Predict IMDb Rating of a Movie

Lei Jiang

19 May 2020

Introduction

IMDb (Internet Movie Database) [1] is an online database of information related to films, televisions, videos, video games, and streaming content online – including information about cast, production crew and personal biographies, plot summaries, trivia, ratings, and fan and critical reviews.

IMDb score is determined by the formula below [2]:

$$W = \frac{Rv + Cm}{v + m}$$

where:

W = weighted rating

R = average for the movie as a number from 0 to 10 (mean) = (Rating)

v = number of votes for the movie = (votes)

m = minimum votes required to be listed in the Top 250 (currently 25,000)

C = the mean vote across the whole report (currently 7.0)

In this project, our goal is to build machine learning models to predict the IMDb score based on a variety of attributes of movies, compare the model performances and pick the best model for deployment.

Dataset

Data source is the movie_metadata.csv file from the following link:

https://github.com/sundeepblue/movie_rating_prediction/

The raw data have 5,043 data points and 28 columns excluding id column with 16 numerical variables and 12 object variables. See Table 1 below. The target variable we want to predict is *imdb_score*.

Table 1. Raw data variables information.

```
RangeIndex: 5043 entries, 0 to 5042
Data columns (total 29 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   id                         5043 non-null   int64
 1   color                      5024 non-null   object
 2   director_name              4939 non-null   object
 3   num_critic_for_reviews     4993 non-null   float64
 4   duration                   5028 non-null   float64
 5   director_facebook_likes    4939 non-null   float64
 6   actor_3_facebook_likes     5020 non-null   float64
 7   actor_2_name               5030 non-null   object
 8   actor_1_facebook_likes     5036 non-null   float64
 9   gross                      4159 non-null   float64
 10  genres                     5043 non-null   object
 11  actor_1_name               5036 non-null   object
 12  movie_title                5043 non-null   object
 13  num_voted_users            5043 non-null   int64
 14  cast_total_facebook_likes  5043 non-null   int64
 15  actor_3_name               5020 non-null   object
 16  facenumber_in_poster       5030 non-null   float64
 17  plot_keywords              4890 non-null   object
 18  movie_imdb_link            5043 non-null   object
 19  num_user_for_reviews       5022 non-null   float64
 20  language                   5031 non-null   object
 21  country                    5038 non-null   object
 22  content_rating             4740 non-null   object
 23  budget                     4551 non-null   float64
 24  title_year                 4935 non-null   float64
 25  actor_2_facebook_likes     5030 non-null   float64
 26  imdb_score                 5043 non-null   float64
 27  aspect_ratio               4714 non-null   float64
 28  movie_facebook_likes       5043 non-null   int64
```

Data Preparation and Preprocessing

When check the missing data in the raw data as shown in Table 2, we found *budget* and *gross* variables are heavily missing with 9.7% and 17.5% missing data, respectively. Other than these two, all other variables miss only below 6.5% of data. From intuition, *budget* and *gross* variables seem to be important for our prediction and hard to impute, I decide to drop these instances missing values. After drop NA, we still have 3,756 instances which is sufficient for modeling.

Table 2. Check missing data.

	n Missing	% Missing
country	5	0.100000
actor_1_facebook_likes	7	0.140000
actor_1_name	7	0.140000
language	12	0.240000
facenumber_in_poster	13	0.260000
actor_2_name	13	0.260000
actor_2_facebook_likes	13	0.260000
duration	15	0.300000
color	19	0.380000
num_user_for_reviews	21	0.420000
actor_3_name	23	0.460000
actor_3_facebook_likes	23	0.460000
num_critic_for_reviews	50	0.990000
director_facebook_likes	104	2.060000
director_name	104	2.060000
title_year	108	2.140000
plot_keywords	153	3.030000
content_rating	303	6.010000
aspect_ratio	329	6.520000
budget	492	9.760000
gross	884	17.530000

Exploratory data analysis (EDA)

First, I checked the distribution of our target variable *imdb_score* before and after data preprocessing.

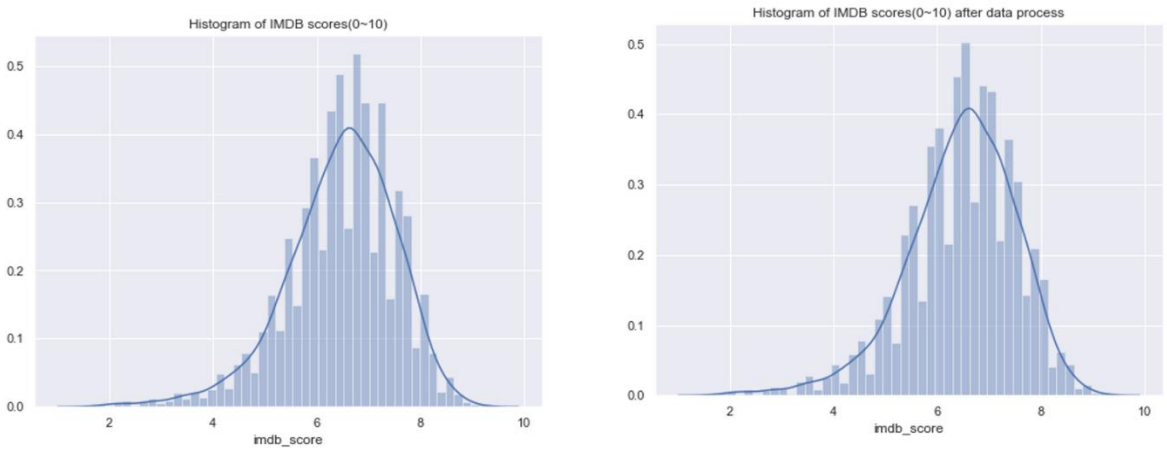

Figure 1. Distribution of target variable *imdb_score* before (left) and after(right) data preprocessing.

The plot on the left is the raw data *imdb_score* distribution while the plot on the right is the *imdb_score* distribution after data preprocessing and feature engineering. As we can see from the plots above, distribution almost maintained after the processing which indicates the train and test data can represent the original data very well.

To examine the correlations of variables to our target variable *imdb_score* a Heatmap of variables Pearson correlations was generated. See Figure 2. From the plot, we identify the most important feature is *num_voted_users* which can also be confirmed by how the score calculated in the introduction.

Form the heatmap of corr of numerical variables, we can also see *imdb_score* positively highly related to (~>0.3) *movie_fb_likes, num_user_for_reviews,num_voted_users,gross, director_fb_likes, duration,num_crtic_for reviews*. And the variables that negatively highly related to *imdb_score* are *title_year, facenumber_in_poster.*

By intuition, I expect *movie_facebook_likes* to be highly correlated to *imdb_score*. However, based on the heatmap, it is not as highly correlated as *num_voted_users* and *num_user_for_reviews*. It is about the same level of correlation as duration and num_crtic_for reviews. We can use feature importance plot from models to verify this later.

In addition, many variables are highly correlated, so decide to use tree-based methods that do require multicollinearity assumption like multiple linear regression.

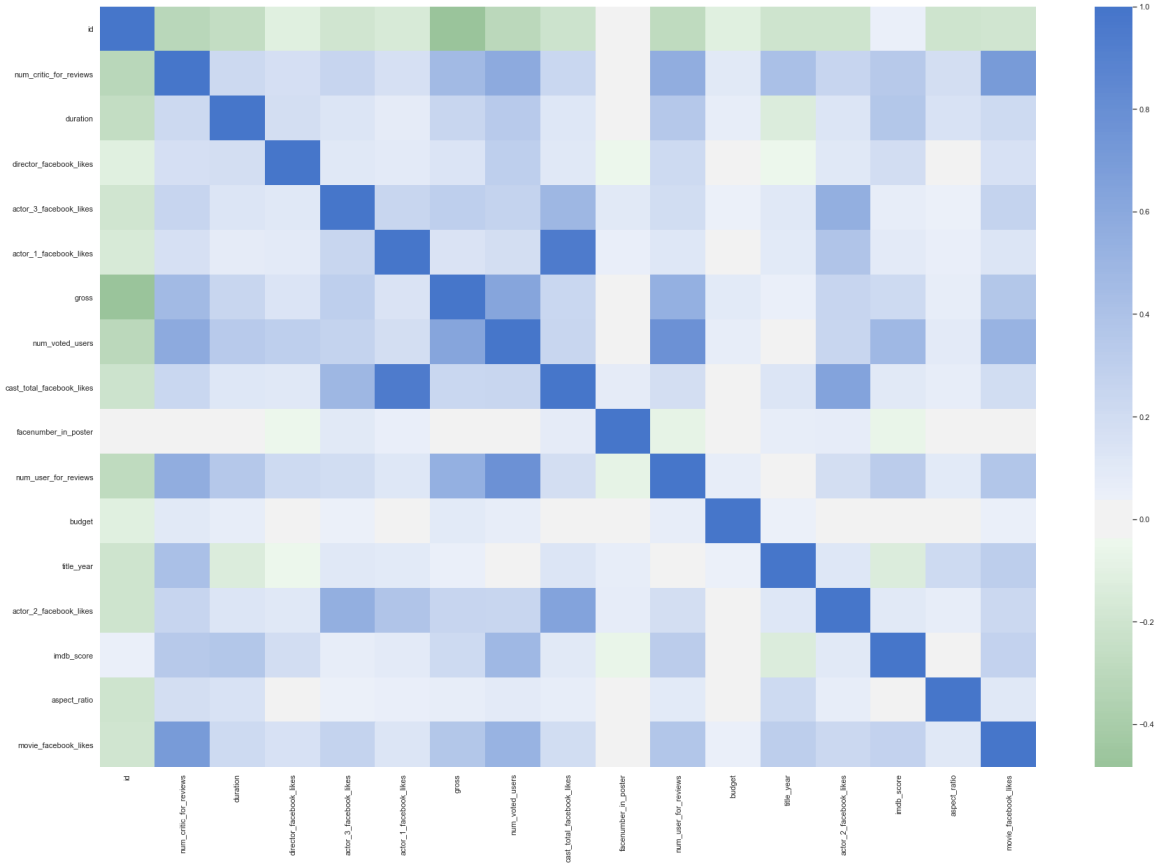

Figure 2. Heatmap of variables Pearson correlations.

Further analysis found this collection of data contains movies in 34 languages and 45 countries. Majority of the data in this collection are movies in English. The "language vs sample size" plot (Figure3) indicates data in languages other than English are very limited. Thus, the prediction for movies in other language could be less accurate. This is how sometimes the models could be biased.

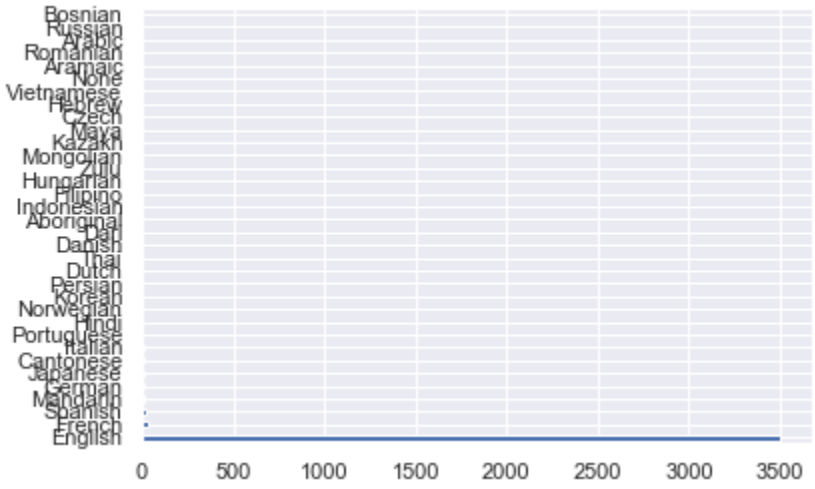

Figure 3. language vs sample size.

From the pair plot of important variables below (Figure 4A), we can see the number of instances of variables *num_user_for_reviews* and *num_voted_users* tend to be small in other languages rather than English. In addition, *num_user_for_reviews* and *num_voted_users* are highly correlated. If we build a multiple linear regression model, we need to exclude one of these two features.

movie_facebook_likes variable should be highly correlated with target variable imdb_score in my intuition. When take a closer look of the scatter plot of these two variables as shown in Figure 4B(left), the result in the plots shows the movies liked a lot tend to get higher score but they are not linearly correlated. In addition, there are many movies that were not liked a lot but get high rating.

num_voted_users is another variable that was identified as highly correlated with target variable imdb_score Figure 4B(right). The result shows highly voted movies tend to get higher score, but they are not linearly correlated.

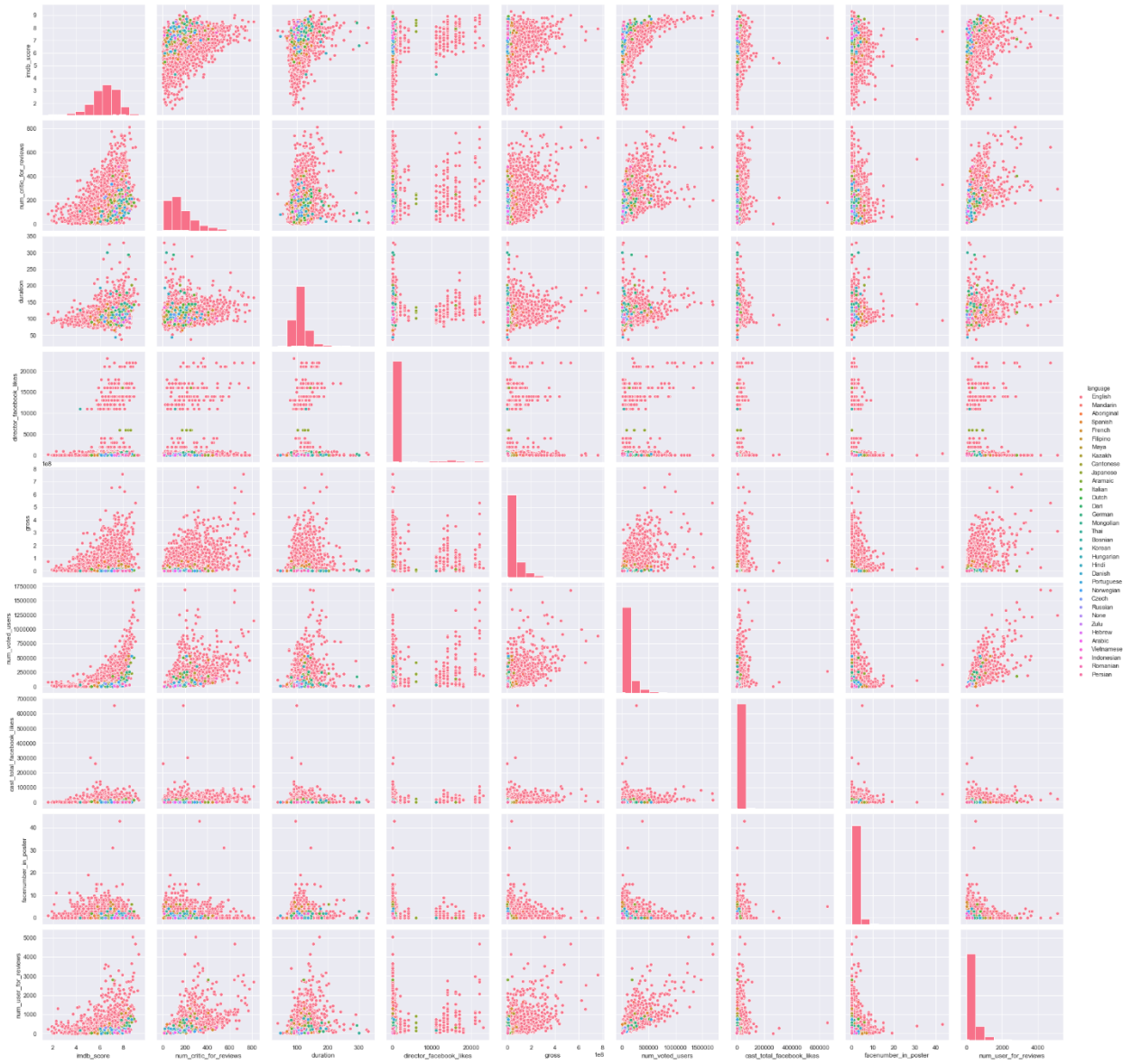

Figure 4A. Pair plot of important variables.

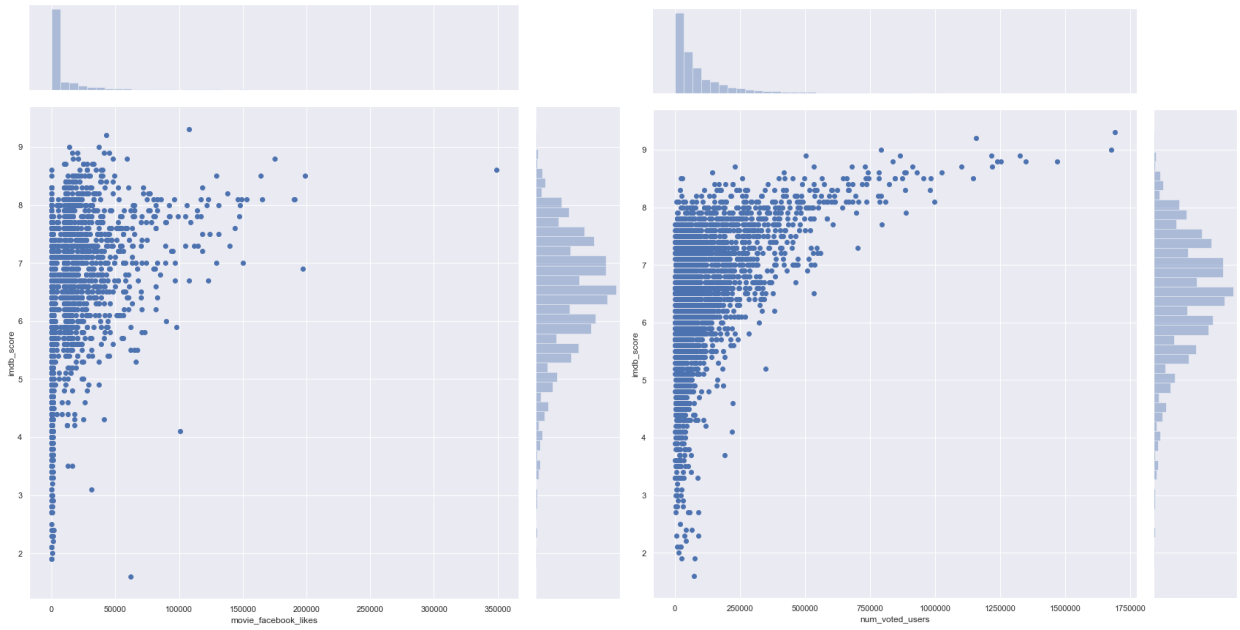

Figure 4 B. Left: *movie_facebook_likes* vs. *imdb_score*. Right: *num_voted_users* vs. *imdb_score.*

Sample Size of each genre is shown below in Figure 5A. Drama, Comedy and Thriller are the three genres with most data points. From Figure 5. B. Average Rating of each genre, we can see excluding Film-Noir genre which has only one data point, History, Biography, and War are the top 3 highest rated genres.

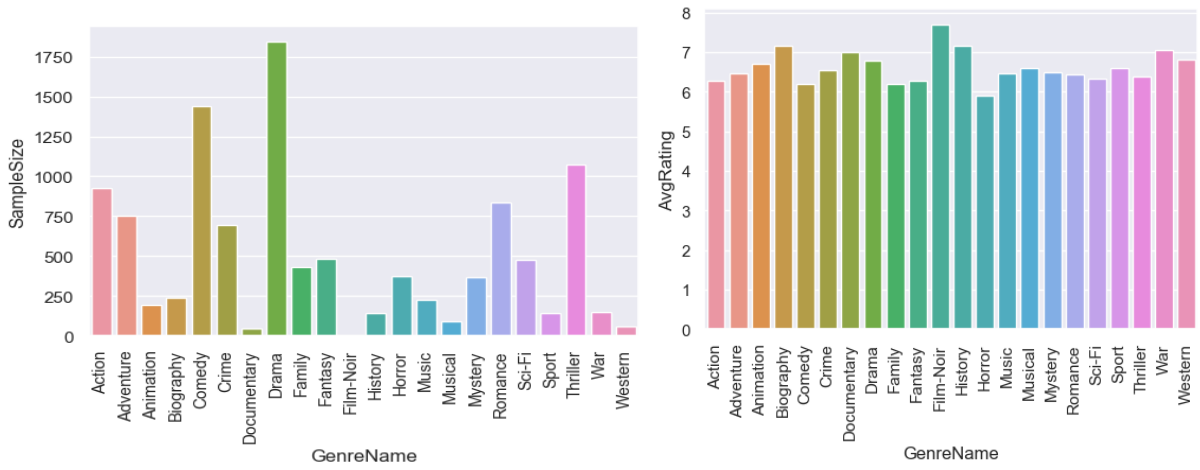

Figure 5. A. Sample Size of each genre (left). B. Average Rating of each genre (right).

Next, I analyzed the average rating from top directors by frequency who have more than 11 movies (Figure 6A) and top actor_1 who have more than 21 movies (Figure 6B) included in this dataset.

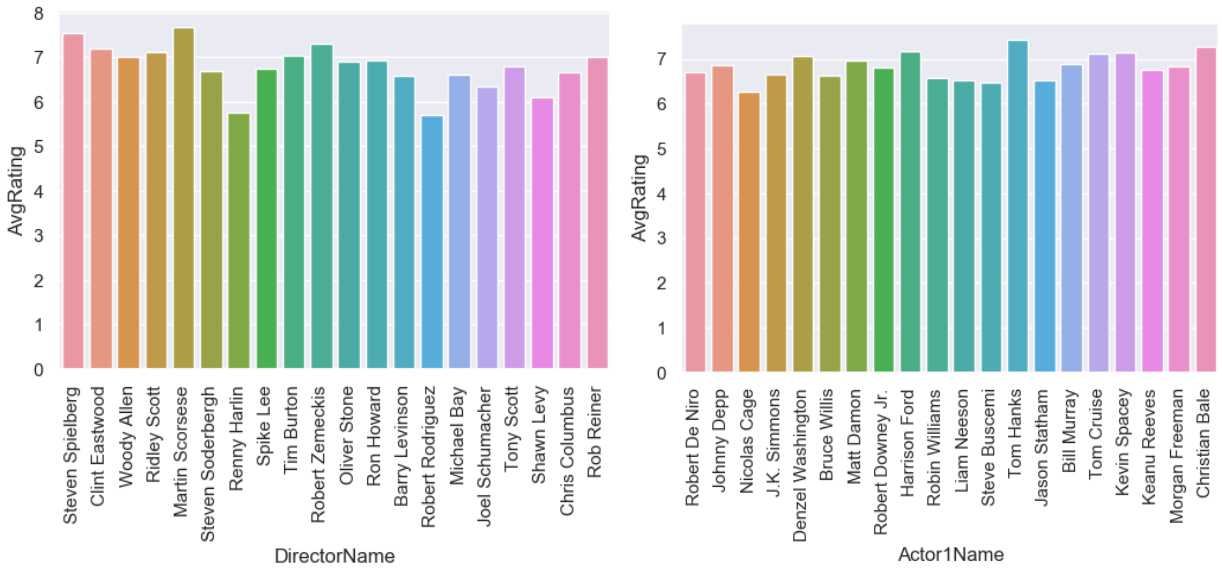

Figure 6. A. Movie Average Rating from top Directors (left). B. Movie Average Rating from top Actor1(right).

As we can see from Figure 6A, director Renny Harlin who directed 15 movies and Robert Rodriguez who directed 13 movies both have average movie rating below 6. As we can see from Figure 1., the median *imdb_score* is around 7. Thus, the average rating from some of the top directors are below average.

Then I checked the directors with best and worst average rating as shown in Table 3. We can see top frequent directors are not the top-rated directors. Moreover, top rated directors can get AvgRating score above 8.5 while worst rated directors could get AvgRating score as low as 2.1.

Table 3. Top 5 and bottom 5 avg rated directors.

	DirectorName	AvgRating
1272	Vondie Curtis-Hall	2.1
1314	Alex Zamm	2.3
742	Roger Christian	2.4
265	Jason Friedberg	2.6
973	Aaron Seltzer	2.7
...	...	...
1447	Damien Chazelle	8.5
1643	Majid Majidi	8.5
1542	Charles Chaplin	8.6
1221	Tony Kaye	8.6
1614	Akira Kurosawa	8.7

Feature Engineering

As we can see from Table 1, majority of the attributes are numerical. For these features we can change the type form float64 to float16 or int16 to save memory use. Since our data set is not particularly large and the training time is acceptable, I skipped this step.

For text column without too many levels such as *color, country, language,* and *content_rating*, I used one hot encoding.

There are several features transformed by NLP methods. Instead of simply dropping the text data columns like *genres, plot_keywords* and *movie_title*, I used on hot encoding for *genres* and NLP treatments to the other two. Since *plot_keywords* has not contextual information, I used TFIDF which is a feature based on term frequency and document frequency to vectorize the text and PCA to reduce the dimension. The train, validation and test data all went through the same processing. For *movie_title*, because the words in the title are usually contextual and can be considered as a short sentence, I used Universal Sentence Encoder (USE) which is a pretrained language model with state-of-the-art Transformer architecture to vectorize the text.

For the columns of *director_name* and actor names I used label encoding and used these features in LGB as categorical features. While in XGB model I removed these features since XGB need to take one hot encoded feature. For future predictions: unknown actors or directors in new test data just label as unknown. The code for achieve all the above treatments can be found in the feature engineering section of “Predict IMDB rating of a movie_fullFeatures-XGBbase.ipynb” on Github.

In summary, the following feature engineering were performed on text features:

-- one hot encoding features for *genre*

-- label encoding for people's name features

-- Create *plot_keywords* Features (TFIDF and then PCA)

-- *movie_title* Features (USE)

Data after preprocessing and feature engineering is saved in a csv file called “df_raw2_final.csv” for easier future retrieval.

Modeling

There are four types of models I built – XGBoost (XGB), lightgbm (LGB), multilayer perceptron (MLP), and Multiple Linear Regression (MLR). I used Root Mean Squared Error (RMSE) as the evaluation metrics. For each type of model except MLR, I built a baseline model first and then fine tune the hyper-parameters or Neural Architectures.

Train, test split

In order to split the data into Train, validation and test sets, I used np.random.choice function to randomly choose 360 data points and reserve as test. The related code is in notebook "LightGBMbase.ipynb".

To make sure the models are comparable, all models are built on the same train and test datasets.

Neural Architecture Search for MLP

Combinations of different layer number, units, learning rate and weigh decay were tested for the optimal neural architecture as shown in Table 4. For Neural Architecture Search, we chose gradient descent with momentum (tf.keras.optimizers.SGD(lr=learning_rate, momentum=0.9)) since it tends to work faster. For example, the training time for baseline model with 'adam' optimizer is 3min 25s while models with SGD optimizer and same number of layers, units and epochs only used 2min 10s. In the results table (Table 4), mse_score and rmse_score are the values of validation mse and rmse, respectively.

Table 4. Neural Architecture Search Results.

	layer	units	learning_rate	weight_decay	mse_score	rmse_score
0	8	300	0.05	1e-05	1.5552	1.24708
1	8	300	0.05	0	13.532	3.67859
2	8	300	2e-05	1e-05	1.88904	1.37442
3	8	300	2e-05	0	1.85531	1.3621
4	8	500	0.05	1e-05	137.774	11.7377
5	8	500	0.05	0	202.132	14.2173
6	8	500	2e-05	1e-05	2.004	1.41563
7	8	500	2e-05	0	2.12648	1.45825
8	9	300	0.05	1e-05	1.40082	1.18356
9	9	300	0.05	0	1.39606	1.18155
10	9	300	2e-05	1e-05	1.9101	1.38206
11	9	300	2e-05	0	1.86405	1.3653
12	9	500	0.05	1e-05	947.556	30.7824
13	9	500	0.05	0	113.737	10.6647
14	9	500	2e-05	1e-05	2.10081	1.44942
15	9	500	2e-05	0	2.06796	1.43804
16	10	300	0.05	1e-05	119.392	10.9267
17	10	300	0.05	0	12.9626	3.60036
18	10	300	2e-05	1e-05	1.96788	1.40281
19	10	300	2e-05	0	2.00948	1.41756
20	10	500	0.05	1e-05	992.923	31.5107
21	10	500	0.05	0	182.712	13.5171
22	10	500	2e-05	1e-05	2.0948	1.44734
23	10	500	2e-05	0	2.17722	1.47554
24	11	300	0.05	1e-05	1.29036	1.13594
25	11	300	0.05	0	153.763	12.4001
26	11	300	2e-05	1e-05	2.06576	1.43728
27	11	300	2e-05	0	2.05495	1.43351
28	11	500	0.05	1e-05	489.626	22.1275
29	11	500	0.05	0	154.037	12.4111
30	11	500	2e-05	1e-05	2.19935	1.48302
31	11	500	2e-05	0	2.2022	1.48398

The reason that rmse in Neural Architecture Search is not as good as baseline model is we did not exhaustedly search for the optimized learning rate for the SGD optimizer while in baseline MLP model, we used optimizer='adam' which does not require to specify a learning rate.

In addition, we need to consider the training curve in addition to final validation rmse. We need to pick a epoch number that with stable train and validation loss curve. For example, we would choose the model with Figure 7B rather than Figure 7A.

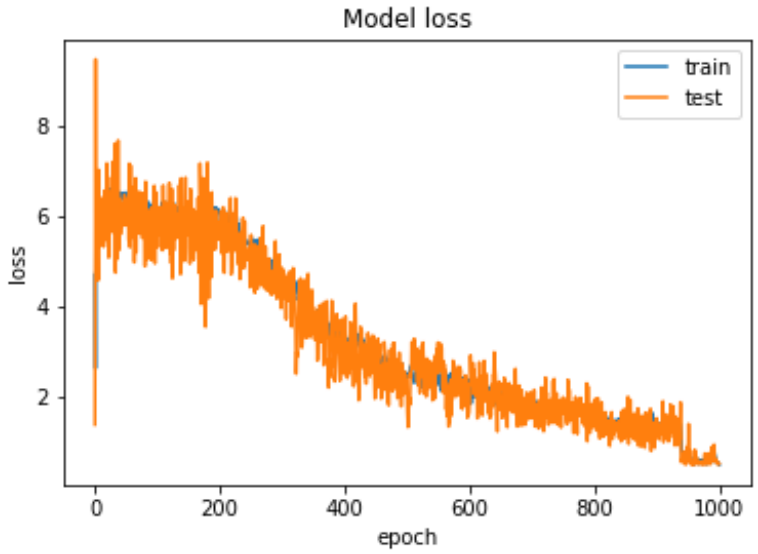

Figure7. A. Training curve for MLP DN_9_layer300_Unites_0.05_learningRate_1e-05_weightDecay_loss

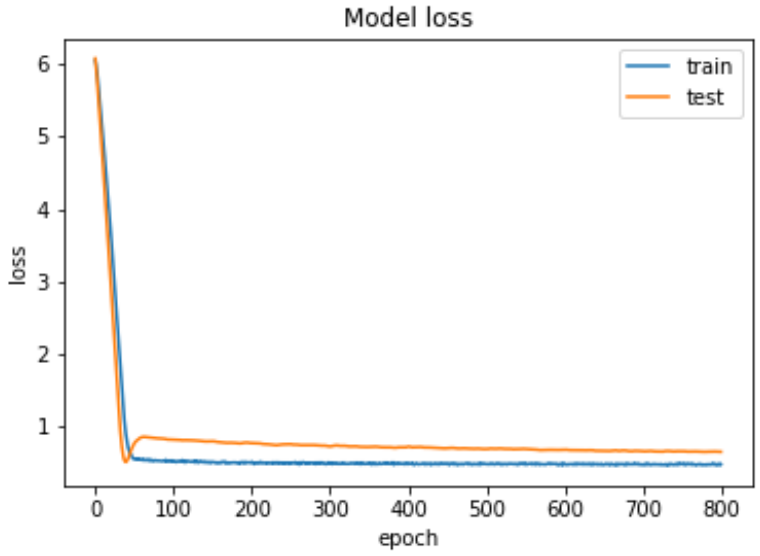

Figure 7. B. Training curve for MLP DN_9_layer300_Unites_2e-05_learningRate_0_weightDecay_loss

For MLR model, important features from XGB and LGB were used in the model. I checked assumptions for MLR and exclude highly correlated features to make sure there is little or no multicollinearity among the features.

Results

Feature Importance plots were generated from XGB, LGB and MLR models (Figure 8).

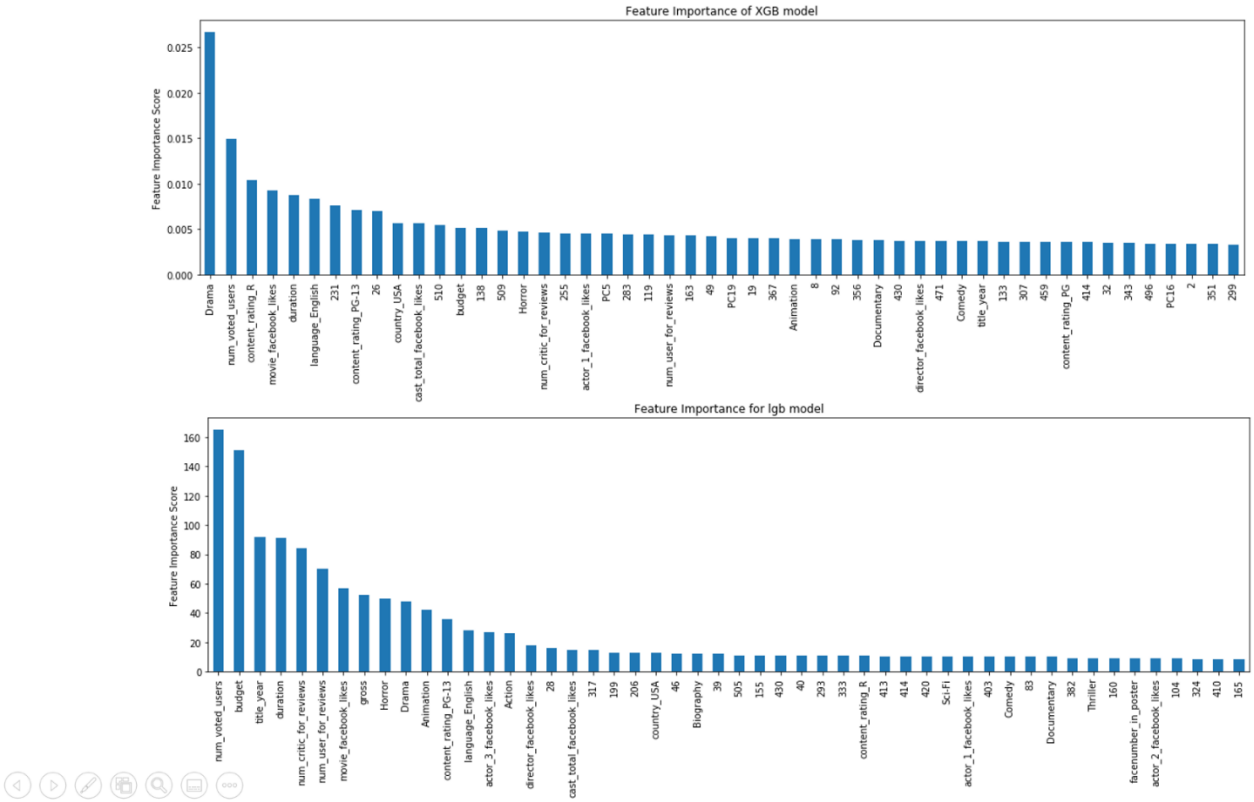

Figure 8. A. Feature Importance of XGB (upper) an LGB (lower) models.

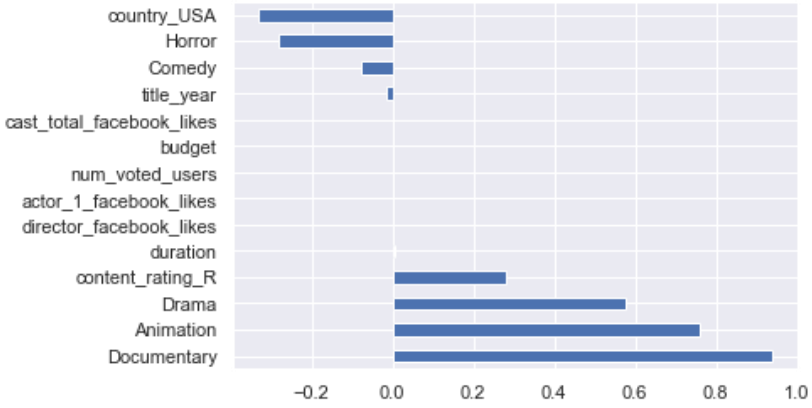

Figure 8. B. Feature Importance of MLR model.

As we can see, XGB and LGB picked many important features in common, such as *num_voted_users, budget, duration, content_rating_R, content_rating_PG-13*, and *cast_total_facebook_likes*. These important features in common were used in MLR model. However, LGB model picked *gross* feature but XGB did not.

In XGB and LGB feature importance plots, several features from *plot_keywords* TFIDF is important.

Surprisingly, in XGB model, *plot_keywords* feature PC5 is as important as *actor_1_facebook_likes*. However, NLP number features such as *plot_keywords* Features (TFIDF and then PCA) and *movie_title* Features (USE) are lack of interpretability. None of the *plot_keywords* features are picked in top 50 list for LGB model though.

Another important feature both models have in common is Drama. In the EDA section, we can see over 1750 films were labeled as Drama. That is a large portion of the entire dataset.

Figure 8B shows in MLR model, the features positively or negatively contribute to the target. A table and visualization of actual and predicted values for MLR model can be found in appendix Figure 10 and Table 6.

Model Comparison

The final Model performance were summarized below in Comparison table (Table 5) and plot (Figure 9). We can see LGB outperformed all other models with the lowest test RMSE. There was a 5.2% reduction in prediction error after hyper-parameters tuning of LGB model. XGB performance is close to LGB with 2% larger error. MLR has 18% larger error than XGB. MLP has the largest RMSE. MLP errors could be extremely too large if neural architectures and parameters are not well configured. In addition, we only have about 3000 data points for training this MLP model which consider to be a small sample size for deep learning.

Table 5. Model performance – test RMSE and Training Time

Model Name	Root Mean Squared Error	Training Time
XGB	0.7566	38.1 s
LGB	0.7354	24.3 s
MLP	1.1672	3min 25s
Multiple Linear Regression	0.8925	2.46 ms

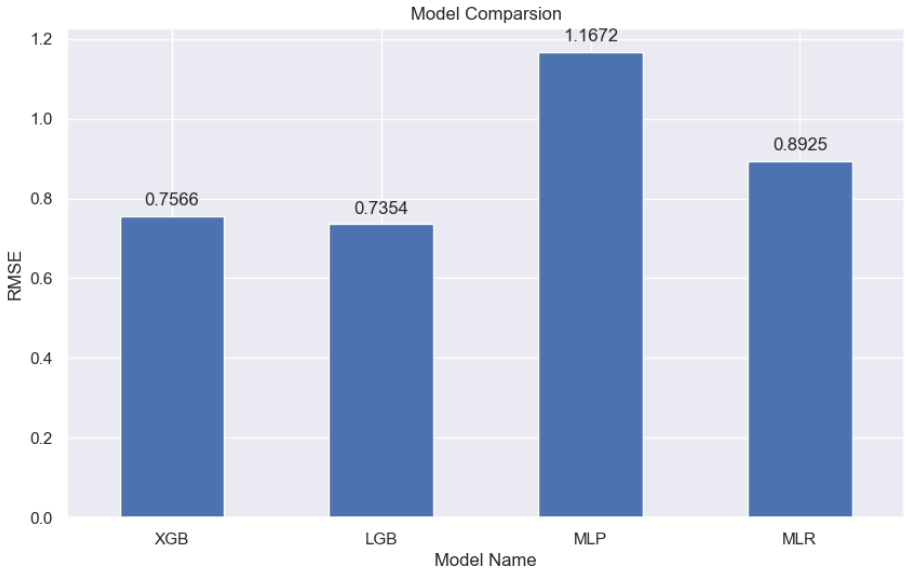

Figure 9. Model performance comparison - RMSE.

In terms of model interpretability, MLR can print out coefficients so it is of good interpretability.

XGB and LGB have better interpretability since they have tree plots (Figure 11, 12 in Appendix) and feature importance plots generated. On the other hand, MLP is a black box and lack of interpretability. Another aspect of model performance is the efficiency. MLP is slow due to not using GPU to increase training efficiency.

Conclusions

Based on the Model performance considering RMSE and Training Time, the best model I would choose for deployment is LGB which has the best test RMSE after hyper-parameters tuning as well as the second fastest training time.

The decision also depends on the requirement of deployment, if we deploy the model on the edge like wearable devices or smart phones where the computation power is limited and the prediction accuracy is not critical, we may also deploy a simpler model like MLR since it requires very little computation power and run extremely fast (2.46 ms).

Future Work

Due to a limited time budget, I did not dive deep on missing data imputation. There are ways of imputation of missing data in the effort to maintain more data for training such as median, mean, kNN, MICE, and Maximum Expectation. I would carefully test these methods of imputation and see which one works best for this dataset.

We covered two tree-based method – XGB and LGB in this project. Another type of model worth trying is Random Forest with its recursive feature elimination (RFE) function.

In addition, XGB allows visualization of feature interactions which would be interesting to check.

Codebase

https://github.com/lj89/Predict-IMDB-rating-of-a-movie

Reference

[1] https://en.wikipedia.org/wiki/IMDb

[2] https://www.quora.com/How-is-a-film-or-shows-IMDb-score-determined

Appendix

Table 6. Actual and predicted values for MLR model

	Actual	Predicted
0	7.6	7.113483
1	5.2	5.656713
2	5.7	6.160229
3	6.8	6.496999
4	8.4	8.346657
5	5.5	6.171479
6	6.7	6.657230
7	6.0	5.855501
8	7.5	7.373584
9	8.8	10.956428
10	4.4	6.111808
11	5.7	5.856075
12	6.3	6.042522
13	6.9	6.294487
14	6.9	6.531049
15	7.0	6.180618
16	8.1	7.820749
17	6.8	6.931240
18	5.9	6.176572
19	5.4	6.123233
20	7.8	7.059287
21	7.2	6.182830
22	7.0	6.421030
23	7.3	6.495419
24	6.5	5.717922

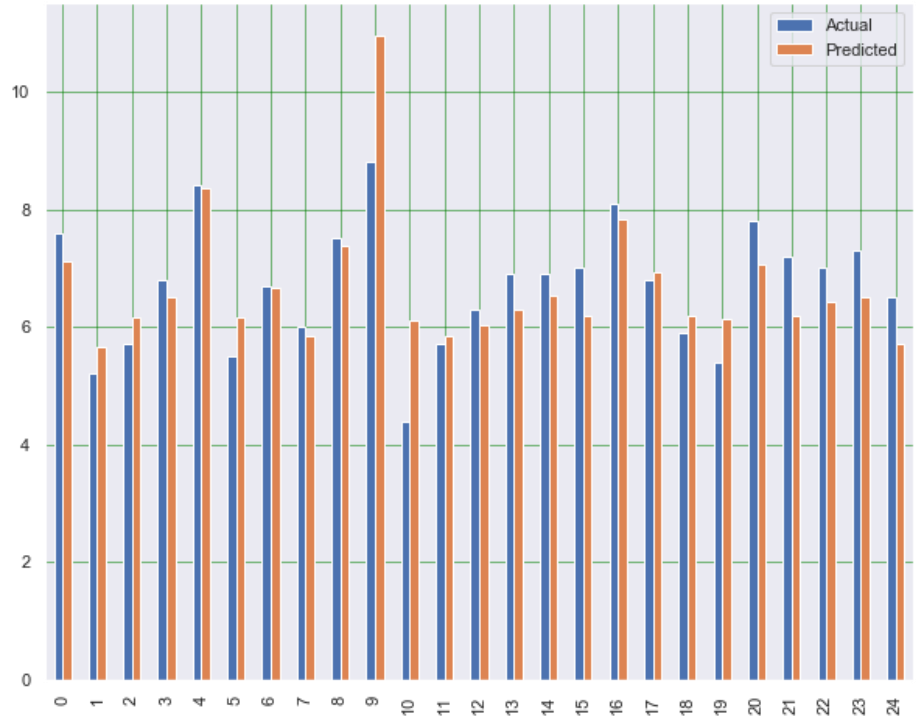

Figure 10. Actual and predicted values for MLR model

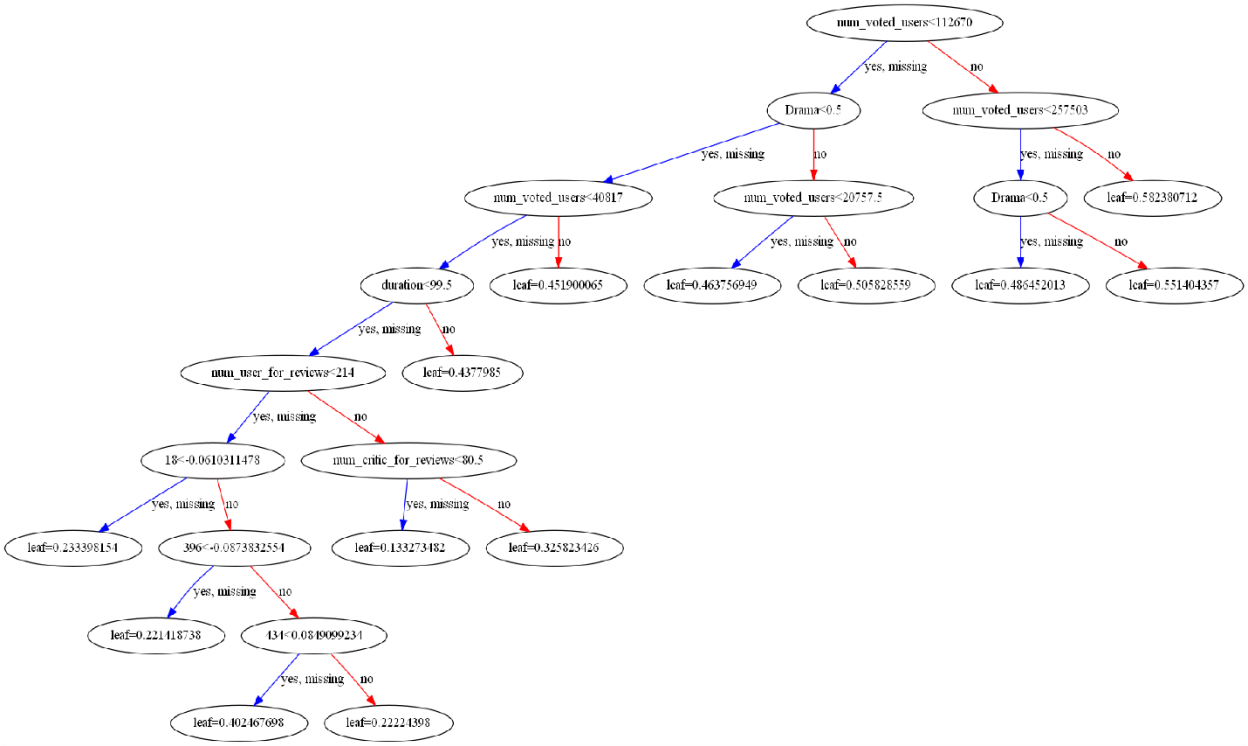

Figure 11. One of the decision tree from XGB model

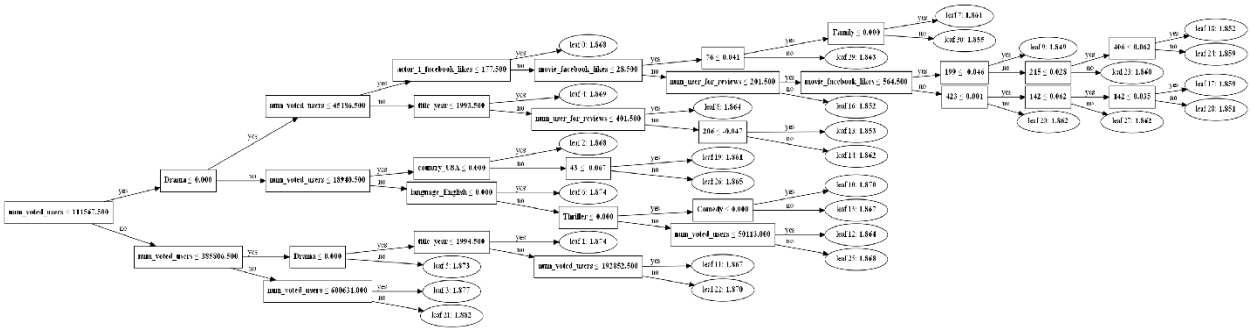

Figure 12. One of the decision tree from LGB model