Project Report: Kaggle Music Recommendation Challenge

Guang LU (luguangseu@gmail.com, +41793742104)

1 Project Goal

Providing accurate music track recommendations to music listeners is of existential commercial value for music streaming companies such as Deezer. This functionality is a key element to capture the personalized needs of users and if successful can attract new paying customers and generate additional business value to the stakeholders. Therefore, the goal of this kaggle challenge is to employ machine learning (ML) algorithms to predict whether or not certain users listened to the first music track of a given music sequence 'Flow', which is the music recommendation engine from Deezer, has proposed to them. Here 'listened' is indicated as '1' for an at least 30 seconds listening; otherwise 'listened' is marked as '0'. In this task the evaluation metric is the ROC AUC.

2 State-of-the-Art Research and Problem Analysis

Music recommendation is indeed a very active research topic. A very recent review paper by Portugal et al. [1] summarizes the use of state-of-the-art ML algorithms in developing recommender systems. Broadly speaking, the principle behind a music recommender falls into two streams: content-based and collaborative filtering. The former recommends music through extracting useful information from the track content and metadata such as media, album, release date, and artist; while the latter infers user preferences via identification of the likelihood among the tracks that the user has listened or rated. Although a general argument exists that collaborative filtering usually outperforms content-based methods, this recommendation strategy faces the cold-start problem when usage data from new users is not yet available [2].

In this challenge, Flow employs collaborative filtering to suggest the right music tracks to users. At the same time, the software detects the spontaneous response by the user, i.e. whether he/she listened to the music or not. One must be aware of the differences that exist between this task's requirement and the company's interests in general. This task requires ONLY correctly predicting the users' listening state in the given test. In reality the criteria can be others such as predicting users' overall listening state during a given period of time, or correctly recommending the very first song to a listener group whose usage data is not yet available. Thus one admits that the predictive algorithms which are suitable for this task may not be ideal/sufficient for the wider issues and the overall business value that the companies care about. Despite this argument I have implemented the solutions in this report by my own aiming to gaining additional insights into the challenge. It is anticipated that the solutions provided here can be treated as a starting point for more general situations.

3 Data Description and Preprocessing

The original training dataset contains in total 7558834 samples measured within a one-month listening history. These samples span over 19918 users making sure that each user has at least one historical record. The test dataset consists 19918 samples in a list of the first recommended tracks for the same user group. Each sample is represented by 15 features composed of the user and the track information (Fig. 1). In order to apply ML algorithms on top of the task, several data preprocessing steps are performed:

- Drift is observed in the distribution of values of features, such as 'genre_id', 'listen_type', 'context_type', and 'user_id', between the train and test datasets. The data drift needs to be mitigated. A straight-forward way to do so is by only considering the samples in the train set whose feature values also appear in the test set. This works best for discrete features having only a few values; otherwise the drift can possibly be modelled by e.g. a linear model [3].
- To validate the ML algorithms on local machine, the original training dataset is sorted in an ascending order according to 'ts_listen' for each 'user_id'. The new test set 'test_new' is defined as a combination of

the latest sample in time for each user, i.e. 19918 samples in total. Consequently the new train set 'train_new' is defined as the resulting 7538916 samples from the original training dataset, only corresponding to the first 19666 users' listening history.

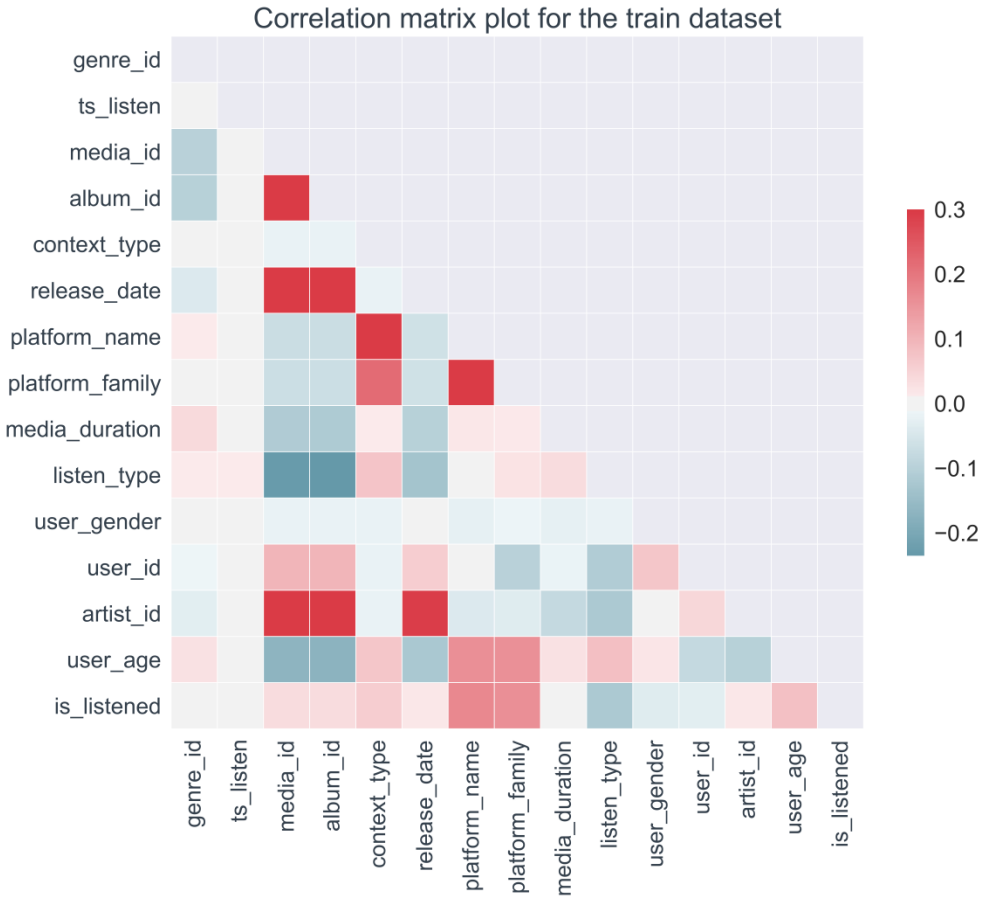

FIGURE 1 Correlation matrix plot for the training dataset shows the pattern lying under the 15 features. The feature 'is_listened' is the predicting target of the ML algorithm. Interestingly 'is_listened' is negatively correlated with 'listen_type' in a relatively strong manner.

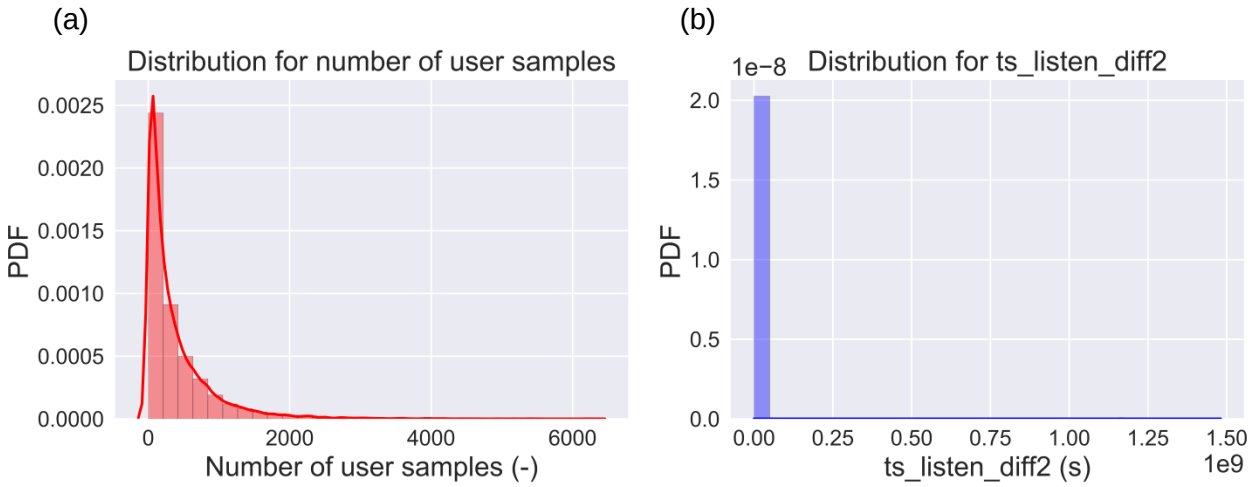

FIGURE 2 Plots of PDF distribution for the (a) number of user samples and (b) ts_listen_diff2 in the train_new. Both parameters are highly unevenly distributed showing the skewness of the training data.

- Some data drift can be observed in the distribution of values of features, such as ‘genre_id’ and ‘media_duration’, between the train_new and test_new. This is corrected by removing the samples in the train_new whose ‘genre_id’ and ‘media_duration’ values do not appear in the test_new.
- The numbers of samples recorded for users in the train_new are highly unevenly distributed (Fig. 2a).
- Two new features are added to train_new and test_new: for each sample belonging to a certain ‘user_id’, ‘ts_listen_diff1’ refers to the elapsed time since last listening, and ‘ts_listen_diff2’ indicates the time gap between this sample and the one of the same ‘user_id’ in the test_new. Note in Fig. 2b the highly uneven distribution of the parameter ‘ts_listen_diff2’.
- Train_new and test_new are separately split into features, without ‘is_listened’, and labels, i.e. ‘is_listened’. Log transformations are applied to features to prevent skewed and wide data distributions. In addition, feature values are rescaled such that they all obey a standard normal distribution.
- No significant covariant shift [3] is found between the feature values in train_new and test_new.

4 Feature Importance and Feature Selection

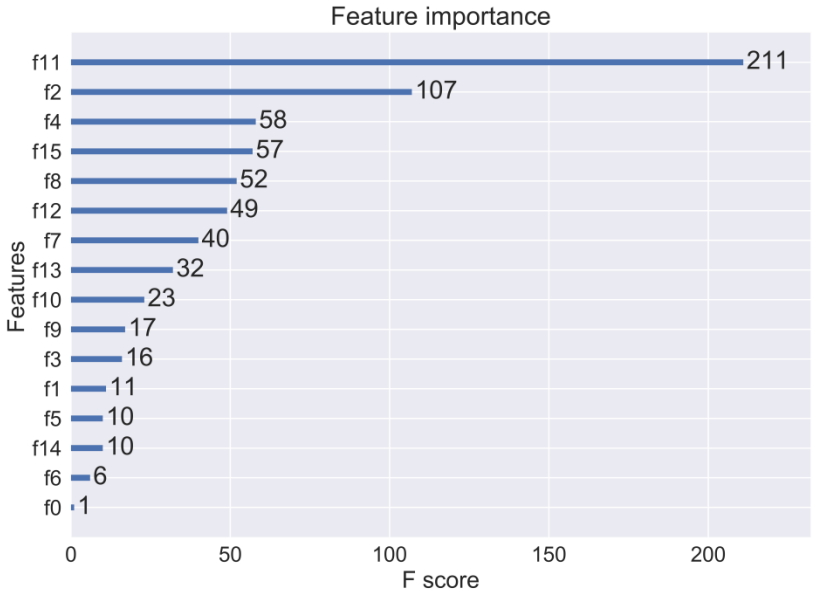

FIGURE 3 Importance of features identified in the train_new using the XGBoost library in Python.

Here I measure feature importance by using the XGBoost library in Python. The selection later aims to positively influence the predictive performance of the ML algorithms, by fitting models only on the important features. Multiple thresholds are tested for a proper feature selection. As shown in Fig. 3, after manually mapping the feature indices to the original data fields in the train_new one concludes that the most important feature is ‘user_id’. Considering the trade-off between model complexity and accuracy, I define the feature importance threshold at 0.016, leading to that the first 12 most important features are identified as the ‘important features’.

5 Modelling Methods

To start, I implement three ML algorithms as the benchmark modelling (B), which are used as the comparison basis for the next more ‘advanced’ modelling. Tab. 1 lists the pros and cons for the each chosen algorithm. It is intended to investigate the performance of both linear and non-linear classifiers on this task, and also to determine if the ML ensemble meta-algorithms such as boosting can improve the predictive accuracy. Due to the

limited training data and time for this project, neural networks deep learning is not selected here as one benchmark modelling method.

TABLE 1 Advantages and disadvantages of the chosen three ML algorithms for the benchmark modelling.

ML Algorithms	Advantages	Disadvantages
Logistic Regression	Behaves robust, and scales well to large number of samples and features	Limited prediction capacity as it is only a linear model
Random Forest	Reduces overfitting, good results with model training on a small sample number	Not easy to interpret, bad scalability for large number of features
Tree Boosting with XGBoost	Enjoys very good model performance, supports parallel computing, integrates seamlessly with Python Scikit-Learn	Generally slower compared to Light GBM and CatBoost

Here the model fitting parameter 'class_weight' is set to 'balanced' considering the various number of samples observed among the different users. Furthermore, the predictive force of the above benchmark models is 'improved' from the following perspectives:

- *Benchmark Modelling + Feature Selection (BF)*: The same three ML algorithms (Tab. 1) are again trained on the train_new but using only the selected important features from Section 4. This is intended to decrease the potential influence of the noisy data on the final predicting output.
- *Benchmark Modelling + Feature Selection + Weighting Factor (BFW)*: Based on BF the weighting factor is evaluated on the training samples according to 'ts_listen_diff2' incorporating the consideration that tracks listened long time ago might not significantly influence the users' current choice. Here the weights are calculated as weights = 1 / (numpy.log(train_new ['ts_listen_diff2'].values+1)+1).
- *Benchmark Modelling + Feature Selection + Weighting Factor + Local Modelling (BFWL)*: Based on BFW, for some users having more than certain sample numbers in the train_new, individually local ML models are built only on these users' training data aiming to raise the corresponding predictive accuracy, and, in turn, the overall predictive accuracy; otherwise BF is utilized for prediction. Here the threshold of sample numbers to activate local models is chosen at 3500 for initial testing purpose.

6 Results and Discussion

Data preprocessing is performed using Python and the additional libraries such as Pandas, Numpy, Matplotlib, and Seaborn, and training and testing are mainly performed using the Python Scikit-Learn package. From Fig. 4 it turns out that the best ML algorithm implemented here is Tree Boosting with XGBoost, while its performance on the test_new in ROC AUC and mean squared error varies slightly among the four different modelling methods (the standard deviation is 0.00057 and 0.00052 respectively). The Logistic Regression algorithm behaves not too much worse, showing that it captures to a certain extent the non-linear effect lying inside the data. It seems that the Random Forest algorithm does not work well for this task.

The proposed solutions are efficient when implemented within the Scikit-Learn framework and can be further improved when given more time considering the following aspects:

- About *Weighting Factor*: In BFW the sample weights calculated according to 'ts_listen_diff2' should be optimized. One notices that introducing the existing weighting factors alone to the ML algorithms (except Logistic Regression) does not necessarily increase the ROC AUC score. In this sense even the samples in train_new recorded too long time ago, e.g. longer than the median value of 'ts_listen_diff2' over the entire dataset, can be dropped out to speed up the ML training. Using Tree Boosting with XGBoost as the base ML algorithm, this gives a RUC AOC score 0.6058 and a mean squared error 0.3180.

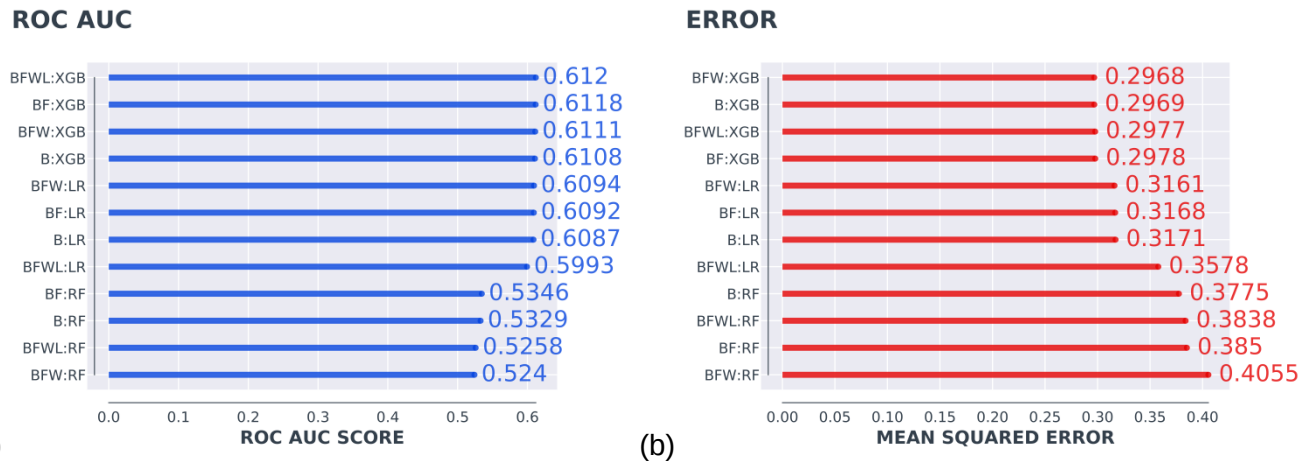

FIGURE 4 Performance of the three ML algorithms [LR: Logistic Regression; RF: Random Forest; XGB: Tree Boosting with XGBoost] on the test_new in (a) ROC AUC score and (b) mean squared error using the four modelling methods [B: Benchmark Modelling; BF: Benchmark Modelling + Feature Selection; BFW: Benchmark Modelling + Feature Selection + Weighting Factor; BFWL: Benchmark Modelling + Feature Selection + Weighting Factor + Local Modelling].

- About *Local Modelling*: In BFWL the threshold should be optimized to determine the 'best' number of samples used for the activation of local models. An initial trial has been performed ranging the threshold from 3000 to 6000 in a step of 500. However, the ROC AUC scores obtained are not too different regardless of the ML algorithms. Clearly finding the optimal weight for the combined models is absolutely critical. One more valuable perspective is that the local models need not necessarily to be trained on each individual user basis. Identifying the similarity between the users and establish local models on top of them may be even beneficial.
- About *Cold-Start Problem*: There is only 19666 users in the train_new, resulting in 252 users in the test_new without having any historical samples in the train_new. Using BF Fig. 5 plots for these 252 listeners the RUC AOC scores and mean squared errors, which are much worse compared to the correspondingly overall scores. Apparently one must consider gathering more information about these users or utilizing more 'advanced' ML algorithms on the existing data to predict their preference.
- About *Parameter Tuning*: Parameter turning is a must to improve the model. When given more time I would like to perform turning of the Tree Boosting with XGBoost algorithm on top of BFWL referring to this Guide and GitHub.

The solutions in this report lay a foundation for responding to the requirements for music recommendation in industry. To this end one can also refer to more 'advanced' ML modelling such as Representation Learning and Sequence Learning. An example for the former is Matrix Factorizations (MF), which have been very successfully implemented for recommender systems e.g. by Netflix in the collaborative filtering setting [4, 5]. Given enough data, this model-based method is potentially a competitive solution for the general issue that Deezer is interested in, i.e. to recommend the first song correctly. Similar to movie reviews, the ratings in this context are thus 0 (not listened) and 1 (listened). On the other hand, sequence learning handles users' historical samples in a sequence manner. The major challenge there is to turn variable length sequences, i.e. user listening history, into a fixed length feature vector for the subsequent supervised learning. If this task is properly solved, sequence learning can be also beneficial for solving the cold-start problem for recommender systems. The state-of-the-art solutions include RNNs/LSTMs, or CNNs such as wavenet [6, 7].

Finally, I intend to emphasize again the statement made in Section 2: 'the predictive algorithms which are suitable for this task may not be ideal/sufficient for the wider issues and the overall business value that the companies care about'. During the report finalization I am aware of this interesting Post. Indeed, in the real music

recommendation cases situation can be much more complex, involving such as difficulty in gathering clean/trustworthy data, great data shift among different user groups, and rapidly changing user preferences impacted by the social environment. Moreover, a developed music recommender system can even 'guide' user's preference, i.e. the recommended music interacts with user and thus influences his/her behavior to stay or leave, and, in turn, the training data reservoir for the future predicting. All these aspects greatly impact the competitive strength of music streaming companies and need to be properly treated in products.

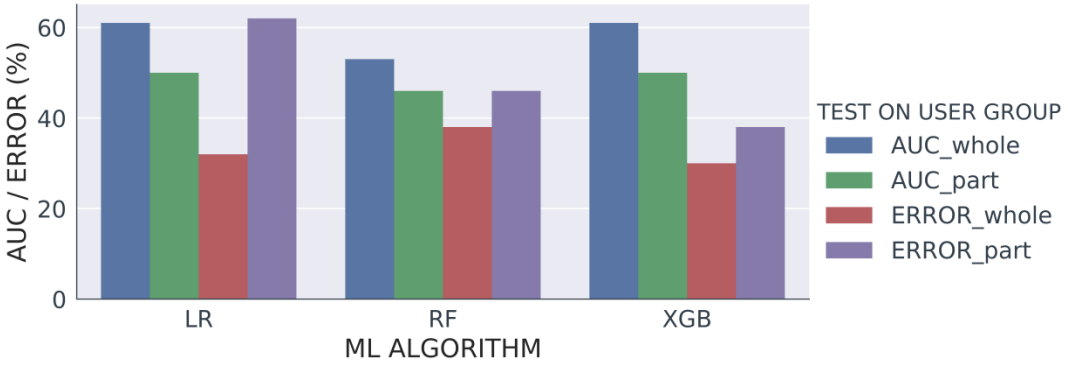

FIGURE 5 Plot of ROC AUC scores and mean squared errors obtained using the three ML algorithms and the BF modelling method for different user groups. The 'whole' group refers to all the users in the test_new; the 'part' group contains the users who only appear in the test_new but not train_new.

7 Conclusions and Future Work

In this competition I have implemented three ML algorithms, namely Logistic Regression, Random Forest, and Tree Boosting with XGBoost, using the Scikit-Learn package in order to solve the music recommendation challenge faced by Deezer. The major conclusions are: (1) the Tree Boosting with XGBoost algorithm, combined with feature selection, weighting factor, and local modelling techniques, gives the highest ROC AUC score 0.612; (2) the performance of the ML algorithms can be further improved considering tuning key modelling elements such as weighting factor, local modelling threshold, and ML model parameters; (3) the competition solutions CANNOT be simply, directly applied on the real music recommendation cases.

As for future work, I am interested in investigating how representation learning and sequence learning perform on the given task. This can be beneficial for solving the cold-start problem that matters to the stakeholders.

8 Appendix

Complete Jupyter Notebook code for this report: https://github.com/lugulugu/DSG17_Online_Phase.

9 References

[1] The use of machine learning algorithms in recommender systems: A systematic review, by I. Portugal, P. Alencar, and D. Cowan, 2018.
[2] Deep content-based music recommendation, by A. van den Oord, S. Dieleman, and B. Schrauwen, 2013.
[3] Dataset shift in machine learning, by J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N.D. Lawrence, 2017.
[4] Matrix Factorization Techniques for Recommender Systems, by R. Bell, Y. Koren, and C. Volinsky, 2009.
[5] Matrix Factorization Recommender Systems Netflix Paper Implementation, by williamFalcon on GitHub, 2016.
[6] Sequence to Sequence Learning with Neural Networks, by I. Sutskever, O. Vinyals, and Q.V. Li, 2014.
[7] Sequence Learning, by sikoried on GitHub, 2018.