Course Project

Recommender Systems:

Deezer Recommender System

with TensorFlow Recommenders

Submitted to:

Dr. Guang Lu
guang.lu@hslu.ch

Authors:

Nathanael Simon Jost
natahael.jost@stud.hslu.ch

Olivier Gisiger
olivier.gisiger@stud.hslu.ch

Table of Contents

Figures:

Tables:

1 Problem Statement

1.1 Deezer

Deezer is a French online music streaming service, with about 90 million licensed tracks ("Deezer," 2023). Its services are available in more than 180 countries and counts over 16 million monthly active users. Deezer was founded in 2007. As other streaming platforms, such as Spotify, a crucial task of Deezer is to recommend new songs to customers that they might like.

1.2 Recommender Systems

Recommender systems are models, that are used to predict a customer's interest in each article. In a case like Deezer, a music streaming platform, such an article may be a song. Other cases could be movie streaming (recommended article: a movie) or online book shops (recommended article: a book). The basic concept of such recommender systems is to look at the behavior of a customer A and trying to predict another customers B behavior based on the behavior of customer A (Aggarwal, 2016).

1.2.1 Collaborative Filtering

Collaborative filtering is a technique that relies on analyzing the interactions between users and items to identify patterns and build a model of user preferences. It works by finding users who have similar preferences and recommending items that these users have interacted with, but the current user has not yet seen. In collaborative filtering, groups of users are identified who have similar interests based on their past interactions with items. This approach can be effective even when there is little information available on the items because it relies on the behavior of other users to make recommendations. Collaborative filtering can be further divided into two types: user-based and item-based collaborative filtering.

User-based collaborative filtering recommends items to a user based on the preferences of other users who are similar to them. It works by finding users who have similar preferences based on their past interactions with items and recommending items that these users have interacted with, but the current user has not yet seen. This approach can be effective when there are many users with similar preferences.

Item-based collaborative filtering recommends items to a user based on the similarity between the items. Past interactions of users with items are analyzed and pairs of items that are frequently interacted with together identified. Items are then recommended to a user that are similar to the items they have already interacted with. This approach can be effective when there are many items with similar attributes.

1.2.2 Content Based Recommendation

Content-based filtering is a technique that relies on analyzing the features of an item to create a profile for it. The system builds a profile for each item based on its attributes, such as genre, director, actor, or keywords. The system then recommends items to a user that have similar attributes to the items the user has interacted with in the past. For example, if a user has watched several action movies, the system may recommend more action movies to the user based on the similarity of the attributes. In Content-based filtering, one identifies the features of an item that are likely to be of interest to a user based on their past interactions with similar items. This approach can be effective when the user's preferences are well-defined and there is a lot of data available on the items.

1.3 Kaggle Challenge

As mentioned before, a crucial task of a streaming platform like Deezer is recommending songs to cutomers. Therefore, Deezer launched a Kaggle challenge to let the online community of Kaggle develop an algorithm, that would recommend matching songs for Deezer's recommendation radio «flow» (*DSG17 Online Phase*, n.d.). In a first phase of the challenge, the goal was to predict, whether a user would listen to a song (for more than 30 seconds before pressing the «skip-button») or not.
The first phase of the competition took place in April 2017 to May 2017.

1.4 Dataset

The dataset used in this course project differs from the original dataset – but the provided columns are the same. The actual dataset, that is used in this course has 15 columns and counts 7'558'834 rows in the training dataset and additional 19918 rows in the test dataset. Table 1 gives a brief overview of the provided columns.

Table 1: Columns contained in Dataset. (DSG17 Online Phase)

Column	Description
genre_id	identifiant of the genre of the song
ts_listen	timestamp of the listening in UNIX time
media_id	identifiant of the song listened by the user
album_id	identifiant of the album of the song
context_type	type of content where the song was listened: playlist, album ...
release_date	release date of the song with the format YYYYMMDD
platform_name	type of os
platform_family	type of device
media_duaraion	duration of the song
listen_type	if the songs was listened in a flow or not
user_gender	gender of the user
user_id	anonymized id of the user
artist_id	identifiant of the artist of the song
user_age	age of the user
is_listened	1 if the track was listened, 0 otherwise

1.4.1 Explorative Data Analysis

In order to understand the underlying structure of the dataset, an explorative data analysis («EDA») was performed. The EDA includes several analyses: this chapter focuses on son key-findings. For the full analysis, see appendix.

Figure 1: Song release dates in test- and train data

Figure 1 shows the proportion of songs released in a specific year, starting from 1980. There are songs in both datasets, that were released before, but they would not be visible as a large part of songs was released in recent years. Since data was collected in 2016 (see figure 2) mainly very new releases were considered.

Figure 2 shows the timepoint, on which the observation was collected: when did the user listen to a song.

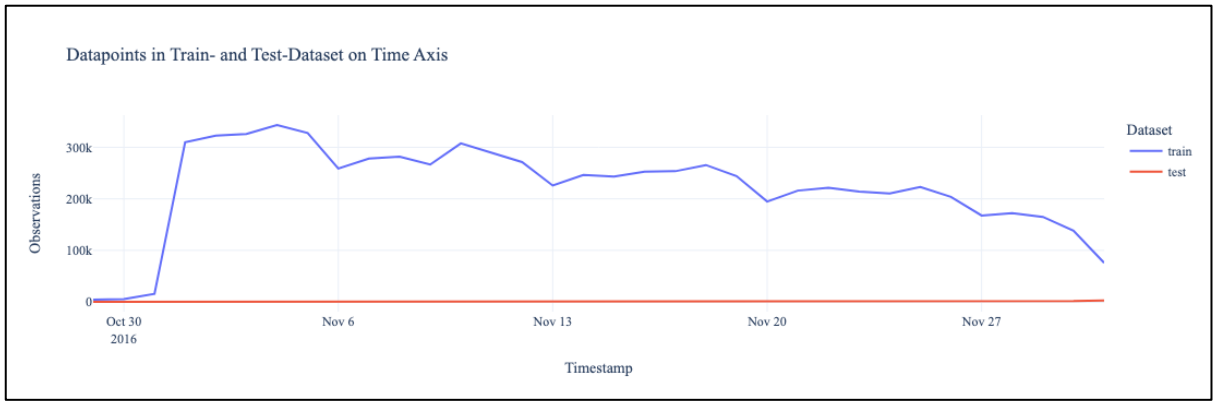

Figure 2: Observations over time

Data was collected in November 2016, this is true for train and test data. There is far less data collected for the test dataset than for the train dataset. All data for a specific user in the train dataset were collected before the test data was collected (Appendix).

All users considered in both datasets are between age 18 and 30 (figure 3). However, there seems to be a small difference in age, regarding the dataset: users in the train dataset seem to be older, users in the test dataset younger. As data comes from the same users (there cannot be an observation in the test dataset if the user was not in the train dataset) this behavior is surprising.

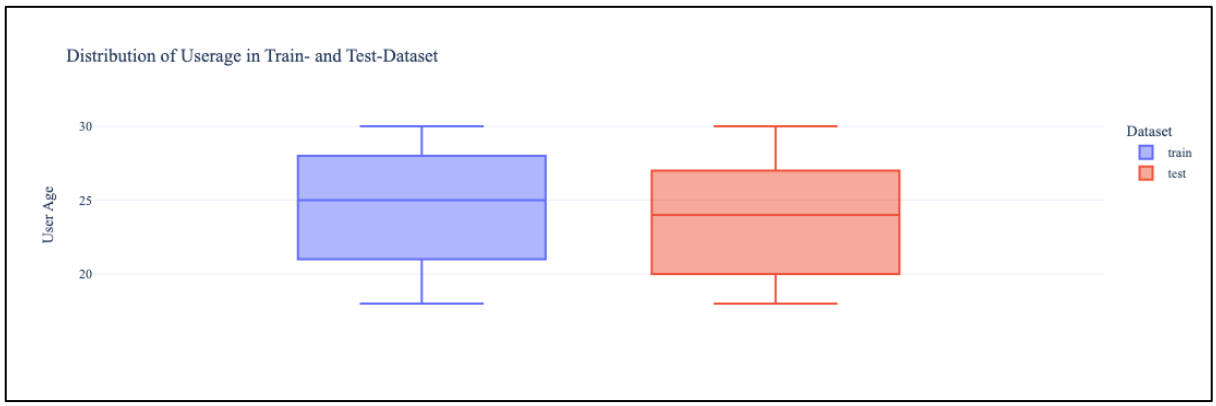

Figure 3: User age in train and test dataset

2 Winning the Kaggle Challenge

The Kaggle challenge took place in spring 2017. At this time, tensorflow recommenders was not released and therefore was not used in the solution to this problem.

Traditionally, such problems were approached either with collaborative filtering or content-based algorithms (eg., Ricci et al., 2015) (see section 1.2.2). However, one major problem of collaborative filtering techniques is scalability. That is, they require a lot of computing power if faced with large datasets. Depending on the given data, content-based approaches do not suffer as much from this problem, since by tendency, one has a lot more users than items. By its nature, content-based filtering is best used if a lot of information about the items is provided. However, the present data contains a lot of information about users, such as age and gender as well as some information about items, as the genre, album, release date and others.

Given the problems described, an optimal solution would consider the required computing power, the (relevant) information about users as well as properties of the items. A model unifying the properties mentioned is the retrieval – ranking model (eg., Costa & Roda, 2011, *Recommending Movies*, 2022). The latter is often composed of two stages. The goal of the first stage (retrieval) is to subset an optimal subgroup of items a user may be interested in. This stage is often itself composed of two different models: one model trains user representations, and one item representations, whereby the two are multiplied to acquire a user-item match score based on which the aforementioned subset is defined. The subset is then used in the ranking stage, which works as a fine-tuning model, optimizing the predictions of the users, i.e., the user-item match (*Using Side Features*, 2022). This approach is efficient in that the retrieval stage preselects a subset, and therefore uses less computational resources. Moreover, content information can be incorporated into the ranking model, which allows for a better fine-tuning and ultimately more accurate prediction.

For the present project, we therefore chose to solve the challenge using a retrieval – ranking model using the TensorFlow recommender systems library (*TensorFlow Recommenders*, n.d.), focusing on a retrieval model for the scope of the current project. In a first step, the songs were filtered to assure that only song that were listened to were taken as input data. Next, a train – test split was incorporated and subsequently, the user embeddings as well as the model embeddings were defined, based on unique lists of the latter two. Subsequently, the retrieval model was trained (100 epochs) and evaluated based on the accuracy of top k recommendations.

Figure 4 shows the test accuracy of the recommended songs: The test accuracy for the top 1 recommendation lies at 11.3 %, for top 100 recommendation it lies at 23.6 %. This numbers result of a presumably, slightly overfitted training process, as the train accuracies lie at 10.9 % and 34.2 %, for top 1 and top 100 recommendations respectively.

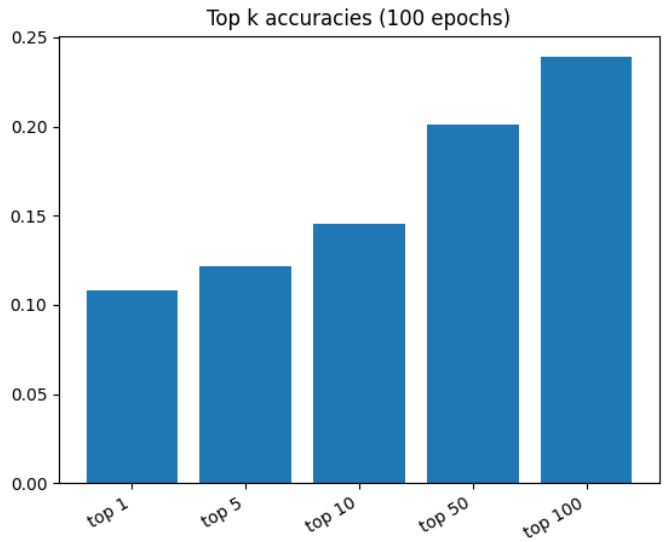

Figure 4: Test accuracy of (ranked) recommended songs.

3 Recommendations for Deezer

Implementing a real-life recommender engine is no easy task. The most challenging problem is perhaps the processing of millions of datapoints and getting a result quickly. Although production level recommenders sometimes have latency budgets (i.e., acceptable latency), generally, the faster a recommendation is output, the better. Therefore, Deezer should consider this issue in the decision for their engine architecture. The tower model, composed of different stages, including a preselection (retrieval) is a good approach to prioritize economic resource use. Evidently, the latter makes sense from a business perspective as well, since reducing computing resources ultimately reduces costs. Nevertheless, our suggestion of a tower model does not restrict the different towers to a certain architecture. That is, there is creative potential in the architectural choice; for example, retrieval does not have to be based on simple embedding mappings of users and items. One could also base the retrieval model on knowledge graphs containing a variety of information about users and songs. In the same sense, the ranking model architecture could be further developed as well. However, resources would always have to be considered, because otherwise it would misappropriate the whole tower logic. Therefore, Deezer has a variety of possibilities in the choice of their model architecture, and we suggest they build several models and carefully test and contrast them with regards to speed and accuracy.

To optimize overall performance of the model, Deezer should also closely look at the data they collect and use as input for the engine, since any predictive model can only get as good as its data. In this regard, there are several topics that need addressing. First of all, the data provided for the current project is exclusively implicit. That is, they do not represent feedback from the users but are rather observed. Consequently, Deezer could ask its users specifically via survey, for example to list their preferences, and rate songs on a scale. Note that it is not necessary to ask all users; given a sizeable

customer base, a representative sample should suffice to approximate the distribution of users' variables. This method has several advantages, since it allows for generally explorative hypothesis building (e.g., to identify new variables with predictive value), identification of possible biases in the engine as well as generally validating the model performance. Besides the selection of variables, one should also consider the scale of the variables. The dataset provided for the present project uses 30 seconds as threshold for coding if the user listened to the song or not. We suggest that Deezer refrains from dichotomizing the time a user listened to, since it stands to reason that the latter is not accurate in many cases. For example, if a user listened to a 20-minute song for 40 seconds only, one can argue that the song was not actually listened to. Much more interesting would be to try and predict how long a user listens to a song. If this would turn out to not be feasible, one could take a proportion of a song to code whether it was listened to (e.g., if 60% of the song was listened to). Aside from the scaling, an additional consideration of the present data concerns the sequential nature of the listening of songs. That is, since a lot of users do not listen to a single song only, Deezer should carefully consider how to leverage the influence of previous songs listened to for the recommendation of future songs. For example, if a user is in the mood for a certain genre or even artist, the algorithm should optimally be able to detect the mood and assign a higher probability for songs of the same genre / artist.

Besides model architecture and data collection/input, a production-level recommendation engine presupposes the consideration of a variety of issues. For example, the pipeline for retrieving data as input for the model must carefully be considered, which in turn affects the choice of database, its language, as well as the cloud computing platform. Similarly, the latencies of the data delivery (model prediction to user) must be reduced as much as possible. Therefore, Deezer should carefully evaluate the use of filters (e.g., filtering songs the user already listened to) to reduce the amount of data going through the pipeline and think about which computations can be executed where (e.g., on (local) Deezer servers, cloud service, or the application of the user). Additionally, given the amount of data, it would be advisable for Deezer to train the models using GPU resources.

4 What Deezer can learn from this Challenge

Following chapter will briefly summarize, what would make a successful participation in the Kaggle challenge and would also help Deezer's radio «flow» to make better recommendations. Lastly, differences between a competition setting and a real-world problem will be discussed.

4.1 Similarities between the Kaggle Challenge and Deezer's Problem

Evidently, our solution to the challenge is not ready for a real-world setting. This is mostly due to the scope of the present project, which is very limited when compared with a large music streaming service. However, there are still some overlaps of our solution and what we would recommend Deezer to consider when implementing a recommendation engine. First, the base architecture we used is a good approach to the problem due to its scalability, the preselection method (retrieval) is optimal to handle large amounts of data which must be processed fast. However, the presented approach does not include fine-tuned model architectures, especially a fine-tuned ranking model. Therefore, a real-world development of an engine would have to carefully consider the model architecture and test different versions and evaluate and validate performance in an iterative process, which needs a lot of manpower and computing power.

A major difference in this regard is the use of appropriate applications and hardware. First, our solution does not require the use of databases and it does not need to recommend songs real-time, which are crucial points to consider for Deezer. Secondly, the presented solution does not require the use of GPU, in contrast to Deezer's optimal implementation.

Additionally, a discrepancy between our solution and the recommendation for Deezer is arising through the limited data. This concerns both the sample size as well as the selection of variables, and the coding/scaling of variables. With the data provided, we are not able to recode variables (e.g. how much time the songs were listened to). Additionally, we do not have qualitative information about the different variables; it would for example have been beneficiary to know the actual genres instead of a numeric representation of the latter. Moreover, it stands to reason that Deezer has more information available, both about users and items. For example, the information about which songs / artists were added to the Deezer service is crucial information, since artists that are available for a long time probably display a different listening pattern than recently added songs. Secondly, the present data only provides the information if a song was presented in a flow or not, albeit not how the flow-algorithm works, which is an information that should be leveraged.

Lastly, the amount of data is crucial. The dataset used for the Kaggle-Challenge counts about 7.5 million records. Although this amount of data leads to severe problems at training models on a local machine – predictions in competition and a real-world setting may get more accurate with even more data. In summary, the approach to the challenge presented in the current project provides a first step into

implementing a production-level recommendation engine. However, as described, there is a multitude of issues to consider and overcome to actually be ready for customers.

Some methods that were applied – or could have been applied - in competition, can actually help make better predictions in a real-world setting. Such a method is sampling, which allows an algorithm to learn from only parts of a whole dataset and therefore be faster in training. Obviously, this method also has its downside: sampling too much, which means leaving out too many observations, can lead to a lower test accuracy. In an application, this would mean to not recommending suitable songs to a customer.

4.2 Where the challenge differs from the real world

The Kaggle shows a legitimate approximation to solving recommendation problems. However, the setting in the challenge is not to be confused with Deezer's real recommendation problem.

There are dependencies in data, that are not considered in the challenge in this experiment. Such a dependency may be the attribute «genre_id»: modeled as a nominal variable, there seem to be genres that are farther apart than others. A model is able learn from data and similar users, but as initial state, genres could be modelled in a distance matrix and recommendations could be made based on distances.

5 Bibliography

Aggarwal, C. C. (2016). *Recommender Systems*. Springer International Publishing. https://doi.org/10.1007/978-3-319-29659-3

Costa, A., & Roda, F. (2011). Recommender Systems by means of Information Retrieval. *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*, 1–5. https://doi.org/10.1145/1988688.1988755

Deezer. (2023). In *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Deezer&oldid=1140909680

DSG17 Online Phase. (n.d.). Retrieved March 11, 2023, from https://kaggle.com/competitions/dsg17-online-phase

Recommending movies: Retrieval | TensorFlow Recommenders. (2022, December 14). TensorFlow. https://www.tensorflow.org/recommenders/examples/basic_retrieval

Ricci, F., Rokach, L., & Shapira, B. (Eds.). (2015). *Recommender Systems Handbook*. Springer US. https://doi.org/10.1007/978-1-4899-7637-6

TensorFlow Recommenders. (n.d.). Retrieved March 17, 2023, from https://www.tensorflow.org/recommenders

Using side features: Feature preprocessing | TensorFlow Recommenders. (2022, December 14). TensorFlow. https://www.tensorflow.org/recommenders/examples/featurization

6 Appendix

All Code is hosted on GitHub. Please consult the README.md file.

Link to GitHub-repository: https://github.com/oliviergisiger/REC03_H2201_DeezerRecommender