KKBox Music Recommendation System

Xinyue Wang
Linda Xu
Nian Liu
Department of Computer Science
University of Southern California
Los Angeles, CA 90007

April 29, 2018

Abstract

Our project originated from a Kaggle competition that challenged us to build a better music recommendation system using a dataset donated from KKBOX, Taiwan's leading music streaming service. The defined objective was to predict the probability that a specific user would listen to a certain song again within a month of having listened to it once. The project required that the submission file contain two columns: "id" (a unique song and user combination) and "target" (the corresponding predicted probability). The performance of our submission was then evaluated on the area under the ROC curve between the predicted probability and the observed target. To tackle this project, we performed tasks in the following order: data exploration and preprocessing; model considerations and selection; and model refinement via parameter tuning. We first modified the training set that the competition provided us by expanding features, imputing missing data values, creating new features, and label encoding categorical features. Then we considered several models to use. Due to the enormous size of the training set, several models were impractical since training would take many months. We decided eventually to narrow down to using XGBoost and LightGBM. While XGBoost is more accurate than LightGBM, it is more time consuming and ultimately could not serve our purposes. When we used XGBoost, we could only train a model Thus, we ended up choosing LightGBM for its higher efficiency yet slightly lower accuracy.

1 Introduction

This project originated from a Kaggle competition that was first introduced in early 2018 at the 11th ACM International Conference on Web Search and Data Mining. This competition challenged applicants to build a better music recommendation system using a dataset donated by KKBOX, a Taiwanese music

streaming service. With the rising number of music streaming platforms available, these companies also hope to improve their recommendation algorithms to satisfy clients on both the artist end and the user end.

The goal of this project was to create a music recommendation system that better recommended new music to both new and existing KKBOX users across different musical genres. Thus, a difficulty that inherently existed was the fact that users may enjoy many musical genres, and the system must take into account of the users' varied interests. The practical manifestation of this goal was a predicted probability that a certain user would listen to a specific song again after having listened to it once. The submission required a two column result consisting of a "id" or unique user-song combination with its corresponding probability value.

The competition provided 6 files in total:train.csv,test.csv, members.csv ,songs.csv ,$song_extra_info.csv$,$sample_submission.csv$Like what their names suggest, ı$train.csv$ȷ and ı$test.csv$ȷ were the training and testing set that the competition provided.

The process we took to tackle this project can be roughly described in the following sequence: data exploration and preprocessing; model considerations and selection; and model refinement via parameter tuning. We first explored the dataset that we were given and performed preprocessing to finalize our training set. Then, we considered several machine learning models, weighing the pros and cons of each until finally selecting LightGBM. We then spent the remainder of the project fine tuning the parameters for LightGBM to improve our accuracy as much as possible. Lastly, we submitted our project to Kaggle to get our rankings.

It should be noted that the dataset that we were provided was extraordinarily large; the training set has over 7 million samples. The dataset in of itself gave us the greatest complexity of the project. The sheer size of the dataset was an important factor in deciding the learning algorithm that we used. The size of the dataset was also the limiting factor in a lot of ways because it restricted the possibility of using several models. Yet, we wanted to preserve the quality of our model performance and chose not to reduce the training set size. Instead, we moved forward by choosing and adapting a learning algorithm that best suited our needs.

2 Methods

2.1 Data Exploration

To get acquainted with the problem and data at hand, we first took steps to explore and visualize the data that the competition had provided us. To do so, we constructed various graphs and charts that gave us a broad understanding of the data. For example, we explored interactions between features by plotting the pairwise scatterplots and heat maps. We then also plotted bar charts to examine the distribution of values from a specific feature. Lastly, we wanted to

visually assess how balanced the data is. We did so by creating pie charts out of the values from a specific feature. We created these pie charts for both the training and test set to enable comparison between the two datasets. If we can see a distinct imbalance in the distribution of a feature between the training and test set, then we must address it.

2.2 Data Preprocessing

To have a robust training set for our model, we made some modifications to the existing training set that the competition provided. The changes that we made can be summarized by the following points: feature expansion, data imputation, feature creation and label encoding of categorical features

It should be noted that the default training set originally provided contains only a limited number of features: the tab name, screen name, and entry point of where the user first accessed a particular song (see Table 1). The competition instead packaged additional information about individual KKBOX users and songs on their platform in separate files: "members.csv" and "songs.csv," respectively. Additional information about each user and song from those files could be connected to the training set file by matching the User ID and Song ID between the files. Thus, one of the first steps that we took to modify the training set was to expand the user and song features by adding data found in "members.csv" and "songs.csv" to their corresponding sample in "train.csv" via User ID and Song ID matching.

The second step we took to alter the training set was to resolve the issue of missing values. We first found which features had missing values and then filled the missing slots with pre-set values. Following data imputation, we took steps to create new features. Since our sample size is more than 7 million rows, we decided to add more features for each tuple. We first divide the $registration_init_time$ and $expiration_date$ from the original files into separate year, month and data features. Since the song features—artist, composer, and lyricist—could have more than one value each, we then counted the number of artists, composers, and lyricists for each song to make them into 3 more features.

Lastly, we performed label encoding for categorical features using the class LabelEncoder from the sklearn.preprocessing library. This changed our categorical features into numeric features. All in all, we had a total of 27 features at the end of preprocessing. We ultimately had a training set size of 7,377,418 samples and 27 features. This modified dataset was then used to train our model and perform predictions.

2.3 Model Selection

We first considered using K-Nearest-Neighbors (KNN) and Support Vector Machine (SVM) for our model. However, due to our sheer sample size, these models would require a few months to complete running, proving to be highly impractical. Thus, by researching the conference paper, we chose XGBoost and

LightGBM. First, we trained the model with XGBoost. When passing the one percent data, it took 2 days to finish training. The result showed that this method was not very efficiency though it could achieved the accuracy of 0.60339. To improve the accuracy and the time efficiency, we then implemented the model with LightGBM.

LightGBM was developed by Microsoft which was a gradient boosting machine. Compared with XGBoost, LightGBM used leaf-wise tree growth and it could support parallel learning which means it could handle large-scale of data. Looking at the important features of the LightGBM, it had one parameter that was number of leaves which was different than other boosting algorithms. At the first time of training LightGBM, we set the parameters with random values to see whether the result and its training time are accept or not. Then we used 10-fold cross validation results to choose the parameter with best accuracy.

2.4 Model Refinement

To keep improving the performance of the LightGBM, we started to perform parameter tuning. We first tried manual tuning and then automated tuning. Manual tuning was a kind of greedy search and we tuned one parameter at a time. We first gave several separate possible values for the each parameter to tune, the model would keep the best value for that parameter and then moved to find the next parameter we want to tune. During this tuning process, we used 10-fold cross validation to figure out the value of the parameter with the highest accuracy. Each distinct value of parameter will take at last half an hour to finish.

After manual tuning, we found that we could perform automated tuning, which would give us more accurate predict result based on the best parameters we found in manual tuning. We set a range based on previous value and using Bayesian-Optimization to find the best parameters in that range. After several runs, we sort the result and show the result along with the parameters so we could find which set is best for our model.

The difference with the manual tuning is that automated tuning is tuning multiple parameters at the same time. Thus, it could accelerate the tuning process. At last, based on the tuning process, we chose the best parameters and put them into final LightGBM model using whole data set and train the new model. Finally, we submitted the final predicted results to Kaggle.

3 Results

3.1 Model Selection

For XGBoost, we found that the top three most important features are $msno_id(userid)$, expiration date of the membership and the initial registration time of the membership. After submitting our predicted outcome of test set on Kaggle, we got the auc score of 0.60339. For LightGBM, We found that the top three most

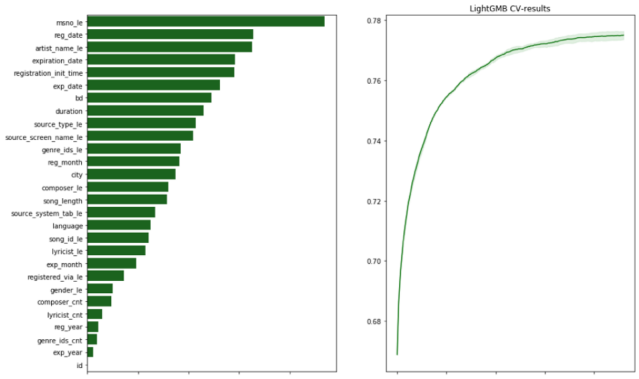

Figure 1: important features and roc curve for lightgbm

```
lgb_params = {
    'learning_rate': 0.3,
    'max_depth': 10,
    'num_leaves': 17,
    'bagging_freq' :3.5
    'objective': 'binary',
    'metric': {'auc'},
    'feature_fraction': 0.3,
    'bagging_fraction': 0.3,
    'bagging_freq': 3,
    'max_bin': 128,
    'min_gain_to_split':1,
    'lambda_l2':97,
    'lambda_l1' :97}
```

Figure 2: best parameters for automated tuning of lightgbm

important features of influencing the result are msno-id, registration date and the artist name.

For the first time of training LightGBM, after using cross validation, we found that the best number of boosting iterations was 461 rounds. And the best auc score was 0.64758, our rank of all above 1500 competitors was the 805th, shown at figure 1

3.2 Model Refinement

After automated parameter tuning, the values of the each parameter was shown as figure 2. And our auc score was improved to 0.65812 and our rank was improved to top 705 among all competitors shown as figure 3.

4 Discussion

For data exploration parts, we found that the relationships between each feature is unclear, but the heatmap gave us some understanding of the relationship of features. For example, from the heatmap of the member table and song table,

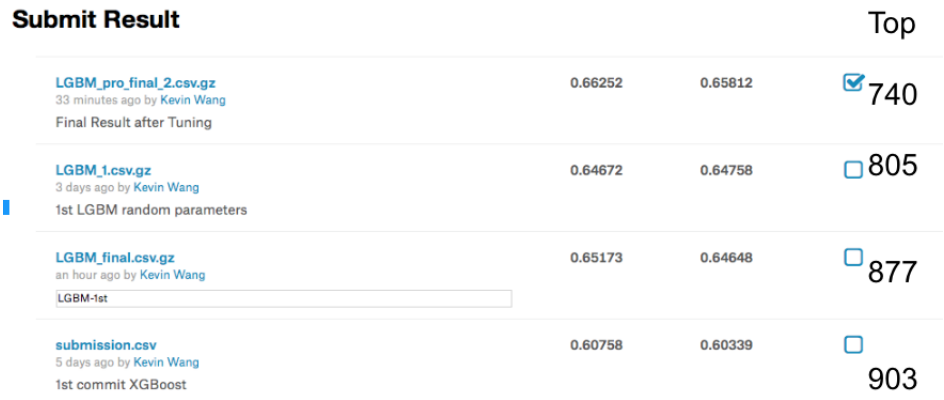

Figure 3: auc scores and ranks for xgboost and lightgbm

we found that the highest common relationship is between the length of the music and the user's registration method followed by the age of the user and the user's location-the city. From the bar charts, for example, among the feature – source type, we found that the largest number of users chose to listen to the music by choosing from their local library. Another example was among the feature-language of songs. We found that users chose to listen the songs in type 52 of language.

From the results, we could conclude that LightGBM perform better and more efficient than other methods like XGBoost, SVM and KNN. Though the increase of the accuracy by using LightGBM was 0.025 percent compared with the XGBoost after took so mang works, since the highest accuracy achieved by other competitors were just above 0.7 percent and the huge dataset had 7 million samples, we regarded that LightGBM and model tuning had a significant improvement to train the model, which will definitely give us a better prediction result.

5 Conclusions and Future Considerations

As our trained model could provide more than 66 percent correct answers to the future predict problem, it is useful in our daily life. The future prediction is always the most difficult part in all machine learning problems. The trained model could give us a pretty good prediction. And there is no doubt the model will help the kkbox music company to make better decisions.

Considering increasing the accuracy of the model, we could use work-stacking method to make a better model. We may need to train different models and give the output of different models a different weight. By combining all the output as the final output and choosing good weights, we will have a better result and a more accurate prediction.

References

https://www.kaggle.com/c/kkbox-music-recommendation-challenge
https://www.kaggle.com/yohanb/yet-another-lightgbm-starter-8gb-ram
IN[8]
https://www.kaggle.com/garethjns/microsoft-lightgbm-with-parameter-tuning-0-823?scriptVersionId=1751960