Master of Science in Applied Information and Data Science

Lucerne University of Applied Sciences and Arts

Course Capstone Project Report

Deezer Music Recommendation

Source: (D-X, 2022)

Students	**Martin Biber, Quyen Duong, Kirsten Tallner**
Experts	**Dr. Guang Lu**
Module	**Recommender Systems**
Date of submission	**21 February 2022**

Contents

Figures

Tables

1 Introduction

1.1 Motivation and research questions

Music streaming has boomed and changed the way people listen to music worldwide (VanDyke, 2021). Numerous well-known companies such as Spotify, Apple Music, Tidal, Amazon Music Unlimited, YouTube Music, and Deezer provide millions of songs from diverse artists charging a subscription fee (Mary Stone, 2022). With the fierce competition, building and implementing an excellent music recommender system is one of the competitive advantages and enables the music provider to predict and suggest the most potential songs to their users (Adiyansjah, Gunawan, & Suhartono, 2019). One of the big players, Deezer, runs the music streaming service in more than 180 countries. Having the music recommendation built right is critical to appeal to and preserve customers. Inspired by the great importance of today's music recommendations, this project referred to a challenge provided on Kaggle between 15/04/2017 and 31/05/2017[1] focused on recommending songs to Deezer users (Kaggle, 2017). The corresponding data is available on the Kaggle website and was additionally provided by Dr. Guang Lu on Ilias (Lu, 2022).

The goals of this project were to get familiar with and build up recommender systems as well as to answer the following research questions:

- What would you do to win this competition?
- What would you do to solve the problem that Deezer is interested in?
- Why might the two solutions not overlap, or why they do?

1.2 Data description

Two datasets were provided: *train.csv* (7'558'834 rows, 15 columns) and *test.csv* (19'918 rows, 15 columns). Both datasets contained the attributes *'genre_id', 'ts_listen', 'media_id', 'album_id', 'context_type', 'release_date', 'platform_name', 'platform_family', 'media_duration', 'listen_type', 'user_gender', 'user_id', 'artist_id', 'user_age'*. The dataset *train.csv* also contained the column for the dependent binary variable *'is_listened'*, while it was absent in *test.csv*. The column '*is_listened*' was encoded as 0 for songs being played not at all or less than 30 seconds, while it was encoded as 1 for songs being played for longer. Furthermore, *test.csv* revealed the column *'sample_id'*. All columns were presented in the integer datatype.

1.3 Methodology

As a first step, an explanatory data analysis (EDA) was performed to get familiar with the given data (see Chapter 2). Data was then prepared for classification and recommendation purposes (see Chapter 3). Next, different recommendation systems were built. In addition, several classification approaches were tried to get a deeper understanding of similarities and differences between those two concepts, which could even be brought together within hybrid recommendation systems. The codes on recommender systems were adapted and based on Dr. Lu's class as well as various other online sources such as *DataCamp*'s courses, *scikit-learn* webpage, *GitHub*, *Udemy*'s courses, and *Towards Data Science*'s articles.

Quality of supervised classifications and recommendations were evaluated by focusing on Receiver Operating Characteristics Area Under the Curve (ROC AUC) score as required in the Kaggle challenge (see under the *Evaluation* section on Kaggle). Besides, other classification metrics such as accuracy, recall, and precision were optionally calculated. The summary of our steps (see Table 1); note that some steps were sequential while others were processed in a more agile manner:

Step	Tasks	Description	Corresponding file
1	EDA, data visualization	Understand the dataset and each variable	main_EDA.jpynb
2	Data preparation	Subsampling, split into train and test data	deezerData.py
3	Fit several models with classification approaches	Get insights into common classification approaches	All files starting with supplement_XXX.jpynb

[1] https://www.kaggle.com/c/dsg17-online-phase/

4	Fit several models with recommendation approaches	Practice and transform learned knowledge from the class into practice, get more understandings of recommendations in real-world business	main_MatrixFactorizationV2.py main_SVD.py main_keras_deepLearning.py main_keras_deepLearning_KerasTuner.py
5	Cross-check for each method and discuss the results	Choose the final techniques that are most relevant in this case	RS_Project_Report.pdf
6	Reflect, research, and write answers to the research questions	Get a bigger picture of the project and music recommender system	RS_Project_Report.pdf
7	Self-reflection	Summarize learned lessons, reflect on the courses, machine learning algorithms, and their application in recommendation	RS_Project_Report.pdf

Table 1: Summary of project steps and their corresponding files

2 Exploratory data analysis

The meaning of each variable and details of EDA can be found under the file *main_EDA.jpynb*. One essential point is that *'user_id'* and *'media_id*' were not present only once in the whole dataset, indicating that one user might be presented in several rows and listened to different songs or several times to the same song. In other words, one observation denotes one user in relation to one media at a certain timestamp. Hence, there are several combinations of ID variables.

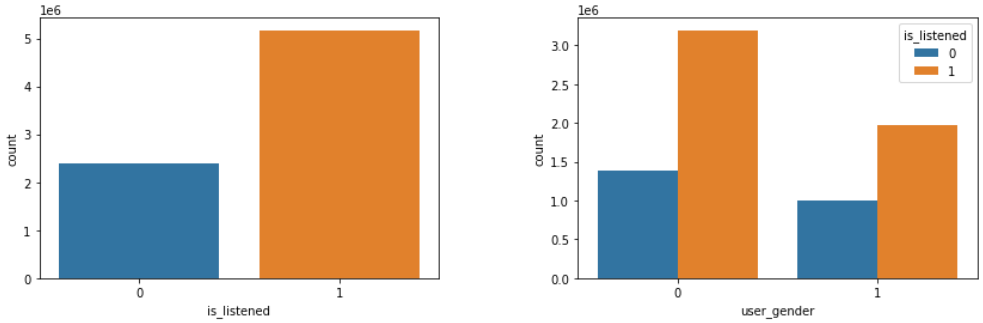

Figure 1: Number of is_listened (left), and the bar charts of user_gender categorized by is_listened (right)

All variables with ID such as 'user_id', 'media_id', 'album_id', and 'artist_id' can be considered factors or ordinal data types, which uniquely present for each entity. Nevertheless, we kept them as integer during the EDA process. The train set contains no missing value. In total, there are 19'918 unique ‘user_id’, 2922 ‘genre_id’, 452'965 ‘media_id’ (i.e., unique songs), 151'464 albums, 67'135 artists, 3 platforms, and 13 different user ages. The ID values can be very high, so it is crucial to scale columns before applying any machine learning model. The types of scaling, such as StandardScaler, or column-wise scaler, were applied according to each model.

Next, the variance inflation factor (VIF) above 10 in general signifies the presence of high multicollinearity, and VIF between 5 to 10 should also be examined further. In this case, the 'album_id' and 'media_id' were highly colinear but dropping the 'album_id' eliminated this problem. Also, it was more reasonable to keep 'media_id' because of its importance for the recommendation itself. About the response variable 'is_listened', the number of class 1 (~5.1 million observations) is twice as high as that of class 0 (~2.3 million observations) (see Figure 1, left). Since the gender assignment values 0 and 1 in the 'user_gender' column were not explained in Kaggle's challenge description, we could only detect more users with gender 0 than for gender 1. Moreover, gender 0 tended to listen to the suggested song rather than gender 1 (Figure 1, right). The user age varied from 18 (minimum) to 30 (maximum) years (see Figure 2, left). Moreover, Figure 2 (on the right) exhibits the ratio of whether users listen to the songs concerning their age. The orange bars (decoded as 1) denote listening, while

the blue bars (decoded as 0) presented skipping the song. The proportion between the two colors changed when the user's age increased. Thus, younger users seemed to skip the

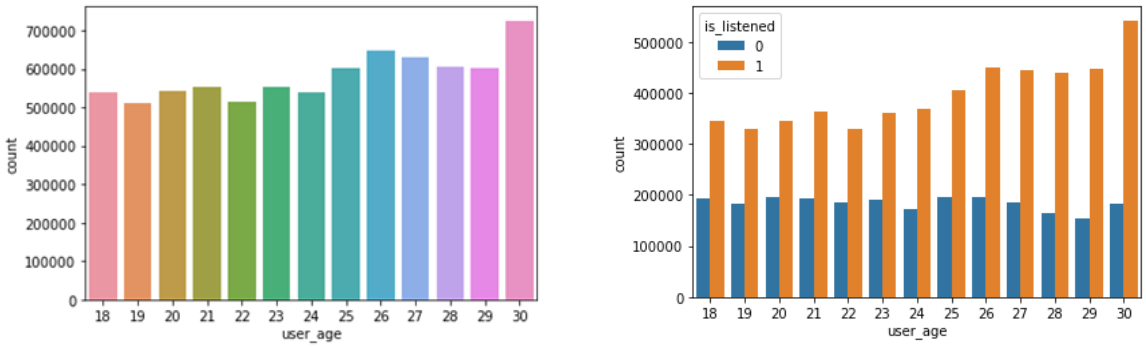

Figure 2: User age of Deezer users, in general (left), separated by implicit user rating 'is_listened' (right).

Besides, several 3D graphs and heatmap correlation plot for all variable pairs were generated, but no critical pattern was found. Also, the more complex visualizations of an enormous dataset were very time-consuming or required much computational power (e.g., it was impossible to generate a Seaborn pair plot because of a too big array).

3 Data preparation

Because *test.csv* revealed the additional column *'sample_id'* and we were allowed to do the following for this project, *train.csv* was split into the train (80%) and test (20%) data.

As different classification/recommendation approaches were tried, data subsampling was harmonized to first, make the results comparable if possible, and second, to share a single file for collaborative project work. For this, data import and subsampling were defined within a separate python file ('deezerData.py'), which was called from various classification or recommendation scripts.

To shorten calculation time and reduce performance issues, the data imported from *train.csv* was reduced to 12'923 rows and 15 columns in total. With this sample size, fast execution of the code proved code operability and might be expanded to bigger sample sizes in future projects. Note that you could also run the data import for the whole data (without creating a subsample) just by setting the optional parameter "subsample" to False in the script 'deezerData.py'. However, it is not advised to use the whole dataset due to performance issues on our private computers.

The subsample itself consisted of the first user IDs being present in the dataset, which might have resulted in higher numbers of recommendations per user being present in the subsample. Note that this subsample might not be the most representative for this dataset but was taken for demonstration purposes. As an additional feature, the 'readData' function in 'deezerData.py' could be called by setting the optional parameter 'groupBy' to 'True' to convert the (subsampled) data into a grouped structure. This grouped data was needed for 'main_keras_deepLearning.py', which did not run with duplicated IDs. However, grouping was not strictly necessary for supplemental classification approaches.

For the recommendation systems, original data was taken as it was, not being further preprocessed. Potential data clean-up could be performed, e.g., on the 'release_date' column, which revealed unrealistic release dates in the far future ('30000101' of format YYYYMMDD, meaning 01 Jan 3000). Time-specific features such as year, month, weekday, and hour could conceivably be created from the timestamp when the songs were listened ('ts_listen'). Furthermore, due to initial data exploration, columns showing high multicollinearity could be removed, such as 'album_id', which highly correlated with 'media_id'. For potential data clean-up and transformation steps, please refer to Chapter 2.

4 Fitting and evaluating models

4.1 Classification approach

Before applying the recommendation approach, several machine learning techniques for binary classification were accomplished: Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Tree, and Random Forest. Although being fully aware that this supervised learning approach was not our main focus in this project, it still served two purposes: (1) predict the value for 'is_listened' to win the Kaggle challenge and (2) get a better comprehension of similarities and differences between two different approaches.

Due to the page limitation, we do not explain each machine learning technique in detail, but they can be found in separated scripts starting with "supplement_techniquename.ipynb". The general procedure of all classification methods began with building a pipeline for scaling and initiating classifiers such as KneighborsClassifier() or SVM(). The Decision Tree model did not require feature scaling since they were not sensitive to variance (Thenraj, 2020). Secondly, several hyperparameters were tuned with different estimation tools, namely *GridSearchCV*, *RandomSearchCV*, or *BayesSearchCV*. Next, models were fitted according to these best parameters evaluated on the train data, followed by prediction on the test data. Lastly, the evaluation of the models by using metrics such as ROC AUC and accuracy for each technique was recorded. Note that each machine learning method might change slightly, but the result and summary are displayed in Table 2:

Method name	ROC AUC	Accuracy	Tuned Hyperparameters	Estimation tool	Cross validation
Decision Tree	0.74	0.77	• Maximum depth of tree • Maximum features • Minimum samples leaf • Criterion: gini or entropy	RandomizedSearchCV	✓
K-Nearest Neighbor	0.73	0.76	• Number of neighbors	GridSearchCV	✓
Random Forest	0.73	0.78	• Number of estimators n_estimators • Maximum depth • Criterion: gini or entropy	BayesSearchCV	✓
Support Vector Machine with kernel sigmoid	0.61	0.65	• Cost C • Gamma • Kernel	GridSearchCV	✓
Logistic regression	0.56	0.67	• Regularization parameter C • Penalty for regularization technique of Lasso (L1) or Ridge (L2)	GridSearchCV	✓
Support Vector Machine with kernel rbf	0.50	0.67	• Cost C • Gamma • Kernel	GridSearchCV	✓

Table 2: Classification approach's summary

4.2 Recommendation approach

4.2.1 Matrix factorization

Matrix factorization is a classification method (Wikipedia, 2022) that was introduced by Simon Funk in his blog post (Funk, 2016). He knew that he could divide a matrix with singular value decomposition (SVD) into smaller matrices. This was established knowledge at that time. The problem was that this procedure was well known for a matrix without missing values.

Simon proposed the idea to only use the available entries of the Matrix and use an iterative approach to minimize the error for those values. The sigma matrix of the SVD is basically only scaling one of the two other matrices. It is therefore assumed that this sigma matrix is included in one of the two matrices. It is therefore

possible to calculate the rating matrix with just two matrices P and Q. The size of the two matrices is defined by the rows of the R-matrix and columns which correspond to the users and items respectively. The second dimension of the two matrices are the latent factors (K) which can be chosen by the user. The smaller the number the larger the size reduction of the matrices.

The two matrices are changed as long as the errors for the existing values can be further reduced. The two matrices have no missing values because they are filled with random numbers at the beginning. With that it is possible to calculate the missing values in the original matrix.

The used script is closely related to the medium article from Andre Ye (Ye, 2020) and the Wiki from the University of British Columbia. The imported data frame is being converted into a pivot table (R-matrix) with "user_id" as index and "media_id" as features. The table is filled with the values from "is_listened". The NaN's are filled with -1. The latent factor was two which is a very low number to keep the calculation time as low as possible. The two matrices P and Q are then filled with random numbers. The function "matrix_factorization" calculates the values for the known entries and uses the difference to the true value to calculate the error. This error is then used to adjust the values in P and Q. The adjustment formula is a simplification of a gradient calculation. This iteration is done 100 times per default.

The first figure shows the errors for 500 iterations. This resulted in an auc of 0.83 comparing R-hat with the original R-matrix. The middle figure shows the error with 20 latent factors and 100 iterations, and the last figure shows the error for 20 latent factors and 500 iterations. Middle and right figures result in an AUC of 0.85. It, therefore, seems that the latent factors have a higher impact on the AUC.

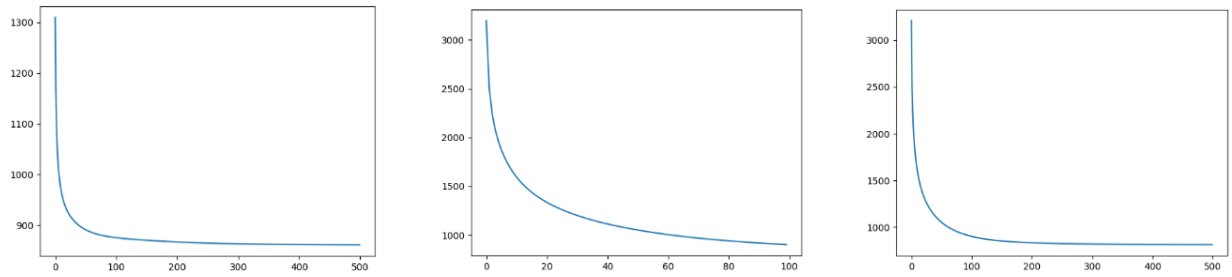

Figure 3 Error development in relation number of iterations. Two latent factors (left) and twenty (middle and right)

The missing values in the R-matrix can now be calculated with the two optimized matrices, P and Q. Matrix factorization allows therefore to calculate the rating for all users' items combination.

The approach described above could be improved in several ways. Most importantly, the gradient descent should be changed to reduce the error further and faster. To get a better AUC, the latent factors should be increased. These optimizations were not done and would be an essential step if the result were uploaded to Kaggle. There are python packages that allow using matrix factorization much easier.

One problem with this method is that the matrix size grows extremely fast, which can lead to the issue that not all user-item combinations can be calculated in one go.

4.2.2 SVD / SVD++

SVD is a compelling method. It can be used in very different fields. Some of them are mentioned in Steven Brunton and Nathan Kutz (Brunton & Kutz, 2019). We chose the proposed python package *Surprise* in contrast to the matrix factorization in the previous subchapter. There are other python packages around for this method like the very well-known SciPy package. Both methods were used for our data subset. While SVD was also successfully tested on the whole train file, the SVD++ did not give back a result even after several hours of calculation.

The functions work without giving any specific parameters, as seen in the script. The SVD++ uses additional factors (Wikipedia 2022). The results of our data set only show a small difference between the two methods (RMSE: 0.3682 vs 0.3647). As we could not run it on the whole data, we cannot say how different the calculation speed and the accuracy metrics are.

If we were to participate in the Kaggle competition, we had to investigate the SVD++ further as a result was better, which is the only factor that counts for Kaggle. The advice for Deezer would certainly be to go with the SVD, which is faster, and the performance is similar.

4.2.3 Deep learning using Keras Model API

For this approach, two files were created that share similar code but differ in hyperparameter tuning: ‘main_keras_deepLearning.py’ (no hyperparameter tuning) and ‘main_keras_deepLearning_KerasTuner.py’ (hyperparameter tuning). As basis, the script '*mf_keras_deep.py*' (which was provided by the lecturer in section '*Recommender System_Lu_Day4_Part2/RSDLP_Code*') was used.

Code was restructured and put into different functions. Dataframes were constructed for output: One for summarizing true and predicted values for the dependent variable *'is_listened*' and the other one for the recommendations to a given user themselves (see Figure 4). In addition, the code was also expanded to print model quality metrics for the last epoch.

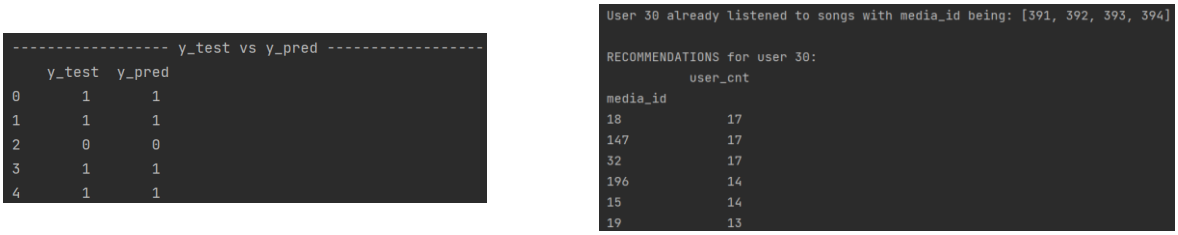

Figure 4: Console output for predicted and true response variable (left), recommendation for user (right).

Data was imported by calling the 'readData' function from the self-constructed script 'deezerData.py'. It was called with the parameter 'groupBy' set to 'True' in order to access grouped 'is_listened' data from the subsample (as explained in chapter 3) on 'user_id', 'media_id', 'genre_id' and 'artist_id'.

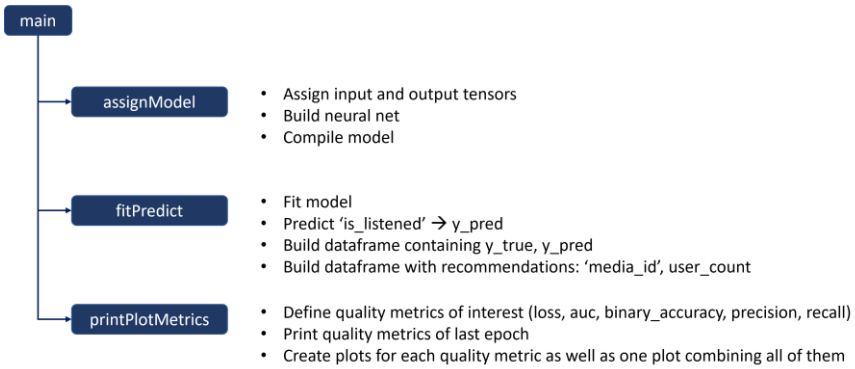

Figure 5: Basic structure of main_keras_deepLearning.py.

Three functions were called from main (see Figure 5): Input and output tensors were assigned within function 'assignModel'. There, also the neural net itself was built by defining dense layers with different number of perceptrons and activations functions. The last dense layer was built on only one perceptron with a sigmoid activation function for binary classification purposes. The model was compiled using 'BinaryCrossentropy(from_logits=True)' as loss function and SGD as optimizer. It was then returned by the function 'assignModel' and used to fit the train data within 'fitPredict' function.

With this fitted model, prediction of dependent variable *'is_listened'* was performed, and the result was merged with the true values in a data frame for console output. Moreover, the predicted values were combined with the independent variables test data. This dataframe was further restructured to show ten recommendations to a given user by sorting the recommendations according to the number of users who already listened to the respective songs. Both, songs that a given user already listened to as well as recommendations for other songs, were printed on the console. In addition, quality metrics were both printed (for the last epoch) and plotted (for all epochs) by calling the function 'printPlotMetrics').

For the approach without hyperparameter tuning, *number of epochs* (50), *batch size* (128) and *validation data* (test data) were set. Quality metrics for the last epoch were as follows for metric (validation metric): loss: 0.47

(0.53), auc: 0.88 (0.75), binary_accuracy: 0.88 (0.77), precision: 0.94 (0.86), recall: 0.89 (0.80), see also Figure 6. Note that the values might slightly vary if no seed was set.

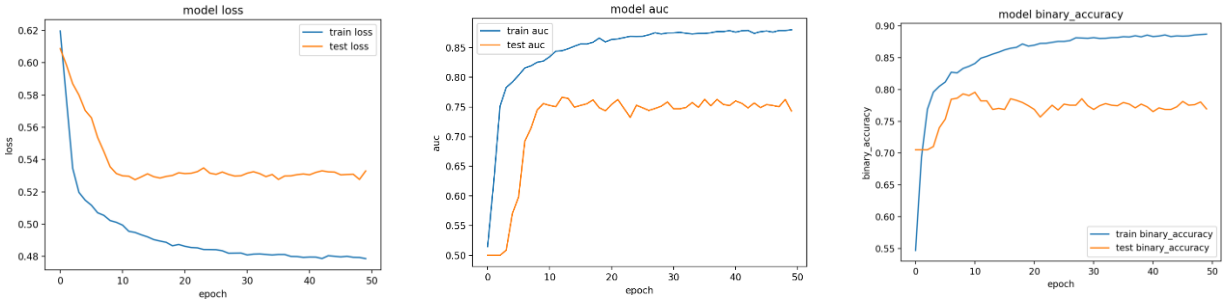

Figure 6: Quality metrics derived from the model without hyperparameter tuning. Loss (left), AUC score (middle), binary accuracy (right) for train (blue) and test (orange) data.

The model where hyperparameter tuning was applied (using *BayesianOptimization Tuner* from *Keras),* included an additional function to evaluate the best hyperparameters for a given set. In our project, we mainly focused on the optimal number of perceptrons for the input layer. Therefore, a range between 32 and 512 for perceptron size was screened. The evaluated number was then directly applied to build up the model. Furthermore, the optimal epoch size was evaluated by searching for the best quality metrics. Those were found as follows for metric (validation metric): loss: 0.46 (0.54), auc: 0.92 (0.70), binary_accuracy: 0.92 (0.76), precision: 0.96 (0.83), recall: 0.93 (0.83), see also Figure 7. Note that the values might slightly vary if no seed was set.

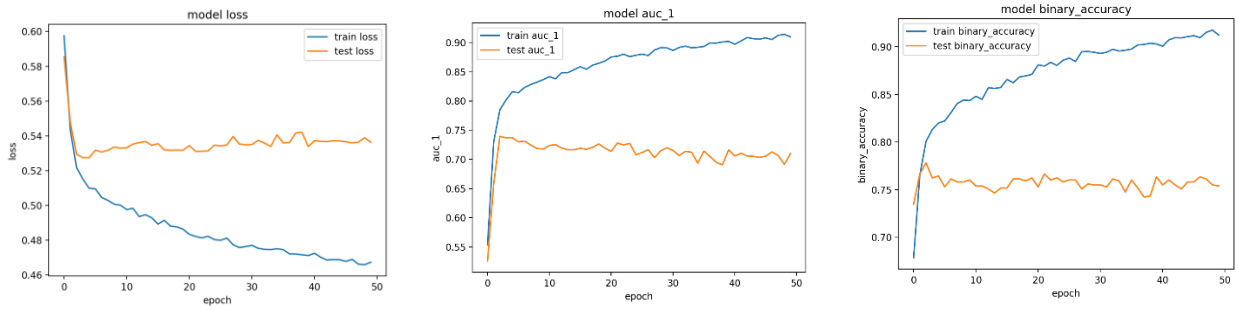

Figure 7: Quality metrics derived from the hyperparameter tuned model. Loss (left), AUC score (middle), binary accuracy (right) for train (blue) and test (orange) data.

This was our first trial in getting to know and applying hyperparameter tuning on neural nets. There is many different spots within the neural net (e.g. number of hidden layers, number of perceptrons per layer, learning rate, steps, ...) where you could evaluate the best hyperparameters. For this project, we hardly included potential hyperparameters within the search grid as it was not the main focus in getting familiar with recommender systems.

So far, the hypertuning approach turned out to enlarge overfitting in our example as the discrepancy between the quality metrics derived from test and train data increased. This approach needs to be optimized in the future and should be regarded as exemplary within this project. However, it made us aware of this potential issue that comes with hyperparameter tuning and even highlights it in the generated plots (see Figures 6 and 7). Therefore, it is always worth looking at the data, how metrics behave during the whole process and furthermore emphasizes that only looking at a single output value does not tell you the whole truth.

5 Discussion

5.1 What would you do to win this competition?

First, it is essential to get familiar with the provided data and perform data cleaning or transformation steps where necessary. Furthermore, it is critical to understand what information is encoded in which column and to figure out the required quality metric (here: ROC AUC) to focus on. As a next step, we need to get familiar with common tools such as *scikit-learn*, *Keras*, and *TensorFlow* and methods like matrix factorization or SVD.

Furthermore, we should firmly deepen our skills in building up recommender systems. Not only following a single approach but also trying different ones to evaluate the best performing model finally. In general, hyperparameter tuning (e.g. by using scikit learn's *GridSearchCV* or *BayesSearchCV*) could be included in your work to improve the model's predictability as well as cross-validation to improve reliability (meaning to get aware of potential overfitting or selection bias). But nevertheless be aware, that hyperparameter tuning will mainly improve your model's performance to a small extend and you should be further aware of potential overfitting issues.

If possible, try to determine how the present Deezer recommender system is built and analyze its weaknesses. Exclude those in the code.

5.2 What would you do to solve the problem that Deezer is interested in?

5.2.1 From this challenge's scope

According to Kaggle's challenge introduction, "*Deezer wants to improve Flow, its music recommendation radio. The concept of Flow is simple: it provides a user with the music he wants to listen at the time he wants. And if he does not want to listen to some specific tracks and skips songs by pressing the 'Next song' button, then the algorithm should detect it quickly. In this context, getting the first song recommendation right is really important*" (Kaggle, 2017). In short, Deezer aims to have the best recommendation system to satisfy its users, attract new users, and gain market share within the current fierce competition.

While building up recommender systems, we noticed no explicit recommendations given by the users but implicit ones. Those were related to the time having listened to a song: If a user did not listen to a song or listened only to the first 30 seconds of a song, it was denoted as a bad recommendation. On the other hand, songs listened to for longer were assigned as good recommendations. This circumstance resulted in two issues: 1) There are only good and bad recommendations and no further gradations (as 'is_listened' was only decoded as binary), 2) There are several recommendations of the same user to a song. What could now be taken into account is the following:

For example, if a person listens to music and gets interrupted by phone calls, opens other social media's audio/video, meets somebody, or is distracted by other activities within the first 30 seconds, the Deezer system would decode it a "bad" rating. However, there might be cases that this person might like that song and did not have more time to listen. Consequently, this way wrongly affects the rating. Also, several implicit ratings on a song from one specific user could happen. To be more precise, person A might skip the song X several times and listen to the same song X several times. It might depend on their mood, the time of the day, the period, and many more factors. So how would we solve this issue? Whenever the same song happens several times to the same user, it should be depicted as 1 because that means this person actively chose to listen to it, which might imply a solid preference to that song.

5.2.2 From Deezer company's scope

In addition to implicit ratings, the explicit rating could be implemented in which users can click on a like or a heart or a button "put in my playlist" for a song in Flow's user interface. Moreover, they can also record the number of times the person has listened to a song, the duration they listened to that song (e.g., 1 minute or 3 minutes), and check whether the person listens to the end of that song. By doing so, new variables could be added to the dataset as implicit and explicit ratings.

Likewise, there might be some additional workarounds that could be considered when personalization. For instance, Deezer could use a hybrid recommendation system, where several algorithms and strategies execute simultaneously. It depends on the situation, and then the most suitable technique could be employed for the real-time recommendation. Furthermore, many potential variables about the users could be collected, such as the country of registration, their friends' activity if the legality allows (they might listen to the songs from their friends' playlist if they request to access their friends' playlist in the music streaming system), the time of listening. For instance, person A usually listens to the Deep House genre at 18:00 after her long working day and listens to Pop songs Sunday morning. Then the recommender system should detect this listening routine to suggest the right song at the right time for the right mood. Not only the user's friends are relevant but also similar users who listened to a lot of similar songs are beneficial for recommending potential songs.

Finally, the content-based recommendation is also essential. Using items' attributes or characteristics such as length, language, genre, published, type, artists, description, songs' mood is to find the distance between all items. Text-based similarities should also be deemed with the auto voice transformation to generate song lyrics. Likewise, we could use Bert or Term Frequency Inverse Document Frequency (TF-IDF) or other Natural Language Processing technologies to extract the most frequent phrases or words that one user might listen to. Additionally, building each user profile is crucial to developing an accurate, robust, customized music recommendation.

5.3 Why might the two solutions not overlap or why they do?

To answer this question, we must clarify the objectives behind question 1 and question 2. For question 1, the goal of the Kaggle challenge/competition is "*to predict whether the users of the test dataset listened to the first track Flow proposed them or not*" (Kaggle, 2017). Hence, winning this competition with given evaluation metrics means that the model achieves the highest score for ROC AUC on the provided test set for the column 'is_listened'. This could be treated as a classification problem by implementing a good, precise machine learning algorithm.

For question 2, we should not contemplate the problem solely by predicting users listening to songs but, most importantly, how to build a robust and state-of-the-art recommendation system. Therefore, we strongly considered it a bigger picture for music recommendation, where Deezer cares about operating it in daily business. The given dataset has many limitations with a restricted number of variables. Hence, many new ideas, features, and factors needed to be taken into account for answering the second question compared to the first one. Besides, the metrics should be adapted because ROC AUC is a performance measurement for classification problems. In general, different metrics should also be applied for recommender systems, such as:

- User satisfaction, number of users using the recommender system (in this case, Flow for Deezer), growth rate of users, and economic success
- Technical performance and system's life cycle
- Beyond the accuracy: coverage, diversity, novelty, churn, responsiveness, A/B test

To sum up, both solutions attempt to provide the most precise prediction by applying great algorithms. However, they differ by their true goals: simply applying on one dataset (question 1, static approach) or providing a robust and precise recommendation system in online streaming music services (question 2, dynamic approach). The second question is undoubtedly more complex and more about the practical operation of the growing volume of data.

6 Team members' contribution

With three enthusiastic members, we started the first project meeting immediately after the last day of the class (Friday 04 February 2022). The tasks were not divided initially because everyone would like to fit as many models to explore and learn. To keep the workflow structured, we organized meetings every second day to explain and show each other what we had done. Notably, we performed the EDA part in the beginning then discussed noteworthy points. The next step was to make a uniform subsample together, so the result was intended to be preferably comparable between the different machine learning techniques. Thirdly, many ways were applied to fit and evaluate the models. For example, if some machine learning techniques were overlapping, we would explain the steps to compare the fitting process and results to each other. Hence, we chose the best script at the end for that method.

Each group member contributed equally to the project with curious, open, and humble mindsets, building up an excellent collaborative atmosphere where we could quickly dive into the new topic of recommender systems and exchange problems, ideas, and solutions. In addition, we organized the work in a coordinated way that helped to perform the task quite effectively and efficiently. The collaboration was ensured by using several platforms such as Signal App for instant messages, Zoom meetings for scheduled video calls, and most vitally, GitHub to have version control and push/pull the codes.

7 Reflections

7.1 Martin's reflections

I was excited to be able to take this course as recommender systems are everywhere in our life. I learnt early on from the founder of the WIRED magazine that I should search for unrelated stuff on YouTube to get better search results as this broadens the content that YouTube considers for me. I was a subscriber of Spotify and Tidal and realized that recommendations can vary quite a lot. Due to these experiences, I was interested to learn how recommender systems work.

It was unfortunate to learn at the beginning of the course that the course required a lot of knowledge which will be taught in the coming semester. This led to additional effort to learn the methods which is positive for the coming semester but I'm skeptical that I understood all the methods well enough in such a short time.

I also wanted to take the opportunity to learn DataSpell. This software is the data science version of PyCharm which was launched late last year. It integrates Jupyter Notebook which is an interesting enhancement. As we worked with git, I was also interested in the interaction with git which led to some issues which I couldn't solve so far. DataSpell itself is the application I will use in the futures as it is more versatile for a data scientist compared to PyCharm.

As we all were new to the topic, we organized the group work more like an agile project. We met several times and constantly updated the plan of how to progress group work. This led to the issue that most of us did investigate most of the methods themselves. The teamwork was therefore an important part but took probably a bit of the time that could be used to go even deeper into the topic.

I understood how interesting the matrix factorization is and built a script that did not use any prebuilt package except of Pandas and NumPy to build a function that solves the task. This was a very important step for me as I'm convinced that reusing a working code doesn't lead to understanding the fundamentals of a methods. It was then when I came across the first memory issues. This is what cannot be learnt with existing code. The question that stayed from this experience is how Netflix and other companies can work with much bigger data sets. I would also be interested how SciPy handles this issue.

The SVD and SVD++ worked for a subset but for the full train.csv the SVD worked but the SVD++ seemed to get stuck and didn't show a result even after several hours of calculation. The setup of a block seminar leads to the problem that this kind of issues cannot be discussed. I used the autoencoder to program a neural network. During studying this method, I learnt that it is basically a nonlinear PCA which I think could be a very powerful addition to a data scientist's toolset.

I was aware of the fact we had a script on day four but using a existing code doesn't give me the learnings that I can get from setting one up step by step. I conducted several videos and the blog from Keras. This led me into the next memory issue with our data set. I took the advice from our lecturer, Dr. Guang Lu, and used the script from the class. Even though it gave me another approach I wasn't happy as I think the data was not handled in the way I thought it is logical for our data set. I finally succeeded and my script worked but the result was not very good.

This issues of not knowing why results don't come out as good as hoped is probably the biggest dissatisfaction about this course as I remain with the issue of not understanding what is really needed for a recommender system be it data or model wise.

In conclusion I would say that I learnt many things in a short time but also a few questions remained, and I hope that being confronted with them before starting the next semester with many machine learning courses will help me better understand and interpret the methods that will be taught. It has certainly sparked a lot of ideas how some of the methods could be implemented in my master thesis.

7.2 Quyen's reflections

Recommender Systems thrills me because, as a consumer, I interact with recommendations in daily life through different apps and websites. Derived from my excitement, I read the Toward Data Science article the

teacher sent and did the DataCamp course "Building Recommendation Engines in Python" before the class started to overview the subject.

First of all, let us start with the coding language. As a person who has no prior background in programming and data science, I really appreciate the course even though I found it very difficult and somehow much higher than my current level. In Hochschule Luzern, we had one class called "Python for data scientists," We learned object-oriented programming with exams but no actual project. Afterward, my first Python project was under the module "Data Collection, Integration and Preprocessing", where we scraped a website cleaned and transformed the data. There were seven other data science projects during the second semester, but all were with R. Hence, the Recommender Sytems' group work is my second Python project. It was very cool to practice coding in Python and apply machine learning methods in surprise, scikit-learn, and Keras. By doing so, I realized the fundamental differences between R and Python. For example, many machine learning R's packages can keep the data frame in Tibble form (with tidyverse's style) while scikit-learn works with NumPy array.

Regarding the course content, the lecturer covered a lot of topics, such as data mining, various metrics, Markov chain, different filtering techniques such as content-based, topic modeling, collaborative, user-based, item-based, model-based, memory-based, knowledge graph, hybrid, rule-based, image-based techniques; top-N recommender. Moreover, different algorithms were also discussed, namely Restricted Boltzmann Machine, auto-encoders, deep factorization machine, deep neural network, SVD, TrustMF. Many of them are real-life applications, relevant to various machine learning subjects such as NLP and deep learning. Although it was my first time ever learning these topics, the teacher showed us various latest data methodologies behind the recommendation. For instance, I got to know "reducing dimensionality", one of the topics in the Machine Learning 2 class (I will have in the coming semester). Besides, the MovieLens example and the codes were helpful to demonstrate the techniques. Beyond that, I also loved the three guests' presentations so much because they introduced the application of recommender systems in their companies from different fields. In terms of course materials, Mr. Lu is genuinely an expert on this topic, and he puts a lot of effort into giving us many papers, slides, other practical recommended courses, and many online resources. So I really feel grateful for that.

"*Learning by doing*". I have acquired a lot from the project. Being given the Kaggle challenge, many questions popped up initially. How to connect it with the recommender system? Which methods should we use? How to deal with all variables with "ID"? How to deal with metadata? What are the differences between binary classification and recommendation? Which recommenders' filtering techniques are suitable here? How to deal with these kinds of features? What does platform 1 2 3 mean? However, after summarizing the slides of all techniques, I understood better. Besides, applying different DataCamp courses' approaches to a dataset such as binary classification methods, non-personalized recommendation, user-based collaborative filtering, SVD, the neural network made me grasp the problem better. I started to recognize the differences when I implemented these methods. Yet, some confusions remain due to lack of experience and time limitations. For example, I tried deep learning with Keras with hyperparameter tuning and found the number of neurons and hidden layers, but it was unsuccessful. Whenever I applied the model or visualized the whole dataset, it was very long to compute. During the process, I had the chance to practice Python, use many packages and modules, use Spyder for the first time. Likewise, collaboration via GitHub in R Studio is not new for me, but this was my first time working with different branches. Also, the three questions for the project actually help deliberate and look at the challenge from various perspectives. Finally, the group members were very motivated, highly active, and encouraging. We all could try out many models and easily compare our codes with each other.

All in all, I will do recommendation-related topics for my Master's thesis, and I wish to dive into more filtering techniques such as content-based, NLP, and knowledge graphs in future research. Indeed, this module is valuable for the coming semesters, my thesis, and I have learned substantially.

7.3 Kirsten's reflections

The lecture provided a comprehensive overview of different techniques to build up recommender systems. A lot of material was referenced or provided which I really appreciated. However, it was hard to deepen all the

knowledge within a block week. Several questions popped up afterwards, whenever we were working on the project. The *movielens* example (that was mainly used in the lecture) is in general widely used in most tutorials on recommender systems that you can find on the internet. However, the deezer dataset seemed to be more tricky because there were recommendations not being done explicitly by the user but implicitly by listening more or less than 30 seconds to the song. This resulted in multiple implicite "recommendations" for a song by the same user. This is why grouping of data by 'media_id' and 'user_id' was necessary to get the prediction/recommendation run on the 'main_keras_deepLearning.py'. Would have liked to have heard more about those and other pitfalls during lecture.

The topic itself is really interesting. It was furthermore very intriguing to build up our first neural nets for a real world application. Nevertheless, building up neural nets is still challenging to me, especially to build up the input tensor and to figure out what model is most suitable. Furthermore, doing hyperparameter tuning in combinations with functions turned out to be more challenging as for simple classification model. Even though I tried a lot in the last weeks, there is still much left, that I would like to try and cover in the future.

Because there is so much to learn about recommender systems, it might be interesting to visit a second, more in-depth course at HSLU. Maybe you could provide one in the future.

8 Bibliography

Adiyansjah, Gunawan, A., & Suhartono, D. (2019, September 12-13). Music Recommender System Based on Genre using Convolutional. *Procedia, 157*, 99–109. doi:https://doi.org/10.1016/j.procs.2019.08.146

Brunton, S., & Kutz, J. (2019). Data-driven science and engineering. Machine learning, dynamical systems, and control. Cambridge: Cambridge University Press.

D-X, C. (2022). *Unsplash*. Retrieved February 2022, from https://unsplash.com/photos/PDX_a_82obo

Funk, S. (2016). *Netflix Update: Try this at home*. Retrieved February 14, 2022, from https://sifter.org/~simon/journal/20061211.html

Kaggle. (2017, June 1). *DSG17 Online Phase*. Retrieved February 2021, from Kaggle: https://www.kaggle.com/c/dsg17-online-phase/

Lu, G. (2022, February). *Recommender Systems.* Retrieved February 2022, from ILIAS Hochschule Luzern: https://elearning.hslu.ch/ilias/ilias.php?ref_id=5316505&cmd=view&cmdClass=ilrepositorygui&cmdNode=10f&baseClass=ilrepositorygui

Mary Stone. (2022, February 02). Retrieved February 2022, from Best music streaming services 2022: free streams to hi-res audio: https://www.whathifi.com/best-buys/streaming/best-music-streaming-services

The University of British Columbia. (2021). *Course:CPSC522/Recommendation System using Matrix Factorization*. Retrieved February 15, 2022, from UBC Wiki: https://wiki.ubc.ca/Course:CPSC522/Recommendation_System_using_Matrix_Factorization

Thenraj, P. (2020, June 20). *Do Decision Trees need Feature Scaling?* Retrieved February 2022, from Towards data science: https://towardsdatascience.com/do-decision-trees-need-feature-scaling-97809eaa60c6#:~:text=Takeaway,the%20variance%20in%20the%20data.

VanDyke, E. (2021, 10 27). *The Rise of Music Streaming Services*. Retrieved February 2022, from Global EDGE: https://globaledge.msu.edu/blog/post/57046/the-rise-of-music-streaming-services

Wikipedia. (2022, February). *Matrix factorizatrion (recommender systemsw)*. Retrieved February 14, 2022, from https://en.wikipedia.org/w/index.php?title=Matrix_factorization_(recommender_systems)&oldid=1070495055

Ye, A. (2020, 03 15). *Medium*. Retrieved from Matrix Factorization as a Recommender System: https://medium.com/analytics-vidhya/matrix-factorization-as-a-recommender-system-727ee64683f0