Automatic Playlist Continuation with Approximate Nearest Neighbours

Chris White
Engineering Mathematics
University of Bristol
gd19031@bristol.ac.uk

Elliot Hayward
Engineering Mathematics
University of Bristol
vu19320@bristol.ac.uk

Tom Blain
Mathematics
University of Bristol
pm19769@bristol.ac.uk

Edward Peterson
Engineering Mathematics
University of Bristol
lb19337@bristol.ac.uk

***Abstract*—In recent years, modern machine learning techniques have led to technological revolutions in many industries. Increasing amounts of online musical content has created new opportunities for implementing effective information access services that support music navigation and discovery - also known as music recommender systems. Given a playlist of arbitrary length with accompanying metadata, the task is to recommend tracks that fit the characteristics of the original playlist. Automatic playlist continuation (APC) is a common task of music recommender systems. The problem is taken from Spotify's 'The Million Playlist Dataset (MPD) Challenge' (2020) [1]. This paper proposes a single stage content-based filtering model consisting the application of nearest neighbours algorithms to track feature vectors for APC, utilising the Spotify platform. Initially, adopting k-nearest neighbours and then progressing to an approximate nearest neighbours algorithm. The model is able to significantly outperform the baseline and holds potential for further refinement. The code is freely available on GitHub [2].**

I. Introduction

Over the last 5 years, music streaming services have continued to become increasingly popular. From 2016 to 2021, Spotify alone rose from $\sim$36 million to $\sim$165 million subscribers [3]. As revenues rise, the level of competition between the different streaming platforms will only continue to increase. Providing excellent additional features such as music recommender systems is of great importance to each streaming service. They typically work by utilising data representing the tracks, which are currently within a playlist, in order to predict tracks which would fit well within said, existing playlist. In general, recommender systems are important to streaming services as they keep users engaged with their product and, therefore, increase the chance of users renewing their subscriptions.

This project is inspired by 'The Million Playlist Dataset (MPD) Challenge' (2020) [1], which, in itself, is a recreation of the ACM RecSys Challenge (2018) [4]. The MPD contains one million Spotify playlists and the Spotify API references for every track within each playlist. Here, the MPD is utilised to create a music recommender system. A method of APC which uses machine learning to predict the most suitable tracks for a chosen playlist. More specifically, the task is to suggest a list of new tracks which fit the characteristics of the chosen playlist.

To begin, a set of features for all tracks within the playlists of the MPD is obtained by utilising the Spotify API. Then, the only non-numeric data, the genres of the artist's of each track, are represented using term frequency–inverse document frequency (TF-IDF). Feature vectors are scaled and utilised by a content-based filtering method (CBFM) to carry out APC. The results from this method are analysed using a variety of evaluation techniques, to investigate the effectiveness of the proposed method.

II. Literature Review

There are a multitude of machine learning techniques which can be used to create different recommender systems. The three main methods used within recommender systems are CBFM, collaborative filtering (CF) and hybrid methods, which is the combination of multiple methods into one recommender system. CBFM utilises a user's past activity to find similar objects within a database to suggest to them [5]. In comparison, CF finds like-minded users to calculate predictions for the current user. It does this by finding users who have similar interests and then extracting objects from the database which they are interested in [6].

An example of all three of these methods being used to create music recommender systems can be seen in the work of Markus Schedl [7]. His work focused on recommending music to classical music listeners using a variety of techniques. The methods used include CBFM and CF as well as hybrid techniques. Schedl concluded that hybrid techniques tend to outperform singular models. Marius Kaminskas et al. [8] similarly concluded that the debate between CBFM and CF recommendation techniques is on-going, though collaborative techniques have limitations by design and content based techniques are continuously evolving. Currently the best accepted solution is a hybrid of the two.

Ching-Wei Chen et al.'s [9] analysis of APC (Automatic Playlist Continuation) approaches taken for the ACM RecSys challenge [4] introduces many techniques for consideration. The Creamy Fireflies [10] apply a variety of CBFM and CF methods, aggregating predictions from these models in an ensemble method. Thus, leveraging well-known techniques whilst limiting the required computational resources. While the winning team vl6 [11] utilised a combination of CF and deep learning techniques, optimising levels of recall at over 90%, but at great computational expense. All of the competitive

teams utilise boosting and multi-stage architecture to take advantage of a wide variety of techniques.

For this application of APC, CBFM is the most suitable since, in CBFM, the model does not require any knowledge of other users and in this application the recommendations are user specific. The recommender can be easily scaled up to a large number of users and can recommend more niche items to a singular user. The primary reason for not using CBFM techniques [12] is they usually overly specialise to a users profile, leading to a lack of unexpected items. Furthermore, they require items that can be represented effectively using extracted textual features. Given extensive knowledge of the domain in APC, the latter can be negated.

III. Dataset

The MPD [9] consists of details of 1 million unique public Spotify playlists created by American Spotify users and is the largest dataset of its kind in the world. Each playlist was created between January 2010 and November 2017 and contains 5-250 tracks. The MPD contains the title, track list and a variety of other metadata for each playlist. This includes the Spotify URI for every track within each playlist [13]. Some further information about the contents of the dataset can be seen within Table I.

Variable	Value
Number of tracks	66,346,428
Number of unique tracks	2,262,292
Number of unique artists	295,860
Average number of tracks per playlist	66.35

TABLE I
INFORMATION ABOUT THE MILLION PLAYLIST DATASET

A. Limitations of the Dataset

As this dataset is sampled from playlists created by Spotify users within the USA, it is likely that any recommendations will be biased towards the patterns of these listeners. Potential issues include a tendency to suggest tracks which are sung in English and any model may work poorly on playlists created by users from other countries. Additionally, the MPD only contains playlists created between January 2010 and November 2017 meaning it is not possible for any tracks released after November 2017 to be included within the model and the subsequent recommendations produced.

B. Features from the Spotify API

We added to the data provided in the MPD by making requests to the Spotify API. This was achieved using the Spotify URI (Uniform Resource Indicator) for each track, which is provided within each playlist of the MPD, in order to obtain features that could be used to assess the similarity of different tracks and playlists.

Using the Spotify API, we can request 13 audio features associated with all of the unique tracks within the playlists. These features can be seen in Table II [14]. Furthermore, by requesting information on the artists, extra features can be obtained. These are the popularity of the artist and the genres that each artist is associated with.

Feature	Description
Danceability	A value between 0 and 1 which estimates how easily a person can dance to a track, where a higher value denotes a more danceable track.
Energy	A value between 0 and 1 which measures the intensity and activity within a track.
Key	The grouping of pitches that a track uses.
Loudness	The overall volume of the track $[dB]$.
Mode	A boolean variable which defines whether a track is major (1) or minor (0).
Speechiness	A measure between 0 and 1 of how much speech there is within the track. The closer the value to 1 the more speech within the track.
Acousticness	A confidence between 0 to 1 of whether the track is acoustic, where 1 represents a high level of confidence that the track is acoustic.
Instrumentalness	A value between 0 and 1 in order to predict on whether a track contains no vocals, where a value of 0.5 should represent an instrumental track.
Liveness	A measure of the probability that the music was performed live, with the presence of an audience. Any value above 0.8 implies that the track is live.
Valence	A measure between 0 to 1 describing the musical positiveness conveyed within a track, where a higher score denotes a positive.
Tempo	The number of beats per minute in a track.
Time signature	The estimated time signature of a track, which ranges from 3 to 7.
Duration	The length of the track in milliseconds.

TABLE II
ALL OF THE AUDIO FEATURES OBTAINED FOR EACH TRACK WITHIN THE MPD USING THE SPOTIFY API.

IV. Method

A. Data Pre-processing

A list of all of the unique tracks in the MPD is extracted and then each track is transformed into a vector using the features displayed within Table II, as well as features generated using natural language processing (NLP) techniques and the popularity of the artist associated with the track.

The magnitudes of the features, displayed in Table II, vary greatly. Therefore, the data must be scaled so that any model is not automatically skewed towards certain features ahead of others. For scaling the data, the mean is subtracted whilst the variance is normalised such that a value, z, within the vector representation of a track becomes:

$$z = \frac{x - \mu}{\sigma}, \tag{1}$$

where μ is the mean of the feature across all tracks and σ is the standard deviation of the feature. This operation is performed independently on each of the 13 audio features described in Table II as well as the artists' popularity.

Once all of the tracks in the MPD have been converted into their vector form. To find which of these tracks are a good fit for a playlist the playlists need to be put into the same form. This is done by using the same features as the tracks, but the values of each feature are averaged over all of the tracks within the selected playlist.

B. NLP Feature Generation

The only feature obtained from the Spotify API that is non numeric are the genres associated with the artists. In order to represent the genre information for each track numerically, TF-IDF is used. TF-IDF works well for this application because it encodes the rarity of the terms in the corpus [15]. Rare genres will be given a greater weight in the recommender system which will result in more specific recommendations.

Upon applying TF-IDF to the genres, a vector with a length of 2741 is obtained for each track and for each playlist. This is concatenated to the end of the vector of the audio features to create the final representation of both the tracks and playlists.

By using TF-IDF, the most and least popular genres and sub-genres in the MPD can be identified. The most popular genres include 'pop', 'rock' and 'indie' and the least popular genres include 'amapaense' and 'kodomo'.

C. Generating Recommendations

Initially we used k-nearest neighbours (KNN) to find the closest track vectors for each playlist vector. However, this method was found to be prohibitively slow when working with such a large dataset with a high dimensionality. Therefore, faster alternatives to KNN must be used to counter this problem. An example of one of these exists in the form of approximate nearest neighbour (ANN) algorithms such as Annoy [16]. This method is much faster when dealing with large quantities of high dimensional data [17] and is similar to the algorithm employed for Spotify's own music recommender system [16].

Annoy is a tree-based space partition method, which works by recursively constructing a forest of trees. Each tree is constructed as follows. Firstly, 2 random points in the dataset are chosen and a hyperplane is constructed, which is equidistant from each point. This partitions the dataset and forms the first branches of the tree. Each partition is then partitioned further using the same method, creating further branches of the tree. Partitioning continues until there is at most K items located within each partition. The key to the method is in the fact that points that are close to each other in the dataset are likely to be close together in the tree. Either in the same leaf partition or in a nearby one. A forest of trees is constructed using the same method and random hyperplanes. Constructing the trees, also known as the index, can be very time consuming, however, once constructed, finding the nearest neighbours of new or existing points is significantly faster than classical KNN.

When querying the model, the points within the partition of the queried point, or a subset of nearby partitions for all the trees are candidates for the nearest neighbours. Then, this subset of points can be used to calculate the distances and rank the nearest neighbours [18].

This method is not guaranteed to return all of the closest points since it is an approximate method. However, Annoy can do queries in $\mathcal{O}(log(n))$ time, a significant improvement over classical KNN which queries in linear time.

The Annoy model presents two hyperparameters for tuning, firstly, the number of trees used in the construction of the index, n_t, and the number of leaf nodes explored when querying the model, SK. n_t is chosen during the creation of the trees and is fixed at our chosen value of 10. It is expected that a greater SK value will provide more accurate recommendations at an increased computational cost. This is because more leaf nodes of the index are explored so it is more likely that the true nearest neighbours in the vector space are encountered.

As well as these parameters, the Annoy model allows the use of different distance metrics including Euclidean and angular distances. Angular distance, in Annoy, refers to the Euclidean distance between normalised vectors and is calculated as

$$D_A = \sqrt{2(1 - S_C(\mathbf{A}, \mathbf{B}))}, \tag{2}$$

where S_C refers to the cosine similarity between two vectors and

$$S_C = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||\,||\mathbf{B}||}. \tag{3}$$

Distance metrics that utilise the angle between vectors are commonly used with TF-IDF features because the magnitude of the vectors has no effect. Therefore it is robust when used to compare documents of different lengths.

V. Evaluation

In order to evaluate the predictions of our model, we use a set of evaluation metrics which were previously used to assess the results for the original RecSys Challenge [4].

The playlists are split such that 80% is a reduced set of tracks and 20% is a set of evaluation tracks referred to as 'true tracks'. The reduced set of tracks are used to generate a set of recommendations and the true tracks are used as a ground truth to compare these recommendations to. Our metrics are fundamentally based around the model finding these true tracks.

For a given playlist, let R be the set of recommendations that our model returns and T be the set of true tracks. The precision of this recommendation is defined as

$$Precision = \frac{|R \cap T|}{|R|}. \tag{4}$$

By averaging the Precision over many playlists the R-Precision can be calculated, this will be a value between 0 and 1 where a larger value corresponds to a higher similarity between the recommendations and true tracks.

Normalised discounted cumulative gain (NDCG) assesses the quality of the ranking provided by the model. The higher the more relevant tracks are positioned within R then the higher the NDCG score will be. NDCG is calculated as

$$NDCG = \frac{DCG}{IDCG}, \tag{5}$$

where,

$$DCG = rel_1 + \sum_{i=2}^{|R|} \frac{rel_i}{log_2(i)}, \tag{6}$$

$$rel_i = \begin{cases} 1 & \text{if } R_i \in T \\ 0 & \text{otherwise} \end{cases}, \tag{7}$$

and

$$IDCG = 1 + \sum_{i=2}^{|R \cap T|} \frac{1}{log_2(i)}, \tag{8}$$

Spotify has a feature in which it returns a list of 10 recommended tracks for a playlist. This list can be refreshed to get a new set of recommendations. In order to evaluate our method, this functionality can be reproduced. The first 10 tracks in the recommendations R are shown and if none of the true tracks in T are returned, the next 10 recommendations in R are shown until there is a relevant track. The metric Recommended Song Clicks (RSC) is the minimum number of refreshes required until a relevant track is recommended.

Let R be the set of recommendations from the model such that R_i is the ith ranked track indexed from 0, S_j a set of 10 tracks contained in R, such that $j \geq 1$ is the click number, and let T be the set of true tracks

$$S_j = \bigcup_{i=10(j-1)}^{10j-1} \{R_i\} \tag{9}$$

$$RSC = \min\{\{j \in \mathbb{Z} : T \cap S_j \neq \emptyset\} \cup \{51\}\}. \tag{10}$$

As a point of comparison, a baseline model has been created which generates recommendations comprised of a random selection of tracks. This baseline method can be evaluated using the previously defined metrics and the results can be compared to those obtained from our model. In addition, we can compare our results to the results obtained during the original RecSys challenge.

VI. Experimentation

Upon comparing the use of Euclidean and angular distances, it was found that angular distance significantly outperforms Euclidean distance when used in the Annoy ANN model. The comparison between the distance metrics in relation to the previously defined evaluation metrics can be seen in Figures 1 2 and 3. Our best results are computed using angular distance and a SK value of 10000.

As expected, it was found that increasing the value of SK used when querying the model would increase the performance. Significant performance increases can be seen across all three evaluation metrics for both Euclidean and angular distances. However, this increase in performance comes with the cost of increased computational complexity and time [Figure 4]. This figure shows that for lower values of SK, using angular distance affords major time savings. However, as SK increases, the difference between the use of the two distance metrics reduces.

The baseline method has an RSC score of 50.946. In comparison, the RSC score of our approach was 40.51, roughly a 20% improvement. Similar improvements to the baseline can be seen in the other evaluation metrics, shown in Table III.

The winning team from the RecSys challenge was able to significantly outperform both the baseline method and our method when comparing all three evaluation metrics.

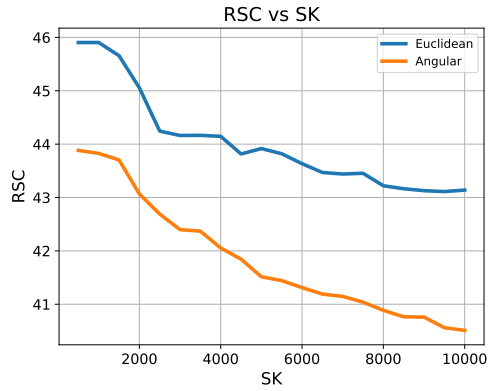

Fig. 1. Figure showing the decrease in RSC as SK is increased.

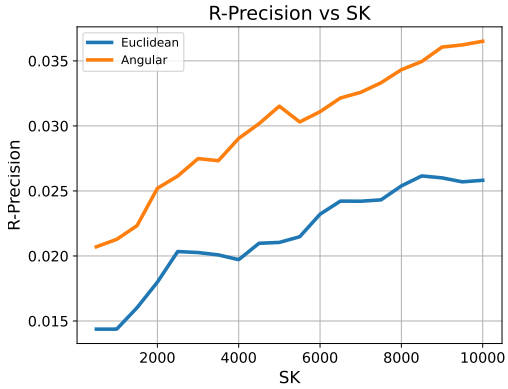

Fig. 2. Figure showing the increase in R-Precision as SK is increased.

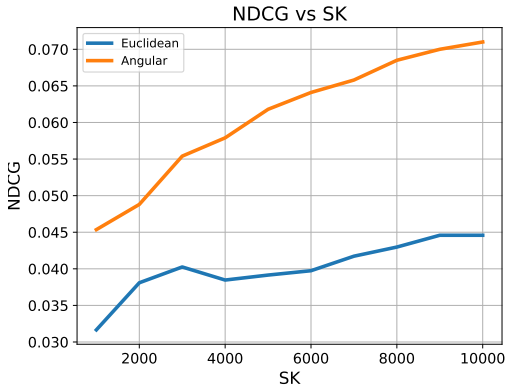

Fig. 3. Figure showing the increase in NDCG as SK is increased.

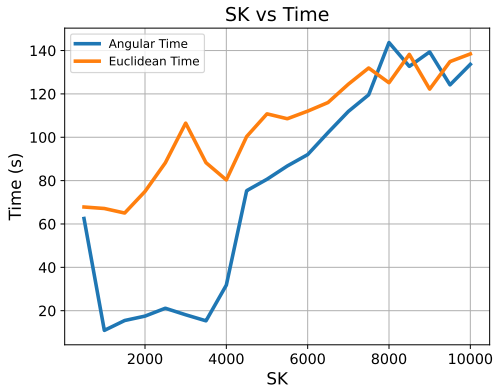

Fig. 4. Figure showing the increased time taken to generate 100 recommendations as SK increases.

Method	RSC	R-Precision	NDCG
Baseline	50.946	1.32×10^{-4}	2.51×10^{-7}
Winner	1.784	0.2241	0.3946
Our Method	40.51	0.03651	0.071

TABLE III
A COMPARISON OF THE PERFORMANCE OF OUR METHOD, THE RANDOM BASELINE AND THE RECSYS CHALLENGE WINNER.

VII. HUMAN RECOMMENDATION EVALUATION

To further evaluate the model, we enabled users to input their own playlists and receive 50 recommendations in a Spotify playlist created using the API.

On playlists with a very specific genre focus, the recommendations were very highly rated. For playlists of varying lengths, recommendations would often match those generated by Spotify itself and make a lot of sense from a human recommendation point of view. The model had more difficulty producing accurate recommendations when the playlists were highly varied. For example, one playlist had tracks from Metallica, Doja cat, and Kanye West to which the system started recommending hard 90s hip-hop and trap next to EDM remixes of Niall Horan and Lenny Kravitz. It's suspected this may be because of the averaging of the playlist features vector in our method. Playlists with moderate variance in genres work well, but those containing around 10-25% outlying tracks produced less appealing recommendations.

Recommendations could be given to low listen count tracks and did not seem to heavily bias the most popular tracks which was praised by users. Many recommendations were unfamiliar tracks by known and liked artists.

Testing on niche sub genres was successful, however, the model had difficulty recommending non-English music even when given a playlist of entirely Japanese jazz songs. Although the feeling and genre of the songs was accurate, all were in the English language. This is likely due to the dataset being collected from entirely US Spotify accounts [III-A].

VIII. DISCUSSION

When reviewing the results, our model is clearly able to outperform the baseline method, obtaining a better score in each of the evaluation techniques. This shows that the method being utilised is better than simply randomly selecting tracks to suggest to the user. Feedback from users showed that the recommendations were often meaningful and were of value.

The use of the Annoy ANN model was a success, allowing a nearest-neighbours approach to be used on a dataset that is prohibitively large for the classic KNN result. It also allows for the model to be tuned for memory and time complexity through the parameter SK.

It can be seen that angular distance performs higher than Euclidean distance. This is likely due to the sparsity in the playlist and song representation due to the use of TF-IDF features. When using TF-IDF, cosine distance is typically used to overcome the sparsity. Therefore, in our representation where the majority of the features are sparse, an angular approach to distance performs better [19]. Furthermore, since the documents formed of the genres are different lengths for the playlists and for the songs, an angular distance metric will be better for suggesting songs for a playlist.

However, when the results are compared to the best-scoring entries from the ACM RecSys Challenge, our model performs significantly worse. This is likely due to the fact that these models use either far more complex methods or take advantage of using hybrid methods in order to improve their performance. Furthermore, these methods require much more time and memory to produce recommendations. For example the winning team, vl6 had a memory requirement of at least 100GBs [11].

Following from section VII, the model may be able to perform better on high variance playlists if it found a way of dismissing outlying tracks that result in bad averaging on the playlist features vector. This could be approached by discarding tracks in a playlist with features significantly outside the range of the other tracks. Alternatively, a new playlist representation could be devised which avoids the averaging of features entirely.

IX. FUTURE WORK

There are several possible methods by which our music recommender system could be improved. These can be broadly categorised as changes to the representation of the tracks and playlists and as changes to the recommendation algorithm itself.

Instead of using TF-IDF to generate NLP vector representations of the genres associated with a track or a playlist, word2vec could be used instead. This would encode the relationship between different genres and sub-genres [20], allowing the system to make recommendations that do not explicitly fit in the same genre but are still closely linked.

Our representation of tracks and playlists has a high dimensionality and therefore it is susceptible to the 'curse of dimensionality'. To reduce the dimensionality, an autoencoder could be used to create a denser, lower dimensional embedding that is able to learn the most important features to represent. This has been successfully implemented in the literature [21].

It is possible to obtain more features from the Spotify API. Therefore, we could improve the model by adding these

additional features to those already obtained. However, to achieve this we would need to find a method of working around the rate limit that is imposed when requesting from the Spotify API. It may also be of interest to look into audio analysis which returns more music theoretical track features. These include the tempo, timbre and features describing the structure of a track [22].

Another way this work could be furthered would be by utilising hybrid recommendation technology. This is shown to be more effective for the recommending of classical music [23]. It is likely that this increased effectiveness would extend to music recommender systems in general. Implementing this would involve creating a CF recommendation system and then combining this with the system proposed here. This could be achieved by building a network of users, using the users who created the playlists within the MPD. This network could then be utilised, alongside the data of a new user, to locate users with similar interests and tracks which these users find interesting can be suggested to the new user.

X. Conclusion

In conclusion, through the vector embedding of Spotify tracks and playlists and the application of the Annoy ANN model we have generated recommendations that are evaluated as better than a random baseline method with scores of 40.51, 0.03651 and 0.071 in RSC, R-precision and NDCG respectively. While a significant improvement on the baseline, our results fall short of those obtained by the best performing methods in the 2018 RecSys Challenge.

This method holds potential for future work such as the use of a learnt, lower-dimensional embeddings for tracks and playlists and the use of CF methods. This would serve to increase the performance of the model and generate better and more relevant recommendations.

References

[1] "The Million Playlist Dataset Challenge," https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge, 2020.

[2] E. Peterson, T. Blain, E. Hayward, and C. White, "Automatic Playlist Continuation with Approximate Nearest Neighbours," 3 2022. [Online]. Available: https://github.com/EMAT31530-21-22/group-project-31

[3] "Music streaming app revenue and usage statistics (2022)," Jan 2022. [Online]. Available: https://www.businessofapps.com/data/music-streaming-market/

[4] "ACM RecSys Challenge," https://www.recsyschallenge.com/2018/, 2018.

[5] "What content-based filtering is and why you should use it: Upwork." [Online]. Available: https://www.upwork.com/resources/what-is-content-based-filtering#:~:text=Content%2Dbased%20filtering%20is%20a,them%20to%20a%20user%20profile.

[6] A. Ajitsaria, "Build a recommendation engine with collaborative filtering," Jun 2021. [Online]. Available: https://realpython.com/build-recommendation-engine-collaborative-filtering/

[7] M. Schedl, "Towards personalizing classical music recommendations," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015, pp. 1366–1367.

[8] M. Kaminskas and F. Ricci, "Contextual music information retrieval and recommendation: State of the art and challenges," *Computer Science Review*, vol. 6, no. 2, pp. 89–119, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013712000135

[9] C.-W. Chen, P. Lamere, M. Schedl, and H. Zamani, "Recsys challenge 2018," in *Proceedings of the 12th ACM Conference on Recommender Systems*. ACM, Sep. 2018. [Online]. Available: https://doi.org/10.1145/3240323.3240342

[10] S. Antenucci, S. Boglio, E. Chioso, E. Dervishaj, S. Kang, T. Scarlatti, and M. Ferrari Dacrema, "Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario," 10 2018, pp. 1–6.

[11] M. Volkovs, H. Rai, Z. Cheng, G. Wu, Y. Lu, and S. Sanner, "Two-stage model for automatic playlist continuation at scale," in *Proceedings of the ACM Recommender Systems Challenge 2018*, ser. RecSys Challenge '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3267471.3267480

[12] M. J. Pazzani and D. Billsus, *Content-Based Recommendation Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 90–98. [Online]. Available: https://doi.org/10.1007/978-3-540-72079-9_10

[13] P. b. C.-W. Chen, "Introducing the million playlist dataset and recsys challenge 2018," Jan 2022. [Online]. Available: https://engineering.atspotify.com/2018/05/introducing-the-million-playlist-dataset-and-recsys-challenge-2018/

[14] Spotify, "Web api reference: Spotify for developers." [Online]. Available: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-features

[15] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, no. 1. Citeseer, 2003, pp. 29–48.

[16] "ANNOY library," https://github.com/spotify/annoy, accessed: 2017-08-01.

[17] M. Aumüller, E. Bernhardsson, and A. J. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," *CoRR*, vol. abs/1807.05614, 2018. [Online]. Available: http://arxiv.org/abs/1807.05614

[18] W. Li, Y. Zhang, Y. Sun, W. Wang, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement (v1.0)," *CoRR*, vol. abs/1610.02455, 2016. [Online]. Available: http://arxiv.org/abs/1610.02455

[19] J. Han, M. Kamber, and J. Pei, "2 - getting to know your data," in *Data Mining (Third Edition)*, third edition ed., ser. The Morgan Kaufmann Series in Data Management Systems, J. Han, M. Kamber, and J. Pei, Eds. Boston: Morgan Kaufmann, 2012, pp. 39–82. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123814791000022

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[21] A. Da'u and N. Salim, "Recommendation system based on deep learning methods: a systematic review and new directions," *Artificial Intelligence Review*, vol. 53, no. 4, pp. 2709–2748, 2020.

[22] Spotify, "Web api reference: Spotify for developers, audio analysis." [Online]. Available: https://developer.spotify.com/documentation/web-api/reference/#/operations/get-audio-analysis

[23] S. Kathavate, "Music recommendation system using content and collaborative filtering methods," *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT)*, vol. 10, 2021.