Movie Ratings Project

Chin Hooi Yap

Contents

1 Overview

This MovieLens Project is part of the HarvardX Professional Certificate in Data Science program on edX. The objective is to build a movie recommendation system. I will be using the tools and methods learned throughout the course and also part of my learning from my work experience.

1.1 Introduction

Movie Recommendation system is a good practical example of supervise learning to predict what rating a user will give to a movie and subsequently recommends suitable movies to the user. As for this project, we will use the Movieslens dataset provided by Grouplens.

Firstly, we will start by creating the main data sets with code that has been provided by the staff from HarvardX. we will then perform exploratory data analysis to better understand the data. Subsequently, we will train a machine learning algorithm using the processed **edx data**, using the previously split test set to validate and optimize the model. Lastly, we will validate our final model's performance with the **validation data**.

We will output the following files in the end of this project:

1. A report in PDF format (by publishing this RMD file as pdf).
2. This report in RMD file.
3. A detailed R script that generates our predicted movie ratings.
4. Various xgb models files (trained using caret) that I used to do the prediction because it takes a very long time to retrain the models and do the prediction. They are uploaded to google drive due to the size limitation of github.
5. Some intermediate data files to speed up the compiling of this report. They are uploaded to google drive due to the size limitation of github.
6. Validation_Set_with_PredictedRatings.csv data file which contains the Validation Set together with the predicted ratings using our final model. It is also uploaded to google drive due to the size limitation of github.

1.2 Objective of project

This project aims to train a machine learning algorithm to predict what rating a user will give to a movie and subsequently recommends suitable movies to the user.

The evalutation metric that we will be using is Root Mean Square Error (RMSE). RMSE is one of the most popular evaluation metrics of accuracy for regression problems in machine learning. A lower RMSE means a lower error for the prediction results, which means the result is better. It can be calculated with the following formula. I have created a funciton specially to ease the evaluation of the models subsequently.

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

```r
RMSE<-function(model,X_test_transformed,y_test){
  test_predict <- predict(model, X_test_transformed)
  residuals<- y_test - test_predict
  RMSE<- sqrt(mean(residuals^2))
  return(RMSE)
}
```

We will experiment a few models and select the model that has the best results (i.e. lowest RMSE).

2 Obtaining the Data

We will download the edx data set and validation set using the code provided as follows. (This is a sample code provided by edx.)

```r
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

library(caret)
library(stringr)
library(tidyverse)



ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Do note that the validation set is only reserved for final validation. For model training and performance optimization, we will use the test set split from the edx data.

3 Method and Analysis

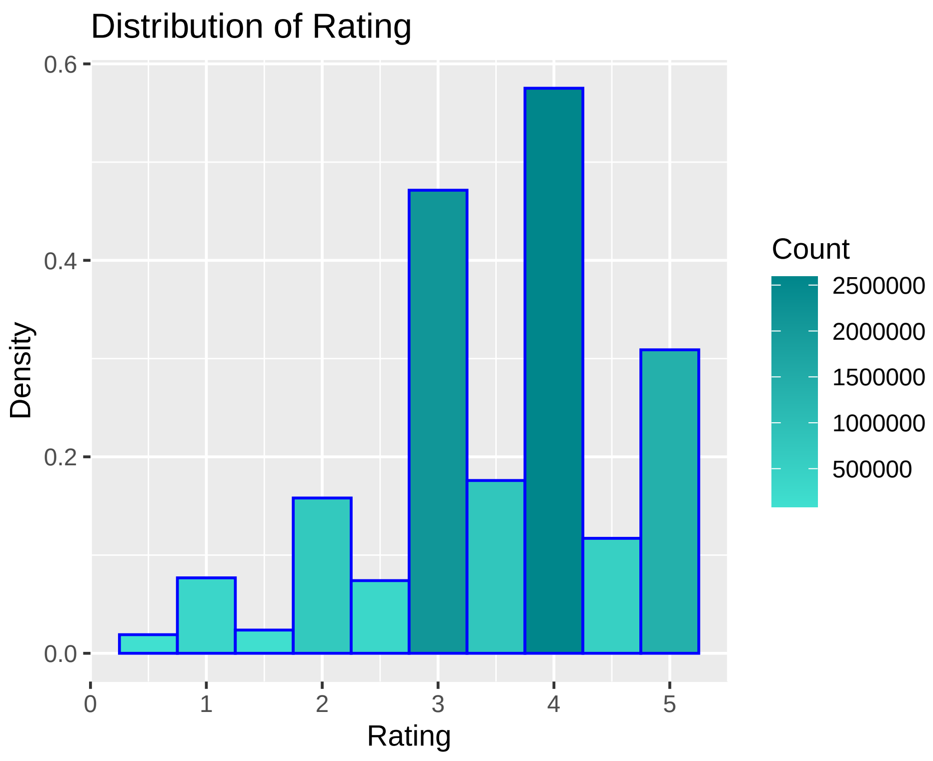
3.1 Exploratory Data Analysis

We will start by answering the following questions using exploratory data analysis.

1) What is the rating that were given most frequently? We can study this by looking at the histogram provided by this piece of code.

It can be seen that 3 and 4 has a much higher frequency as compared to the others, and not many users rate a movie with a half rating (i.e. 0.5, 1.5 and etc).

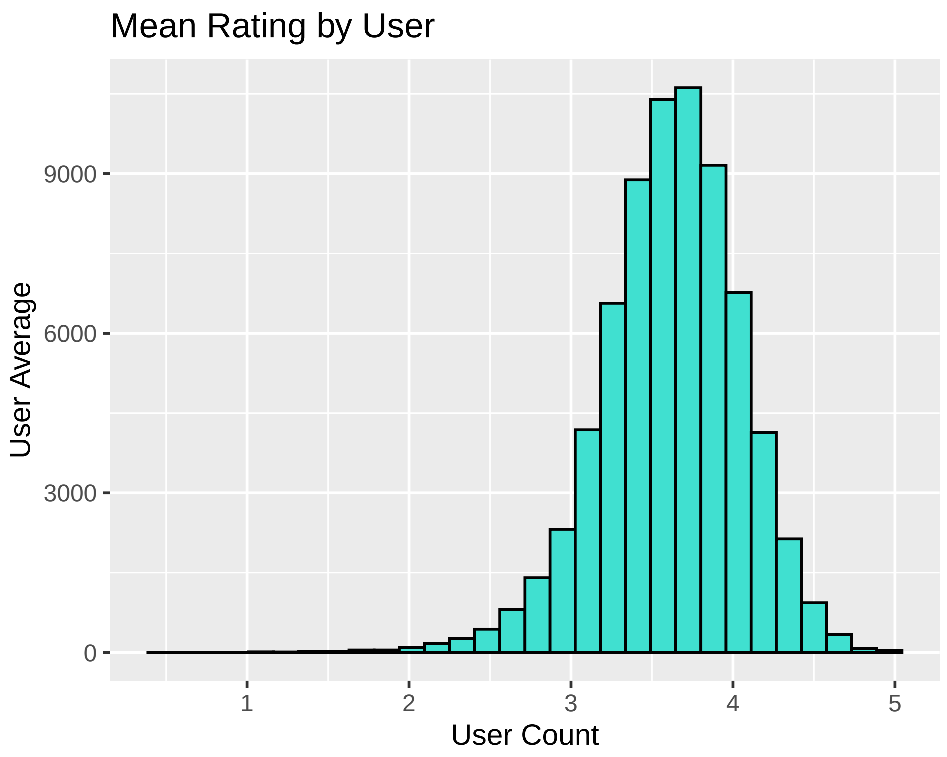
```
edx%>%ggplot(aes(x=rating))+
  geom_histogram(binwidth=0.5,colour="blue",
                 aes(y=..density..,fill=..count..))+
  scale_fill_gradient("Count",low="turquoise",high="turquoise4")+
  xlab("Rating")+
  ylab("Density")+
  ggtitle("Distribution of Rating")
```



2) What is distribution of the rating by user? We can use the following piece of code to produce a histogram of rating by user mean rating.

We can see that many users have a mean rating of 3.6-3.7, quite close to the mean of edx which is 3.51. It is also obvious that some users are harsher than the others, with average rating below 2.5 whereas some have a mean rating of 4.5 and above. Hence, we should include user bias into our model.

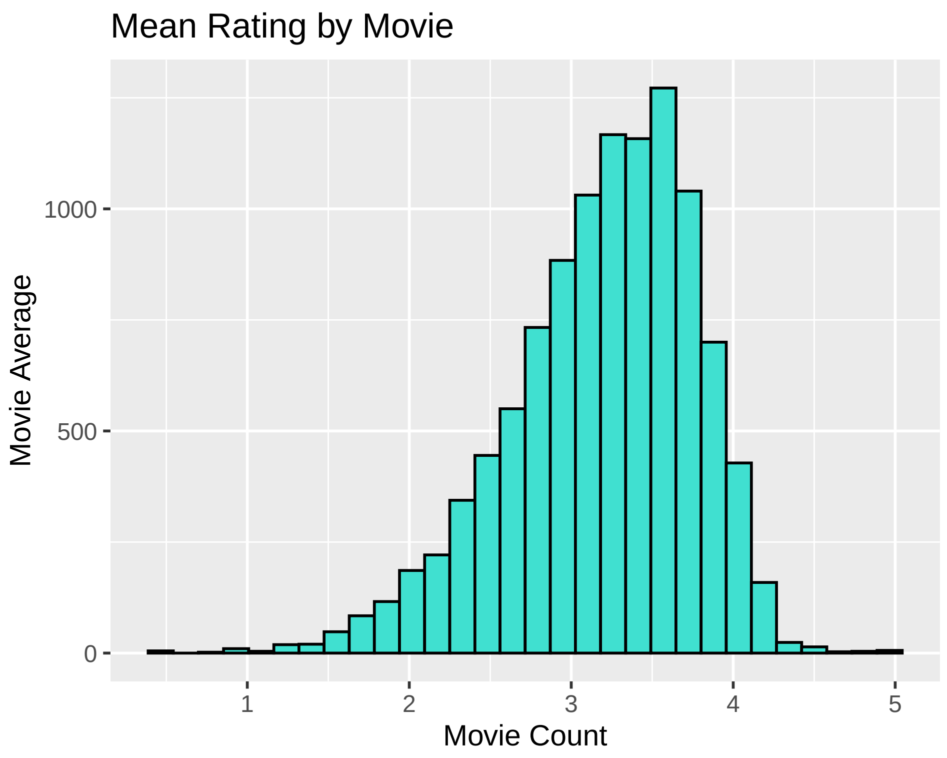
```
#Mean Rating given by the users
edx %>% group_by(userId)%>%
  summarise(UserAverage=mean(rating)) %>%
  ggplot(aes(UserAverage)) +
  geom_histogram(bins=30, color = "black",fill = "turquoise")+
  xlab("User Count")+
  ylab("User Average")+
  ggtitle("Mean Rating by User")
```



3) What is distribution of the rating by movie? We can use the following piece of code to produce a histogram of rating by movie mean rating.

We can see that many movies have a mean rating of 3.5-3.7. It can be shown that some movies are more well liked than the others, have an average rating of 4.5 and above, and the opposite is true. Hence, we should include movie bias into our model.

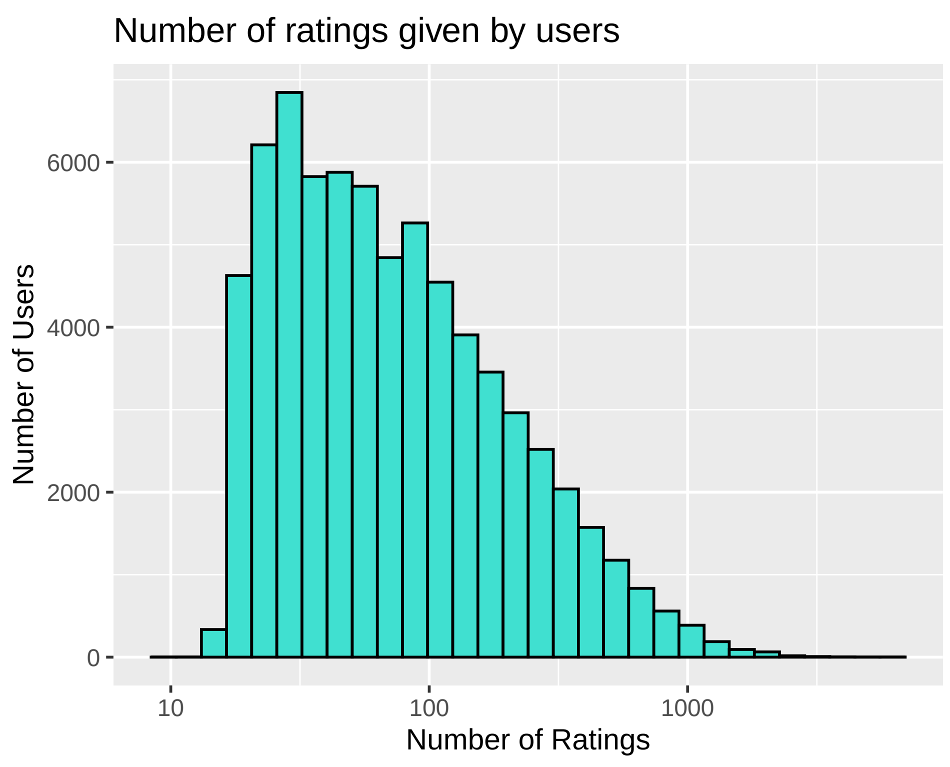
```
#Mean Rating given by the movies
edx %>% group_by(movieId)%>%
  summarise(MovieAverage=mean(rating)) %>%
  ggplot(aes(MovieAverage)) +
  geom_histogram(bins=30, color = "black",fill = "turquoise")+
  xlab("Movie Count")+
  ylab("Movie Average")+
  ggtitle("Mean Rating by Movie")
```



4) What about the distribution of the total number of rating grouped by movies? The following piece of code produce a histogram of total number of ratings grouped by movies.

It can be seen that some movies are rated very frequently while some are not so popular. This is understanable as some blockbusters are much popular than the common movies. Thus, we need to be careful and might need to remove some movies that only have 1 rating.

```
edx %>% count(userId) %>%
  ggplot(aes(n))+
  geom_histogram(bins = 30,color = "black",fill = "turquoise")+
  scale_x_log10()+
  xlab("Number of Ratings")+
  ylab("Number of Users")+
  ggtitle("Number of ratings given by users")
```



3.2 Modelling

3.2.1 Modelling techniques

To train a good model, we will need to go through the following steps:

1. Preprocessing - Preprocess the data to ensure a clean and consistent data frame are input to the model
2. Feature Engineering - Adding more useful features and do experiments to validate the performance.
3. Feature Selection - Select the top features input to the models and remove the remaining to reduce the noise to the model.
4. Hyperparameters tuning - Tune the hyperparameters of the resepctive models to ensure that our models are optimized.

XGBoost is chosen for our modeling for this project as this parellel tree boosting method is able to solve many data science problems in an efficient and accurate way. Its main advantages are the

- Execution Speed
- Model Performance

Details can be found in these links:

https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

https://xgboost.readthedocs.io/en/latest/

3.2.2 Baseline Model

Before we proceed to train our first baseline model, we would need to go through the necessary process of preprocessing and feature engineering.

As for the baseline model, we only preprocess the data by separating all the genres into different columns and giving them a 1 or 0, similar to the concept of one hot encoding for categorical variables. Do note that this process is quite computational resource intensive, you can use the uploaded preprocessed file to save time. The preprocess function can be done by calling the following function.

```r
preprocess<-function(df){
  #remove '-' from Sci-Fi, Film-Noir
  df1<-df
  df1$genres<-gsub('-','',df1$genres)
  #replace no genres listed with NA if it exists
  if(nrow(df1%>%select(genres)%>%filter(genres=="(no genres listed)"))!=0){
    df1 <- df1 %>% na_if("(no genres listed)")
    #df1<-df1 %>% replace_with_na_all(condition = ~.genres == "(no genres listed)")
  }
  #seperate genres column by "|"
  df1<-df1%>% separate(genres, c("g1","g2","g3","g4","g5","g6","g7","g8"))

  #generate unique values of genres, checked that g1 contains all unique values for genres
  unique<-df1$g1%>%unique()
  l2<-list(unique)
  l2[[1]]<-sort(l2[[1]],decreasing = FALSE)


  #generate extra columns for all unique values, 1 if genres contain it, 0 if not
  for (i in l2[[1]]){
    df1<-df1%>%mutate(!!i:=ifelse(apply(df1 == !!i, 1, any), 1, 0))
  }

  #remove intermediate columns
  df1<-df1%>%select(-c(g1,g2,g3,g4,g5,g6,g7,g8))

  return (df1)
}

edx1<-preprocess(edx)
```

Next up, we will do a very simple feature engineering to take care of any columns with NA.

```r
feature_eng_default<-function(df){
  df1<-df
  #remove NA column created during preprocessing to prepare for modeling
  if("NA"%in%colnames(df1)){
    df1<-df1%>%select(-"NA")
  }
  #feature engineering
  #impute NA with 0 for processed genres
  #checkNA with edx1%>%select(COLUMN_NAME)%>%filter(is.na(COLUMN_NAME))
  #only genres has NA, impute with 0
  df1[is.na(df1)]<-0
  return (df1)
}

edx1<-feature_eng_default(edx1)

edx1[,1:8]%>%head(5)%>% knitr::kable()
```

userId	movieId	rating	timestamp	title	Action	Adventure	Animation
1	122	5	838985046	Boomerang (1992)	0	0	0
1	185	5	838983525	Net, The (1995)	1	0	0
1	292	5	838983421	Outbreak (1995)	1	0	0
1	316	5	838983392	Stargate (1994)	1	1	0
1	329	5	838983392	Star Trek: Generations (1994)	1	1	0

```r
edx1[,9:18]%>%head(5)%>% knitr::kable()
```

Children	Comedy	Crime	Documentary	Drama	Fantasy	FilmNoir	Horror	IMAX	Musical
0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0

```r
edx1[,19:24]%>%head(5)%>% knitr::kable()
```

Mystery	Romance	SciFi	Thriller	War	Western
0	1	0	0	0	0
0	0	0	1	0	0
0	0	1	1	0	0
0	0	1	0	0	0
0	0	1	0	0	0

Training xgb regression model with the default hyperparameters to obtain a baseline model.

```r
tune_grid <- expand.grid(nrounds = 100,
                         max_depth = 6,
                         eta = 0.3,
                         gamma = 0,
                         colsample_bytree = 1,
                         min_child_weight = 1,
                         subsample = 1)


trctrl <- trainControl(method = "cv", number = 5)
xgb_first_100_default <- train(X_train_transformed_default,y_train, method = "xgbTree",
                         metric="RMSE",
                         trControl=trctrl,
                         tuneGrid = tune_grid)
```

We will evaluate the performance with the following piece of code.

```
RMSE_default<-RMSE(xgb_first_100_default,X_test_transformed_first,y_test_first)
#creating a RMSE_results to store all RMSE from different models to compare result
RMSE_results<-data.frame(Method = "Baseline_Model", RMSE = RMSE_default)
```

It turns out that the RMSE for baseline model is 1.0234772

Method	RMSE
Baseline_Model	1.023477

3.2.3 Improved Model 1 - with movie and user bias added as features

Next, we are adding movie and user bias, as well as the number of days since the user first rated a movie as features.

```
library(lubridate)
feature_eng_bias<-function(df){
  df1<-df
  df1<-df1%>%mutate(timestamp=as_datetime(timestamp,origin = lubridate::origin, tz = "UTC"))

  #prepare columns for new features- UserAverage, Movie Average,
  #FirstUserRatingDate,FirstMovieRatingDate
  User<-data.frame(df1%>%
                     group_by(userId)%>%summarise(FirstUserRatingDate=min(timestamp),
                                                  UserAverage=mean(rating)))

  MovieAverage<-data.frame(df1%>%
                             group_by(movieId)%>%summarise(FirstMovieRatingDate=min(timestamp),
                                                           MovieAverage=mean(rating)))
  #join to main data frame and produce the intended features,
  #BaselineRating,UserBias,MovieBias,daySinceFirstUserRating,daySinceFirstMovieRating
  df1<-df1%>%mutate()%>%
    inner_join(User,by="userId")%>%
    inner_join(MovieAverage,by="movieId")%>%
    mutate(BaselineRating=mean(rating),
           UserBias=UserAverage-BaselineRating,
           MovieBias=MovieAverage-BaselineRating,
           daySinceFirstUserRating=as.integer(difftime(timestamp,
                                                        FirstUserRatingDate,
                                                        units = "days")),
           daySinceFirstMovieRating=as.integer(difftime(timestamp,
                                                         FirstMovieRatingDate,
                                                         units = "days")))%>%
    select(-FirstUserRatingDate,-FirstMovieRatingDate)
  return (df1)
}
```

```
edx1<-feature_eng_bias(edx1)
```

Building the next model with nrounds = 200 as the default nrounds = 100 is not enough after testing with xgb.cv and calculate the RMSE for the test set.

```
RMSE_200_default<-RMSE(xgb_200_default,X_test_transformed,y_test)
RMSE_results <- bind_rows(RMSE_results,
                          data.frame(Method= "Improved_Model_1_Bias",
                                     RMSE=RMSE_200_default))
```

Hence, we can see that the RMSE for Improved Model 1 is 0.8637766. We will further improve the model by doing features selection in the next section.

Method	RMSE
Baseline_Model	1.0234772
Improved_Model_1_Bias	0.8637766

3.2.4 Feature Selection and Improved Model 2

After that, we will select the top 10 important features to reduce the noise given to the model.

```
importance <- varImp(xgb_200_default, scale=FALSE)
#select top 10 features
features_selected<- rownames(importance$importance)[1:10]
X_train_transformed_fselect<-X_train_transformed%>%select(features_selected)
X_test_transformed_fselect<-X_test_transformed%>%select(features_selected)
```

Retrain model again, this time with nrounds = 400 because

- 200 is still not hitting the early stopping criteria
- Anything above 400 is too computational expensive and will take us very long to train a model

```
tune_grid <- expand.grid(nrounds = 400,
                         max_depth = 6,
                         eta = 0.3,
                         gamma = 0,
                         colsample_bytree = 1,
                         min_child_weight = 1,
                         subsample = 1)


trctrl <- trainControl(method = "cv", number = 5)
xgb_400_default_fselect <- train(X_train_transformed_fselect,y_train, method = "xgbTree",
                                 metric="RMSE",
                                 trControl=trctrl,
                                 tuneGrid = tune_grid)
```

```
RMSE_400_default_fselect<-RMSE(xgb_400_default_fselect,
                               X_test_transformed_fselect,y_test)
```

```r
RMSE_results <- bind_rows(RMSE_results,
                          data.frame(Method= "Improved_Model_2_Feature_selection",
                                     RMSE=RMSE_400_default_fselect))
```

It can be seen that doing features selection really does help on the performance, improving the performance to 0.860211.

Method	RMSE
Baseline_Model	1.0234772
Improved_Model_1_Bias	0.8637766
Improved_Model_2_Feature_selection	0.8602110

3.2.5 Hyperparameters Tuning & Improved Model 3

Xgboost models has the following hyperparameters to tune to obtain the most optimized model for our data analysis.

- eta
- min_child_weight
- max_depth
- colsample_bytree
- subsample
- gamma

3.2.5.1 eta (learning rate) As for eta (learning rate), we tried to reduce from the default 0.3 to 0.1 but we realized that it was too computational expensive with a worse performance (based on crossed validation as shown in the code below).

```r
#Use cross validation to obtain a test performance with lower eta (0.1)
params <- list(nrounds = 400,
               max_depth = 6,
               eta = 0.1,
               gamma = 0,
               colsample_bytree = 1,
               min_child_weight = 1,
               subsample = 1)
Mat1<-data.matrix(X_train_transformed_fselect)
xgbcv <- xgb.cv( params = params, data = Mat1,label = y_train, nrounds = 400, nfold = 5,
                 showsd = T, stratified = T,
                 print_every_n = 10, early_stop_round = 5, maximize = F)
#test-rmse:0.863981+0.000433
#worse than eta=0.3, and much computational heavy for nrounds =400 and eta=0.1
#hence, eta and nrounds should be fixed at 0.3 and 400 respectively
```

3.2.5.2 max_depth and min_child_weight We will now tune max_depth and min_child_weight. These two should be tuned together and first among the others parameters to determine the maximum depth and minimum child weight of the gradient boosting decision trees, basically deciding whether to further split the nodes of the respective decision tree. The resulting number of max_depth and min_child_weight is 8 and 1 respectively.

```r
tune_grid <- expand.grid(nrounds = 400,
                         max_depth = c(4,6,8),
                         eta = 0.3,
                         gamma = 0,
                         colsample_bytree = 1,
                         min_child_weight = c(1,3,5),
                         subsample = 1)


trctrl <- trainControl(method = "cv", number = 5)
xgb_400_depth_child_fselect <- train(X_train_transformed_fselect,y_train, method = "xgbTree",
                                     metric="RMSE",
                                     trControl=trctrl,
                                     tuneGrid = tune_grid)
```

```r
RMSE_400_depth_child_fselect<-RMSE(xgb_400_depth_child_fselect,
                                   X_test_transformed_fselect,y_test)
```

Hence, we can see that the RMSE now is 0.8501908.

3.2.5.3 colsample_bytree and subsample Next, we will tune colsample_bytree and subsample, keeping the previously tuned parameter constant, and others at default value. colsample_bytree and subsample are the parameters that determine the number of columns and rows to select during resampling process for each decision trees. The resulting colsample_bytree and subsample are 1 for both. This could be due to the fact that we have done a features selection of top 10 features, hence the resampling at 100% for both columns and rows.

```r
#next, fix nrounds=400, eta=0.3, max_depth = 8,
#min_child_weight = 1 and tune colsample_bytree and subsample
tune_grid <- expand.grid(nrounds = 400,
                         max_depth = 8,
                         eta = 0.3,
                         gamma = 0,
                         colsample_bytree = c(0.8,1),
                         min_child_weight = 1,
                         subsample = c(0.6,0.8,1))


trctrl <- trainControl(method = "cv", number = 5)
xgb_400_colsample_subsample_fselect <- train(X_train_transformed_fselect,y_train,
                                             method = "xgbTree",
                                             metric="RMSE",
                                             trControl=trctrl,
                                             tuneGrid = tune_grid)
```

```r
RMSE_400_colsample_subsample_fselect<-RMSE(xgb_400_colsample_subsample_fselect,
                                           X_test_transformed_fselect,y_test)
```

Hence, we can see that the RMSE now is the same as xgb_400_depth_child_fselect as the best tuned parameters are the same, the RMSE is 0.8501908.

3.2.5.4 gamma

Last but not least, we are going to tune gamma, this is the factor that decides the regularization of model, normally used to avoid overfitting the model. The resulting gamma chosen is 0, which means no regularization is needed for our model. This is understandable as:

1. Movie bias, user bias and other bias are mostly taken care of by the modelling process.
2. We do not observe any big difference between the training and testing RMSE.

```
tune_grid <- expand.grid(nrounds = 400,
                         max_depth = 8,
                         eta = 0.3,
                         gamma = c(0,3,6),
                         colsample_bytree = 1,
                         min_child_weight = 1,
                         subsample = 1)


trctrl <- trainControl(method = "cv", number = 5)
xgb_400_gamma_fselect <- train(X_train_transformed_fselect,y_train, method = "xgbTree",
                                         metric="RMSE",
                                         trControl=trctrl,
                                         tuneGrid = tune_grid)
```

```
RMSE_400_gamma_fselect<-RMSE(xgb_400_gamma_fselect,X_test_transformed_fselect,y_test)
RMSE_results <- bind_rows(RMSE_results,
                          data.frame(Method= "Improved_Model_3_HyperParameters_tuning",
                                     RMSE=RMSE_400_gamma_fselect))
```

The RMSE now maintained at 0.8502 as the best tuning parameters are the same.

Method	RMSE
Baseline_Model	1.0234772
Improved_Model_1_Bias	0.8637766
Improved_Model_2_Feature_selection	0.8602110
Improved_Model_3_HyperParameters_tuning	0.8501908

We can also see that the RMSE has improved from the baseline model of 1.0235 to 0.8502.

3.2.5.5 Final Model Selection

We will be selecting the final model judging by the model performance. As we have gone through various cycles of improvement, the RMSE reduces from the baseline 1.0234 to 0.8502. Hence the final model we selected is Improved Model 3.

Method	RMSE
Baseline_Model	1.0234772
Improved_Model_1_Bias	0.8637766
Improved_Model_2_Feature_selection	0.8602110
Improved_Model_3_HyperParameters_tuning	0.8501908

3.3 Model Validation with Validation Set

We will now validate our final model with the validation set.

```
validation1<-preprocess(validation)
validation1<-feature_eng_default(validation1)
validation1<-feature_eng_bias(validation1)
X_test_val<-validation1%>%select(-rating,-timestamp,-title)
y_test_val<-validation1$rating

X_test_val_transformed<-transform(X_test_val)
X_test_val_transformed_fselect<-X_test_val_transformed%>%select(features_selected)

RMSE_val_400_gamma_fselect<-RMSE(xgb_400_gamma_fselect,X_test_val_transformed_fselect,y_test_val)
RMSE_comparison<-data.frame(Data_Set = "Test_Set", RMSE = RMSE_400_gamma_fselect)
RMSE_comparison <- bind_rows(RMSE_comparison,
                             data.frame(Data_Set= "Validation_Set",
                                        RMSE=RMSE_val_400_gamma_fselect))
```

Putting everything together in a same table

Data_Set	RMSE
Test_Set	0.8501908
Validation_Set	0.8446133

Do note that validation set is not used for any performance validation until this final step. It can be seen that our performance of 0.8446133 has far exceeded the 0.86490 criteria of this project to qualify for full marks.

3.4 Predicted Rating Generation

We also generate the predicted rating on Validation set with the code as follows,

```
#Generate predicted rating
Predicted_ratings_Validation <- predict(xgb_400_gamma_fselect, X_test_val_transformed_fselect)
Validation_Set_with_PredictedRatings<-cbind(validation1,Predicted_ratings_Validation)
```

4 Conclusion

We have built a series of useful machine learning algorithm to predict the movie rating. It can be seen that by adding user and movie bias, and optimizing our models with features selection and hyperparameters tuning, we can achieve a much better performance as compared to the baseline model. The final model has a RMSE of 0.8446133 which is much lower than the 0.86490 criteria of this project to qualify for full marks. Last but not least, other machine learning models could also improve the results further, but due to time and hardware constraints, we have yet to test it further.

5 Appendix - Enviroment

```r
print("Operating System:")
```

```
## [1] "Operating System:"
```

```r
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          6.3
## year           2020
## month          02
## day            29
## svn rev        77875
## language       R
## version.string R version 3.6.3 (2020-02-29)
## nickname       Holding the Windsock
```