Machine Learning Nanodegree

Capstone Report

Yuanqi Wang

January 15, 2018

I Definition

A Project Overview

This project provides analysis for the "WSDM - KKBox's Music Recommendation Challenge" from Kaggle.[1] This project aims to build a better music recommendation system for KKBox, a music streaming service in Asia, by constructing a model that predicts how likely a user will listen to a particular song *again*.[2]

Kaggle provides a train dataset and a test dataset with song id, user id and system information. The training set has the target variable which has a value of either 1 or 0. 1 indicates that a particular user listened to a particular song more than once. Kaggle provides additional datasets with member information (e.g. age, city) and song information (e.g. length, genre, composer, artist). The performance metric is Area Under the Receiver Operating Characteristic (ROC) curve.

Since the task is to classify whether or not a user is going to listen to a song again, this is in essence, a binary classification problem. Classification problems fall into the umbrella of supervised learning — perhaps one of the most common types of Machine Learning. Classification algorithms have wide applications, such as spam filtering, fraud detection, character detection. Some algorithms for tackling a classification problem include:

- Logistic Regression: estimates the probability of dependent variable belonging to a class on its input features.[3]
- Decision Tree: splits the datasets and classifies instances based on certain criterion.[4]
- Support Vector Machine ("SVM"): finds a separating boundaries that maximizes the margin.[5]
- Ensemble Method: combines the decisions made by a set of models.[6]

[1] <www.kaggle.com/c/kkbox-music-recommendation-challenge.>

[2] KKBox currently employs "a collaborative filtering based algorithm with matrix factorization and word embedding in their recommendation system."<www.kaggle.com/c/kkbox-music-recommendation-challenge.> A recommender system is "a technology that is deployed in the environment where items (products, movies, events, articles) are to be recommended to users (customers, visitors, app users, readers) or the opposite." <www.medium.com/recombee-blog/recommender-systems-explained-d98e8221f468.> The details of a recommendation system is outside the scope of this project.

[3] Hosmer Jr, David W., Stanley Lemeshow, and Rodney X. Sturdivant. Applied logistic regression. Vol. 398. John Wiley & Sons, 2013.

[4] Murthy, Sreerama K. "Automatic construction of decision trees from data: A multi-disciplinary survey." Data mining and knowledge discovery 2, no. 4 (1998): 345-389. Two well-known decision trees algorithms are C4. 5 (developed by Ross Quinlan) and CART (Classification and Regression Trees). I will be using CART algorithm provided by scikit-learn. See <www.scikit-learn.org>

[5] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20, no. 3 (1995): 273-297; Kotsiantis, Sotiris B., I. Zaharakis, and P. Pintelas. "Supervised machine learning: A review of classification techniques." (2007): 3-24.

[6] Dietterich, Thomas G. "Ensemble methods in machine learning." Multiple classifier systems 1857 (2000): 1-15. Examples of ensemble methods include Bagging, AdaBoosting, and GradientBoosting.

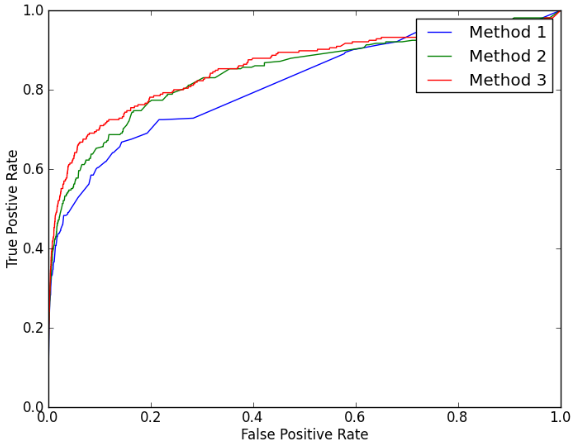
B Problem Statement

This project builds a classification model that predicts how likely a specific user is going to listen a particular song again. I plan to build this model by first exploring and processing the data. This process involves dealing with abnormal and missing values, generating new features when necessary, turning variables into usable format, etc. The next step is to find a high-performance model that could predict probability for a binary class. I attempt to predict the probability of the users listening to particular song again in the testing set, by using the `predict_proba` method. This method is supported by a variety of classification models provided by scikit-learn including `LogisticRegression,` `RandomForestClassifier`, and`GradientBoostingClassifier`. In addition, I will also try the models popular on Kaggle, such as `LightGBM`.

C Metrics

The evaluation metric is the "area under the ROC curve" (AUC) between the predicted probability and the observed target, as described in the Kaggle competition rules.[7] AUC is an effective metric for classification problems, which suits the objective of this project.[8]

The ROC curve, as shown below, plots on a plane with False Positive Rate (FPR) on the x-axis and True Positive Rate (TPR) on the y-axis.[9] Each point on the ROC curve represents a possible classification threshold. The AUC summarizes the entire location of the ROC curve. A model with a larger AUC is likely to also have better overall predictions.



[7] <www.kaggle.com/c/kkbox-music-recommendation-challenge#evaluation>

[8] Hajian-Tilaki, Karimollah. "Receiver operating characteristic (ROC) curve analysis for medical diagnostic test evaluation." Caspian journal of internal medicine 4, no. 2 (2013): 627.

[9] TPR is calculated as true positives divided by all positives. FPR is calculated as false positives divided by all negatives.

Using AUC avoids the problem of making an arbitrary classification threshold. To turn the continuous probability predicted by a model into the binary target variable of 1 and 0, I need to make an arbitrary threshold to classify instances into 1 or 0. For example, when default threshold is 0.5, if the predicted probability is higher than 0.5, the model predicts 1 for the target variable; if the predicted probability is lower than 0.5, the model predicts 0 for the target variable. However, if this arbitrary threshold of 0.5 changes, the distribution of the predictions will also change. The AUC, on the other hand, does not rely on an arbitrary threshold.[10] For AUC, it is the ranking of the probability that matters. Thus, AUC is an appropriate metric to use.

II Analysis

A Data Exploration

i Summary

Kaggle provides all the datasets. It includes five separate files containing information about the users, songs, and the system environment when the event is triggered:[11]

- `train and test:`
  - `msno:` user id
  - `song_id:` song id
  - `source_system_tab:` the name of the tab where the event was triggered (e.g. ”my library”, ”explore”)[12]
  - `source_screen_name:` the name of the layout a user sees.(e.g. ”Local playlist more”, ”Radio”)
  - `source_type:` the entry point a user first plays music on mobile apps (e.g. ”top-hits-for-artist”, ”local-library”)[13]
  - `target:` target variable (target=1 means there are recurring listening event(s) triggered within a month after the users very first observable listening event, target=0 otherwise)

The number of observations is 7,377,418 in train and 2,556,790 in testing set. All variables except for the target variable are string variables. There are 30,755 unique

[10] Hajian-Tilaki, Karimollah. ”Receiver operating characteristic (ROC) curve analysis for medical diagnostic test evaluation.” Caspian journal of internal medicine 4, no. 2 (2013): 627.

[11] See: www.kaggle.com/c/kkbox-music-recommendation-challenge/data

[12] System tabs are used to categorize KKBOX mobile apps functions.

[13] An entry point could be album, online-playlist, song. . . etc.

users in the training set and 25,131 in testing set. 21,483 users are in both train and testing sets. The testing set has 3648 users that do not exist in training set. Similar patterns can be found for `song_id, source_system_tab, source_screen_name, source_type as presented in table below:`[14]

variable	in train	in test	in both
song id	359,966	224,753	164,880
source system tab	10	10	10
source screen name	21	23	21
source type	13	13	13

- `members:`
  - `msno:` user id
  - `city:` e.g. 1, 3, 7
  - `bd:` age (e.g. 0, 24, 35)
  - `gender:` e.g. male, female, NaN
  - `registered_via:` registration method (e.g. 4, 7, 9)
  - `registration_init_time:` format %Y%m%d
  - `expiration_date:` format %Y%m%d

  The number of observation is 34,403. The count of unique values is 21 for `city`, 95 for `bd`, 2 for `gender`, 6 for `registered_via`, 3,862 for `registration_init_time` and 1,484 for `expiration_date`.

- `songs:`
  - `song_id:`
  - `song_length:` in ms
  - `genre_ids:` Some have multiple genres. They are separated by " | " (e.g. 726, 864|857|850|843)
  - `artist_name:` e.g. Art Blakey & The Jazz Messengers, Coldplay
  - `composer:` e.g. TEDDY|FUTURE BOUNCE|Bekuh BOOM, Robert Schumann (1810-1856)
  - `lyricist:` e.g. Park|Se Jun / Han|Jun Nakata, Deep White/Arys Chien

[14] Note that in the summary section, I do not count missing value (NaN) as a unique value.

  - `language`: e.g. 3, 31, 52

The number of observation is 2,296,320. The count of unique values is 1,045 for `genre_ids`, 222,363 for `artist_name`, 329,824 for `composer`, 110,926 for `lyricist`, and 10 for `language`. `artist_name, composer and lyricist` are in various languages including Traditional Chinese, Thai, Japanese, Korean, and English. When there are multiple artists for the same song, the artists are separated by "|", "+", "&", "/", and "\\", etc. This applies to lyricist and composer as well.

- `song_extra_info`:
  - `song_id`
  - `name`: song name (e.g. Aloha| E Komo Mai (Theme Song from Lilo & Stitch: The Series), Let Me Love You)
  - `isrc`: International Standard Recording Code, theoretically can be used as an identity of a song.[15] (e.g. TWUM71200043, USSM11301446)

The number of observation is 2,295,971. The count of unique values is 1,168,979 for `name`, 1,806,825 for `isrc`.

After merging the `train, members, songs,` and `song_extra_info`, there are a total of 19 features (excluding `target`).

ii Target is balanced

Approximately 50% of the target variable in the training set is equal to 1. The data is well balanced.

iii New features

From the `isrc` variable, country, year and record label can be inferred. For each song, the number of genres it belongs to, the number of artists, composers, lyricists are counted, respectively. I generated binary variables that identifies if the composers, the artists, and the lyricists for a song are the same entity.

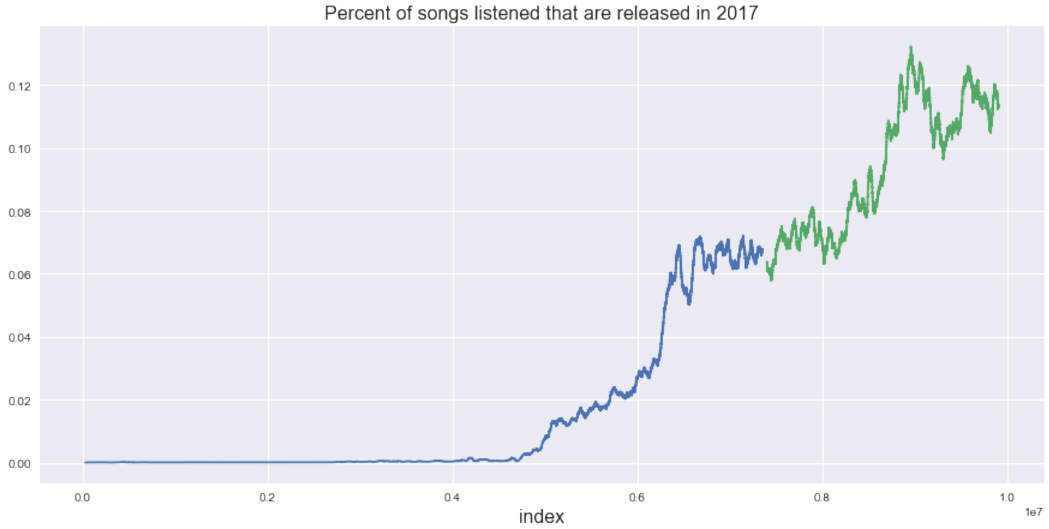
iv Data is ordered chronologically

As Kamil Kaczmarek demonstrated in his work, the train and test data are implicitly ordered chronologically.[16] The chart below shows how often songs released in

[15] However, what worth to note is, ISRCs generated from providers have not been officially verified; therefore the information in ISRC, such as country code and reference year, can be misleading/incorrect. Multiple songs could share one ISRC since a single recording could be re-published several times.

[16] <www.kaggle.com/kamilkk/i-have-to-say-this>

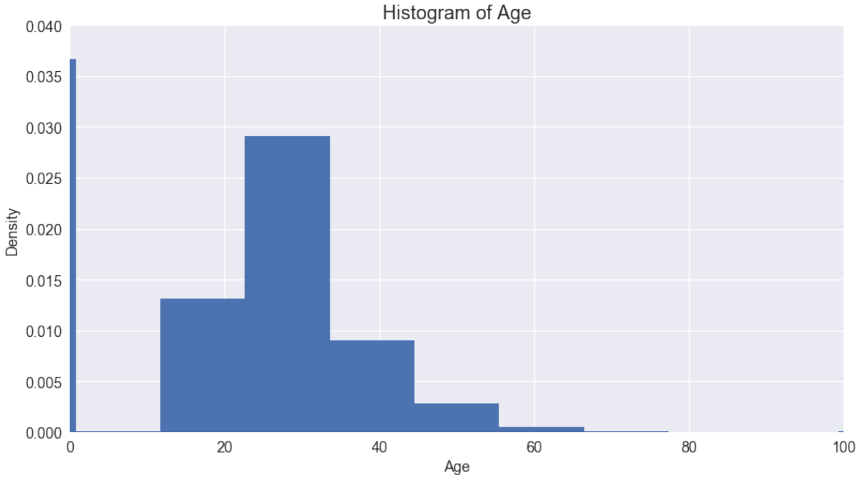
2017 appear in a moving window of 50,000 rows, sorted by index. The blue line represents data from the training set, and the green line represents data from the testing set. The x-axis is the index. This chart reveals that songs that came out in 2017 do not appear in the first 3,000,000 rows. The frequency of 2017 song starts to increase afterwards. This shows that the data is ordered by time.



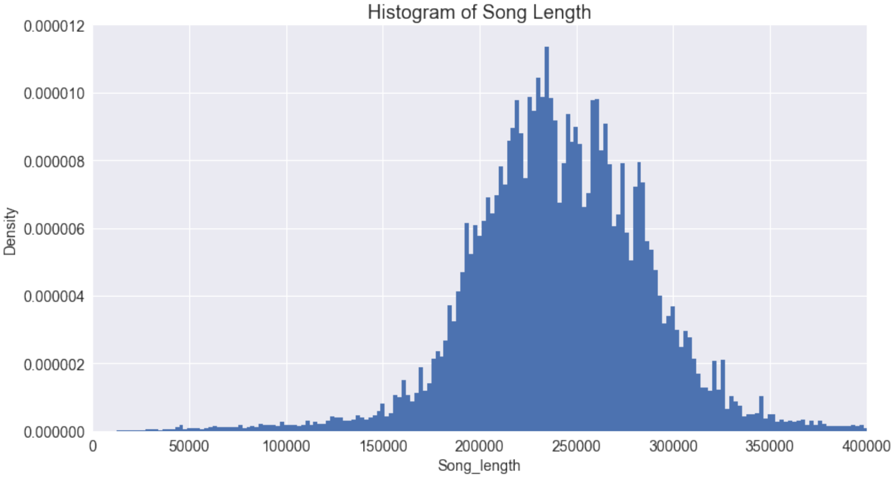
Given that the data is ordered by time, I should be careful about mixing information from the future with information from the past. For example, I should not shuffle the data to create a cross validation set. However, this time-series element also provides opportunities to generate new features based on when the event occurs. For example, I could measure the interval between a song is played by different users.

v Abnormalities

To detect abnormalities, I start with the basic statistics of the numeric features: age and song length. For age, the min and max is -43 and 1,051, respectively. These extreme values show that the age variable contains outliers. After removing the negative values, and values over 100, the histogram of age below shows that a significant amount of age is 0 or around 100. These values are replaced with the median of age.



For song length, the min and max is 1.4 second and 10,851 second (181 minutes). The histogram below shows a bell-shape distribution, which fits my expectation.

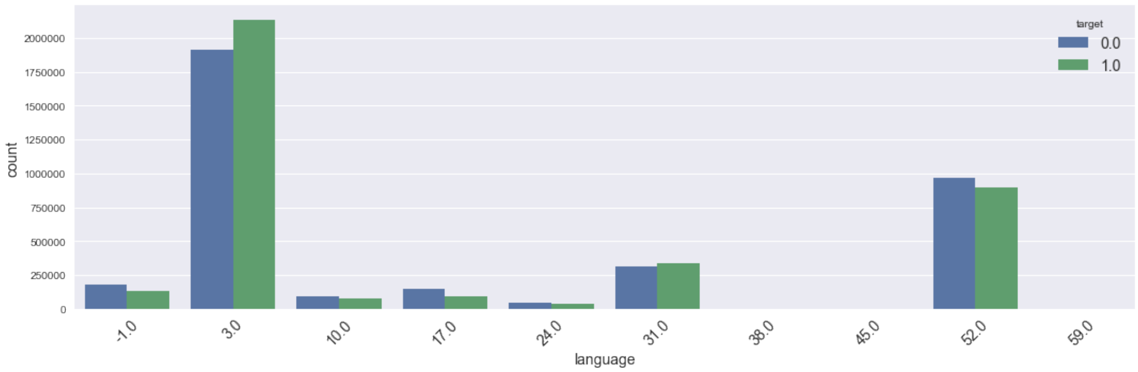


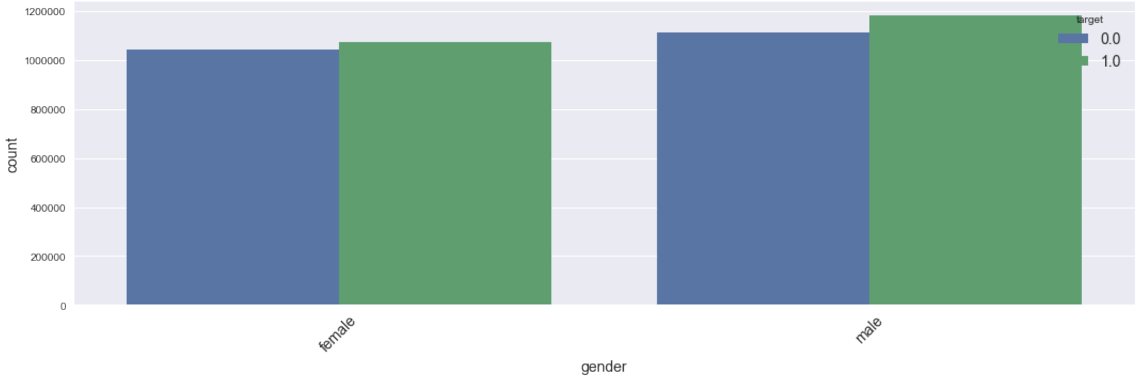
vi Missing

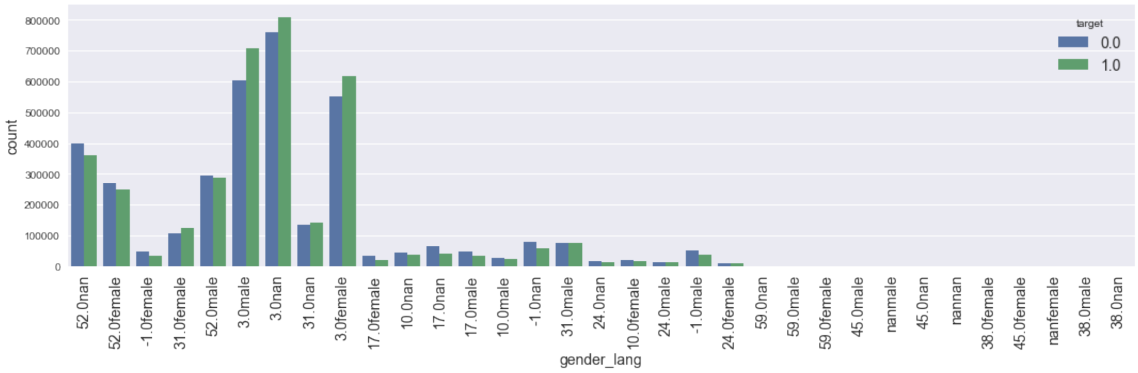
The following variables have missing values: `source_screen_name` (5.8%), `source_system_tab` (0.2%), `source_type` (0.3%) `song_length` (0.001%), `composer` (23.1%), `lyricist` (44.3%), `genre_ids` (1.6%), `language` (0.002%), `gender` (40.4%), `name` (0.02%), `isrc` (7.8%) For categorical variables, I fill the missing value as a new category ”nan”, or an new integer when the categorical variable is represented in integer. For song length and age, the median value, respectively.

B Exploratory Visualization

As shown in this plot below, certain variables have a higher proportion of target being 1 while others have similar level of target being 1. For example, when `language` is equal to 3, the count of target being 1 is visibly higher than the count of target being 0. While for gender, the difference between target being 1 or 1 for female is small. However, a combination of these patterns reveals an interesting pattern: when language is 3, the ratio of male having a target of 1 is higher than that of female. Therefore, I will generate combinations of categorical variable to reveal such patterns.







C Algorithms and Techniques

As discussed above, I‘ve tried `RandomForest`, and `LightGBM`(Light Gradient Boosting Machine) to predict the probability of target being 1. Give the large size of the training set, I started with only the last 1 million rows so as to compare different algorithms and tune parameters more efficiently. GridSearchCV is used to fine-tune `RandomForest`.

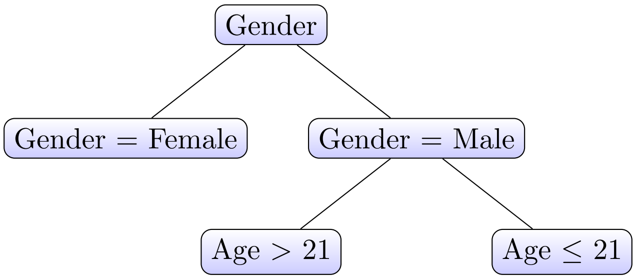
LightGBM is a gradient boosting framework that uses decision tree based learning algorithm, while RandomForest employees a bagging framework using tree learners. [17] I will discuss the basic concepts of LightGBM and RandomForest in the next section:

i Decision Tree

Both LightGBM and RandomForest use decision tree as the base weak classifier. Decision tree is one of the most common machine learning algorithms. The basic idea of decision tree is to breakdown the data into smaller groups based on certain rules. The tree is built top-down from a root node, and by partitioning the data into subsets that contain instances with similar values (homogeneous). This homogeneity can be measured in different ways. In lightGBM, it is measure by ”cross entropy”.[18] To give a specific example of decision tree, imagine a dataset that consists age and gender information of five people, and whether they play video games or not:

ID	Age	Gender	Play Video Game?
Britta	25	Female	No
Shirley	30	Female	No
Abed	20	Male	Yes
Troy	20	Male	Yes
Pierce	60	Male	No

If we split the dataset by gender first, then if age is larger 21, then we end up with three nodes. Within each node, the answers to do they play video game or not are the same. The top left node consist of two females who do not play video games. The bottom left consist of one male older than 21 who does not play video games, and the bottom right node consist of two male who is younger than 21 who do play video games. Now if we were to predict a new person, Jeff, 35 and male, plays video game or not, by following the tree, we would arrive the answer that he does not.



[17] Pushkar Mandor, “What is LightGBM, How to implement it? How to fine tune the parameters?” <www.medium.com>; Breiman, Leo. ”Random forests.” Machine learning 45, no. 1 (2001): 5-32.

[18] <www.ightgbm.readthedocs.io/en/latest/Parameters>

ii Boosting

LightGBM uses a boosting ensemble method. Boosting consists of sequentially applying the base learner to re-weighted versions of the initial dataset adaptively.[19] Observations misclassified in the last round get a larger weight on the next iteration, while weights decrease for observations classified correctly previously. This change in weights forces the algorithm to give extra consideration on the hard-to-classify observations.[20]

iii Bagging

RandomeForest uses a bagging ensemble method. Bagging consists of sequentially selecting a random sample with replacement of the training set, and computing the base classifier, in order to obtain a bundle of classifiers. The final classifier (or the Forest) is the average over or the majority vote of all base classifiers.[21] In addition, features are always randomly permuted at each split. This is also referred to as "feature bagging."[22]

iv Parameters

LightGBM uses leaf-wise tree growth algorithm. This leaf-wise strategy grows the tree by splitting the data at the nodes with the highest loss change. There are a variety of parameters in LightGBM to fine-tune. I list a few important parameters that I used to define the problem, avoid over-fitting and achieve better accuracy:

- boosting: defines the type of algorithm you want to run.

  default = 'gbdt' (traditional Gradient Boosting Decision Tree).

- objective: specifies the application of the model, whether it is a regression problem or classification problem.

  default = 'regression', changed to 'binary'.

- learning_rate: determines the impact of each tree on the final outcome. It works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates.

  default = 0.1, changed to 0.05

[19] Lemmens, Aurlie, and Christophe Croux. "Bagging and boosting classification trees to predict churn." Journal of Marketing Research 43, no. 2 (2006): 276-286.

[20] Lemmens, Aurlie, and Christophe Croux. "Bagging and boosting classification trees to predict churn." Journal of Marketing Research 43, no. 2 (2006): 276-286.

[21] Lemmens, Aurlie, and Christophe Croux. "Bagging and boosting classification trees to predict churn." Journal of Marketing Research 43, no. 2 (2006): 276-286.

[22] Sklearn RandomForestClassifier documentation, <www.scikit-learn.org>

- `num_leaves`: specifies the number of leaves in full tree. Large number may cause over-fitting. Can use this to deal with over-fit.

  default = 31, changed to 64.

- `min_data_in_leaf`: specifies minimal number of data in one leaf. can use this to deal with over-fit.

  default = 20.

- `max_depth`: limits the max depth for tree model.

  default = -1, means no limit. I changed it to 10.

- `bagging_fraction`: randomly select part of data without resampling. can be used to speed up training. can be used to deal with over-fitting.

  default = 1.0, ranges from 0.0 to 1.0. I changed it to 0.9.

- `bagging_freq`: defines the frequency for bagging.

  default = 0, means disable bagging. I changed it to 1.

- `feature_fraction`: set the fraction of features the model will randomly select on each iteration. can be used to speed up training and deal with over-fitting.

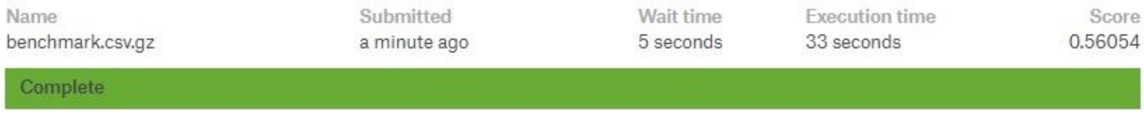
  default = 1.0, ranges from 0.0 to 1.0. I changed this to 0.8.

- `metric`: sets how the model is evaluated

  default = binary_logloss for binary classification. I changed it to auc

D Benchmark

The benchmark model use the `LogisticModel` without tuning the parameters. The input data is the training data after completing the first 4 steps of preprocessing as described in Section III. A. The AUC score is 0.56.



Name	Submitted	Wait time	Execution time	Score
benchmark.csv.gz	a minute ago	5 seconds	33 seconds	0.56054

Complete

III Methodology

A Data Preprocessing

Here are the steps I take to process both train and test dataset before feed the data into the model:

1. Clean data: for `‘genre_ids’`,`‘artist_name’, ‘composer’, ‘lyricist’`, I removed trailing blanks, capitalized all string values, and replaced symbols such as “|”, “+”, “&”, “/”, and “\\” with space. For age variable, I replaced values smaller than 0 and larger than 70 with missing values. The missing values will be imputed in later steps.

2. Generate first set of new features: I generated the count of unique artists, composers, lyricists, and genres for each song by counting the number of spaces. I generated binary variables that identify if composer, artist and lyricist are the same entity. I generated song year, song country, and record label based on `isrc`. That is, the first two digits in `isrc` represents the country the song is released in, the next three digits represents the record label the song is associated with, the next two digits are represents the song in which the song is released. In addition, I generated a proxy for time by using the index of the each instance, as the data does not have a time variable.

3. Fill in missing value: the missing values for the following categorical variables are filled in as a new category“nan”: `‘msno’, ‘song_id’, ‘gender’, ‘source_screen_name’, ‘source_system_tab’, ‘source_type’, ‘genre_ids’,‘artist_name’, ‘composer’, ‘lyricist’, ‘gender’`. The missing values for the following variable are filled with -5 (a category new to each of these features): `‘city’, ‘language’, ‘registered_via’`. The missing value for `‘song_length’` and `‘bd’` are filled with their respective median values. `‘song_year’` is filled with previous non-null value. I note that certain variables have “null” as a category, however, I treated these “null” values as a separate category than the values that are simply missing.

4. Encode label: I used the LabelEncoder from sklearn to transform non-numerical labels to numerical labels.

5. Generate second set of new features: an interesting fact of the data is that it is ordered chronologically. That means history could lend insights to the future. That is, the testing set could have features that are generated from the training set.[23]

   (a) I took approximately the last 35% of training set (the size of the testing set) as the input to the model. The rest of the training set are used as history to generate features for this 35% of the training set. Similarly, the testing set uses the training set as history to generate features as well.

   (b) I generated history-informed features for both train and testing set:

   - mean of target from history by categorical features (singles and pairs). For the final training set, these mean of target by group features are generated

[23] All credit in this step goes to Vasiliy Rubtsov, see <github.com/VasiliyRubtsov/wsdm_music_recommendations>

from its history — the first 65% of the training set. For test set, the mean of target by group features are generated from its history — the entire train set.

- count of unique values by the categorical features (singles and pairs). For both dataset, this is the count of its history.
- difference in `‘time’` between a value of categorical features appear again (singles and pairs)

The final training set that feeds into the model has 2,925,660 instances and 287 features. The reason that I used only the last part 35% of the training set is that I think that the distribution of the earlier data is different than the distribution of data from later period. As demonstrated in the Analysis section, the number of songs leased in 2017 does not even show up in the first half of the data, Therefore, I used the size of the testing set, plus 5% of the training set, as the basis for the final training set. [24]

One complication I had was when generating certain features, if I should use the entire dataset including the “future”, or look strictly at the historic data. Initially, I generated the unique count of the entire data for several categorical data, assuming that this is information shared by both the train and test set. However, after empirical evidence shows that this will leads to over-fitting of the train data and poor generalization to the unseen set. Therefore, I generate the count value using just the history.

Note that due to limitation of computational resource, I was not able to generate as many features that I'd like to experiment. I will discuss some ideas for additional features in later sections.

B Implementation

The implementation of the solution includes the following decisions: which algorithms I should use, which metrics I should rely on, which parameters I should tune and how, which datasets I should pipe into the algorithm. To be more efficient in comparing models and tuning hyper-parameters, I used only 1 million rows of the final training set and the last 500,000 rows as validation set.

I started with the following three algorithms: `RandomForest`, `AdaBoost`, and `LightGBM`. Since this is a classification problem, I used the `RandomForestClassifier` and `AdaBoostClassifier` from sklearn. For `LightGBM` model, I set the `‘objective’` to ‘binary’. To change the metric to AUC, I passed ‘roc_auc’ in the `‘scoring’` parameter for `RandomForestClassifier` and `AdaBoostClassifier`. For `LightGBM` model, I set the `metric` to ‘auc’.

[24] I recognize that I could empirically test what window of the training set resembles the most of the testing set — nevertheless it is of less importance to me and I will leave it as things to consider in the future.

To tune the hyper-parameters for each algorithm, I used GridSearchCV for `RandomForestClassifier` and `AdaBoostClassifier`. Since this is a panel data, I did not shuffle the data to create validation set. Instead, I created validation set by calling sklearn's `TimeSeriesSplit` and passing it through the models. For `LightGBM`, I used 1 million instances as training data and last 500,000 of the training set as validation to tune the hyper-parameters. Given the size of the data and the training time for each model, I choose to use `LightGBM` as the final model, because it is "optimized for speed and memory usage."[25]

One complication is that certain parameters are dependent on the size of the data. But since I only used a fraction of the data to tune the parameters, the tuned parameter based on a partial data might not work for a larger dataset. And changing this parameter might set off a chain reaction in other parameters. I tried to navigate the complicated interactions of many hyper-parameters in the `LightGBM` by using GridSearchCV with `LightGBM` as documented in its GitHub repository,[26] however it gives a deprecation warning of 'max_bin". I wasn't able to resolve this issue so I tuned the model by hand.

C Refinement

The first task is to select which model to use. The second step is to tune the hyper-parameters to improve the model based on the AUC. The three models that I have tried are: `RandomForest`, `AdaBoost`, and `LightGBM`. `LightGBM` is consistently faster.[27] It also has comparable, if not better, AUC scores.[28] Therefore I chose `LightGBM`. The second step is to tune the hyper-parameter of `LightGBM`. As I listed earlier, there are many hyper-parameter I could choose to adjust. Since `LightGBM` over-fits easily, the main focus of the hyper-parameter tuning is to achieve better accuracy while avoiding over-fitting.

I used a small learning rate with large number of iterations to get a better accuracy. Meanwhile, I adjusted `num_leaves` to control for over-fitting. Other hyper-parameters that could help with over-fitting include `min_data_in_leaf` and `max_depth`. In addition, I could choose a different boosting methods, such as "dart" ("Dropouts meet Multiple Additive Regression Trees") which have pairing hyper-parameters that deal with over-fitting. Another advantage of `LightGBM` is its speed. One way to increase the speed is using only a subset of features each iteration. This is enabled by the `feature_fraction` method. Another way is to select part of the data randomly through `bagging_fraction`. I employed both methods in my model.

[25] <www.lightgbm.readthedocs.io>

[26] <www.github.com/Microsoft/LightGBM/blob/master/examples/python-guide/sklearn_example.py>

[27] Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "LightGBM: A highly efficient gradient boosting decision tree." In Advances in Neural Information Processing Systems, pp. 3149-3157. 2017.

[28] [[demonstrate empirally.]]

For the aforementioned hyper-parameters, I adjusted one of them each time and recorded the AUC of the training set. When the hyper-parameters are set to default, the AUC of validation set was 0.69474. After iterations of tuning the hyper-parameters, the AUC of validation set increased to 0.70124.

IV Results

A Model Evaluation and Validation

The final algorithm I chose is `LightGBM`. All else equal, `LightGBM` runs much faster than `RandomForest` and `AdaBoost`. It also has AUC score similar to, or better than, the other two classifiers. The correlation between `LightGBM` and `RandomForest` is around 0.9.

The result from the final model (`LightGBM`) has an AUC of 0.70753 on the testing set. This aligns with my expectation. The AUC of the testing set ranks in top 10% of this Kaggle competition.

Since many features are based on the historic mean, historic unique counts, and time between a value next time, it should generalize reasonable well into the unseen data as the prediction relies on not only the unseen data, but also the historical data.

Finally, the final algorithm gives insight to which variables are more important both in terms of information gain and the number of time it is used to split the data. This serves as a sanity check as to the reasonableness of the solution.

B Justification

Compared to the benchmark model, the final model has a higher AUC score. This comes from a) more and better features and b) improved algorithm.

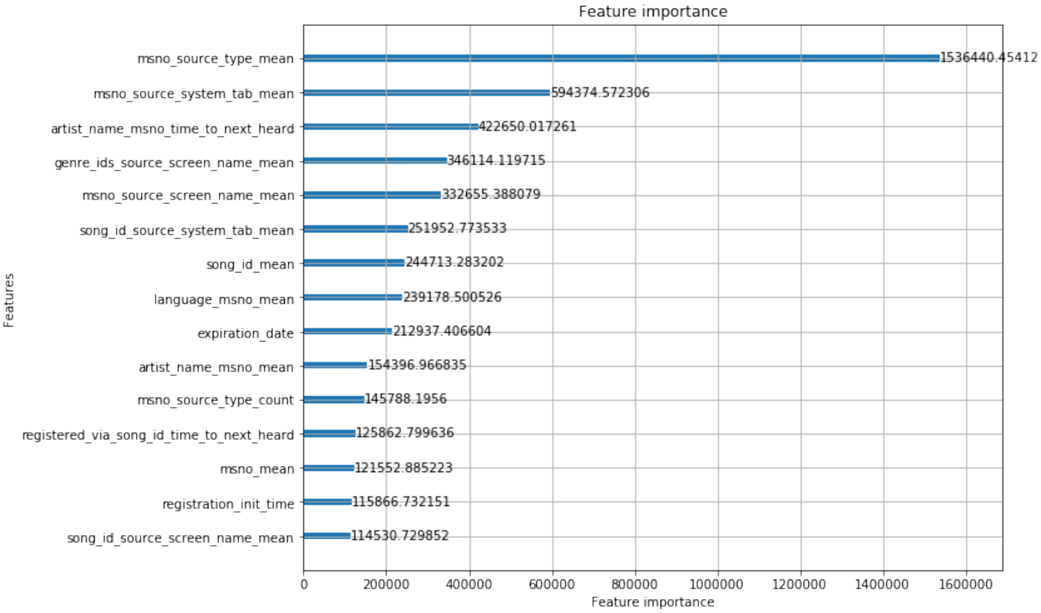
The final dataset takes into consideration of not only the user, song, environment information, but also insights from the combination of the user, song, and environment. For example, one would assume that a popular song is more likely to be listed for more than once. The measure of a popular song the number of times it has been listened to by all users. This is indeed capture by the "song_id_count" feature. Or you could imagine that a particular user, who like to listen to curated play-list repeatedly in the past, when we observe this user plays a song from a curate ply-list, the probability of that song being listen to again should be fairly high. This is also capture by the generated features.

The decision of which algorithm to use is based on mostly its training speed. All three models use decision tree as the weak classifier, however, they train the data differently. While `RandomForest` uses a bootstrap sample of the training set to grow each of its decision trees,[29] both LightGBM and AdaBoost generate a sequence of trees, each grown on the residuals of the previous tree.[30] All three models have high accuracy scores, but boosting method has slightly higher in general.[31] AdaBoost takes longer to train than RandomForest.

V Conclusion

A Free-Form Visualization

As discussed earlier, the most important features measured by 'gain - the average gain of the feature when it is used in trees (default), and split - the number of times a feature is used to split the data across all trees, include: artist_name_msno_count, msno_count, expiration_date, language_msno_count, msno_source_type_count, msno_mean, msno_source_type_count.



B Reflection

This project is to solve a binary classification problem using AUC as the metric. It started by importing, combining and merging different datasets. Then I explored the data

[29] Liaw, Andy, and Matthew Wiener. "Classification and regression by randomForest." R news 2, no. 3 (2002): 18-22.

[30] Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. "LightGBM: A highly efficient gradient boosting decision tree." In Advances in Neural Information Processing Systems, pp. 3149-3157. 2017.

[31] See, for example: Ogutu, Joseph O., Hans-Peter Piepho, and Torben Schulz-Streeck. "A comparison of random forests, boosting and support vector machines for genomic selection." In BMC proceedings, vol. 5, no. 3, p. S11. BioMed Central, 2011.

for unusual values such as missing, duplication, extreme values. This exploratory analysis also helped me gain ideas for cleaning and generating features later. After gaining a good understanding of the data, I preprocessed the data by handling missing and extreme values, and generate several variables that I think might be helpful. I then turned string variables that are not usable for Machine Learning problem to numbers. I use this dataset as the input to a Logistic Regression model, used as a benchmark model. After benchmarking, I selected the time window of the training set, and generated new features, based on the insight that this is a chronologically ordered data. After the training set was generated, I used three models to train on a sample of the data: `RandomForest, AdaBoost, LightGBM`. GridSearchCV technique and time serious split method for validation are used. Due to the slow training speed, `AdaBoost` dropped out of the race early on. Even though `RandomForest` and `LightGBM` have similar AUC scores, `LightGBM` is much faster. Therefore, I chose to go forward with `LightGBM`. After tuning several parameters of `LightGBM` to control the overfitting issue, I obtained a final model that improved drastically the AUC score, compared to the benchmark model.

One interesting aspect feature of this project is that it is ordered chronologically, therefore, I paid special attention to not leak information backwards when training the data.

One difficulty of this project is the hardware limitation — I simply could not generate as many features as I want because it is running out of RAM. I tried to save memory cost by changing the data type (int64 to int16), however, this might cause some truncation issues. In addition, tuning the parameters for the `LightGBM` is a very iterative process. There are many moving pieces and therefore record keeping is very important. I could also have passed it through the sklearn GridSearch function.

C Improvement

First, I could further clean the data. For example, I noticed that the same singer is represented in the data with or without an English name. In addition, there are relatively large proportion of missing variables for certain features such as age. I could run a supervised learning model, linear regression, for example, to fill the missing values. Second, there are other features I could have generated. One series of features I would like to generate is one-hot encoder based on the genres. That is, each genre would be a column, and for each genre the song belongs to,the value of that genre column will be 1, otherwise it will be 0. Currently, if a song has multiple genres, it will consider that as a separate genre than a combination of different genres. However, during implementation, this takes up too much space so I dropped this idea. Another feature I could add is based on matrix factorization, a technique used for collaborative filtering.