Spotify Music Recommendation

BT5153 Applied Machine Learning in Business Analytics – Group 06 Project Report

Li Rui (A0231933J) Tang Ning (A0232009U) Fu Hanqi (A0231974Y) Gan Bingzheng (A0232004A) Pei Ziang (A0105574H)

Abstract

Online streaming music industry has grown rapidly in recent years. How to provide meaningful and engaging song recommendations to users becomes increasingly important. Recommendation system has been widely adopted by industries like e-commerce to improve user experience and grow business impact, and can benefit the music industry in a similar way.

Thus in this project, we explored 5 different machine learning based recommendation models using user preference and song feature datasets from Spotify. These models include: 1. content-based filtering; 2. user based collaborative filtering; 3. NMF model; 4. SVD model and 5. auto-encoder model. We also explored combining these models into an ensemble model. After evaluating with Hit Ratio and NDCG, user based collaborative filtering is identified as the best-performing model.

1. Introduction

As we know, in recent years, recommendation systems that use machine learning algorithms to improve user experience and business performance have been widely used, especially in the music streaming media industry, which greatly improves the user experience and viscosity of users.

Back in the 2000s, Songza kicked off online music curation using manual curation [1] to create playlists for users. But it was manual and simple, and therefore couldn't take into account the nuance of each listener's individual music taste.

Later, Pandora improved this in a creative way by employing a slightly more advanced approach to manually tag attributes of songs. They asked a group of people to listen to music and chose a bunch of descriptive words as tags for each track. In this way, Pandora's algorithm could simply filter for certain tags to curate playlists of similar-sounding music.

Nowadays, global audio streaming subscription service giant Spotify, which reportedly has 180 million subscribers and 82 million tracks [2], has invested heavily in building recommendation features. Discover Weekly is Spotify's fully personalized playlist management at scale, with more than 2.3 billion hours of total playtime [3] 5 years after its launch. Bandits for Recommendations as Treatments, or BaRT for short, is one of the key recommendation algorithms running on Spotify that generates weekly lists of the most relevant and engaging music for users every Monday.

Spotify uses three main functions in BaRT: Natural Language Processing (NLP), which work by analyzing text, Audio models, which work by analyzing the raw audio tracks themselves., and Collaborative Filtering (CF), which work by analyzing your behavior and others' behavior. They then further improved the model in April 2021, releasing a dynamic model called the Preference Transition Model that not only addresses the filter bubble problem, but also enhances user interaction and engagement. These recommendation models are all connected to Spotify's much larger ecosystem, which includes giant amounts of data storage and uses lots of Hadoop clusters to scale recommendations and make these engines work on giant matrices, endless internet music articles, and huge numbers of audio files.

2. Problem Statement

With above background in mind, we set out to propose the problem statement: how to develop a music recommendation system that combines machine learning knowledge learning, our research and understanding, and some concepts from Spotify's BaRT algorithm to improve online streaming music users' experience

This problem statement contains 2 main aspects:

1. Music recommendations for existing users: learn their preferences through past behavior and recommend songs accordingly
2. Music recommendation for new users and songs (i.e. cold start problem):

- For New Users: By clustering songs by their features,

we plan to select songs closest to cluster centroid for each cluster and recommend them to new users.

- For New Songs: By predicting the cluster that new songs belong to, we plan to recommend this new song to the users who like this corresponding cluster.

3. Data

3.1. Dataset Description

In this project, we not only selected a dataset called "spotify" from Kaggle, but also further explored and scraped a dataset called "spotify_audio_feature" from the online API provided by Spotify.

3.1.1. KAGGLE DATASET "SPOTIFY"

The first dataset "spotify", can be downloaded directly from Kaggle 4. It contains records of tracks in users' Spotify playlists based on their sharing with #nowplaying tag via twitter. The dataset contains around 13 million rows of data and the detailed information on variables is shown below:

Variable	Type	Descriptions
user_id	string	Unique hash of the user's Spotify username
artist	string	The name of the artist of the track
track	string	The title of the track which user listened to
playlist	string	The name of the playlist that contains this track

Table 1. Kaggle Dataset "spotify"

3.1.2. SCRAPED DATASET "SPOTIFY_AUDTIO_FEATURE"

Besides tracks in different users' playlists, we also explored and found other song features, such as danceability, energy and so on from the scraped dataset "spotify_audio_feature" using Spotipy[5], a lightweight Python library for the Spotify Web API.

There are two main steps to find information related to songs. Firstly, we obtain specific track id by searching the combination of artist and track name. Then, we call the API again to get access to audio features for different tracks with their unique track ids.

This dataset contains additional features for songs obtained from the first dataset and is composed of 1,335 songs. The description of the selected song features is shown below:

Variable	Type	Descriptions
track	string	The title of the track which user listened to
danceability	float	How suitable a track is for dancing
energy	float	A measure of intensity and activity, energetic tracks feel fast, loud, and noisy
key	integer	The estimated overall key of the track
loudness	float	The overall loudness of a track in decibels(db)
mode	integer	The modality (major or minor) of a track
speechiness	float	The presence of spoken words in a track
acousticness	float	Whether the track is acoustic
instrumentalness	float	Whether a track contains no vocals, rap or spoken word tracks are "vocal"
liveness	float	The presence of an audience in the recording
valence	float	The musical positiveness conveyed by a track
tempo	float	The overall estimated tempo of a track in beats per minute (BPM)
type	string	The type of the song
id	string	Unique id of the song
uri	string	A uniform resource indicator code for the song
track_href	string	The URL for the song
analysis_url	string	URL to a low-level audio analysis
duration_ms	integer	The duration of the track in milliseconds
time_signature	integer	An estimated overall time signature of a track

Table 2. Scraped Dataset "spotify_audio_feature"

3.2. EDA

To better understand the data, we performed exploratory data analysis on both datasets.

Firstly, we would like to understand the relationship between users and tracks. So we group the tracks and user ID respectively and see the number of the user ID, and in another case, number of the tracks with the visualization of two histograms.

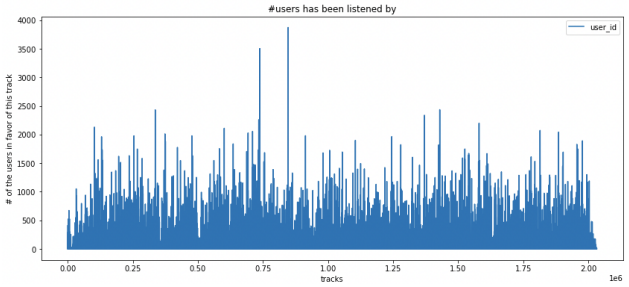


Figure 1. The Number of Times A Track is Listened to by Users

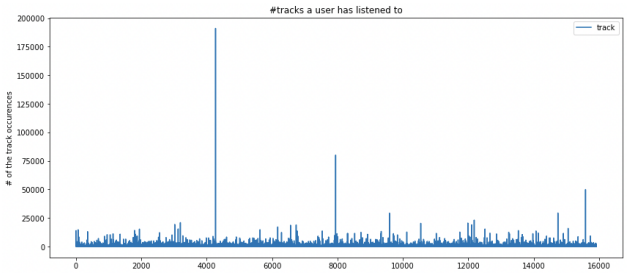


Figure 2. The Number of the Tracks Each User Listened

In Figure 1, we could see the distribution of the favoured tracks is not so skewed, which probably means each kind of music is welcomed by a group of audience, and there are a few tracks that are really on-trend and be in favour by a huge group of users. From Figure 2 We can also know that except for some extreme users, most of the users' music collection numbers are on average.

Furthermore, to understand the distribution more comprehensively, we analyzed and drew Figure 3 and Figure 4 to show the track and user distribution by listening count, it helps us understand more about the general skewness and the extremeness.

Besides, to understand more about audio features' statistics characteristics, according to the data type of the features, we performed a few descriptive analyses and obtained the visualizations below for some of the audio features (from Firgure 5 to Figure 9):

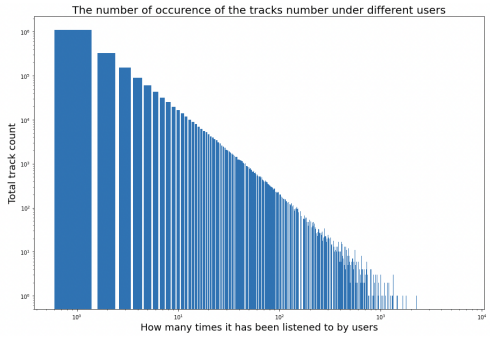


Figure 3. Track Distribution by How Many Time it's Listened

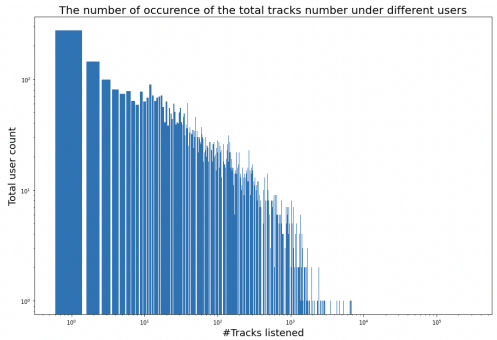


Figure 4. User Distribution by How Many Track they've Listened

- The distribution of the track length fall in a certain range of lengths
- There is significant skewness for duration and loudness histogram distribution
- The keys of the tracks are almost evenly distributed
- Most of the time signature of the anthems are ¾, in mode 0

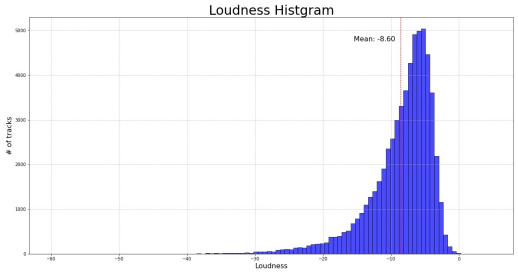


Figure 5. Distribution of Track Loudness

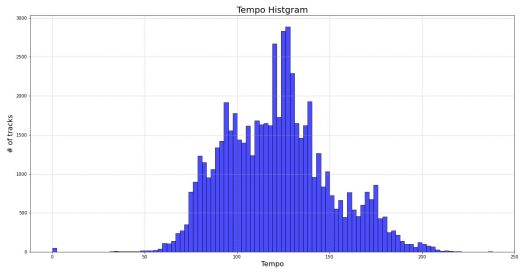


Figure 6. Distribution of Track Tempo

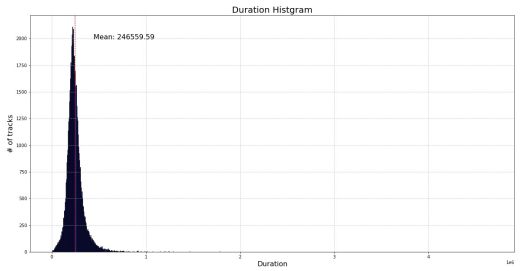


Figure 7. Distribution of Duration

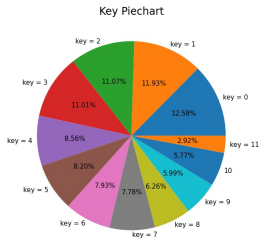


Figure 8. Distribution of Key

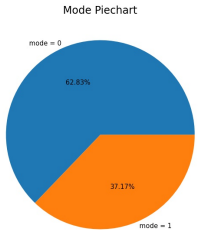


Figure 9. Distribution of Mode

3.3. Data Pre-processing

The original Spotify playlist dataset from Kaggle is more unstructured. To make it more convenient and reduce overall costs for further audio feature crawling as well as model study, the following preprocessing steps have been applied to the dataset:

1. Drop rows which contain null values of any of user_id, artist, track, or playlist
2. Drop rows with unrecognized value under each column
3. Drop rows in which their tracks are no longer available in Spotify with the help of Spotify API
4. Based on our findings in EDA, only rows are kept which meet two criteria: (I) each user has listened to different songs within the range between 100 and 300 (II) each song has been listened to by 500 to 1000 users. In the original dataset, there are a large number of users who only listened to a few songs which are not listened by any other users.It is impossible to find similar users to do the recommendation. So only users and songs with high frequency are selected.

After cleaning, there are 352,665 rows with 2,206 unique users and 1,335 distinct tracks. Then we crawled the audio features of each track using Spotify API. Since the data is precleaned by Spotify for the use of developers, only irrelevant columns are dropped for cleaning.

3.4. Train Test Split

To evaluate and compare the performances between different recommendation models, train test split is applied to the Spotify playlist dataset. For all the users, 80% of the songs they listened to are kept as a train set while the other 20% are used as a test set. Hence, we assume that these songs in the test set are not listened to by those users when training our model. By comparing the recommendation to every user and the test set of them, we could get the performance of the model. Details of evaluation could be found in Section 5.

4. Modelling Evaluation

4.1. Modelling

There are five individual models including Content Based Model, User Based Collaborative Filter Model, NMF Model, SVD Model and Auto-encoder Model. Except for Content Based Model, the other four models work together as a combined model to get the final recommendation. The details of these models will be introduced in the following Sections (4.1.1 to 4.1.5). All five models could make recommenda-

tions separately and their performance is evaluated by Hit Ratio and NDCG. Except for Content Based Model, the other models only used the user-item-matrix where every row is a user vector and every column is an item vector.

For easy comparison, every model recommends 10 songs to every user. Since the recommendations of the combined model are based on scores defined in four different models, these scores are normalized with the following equation:

$$Score_{in} = \frac{Score_i - Score_{min}}{Score_{max} - Score_{min}}$$

$Score_{in}$ is the normalized score of the ith song recommended while $Score_i$ is the original score of the ith song. $Score_{max}$ and $Score_{min}$ are the max score and min score of the songs among the 10 songs recommended to this user.

After this normalization, the scores of 10 songs recommended to every user by every model will be distributed from 0 to 1. Then we multiply the Hit Ratio of every model to their scores and combine the recommendation list from these four models together to get the final result. If a song appears in multiple recommendation lists or in the list of the model having higher Hit Ratio, or the song itself has a high score in a model, it tends to get a better rank in the final recommendations.

4.1.1. Content Based Filtering

Content based filtering is used as a start-up model. This model works under the assumption that users will like songs that have similar characteristics. To start with, K-means is performed on the dataset "spotify_audio_feature" to identify 13 distinct clusters of songs based on various features like danceability, tempo, duration etc. Then for each user in the train set, we identified the top 2 clusters that contain the most number of songs they have liked before, and selected 5 songs closest to the cluster center as the recommendation candidates. These 10 songs are then evaluated against the actual list of songs this user likes from the test set.

Apart from making recommendations, this model could be used to solve cold start which is the most significant problem for the following models because they all rely on the interaction between users and songs but this Content Based Model mainly focuses on the features of songs themselves. Content Based Model will select the song which is closest to the centroid of every cluster (13 songs in total) and recommend them to all the new users. Besides, when a new song is released, feeding it to Content Based Model to predict which cluster it belongs to. Then find out the users liked the songs in this cluster and recommend this new song to these users.

4.1.2. User Based Collaborative Filtering

User Based Collaborative Filtering is utilized to look for a certain number of users similar to the target user, take the songs listened to by these similar users and recommend songs which have never appeared in the targets' playlists.

The initial step is to transpose the user-item matrix, and it takes user_id as columns and song_id as rows to better calculate the Pearson Correlation coefficient. The value contains only 1 and 0, which represents whether or not the user has listened to the song. Based on the Pearson Correlation of users (representing similarity between users), we find out the top three users who have similar tastes with the target user. Then we iterate over each similar user's playlist, for each of the songs which does not appear in the target's playlist, we assign the Pearson Correlation between this user and target user as score to this song. After all iterations, we combine the playlist and take the sum of the score if the song appears in playlists from multiple similar users. For example, if the song appears in the playlist of both similar users A and B, then we sum up the Pearson Correlation of both users as the song's score.

For each target user, we list the final similar song with their scores. The first ten songs obtained from the descending order of the list are regarded as the recommended songs of User Based Collaborative Filtering.

4.1.3. Auto-encoder Model

The code of Auto-encoder Model is taken from the sample code of this course [7] with minor changes. The basic concept of Auto-encoder Model is similar to NMF Model and SVD model. All of them are used to reconstruct the sparse user-item-matrix. This matrix is decomposed to latent space with patterns hidden behind users' tastes. In other words, in the user-item-matrix, a user vector is the interaction between this user with all the songs (1 if listened, 0 if not listened, the dimension of this vector is 1135), in the latent space, this user vector will be some number representing some hidden tastes with lower dimension. In our NMF and SVD Model, we set this dimension to be 8 which means we represent the taste of a user with only 8 hidden features. These 8 hidden features are extracted automatically from his/her interaction with songs. For the Auto-encoder Model, the dimension of latent space is set to be 512.

For NMF and SVD Models, multiplying the user matrix (2206 * 8) and item matrix (8 * 1135) divided by the length of corresponding vector lengths, we could get a matrix (2206 * 1135) with cosine similarity between every user and every song because the cosine similarity is calculated by the following equation:

$$Cos\_Sim_{ij} = \frac{\overrightarrow{u_i} \cdot \overrightarrow{v_j}}{\|\overrightarrow{u_i}\| \cdot \|\overrightarrow{v_j}\|}$$

Cos_Sim_{ij} is the cosine similarity between user i and song j. $\overrightarrow{u_i}$ is the ith user vector while $\overrightarrow{v_j}$ is the jth item vector. $\|\overrightarrow{u_i}\|$ and$\|\overrightarrow{v_j}\|$ are the length of them.

For the Auto-encoder Model, the encode and decode process is done in corresponding layers and the scores representing similarities between users and songs are given in the output layer. Then, the 10 songs with highest scores in every row of the matrix (every user) are taken as the recommendation of the Auto-encoder Model to those users.

Simple hyperparameter tuning is done for Auto-encoder Model, the dimension of encode layer, decode layer and latent space is tuned and the performance is shown below:

Hyperparameter						
Encode Layer Dimension	64	128	256	512	1024	1024
Latent Space	32	64	128	256	512	1024
Decode Layer Dimension	64	128	256	512	1024	1024
Performance						
Hit Ratio	0.273	0.300	0.326	0.373	0.506	0.454
NDCG	0.128	0.148	0.159	0.189	0.270	0.237

It can be seen that the performance is the best when the sizes of encode layer, latent space and decode layer are 1024, 512 and 1024, respectively.

4.1.4. NMF Model

Non-Negative Matrix Factorization, also known as NMF, the basic concept is introduced in Section 4.1.3. After decomposing the user-item-matrix with NMF() from sklearn.decomposition library and getting user-matrix and item-matrix in latent space, multiplying user-matrix and item-matrix and divided by the length of corresponding vector (including user vector and item vector), the cosine similarity matrix is found. Similar to the Auto-encoder Model, the songs with highest similarity in every row are recommended to the corresponding user.

4.1.5. SVD Model

Singular Value Decomposition, also known as SVD, the basic concept is introduced in Section 4.1.3. After decomposing the user-item-matrix with TruncatedSVD() from sklearn.decomposition library and getting user-matrix and item-matrix in latent space, multiplying user-matrix and item-matrix and divided by the length of corresponding vector (including user vector and item vector), the cosine similarity matrix is found. Similar to the Auto-encoder Model, the songs with highest similarity in every row are recommended to the corresponding user.

4.2. Evaluation

4.2.1. Evaluation Metrics

In this report, we choose Hit Ratio and normalized discounted cumulative gain (NDCG) [6], which are commonly used to evaluate the recommendation systems, to be our evaluation metrics.

Hit Ratio: measures the percentage of the users for which at least one of the actual songs are included in the recommendation list with length k among all the users. When the Hit Ratio is higher, more users would be the hit users and could be recommended properly, therefore, the model performance would be better. Below is the formula for the Hit Ratio.

$$\text{Hit Ratio} = \frac{\text{Number of hit users with k recommendations}}{\text{Number of total users}}$$

Normalized discounted cumulative gain (NDCG): For Hit Ratio, as long as we can predict one of the actual songs for a user, the user will be a hit user. However, it couldn't measure the ranking performances of the recommendations. If the right recommendations are located at the top of the recommendation list, the performances should be better.

Therefore, we also use NDCG to evaluate our recommendation system. It can measure not only how relevant the results are but also how good the ordering is. The discounted cumulative gain (DCG) is the sum of relevance up to a position k in the recommendation list discounted by the rank. However, the discounted cumulative gain couldn't be used to compare the systems recommending different numbers of items. This can be solved by dividing the ideal discounted cumulative gain (IDCG), which is the discounted cumulative gain for the most ideal ranking, to obtain the normalized discounted cumulative gain. To obtain the overall NDCG, we take the average of NDCG of the users in the test set. The formulas are shown as below:

$$DCG_i(k) = \sum_{j=1}^{k} \frac{G_j}{log_2(j+1)}$$

$$IDCG_i(k) = \sum_{j=1}^{|I(k))|} \frac{G_j}{log_2(j+1)}$$

$$NDCG_i(k) = \frac{DCG_i(k)}{IDCG_i(k)}$$

$$overallNDCG_i(k) = \frac{1}{N} \cdot \sum_{1}^{N} NDCG_i(k)$$

In these formulas, i represents the individual user, k is the number of the recommendation for each user and j is the position of the recommendation. G is the relevance of the recommended song, which is binary with 0 or 1 in our case.

I(k) means the ideal list of the recommendation and N is the total number of the users.

For both Hit Ratio and NDCG, we choose to recommend 10 songs for each user, therefore, k is 10.

4.2.2. Model Comparison Explanation

The performance of models is demonstrated in the table below:

Model	Hit Ratio	NDCG Score
Content Based Filtering	0.218	0.100
User Based Filtering	0.660	0.371
NMF	0.552	0.288
SVD	0.584	0.299
Auto-encoder	0.481	0.256
Combined	0.573	0.325

Table 3. Model Performance Comparison

It can be seen that the performance of the User-based CF Model is the best in both Hit Ratio and NDCG.

The performance of the Auto-encoder Model is not good although the latent space dimension is much larger than the other two similar models, NMF and SVD, (512 to 8). The possible reason is the limited sample volume because the number of parameters in the Neural Network is huge.

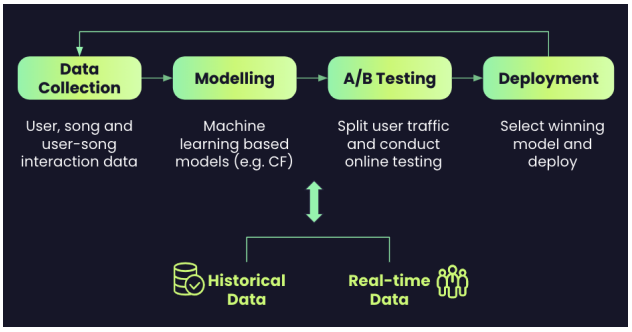
We can also see that the performance of content-based filtering is the lowest among all models. This is expected as the model takes in explicit features like danceability and tempo of songs and makes predictions accordingly. Latent features that determine whether a user likes a song or not cannot be properly understood and leveraged like in collaborative filtering models.

The recommendation considering all the models except Content-based Model is not as good as the best model (User-based CF Model) but is still better than most of the individual models. This is not surprising because the weight of the CF Model is diluted by other models. However, we just tested them on this one dataset. The other models may outperform on another dataset. Hence, this combined model is still meaningful because it could get a more stable performance than individual models.

5. Strategy

After implementing and evaluating the 5 recommendation models above, we proceed to propose how a music company in the real world like Spotify can adopt and implement an effective music recommendation system.

We introduce an iterative workflow below:



- Step 1 - Collect user behaviour data towards songs and song features.
- Step 2 - Perform various machine learning based models illustrated above using data collected from step 1.
- Step 3 - Split user traffic into smaller segments randomly and test the models from step 2 online. Note that in this step, evaluation metrics should also include click through rate and user stay time etc to accurately measure model's impact.
- Step 4 - From online A/B testing in step 3, the best-performing model(s) can be selected for a full traffic deployment.
- Last but not least, this is an iterative workflow, meaning that after the first deployment, continuous data collection, modeling and A/B testing should be performed.

On top of the 4 steps, it is also important to enable both historical and incremental data pipeline so that models can always learn from a comprehensive and latest dataset for optimal performance.

6. Limitations and Further Studies

In this section we will discuss a few limitations in this project and can be explored for further studies :

6.1. Data limitations

There are several limitations from the perspective of the dataset itself. The initial playlist dataset contains more than 12 million pieces of data. Due to the capability as well as the model performance, we only take a subset of the original data to make sure that the subset is not sparse. However, we also add some bias to the subset when we take only a certain range of tracks or users. Moreover, during our exploration, we only take the name of the song into consideration while there exists a situation in which each

song may have different versions. For instance, there is a remastered version of a previous song, but the audio features between each other might be the same. With this saying, it is also possible to take artists into account and regard the same song with different artists as two distinct songs for further exploration.

6.2. Auto-encoder hyperparameter tuning

Only the dimensions of encode, decode layer and latent space are tuned. Many other hyperparameters, such as the optimizer and activation, are not tuned. The Auto-encoder Model may have better performance when it is tuned with more combinations.

7. Conclusion

To conclude, we started off by introducing music recommendation's background and importance, then set out to propose the problem statement of how to build an effective music recommendation system to improve user experience. By using the Spotify datasets on user preference and song features, 5 machine learning based recommendation models and 1 combined model are implemented and evaluated using Hit Ratio and NDCG score. And used based collaborative filtering is identified as the best-performing model.

Limitations around data and the auto-encoder model are also discussed to shed some light onto the next steps. We finally conclude the project with a proposed workflow for online music providers like Spotify to adopt and iteratively improve their recommendation.

The code implementation of this report can be found on this link: https://github.com/ningtang0224/bt5153_group_6

8. References

[1] How Does Spotify Know You So Well? How does Spotify know you so well? (n.d.). Retrieved April 24, 2022, from https://scribe.rip/p/spotifys-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe

[2] About Spotify. Spotify. (2022, February 2). Retrieved February 17, 2022, from https://newsroom.spotify.com/company-info

[3] Spotify users have spent over 2.3 billion hours streaming discover weekly playlists since 2015. Spotify. (2020, July 9) Retrieved February 17, 2022, from https://newsroom.spotify.com/2020-07-09/spotify-users-have-spent-over-2-3-billion-hours-streaming-discover-weekly-playlists-since-2015

[4] Larxel. (2021, November 15). Spotify playlists. Kaggle. Retrieved February 17, 2022, from https://www.kaggle.com/andrewmvd/spotify-playlists

[5] Home: Spotify for developers. Home — Spotify for Developers. (n.d.). Retrieved February 17, 2022, from https://developer.spotify.com/

[6] Ranking Evaluation Metrics for Recommender Systems. Retrieved April 24, 2022, from https://towardsdatascience.com/ranking-evaluation-metrics-for-recommender-systems-263d0a66ef54

[7] BT5153 Applied Machine Learning for Business Analytics - Lecture 5 AutoEncoder. from https://bt5153msba.github.io/note/blogs05.html