Music Box Churn Prediction and Recommender System

For details, please refer to my Github. If GitHub fails to load the jupyter notebook, nbviewer might be a great help.

Author: Pengfei Li

1. Data Description

The data is the log data of a music box, namely a music player software. The log data has three categories: play, download and search. It can be accessed from the link. The download and search data are relatively naive with several features, e.g. user id, song id, device, song name, singer, etc. More features, e.g. play time and song length can be found in the play data.

With the data at hand, I am going to implement the churn prediction on the users of the music box and deploy a recommender system for the users.

2. Churn Prediction

2.1 Problem Definition

The churn prediction is an important topic of machine learning and data science in marketing analytics and customer relationship management. Put it another way, a churner is a customer that stops using the product of some company. Our goal is to apply machine learning algorithms and techniques to predict the likelihood that a user will churn. Therefore, we can improve the customer engagement from the result of the model.

2.2 Data Preprocessing

2.2.1 Data Cleansing

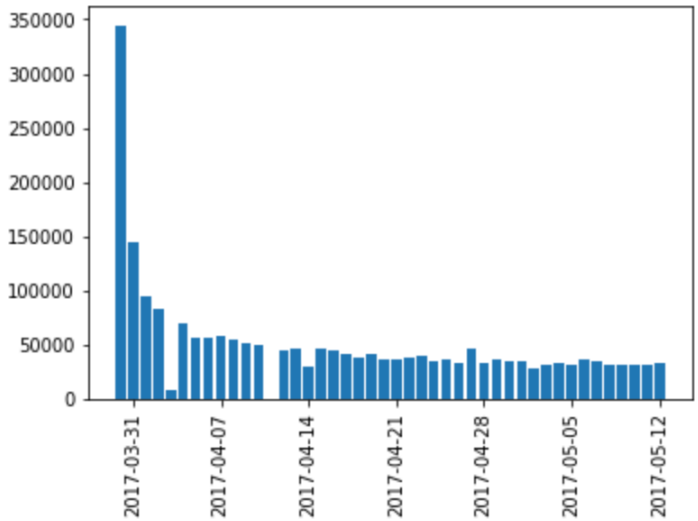
The original data is extremely untidy. I used python to download, unpack, parse and split the all 'tar.gz' files into three main datasets: play, search, download, plus a user id summary dataset.

As for the user id dataset, outliers were detected by observing the descriptive statistics of the counts of every user. The records of some users can be up to nearly 100K. A reasonable deduction is that they are bots the music box used to test the software, which is a trick of most app developing companies. However, they are meaningless for churn prediction. Therefore, I stripped them out. Finally, the number of distinct users dropped down to ~100K.

As the model was run on a local PC. in order to make the parameters tunable, I randomly down-sampled the play, search, and download datasets on the user level by 0.1.

2.2.2 Churn Label and Features Generation (PySpark SQL)

Time period of this log dataset is 44 days. The bar plot of record counts vs dates indicates that the number of records generally decrease to a stable level.



The churn label id defined in the following way. The first 30 days (a month) is treated as the feature window and the last 14 days (2 weeks) as the label window. The user who is active (has records: play, download, or search) in feature window and inactive (no record) in label window is thought of as a churner (label 1). The user who is both active in the feature and label window is regarded as an active user (label 0). The number of churners is twice active users.

```
+-----+-----+
|label|count|
+-----+-----+
|    1| 6679|
|    0| 3078|
+-----+-----+
```

Following features are created,

a. Frequency of events: play (P), download (D), search (S), over time window (days, count from the last day of the feature window above) 1, 3, 7, 14, 30, 3*5 features

b. Recency of events: P, D, S, from the last day of the feature window, 3 features

c. Acceleration: ratio of frequency, play_1d_over_play_7d, play_1d_over_down_1d, play_1d_over_down_7d, 3 features

d. Device type: android or iOS, one-hot encoding, 2 features

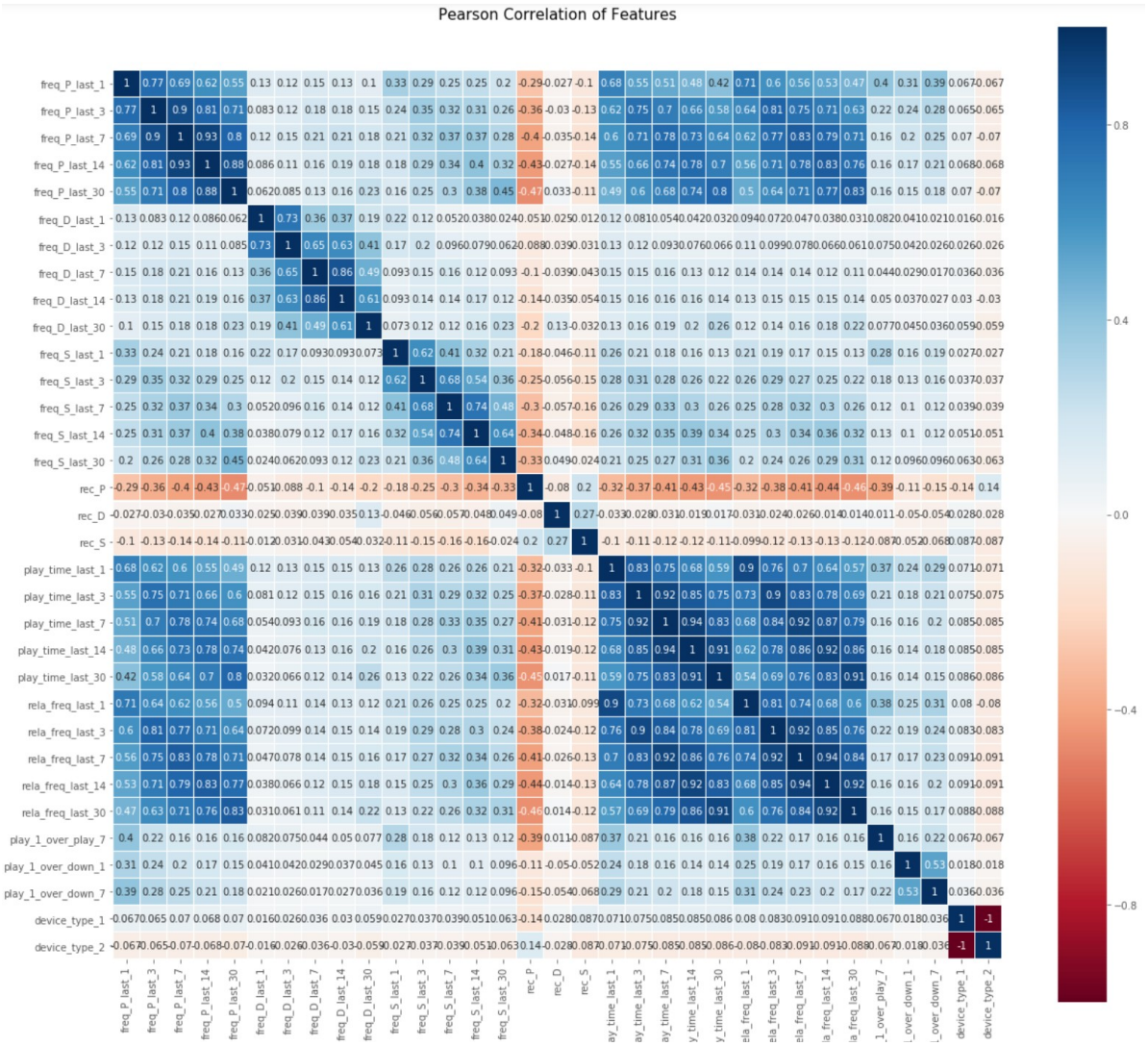
e. Play time (seconds) over the time window 1, 3, 7, 14, 30, 5 features

f. Relative frequency if the ratios of paly time and song length is greater than 80%, counted as 1 relative frequency, over the time window 1, 3, 7, 14, 30, 5 features

N.B. Features in e and f are only for the play dataset. And due to the process of generation, 0s are used to denote no record. Furthermore, play time and song length features need to be cleaned.

2.3 Feature Exploration (Seaborn plot)

As can be seen below, many features are correlated due to the generation.



Frequency features do have influence on churn prediction. Roughly speaking, the more the churn and stay detach, the stronger the correlation is. However, how much the effect exactly requires the feature importance analysis.

2.4 Model (scikit-learn and PySpark MLlib)

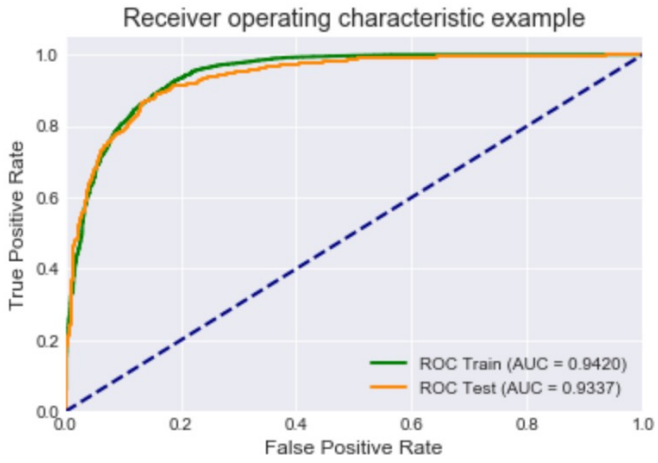
It is a binary supervised classification problem. A stacking model was used to fit the data. In the first level, logistic regression, random forests, AdaBoost, gradient boosting decision trees, XGBoost are implemented. In the second level, a majority vote ensemble method is employed to improve the prediction accuracy.

I also tried to construct the model using Spark, which is a more scalable language. However, the options of algorithms and parameter are not as flexible as scikit-learn, e.g only simple logistic regression, random forests, and gradient boosting trees are accessible, if we just use PySpark. Thus, the followings are mainly about scikit-learn. And the PySpark codes can be seen on my *Github*.

2.5 Model Validation (GridSearch)

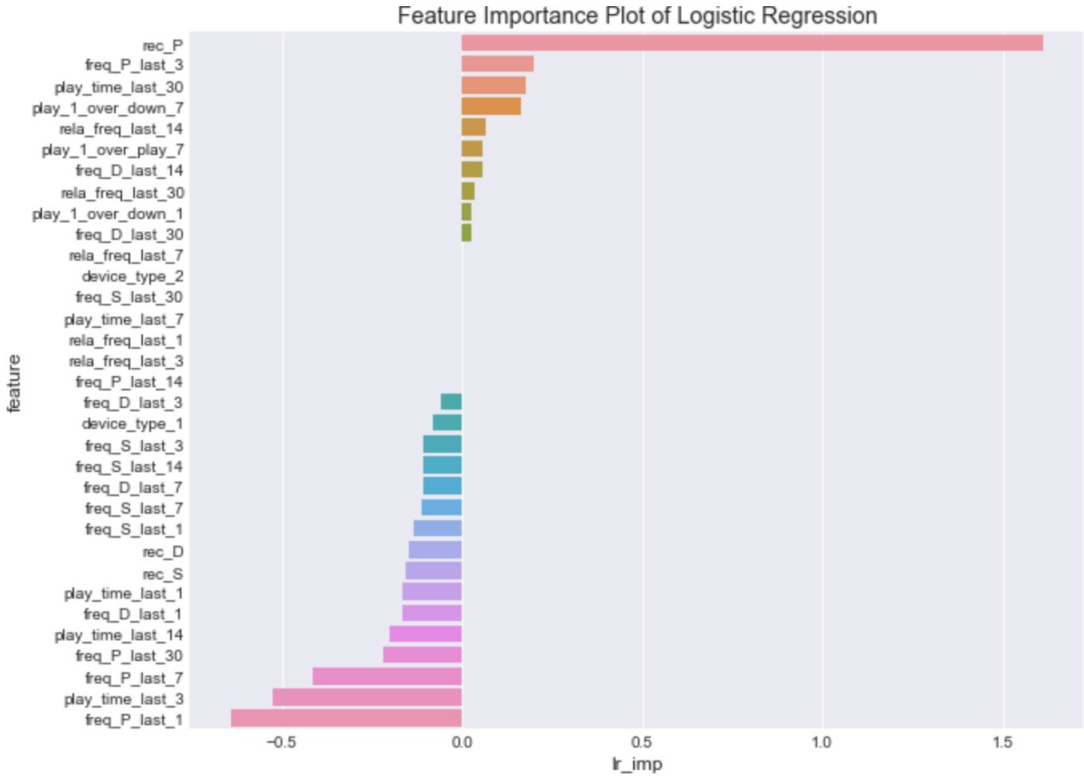
Train-test (8/2) split was first done. I performed the cross-validation on the training dataset to tune the parameters and observe their behaviors on the test dataset. Eventually, all predictions are ensembled by the majority voting.

metrics	train	test
AUC	0.941986	0.933707
Accuracy	0.894521	0.877112
Precision	0.905434	0.892263
Recall	0.944963	0.931321
f1-score	0.924776	0.911374



2.6 Insight Identification

2.6.1 Feature Importance

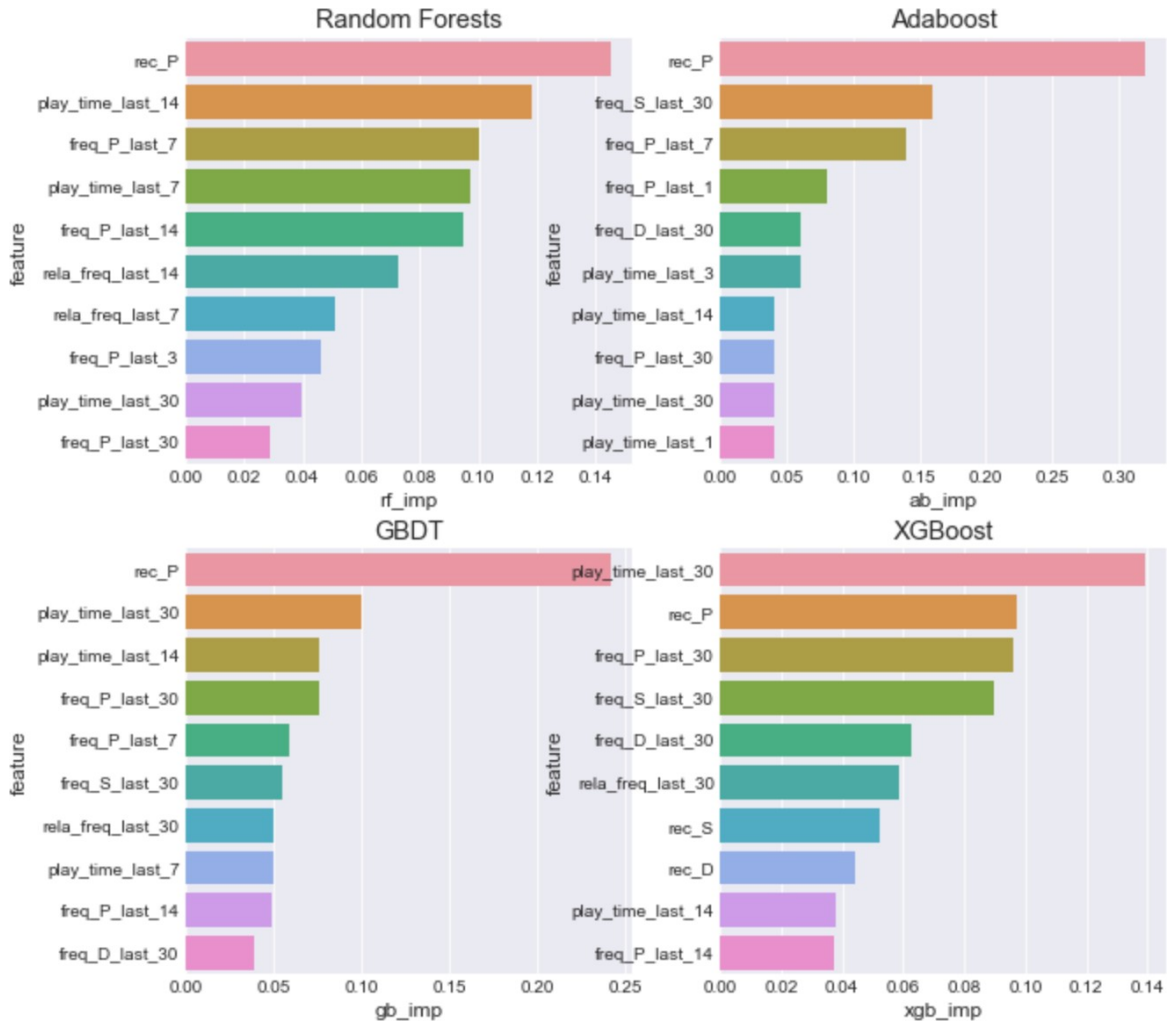


The bar with a positive value represents a positive contribution way and vice versa. If the absolute value of a feature is large, it implies a higher possibility in the corresponding way.

For specific features, the earlier the user played a song for the end the feature window, the more possibly he or she would churn. More frequently the user played, the less likely he or she would give up. However, one interesting phenomenon is that the frequency of play has a positive effect upon the churn. The reason might be over this specific window, the users who played too much get bored with the music box and no longer used it any further.

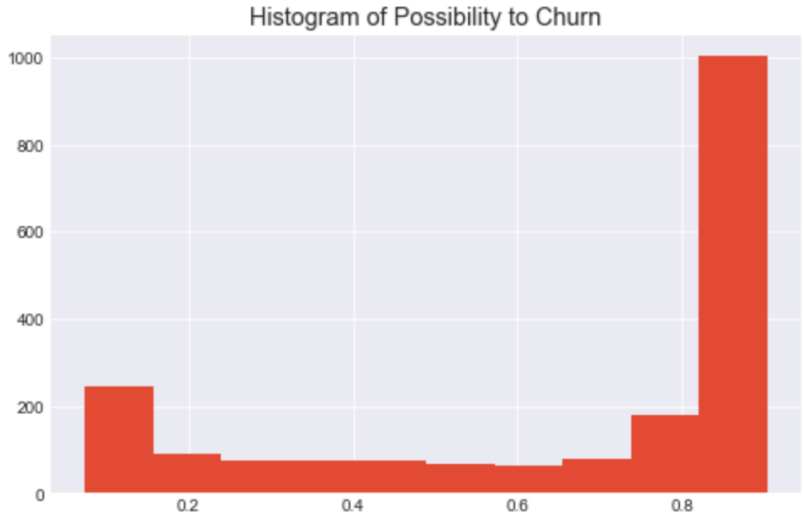
It terms of the last four tree-based models, the behaviors are somehow similar. The top 10 dominant features are plotted. Nevertheless, the contribution way is hard to determine.

The recency of play is a dominant feature, ranking 1st in 3 model and 2nd in the last one. Besides frequencies, the play time are also leading characters. The relative frequency, the ratios of the play time and the song length, ranking low though, are still. Remember we have 33 features and the number of features shown in each subplot is just 10. As the feature of play type, the music box can invest on building a clear, concise and efficient play UI.



2.6.2 Churn Analysis

My model with a test AUC 0.9337 is highly reliable.



In the sense of app, the probabilities to churn of the customers in the test group tells us in the wild the music box is a high-risk app. Although the distribution is bimodal, a larger proportion of users would churn. This app was using a very basic messaging strategy with very simple segmentation, was messaging infrequently to a limited segment of their audience and was gaining almost zero audience engagement. I would recommend this music an aggressive custom engagement policy. They should target more of their users, message more frequently, and add more attractive features to improve the user experience, e.g. in-app message center, play bonus or badge, a sophisticate **recommender system**.

On the user level, a large group of users is likely to churn due to a big churn score. I assign them to the high-risk group. For the app's super fans, with a small churn score, I assign them to the low-risk group. Those who are less confident which way to go are defined as the medium- risk group. The similar engagement policies above can be applied. Our aim is the move the user from the high-risk group to low-risk group, which is easy to monitor by the online data stream or dashboard.

All in all, with feature importance and probability to churn of users, ad hoc policies can be enacted or tested in order to increase the return on investment.

3. Recommender System

3.1 Problem Definition

A recommender system is a system that predicts the rating of an item for a user. Based upon the ratings, we can recommend top-rated items for specific users. The boosting internet brings us much more options than before. When we browse, it is hard and time-consuming for us to choose what we want out of thousands of possible choices. This is where the recommender system comes in. If you visit amazon, some items would be suggested according your search and purchase history.

3.2 Data Preprocessing

3.2.1 Data Cleansing

The process is the same as the section 2.2.1, except one special filter. That is, I removed the users who has no greater than 5 records. The ratings of those users are not taken into account due to an inactive use.

3.2.2 Features Generation (PySpark SQL and pandas DataFrame)

Since there is no explicit rating, we have to create an implicit rating ourselves. In this section, I did not split the whole time period, and use records of all 44 days to generate our target in the recommender system.

Three features were extracted. Play frequency over 44 days, download frequency over 44 days, and relative play frequency (play time over song length) over 44 days.

	freq_P_last_44	freq_D_last_44	rela_freq_last_44
count	72310.000000	72310.000000	72310.000000
mean	3.404868	0.146536	1.113000
std	11.874277	0.401482	5.244973
min	1.000000	0.000000	0.000000
25%	1.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000
75%	2.000000	0.000000	1.000000
max	1496.000000	7.000000	1091.000000

From the chart above, though there are some large values, it is a normal user behavior and just implicates that the user played and download a lot. All features are heavily right-skewed and in different scales.

At first, all features are log-transformed and scaled down to the same level. Then, I put weights 0.2, 0.5, 0.3 due to our intuitive sense of these features. I maintain the download feature is the most important, and the rest two play feature act as a minor role to rate a

song, in which the relative frequency is a bit more important. Lastly, the final scores were bucketed into 5 levels: rating 1, 2, 3, 4, 5.

3.3 Feature Exploration

Three features: user id, song id, and the rating generated in the last section were select the to build the utility matrix for recommender system. In the matrix, there are 43199 observations with 6536 distinct users and 24239 distinct items.

We can see most of our ratings concentrate on 2 and 3.

3.4 & 3.5 Model Training and Validation (Graphlab & PySpark ALS)

GraphLab provides an extremely useful API for recommenders. Three models were built.

A. Item-item Collaborative Filtering

In this model, the item similarities matrix was calculated and used to help predict the rating of a user on an item. Top 5 recommendations for one specific user are shown below.

uid	song_id	score	rank
6871077	6435758	0.0384615384615	1
6871077	58118	0.0384615384615	2
6871077	9561562	0.0384615384615	3
6871077	6451058	0.0384615384615	4
6871077	4756523	0.0384615384615	5

B. Popularity-based Recommender

This recommender only uses the information of item to predict ratings. Naturally, recommendations for every user are the same.

uid	song_id	score	rank
6871077	20870140	5.0	1
6871077	3591502	5.0	2
6871077	3296328	5.0	3
6871077	353224	5.0	4
6871077	981660	5.0	5
9644453	20870140	5.0	1
9644453	3591502	5.0	2
9644453	3296328	5.0	3
9644453	353224	5.0	4
9644453	981660	5.0	5

C. Matrix Factorization with Regularization

In this we have some parameter to tune. Thanks to cross-validation function from GraphLab, we obtain a mean validation RMSE as low as 1.0028.

```
The params of the best model are:
{'side_data_factorization': False, 'regularization': 0.0001, 'user_id': 'uid', 'target': 'imp
licit_rating', 'solver': 'als', 'num_factors': 32, 'item_id': 'song_id', 'max_iterations': 5
0, 'linear_regularization': 1e-09}
 The mean training rmse is:
 0.762239588423
 The mean validation rmse is:
 1.00279733168
```

uid	song_id	score	rank
6871077	6168313	22.5758808797	1
6871077	492211	19.2007549947	2
6871077	23529789	13.6263769811	3
6871077	129386	11.1655148214	4
6871077	312944	10.6914242452	5

As we can see above, the mechanisms of three models are radically different. Thus, the recommendations are not comparable.

By the way, a spark ALS recommender was also built and the rmse is around 1.188.

3.6 Insight Identification

Recommendation is one of the most popular machine learning application. Many platforms have its own API to build a recommender model. GraphLab is employed in this project. With a strong recommender system, the business can personalize the need for each user so as to increase user interaction and enrich purchase or engagement potential. Therefore, the business can maximize their ROI based on the information they can collect on users' taste.