Applying LightGBM to Netflix 2007 KDD Cup

Evan Bosia **Ledia Gebremedhin** **Rahaf Alharbey** **Bobby McDonough**

Abstract

This paper presents our attempt of using gradient boosting on the 2007 Netflix challenge dataset. The data sampling and model training techniques are included. The attempt was less successful than hoped, but demonstrates important lessons of machine learning.

1 Introduction

The task employs the Netflix Prize training data set. This data set consists of more than 100 million ratings from over 480 thousand randomly-chosen, anonymous customers on nearly 18 thousand movie titles. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received by Netflix during this period. The ratings are on a scale from 1 to 5 (integral) stars [1].

1.1 Challenge Background: KDD Challenge

Who Rated What in 2006: Our team's task is to predict which users rated which movies in 2006. We were provided a list of 100,000 (user_id, movie_id) pairs where the users and movies are drawn from the Netflix Prize training data set. None of the pairs were rated in the training set. The task was to predict the probability that each pair was rated in 2006 (i.e., the probability that user_id rated movie_id in 2006). The actual rating and date is irrelevant; just whether the movie was rated by that user sometime in 2006. This is a classification task with the labels "rated": 1 and "not-rated": 0 [1].

Winners were determined, by computing the root mean squared error (RMSE) between individual predictions and the correct answers. That is, if your prediction for an item is $Y^{(i)}$, the correct answer for the item is $Z^{(i)}$ and we have N items:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Z^{(i)} - Y^{(i)})^2} \tag{1}$$

1.2 Related Works

The winning team of the "Who Rated What" challenge was from the Hungarian Academy of Science. The team participated in both the "Who Rated What" and "How Many Rating" challenges by implementing the root mean square error method. In their prediction for efficiency, they combined the predicted number of ratings for each movie based on time series prediction,the predicted number of ratings by each user by using the fact that ratings were sampled proportional to the margin, the low rank approximation of the 0–1 matrix of known user–movie pairs with rating, the movie–movie similarity matrix, and association rules obtained by frequent sequence mining of user ratings considered as ordered item-sets. Their final prediction reached RMSE 0.256, gaining 0.007 over the first runner up and 0.023 over all zeroes prediction. In this process they made a simple estimation based on user-movie independence assumption that uses time series analysis and prediction and implementation of an SVD and item-item similarity using root mean squared error. Using the simplest approach method, they made some assumptions to predict a constant everywhere. In their assumption, a 7804

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

rating in the 100,000 sample would give them RMSE of 0.268 and that might reach 5 to 6th place in the cup and RMSE of "0.279 of trivial all zeroes prediction would also reach 10-13th place in the cup" [4].

The second place team of "Who Rated What" was Advanced analysis team of Neo Metrics. They presented two selections for their prediction: the selection of the exploratory variables for model construction and second includes selection of the prediction model in order to make the prediction. Primarily, the dataset was cleaned to remove the disparities including duplicate records and time dependencies, an algorithm for data sampling was proposed and used in the paper. In their research, logistic regression was used to model the data hence the input data features were transformed using decision trees and were named as intelligent variables. The variables consist of movie variables, user variables, and movie-user. The logistic regression model was selected with a RMSE of 0.263 which is superior to the decision trees and random forest which were used as a comparison [6].

1.3 Gradient Boosting

We chose to use gradient boosting decision trees as our model for the challenge. In general, the family of boosting methods is a method of training that combines a group of weak learners into a strong model. The main idea of boosting is to add new models to the ensemble sequentially. Gradient boosting refers to the method of training the weak learners. Every iteration, a new weak learner is trained off of the error of the previous learner. This is a method of gradient descent, hence the name "Gradient Boosting", where the new learners are moving the overall loss function down the gradient [2].

Decision trees are the type of model being used in gradient boosting. This type of model can be used as a classifier which works for our application. The basic idea of a decision tree is the model can be viewed as a flow chart of conditionals. Following the flow chart of conditionals down the tree, using the input data instance, will reach a prediction [7].

2 Data Methods

SAS 9.4 Software was used for creation of the analytic dataset on both local computer and SCC.

All movies and users were segmented by year to make sure that if the year of the rating has any influence on the training dataset given the test set is in a future year. This also confirms that no user is added for not rating a movie that was not on Netflix in the timeframe they were active. The user/movie combos were then separated and randomly sorted to create new user/movie pairs. These pairs were then checked against the training dataset to see that they truly never happened. If they had in that year or some other year the random data was dropped. This process was completed twice to ensure adequate non-values for the logistic model.

The new data points were simulated to be proportional to the number of events that happened on a yearly basis by both user and movie. Through trimming of values that occurred naturally this proportionality did change slightly. One extreme way this could happen is if a user rated a large portion of the 18k movies the likelihood of simulating a value that never happened randomly is less likely than a user who rated a few movies. This was rare and less likely to happen in reverse (even commonly rated movies had a pool of 480k users to pick for negative values). This proportionality assumption may have needed to be reversed for proper simulation of our negative results that will be addressed in our discussion section.

2.1 Feature Engineering

Based on User ID:

- Number of Ratings for user: sum of movies rated by user; users that rate more movies are more likely to have rated the test movie
- Number of Ratings for user in 2005: users that rate more movies in the year before our test information are more likely to have rated the test movie
- Average Rating from User: average rating of all movies for user, whether users who are more or less favorable in their reviews are likely to rate a test movie

- User Ratings per Day: sum of ratings/total time user has been rating movies in training set, users who rate movies more frequently are more likely to rate a test movie
- User Entry Year: minimum year of rating, users who entered the program more recently may have a higher likelihood of rating a movie in the test set.

Based on Movie ID:

- Number of Ratings for movie (sum of users who rated movie; movies that are rated more often are more likely to have rated by the test user)
- Number of Ratings for user in 2005 (sum of users who rated movie in 2005; movies that are rated more often in the year prior to our test are more likely to have rated by the test user)
- Netflix release year (first year the movie was rated in the training set, newer movies to the service may be more likely to be rated in 2006)
- Movie Ratings per Day (sum of users who rated the movie divided by the range of days the movie has been reviewed, movies that are rated more frequently are more likely to be rated in our test set)
- Release Year (check to see if when the movie was in theaters has any effect on user rating the movie)
- Average Rating for Movie (Average rating of the movie across all users, checking if higher rated movies are more likely to be rated in the future)

3 Model Methods

3.1 Model Selection

We chose to use gradient boosting a gradient boosting model for predictions. Gradient boosting fits the challenge well:

- gradient boosting does well with large datasets
- gradient boosting does poor with extrapolation

Because the problem is a binary classification problem, there is no risk of data extrapolation in the predictions, because labels can only be 0 or 1. Therefore, gradient boosting should be able to take advantage of of the strength of large dataset training without the downside of extrapolation.

We chose to use the LightGBM framework to build the gradient boosting model for the challenge. LightGBM is a fast gradient boosting framework developed by Microsoft which supports different training algorithms and many parameters for tuning [3].

3.2 Experiments

The parameters for the model were chosen after researching the effects of different parameters on the model. The parameters were chosen mostly for accuracy, as there was enough training data to avoid overfitting. These parameters are available in table 1.

The first 50M samples were used for training. Of these samples, 25% were separated to use as a validation set. The model was trained for 100, 250, 500, 1000, and 10000 iterations. The results of the 1000 and 10000 iteration test are in table 2.

The model size after each training iteration increases because every iteration adds a new weak-learner to the model as part of gradient boosting. The implication of this is the longer the model is trained for, the larger (and slower) it gets. There are diminishing returns for training the model too long, as the accuracy improvement becomes insignificant compared to the size of the model. For the results section 4, we chose the 1000 iteration model. The 10000 iteration model did not improve enough on the results to be worthwhile analyzing, as it is prohibitively large.

Parameter	Value	Reason
objective	binary	This is a binary classification problem
boosting	gbdt	Stable gradient boosting algorithm
metric	binary_logloss	This is a binary classification problem
max_depth	12	Deeper models are more accurate
max_leaves	2047	Set to a 2^{n-1} value. More leaves = slower but more accurate
learning_rate	0.1	Faster convergence
num_threads	8	Allow LightGBM to parallelize training on 8 CPU cores
min_data_per_leaf	200	Higher fights overfitting
bagging_fraction	0.7	Fraction of samples the model is trained on converged speed
bagging_frequency	5	Number of iterations to change bags of samples

Table 1: list of hyperparameters used

Training Iterations	Model Size (MB)	Recall(+)	RMSE
1000	94	0.2	0.42
10000	856	0.23	0.45

Table 2: Comparison of 1000 and 10000 iterations

4 Results

We chose the 1000 iteration model to analyze. It did not perform very well on the answer dataset. The distribution of the answer dataset is 7804 positives and 92196 negatives. This means a model that only selects 0 classification has an accuracy of 0.922. The accuracy of the gradient boost model is only 0.81. In the spirit of the competition, selecting negative for every data-point does not really solve the problem "Who rated what?", but it is an interesting result nonetheless.

Actual \Prediction	0	1
0	80274	11922
1	6215	1589

Table 3: Confusion Matrix

The confusion matrix paints the story of what is happening. The quality of prediction is not strong, especially positive predictions. Most positive predictions produced by this model are actually wrong! The fact that 0 was called well could just be an artifact of the distribution.

label	precision	recall	f1-score	support
0	0.93	0.87	0.90	92196
1	0.12	0.20	0.15	7804

Table 4: Model Performance Metrics

The notable values from the classification report are the low values for positives in precision, recall, and f1 score. This means for the task of determining "Who rated what?", the model performs poorly.

5 Discussion

The predictions produced by our model are not particularly good; the question remains: *how can the predictions be improved?* In this section we identify a few possible areas of improvement:

1. Feature Engineering
2. Synthetic Data Generation
3. Hyperparameter Tuning

5.1 Feature Engineering

The improvements between using 5M, 10M, 25M, and 50M training instances seemed pretty insignificant, at least insignificant to the point where training on the full dataset would not improve the results to the winning paper levels. That leaves adding features to improve the quality of each data instance. The data as run had fourteen usable features. Figure 1 shows that some features have very little importance to the model, including *Year*, *User_Entry_Year*, and *Netflix_Release_Year*.

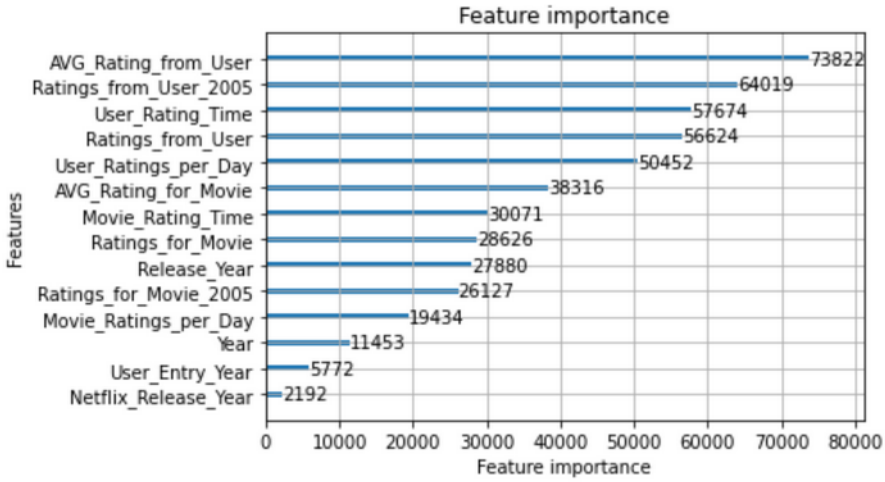

Figure 1: Feature Importance

Some of these features are heavily correlated, meaning the value the model can get from the correlated features is less. Figure 2 shows the correlation matrix between the different movie features. Correlation values can be between -1 and 1, where the magnitude indicates the strength of the correlation.

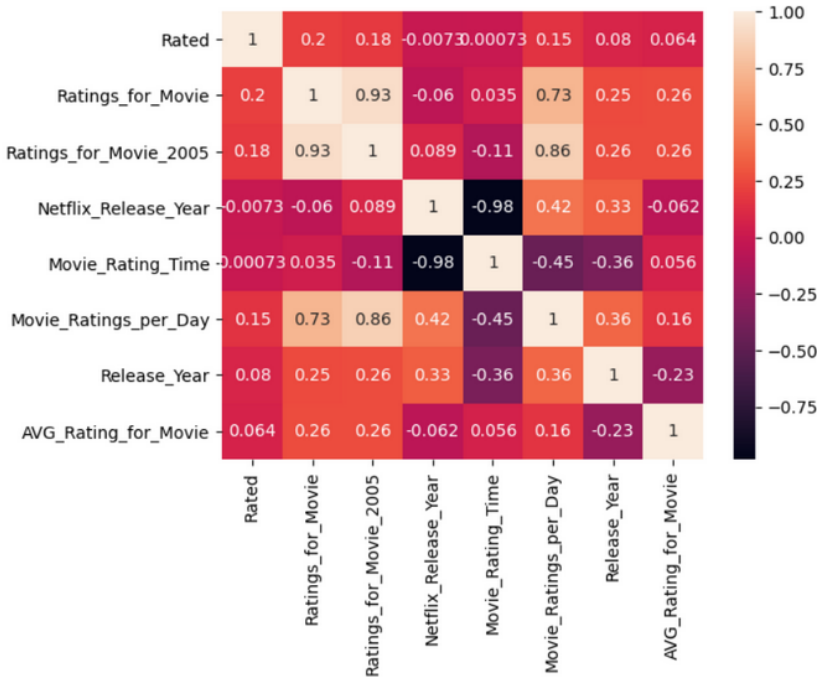

Figure 2: Movie Features Correlation

The quality of the features is likely a major factor in the under-performing of the model. Some of the features are inconsequential to the model. Some of the features are highly correlated and therefore contain roughly the same information as other columns. There is probably not enough information for the weak-learners in the gradient boost model to learn off of.

Using external data could be used to improve the quality of at least the movie features. There are datasets available online for movies such as IMDB. This would add features such as genre, director, actors, IMDB score, etc, which could make the model more effective through improving the dimensionality of the dataset. Aside from that, there are other features that could be extracted from the existing dataset, such as "time since last rating" for both movies and users.

5.2 Synthetic Data Generation

As mentioned in Section 2, the negative samples for the dataset were generated synthetically through a random shuffling of user-movie pairs. This would result in close-to-uniform distribution of user-movie pairs. The sampling method did not allow for duplicate user-movie pairs with positive ratings; assuming a user has been chosen, the number of ratings the user has made is far less than the number of movies available, so there is little difference from having all of the movies available.

This close-to-uniform distribution might not be the correct distribution of negatives that matches the real case. Some of the user-movie pairs generated may have high probability being rated, assuming a strong model prediction. Just because the user did not rate the movie does not mean that there was a higher probability of the user rating the movie versus other movies available. The uniform distribution of negatives would therefore be more akin to "noise" between the two classes. This is not good for training a model!

The solution could be to make an assumption on the distribution of negatives. The dataset given only includes user, movie, rating, and date rated. An assumption to make is that users or movies with less ratings are more likely to be negatives. The idea is that movies (or users) with fewer ratings are more likely to receive "not rated events", so should be more prevalent in the negative data distribution. There would hopefully be more separation between the rated and not-rated classes in this case.

5.3 Hyperparameter Tuning

LightGBM exposes many parameters to fine-tune the quality of the model. Using websites like *https://sites.google.com/view/lauraepp/parameters*[5] helped choose parameters, but at the end of the day this was a completely new framework/model for us. There is a decent chance the parameters chosen for the model are not conducive of getting a good result. It is not clear how much changing the parameters would have improved the model, but on the assumption that we did not choose the optimal ones, there is possible improvement.

6 Conclusion

Our results were not nearly as good as the winning papers from 2009. The key takeaway from this is that model selection is not the most important factor in developing a strong machine learning classifier. The most important factor is the quality of the data used. Feature engineering and data generation with the correct distribution could have made a big difference in the quality of predictions. This is especially notable when the winning papers have half the root mean square error as our model. Despite the poor results, it was a good dive into large data machine learning and clearly demonstrated the importance of quality data.

References

[1] Kdd cup and workshop 2007, 2007. URL `https://www.cs.uic.edu/~liub/Netflix-KDD-Cup-2007.html#download`.

[2] Alois Knoll Alexey Natekin. Gradient boosting machines, a tutorial. Technical report, fortiss GmbH and Department of Informatics, Technical University, 2007.

[3] Thomas Finley Taifeng Wang Wei Chen Weidong Ma Qiwei Ye Tie-Yan Liu Guolin Ke, Qi Meng. Lightgbm: A highly efficient gradient boosting decision tree. Technical report, Microsoft Research, Peking University, Microsoft Redmond, 2017.

[4] Miklós Kurucz, András A. Benczúr, Tamás Kiss, István Nagy, Adrienn Szabó, and Balázs Torma. Who rated what: a combination of svd, correlation and frequent sequence mining. Technical report, Data Mining and Web search Research Group, Informatics Laboratory Computer and Automation Research Institute of the Hungarian Academy of Sciences, 2007.

[5] Laurie. Laurie++: xgboost / lightgbm. URL `https://sites.google.com/view/lauraepp/parameters`.

[6] Jorge Sueiras, Alfonso Salafranca, and Jose Luis Florez. A classical predictive modeling approach for task "who rated what?" of the kdd cup 2007. Technical report, Neo Metrics, 2007.

[7] Jerry Zhu. Machine learning: Decision trees. URL `http://pages.cs.wisc.edu/~jerryzhu/cs540/handouts/dt.pdf`.