Music Recommendation with LightGBM

Zhang Tianyu
20665571

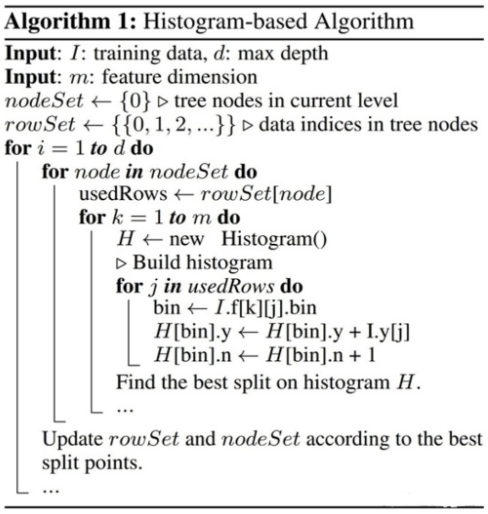
https://github.com/Jason-Zhang97/Music-Recommendation-with-LGBM

In the project, in order to make music recommendation decision, I use LightGBM algorithm to build the model and investigate some important features that have great influence on the consumers' relisten behavior. In the report, I'll first give a brief introduction of the theory and algorithm of LightGBM, then introduce the dataset I use to build, train and test the model. After that, I will show the implementation and result of LGBM. And final the feature importance and implication will be given.

1.1 Theory and algorithm

LightGBM, proposed by Microsoft, is mainly used to solve the problems encountered by GBDT in massive data, so that it can be better and faster used in practice. From the name of LightGBM, we can see that it is a light gradient elevator (GBM), which has the characteristics of fast training speed and low memory occupation compared with XGBoost. Considering the large amount of data in this paper, we choose LGBM model instead of XGBoost. Compared with XGBoost, LGBM algorithm has three aspects improvement: Gradient-based One-Side Sampling, histogram algorithm and Exclusive Feature Bundling algorithm.

Above all, GBDT is an ensemble decision tree, which is trained by sequence. The main cost of GBDT is in the learning decision tree, and the most time-consuming step is to find the optimal split point. At present, the most popular method is to select split points by pre sorting, but this method is inefficient and memory consuming. Another popular method is based on histogram, which is realized by layering continuous features into discrete columns, greatly improving efficiency. This algorithm is improved on this basis.



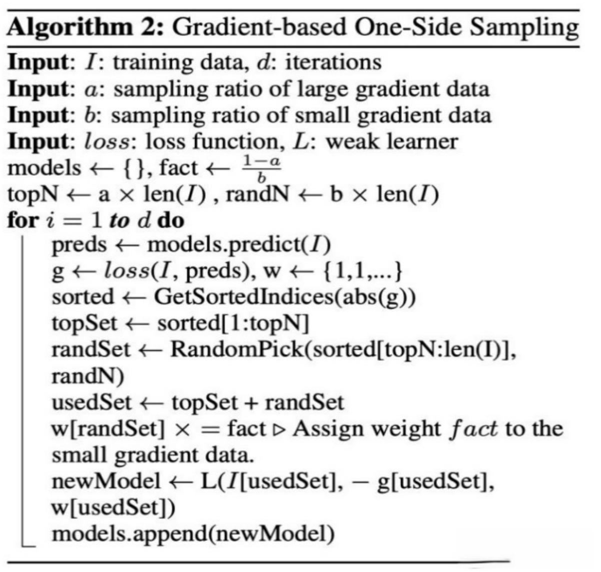
Algorithm 1: Histogram-based Algorithm

```
Input: $I$: training data, $d$: max depth
Input: $m$: feature dimension
$nodeSet \leftarrow \{0\}$ ▷ tree nodes in current level
$rowSet \leftarrow \{\{0, 1, 2, ...\}\}$ ▷ data indices in tree nodes
for $i = 1$ to $d$ do
    for $node$ in $nodeSet$ do
        usedRows $\leftarrow rowSet[node]$
        for $k = 1$ to $m$ do
            $H \leftarrow$ new Histogram()
            ▷ Build histogram
            for $j$ in $usedRows$ do
                bin $\leftarrow I$.f[k][j].bin
                $H$[bin].y $\leftarrow H$[bin].y + I.y[j]
                $H$[bin].n $\leftarrow H$[bin].n + 1
            Find the best split on histogram $H$.
            ...
    Update $rowSet$ and $nodeSet$ according to the best split points.
    ...
```

Figure 1Histogram-based Algorithm

- **Gradient-based One-Side Sampling**

The gradient size of GBDT algorithm can reflect the weight of samples. The smaller the gradient, the better the model fitting. The gradient based one side sampling algorithm (GOSS) uses this information to sample, reducing a large number of samples with small gradient. In the next calculation, only the samples with high gradient need to be paid attention to, greatly reducing the calculation amount. The specific algorithm is as follows:



```
Algorithm 2: Gradient-based One-Side Sampling
Input: $I$: training data, $d$: iterations
Input: $a$: sampling ratio of large gradient data
Input: $b$: sampling ratio of small gradient data
Input: $loss$: loss function, $L$: weak learner
models ← {}, fact ← $\frac{1-a}{b}$
topN ← a × len($I$) , randN ← b × len($I$)
for $i = 1$ to $d$ do
    preds ← models.predict($I$)
    g ← $loss$($I$, preds), w ← {1,1,...}
    sorted ← GetSortedIndices(abs(g))
    topSet ← sorted[1:topN]
    randSet ← RandomPick(sorted[topN:len(I)],
    randN)
    usedSet ← topSet + randSet
    w[randSet] × = fact ▷ Assign weight $fact$ to the
    small gradient data.
    newModel ← L($I$[usedSet], − g[usedSet],
    w[usedSet])
    models.append(newModel)
```

Figure 2 Gradient-based One-Side Sampling

We can see that GOSS sorts the samples based on the absolute value of the gradient in advance (without saving the result after sorting), and then gets the samples with a large gradient in the first a% and b% of the total samples. When calculating the gain, we can multiply (1-a)/b to enlarge the weight of samples with small gradient. On the one hand, the algorithm pays more attention to the samples with insufficient training. On the other hand, it multiplies the weight to prevent the sampling from causing too much influence on the distribution of original data.

- **Histogram algorithm**

The basic idea of histogram algorithm is to discretize continuous features into k discrete features, and construct a histogram with width of K for statistical information. Using histogram algorithm, we can find the best splitting point by traversing K bin instead of traversing data.

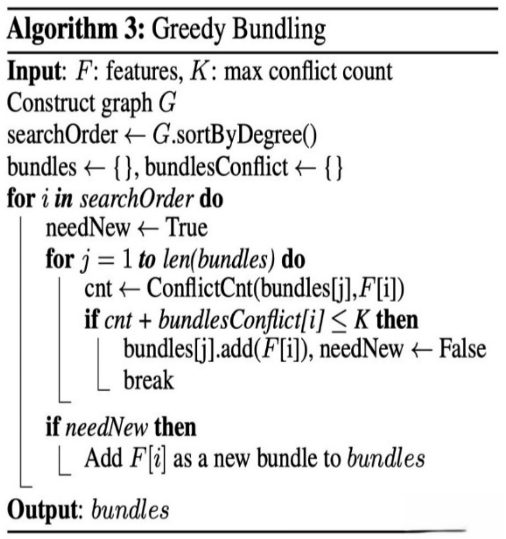
When constructing the histogram of leaf node, we can also reduce the computation by half by subtracting the histogram of parent node from that of adjacent leaf node. In the practical operation, we can calculate the leaf nodes with small histogram first, and then use the histogram as the difference to get the leaf nodes with large histogram.

- **Exclusive Feature Bundling**

High dimensional features are often sparse, and features may be mutually exclusive. For example, two features are not taken as non-zero values at the same time. If two features are not completely mutually exclusive (for example, if only a part of the case is not taken as non-zero values at the same time), the mutual exclusion rate can be used to express the degree of mutual exclusion. The exclusive feature bundling (EFB) algorithm points out that if some features are fused and bound, the number of features can be reduced. So here comes a question: Which features can be bound together?

EFB algorithm constructs a weighted undirected graph based on the relationship between features and transforms it into graph coloring algorithm. We know that graph coloring is a NP-Hard problem, so we use greedy algorithm to get the approximate solution. The algorithm steps are as follows:

If the feature dimension reaches million level, the calculation amount will be very large. In order to improve efficiency, we propose a faster solution. The strategy of sorting according to node degree is changed to sorting according to non-zero value technology, because the more non-zero value, the greater the probability of mutual exclusion.



```
Algorithm 3: Greedy Bundling
Input: $F$: features, $K$: max conflict count
Construct graph $G$
searchOrder ← $G$.sortByDegree()
bundles ← { }, bundlesConflict ← { }
for $i$ in searchOrder do
    needNew ← True
    for $j = 1$ to len(bundles) do
        cnt ← ConflictCnt(bundles[j],$F$[i])
        if $cnt + bundlesConflict[i] \leq K$ then
            bundles[j].add($F$[i]), needNew ← False
            break
    if needNew then
        Add $F[i]$ as a new bundle to $bundles$
Output: $bundles$
```

Figure 3Exclusive Feature Bundling

1.2 Data description

We want to predict the chances of a user listening to a song repetitively after the first observable listening event within a time window was triggered. If there are recurring listening event(s) triggered within a month after the user's very first observable listening event, its target is marked 1, and 0 otherwise in the training set. The same rule applies to the testing set.

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 use of public data to increase the level of accuracy of your prediction is encouraged.

The train and the test data are selected from users listening history in a given time period. Note that this time period is chosen to be before the WSDM-KKBox Churn Prediction time period. The train and test sets are split based on time, and the split of public/private are based on unique user/song pairs.

Tables' information

➢ **train.csv**

- msno: user id
- song_id: song id
- source_system_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. target=1 means there are recurring listening event(s) triggered within a month after the user's very first observable listening event, target=0 otherwise .

➢ **test.csv**

- id: row id (will be used for submission)
- msno: user id
- song_id: song id
- source_system_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.

➢ **sample_submission.csv**

sample submission file in the format that we expect you to submit

- id: same as id in test.csv
- target: this is the target variable. target=1 means there are recurring listening event(s) triggered within a month after the user's very first observable listening event, target=0 otherwise .

➢ **songs.csv**

The songs. Note that data is in unicode.

- 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**

user information.

- msno
- city
- bd: age. Note: this column has outlier values, please use your judgement.
- gender
- registered_via: registration method
- registration_init_time: format %Y%m%d
- expiration_date: format %Y%m%d

➢ **song_extra_info.csv**

- song_id
- song name - the name of the song.
- isrc - International Standard Recording Code, theoretically can be used as an identity of a song. 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.

1.3 Light GBM implementation and results

During this part, we'll introduce the establishment of Light GBM, show the algorithm result and analyze the contribution of different features. We first merge the different datasets. In order to find more useful data features in our Light GBM algorithm, we intuitively create some features, **"song_id, source_system_tab, source_screen_name, source_type, genre_ids, language, city, registered_via, registration_year, expiration_year, membership_days, song_year, play_count, play_count_artist, play_count_msno"**.These features we create are evolved from the existing important features which may have great contribution in Light GBM algorithm. We'll analyze the feature contribution and see whether the created features have valuable influence in Light GBM.

For timely purpose of our algorithm, we set the number of fold in the Light GBM algorithm to be 3. Besides, there are many different parameters in the Light GBM and some of them are very important to the performance of algorithm. We have tried many values in these parameters. In the final version, we set the number of leaves (num_leaves) to be 2^8 because the large num_leaves can make the algorithm search is more refined. The learning rate we set equals to 0.2 because Light GBM is a fast algorithm and we don't need a very high learning rate. We set the number of rounds to be 80. And the rest of parameters, most of them we set them around their default parameter values.

We use the selected features and parameters to run the Light GBM algorithm prediction section and calculate the "trainings rmse", "valid_1s rmse" and "val rmse score" which are the performance evaluation standard in this competition. The following chart is the result of "val rmse score" in all the 3 folds. It shows that the rmse score is quite small and also shows the great efficient of our Light GBM algorithm.

Fold No.	1	2	3	mean
val rmse score	0.4559	0.4662	0.4820	0.4680

Form 1: "val rmse score" over 3 folds

As we have created many new features in the dataset, we need to evaluate the feature importance and analyze these features. We draw the following plot to show all feature importance over 3 folds in average.

From the figure below, it is obvious that the time-length of membership is of vital importance to the relisten event and it is identical to the intuition that music fans have a habit of listening to music so relisten is more likely for them to happen. The second important feature is the song_id, which means the songs themselves can influence its probability of being relistened, which is also identical to the assumption. In addition to these two features, play_count_msno, genre_ids, song_year, source_screen_name, city and source_type also have certain contribution in our Light GBM algorithm but not very significant. So it seems like the members information and the song_id are valuable in Light GBM algorithm prediction and for the further research, we may focus on these aspect and dig up more information to do the recommendation decision.

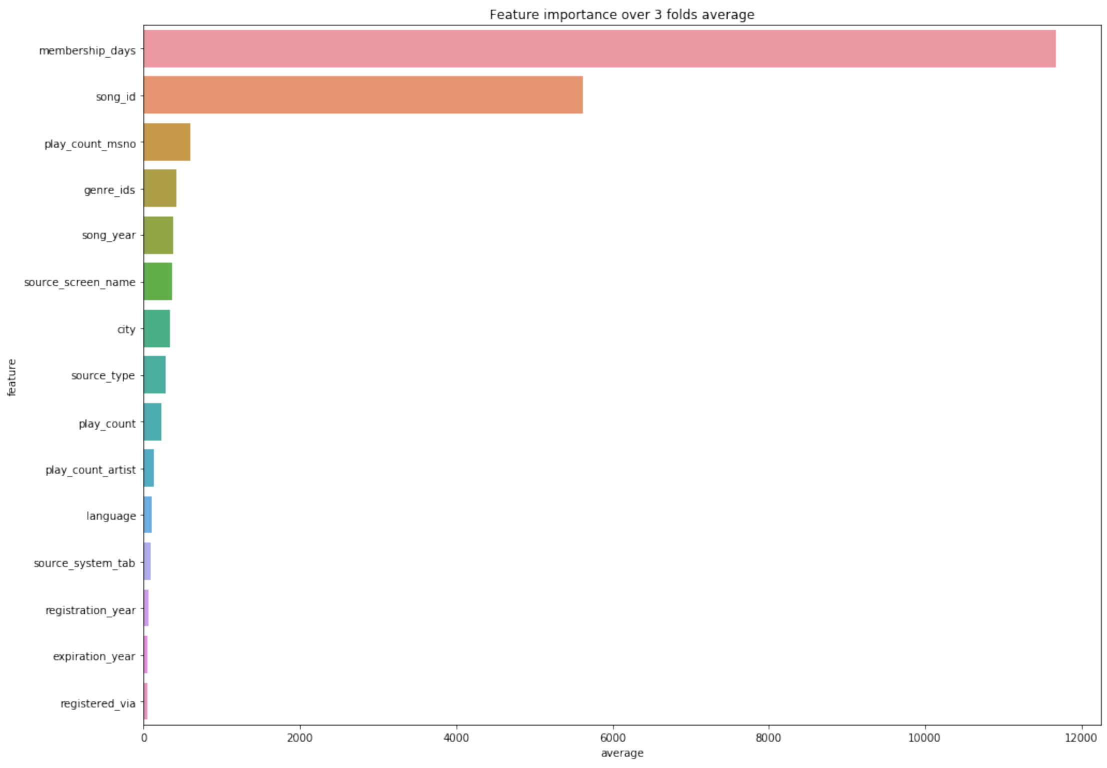


Figure 4 All feature importance over 3 folds average