25 first songs and tried to guess the remaining $n_i - 25$ songs for each query playlist i for the playlists with more than 25 tracks and discarded the rest, which, finally, resulted in 800000 playlists in the training dataset, 75549 playlists in the validation and 75313 in the testing dataset.

It is worth noting that as a preprocessing step, we had to read the JSON files that constituted our dataset into pandas (10) dataframes and store them in `.csv` files. We created 3 dataframes: a `playlists` dataframe, where we store the metadata of each playlist (`name`, `collaborative`, `pid` etc), a `tracks` dataframe, where we store the metadata of each track (`artist name`, `track uri` etc) and a `playlists-track` dataframe, where each entry is a tuple of the form (`playlist id`, `track id`, `artist id`, `position` of track in playlist, `value`), which contains only the positive interactions between playlist and tracks, so the field `value` is equal to 1. The need for different dataframes is storage efficiency. If we had only one dataframe for playlist and track interactions, then we would need to store the playlist and track metadata at each entry, which is wasteful, since we would repeat the same information unnecessarily multiple times.

To facilitate content based filtering, we augmented our dataset with audio features for each track we extracted using the Spotify API (7), such as `acousticness`, `danceability`, `energy`, `instrumentalness`, `mode`, `loudness`, `speechiness`, `tempo`, `time_signature` and `valence`. For the most part, those features were already scaled to 0-1, but some of them, such as `loudness` and `tempo` were not, with the latter having considerably larger values than the rest of the features, we used `StandardScaler` from `Scikit-Learn`(11) to scale all features, so that one does not dominate the others in terms of significance, only due to its scale.

4 METHODS

4.1 Matrix Factorization using Alternating-Least Squares

Let r_{vi} be a set of binary variables, which indicate that track i exists (one or more times) in playlist v, meaning that

$$r_{vi} = \begin{cases} 1 & \text{if } i \in M_v \\ 0 & \text{if } i \notin M_v \end{cases} \tag{1}$$

Because of the absence of negative feedback, any track that is not in the playlist ($r_{vi} = 0$) is automatically considered as negative feedback, which, as we mentioned before, is problematic. That is why we introduce a hyperparameter α, which acts as a positive weight for the positive samples and a set of variables $c_{\upsilon i}$ where

$$c_{\upsilon i} = 1 + \alpha r_{\upsilon i} \tag{2}$$

The goal is to find a vector $x_\upsilon \in \mathbb{R}^f$ for each playlist υ and a vector $y_i \in \mathbb{R}^f$ for each track i that will factor the playlist-track associations. The predictions can then be the inner products $\hat{p_{\upsilon i}} = x_\upsilon^T y_i$. Essentially, those vectors strive to map playlists and tracks into a common latent factor space where they can be directly compared. Those factors are found by minimizing the cost function

$$\min_{x_*, y_*} \sum_{\upsilon, i} c_{\upsilon i}(r_{\upsilon i} - \hat{p_{\upsilon i}})^2 + \lambda \left(\sum_\upsilon ||x_\upsilon||^2 + \sum_i ||y_i||^2 \right) \tag{3}$$

The term $\lambda \left(\sum_\upsilon ||x_\upsilon||^2 + \sum_i ||y_i||^2 \right)$ is used to regularize the model such that it will not overfit on the training data. By fixing one of the matrices x or y, we obtain a quadratic form which can be solved directly. The solution of the modified problem is guaranteed to monotonically decrease the overall cost function. By applying this step alternately to the matrices x and y, we can iteratively improve the matrix factorization.

Even though MF techniques are simple and do not require domain knowledge, they cannot easily handle fresh items. If a track (or playlist) that has not been seen during training is added, then the algorithm does not have an available emdedding for this track/playlist and cannot query it. For this task, methods such as Weighted Alternating Least Squares (WALS) can be used. Another disadvantage of the MF methods is that they cannot easily take advantage of side features that are domain-specific, hence missing out the opportunity on potential improvement (12).

4.2 K-Nearest Neighbor for Collaborative Filtering (CF)

In order to overcome the aforementioned issues, we employed a simplified version of the K-nearest neighbor CF algorithm of (2). For each query playlist u we define the cosine similarities with each playlist υ in the training set as

$$s_{cf_{u\upsilon}} = \sum_{i \in I} \frac{r_{ui} r_{\upsilon i}}{||R_u||_2 ||R_\upsilon||_2} \tag{4}$$

where r_{ji} is relevance value for track i in playlist j which can be either 1 or 0 depending on if track i exists in playlist j. For each query playlist u, we estimate the score for track i that is not in playlist u as

$$\hat{r_{ui}} = \frac{\sum\limits_{\upsilon \in N_k(u)} s_{cf_{u\upsilon}} \cdot r_{\upsilon i}}{\sum\limits_{\upsilon \in N_k(u)} s_{u\upsilon}} \tag{5}$$

where $N_k(u)$ are the k most similar playlists to playlist u as defined by the cosine similarity of equation 4. We also take advantage of the fact that rare items define similarity better than common items. In the context of automatic playlist continuation, two playlists are more likely similar if they include the same low popularity tracks than if they share tracks that a very large number of other lists also include. Therefore, we modify equation 4 by multiplying with a factor proportional to the inverse item frequency:

$$s_{cf_{uv}} = \sum_{i \in I} ((f_i - 1)^{\rho} + 1)^{-1} \frac{r_{ui} r_{vi}}{||R_u||_2 ||R_v||_2} \tag{6}$$

where f_i denotes the number of playlists containing track i. The new similarity values of 6 are used in equation 5. The tracks that are recommended are the 500 tracks with the greatest estimated relevance values.

4.3 Augmenting K-Nearest Neighbor for CF with content-based filtering

We define the content-based cosine similarity between 2 playlists v and u as the cosine similarity:

$$s_{cb_{vu}} = \frac{\hat{f}_v \cdot \hat{f}_u}{||\hat{f}_v||_2 ||\hat{f}_u||_2} \tag{7}$$

where $\hat{f}_p$ is a vector consisting of the mean audio features values across all tracks in the playlist. The content-based similarity qualifies how similar the mean audio features of two playlists are. To combine the CF (collaborative filtering) similarity with the CB (content-based filtering) similarity we define the hybrid similarity as a weighted average of the two as

$$s_{hybrid_{vu}} = w_1 \cdot s_{cf_{vu}} + w_2 \cdot s_{cb_{vu}} \tag{8}$$

where w_1 and w_2 are hyperparameters, which control the contribution of each similarity in the final ranking.

5 EXPERIMENTAL EVALUATION

5.1 Experiments

After splitting the dataset into an 80/10/10 training/validation/testing split and simulating missing entries in the validation and testing datasets (see section 3), we tuned the hyperparameters of all 3 of the proposed methods on the validation dataset and performed the final evaluation on the testing dataset. For each recommendation model, we chose a range of the hyperparameters that was suggested by the respective source publication, performed grid search, evaluating all combinations on the validation dataset and, in the end, selected the hyperparameter combination that performed best on the validation dataset. More specifically, for the matrix factorization method using alternating least squares, the tunable hyperparameters were the number of latent factors F, the weight of the positive samples α and the regularization strength λ. For the K-Nearest neighbor for collaborative filtering, the hyperparameters were the number of nearest neighbors K and ρ which controls the importance of the popularity of the tracks in the model. Finally, for the K-nearest neighbor augmented with content-based filtering algorithm, we chose to fix the values of K and ρ to the optimal values we found using grid search for the purely collaborative filtering based K-nearest neighbor and tuned the weights w_1 and w_2 that control the contribution of the CF and CB similarity values in predicting track relevance. Table 1 enlists the values of the hyperparameters we tried for each model.

To test the efficiency of each proposed algorithm, we utilized the R-precision metric. If we denote the set of recommended tracks for a playlist i as R and the set of withheld groundtruth tracks as G, then R-precision is the number of relevant recommended tracks divided by the number of withheld tracks and averaged over all playlists in the validation/testing dataset P. More formally:

Model	Hyperparameter	Values
MF w/ ALS	Factors	$[50, 100, 150, 200, 500]$
MF w/ ALS	α	$[0, 0.1, 1, 10, 100, 50]$
MF w/ ALS	λ	$[1, 10, 20, 40, 100]$
KNN w/ CF	K	$[100, 500, 1000, 2000]$
KNN w/ CF	ρ	$[0, 0.1, 0.3, 1.0]$
KNN w/ CF & CB	$[w_1, w_2]$	$[(0.0, 1.0), (1.0, 0.0), (1.0, 0.5), (1.0, 1.0), (1.0, 2.0)]$

Table 1: Models and range of tested hyperparameters

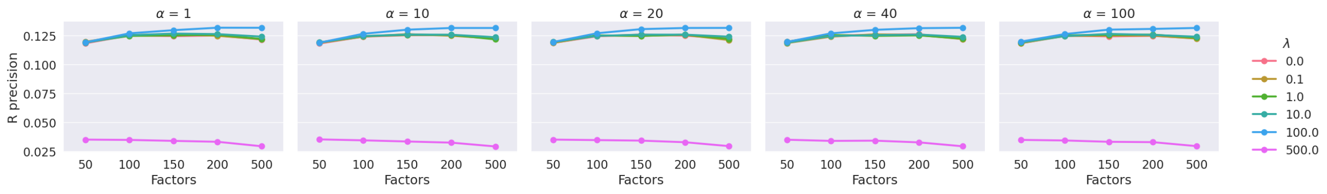


Figure 2: R-precision for MF w/ ALS

$$\frac{1}{|P|}\sum_{i=1}^{|P|}\frac{|G_i \cap R_{i_{1:|G|}}|}{|G_i|} \tag{9}$$

We aim for R-precision to be as high as possible. For reference, the team at the top of the leaderboard in the ongoing online competition, achieved an R-precision of 0.22 on the challenge dataset, while the 50th place is at 0.146, so an R-precision in this range can be deemed as satisfactory.

5.2 Results

5.2.1 Matrix Factorization using Alternating-Least Squares

Figure 2 shows the R-precision on the testing dataset achieved for varying combinations of hyperparameters.

The first observation that could be made is that the value of α in our case does not majorly affect the results. The idea of weighing the positive interactions makes much more sense in the cases where $r_{\upsilon i}$, which indicates if a song i exists in playlist υ, is not a binary variable and higher values of r_υ indicate a more positive relationship, for example number of clicks, amount of screentime etc.

In addition, performance improves when increasing the number of latent factors, but slightly decreases if we increase them over a certain threshold. A model with a small number of factors is too simplistic and could potentially underfit the data, because it fails to capture underlying patters and relationships. Conversely, a model with a large number of factors overfits the data, because it captures noise and random fluctuations.

On the topic of overfitting, the regularization parameter λ can help mitigate this phenomenon. As the regularization parameter becomes larger, the performance improves. The positive effect of the regularization parameter is especially prominent for a large number of factors, where the difference in the performance between models with different λ is evident. The improved performance when $\lambda \neq 0$ indicates that there is, indeed, an overfitting issue. However, for the largest value of λ we tested our model with, the performance is severely diminished, indicated that many values in the latent factor matrices were pushed to zero or very

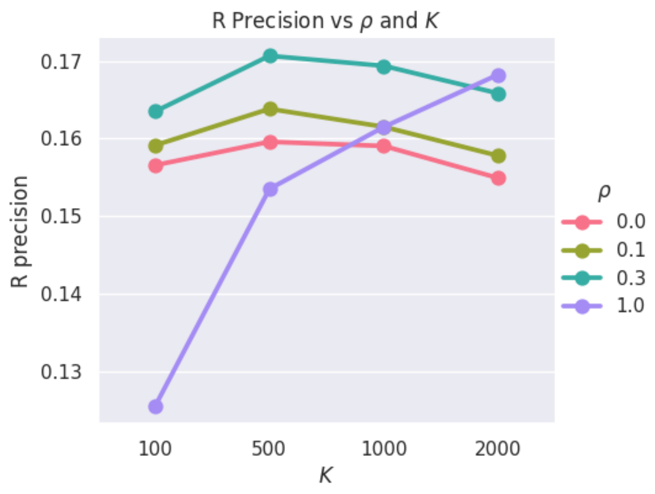


Figure 3: Results for KNN for CF

small values, resulting in an overly simplistic model that fails to capture the underlying patterns in the data. For the optimal hyperparameter values as evaluated in the validation dataset (factors = 200, λ = 100 and α = 1), we achieve an R-precision of 0.13 on the testing dataset.

5.2.2 K-Nearest Neighbor for Collaborative Filtering (CF)

Figure 3 depicts the results of our experiments for K-Nearest Neighbor for Collaborative Filtering.

Many interesting observations could be inferred from this plot. The number of nearest neighbors K has a similar effect on our model as the number of latent factors had in the MF model. For a small number of factors, our model might be picking up random noise in the data, resulting in a lower bias but high variance model. For a large number of K, the variance is decreased, because the model becomes less sensitive to outliers and noise, but the bias in increased. Hence, a moderate value of K produces the best results.

Our model is especially sensitive to the value of ρ. For $\rho = 0$, the model does not take into consideration the inverse popularity of each track and as ρ increases, the significance of the inverse popularity of each song increases as well. We can confirm that a value of $\rho = 0.3$ (as in (2)) performs optimally. The variance in the performance for different values of K also increases as the value of ρ increases, signifying that weighing by the inverse item popularity renders our model more sensitive to hyperparameter changes. For the optimal hyperparameter values as evaluated on the validation dataset (K = 500, ρ = 0.3), we achieve an R-precision of 0.16 on the testing dataset.

5.2.3 K-Nearest Neighbor for CF with content-based filtering

Finally, fig 4 presents the result of the CF-CB hybrid method for different combinations of weights w_1 and w_2.

The logic behind choosing those particular weights was that for $w_2 = 0$ and $w_1 = 1$, we retrieve the CF model, for $w_1 = 0$ and $w_2 = 1$, we get a purely CB model and for $w_1 = 1$ and $w_2 = (0.5, 1.0, 2.0)$, we get models where the content-based similarity is half as important or equally important or twice as important as the collaborative-filtering similarity respectively. Unfortunately, any combination of parameters that results in a model different than the purely CF based one, yielded very disappointing results. The reasoning

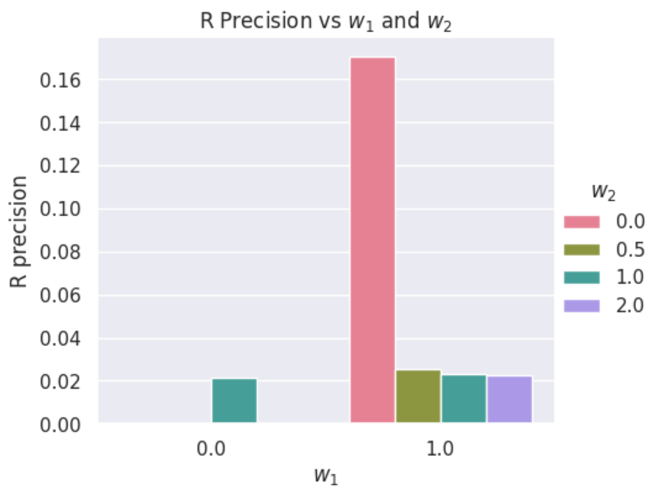


Figure 4: Results for Hybrid Recommendations

behind this might be the fact that tracks that belong to entirely different genres might still exhibit similar characteristics, for example an EDM playlist can be equally as loud and fast as a metal playlist. Also, some playlists are centered around a certain topic or mood and, hence, can contain diverse songs. Content-based filtering recommends very similar, in terms of audio features, tracks, whereas playlist creation relies an exploration and discovery of diverse tracks. Finally, the quality of features is also a limiting factor here. Even though the Spotify API from where we crawled the data is considered a reliable source, the features were vague and probably failed to capture nuances in user preferences.

6 CONCLUSION/FUTURE WORK

To conclude, our experiment results mostly align with existing literature. In more than one cases, we explored the bias-variance trade-off, which makes hyperparameter tuning a necessity for producing accurate and reliable results. Furthermore, we confirmed that Collaborative Based filtering typically works better than Content Based Filtering and is more reliable and robust. We also explored how difficult the task of automatic playlist continuation actually is. It is important that the list of recommendations is diverse enough, but also still similar to the content of the query playlist (exploration-exploitation trade-off). It also requires prior effort and feedback from the user to produce satisfactory recommendations.

Despite the simplicity of our models, we still managed to produce results that align with the state-of-the-art. Had we had more computation resources, we could try out deep learning models and benefit from their increased representational capacity. The prohibiting factor here was the size of our dataset; with over 2 million songs, if we created a neural network that predicts the probability for each song for each query playlist, we would have to have an output layer with over 2 million neurons! Decreasing the size of the dataset also would not be an option; choosing to include a smaller number of playlists in the training dataset, limits the number of songs in our universe of tracks. The missing tracks could be existing in the query playlists, but will never be included in the recommendations, since they were never included in the training dataset to begin with, which would result in a decrased and non-representative R-precision metric.

In the future, we would like to explore the issue with hybrid recommendations more deeply and perhaps submit our own version of the algorithm to the online competition.

7 AUTHOR CONTRIBUTIONS

Both authors contributed equally in this project but in different ways. The idea of the project subject, the literature review and the dataset cleaning and preprocessing, can be attributed to Ioannis, while the development of the algorithms and experiments was the task of Ioanna. The report was written by both authors.

REFERENCES

1. Hu, Y., Y. Koren, and C. Volinsky, 2008. Collaborative filtering for implicit feedback datasets. *In* 2008 Eighth IEEE international conference on data mining. Ieee, 263–272.

2. Kelen, D. M., D. Berecz, F. Béres, and A. A. Benczúr, 2018. Efficient K-NN for playlist continuation. *In* Proceedings of the ACM Recommender Systems Challenge 2018, 1–4.

3. Ludewig, M., I. Kamehkhosh, N. Landia, and D. Jannach, 2018. Effective nearest-neighbor music recommendations. *In* Proceedings of the ACM Recommender Systems Challenge 2018, 1–6.

4. Zamani, H., M. Schedl, P. Lamere, and C.-W. Chen, 2019. An analysis of approaches taken in the acm recsys challenge 2018 for automatic music playlist continuation. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10:1–21.

5. Kula, M., 2015. Metadata embeddings for user and item cold-start recommendations. *arXiv preprint arXiv:1507.08439* .

6. He, X., L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, 2017. Neural collaborative filtering. *In* Proceedings of the 26th international conference on world wide web. 173–182.

7. Spotify API documentation for retrieving audio features. https://developer.spotify.com/documentation/web-api/reference/get-audio-features.

8. Spotify Million Playlist Dataset (MDP). https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge.

9. AI Crowd Spotify Million Playlist Dataset Competition. https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge.

10. McKinney, W., et al., 2015. Pandas, python data analysis library. *URL http://pandas. pydata. org* 3–15.

11. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12:2825–2830.

12. Google Recommendation Systems Course. https://developers.google.com/machine-learning/recommendation/collaborative/matrix.