Re-listen or Not, this is the Question A Kaggle Challenge on Music Recommender System

Zhaoyi Hou*, Jiaxuan Ren*, Kaihan Zhu*, Fangzhou Yu*
University of Pennsylvania, Philadelphia, USA
{joeyhou,rjx,zkh,fzhyu}@seas.upenn.edu

ABSTRACT

Predicting whether or not a user would like a particular song has became an important but challenging matter for online music services. If they can accurately predict such information, they can feed the users with delightful content and therefore grow their business rapidly. However, the challenging side of this problem is the increasing number of music and the growing number of users. This two factors make the music recommending without enough historical data an over-whelming task. In this project, we picked a particular data challenge from the 11th ACM International Conference on Web Search and Data Mining (WSDM 2018), which requires us to predict whether or not a user will re-listen to a song that was recommended to him or her previously. After learning from the discussion in the Kaggle challenge website and investigating relevant research papers in the field of recommender system, we experimented several different methods for re-listen prediction. Our results show both the importance of feature engineering and also the effectiveness of click-through-rate (CTR) prediction models in the problem of music re-listen prediction.

KEYWORDS

Recommender system, Click-through-rate, Data Mining

1 INTRODUCTION

In this project, we try to predict whether a user of a music app would re-listen to a song after he or she listened to that song for the first time. As a data challenge from 11th ACM International Conference onWeb Search and Data Mining (WSDM 2018), more than 1,000 teams made their submissions before us and the top player gives 0.75 testing AUC in the leaderboard.

As a course project, our main focus is not to achieve certain performance on the leaderboard, but to experience the whole modeling pipeline from raw, uncleaned data to final prediction. We would also like to explore different models, espcially the state-of-art models in recommending system, to see if we can apply the machine learning theory we learned from class into practice.

In fact, this problem is also a great project for machine learning to solve: the huge amount of user behavior records and the high dimensional nature of the data makes it hard for any human to figure out a patter, while the input and output of this problem is well defined as a classification problem. Also note that, although it seems like a task to "recommend music", a more similar example in real world should be click-through-rate(CTR) prediction. Another goal of this project is to apply state-of-art CTR prediction models to see if they can also be applied to music re-listen prediction.

CIS-520, Intro. Machine Learning, Fall2021

user_id	song_id	artist
30,755	359,966	222,363

Table 1: Number of Unique Values

2 DATASET

The dataset is publicly available on Kaggle.com[13]. It consists of multiple parts, training set, user set, and song set. The main training set contains 7,377,418 entries, each with 5 features - User id, Song id, The name of the tab where the event was triggered, Name of the layout a user sees, and Entry point where the user first played the music - and 1 target - Recurring listening event(s) triggered within a month after the user's very first observable listening event, 1 for event being positive and 0 otherwise.

While we have 7,377,418 user (id) − song (id) pairs, there are $11.07 * 10^9$ pairs in total (see Table 1 for details), which means in a use (id) − song (id) target matrix there will be only **0.07%** meaningful entries. Additionally, 27,0341 out of 359,966 unique songs appears less than or equal to 5 times, so **75.1%** columns in the target matrix have the number of meaningful elements no more than 5, out of 34,403 (see Figure 1 for details). The situation of low counts having overwhelming occurrence also happens to "user id" from the train set and "artist" from the song set (see Figure 2 and Figure 3 for details). For numerous users, only have recurring decisions on very few songs that implies a huge variance or uncertainty on our prediction on the taste of users with similar characteristics. In other words, we are facing a really sparse

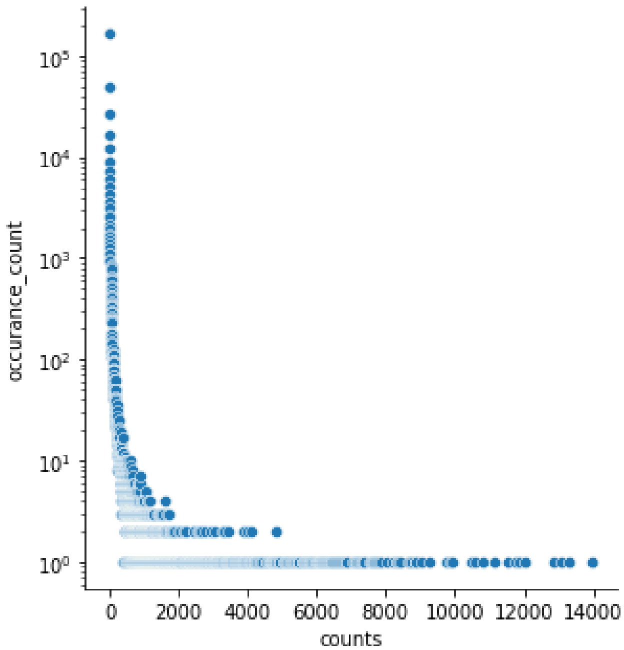

Figure 1: Songs Occurrence

matrix. Sparse matrix is not friendly, even challenging, to ML algorithms in many ways, such as computational expensiveness, speed decreasing, etc. Therefore, since the information provided by original data sets is not ideal for machine learning, we are going to solve this problem before we head to our models by applying matrix factorization, especially SVD, to get embeddings for users/songs/artists and hence construct more valuable matrices for our models.

3 METHOD

3.1 Data Cleaning & Preprocessing

Since the "dirty-nature" of such a real-world dataset, we need to preprocess the data before any other works. We conducted the following preprocessing.

3.1.1 Data Imputation. We first investigated the percentage of missing data (see Table 2 for details). For the missing values in the table, which are all missing more than 0.1% of the data, we imputed them all as "unknown" and latter in the ordinal encoding, they are encoded as a separate class. Other features not noted in the Table 2 have less than 0.1% of missing data, and we directly imputed them with either random sample (categorical data) or average sample (numerical data).

3.1.2 Data Cleaning. In the dataset, one of the feature that is intuitively critical but also consists lots of outliers is the "age" feature. There are around 58% of the age data that are either **less than 0 or greater than 100**, which are both

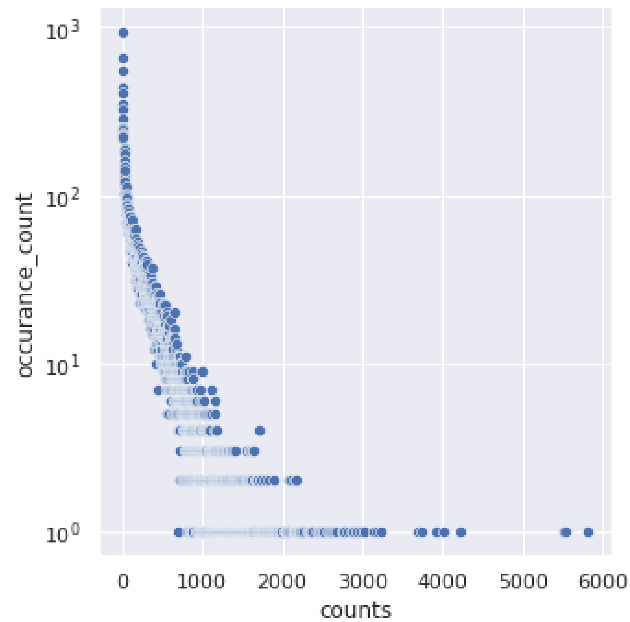

Figure 2: User Occurrence

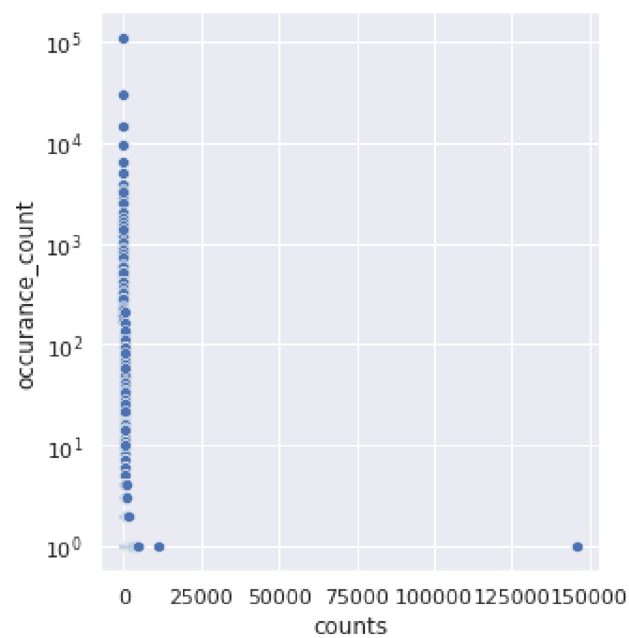

Figure 3: Artist Occurrence

clear signals of being outliers. The way we handle this is by imputing them with the average age given by the "register via" feature, with the assumption that people register the app in the same way would be in the same age range (see Table 3).

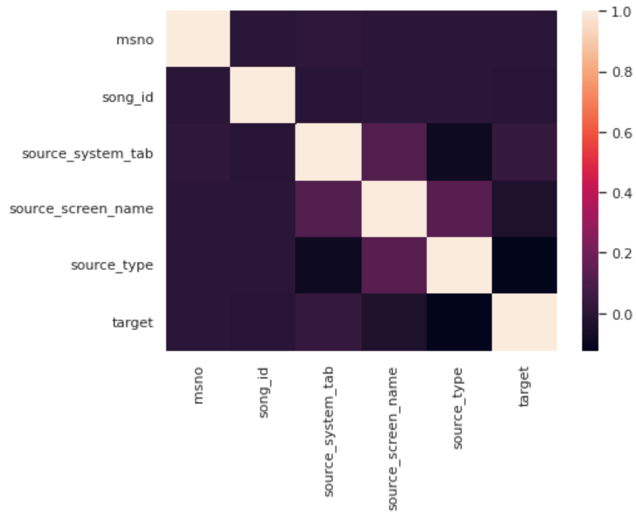

Figure 4: Train Data Correlation

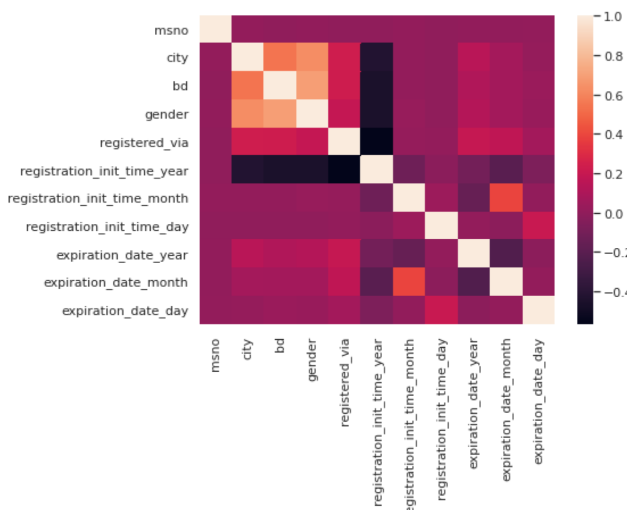

Figure 5: User Data Correlation

Feature	Missing Percentage
source_system_tab	0.35%
source_screen_name	5.60%
source_type	3.00%
genre_ids	4.10%
composer	46.65%
lyricist	84.71%
gender	57.84%

Table 2: Missing Data Percentage

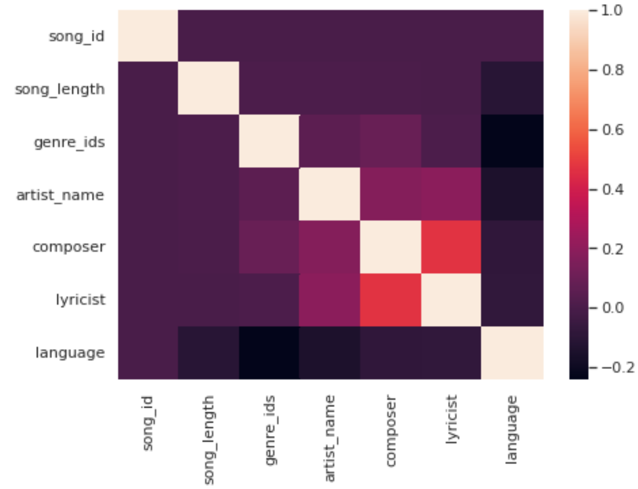

Figure 6: Songs Data Correlation

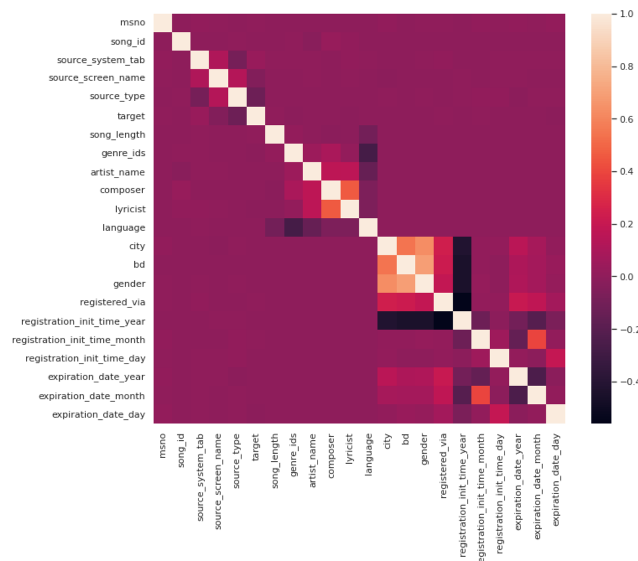

Figure 7: All Data Correlation

3.1.3 Ordinal Encoding. For all the categorical features, we used ordinal encoding to replace the strings and numbers with integer value in $[0, N_{unique\_values})$. In the later part, we either one-hot encode those values or feed them into the model that can embed them into vectors.

3.2 Feature Engineering

Given data fields in the dataset are either categorical data in various formats (strings, integers, etc.) or numerical features in various scales (music length in seconds, age in years, etc.). Therefore, it is necessary to engineer these features to better prepare for the latter parts. In this part, we were inspired by

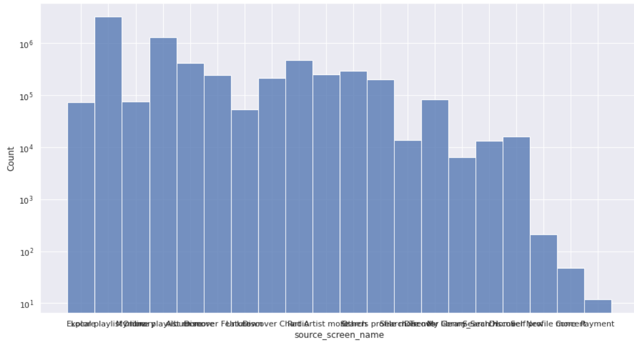

Figure 8: Source Screen Features

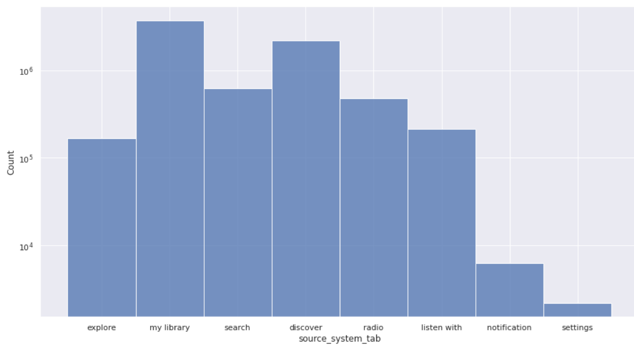

Figure 9: Source System Features

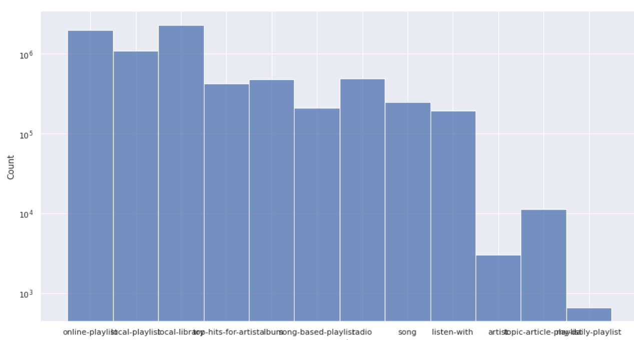

Figure 10: Source Type Features

the discussion in Kaggle.com where previous participants of this challenge shared their thoughts about different ways of feature engineering. In particular, the idea of probabilistic features and the log-count features is inspired by user @lystdo [8].

3.2.1 Probabilities from Categorical Data. Users have different habits and preferences; similarly, songs and artist have different characteristics. For example, a user may have strong previous preference to listen to the songs from a particular screen over others (e.g. home page V.S. artist page); an artist may have lots of songs published compared to other less productive artist. Therefore, we believe different features will have different predicting power condition on either the user or the song. Therefore, we calculated the following conditional features.

3.2.2 Log-count of Catrgorical Data. Another kinds of informative features are the count features. For example, if an artist has lots of songs on the platform, it is an indication of famous artist and might contribute to re-listening behavior. In order to reduce the effect of different magnitute, we calculate the count in the log space.

Average Age	Register Via
26.83	3
26.60	4
30.40	7
30.40	9
25.75	13
28.89	16

Table 3: Average Age by Register Ways

Note: We don't have the exact information about which register method each number represents.

New Feature	Group
P(Source Type | User ID)	User
P(Source Screen Name | User ID)	User
P(Source System Tab | User ID)	User
P(Source Type | Song ID)	Song
P(Source Screen Name | Song ID)	Song
P(Source System Tab | Song ID)	Song
P(Artist Name | User Id)	User & Song
P(Genre | User Id)	User & Song
P(Language | User Id)	User & Song

Table 4: Conditional Probability Features

New Feature	Group
Song | Artist	Song
Song | Composer	Song
Song | Lyricist	Song
Genre | Song	Song
Song | Genre	Song
Song | User ID	User & Song
User ID | Song	User & Song
User ID | Artist	User & Song

Table 5: Logcount Features

3.2.3 Embeddings of song id and artist name using SVD. We constructed a user id to song id matrix and a user id to artist name matrix. In this matrix, if user id i churned on song id j, then the entry at i, j in the user id to song id matrix is 1. Otherwise, 0. Then, SVD is applied on user id to song id matrix to construct 48 principal components for each song id to achieve the purpose of embedding song id's. The same process is done for user id to artist name matrix, but with 16 principal components.

3.3 Models

We defined **random guessing** as our weak baseline, and **logistic regression** as our strong baseline. To utilize the fact that we have many categorical features, we experimented gradient boosting with **LightGBM**. To capture information not accessible to logistic regression, we designed a simple **feed-forward neural net**. We also make use of two open source state-of-art neural net models, **DeepFM and DIFM**, which are specifically designed for recommender systems. Lastly, we **ensemble** different methods to generate better results.

3.3.1 Baseline Logistic Regression. We utilized a simple logistic regression model as our strong baseline model. For baseline model, little data processing has been done. We utilized only the training data, without any additional information from the songs or users dataset, and performed one-hot encoding on all categorical features except userid and songid, which has too many entries to be encoded. We then utilized a logistic regression with logistic loss function

$$f(x) = \frac{1}{1+e^{-x}} \tag{1}$$

which outputs the probability of whether the user would listen to this song again.

3.3.2 Gradient Boosting with LightGBM. Also inspired by the top scorer in this very Kaggle challenge, @lystdo [8], we constructed a LightGBM model to generate the probabilities of churning. LightBGM, different from other gradient boosting machines, is optimized in speed and memory usage. It uses histogram-based algorithms to choose the features and values to split on. For continuous features, it bin them into discrete bins. It only uses a set number of bins rather than all possible values, leading to less memory usage and faster computation. For categorical features, it split them into two subsets. In this way, it no longer requires one-hot encoding for categorical features, which would cause the tree to grow deep and unbalanced to achieve a decent result. This method for categorical features leads to optimal memory usage and higher accuracy with simpler trees. Indeed, we have a lot of categorical features in our data, using LightGBM reduces our memory usage drastically. Therefore, LightGBM is a desirable choice to solve our problem.

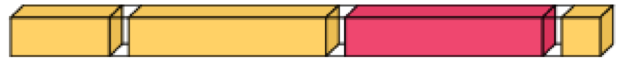

Figure 11: Feed-forward Neural Network Architect

3.3.3 Feed-forward Neural Network. We constructed a simple feed-forward neural network in an effort to capture hidden information that might be missing from logistic regression model. We constructed the neural network with three layers. The first layer is a fully connected layer with 512 neurons, followed by a ReLU activation function; the second layer is a fully connected layer with 1024 neurons, followed by a ReLU activation function; the layer is followed by a batch normalization; finally, we connected the output with one neuron and a sigmoid activation function, which then outputs the probability that we need. (see Figure 11 for detail).

3.3.4 DeepFM & DIFM. Since the problem of re-listen prediction is very similar to the click-through-rate prediction in advertisement prediction, we also tried two neural based model specifically designed for online advertising: DeepFM and DIFM [5] [7]. Both of them are variations of factorization machine, which tries to predict the following:

$$\hat{y_{FM}}(x) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < \boldsymbol{v_i}, \boldsymbol{v_j} > x_i x_j \tag{2}$$

, where the w_i are the real value weights and v_i are the k-dimensional embedding of the i-th feature. With the second-order term $< \boldsymbol{v_i}, \boldsymbol{v_j} >$, the interaction between every two features is captured by the model. In this way, factorization machine is able to accomplish the functionality of collaborate filtering without doing the traditional matrix factorization. The difference between DeepFM and DIFM compared to the original factorization machine is that: both DeepFM and DIFM combined the idea of factorization machine and the deep neural network.

In **DeepFM** (Deep Factorization Machine), the features are used twice: first, they are directly sent to a fully connected neural net to produce a representation of the user; second, they are also sent to the a FM network to produce another representation. In the end, this two vectors will be combined using a learnable weight to produce the final prediction (see Figure 12 for detail).

In **DIFM** (Dual Input-aware Factorization Machine), which was introduced after DeepFM, the main innovation is the combination of DeepFM architect and the idea of self-attention. In the DIFM, the raw input is also used twice, one in a fully connected network and the other in a factorization machine network (FM network). In particular, in the FM network,

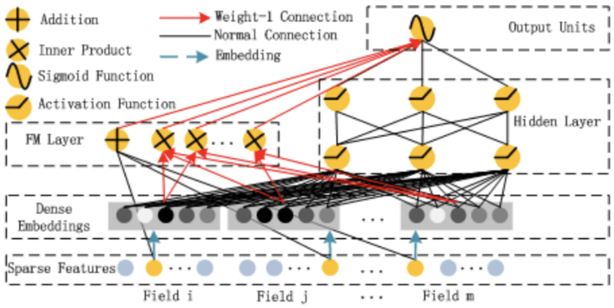

Figure 12: DeepFM Architect[11]

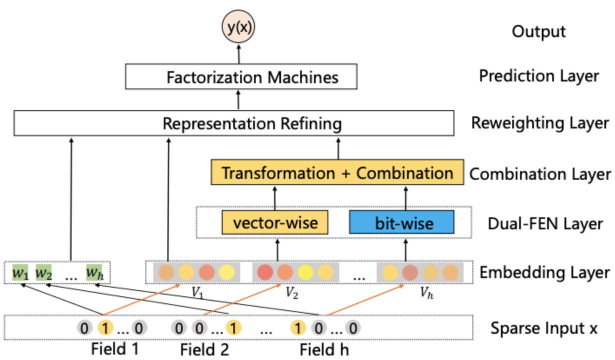

Figure 13: DIFM Architect[7]

the inner-product ($< v_i, v_j >$) used in regular factorization machine and in DeepFM is replace by a Dual Input-aware Factorization Machines [7] which is 1) a bit-wise vector multiplication and 2) a vector level multi-head self-attention. Just like in the DeepFM, the output of the fully connected network and the FM network went through a weighted combination to give the final prediction. (see Figure 13 for detail).

3.3.5 Ensembles of different models. We have ensembled LightGBM and DIFM. We selected the best LightGBM and the best DIFM. We have experimented with different ways of ensemble: 1. weighted average of probabilities, 2. selecting the larger probabilities, 3. selecting the smaller probabilities, 3. selecting probabilities from two models or their average based on whether they agreed to each other or not.

3.4 Loss Function

Since this is a binary classification problem, we will stick to the tradition and use binary cross entropy as our loss function:

$$Loss_{BCE} = -\frac{1}{N}\sum_{i=1}^{N} y_i log(p(y_i)) + (1 - y_i) log(1 - p(y_i))$$

Model	Training Data	Group
Random Guessing	N/A	Baseline (Weak)
Logistic Regression	User Behavior Data	Baseline (Strong)
SVM	Full Feature Set	Machine Learning
LightGBM	Full Feature Set	Machine Learning
DeepFM	Full Feature Set	Deep Learning
DIFM	Full Feature Set	Deep Learning

Table 6: Training Data Group Summary

The reason we choose binary cross entropy is that: since we are trying to predict a probability, we need to measure the differences between our predicted probabilistic distribution and the actual distribution (i.e. target). Therefore, binary cross entropy one it's own can accomplish it.

4 EXPERIMENT AND ANALYSIS

4.1 Evaluation

Since the task is part of a Kaggle competition, we are using the same evaluation metric that competition requires, which is Area-under-the-curve (AUC) of ROC curve. Howerver, one might argue that AUC does not work well with un-balanced data, which is the case in real-world recommender system setup. But in our problem, the data is well-balanced (**51% positive instances**), which means this concern is addressed. Another concern about AUC is that, in the general scenario of making recommendation, not all of the recommendations should be considered equally: only the first few results are important and would be visited by the user, which makes only the first few instances important. In the contrary, AUC consider the mis-match in all instances equally, no matter how close it is with the actual target[6]. This built-in feature of AUC of ROC curve can be troublesome in the "recommand-the-best-song" scenario, where the user is recommended only one or few songs based on the model. However, when it comes to our problem, which requires us to predict the probability of whether the user would listen again, all the negative and positive instances matter roughly equally. That is because, there is no previous assumption about what kinds of songs are shown to the user more often than others. In other words, we want to make sure our model can perform as better in both negative cases and positive cases.

4.2 Training Setup

Note that, all the models in this section are trained on the first 80% of the data (trainning set) and evaluated on the final 20% of the data (validation set).

4.2.1 Logistic Regression. As our strong baseline, we only used user behavior data as our prediction input:

Hyper-parameter	DeepFM	DIFM
Optimizer	Adam	Adam
Learning Rate	1e-5	1e-5
Loss	BCE	BCE
Batch Size	256	256
Epochs	4	5
Embedding Size	16	16
DNN use Batch-Normalization	True	True
Linear Layer L2 Reg.	1e-2	1e-2
DNN Layer L2 Reg.	1e-2	1e-2
Embedding Layer l2 Reg.	1e-2	1e-2
DNN Dropout	0.4	0.4
DNN Hidden Dimensions	(256, 128)	(256, 128)
Num. of Attention Head	N/A	8

Table 7: DeepFM & DIFM Training Setup

source_system_tab, **source_screen_name**, **source_type**. In other words, the song-related and user-related information are not used in prediction. In the experiment, we used Scikit-learn api for training[9]. As for the choice of hyper-parameters, we have the most basic ones: $C = 1$, $max\_iter = 1000$.

4.2.2 SVM. SVM is a great model to be our baseline. It is a widely used model and usually out performs Logistic Regression. In experiment, we used Scikit-learn api for training[9]. We have constructed a SVM model with a L2 regularization parameter of 1. We feed the same data as Logistic Regression to this SVM model, one hot encodings of **source_system_tab**, **source_screen_name**, **source_type**. We have it trained for **7200 minutes**, and it is still running. Therefore, we failed to construct any usable or trained model using SVM. This may be due to the fact that SVM is known to be slow to train. With large amount of data, like ours, it take even longer to train. Therefore, our SVM has not yet converged as the time of writing this report.

4.2.3 Basic Neural Net. We first used TensorFlow Keras[1] built a basic feed-forward neural net which one-hot encoded every feature except for the user_id and song_id, since they are too sparse to encode using one-hot. We then train it on Google Colab GPU with the specified hyper-parameters in Table 9.

4.2.4 DeepFM & DIFM. We use the open-source implementation called DeepCTR-Torch [11] to conduct experiment of DeepFM model and DIFM model on our processed dataset. We trained them using the hyper-parameters shown in Table 7 and evaluate on the pre-set validation set. As for the training environment, we trained both networks with the provided Google Colab GPU (single-core).

Hyper-parameter	Value
N Estimators	100
Learning Rate	1e-1
Leaves	2048
Max Depth	32

Table 8: LGBM Training Setup

Hyper-parameter	Value
Optimizer	Adam
Learning Rate	1e-3
Loss	binary_crossentropy
Batch Size	2048
Epochs	30

Table 9: Simple Neural Net Training Setup

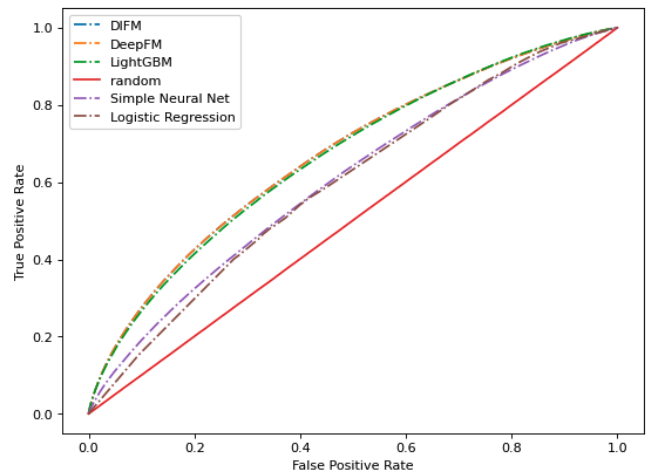

Figure 14: ROC Curve Comparison (All Models)

4.2.5 LightGBM. We used the open-source LightGBM package developed by Microsoft[3] for training experiment. Different number of leaves and max depth had experimented and the best of them is chosen to be our final model. (see Table 8 for detail).

4.3 Results

After training all the models as specified above, we compared their validation and testing AUC as shown in Table 11. We also plotted their ROC curve in the Figuer 14 with respect to the validation dataset (since the actual test set label is only available by Kaggle.com).

4.4 Analysis

4.4.1 Sub-par neural net. An interesting finding regarding neural networks is that its performance is almost the same

Model	Validation AUC	Test Score
Random Guessing	0.5	0.5
Logistic Regression	0.59	0.5
SVM	N/A	N/A
LightGBM	0.67	0.65
DeepFM	0.65	0.57
DIFM	0.68	0.59

Table 10: Model Scores Comparison

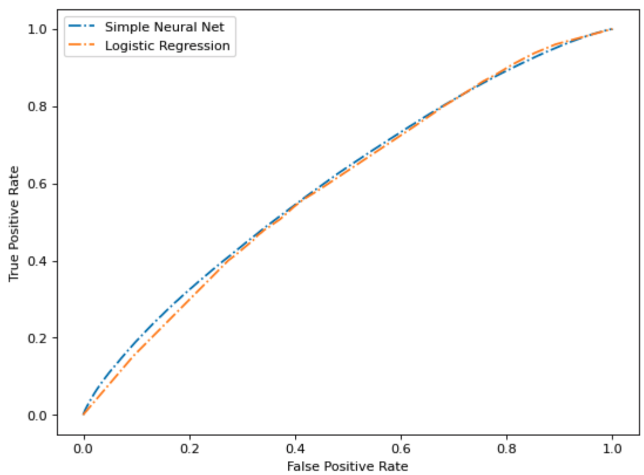

Figure 15: ROC Curve (Logistic Regression V.S. Simple NN)

as logistic regression, with highly similar AUC values. (see Figure 15 for detail). We've tried different data, including one-hot encoding encompassing different datasets, as well as different embeddings. However, all of them worked almost exactly the same as logistics regressions. We've also tried different hyperparameters such as different learning rate, different depths and sizes of the network, as well as different combinations of the layers, but none of them resulted in any significant improvement from a logistic regression model. A possible explanation is that the data is too sparse and has too many missing data such that a neural network cannot extract any more information without further feature engineering.

4.4.2 LightGBM. Hyperparameters of 2048 and 64 for number of leaves and max depth are chosen, respectively. The AUC accuracy for our training data is 0.85 and AUC accuracy for our validation data is 0.67. After training on the whole training data, including the validation data, it achieved a test AUC accuracy of 0.65 on Kaggle private leaderboard. It has the highest AUC accuracy among all tested hyperparameters. We can see that LightGBM overfits the training data a lot, but it still improves validation AUC as training AUC increases.

Leaves	Depth	Train AUC	Val AUC
8	8	0.73	0.64
16	8	0.73	0.65
16	16	0.73	0.65
32	8	0.74	0.65
32	16	0.74	0.65
32	32	0.74	0.65
64	8	0.75	0.65
64	16	0.75	0.65
64	32	0.75	0.65
64	64	0.75	0.65
128	8	0.76	0.65
128	16	0.76	0.66
128	32	0.76	0.66
128	64	0.76	0.66
256	8	0.76	0.65
256	16	0.78	0.66
256	32	0.78	0.66
256	64	0.78	0.66
256	128	0.78	0.66
512	16	0.79	0.66
512	32	0.80	0.66
512	64	0.80	0.66
512	128	0.82	0.66
1024	16	0.82	0.66
1024	32	0.82	0.66
1024	64	0.82	0.66
1024	128	0.82	0.66
2048	16	0.84	0.66
2048	32	0.85	0.67
2048	64	0.85	0.67
2048	128	0.85	0.67

Table 11: LightGBM Hyperparameter Search

4.4.3 DeepFM & DIFM. It is slightly surprising that DeepFM and DIFM are not even close to the LightGBM model. The main problem here is the expensive training time and computational power. To train a DeepFM or a DIFM model, it generally takes 30 minutes for training and validation. Since they also have lots of hyper-parameter to tune, it might be that we are not in the correct hyper-parameter space for this problem (e.g. requires much less learning rate and much more epoches).

Another plausible reason that the DeepFM and DIFM are not out-performing the LightGBM is that, it might have too many redundant features. As mentioned in the original paper about DeepFM [5], factorization machine by itself is creating lots of second-order features that try to capture interaction between different features via embedded vector multiplications. Some of the features might be helpful, for example $< v_{user}, v_{song} >$ could capture the relevance of this user and this particular song, with the assumption that if they are more relevant, this inner product should be larger. But most other combinations, such as $< v_{genre}, v_{city} >$ and $< v_{lyricist}, v_{register\_via} >$, does not make intuitive sense. Therefore, although these two models introduced promising second-ordered features, they also introduced redundancy in features. At least in our experiment, the disadvantage slightly out-weights the advantages.

4.4.4 Ensemble of DIFM and LightGBM. We have experimented with ensembling DIFM and LightGBM. All our emsemble methods produces less than desirable results, except for weighted average. Weighted average of the produced probabilities do increases prediction accuracy. We have discovered that the best weight for the weighted average ensemble is 0.2 for DIFM and 0.8 for LightGBM. It increases the test AUC accuracy from 0.650 to 0.654. Although the increase is marginal, it can be a win or lose for this competition.

5 RELATED WORK

There are many previous work done around this problem. Two of the most significant ones are the first place solution to this very challenge by @lystdo and the matrix factorization.

5.1 Solution by @lystdo

In this person's proposed solution, which eventually achieves an impressive 0.75 AUC score, comprehensive feature engineering and model ensemble are both used. Aside from the probablistic features, log-count features and LightGBM model, which we implement in our solutions, this person also conducts the following:

- More feature engineering: in the dataset, there is another field called "isrc" (International Standard Recording Code), which is a code that contains the year, country and other additional information about the song;
- Trainable embedding: in @lystdo's solution, he/she also included a feed forward neural net that has a trainable embedding for the categorical features (e.g. gender, age, register via, etc.);
- Ensumble: @lystdo made an ensemble of 30 neural network model and LightGBM model in his/her final prediction, which we don't have the time and resources to train on. [8]

5.2 Matrix Factorization

Another important related-work in recommender system is the matrix factorization [2]. As a common collaborative filtering method, matrix factorization creates meaningful representation for the discrete features (e.g. users, songs, artists, etc.) so that these embeddings can help the final prediction. In practice, the matrix to be factorized is the interaction matrix between two features, one of which we are trying to represent. For example, user-song matrix can be factorized into two matrix with shape of $[N_{user} * hidden\_size]$ and $[hidden\_size * N_{song}]$. The way this method accomplish such effect is to conduct a EM-style estimating procedure where the two matrices are estimated after some iterations when reconstruction loss goes below a pre-set threshold. Due to the complexity of this procedure and the limitation of computational resources for large matrix manipulation, we were not able to incorporate this method in our models.

5.3 Click-through-rate (CTR) Prediction

Aside from the technical aspects, as the two sections above mainly focus on, the formulation of our re-listen problem is very similar to a click-through-rate prediction. We are trying to predict re-listen after first listen just like online merchants trying to predict purchasing after first clicking. In this field, methods from many different perspective are proposed. Some of them focus on feature engineering such as keyword-clustering [10]; some of them emphasize the importance of end-to-end learning with light-weight models such as linear regression [4]; while others focus on building the complicated model such as boosting-based models to capture the sophisticated nature of the world [12]. All of them shows the profound nature of CTR prediction as a huge sub-domain of recommending system.

6 CONCLUSIONS AND FUTURE WORK

In conclusion, we showed that with in-depth feature engineering, both neural-based models (DeepFM, DIFM) and boosting-based models (LightGBM) gives above baseline performances. In particular, neural-based models are not out-performing boosting-based model on the hidden test-set, showing the beauty of statistical machine learning once again. However, even without much hyper-parameter tuning, these CTR oriented models still beats the strong baseline and shows that the CTR prediction problem and re-listen prediction are can be solved in a similar manner.
As for the future works, two main possible directions are novel models and feature engineering

- First, due to the limitation as a course project, we did not explore the full potential of feature engineering. In particular, using unsupervised such as auto-encoder and LDA to represent sparse features in the dataset would be a very promising path in this direction.
- Second, a self-defined neural network with trainable embedding would be an ideal middle-ground between LightGBM and DeepFM/DIFM, since it can balance

the over-shot of feature interaction brought up by the DeepFM/DIFM and simultaneously introduce enough non-linearity to model the complicated world.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[2] Dheeraj Bokde, Sheetal Girase, and Debajyoti Mukhopadhyay. 2015. Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science* 49 (2015), 136–146.

[3] Microsoft Corporation. 2021. Welcome to LIGHTGBM's documentation! https://lightgbm.readthedocs.io/en/latest/index.html

[4] Muhammad Junaid Effendi and Syed Abbas Ali. 2017. Click through rate prediction for contextual advertisment using linear regression. *arXiv preprint arXiv:1701.08744* (2017).

[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* 1, 1 (2017), 1–2.

[6] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 5–53.

[7] Wantong Lu, Yantao Yu, Yongzhe Chang, Zhen Wang, Chenhui Li, and Bo Yuan. 2020. A Dual Input-aware Factorization Machine for CTR Prediction.. In *IJCAI*. 3139–3145.

[8] @lystdo. 2017. WSDM - KKBox's Music Recommendation Challenge. https://www.kaggle.com/c/kkbox-music-recommendation-challenge/discussion/45942

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[10] Moira Regelson and D Fain. 2006. Predicting click-through rate using keyword clusters. In *Proceedings of the Second Workshop on Sponsored Search Auctions*, Vol. 9623. Citeseer, 1–6.

[11] @shenweichen. 2021. Welcome to DeepCTR-Torch's documentation! https://deepctr-torch.readthedocs.io/en/latest/index.html

[12] Ilya Trofimov, Anna Kornetova, and Valery Topinskiy. 2012. Using boosted trees for click-through rate prediction for sponsored search. In *Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet Economy*. 1–6.

[13] WSDM2018. 2018. WSDM - KKBOX's Music Recommendation Challenge. https://www.kaggle.com/c/kkbox-music-recommendation-challenge/data