Recommending New Artists to Users of Music Streaming Services

Aleksandr Timofeev
Natasa Krco
Pierre Schutz
{name}.{surname}@epfl.ch
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

1 INTRODUCTION

In this report, we present our work on the Machine Learning for Behavioural Data (CS-421) project in the spring 2021 semester. The project work corresponds to the *Track 3 - Music*, and our group ID is G01. The project concerns the LastFM dataset, which contains behavioral and demographic information on users of music streaming services[5]. We will use this data to construct models that recommend new artists to a user.

Research Question. We aimed to use the dataset briefly described above to perform the following task: For a given user, recommend a list of k artists that the user has not previously listened to. To do this, we first estimate the probability that the given user will listen to the music of each artist, and based on these probabilities, create a list of the top k artists the user has not listened to yet.

Business Value. Music streaming platforms often want to help their users discover new artists and songs (for example, the Spotify Discover Weekly playlist [3]). Also, for a new customer, the platforms want to propose the best artists without having much information about the user's listening history. To answer the research question, we built a solution for artist recommendations that could enable the platforms to propose adapted artist selection for their customers. An extension of the project is to recommending songs and generating playlists. Predicting which artists a user would like can also be used for advertisement (for example, promote an artist's concert).

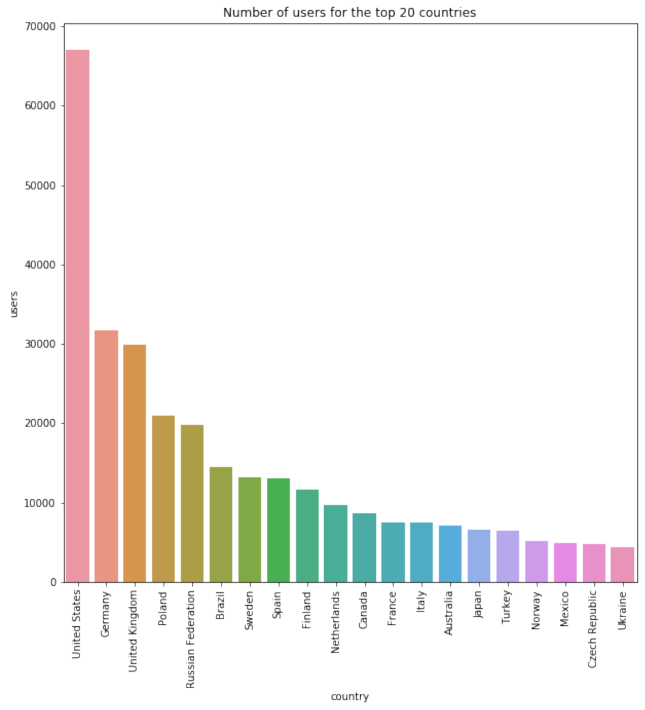

Figure 1: Country distribution of users

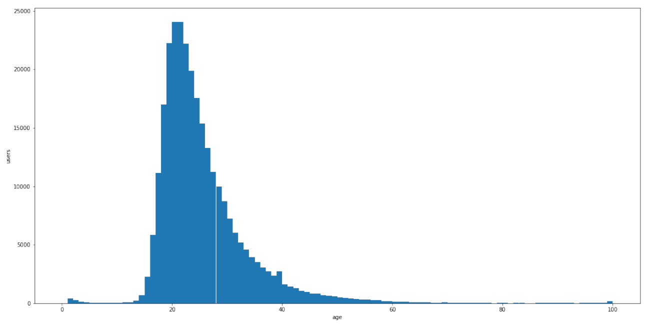

Figure 2: Age distribution of users

2 DATASET

2.1 Data description

The LastFM dataset contains behavioral and demographic information on 360000 users, with observations between 2002 and 2010. The behavioral data consists of a user-artist matrix (with 360000 users and 160000 artists). Each interaction is the number of times a user has played the music of the corresponding artist. This interaction is a tuple (user id, artist id, artist name, number of plays). The resulting matrix contains 17.5 million interactions and has high sparsity - 99.97%. Finally, the demographic dataset provides, for each user, information on gender, age, country, and date of signup.

2.2 Exploratory Data Analysis

The LastFM data originally have two different datasets. The `LastFM1k` have detailed behavioral information (each play is recorded with a timestamp, artist name, and track name) for one thousand users. The `LastFM360k`, contains less detailed behavioral information (total number of plays per artist for each user) but a larger number of different users (360000 users).

During the exploration process, we changed and precise our research questions as we had to chose the most adapted data to run our models. The main trade-off is to choose between a larger number of users against details in the behavioral data. LastFM360 was the most adapted dataset for our final research question (recommend a list of artists to a user) because of the higher number of users and we didn't require the details of the songs. The data being extremely sparse, the high number of user increase our chances of finding similar users but also find a wider range of preferences.

Through this exploration, we tried to answer the following questions: How do behavioral parameters can influence user preferences? How to recommend a diverse range of artists (not only the most famous)?

Demographic: We start observing what features in our demographic dataset have a strong impact on the user's artist preferences. The age and gender parameters contain respectively 21% and 9% of missing values. We observe that the dataset is heavily unbalanced on age with 26 females for 74 males. Looking at the age distribution (Figure. 2), we can see that most users are young (> 90% between 18 and 30), the median being 23 y/o. Regarding the countries, we have data from 239 countries in total. Most users are in North America and Europe, the Top 20 countries contain 82% of the users (Figure. 1). Finally, the signup date distribution (Figure. 4) shows an increasing number of users over the years with the largest growth in 2008 (corresponding to the last full year before data collection).

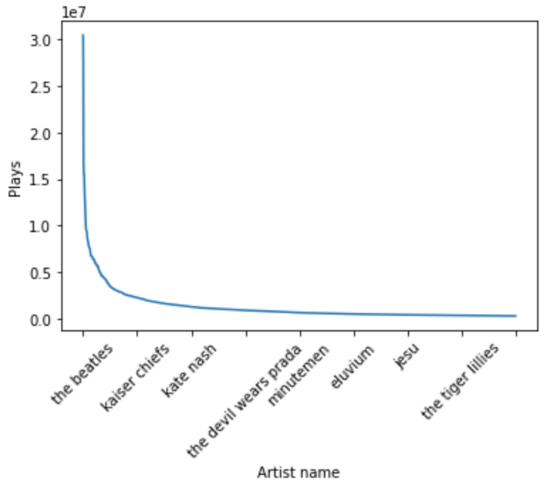

Figure 3: Distribution of plays over artists

Behavioral: First we present the distribution of plays overall artists (Figure. 3). This distribution demonstrates the importance of adapted recommender systems for music streaming platforms. The distribution is long-tailed (there are many plays by famous artists, and a few plays for less popular ones). This might lead to a popularity bias when people listen to only popular artists. The adequate recommender system should handle this problem by also proposing less popular artists to users.

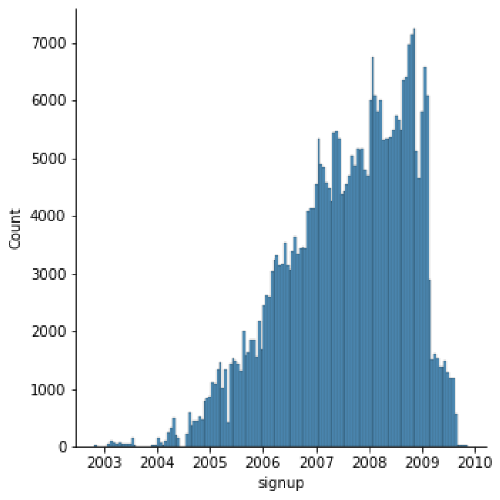

Figure 4: Sign-up date distribution of users

Then, to compare the impact of the demographic features, we use the Kendall rank correlation coefficient [4] measuring the ordinal association between users and artists. For all features, we computed the Kendall similarity for the Top20 artists. The similarity between male and female top artists is 0.18. To observe the similarity in function of the age, we divide the users into groups of 5 years and compare each group together. In Figure. 6, we can observe differences between the groups, but we can observe most of the famous artists. Nevertheless, we do not see an increase in the difference between groups increase the difference of age. We can also observe that the most popular artists are represented in most chunks. Finally, we observe the similarities between the Top20 countries (Figure. 5). Overall, there is a low similarity between the rankings or two different countries. More generally, we note that the Kendall tau metric is sensitive. Due to the high impact from the features in the Top20 rankings, the correlation between groups tends to have small values.

Conclusion: We observe a strong influence of all demographic features on the users' preferred artists. This can be helpful to find adapted neighbors or perform user clustering. We also see that we cannot use directly the number of plays per artist for a recommender system, due to the popularity bias and will need some preprocessing.

2.3 Preprocessing

Dataset reduction: First of all, we found that our computational power is not enough to run training with the complete dataset. Thus, we decided to focus on the USA. It is the biggest country by number of users. This reduction resulted in the dataset with 67000 users, 85000 artists, 3.3 million interactions, and 99.94% sparsity. This new dataset pretty closely replicates the original distribution (Figures 7 and 8), meaning that we can repeat obtained results with the larger dataset if we have more powerful machines.

Preprocessing demographic data. To clean the demographic data we carry out the following steps:

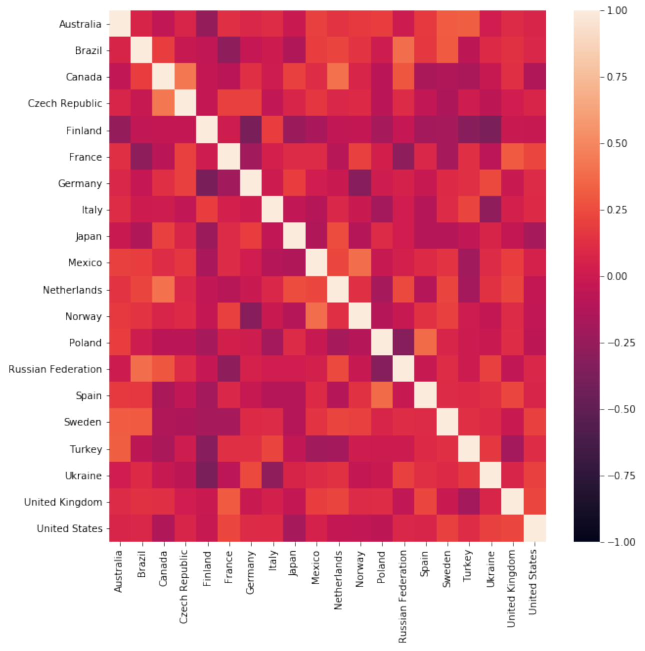

Figure 5: Kendall similarity of Top20 artists per country

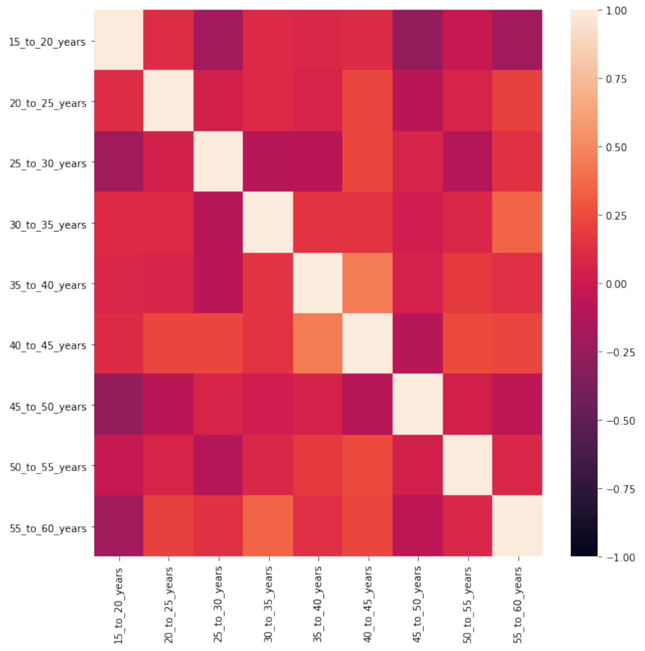

Figure 6: Kendall similarity of Top20 artists per 5 years age group

- Initially, the users are referenced by their emails which are encoded by hashes (long strings). We don't need these hashes, however, they occupy a lot of memory. Thus, we transform them into int numbers. Using the same mapping from hashes to int numbers, we transform the same column in behavioral data.
- During our exploratory analysis, we discovered many weird examples of ages. To avoid these weird values and don't make wrong guesses about them, we allow only values between 8 and 99. All other values are set to NaN. Afterward, this column is one-hot encoded.
- 'Gender' column contains three values: 'm', 'f', and NaN. Again, with the hope that NaNs are systematic, we simply one-hot encode this column.
- We extract four columns from the signup date column: year, month, day, and weekday. This may help to aggregate users in the cohorts. Inside these cohorts, users may demonstrate similar behavioral patterns. For example, users who joined during the promotion campaign.

Preprocessing behavioral data. We found many more issues in the behavioral data. Thus, this part includes not only preprocessing but also cleaning:

- In the same fashion as for user emails, we transform artist IDs, which are hashes, into int numbers.
- We discovered 35 without artist names. Since it is not much, we simply drop it.
- Similarly, 1% of interactions are without artist IDs and, hence we drop them.
- Some artist names are written differently for the same IDs. To make it consistent, we give obe of them to all corresponding samples.
- We also conduct small cleaning of inactive users which may induce bias in the following ML pipeline. The rule is that the number of plays is equal to the number of different listened to artists. There are 18 such users.
- After the previous preprocessing steps, we find the number that the number of users in the demographic data is bigger than in the behavioral one. Thus, we drop those users who don't have interactions with artists.

Totally, we lose 2% of interactions from the behavioral data which is not much.

Preprocessing numbers of plays. In this task, we want to rank the artists for each user and recommend based on it the list of artists. The models which are applied in this work don't do direct ranking. They need to give ratings for some set of artists and using these ratings we can obtain a ranked list of artists. Our dataset has only the number of plays between users and artists which is implicit feedback. Thus, we have many samples where this number is equal to few plays (Figures 9 and 3). It causes low plays overfitting and models predict similar values to all interactions which makes a fair ranking of artists complicated.

One idea which may help here is to apply the logarithm transformation which makes the long-tailed distribution (Figure 9) closer to the normal one (Figure 8). This leads to shifting predicted values closer to the mean, however, it doesn't increase the variance of predictions and even decreases it since the logarithmic function squeezes values. Therefore, this doesn't solve the problem.

A more useful approach is to transfer the task from the implicit to the explicit domain. To this end, we normalize the number of

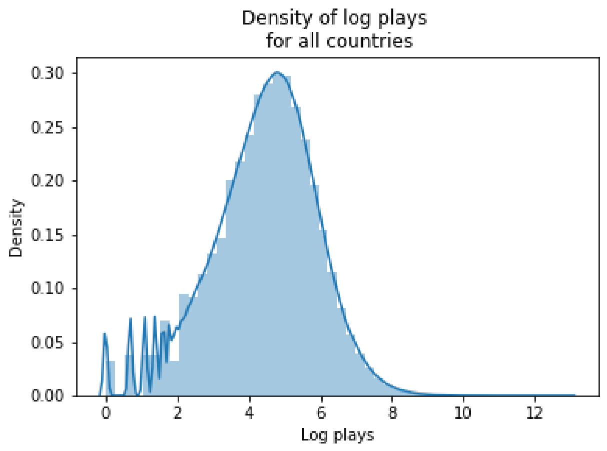

Figure 7: Density of logarithmic plays for the full dataset

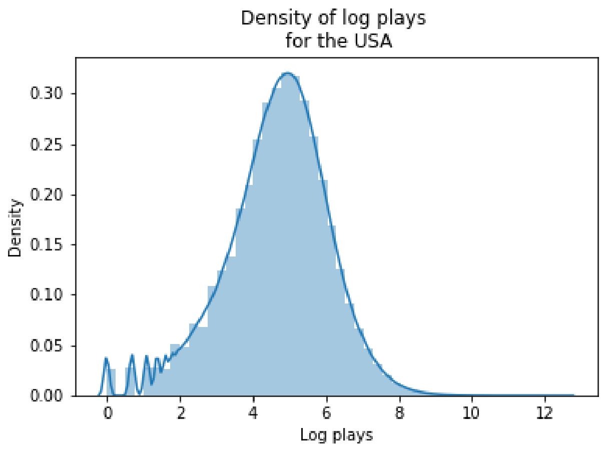

Figure 8: Density of logarithmic plays for USA users

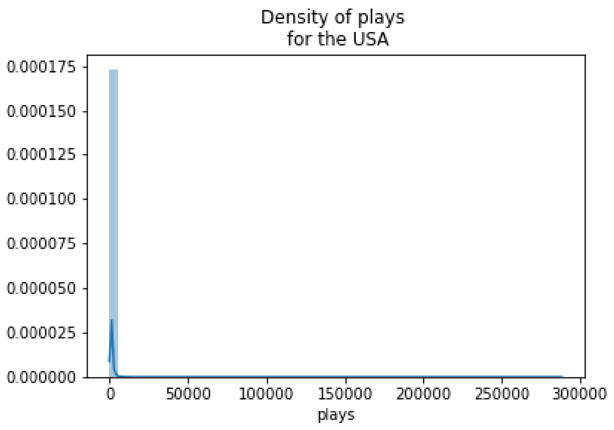

Figure 9: Density of plays for the USA

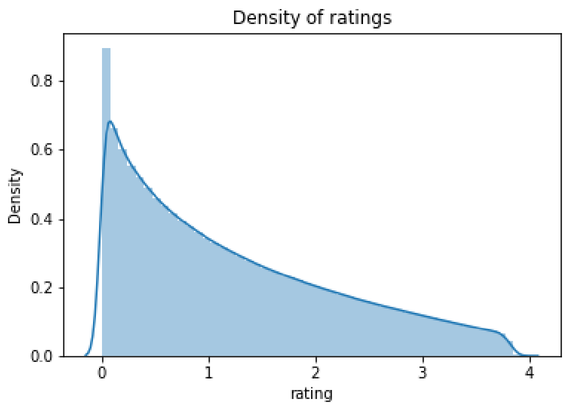

Figure 10: Distribution of ratings

plays for each user. This is important because we should consider users' behavior with regard to themselves. Furthermore, in this case, each user gives at least one the highest rating which increase dataset variance. It is worth noting that the meaning of these values is the frequency of listening to an artist by a given user. The second step is to sort artists for each user according to their frequencies. To obtain ratings, we use this formula:

$$r_{i,k} = 4\left(1 - \sum_{k'=1}^{k} freq_{k'}(i)\right), \quad \forall i \in U,$$

where k denotes a rank of the artist for a given user, U is the set of users. This transforms the number of plays into ratings between 0 and 4. The distribution of rating is depicted in Figure 10. It is worth noting that our models don't use these ratings directly. The target variable for all models is a binary which reflects whether a user and an artist interacted. Ratings are used for other purposes which are explained in Section 3.

2.4 Train-test split

To evaluate our models, we split the dataset into three parts: train, validation, and test. We don't use the cross-validation approach because of the dataset size. This approach would be computationally intensive.

Firstly, the dataset is split into train and test. We hold out 10% randomly chosen interactions for each user. Therefore, we have all users in both parts except cases when users aren't active enough (they have a small number of interactions). Secondly, the same split is performed with the train part to obtain a validation set. As usual, we train a model using the train set, adjust hyper-parameters based on the validation set, and assess the performance of our models using the test set.

Another important point is that the current sets contain only positive samples, i.e. interactions. This differs the implicit feedback from the explicit one. We don't have a negative signal from users. To create this signal artificially and to add variance in our dataset (which should be the remedy for low plays overfitting), we add negative samples. In our case, all pairs of users and artists who didn't interact together serve as negative samples. We give a 0 rating to all these pairs. Initially, we add the same number of negative samples as positive ones to train and validation sets. The test set is needed to assess how our models recommend artists. We don't want it to be simple. Thus, we add nine times more negative samples as positive ones in the train set. [1]

3 MODELS

3.1 Baseline Model

To evaluate the performance of our approaches, we built a trivial model to serve as a baseline. The goal was to reproduce a simple recommendation system. From our perspective, the simplest way to recommend artists is to propose the most famous artist overall. We expect them to have a high chance to be liked by most (as they are by definition popular). Therefore, our baseline model solution consists of finding the k most listened artist (by the number of users) and return them as a list, regardless of the user. The advantages of this model are its simpleness and speed to compute predictions. Nevertheless, it doesn't tackle the problem of popularity bias discussed in the data exploration.

Our following approaches try to solve this issue and propose more personalized recommendations.

3.2 User-User Neighborhood Model (UUNM)

This first model uses neighbors' data to predict the ratings for artists new to a user. These neighbors correspond to the 100 users with the most similar train interactions.

Finding user's neighbors. Given a training dataset with n users that have interactions with m artists, and the total number of interaction i_{train}. The train dataset is a matrix $\mathbf{D} \in \mathbb{R}^{i_{train} \times 3}$ with [user, artist, rating] interactions. To compare user's, we build a matrix $\mathbf{I}$ such that $\mathbf{I} \in \{0,1\}^{n \times m}$ is the indicator function for (users, artists) pair interaction ($\mathbf{I}[u, a]$ if the user u has at least 1 play from of music of artist a, 0 otherwise). We can then compute the similarity between two users using Pearson's correlation. For a user pair u_x and u_y, we have:

$$sim(u_x, u_y) = \frac{\sum_{j=1}^{m}(\mathbf{I}[x,j] - \overline{\mathbf{I}[x]})(\mathbf{I}[y,j] - \overline{\mathbf{I}[y]})}{\sqrt{\sum_{j=1}^{m}(\mathbf{I}[x,j] - \overline{\mathbf{I}[x]})^2 \sum_{j=1}^{m}(\mathbf{I}[y,j] - \overline{\mathbf{I}[y]})^2}} \tag{1}$$

Our resulting model corresponds to a matrix $\mathbf{C} \in \mathbb{R}^{n \times 100}$ with for each user the correlation value of the 100 closest neighbors (with highest Pearson's correlation).

Computing predictions. Given a test dataset with n users, m artists, and i_{test} interactions, we compute the prediction matrix $\mathbf{P}$ such that $\mathbf{P} \in \mathbb{R}^{i_{test}, 3}$ with columns user, artist, and prediction. For a user u_x, an artist a_j, and a set N_x of u_x top 100 neighbors. The predicted rating is:

$$p_{x,j} = \overline{r_x} + \frac{\sum\limits_{k \in N_x} \mathbf{C}[x,k] * (\overline{r_k} - r_{k,j})}{\sum\limits_{k \in N_x} \mathbf{C}[x,k]} \tag{2}$$

[1]We chose this number due to memory and time restrictions

With $\overline{r_x}$ the average rating for user u_x in train data, $\mathbf{C}[x,k]$ the correlation between user u_x and u_k, and $r_{k,j}$ the rating for user u_k and artist a_j.

The result of the user-user neighborhood recommender is a list of the highest predicted ratings for a given user.

Time Optimization. This model turned out to be very time-consuming to compute the similarity matrix or to compute the predictions. We discuss here the few methods implemented to improve the speed of these. We started by computing different sets of neighbors for each user/artist pair by selecting only users that already listened to the artist to predict. We improved this step by building one set of neighbors per user that remaining the same for all artists. This technique enabled to trade-off time against precision. For the similarity matrix computation, we divided the users into groups using demographic features (country and age) to reduce the size of the resulting matrix. Finally, we also reduced the number of artists taken into account by the model by removing all artists with less than 100 listeners.

3.3 Latent Factor Model (LFM)

The second model that we use is the Latent Factor Model (LFM). We aim to use matrix factorization to estimate the probability of an interaction between a user and a given artist. We build a rating matrix M in which rows represent users and columns represent artists such that element $M[i, j]$ is 1 if an interaction between user i and artist j exists, and 0 otherwise. When factorizing this matrix, we use a sigmoid activation function on the estimated value to obtain a valid probability value.

Hyperparameter tuning: We tune the number of latent factors by performing a grid search over values $\{20, 50, 100, 120, 160\}$. Though 160 latent factors give the highest accuracy, it is also the case with the most overfitting, which is why we use 120 latent factors in further experiments.

Training details: We train the model using the Adam optimizer with a learning rate 0.001, a batch size of 2^{16} elements, and Weighted Cross-Entropy loss over 35 epochs. As models using 2 and 3 times the original amount of negative samples tend to overfit when trained over 35 epochs, we regularize these using early stopping - we train them for 15 epochs.

3.4 Neural Matrix Factorization (NeuMF)

Another model, that we use to rate artists and generate lists of relevant artists simultaneously, is NeuMF which is taken from [2] and adapted to our task.

Our model has three branches: user ID, artist ID, and user features. IDs are transformed into two different embeddings, the features are also transformed from sparse one-hot encoded vectors into two dense embeddings by linear layers. The last one is our novelty. The idea is that it may take into account the neighborhood information implicitly. Our data preprocessing, which is described in Section 2.3, results in that user features are completely one-hot encoded vectors. Therefore, users that have similar feature vectors may have similar embeddings which can help better predictions, especially for less active users and in the cold-start setting. Then, the first embeddings of user branches are concatenated and multiplied element-wise by the first embedding of the artist ID branch. The second embeddings of all branches are concatenated and processed

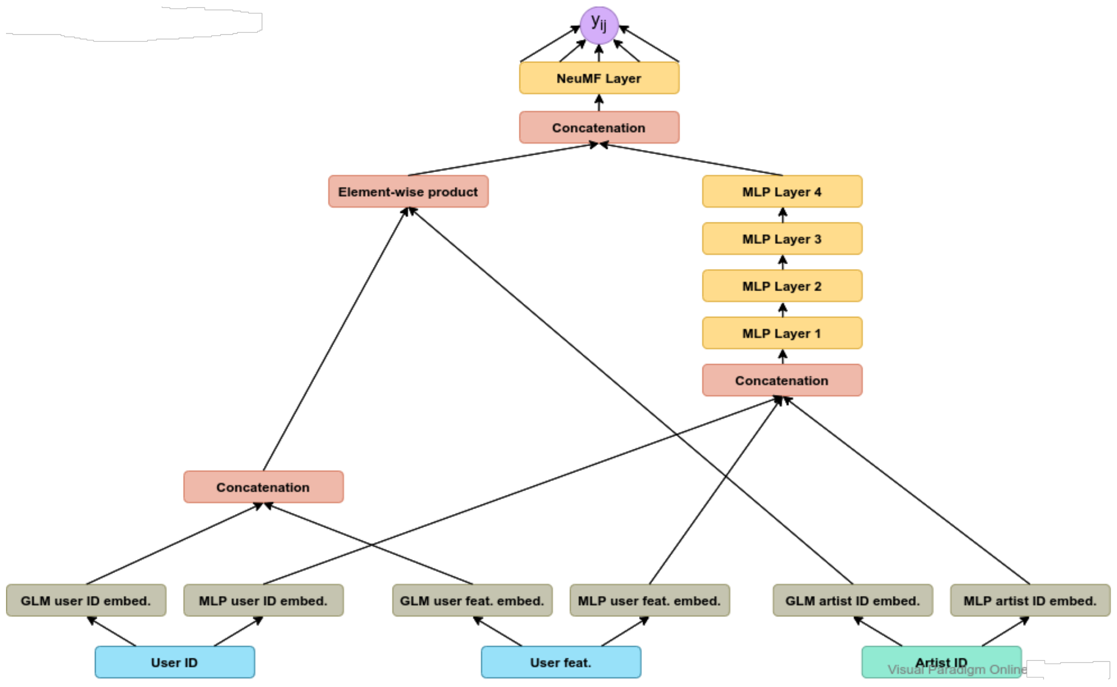

Figure 11: Architecture of NeuMF

by a shallow fully-connected neural network. The neural network output is concatenated to the result of the multiplication and passed together to a linear layer with a sigmoid function which inferences the probability of interaction between a user and an artist. This architecture is summarized in Figure 11.

Training details: To train this model we use the Weighted Cross-Entropy loss and the Adam optimizer with a learning rate of 0.0002 for 18 epochs. The number of latent factors is 16. The batch size is 2^{16}. This size is chosen because of the dataset size. Otherwise, computation takes a long time. To improve results and avoid overfitting we apply dropout at the shallow neural network with the probability parameter 0.5 as well as l_2-regularization with the parameter 0.1 for all layers and embeddings. The activation function is standard - ReLU.

3.5 Training of LFM and NeuMF

As it is aforementioned, to train the last two models, LFM and NeuMF, we use the Weighted Cross-Entropy loss where the target variable is 0 for negative samples (unlistened to artists) and 1 for positive ones (listened to artists). The most important step is to choose the right weights. We needed the weights to give more importance or confidence to artists with the highest ratings. They should help to avoid low plays overfitting.

We try two strategies. The first one is so-called percentile normalization. Specifically, for each user, we split listened artists into four groups based on their ratings.

- *Group 1:* $0 \leq r_{i,j} < 1$
- *Group 2:* $1 \leq r_{i,j} < 2$
- *Group 3:* $2 \leq r_{i,j} < 3$
- *Group 4:* $3 \leq r_{i,j} < 4$

It is worth noting that in this variant, we don't use negative samples and the loss is the Weighted Mean Squared Error. Then, each sample is weighted by the inverse number of artists in its group for each user. Hence samples from small groups have higher weights. However, this approach can't completely help with the low plays overfitting.

Another approach is confidence. Here, we use the described setting with negative samples and the Weighted Binary Cross Entropy. The weights in this case express the confidence in ratings that we give the artist through our transformation from implicit to an explicit domain. The most popular artists have deserved high ratings and we should trust it. But if this artist has a small number of plays, we can think of it as several users tried to listen to the artist and didn't like for example. Therefore the formula for weights is:

$$c_{i,j} = 1 + \alpha r_{i,j}, \quad \forall i \in U, \forall j \in A,$$

where U and A are the sets of users and artists correspondingly and α is a parameter. In our case, we achieve the best results if $\alpha = 0.25$. This approach is used in our experiments and allows overcoming the low plays overfitting.

4 EXPERIMENTAL EVALUATION

4.1 Metrics

To evaluate the models, we use the following metrics:

Root Mean Squared Error (RMSE). The first metric we use is RMSE, to determine the "reconstruction error", that is, the difference between the actual value in the rating matrix, and the predicted value.

Precision @ k. Precision is used to determine the proportion of recommended ratings that the user rated positively.

Recall @ k. Recall is used to determine the proportion of items the user rated positively that the model recommended.

Hit Rate @ k (HR). Hit Rate is calculated by first computing the number of hits for each user. An item is considered a hit if it is recommended to the user, and the user rated it positively. Then, the total number of hits for all users is divided by the number of users in the dataset to obtain the Hit Rate.

Normalized Discounted Cumulative Gain @ k (NDCG). To measure how well the models rank artists, we use NDCG. To calculate the value of the metric, sort the true ratings in order of predicted ranking, divide each by the logarithm of its position and sum up to get the Discounted Cumulative Gain (DCG). Then, divide by the DCG of the ideal order of artists to get NDCG.

Average Reciprocal Hit Rate @ k (ARHR). ARHR is calculated similarly to Hit Rate, but instead of simply summing the number of hits, the ranking of each item is taken into account. The reciprocal value of the rank of each hit is summed up, and the total sum for all users is divided by the number of users to obtain the final value of the metric.

4.2 Results

Basic setting. The results of the test in the basic setting are summarized in Table 1. Besides all described models, we additionally conduct experiments in three modifications of the NeuMF model: without user features branch, with all branches i.e. as it is depicted in Figure 11, without user IDs branch.

First of all, we want to notice that all our models outperform the baseline. Hence, they are meaningful and propose more interesting recommendations than simply the most popular ones. It is seen that the UUNM exhibits the best performance by the metrics RMSE, Precision@10, Recall@10, HR@10, and not too worse by NDCG@10 and ARHR@10. In a meantime, the LFM model is the best of the rest metrics. In addition, the NeuMF model is also slightly better by these metrics than the UUNM. It implies that the UUNM suggests more relevant recommendations while the factorization-based models are better in positioning them. What is important more depends on the application. For example, on one hand, if users play a random list of tracks, they don't want to pay attention to what is playing and all tracks must be relevant otherwise it can be bothering. On the other hand, when users search for a new artist, we can save his/her time placing the most relevant recommendations as high as possible.

Another important trade-off is between Precision@10 and Recall@10. The high precision means that all our recommendations are relevant which again useful in the first example of the previous paragraph with random lists (we don't waste the user time). The high recall implies that a model tries to cover all relevant examples which are useful in the second example when we have time to filter the proposed artists and choose the best one. Because our models recommend the list of artists, we believe that it is more applicable in the second example which requires a higher recall. That is what we observe for all our models in Table 1.

Although all models are better than the baseline, it is hard to say that we give many good recommendations at high positions which should satisfy our users. For example, the UUNM has 3.49 by HR@10 which means that on average each user sees only 3-4 relevant recommendations out of 10 as well as NDCG@10 is 0.67 which implies that the model puts relevant recommendations at the top not often.

The bad performance of the NeuMF model and small variance in the results in dependence on the including/excluding user branches can be explained by the lack of tuning architecture parameters. One layer for the user features branch might be not enough as well as we need to find a better number of layers for the shallow neural network, check different activations, etc.

Despite the performance of the UUNM, we can't proceed in the following experiments with this model since its inference time is too long. The good performance of this model also tells us that considering the neighborhood information is important in this task.

Different percentage of negative samples. Originally, we add the same number of negative samples as positive ones to the train set. The logic is that usually, ML algorithms work better in balanced settings. However, it is better to verify. To this end, we conduct experiments with different percentages of the negative samples for each user in the train set with LFM and NeuMF models. The results are presented in Figure 12.

One may notice that the best performance for the NeuMF is achieved when the percentage of negative samples is about 0.8. The same can be concluded for the LFM. Though, its plots are not monotonic as for the NeuMF. Therefore, our original hypothesis about the same number of negative samples is not correct and the better performance can be achieved with an imbalanced setting. The further increase of the percentage is restricted by memory and time requirements that we can't ensure.

Cold-start. Lastly, because of the NeuMF model structure with the user features branch, we decided to explore how it works in the cold-start setting. Intuitively, this branch should generate for users who are close to those in the train set similar embeddings. Therefore, these embeddings implicitly take into account the neighborhood information and may improve predictions for new users. To this end, we hold out 1000 users to test the cold-start setting. The results are provided in Table 2.

We see that all models again outperform the baseline. The model with both user branches is the best one. However, results differ only slightly which again means that we need better tuning of architecture parameters. It is seen that models are good enough in giving relevant recommendations and positioning them.

5 FUTURE WORK

First of all, we didn't do any feature selections though two of our models are based on user features. For example, it is possible to decrease the number of features by PCA keeping the most important ones. Careful selection may boost the performance of all models at the same time.

Another interesting area is to explore the performance of the models by coverage, diversity, and/or novelty. These metrics are important for building powerful recommending systems. Because we don't want to ignore some not popular artists as well as we should propose something new and surprise users for the sake of their service satisfaction.

Table 1: Comparison of models in the basic setting

Models	RMSE	Precision@10	Recall@10	HR@10	NDCG@10	ARHR@10
Baseline	-	0.02	0.04	0.18	0.09	0.07
UUNM	**0.20**	**0.35**	**0.71**	**3.47**	0.61	1.13
LFM	0.25	0.30	0.62	3.04	**0.67**	**1.24**
NeuMF w/o UF	0.26	0.28	0.59	2.76	0.62	1.14
NeuMF w/ UF	0.25	0.29	0.59	2.91	0.64	1.16
NeuMF only UF	0.25	0.29	0.59	2.90	0.64	1.15

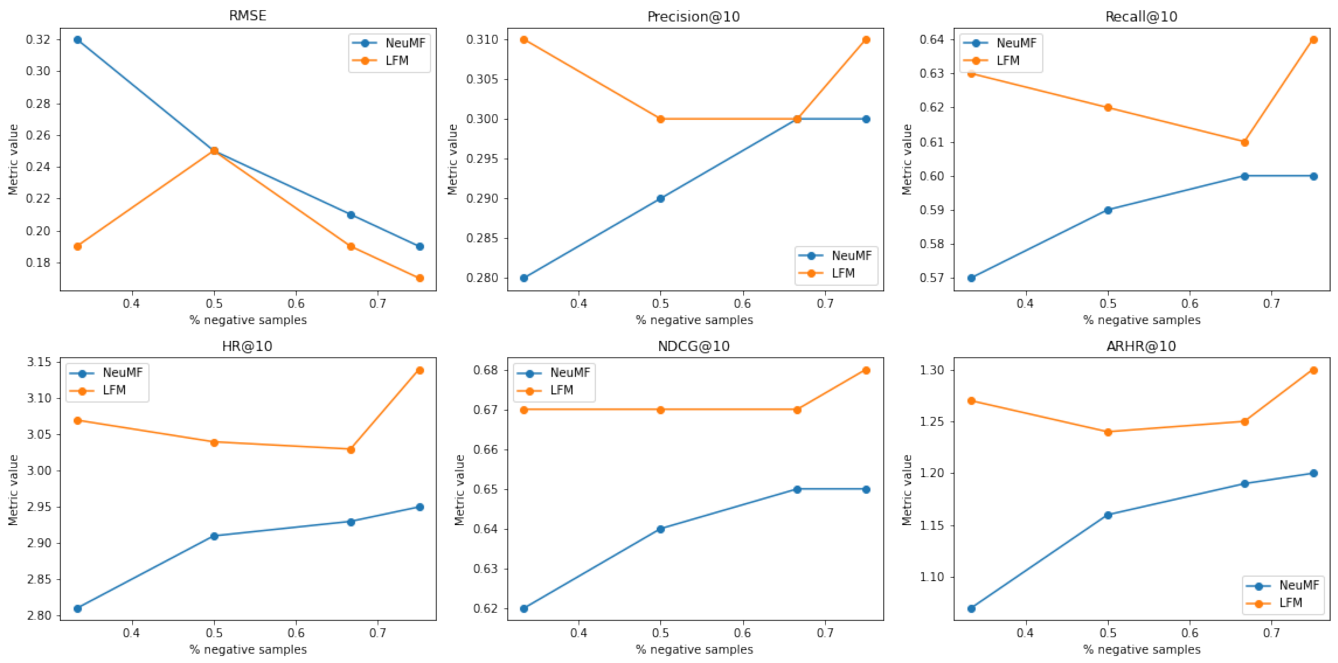

Figure 12: Results of LFM and NeuMF models with different number of negative samples in the test set

Table 2: Results of the NeuMF model in the cold-start setting

Models	RMSE	Precision@10	Recall@10	HR@10	NDCG@10	ARHR@10
Baseline	-	0.18	0.03	1.76	0.47	0.72
NeuMF w/o UF	**0.26**	0.84	0.15	8.10	**0.77**	2.43
NeuMF w/ UF	0.28	**0.86**	**0.16**	**8.57**	**0.77**	**2.53**
NeuMF only UF	0.27	0.85	0.16	8.51	**0.77**	2.51

Time and space optimization should be implemented to improve the User-User Neighborhood model and speed up the prediction process. We can also improve the similarity metric computation using ranking values (instead of binary interactions) or try user-neighborhood learning techniques.

The main task for further work on the LFM model is to use regularization to reduce overfitting. Combined with regularization, tuning the number of latent factors could be done more extensively, as currently the value with which the model achieves the highest accuracy overfits.

Finally, the NeuMF model should be better than at least the LFM. The poor results might be explained by the lack of tuning architecture parameters. Correspondingly, the feature should include strengthening of the user feature branch. Perhaps, it doesn't provide good embedding using one layer. Additionally, it is better to push explicitly the model to create good embeddings through adding auxiliary losses or using an autoencoder structure. Furthermore, it would be nice to explore other parameters of the neural network such as the depth and the width, different activation functions, etc. If this doesn't help, it is possible to include the neighborhood information, which is important according to our experiments, using other approaches. For example, in [1], the neighbors are those who listened to the same artist. This way, we can also use the neighborhood information for artists because we can connect them through the same users. Another approach is to find clusters of users and include a cluster number through embeddings.

6 CONCLUSION

This project goal was to build and evaluate machine learning models for recommending new artists to music streaming services' users. Also, use appropriate methods of data processing and model tuning to improve their performance. All implemented models outperformed the baseline model, which shows that they are performing and produce meaningful results. In the basic setting, the User-User Neighborhood model performed the best but took too long to test. This challenge motivates further work on optimizing the inference time to enable computing new experiments in the other settings that showed improvement for the LFM and NeuFM models. We improved the performance of the LFM and NeuMF models by increasing the number of negative samples, and NeuMF had solid results when tested in the cold-start setting. As there are still many ways to further tune these models, with further work on proper tuning and optimizing, these results could be a good starting point for building a recommender system.

REFERENCES

[1] Ting Bai, Ji-Rong Wen, Jun Zhang, and Wayne Zhao. 2017. A Neural Collaborative Filtering Model with Interaction-based Neighborhood. 1979–1982. https://doi.org/10.1145/3132847.3133083

[2] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[3] Spotify. 2021. Discover Weekly Playlist. https://www.spotify.com/us/discoverweekly/.

[4] Wikipedia. 2021. Kendall rank correlation coefficient. https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient.

[5] Òscar Celma. 2010. Last.fm Dataset 360k users user top artists. http://ocelma.net/MusicRecommendationDataset/lastfm-360K.html.