Machine Learning Engineer Nanodegree

Shubham Gupta

November 7, 2017

I Defination

Project Overview

Not many years ago, it was inconceivable that the same person would listen to the Beatles, Vivaldi, and Lady Gaga on their morning commute. But, the glory days of Radio DJs have passed, and musical gatekeepers have been replaced with personalizing algorithms and unlimited streaming services.

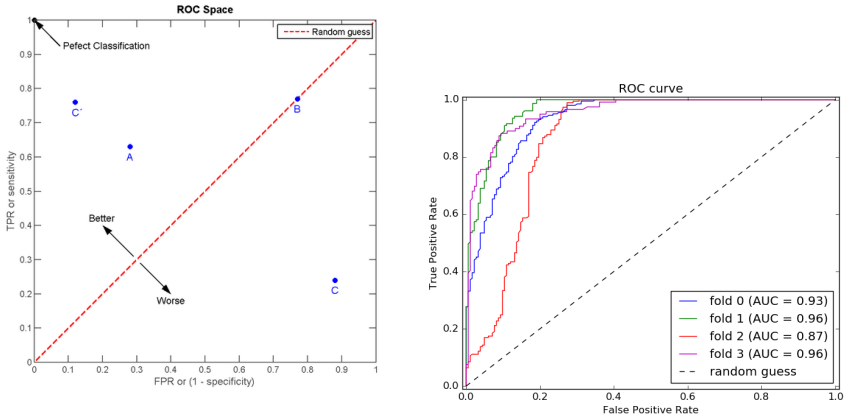
While the publics now listening to all kinds of music, algorithms still struggle in key areas. Without enough historical data, how would an algorithm know if listeners will like a new song or a new artist. And, how would it know what songs to recommend to its brand new users. Since content recommendation is at the heart of most subscription-based media stream platforms. Hence a good recommendation system can vastly enhance user experience and increase user engagement.

Problem Statement

The goal of the project is to exploit the provided user history and music metadata to develop a recommendation system. Given a copy of user listening history, the aim of the project is to predict what songs a set of predetermined users would listen to during the next month. The dataset is from KKBOX, Asia's leading music streaming service, holding the worlds most comprehensive Asia-Pop music library with over 30 million tracks. They currently use a collaborative filtering based algorithm with matrix factorization and word embedding in their recommendation system but believe new techniques could lead to better results. I will be tackling this problem as a binary classificaition problem and will apply supervised learning techniques to classify whether the user will listen to a song the next month or not.

Metrics

The predictions are evaluated on area under the ROC (AUROC) curve between the predicted probability and the observed target. The ROC (Receiver



(a) The ROC space and plots of the four prediction examples.

(b) The ROC curve of four predictors with thier AUC score.

Figure 1: The AUROC curve.

Operating Characteristic) curve is created by plotting the true positive rate (TPR) or recall against the false positive rate (FPR) or probability of false alarm at various threshold settings. A ROC space is defined by FPR and TPR as x and y axes, respectively, which depicts relative trade-offs between true positive and false positive. Since TPR is equivalent to sensitivity and FPR is equal to 1-specificity, the ROC graph is sometimes called the sensitivity vs (1-specificity) plot.

The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% sensitivity (no false negatives) and 100% specificity (no false positives). The (0,1) point is also called a *perfect classification*. A random guess would give a point along a diagnol line (the so-called *line of no-discrimination*) from the left bottom to the top right corners. Therefore, a classifier that does a very good job separating the classes will have an ROC curve that hugs the upper left corner of the plot. Conversely, a classifier that does a very poor job separating the classes will have an ROC curve that is close to the diagnol line.

Since this project is based on the WSDM Kaggle competition, I will take the leaderboard score as my final evaluation.

II Analysis

Data Exploration

The data is provided by the WSDM-KKBox Music Recommendation Challenge on Kaggle. KKBOX provides a training data set consists of information of the first observable listening event for each unique user-song pair within a specific time duration. Metadata of each unique user and song pair is also provided. The various data files along with their data fields are described below.
train.csv: The training file consists of 7377418 rows and 6 columns.

- msno: user id
- song_id: song id
- system_source_tab: the name of the tab where the event was triggered. System tabs are used to categorize KKBOX mobile apps functions. For example, tab my library contains functions to manipulate the local storage, and tab search contains functions relating to search.
- source_screen_name: name of the layout a user sees.
- source_type: an entry point a user first plays music on mobile apps. An entry point could be album, online-playlist, song .. etc.
- target: this is the target variable only present in train.csv. target=1 means there are recurring listening event(s) triggered within a month after the users very first observable listening event, target=0 otherwise.

test.csv: The test file consists of 2556790 rows and 6 columns.

- id: row id (will be used for submission, only present in test.csv)
- The remaining columns of the test.csv are same as the train.csv, except the test.csv does not contain target variable.

songs.csv: The songs. Note that data is in unicode. The songs file consists of 2296320 rows and 7 columns. Each row corresponds to a unique song id.

- song_id
- song_length: in ms
- genre_ids:genre category. Some songs have multiple genres and they are separated by
- artist_name, composer, lyricist, language.

members.csv: The members file consists of 34403 rows and 7 columns. Each row corresponds to a unique user id.

- msno, city, gender
- bd: age.Note: this column has outlier values, which is tackled later.
- registered_via: registration method
- registration_init_time: format %Y%m%d
- expiration_date: format %Y%m%d

The train.csv doesn't include any information about the user and the songs played by the user except for their user ids and song ids repectively. Hence, these information must be extracted from songs.csv and members.csv and merged with the train.csv.

The train and test file consists of 359966 and 224753 unique song ids respectively. Although 26.6% of the song ids in the test file are not present in train.csv. Similarly, the train and test file consists of 30755 and 25131 unique user ids(msno) respectively. Although 14.51% of user ids in the test file are not present in train.csv. This poses a problem in evaluating a trained model, since certain song ids and user ids present in test.csv are absent in train.csv.

Exploratory Visualization

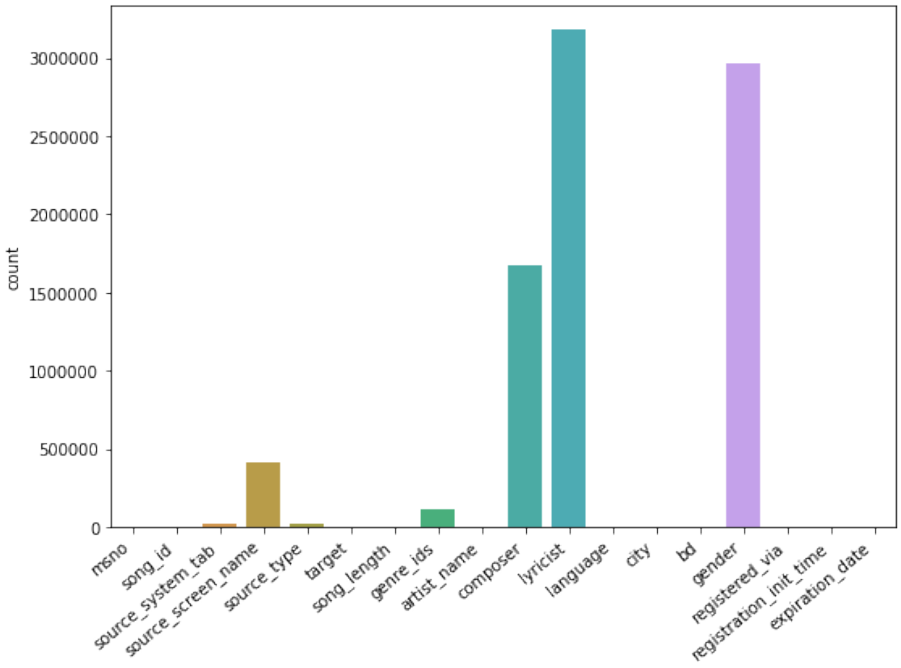
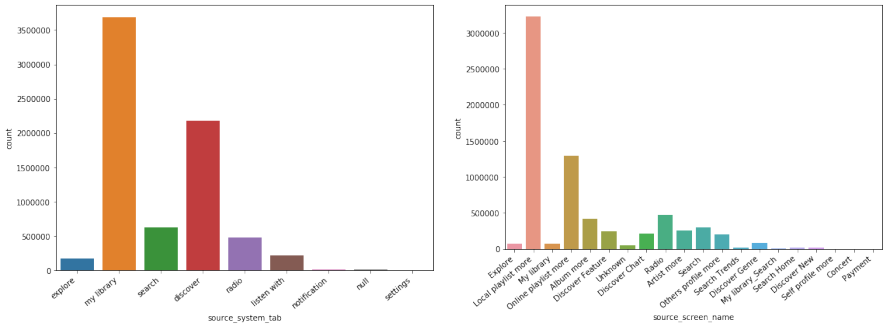


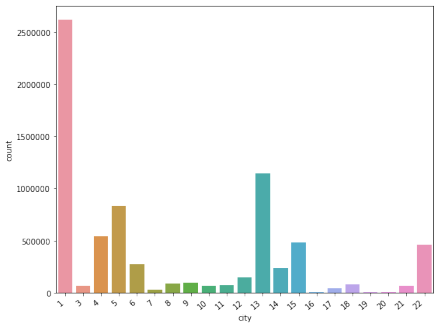
Figure 2: Null values in data.

Figure 2 represents the number of null values in each feature/column of the training data. From the figure we can see that the features 'composer', 'lyricist' and 'gender' contain a large number of null or nan values. Such a large number of null values will hinder a model to learn from the data. Hence, these columns must be removed from the data. While the null values in the remaining features are small in number and can be imputed before training the model.

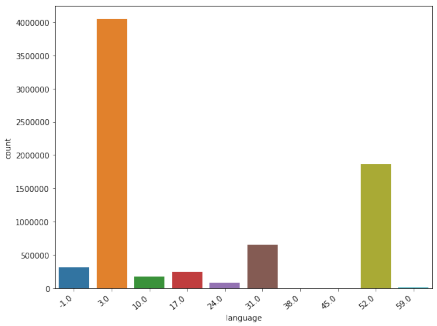


(a)

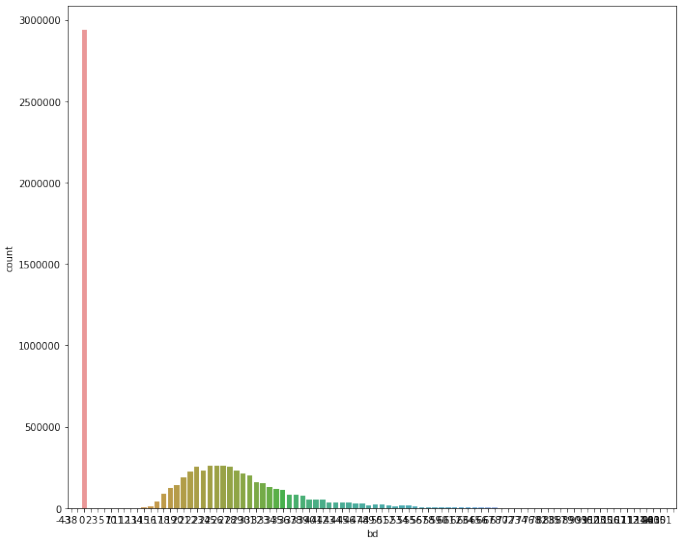
(b)



(c)



(d)



(e)

Figure 3: Countplot of different features of the data.

Figure 3 shows count plot of different features of the data. From the figure we can see which variable in each feature occurs the most. For example, in figure 3:(d), most of the songs are listened in language labeled as '3.0'. The figure 3:(e) represents the count plot for feature 'bd', which represents the age of the user. In the figure we can see that a large number of users have age labeled as '0' and on futher analysis it is found that the median of the feature 'bd'(age) is also zero, which seems to be misleading as age cannot be zero. Hence this feature should be removed from the data.

Algorithms and Techniques

(a) Level-wise tree growth in XGBoost. (b) Leaf wise tree growth in Light GBM.

Figure 4: Boosted Decision Trees

The KKBOX's Music Recommendation problem is a binary classification problem, hence I will be using supervised learning algorithms to train the model. The problem dataset includes both numerical and categorical features. The numerical features consists of outliers, while some of categorical features are prelabeled integers (such as genre) and consists of a large number of unique values. Decison trees, a tree based supervised learning algorithm are immune to outliers and can handle large number of categorical features. Therefore, for this problem I will be using two famous boosted decision tree algorithms to train the model, Light Gradient Boost(Light GBM) algorithm and XGBoost algorithm.

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm. Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy. Also, it is suprisingly very fast, hence the word 'Light'.The following parameters can be tuned to optimize the Light GBM classifier:

- num_leaves: Set the number of leaves to be formed in a tree.
- max_depth: Specifies the maximum depth to which tree can grow.
- bagging_fraction: Used to perform bagging for faster results.
- max_bin: Setting it to high value can improve accuracy but slow down training process.
- learning_rate: Sets the rate at which the model learns.

The second algorithm used is XGBoost. Technically, XGBoost is a short form for Extreme Gradient Boosting, which is similar to gradient boosting framework but more efficient. It has both linear model solver and tree learning algorithm. For this problem, I will be using the tree learning algorithm, which splits the tree level-wise to find the best fit.Some of the parameters that can be tuned to optimize this classifier are as follows:

- eta: Step size shrinkage used in update to prevent overfitting.
- gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree.
- max_depth: Maximum depth of a tree.
- eval_metric: evaluation metrics for validation data.
- min_child_weight: Minimum sum of instance weight needed in a child.

Benchmark

For this problem, the benchmark score set by the WSDM-KKBox Music Recommendation Challenge is 0.60643. I will use this auroc score as the benchmark for the trained models.

III Methodology

Data Preprocessing

The 'train.csv' file deos not contain any information about the songs played by the user. Similarly, it also doesn't include any information about the user. Thus the song and user details need to be imported from 'songs.csv' and 'members.csv' respectively. The 'songs.csv' and 'members.csv' are merged with 'train.csv' on 'song_id' and 'msno' respectively.This forms our training dataset. A similar merge is performed on 'test.csv' to form the testing data.

In Exploratory Visualization section, we saw that the columns 'composer', 'lyricist' and 'gender' contain a very large number of 'null' values. Hence, these columns are removed from both the training and testing dataset. The remaining null values in the training and testing data are replaced by a contant value. Next, the columns 'registration_init_time' and 'expiration_date' contain registration and expiration dates in YYmmdd format. From each of the columns year, month and date are extracted and stored in seperate columns for both training and testing data. After the extraction, the columns 'registration_init_time' and 'expiration_date' are removed from both training and testing data. Also, the feature 'bd', as seen in the Exploratory Visualization section, is misleading and hence is removed from both training and testing dataset.

Since, the training and testing data consists of categorical features, hence we implement a preprocessing technique called 'Label Encoding' which is used

to transform non-numerical labels to numerical labels. Finally, we produce a validation set from the training data to check the accuracy of the trained model.

Implemenation

I trained the model using two boosted decision tree algorithms, Light GBM and XGBoost. First, I implemented the Light GBM algorithm. Before training the model, the training and validation dataset is converted into a light GBM Dataset using lightgbm library.

```
d_train = lgb.Dataset(X_train, label=y_train)
d_valid = lgb.Dataset(X_valid, label=y_valid)
```

Next a watchlist is created to keep a watch on the auroc score of training and validation dataset. Now the parameters of the classifier are initialised with random values.

- learning_rate: 0.3
- application: binary (setting classification to binary classification)
- max_depth: 8
- num_leaves: 2**8
- metric: 'auc' (Setting evaluation metric as Auroc metric.)
- max_bin: 128

The model is trained for '300' num_boost_rounds/iterations with early_stopping_rounds set to 10 i.e. training the model until the validation scores doesn't improve for 10 rounds. Finally, the trained model is used to predict the labels of the test dataset.

```
watchlist = [d_train, d_valid]
model=lgb.train(params, train_set=d_train, num_boost_round=300,
        valid_sets=watchlist, early_stopping_rounds=10,
        verbose_eval=10)
```

Next, the model is trained using the XGBoost algorithm. But, before training the model, the training and validation datasets are converted into DMatrices using XGBoost library.

```
d_train = xgb.DMatrix(X_train, label=y_train)
d_valid = xgb.DMatrix(X_valid, label = y_valid)
```

Since, the parameters of XGBoost are similar to that of Light GBM, hence to produce a comparable model, the parameters of XGBoost are tuned to the same values to which the Light GBM paramters were tuned.

- objective: 'binary:logistic' (setting classification to binary classification with output predicting probability of the labels.)
- eta: 0.3

- max_depth: 8
- eval_metric: 'auc'(setting evaluation metrics to Auroc.)
- max_bin: 128
- max_leaf_nodes: 2**8

Now a watchlist is created and the model is trained for 300 iterations/num_boost_rounds with 'early_stopping_rounds' set to 10 i.e. the model will train until the validation auc doesn't improve for 10 rounds. Finally, the model is used to predict labels of the test dataset.

```
watchlist = [(d_train, 'train'), (d_valid, 'valid')]
model = xgb.train(params,d_train,300,watchlist,
                  early_stopping_rounds=10,verbose_eval=10)
```

Refinement

	Light GBM	XGBoost
training auc score	0.806967	0.804474
validation auc score	0.66087	0.659973
testing auc score	0.65243	0.65239
execution time	550 sec	2289 sec

Table 1: Results of Initial Solution.

Table 1 shows the results of the initial solution implemented for the problem. From the table, we can see that the testing score (or Leaderboard score on Kaggle) attained by the two models is better than the benchmark set for the problem. The validation auc score and testing score of the Light GBM model is better than that of XGBoost model. Although, the difference between the testing auc scores of the two models is very small, but on comparing the execution time we can see that Light GBM model is approximately four times faster than XGBoost. Therefore, Light GBM seems to be a better model.

We further tune the parameters of Light GBM to improve the testing/leaderboard score. Parameters can be tuned by either using a cross-validation technique (such as RandomSearchCV or GridSearchCV) or manually. Since using a cross-validation technique is computationally expensive, hence I tuned the parameters manually. I tuned 'learning_rate', 'num_leaves', 'max_bin' and 'max_depth' to produce better results.

Table 2 represents the testing auroc score achieved by tuning various parameters of the Light GBM model. For the first set of parameters, I decreased the learning rate and increased the num_leaves and max_depth to increase the accuracy of the model. This set of parameters produced a testing auroc score of

S.No.	num_boost_rounds	max_depth	learning_rate	num_leaves	max_bin	testing auc score
1.	300	10	0.15	2**10	128	0.65652
2.	500	15	0.10	2**15	256	0.65606
3.	500	16	0.10	2**10	256	0.66027

Table 2: Testing scores at different tuned parameters of Light GBM model.

0.65652, which is better than the initial results. In the next set of parameters, I further increased the value these parameters along with max_bin to increase accuracy. Since increasing max_bin makes the training slower, hence I increased the number of iterations(num_boost_rounds) for which the model trains. This set produced a testing auc score of 0.65606, which is less than the previous score. Hence we can say that the model is overfitting. Therefore, in our third set of parameters, I decreased the value of num_leaves, to prevent overfitting. This set produced a testing auc score of 0.66027, which is the highest of all the previous scores.

From Table 2, we can see that the third set of the parameters produce the highest testing auc score of '0.66027'. Hence this configuration of parameters is selected as the final solution for the problem.

IV Results

Model Evaluation and Validation

Table 1 clearly shows that Light GBM model beats XGBoost model both in terms of accuracy(AUROC score) and execution time. In fact Light GBM is almost four times faster than XGBoost. Hence, Light GBM is selected to implement the final solution of the problem. Table 2 shows the values to which various parameters of Light GBM model are tuned to obtain more accuracy.

The second set of parameters in Table 2 shows that on setting 'num_leaves' to a large value leads to overfitting and hence decrease in testing auc score. In the third set, the parameters 'num_leaves' is set to a lower value while increasing the 'max_depth' to which the tree can grow. This produces a boosted tree that doesn't overfit while producing a higher testing auc score. Thus the third set of parameters (see Table 2) are selected as the final parameters for the model. Thus, the final model is a Light GBM model with the following parameters:

- learning_rate: 0.10
- application: 'binary'
- max_depth: 16
- num_leaves: 2**10

- metric: 'auc'
- max_bin: 256
- num_boost_rounds: 500

The model achieves a high auroc score of 0.66027 on the testing data, which is not only unseen but also consists of user and song ids that are absent in the training dataset (see Data Exploration section), hence we can conclude that the model generalizes well to unseen data. Also, the model is fairly robust because although the data has ouliers and missing values, the model is still able to produce a descent auroc score.

Justification

The final Light GBM model achieves an auroc score of 0.66027 which is much better than the benchmark score (0.60643) set for this problem. Since Light GBM follows leaf-wise split to form complex trees and uses histogram based algorithm i.e it buckets continuous feature values into discrete bins, it is able to acheive both high accuracy and fast training speed (see Table 1). Thus, Light GBM seems to be a better choice among the available boosted tree algorithms. And from Table 2 we can see that by allowing the tree to grow deeper with descent number of leaves and learning rate, it is able to achieve a higher auroc score.

Lastly, the final model may not be 100% accurate, but it is nearly 66% accurate, which can helpful when deciding whether the user will listen to a song more than once in a month.

V Conclusion

Free-Form Visualization

Figure 5 represents a plot between the rolling mean of the songs released in the year 2017 and indices of the training and testing data. The curves for both train and test data show an increasing trend indicating the occurence of songs, released in year 2017, near the end of the indices for both train and test data. So, the songs released in the year 2017 are stacked near the end of both the training and testing data. Therefore, we can conclude that the data is chronologically sorted. This characteristic of data is really helpful in producing a good validation dataset for the model.

Since, the data is chronologically ordered therefore, we cannot shuffle the data while creating a validation dataset as shuffling destroys the order in the data. Hence, we select the last 25% rows of the training dataset as our validation dataset.

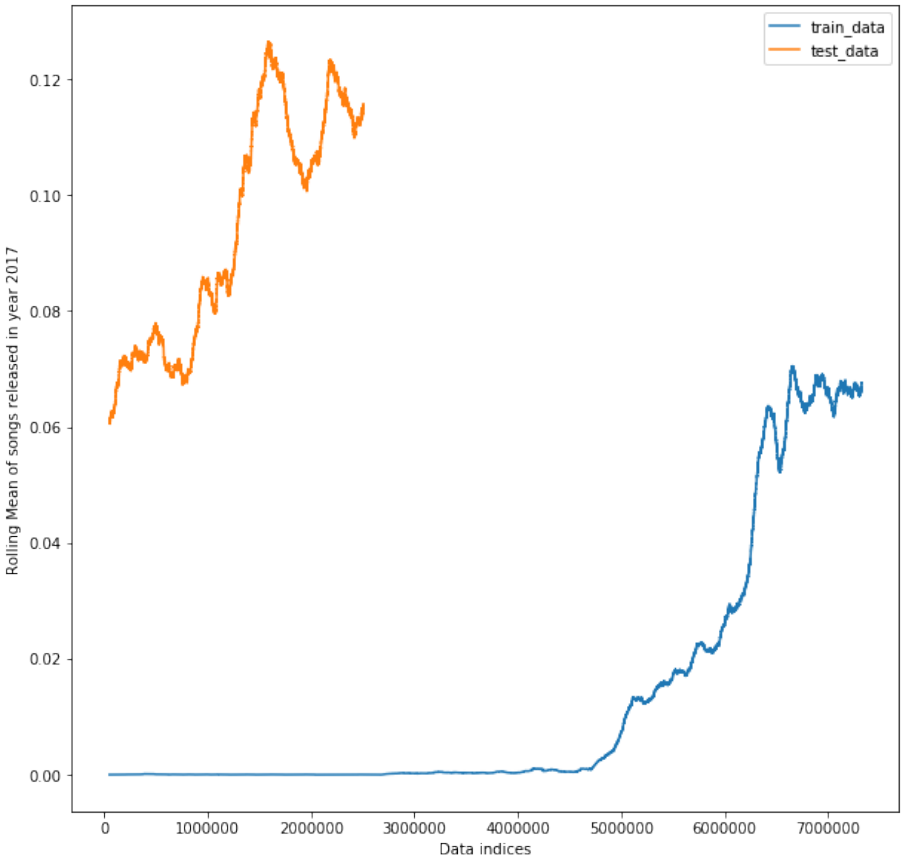


Figure 5: Tracing chronological order in data.

Reflection

The process used for this project is summerized in the following steps:

- The problem's background and domain is investigated and a suitable metric for the problem is selected.
- The available datasets are downloaded and analysed.
- Important characteristics and features within the data are highlighted.
- Possible models/ algorithms to solve the problem are proposed and a benchmark model/ result is set.
- Next the data is pre-processed and fed as inputs to the set of possible models.
- All the possible models are trained and the best performing model among them is selected.

- Parameters of the selected model are further tuned (many times) till it produces better results.

The presence so many different types of features with a lot of missing values made feature analysis and selection part hard for me. Another problem I faced was coming up with a good validation dataset, because of the presence of chronological order in the data. Lastly, tuning Light GBM parameters required a lot of effort and patience. And although my final model may not be the best, but it does produce a descent solution and performs upto my expectations.

Improvement

There are many possible improvements that can help improve the current solution. One of the improvements would be that instead of replacing all null values, present in different features, with the same value, we can replace null values in numerical features with their mean and in categorical features with their mode. Another way to improve the accuracy would be to produce deeper trees with a slower learning rate.

Since chronologically order in the data was discovered at later stages of implementation, hence is not fully exploited in the current solution. So, solving the problem as a time-series problem might result in better solutions. Lastly, since neural networks use back-propogation to learn embeddings in the data, hence I think deep neural networks might help in getting better results.

References

- https://www.kaggle.com/c/kkbox-music-recommendation-challenge
- https://github.com/Microsoft/LightGBM/blob/master/examples/python-guide/plot_example.py
- https://machinelearningmastery.com/visualize-gradient-boosting-decision-trees-xgboost-python/
- http://xgboost.readthedocs.io/en/latest/model.html
- https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/
- https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- https://www.kaggle.com/kamilkk/i-have-to-say-this
- https://www.dailyfx.com/forex/education/trading_tips/daily_trading_lesson/2012/09/18/How_to_Read_a_Moving_Average.html