MovieLensCapstone: Predicting Movie Recommendations through Machine Learning

Sahibzada Ali Mahmud

June 17, 2019

Executive Summary

This project is based on using an appropriate machine learning algorithm to predict movie recommendations using the 10M Version of Movie lens Dataset which can be downloaded here. The intention behind completing this Capstone Project is to test the skills acquired throughout the Professional Certification in Data Science course, and apply them in solving a real world data science proeblem. In our case, we shall analyze the available data set and use the Random Forest approach for prediction. Random forest algorithm is a supervised classification and regression algorithm in which greater is the number of trees, greater is the accuracy of the algorithm. It offers flexibilty since it can be used for classification and prediction. The git repository for this project can be accessed here.

1. Methodology

In this project, after installing the required packages and loading the libararies, the data contained in MovieLens is observed. The subset of the full MovieLens data set is used for our analysis since after repeated attempts, RStudio running on the machine used for this project could not process the large MovieLens data set. Another attempt was made on using RStudio Cloud, however, that did not work out either because of the very large processing times. Therefore, it was decided to use the subset of Movielens that was used in the Machine Learning course for the Profession Certification in Data Science. Exploratory data analysis was carried out to get insights into the data set. While using the Random Forest method for training the model and predicting the ratings, again the machine failed to execute the algorithm even in 4 hours. Therefore, the code presented in the script and this report has been validated and run in pieces. After several attempts, due to the very large processing times, the results could not be obtained. However, based on the best practices, the random forest model performance is assesed through the code given in section 9. The Root Mean Square Error(RMSE) is used in section 8 to calculate the error and hence the accuracy of the model.

2. Loading Libraries and Data

```r
#Install Required Packages if Necessary
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse

## Warning: package 'tidyverse' was built under R version 3.4.4

## -- Attaching packages --------------------------------------------- tidyverse 1.2.1 --

## v ggplot2 2.2.1     v purrr   0.2.4
## v tibble  1.4.2     v dplyr   0.7.4
## v tidyr   0.8.0     v stringr 1.3.0
## v readr   1.1.1     v forcats 0.3.0
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
## Warning: package 'tibble' was built under R version 3.4.4
```

```
## Warning: package 'tidyr' was built under R version 3.4.4
```

```
## Warning: package 'readr' was built under R version 3.4.4
```

```
## Warning: package 'purrr' was built under R version 3.4.4
```

```
## Warning: package 'stringr' was built under R version 3.4.4
```

```
## Warning: package 'forcats' was built under R version 3.4.4
```

```
## -- Conflicts ------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Warning: package 'caret' was built under R version 3.4.4
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(dslabs)) install.packages("dslabs", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: dslabs
```

```r
if(!require(randomForest)) install.packages("randomForest", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
#Load Required Libraries
library(dslabs)
library(tidyverse)
library(caret)
library(randomForest)

#Load Data
data("movielens")
```

3. Oberving Data

```
#Check the Structure of movielens
str(movielens)
```

```
## 'data.frame':    100004 obs. of  7 variables:
##  $ movieId  : int  31 1029 1061 1129 1172 1263 1287 1293 1339 1343 ...
##  $ title    : chr  "Dangerous Minds" "Dumbo" "Sleepers" "Escape from New York" ...
##  $ year     : int  1995 1941 1996 1981 1989 1978 1959 1982 1992 1991 ...
##  $ genres   : Factor w/ 901 levels "(no genres listed)",..: 762 510 899 120 762 836 81 762 844 899 .
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ rating   : num  2.5 3 3 2 4 2 2 2 3.5 2 ...
##  $ timestamp: int  1260759144 1260759179 1260759182 1260759185 1260759205 1260759151 1260759187 1260
```

```
#Check the first 10 Observations
head(movielens, n=10)
```

```
##    movieId                                   title year
## 1       31                         Dangerous Minds 1995
## 2     1029                                   Dumbo 1941
## 3     1061                                Sleepers 1996
## 4     1129                    Escape from New York 1981
## 5     1172 Cinema Paradiso (Nuovo cinema Paradiso) 1989
## 6     1263                        Deer Hunter, The 1978
## 7     1287                                 Ben-Hur 1959
## 8     1293                                  Gandhi 1982
## 9     1339         Dracula (Bram Stoker's Dracula) 1992
## 10    1343                               Cape Fear 1991
##                              genres userId rating  timestamp
## 1                             Drama      1    2.5 1260759144
## 2  Animation|Children|Drama|Musical      1    3.0 1260759179
## 3                           Thriller      1    3.0 1260759182
## 4  Action|Adventure|Sci-Fi|Thriller      1    2.0 1260759185
## 5                              Drama      1    4.0 1260759205
## 6                          Drama|War      1    2.0 1260759151
## 7             Action|Adventure|Drama      1    2.0 1260759187
## 8                              Drama      1    2.0 1260759148
## 9   Fantasy|Horror|Romance|Thriller      1    3.5 1260759125
## 10                          Thriller      1    2.0 1260759131
```

4. Exploratory Data Analysis

```r
#Check the distribution of Movie Ratings
summary(movielens$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.500   3.000   4.000   3.544   4.000   5.000
```

```r
#Check the Number of Rows and Columns in the Dataset
paste('The movielens dataset has',nrow(movielens),'rows and',ncol(movielens),'columns.')
```

```
## [1] "The movielens dataset has 100004 rows and 7 columns."
```

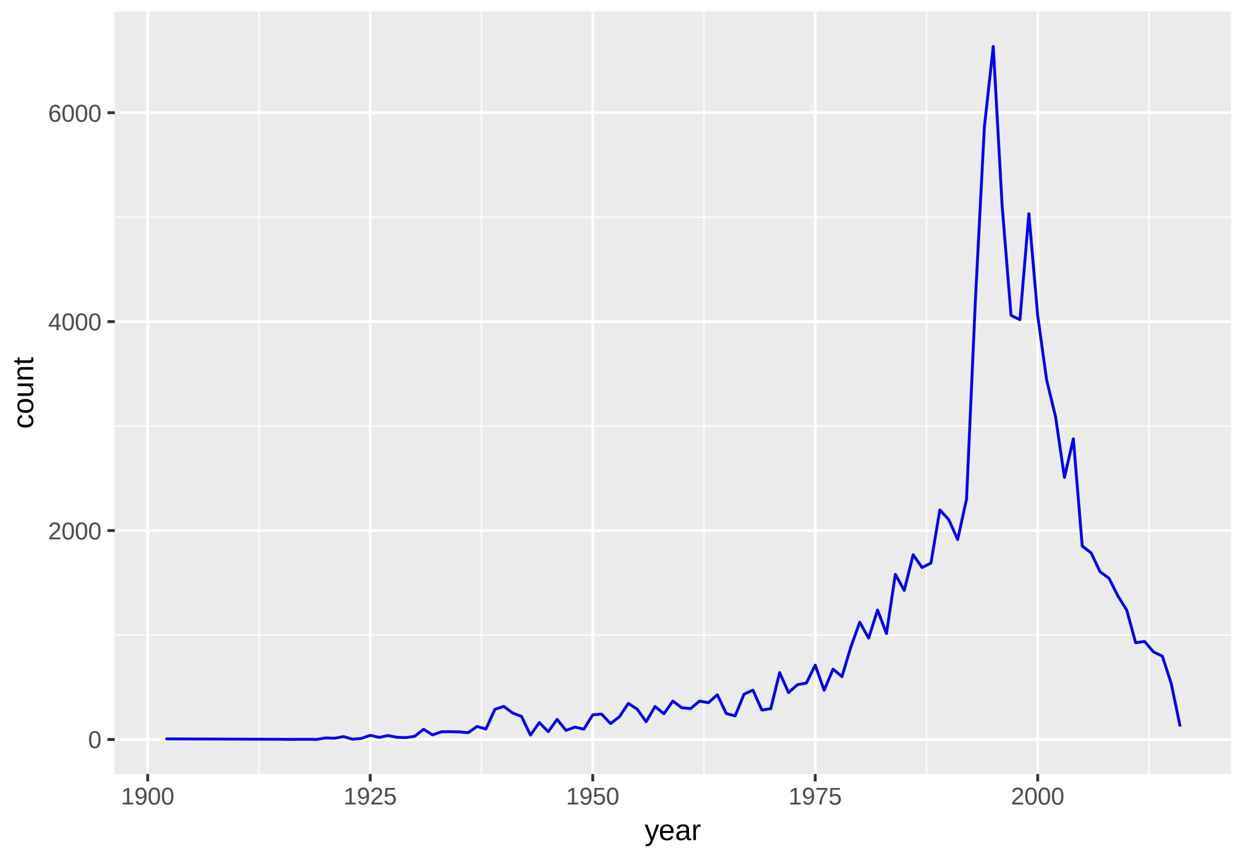
```r
#Check Number of Distinct User and Distinct Movies
movielens %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1     671     9066
```

```r
#Check Number of Movies per Year
movies_per_year <- movielens %>% na.omit() %>% select(movieId, year) %>% group_by(year) %>% summarise(co
print(movies_per_year)
```

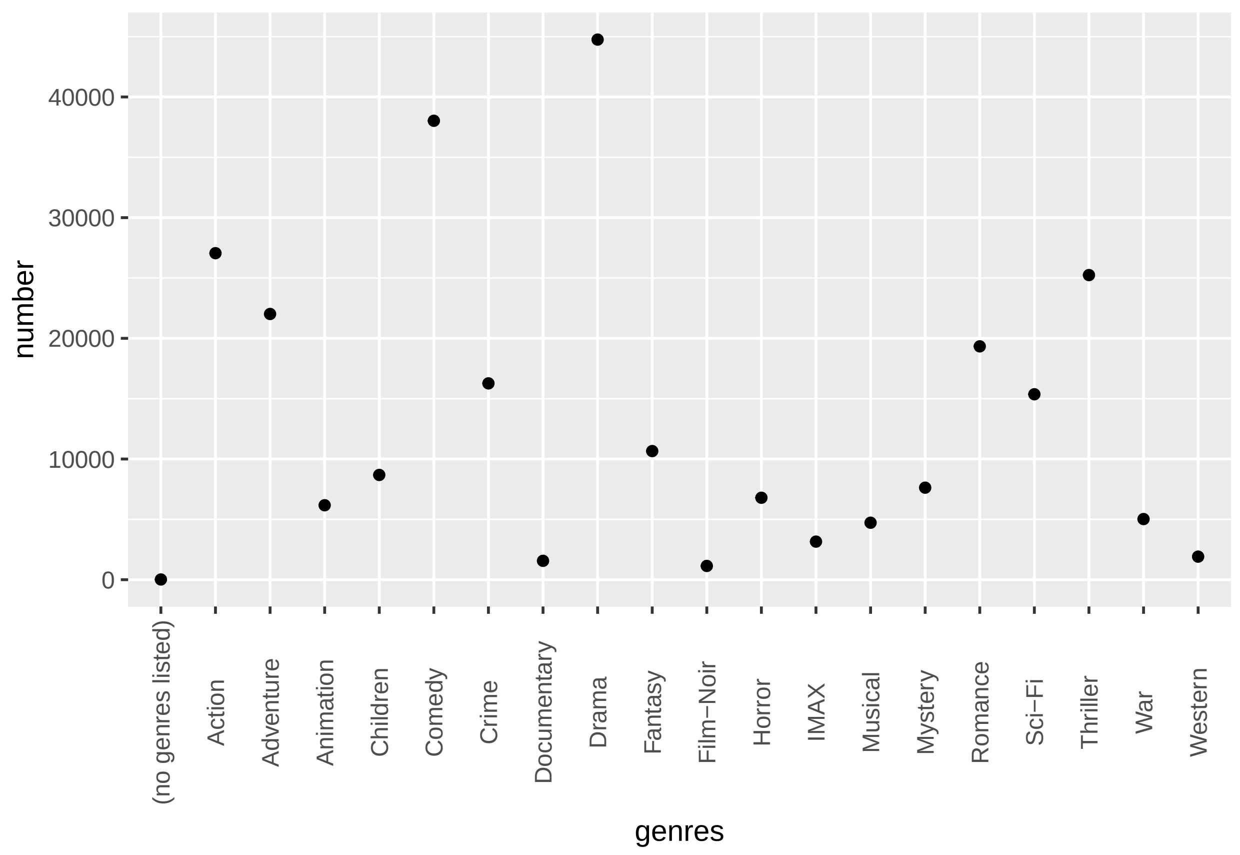
```
## # A tibble: 103 x 2
##     year count
##    <int> <int>
##  1  1902     6
##  2  1915     2
##  3  1916     1
##  4  1917     2
##  5  1918     2
##  6  1919     1
##  7  1920    15
##  8  1921    12
##  9  1922    28
## 10  1923     3
## # ... with 93 more rows
```

```
#Plot Number of Movies per Year
movies_per_year %>% ggplot(aes(x = year, y = count)) + geom_line(color="blue")
```

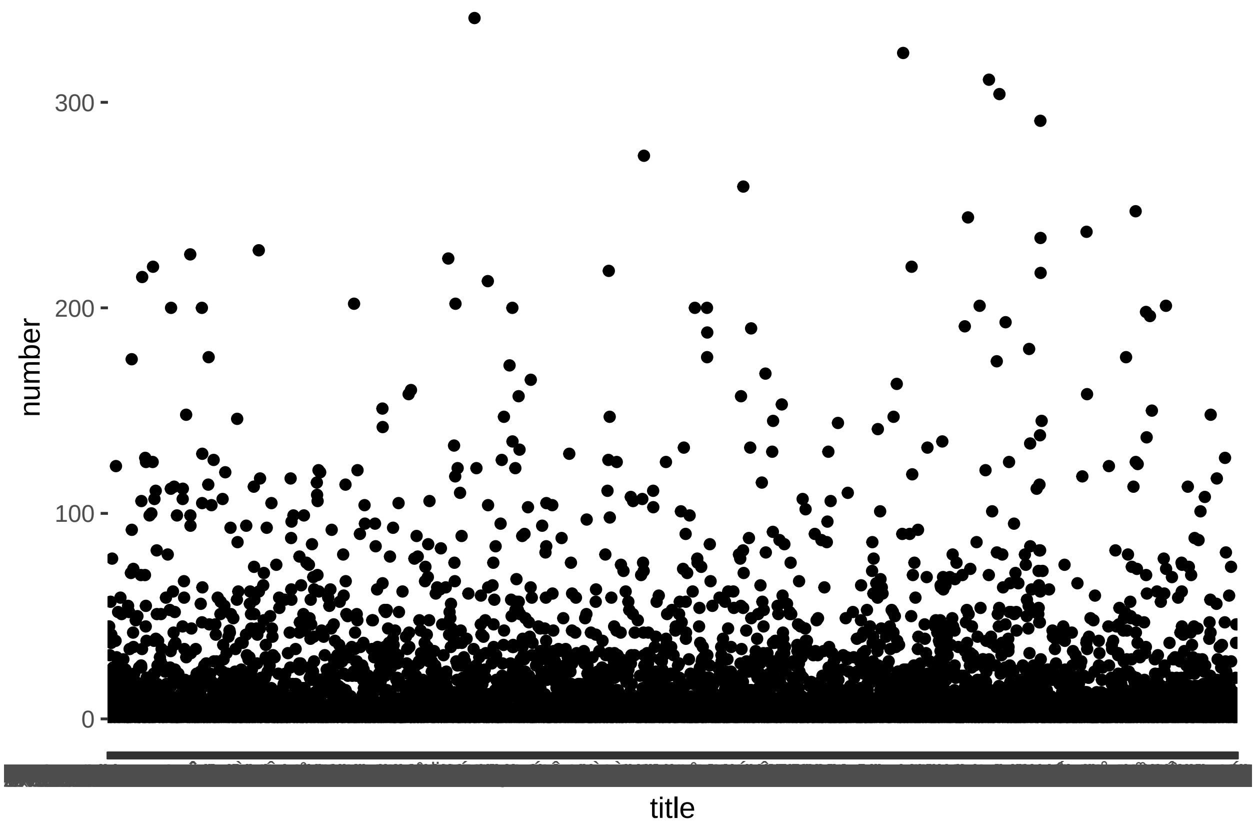


```
#Checking popular genre
genres_df <- movielens %>% separate_rows(genres, sep = "\\|") %>% group_by(genres) %>% summarise(number

#Plotting Popular genre
genres_df %>% ggplot(aes(x = genres, y = number)) + geom_point() + theme(axis.text.x = element_text(ang
```



```
#Plotting Ratings per movie
ratings_per_movie <- movielens %>% group_by(title) %>% summarise(number = n())
ratings_per_movie %>% ggplot(aes(x = title, y = number)) + geom_point()
```



5. Separating training and validation data

```r
##Create a Training Data Partition
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

#Creating the Validation data 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)
```

```
## Joining, by = c("movieId", "title", "year", "genres", "userId", "rating", "timestamp")
```

```r
edx <- rbind(edx, removed)
```

6. Using Random Forest Method for Predicting Outcomes

```r
#Setting Seed for Reproducibility of Results
set.seed(20)
```

```r
#Using the random forest method for training the model
trained_rf1 <- randomForest(rating~userId + movieId, data = edx)

#View the training results.
print(trained_rf1)

#Make Predictions using the trained model
predict_rf1 <- predict(trained_rf1, validation, type = "response")

#View the Prediction results.
print(predict_rf1)
```

7. Caculating Accuracy of Random Forest Method through RMSE

If $\hat{y_{u,i}}$ is the predicted outcome for movie i by user u and $y_{u,i}$ is the rating for movie i by user u, while N is the number of user-movie combinations, then the Root Mean Square Error (RMSE) is calculated as:

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

```r
#An RMSE Function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

#Calculating the Error through RMSE
RMSE(validation, predict_rf1)
```

8. Evaluating Model Performance

```r
#Using repeated 10 folds cross validation
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 10)

#Set up tuning grid for random forest
grid_rf <- expand.grid(.mtry = c(2, 4, 8, 16))

set.seed(20)
#Using Kappa metric to select the best model
m_rf <- train(rating~movieId + userId, data = edx, method = "rf", metric = "Kappa", trControl = ctrl, t

#Set up tuning grid for C5.0 Boosted Tree
grid_c50 <- expand.grid(.model = "tree", ,trials = c(10, 20, 30, 40), .winnow = "FALSE")

set.seed(20)
#Using the Kappa metric again
m_c50 <- train(rating~movieId + userId, data = edx, method = "C5.0", metric = "Kappa", trControl = ctrl

#For Comparison, the Random Forest model results are:
m_rf
```

```
#The results of boosted C5.0 model are:
m_c50
```

9. Conclusion

In this project, we used the random forest algorithm to predict movie ratings based on two predictors i.e. movieId and userId. The main problem that was encountered was the limited processing power of the available machine for this project. RStudio Cloud was also tried as an alternative, however, it took very large processing times also. Therefore, it was not possible to get the results of the random forest algorithm. However, the code presented in this report has been tested in pieces and it works without giving any error except the part where the random forest algorithm is applied for creating a trained model and then applying it to predict the outcomes i.e movie ratings. A git repository has been created for the project to make it convenient for others to have a look at the code and those with enough processing and memory resources can run and check it.