Recommendation Engine on Movielens Data

Jaydeep Chakraborty

EDX Data Science Capstone Project

June 21, 2020

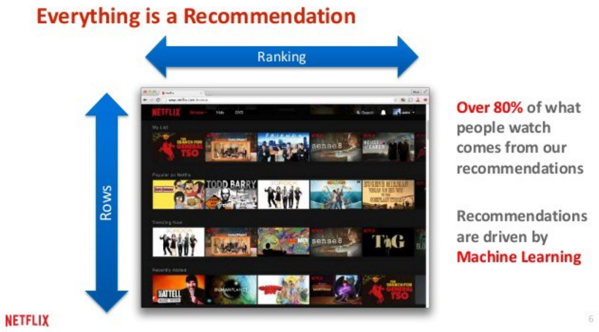
Outline

Building a movie recommendation engine

- The work done here is part of the Capstone Project submission for EDX Data Science Course
- The challenge involves predicting the ratings of a given movie for a given user
- We need to build a recommendation engine model to achieve this
- The data provided includes 10Mn observations with User Id, Movie Id, Ratings, Timestamp of Rating & Movie Genre. You can get the data from Grouplens' Movielens research
- There are 2 datasets:
  - Training dataset (with 90% observations) which will be used for building the model
  - Validation dataset (with 10% observations) which will be used for checking the error rate in prediction
- We need to build a model that can predict with RMSE less than 0.86490

Outline

All e-commerce companies use recommender systems


Everything is a Recommendation
Ranking
Rows
NETFLIX
Over 80% of what people watch comes from our recommendations
Recommendations are driven by Machine Learning
6

- 80% of stream time is achieved through Netflix's recommender system. It improves retention rate of existing customers, translating to savings on customer acquisition (estimated $1B per year as of 2016)
- 35% of all sales on Amazon are estimated to be generated by recommendation engine
- Linkedin utilises data from our past experience, current job titles & endorsements to suggest jobs

Outline

Recommender systems using 6 broad approaches

In this exercise we will use all the approaches except Deep Learning

- Linear Model: Ratings are predicted by adding individual movie, user & genre bias or effects to overall mean ratings. Learn more here.
- Linear Model with Regularization: Improves the liner model by adding a penalty to take care of high errors in effect of movies or users with lower number of ratings. Learn more here.
- Memory-Based Collaborative Filtering: Considers past behavior of users. Makes recommendation for an user by comparing with similar users basis their choice of movies and ratings. Learn more here.
- Model-Based Collaborative Filtering: Considers matrix factorization. Predicts the unknown ratings by changing the user-movie-rating matrix into a dot product of 2 lower dimensional orthogonal matrices. The matrices represents several important latent attributes of users & movies respectively. Learn more here.
- Ensemble Methods: Considers supervised learning algorithms for classification. Ratings are predicted by combining the predictions of several machine learning algorithms. Learn more here.
- Deep Learning: Considers matrix factorization. The 2 lower dimensional matrices doesn't need to be orthogonal. Uses 2 embedded neural network layers to find out the values of Users by Latent Factors matrix & Movies by Latent Factors matrix. Predicts the ratings through a layer that performs the dot product of output from embedded layers. Learn more here.

Outline

Steps involved in building the recommender system

- Environment set up: All required packages for data manipulation, visualization, model building & pdf authoring is installed & loaded. Separate R file for all user defined functions is loaded in the memory.
- Data load: 2 sets of data is imported from Movielens site - (i) A smaller dataset of 100K observations (ii) A larger Dataset or 10Mn observations. Each dataset is further divided into training (90% observations) & validation. This is done using the code provided in Capstone Project course.
- Data preparation: Derived variables like - year, month, day, day of week & hour are created from timestamp variable. Individual variables for each genre type is created to indicate the presence of respective genre.
- Data exploration: This is performed on 100K dataset to get the insights quickly on the local machine with limited memory. 2 types of exploration is performed - (i) Univariate analysis: Each variable - user id, movie id, rating, timestamp & genre are explored individually through visualizations & summary statistics. This helps us understand the overall trend in each variable. (ii) Bivariate analysis: Each variable is compared with ratings to understand effects of the independent variables with our dependent variable (ratings)
- Model selection: 15 different algorithms are built on training data from 100K dataset. RMSE on testing data of 100K dataset is compared for each algorithm. Algorithm with best performance i.e. lowest RMSE is selected. 100K dataset is used so that models can be built quickly on local machine with limited memory.
- Final model build: Selected algorithm from previous step is used to build the final model on training data from 10Mn dataset. This final model is then tested on validation data from 10Mn dataset. RMSE achieved from this is used for project submission.

Key aspects of the solution approach

- Uses 100K sample data to choose the best technique to run on 10Mn dataset. This allows us to try larger number of modeling techniques in quick time on a local machine with limited memory.
- 13 different modeling techniques were tried across 5 broad groups - (1) Linear Models, (2) Linear Models with Regularization (3) Memory Based Collaborative Filtering (4) Model Based Collaborative Filtering & (5) Ensemble Models
- Extends the learning from the program by considering Genre Effects in Linear Models.
- Utilizes all available variables to create several derived variables from genre & timestamp. They were used not only for explorations to draw interesting insights, but also used for model building to realize their impact on rating prediction.
- The project report is built using RMarkdown and binb package to enable pdf output with slides. Slides makes it easy to navigate and understand. In almost all cases (except a few) I have tried to keep only 1 slide per key step.

Facilitating easy understanding & execution of code

- The file InternalFunctions.R contains all the functions created by me. Each function allows us to implement a particular step of the solution. Several of these functions are called multiple times in the main code. This allows us to keep the main code simple and modular.
- All important datasets, models, predicted rating & rmse are provided separately. In case one doesn't want to execute the entire code due to time constraint or machine limitations, one can simply load these files and see the outputs themselves. All the saved objects have detailed & intuitive names for easy understanding.
- Detailed comments are provided in the report as well as the code to enable ease of understanding.
- Variable naming is kept uniform and intuitive to enable easy understanding throughout the program.

Navigating the project folder

- RecommendationEngine.pdf -Report
- RecommendationEngine.Rmd -R Markdown file for creating the report
- RecommendationEngine.R -Main code
- InternalFunctions.R -All functions created by me. These functions will be needed in main code.
- ExternalFunctions.R -All functions required by SlopeOne and SVD Approximation algorithms.
- SavedObjects -All the saved datasets, model objects, predicted rating vectors & RMSEs. All of them have uniform and intuitive names for easy understanding.
- All other files -Are used for quick report generation.

Outline

Environment set up

1. Defining list of all required packages

```
required.packages.data.manipulation <- c('Hmisc','data.table','plyr','tidyverse','pander','lubridate')
required.packages.visualization <- c('RColorBrewer','ggplot2','gridExtra')
required.packages.model <- c('caret','recommenderlab','recosystem','h2o')
required.packages.authoring <- c('rmarkdown','binb')
required.packages <- c(required.packages.data.manipulation,
                       required.packages.visualization,
                       required.packages.model,
                       required.packages.authoring)
```

2. Installing required packages if needed

```
packages.to.install <- required.packages[which(!required.packages %in% installed.packages()[,1])]
if(length(packages.to.install)>0) {
  cat('Following packages will be installed:\n', packages.to.install)
  install.packages(packages.to.install)
  packages.to.install <- required.packages[which(!required.packages %in% installed.packages()[,1])]
}
if(length(packages.to.install)>0) cat('Failed to install:\n', packages.to.install) else
  print('All required packages are installed.')
```

3. Loading in memory

```
#Loading required packages in memory
sapply(required.packages, require, character.only = TRUE)

# Loading user defined functions created to make the code modular & easy to understand
source('InternalFunctions.R')
```

Data import

1. Importing 10Mn Dataset & Creating Train & Test. Used for model finalization.

```
# Note: this process could take a couple of minutes
# Alternatively, you can load the datasets using load('DataLargeOriginal.rdata')
train.and.test = data.load('10mn')

# Using suffix B in the names for Big (10Mn) and S for Small (100K) dataset size
edxB = train.and.test$edx
validationB = train.and.test$validation
rm(train.and.test)

# Performing basic audit on data loaded
data.audit(edxB)
data.audit(validationB)

#save(edxB, validationB, file = 'DataLargeOriginal.rdata')
```

2. Importing 100K Dataset & Creating Train & Test. Used for selecting right algorthm.

```
# Alternatively, you can load the smaller datasets using load('SavedObjects/DataSmallOriginal.rdata')
train.and.test = data.load('100K')

# Using suffix B in the names for Big (10Mn) and S for Small (100K) dataset size
edxS = train.and.test$edx
validationS = train.and.test$validation
rm(train.and.test)

# Performing basic audit on data loaded
data.audit(edxS)
data.audit(validationS)

#save(edxS, validationS, file ='DataSmallOriginal.rdata')
```

Data preparation

Skip step on 10Mn Dataset. These variables are used only on 10K dataset for exploration & model building

1. Getting all the genres in separate columns.

```
# Adding genre flags on larger dataset (10Mn)
max.count.of.genres.in.a.movie <- max(str_count(unique(edxB$genres),'\\|'))+1
genre.colnames <- paste0('genre',c(1:max.count.of.genres.in.a.movie))
genre.summary <- edxB %>% separate(col = genres, into = genre.colnames,sep = '\\|', fill = 'right') %>%
  select(all_of(genre.colnames)) %>% gather() %>% group_by(value) %>% summarize(genre.count=n()) %>%
  arrange(desc(genre.count)) %>% filter(value != 'NA') %>%
  mutate(genre.perc=round(100*genre.count/nrow(edxB),0))
# Note: High run time expected.
edxB <- genre.detect(edxB)
validationB <- genre.detect(validationB)

# Adding genre flags on smaller dataset (100K)
# Alternatively load('SavedObjects/DataSmallWithGenreDateVars.rdata')
max.count.of.genres.in.a.movie <- max(str_count(unique(edxS$genres),'\\|'))+1
genre.colnames <- paste0('genre',c(1:max.count.of.genres.in.a.movie))
genre.summary <- edxS %>% separate(col = genres, into = genre.colnames,sep = '\\|', fill = 'right') %>%
                select(all_of(genre.colnames)) %>% gather() %>% group_by(value) %>%
                summarize(genre.count=n()) %>% arrange(desc(genre.count)) %>% filter(value != 'NA') %>%
                mutate(genre.perc=round(100*genre.count/nrow(edxS),0))
edxS <- genre.detect(edxS)
validationS <- genre.detect(validationS)
```

2. Adding date variables

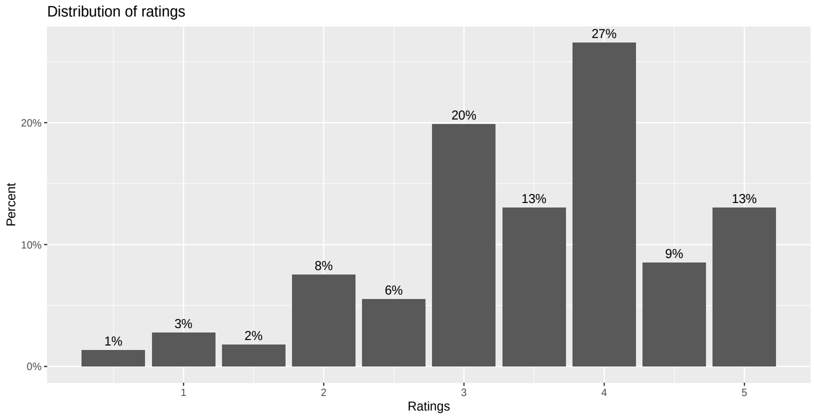
```
# Adding date variables on larger dataset (10Mn)
edxB <- edxB %>% mutate(date = as_datetime(timestamp), yr = lubridate::year(date),
mnth = lubridate::month(date), dt = lubridate::day(date), day = wday(date), hr= lubridate::hour(date))
validationB <- validationB %>% mutate(date = as_datetime(timestamp), yr = lubridate::year(date),
mnth = lubridate::month(date), dt = lubridate::day(date), day = wday(date), hr= lubridate::hour(date))
# Adding date variables on smaller dataset (100K)
edxS <- edxS %>% mutate(date = as_datetime(timestamp), yr = lubridate::year(date),
mnth = lubridate::month(date), dt = lubridate::day(date), day = wday(date), hr= lubridate::hour(date))
validationS <- validationS %>% mutate(date = as_datetime(timestamp), yr = lubridate::year(date),
mnth = lubridate::month(date), dt = lubridate::day(date), day = wday(date), hr= lubridate::hour(date))
```

Outline

1. Exploring dependent variable on 100K dataset

Exploring ratings or our dependent variable, that we wish to predict

```
edxS %>% ggplot(aes(x=rating)) + geom_bar(aes(y = ..prop..), stat="count") +
  geom_text(aes(y = ..prop.., label = scales::percent(..prop..,accuracy=1)), vjust = -0.5, stat="count") +
  scale_y_continuous(labels = scales::percent) + labs(y = 'Percent', x = 'Ratings') +
  ggtitle('Distribution of ratings')
```

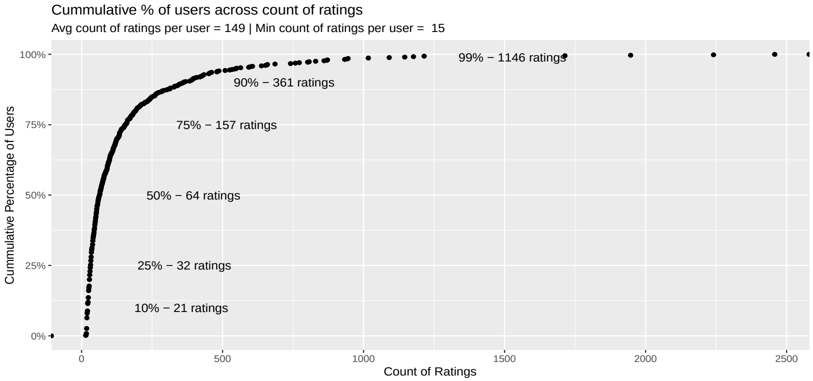


60% of all ratings are between 3 to 4.

2. Exploring independent variables on 100K dataset

UserId Exploration

```
p <- edxS %>% group_by(userId) %>% summarize(count_ratings = n()) %>% ggplot(aes(x=count_ratings)) +
    stat_ecdf(geom = "point") + scale_y_continuous(labels = scales::percent) +
  labs(y = 'Cummulative Percentage of Users', x = 'Count of Ratings')
cumm.plot.with.labels.users(p)
```



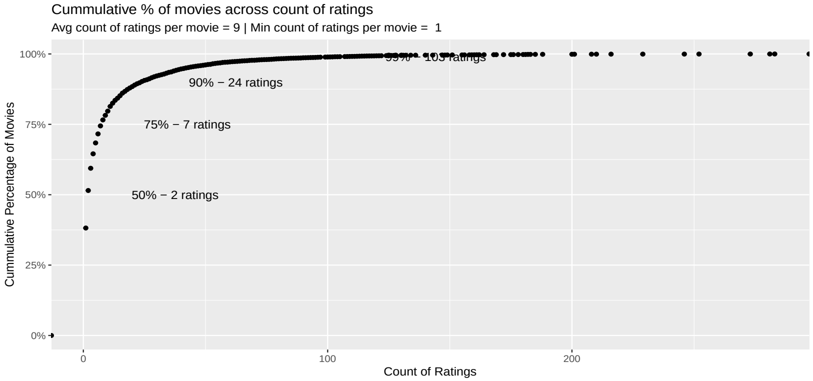
```
##  Min UserId =  1  |  Max UserId =  610  |  Count of distinct UserIds =  610
```

There are 610 users who have watched 91K movies. Meaning on an avg every one has watched 150 movies, however this is misleading as a few users have watched a very high number of movies. In fact 75% of users have actually watched less than 157 movies and 50% of users have watched less than 64 movies.

2. Exploring independent variables on 100K dataset (cont.)

MovieId Exploration

```
p <- edxS %>% group_by(movieId) %>% summarize(count_ratings = n()) %>% ggplot(aes(x=count_ratings)) +
  stat_ecdf(geom = "point") + scale_y_continuous(labels = scales::percent) +
  labs(y = 'Cummulative Percentage of Movies', x = 'Count of Ratings')
cumm.plot.with.labels.movies(p)
```



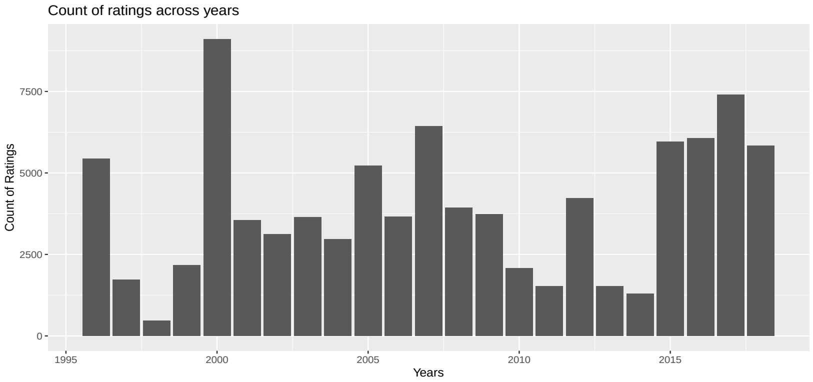
```
##  Min UserId =  1  |  Max UserId =  193609  |  Count of distinct UserIds =  9724
```

There are 9724 distinct movies and the range of movie Ids vary from 1 to 193K. While avg count of ratings per movie is 9, it is misleading as 50% of movies have only upto 2 ratings and 90% of movies have only upto 24 ratings. This is important insight as, in a larger dataset with more ratings per movie predictive ability of any modeling technique will improve over the smaller dataset.

2. Exploring independent variables on 100K dataset (cont.)

Date Exploration

```
edxS %>% group_by(yr) %>% summarize(count_ratings = n()) %>% ggplot(aes(x = yr, y=count_ratings)) +
  geom_bar(stat = 'sum',show.legend = F) + labs(x = 'Years', y = 'Count of Ratings') +
  ggtitle('Count of ratings across years')
```



```
##  Min Year =  1996 | Max Year =  2018
```

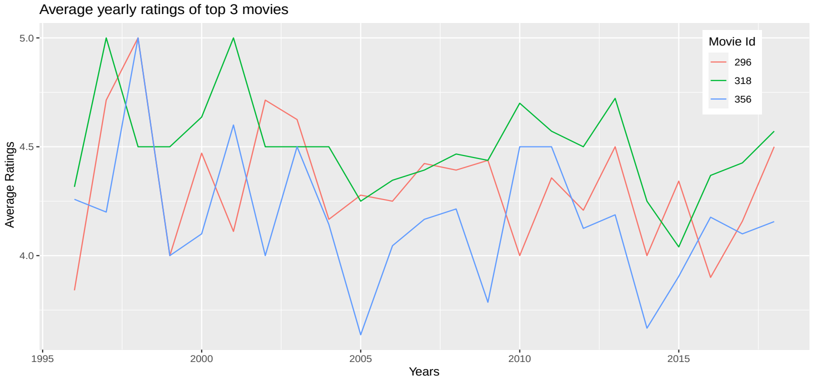
Ratings have been done in the period between 1996 & 2018. However some years have higher representation than others.

2. Exploring independent variables on 100K dataset (cont.)

Date Exploration - Top 3 Movies

Lets see if rating of the same movie varies across the years Lets choose the 3 most rated movies and lets look at their avg rating every year

```
top.3.rated.movies <- edxS %>% group_by(movieId) %>% summarize(count_ratings = n()) %>%
  arrange(desc(count_ratings)) %>% slice(1:3) %>% pull(movieId)
edxS %>% filter(movieId %in% top.3.rated.movies) %>% group_by(movieId, yr) %>%
  summarize(avg_ratings = mean(rating)) %>% ggplot(aes(x=yr, y = avg_ratings, color = as.factor(movieId))) +
  geom_line() + labs(x = 'Years', y = ' Average Ratings', color = 'Movie Id') +
  ggtitle('Average yearly ratings of top 3 movies') + theme(legend.position = c(0.9, 0.85))
```

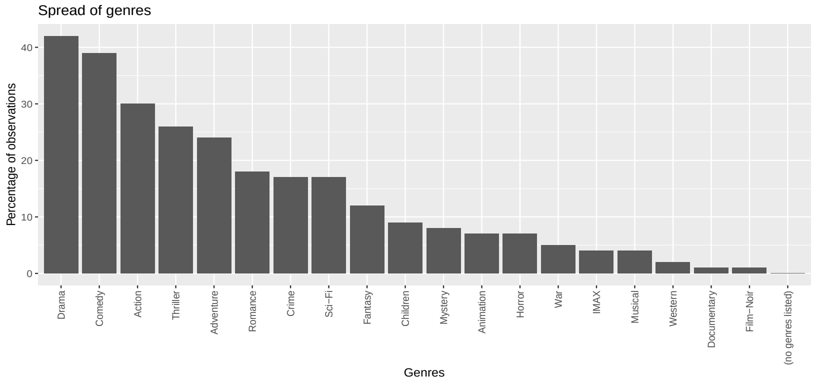


It is interesting to see that, while we might feel that a movie rating should not change much across the years, however, we can see that while usually among years avg rating of the same movie changes by 0.5 rating, there are instances when the change is more than 1 rating, which is a significant change. Time lapsed from the release of movie can be a good variable in rating prediction.

2. Exploring independent variables on 100K dataset (cont.)

Genre Exploration

```
genre.summary %>% ggplot(aes(x = reorder(value,-genre.perc), y = genre.perc)) +
  geom_bar(stat = 'sum', show.legend = F) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
  labs(x = 'Genres', y = 'Percentage of observations') + ggtitle('Spread of genres')
```

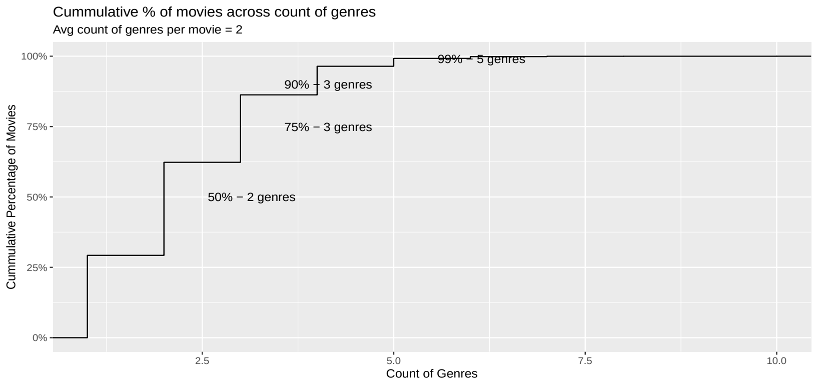


Most popular genres are Drama & Comedy, with around 40% of observations with these genres.

2. Exploring independent variables on 100K dataset (cont.)

Genre Exploration

```
p <- edxS %>% mutate(genre.count = rowSums(edxS[,7:26])) %>% group_by(movieId) %>%
  summarize(genre.count = mean(genre.count)) %>% ggplot(aes(x = genre.count)) + stat_ecdf(geom='step') +
  scale_y_continuous(labels = scales::percent) +
  labs(y = 'Cummulative Percentage of Movies', x = 'Count of Genres')
cumm.plot.with.labels.movies.genres(p)
```

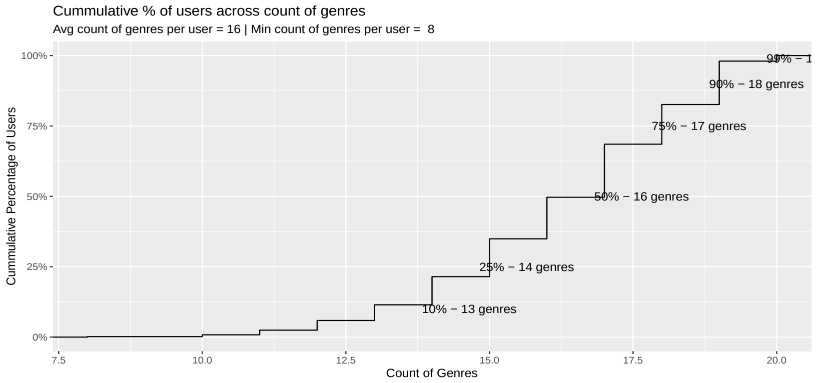


Avg number of genres per movie is 2.

2. Exploring independent variables on 100K dataset (cont.)

Genre Exploration

```
x <-  aggregate(edxS[,7:26],by = list(userId=edxS$userId),function(genre) if(any(genre==T)) return(1) else
  return(0))
x$Cnt.Genres.by.Users <- rowSums(x[,-1])
p <- x %>% ggplot(aes(x = Cnt.Genres.by.Users))+ stat_ecdf(geom='step') +
  scale_y_continuous(labels = scales::percent) +
  labs(y = 'Cummulative Percentage of Users', x = 'Count of Genres')
cumm.plot.with.labels.users.genres(p)
```



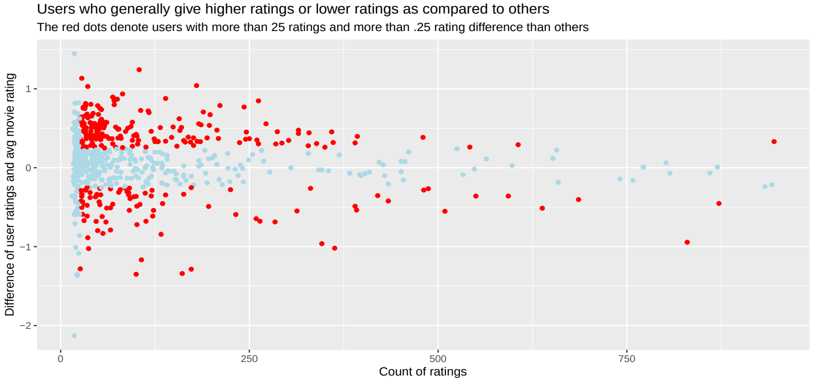
Avg number of genres per user is 16 & min number of genres watched by any user is 8.

3. Exploring relationship of ratings with independent variables

Ratings & Users

Is there a bias of users as they give ratings. Are there users who generally give lower ratings & are there users who generally give higher ratings?

```
edxS[,1:3] %>% group_by(userId) %>% mutate(avg.user.rating = mean(rating)) %>% ungroup() %>%
         group_by(movieId) %>% mutate(avg.movie.rating = mean(rating)) %>% ungroup() %>%
         mutate(diff.user.and.movie.rating = rating-avg.movie.rating) %>% group_by(userId) %>%
         summarize(count.of.ratings = n(), avg.diff.in.ratings = mean(diff.user.and.movie.rating)) %>%
  filter(count.of.ratings<1000) %>% ggplot(aes(x=count.of.ratings, y = avg.diff.in.ratings, color =
  ifelse(count.of.ratings>25 & abs(avg.diff.in.ratings)>0.25, 'High Diff', 'Normal'))) +
         geom_point(show.legend = F) + scale_color_manual(values=c("red", "lightblue")) +
  labs(x = 'Count of ratings', y = 'Difference of user ratings and avg movie rating') +
         ggtitle('Users who generally give higher ratings or lower ratings as compared to others', subtitle
   = 'The red dots denote users with more than 25 ratings and more than .25 rating difference than others')
```



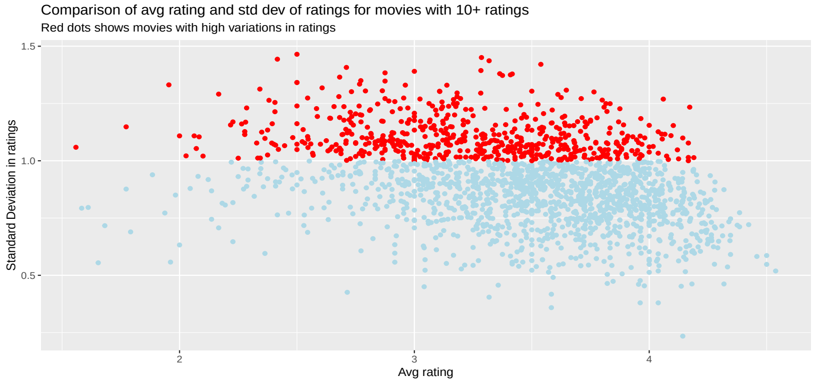
There is a user bias when ratings are given. Some users generally give lower than others and some higher.

3. Exploring relationship of ratings with independent variables (cont.)

Ratings & Movies

Is there variations within a movie ratings? We already know that 50% of movies have only 2 ratings. Our analysis will not make sense if we do not consider movies with at least a few more ratings. Lets consider movies with at least 10 ratings and lets see if there are variations in ratings.

```
edxS[,1:3] %>% group_by(movieId) %>% filter(n()>10) %>% summarize(count.of.ratings = n(), avg.rating =
mean(rating), sd.rating = sd(rating)) %>%
  ggplot(aes(x=avg.rating, y=sd.rating, color = ifelse(sd.rating>=1,'High variation','Less variation'))) +
  scale_color_manual(values=c("red", "lightblue")) + geom_point(show.legend = F) +
  labs(x = 'Avg rating', y = 'Standard Deviation in ratings') +
  ggtitle('Comparison of avg rating and std dev of ratings for movies with 10+ ratings', subtitle =
          'Red dots shows movies with high variations in ratings')
```

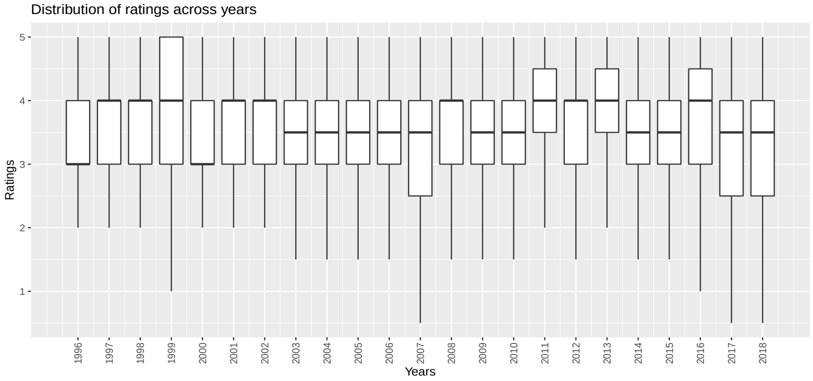


There is considerable variations within the ratings. It can happen in lower rated movies as well as higher rated movies.

3. Exploring relationship of ratings with independent variables (cont.)

Ratings & Year

```
edxS%>% ggplot(aes(y = rating, x = yr, group = yr)) + geom_boxplot(outlier.shape = NA) +
  scale_x_continuous(name ="Years", breaks=seq(1996,2018,1)) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
  labs(x = 'Years', y = 'Ratings') + ggtitle('Distribution of ratings across years')
```

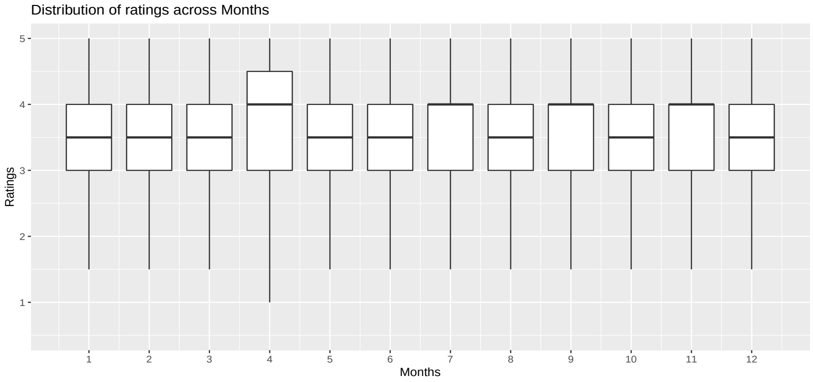


There are few years when ratings are higher or lower than usual.

3. Exploring relationship of ratings with independent variables (cont.)

Ratings & Months

```
edxS%>% ggplot(aes(y = rating, x = mnth, group = mnth)) + geom_boxplot(outlier.shape = NA) +
  scale_x_continuous(name ="Months", breaks=seq(1,12,1)) + labs(y = 'Ratings') +
  ggtitle('Distribution of ratings across Months')
```

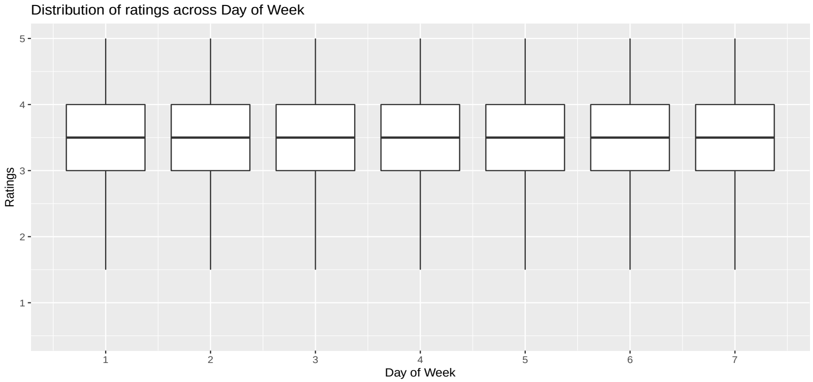


There are few months when ratings are higher than usual.

3. Exploring relationship of ratings with independent variables (cont.)

Ratings & Day of week

```
edxS%>% ggplot(aes(y = rating, x = day, group = day)) + geom_boxplot(outlier.shape = NA) +
  scale_x_continuous(name =”Day of Week”, breaks=seq(1,7,1)) + labs(y = ’Ratings’) +
  ggtitle(’Distribution of ratings across Day of Week’)
```

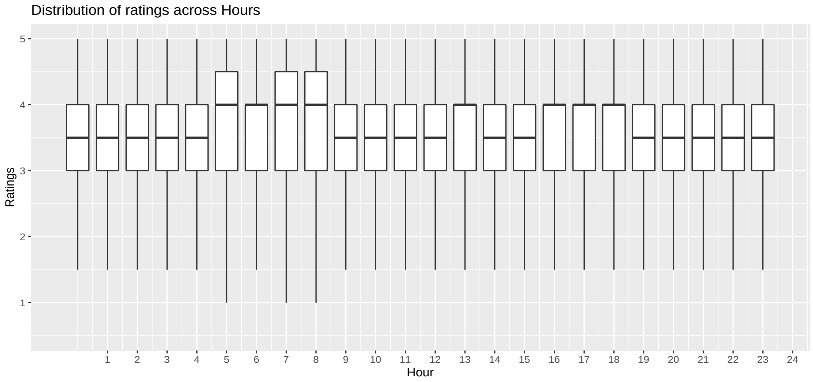


Ratings do not change basis day of week.

3. Exploring relationship of ratings with independent variables (cont.)

Ratings & Hour

```
edxS%>% ggplot(aes(y = rating, x = hr, group = hr)) + geom_boxplot(outlier.shape = NA) +
  scale_x_continuous(name ="Hour", breaks=seq(1,24,1)) + labs(y = 'Ratings') +
  ggtitle('Distribution of ratings across Hours')
```

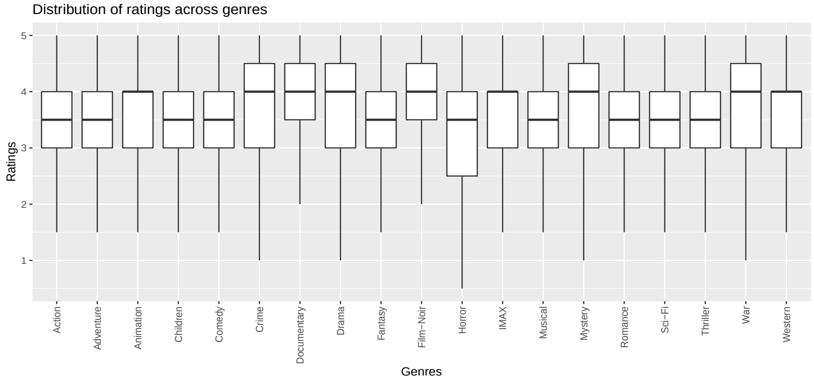


Ratings are higher early in the morning or during evening than usual.

3. Exploring relationship of ratings with independent variables (cont.)

Ratings & Genre

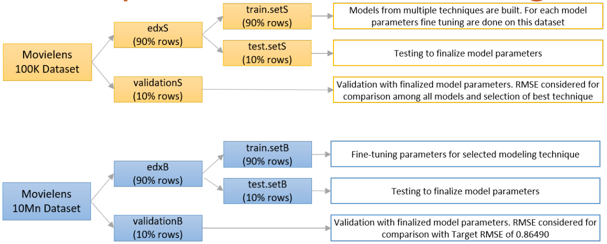
```
edxS[,c(3,7:25)] %>% gather(key = 'genre', value='presence',-rating) %>% filter(presence == T) %>%
  ggplot(aes(x=genre, y = rating, group = genre)) + geom_boxplot(outlier.shape = NA) +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))+
  labs(x = 'Genres', y = 'Ratings') + ggtitle('Distribution of ratings across genres')
```



There is a significant impact of genres on ratings.

Outline

1. Data Preparation for Model Building


Movielens 100K Dataset
edxS (90% rows)
validationS (10% rows)
train.setS (90% rows)
test.setS (10% rows)
Models from multiple techniques are built. For each model parameters fine tuning are done on this dataset
Testing to finalize model parameters
Validation with finalized model parameters. RMSE considered for comparison among all models and selection of best technique
Movielens 10Mn Dataset
edxB (90% rows)
validationB (10% rows)
train.setB (90% rows)
test.setB (10% rows)
Fine-tuning parameters for selected modeling technique
Testing to finalize model parameters
Validation with finalized model parameters. RMSE considered for comparison with Target RMSE of 0.86490

Creating train_set & test_set on edxB (larger dataset)

```
test.and.train.big <- create.test.and.train.from.edx(edxB)
train.setB <- test.and.train.big$train_set
test.setB <- test.and.train.big$test_set
rm(test.and.train.big)
```

Creating train_set & test_set on edxS (smaller dataset)

```
test.and.train.small <- create.test.and.train.from.edx(edxS)
train.setS <- test.and.train.small$train_set
test.setS <- test.and.train.small$test_set
rm(test.and.train.small)
```

Creating a blank dataset to store all model RMSE for comparison

```
rmse.results <- data.frame(SNo = integer(), ModelType = character(), Algorithm = character(),
RMSE = double(), stringsAsFactors=F)
```

2. Baseline Model

Model 1: Random Rating with equal probability

In this baseline model we randomly predict every rating by choosing a rating between 0.5 to 5 with each rating having an equal probability of selection.
The model is trained on edxS (91K) and tested on validationS (9.7K).

```
model.param.rating.prob <- rep(1/length(unique(edxS$rating)),10)
predicted.rating <- sample(seq(0.5,5,0.5), size = nrow(validationS), replace = T, prob = model.param.rating.prob)
model.rmse <- rmse(validationS$rating,predicted.rating)
rmse.results[nrow(rmse.results)+1,] <- c(1, 'Baseline model', 'Random rating with 0.1 prob', model.rmse)
```

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563

2. Baseline Model (cont.)

Model 2: Random Rating with existing probability in training dataset

In this baseline model we randomly predict every rating by choosing a rating between 0.5 to 5. But this time we decide the probability of any value to get picked up basis the existing probability in the training dataset (edxS). The model is trained on edxS (91K) and tested on validationS (9.7K).

```
model.param.rating.prob <- edxS %>% group_by(rating) %>% summarise(perc = n()/nrow(edxS)) %>% pull(perc)
predicted.rating <- sample(seq(0.5,5,0.5), size = nrow(validationS), replace = T, prob = model.param.rating.prob)
model.rmse <- rmse(validationS$rating,predicted.rating)
rmse.results[nrow(rmse.results)+1,] <- c(2, 'Baseline model', 'Random rating with existing prob', model.rmse)
```

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903

2. Baseline Model (cont.)

Model 3: Mean Rating

In this baseline model we consider average of all ratings in the training data as the prediction for any userId or movieId in testing data. The model is trained on edxS (91K) and tested on validationS (9.7K).

```
model.param.meanRating <- mean(edxS$rating)
predicted.rating <- rep(model.param.meanRating,nrow(validationS))
model.rmse <- rmse(validationS$rating,predicted.rating)
rmse.results[nrow(rmse.results)+1,] <- c(3, 'Baseline model', 'Mean rating', model.rmse)
```

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498

3. Linear Model

Model 4: Movie Effect

In this linear model we add to the mean ratings, the individual movie effects. Individual movie effects and overall mean ratings are calculated in the training data and then predictions are made at a movie level in the test dataset.

```
mean.rating <- mean(edxS$rating)
movie.effect <- edxS %>% group_by(movieId) %>% summarize(b_i = mean(rating - mean.rating))

predicted.rating <- validationS %>% left_join(movie.effect,by='movieId') %>%
  mutate(pred = mean.rating + b_i) %>% pull(pred)
model.rmse <- rmse(validationS$rating,predicted.rating)
rmse.results[nrow(rmse.results)+1,] <- c(4, 'Linear model', 'Movie effect only', model.rmse)
```

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328

3. Linear Model (cont.)

Model 5: Movie & User Effect

In this linear model we add to the mean ratings, the individual movie & user effects. Individual movie effects, user effects & overall mean ratings are calculated in the training data & then predictions are made at a movie level in the test dataset.

```
mean.rating <- mean(edxS$rating)
user.effect <- edxS %>% left_join(movie.effect,by='movieId') %>%
  group_by(userId) %>% summarize(u_i = mean(rating - mean.rating- b_i))

predicted.rating <- validationS %>% left_join(movie.effect,by='movieId') %>%
  left_join(user.effect,by='userId') %>%
  mutate(pred = mean.rating + b_i + u_i) %>%
  pull(pred)
model.rmse <- rmse(validationS$rating,predicted.rating)
rmse.results[nrow(rmse.results)+1,] <- c(5, 'Linear model', 'Movie & user effect', model.rmse)

view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639

3. Linear Model (cont.)

Model 6: Movie, User & Genre Effect

In this linear model we add to the mean ratings, the individual movie, user & genre effects. This is an extension of what we learnt in the program. Top 6 genres (basis frequency) effects are considered individually. Individual movie effects, user effects, genre effects & overall mean ratings are calculated in the training data & then predictions are made at a movie level in the test dataset.

```
mean.rating <- mean(edxS$rating)
user.effect.Drama <- edxS %>% left_join(movie.effect,by='movieId') %>%
  group_by(userId,Drama) %>% summarize(u_i_Drama = mean(rating - mean.rating- b_i))
edxS2 <- edxS %>% left_join(movie.effect,by='movieId') %>%
  left_join(user.effect.Drama,by=c('userId','Drama'))
user.effect.Comedy <- edxS2 %>% group_by(userId,Comedy) %>%
              summarize(u_i_Comedy = mean(rating - mean.rating- b_i-u_i_Drama))
edxS2 <- edxS2 %>% left_join(user.effect.Comedy,by=c('userId','Comedy'))
user.effect.Action <- edxS2 %>% group_by(userId,Action) %>%
  summarize(u_i_Action = mean(rating - mean.rating- b_i-u_i_Drama-u_i_Comedy))
edxS2 <- edxS2 %>% left_join(user.effect.Action,by=c('userId','Action'))
user.effect.Thriller <- edxS2 %>% group_by(userId,Thriller) %>%
  summarize(u_i_Thriller = mean(rating - mean.rating- b_i-u_i_Drama-u_i_Comedy-u_i_Action))
edxS2 <- edxS2 %>% left_join(user.effect.Thriller,by=c('userId','Thriller'))
user.effect.Adventure <- edxS2 %>% group_by(userId,Adventure) %>%
  summarize(u_i_Adventure = mean(rating - mean.rating- b_i-u_i_Drama-u_i_Comedy-u_i_Action-u_i_Thriller))
edxS2 <- edxS2 %>% left_join(user.effect.Adventure,by=c('userId','Adventure'))
user.effect.Romance <- edxS2 %>% group_by(userId,Romance) %>%
  summarize(u_i_Romance = mean(rating - mean.rating- b_i-u_i_Drama-u_i_Comedy-u_i_Action-u_i_Thriller-u_i_Adventure))
edxS2 <- edxS2 %>% left_join(user.effect.Romance,by=c('userId','Romance'))
user.effect.OtherGenre <- edxS2 %>% group_by(userId) %>%
  summarize(u_i_OtherGenre =
mean(rating - mean.rating-b_i-u_i_Drama-u_i_Comedy-u_i_Action-u_i_Thriller-u_i_Adventure-u_i_Romance))
edxS2 <- edxS2 %>% left_join(user.effect.OtherGenre,by='userId')
```

3. Linear Model (cont.)

Model 6: Movie, User & Genre Effect. Code continued.

```
predicted.rating <- validationS %>% left_join(movie.effect,by='movieId') %>%
  left_join(user.effect.Drama,by=c('userId','Drama')) %>%
  left_join(user.effect.Comedy,by=c('userId','Comedy')) %>%
  left_join(user.effect.Action,by=c('userId','Action')) %>%
  left_join(user.effect.Thriller,by=c('userId','Thriller')) %>%
  left_join(user.effect.Adventure,by=c('userId','Adventure')) %>%
  left_join(user.effect.Romance,by=c('userId','Romance')) %>%
  left_join(user.effect.OtherGenre,by='userId') %>%
  mutate(pred = mean.rating + b_i + coalesce(u_i_Drama,0) + coalesce(u_i_Comedy,0) +
           coalesce(u_i_Action,0)+ coalesce(u_i_Thriller,0) + coalesce(u_i_Adventure,0) +
           coalesce(u_i_Romance,0) + coalesce(u_i_OtherGenre,0)) %>% pull(pred)

model.rmse <- rmse(validationS$rating,predicted.rating)
rmse.results[nrow(rmse.results)+1,] <- c(6, 'Linear model', 'Movie, user & genre effect', model.rmse)
rm(edxS2)
```

Adding the genre effects improves the model predictions, but only marginally.

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152

4. Linear Model with Regularization

Model 7 : Movie & User Effect with Regularization

In this linear model we regularize the movie & user effect so that we can reduce the errors caused by considering the effects of less popular movies or users. train.setS & test.setS are used for finding the optimal regularization parameter. validationS is used to calculate the final RMSE.

```
#Choosing different lambda for movie & user. The range of lambdas for testing is selected after multiple iterations.
#Refer the code for further details. Here in order to reduce time, smaller range closer to optimal lambdas are used.
lambdas_m <- seq(2, 5, 0.5)
lambdas_u <- seq(2, 5, 0.5)
lambdas <- expand.grid(lambdas_m, lambdas_u)

# Note - this will take time to run.
rmses <- apply(lambdas, 1, model.regularization, train=train.setS, test=test.setS)
cat('Best model parameters \nMovie Lambda - ', lambdas[which.min(rmses),1],
    '\nUser Lambda - ', lambdas[which.min(rmses),2], '\nMin RMSE: ', min(rmses))

rmse.linear.regularized <-
  model.regularization(c(lambdas[which.min(rmses),1],lambdas[which.min(rmses),2]), edxS, validationS)
rmse.results[nrow(rmse.results)+1,] <- c(7, 'Linear model', 'Movie & user effect with reg.', rmse.linear.regularized)
```

We can see that regularization improves the model predictions with movie & genre effect

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788

4. Linear Model with Regularization (cont.)

Model 8 : Movie, User & Genre Effect with Regularization

In this linear model we regularize the movie & user effect while considering genre effect.
This is again an extension of the regularization model that we learnt in the program.

```
# Choosing different lambda for movie & user.
# The range of lambdas for testing is selected after multiple iterations.
# Refer the code for further details.
# Here in order to reduce time, smaller range closer to optimal lambdas are used.
lambdas_m <- seq(2, 5, 0.25)
lambdas_u <- seq(15, 20, 0.25)
lambdas <- expand.grid(lambdas_m, lambdas_u)

# Using train.setS & test.setS to estimate the lambdas.
rmses <- apply(lambdas, 1, model.regularization.with.genre, train=train.setS, test=test.setS)
cat('Best model parameters \nMovie Lambda - ', lambdas[which.min(rmses),1],
      '\nUser Lambda - ', lambdas[which.min(rmses),2], '\nMin RMSE: ', min(rmses))

# Using these model parameters (movie & user lambda) on validation dataset
rmse.linear.regularized.genre <-
model.regularization.with.genre(l = c(lambdas[which.min(rmses),1],lambdas[which.min(rmses),2]), train = edxS, test =
rmse.results[nrow(rmse.results)+1,] <-
  c(8, 'Linear model', 'Movie, user & genre effect with regularization', rmse.linear.regularized.genre)
```

- We can see that this significantly improves the model predictions and this is already better than the target RMSE of 0.86490 on the validation data.
- We should be able to improve this further by considering more individual genre effects

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533

5. Memory-Based Collaborative Filtering

Model 9 : User Based Collaborative Filtering (UBCF)

Important consideration for Memory-Based Collaborative Filtering

- For Collaborative Filtering technique, we can not use the existing training & validation sets
- We will need same items (or movies) for all users in training & validation
- Also for validation we will need same set of users with some ratings hidden away
- The model will predict the unknown ratings for the users in validation set basis the known ratings

We can not compare the RMSE of these models with what we got from other models

- We will also need to reduce the number of users and movies further for this model to run otherwise R fails to allocate memory in larger matrix
- This can be done by considering movies & users with a minimum threshold of ratings
- Hence RMSE achieved from UBCF will not be comparable with RMSEs achieved earlier, however they will still give a good indication of performance
- If RMSE achieved on popular movies & popular users is not the best RMSE among all other techniques than we will not worry about this method
- If UBCF RMSE is better than all other methods then we will try to figure out how to compare RMSEs in a better way

Steps involved in building Collaborative Filtering model on our 100K movie data set

- 1. Make union of edxS & validationS to make one single dataset - 'all.data.S'
- 2. Create a selection from this dataset for popular movies & users - 'popular.all.data.S'
- 3. Convert this dataset to Real Rating Matrix format (as RecommenderLab package needs this package) - real.rating.popular.all.data.S
- 4. Create 3 real rating matrices from this: – (a) Training : training.all.data.S – (b) Validation Known : validation.known.all.data.S – (c) Validation Unknown : validation.unknown.all.data.S
- 5. Run User Based Collaborative Filtering (UBCF) model on training.all.data.S : model.UBCF
- 6. Predict unknown ratings on validation.known.all.data.S using model.UBCF
- 7. Measure RMSE by comparing the unknown ratings with ratings on validation.unknown.all.data.S

5. Memory-Based Collaborative Filtering

Model 9 : User Based Collaborative Filtering (cont.)

```
# 1. Make union of edxS & validationS to make one single dataset - 'all.data.S'
all.data.S <- rbind(edxS[,1:3], validationS[,1:3])

# 2. Create a selection from this dataset for popular movies & users - 'popular.all.data.S'
minRowCnt <- 20 # Selecting users who have rated at least 20 movies
minColCnt <- 50 # Selecting movies which has been rated by at least 50 users
popular.all.data.S <- all.data.S %>% group_by(movieId) %>% filter(n()>minColCnt) %>% ungroup() %>%
  group_by(userId) %>% filter(n()>minRowCnt) %>% ungroup()

# 3. Convert this dataset to Real Rating Matrix format
# as RecommenderLab package needs this package : real.rating.popular.all.data.S
# I have created my own function to achieve this
real.rating.popular.all.data.S <- convert.all.data.to.realRatingMatrix(popular.all.data.S)
real.rating.popular.all.data.S # 466 users and 436 movies selected

# 4. Create 3 real rating matrices from this: (a) Training - training.all.data.S
# (b) Validation Known - validation.known.all.data.S (c) Validation Unknown - validation.unknown.all.data.S

# Checking for R Version and using set.seed function appropriately
if(as.numeric(R.Version()$major)==3 & as.numeric(R.Version()$minor) >5)
  set.seed(1, sample.kind="Rounding") else set.seed(1)

n_fold <- 10  # k value for k fold cross validation
items_to_keep <- 15  # Items to consider in training set (less than min no of ratings )
rating_threshold <- 3.5  # Considering a rating of 3.5 as good rating across all movies

eval_sets <- evaluationScheme(data = real.rating.popular.all.data.S, method = "cross-validation",
                              k = n_fold, given = items_to_keep, goodRating = rating_threshold)
training.all.data.S <- getData(eval_sets, "train")  # training set
validation.known.all.data.S <- getData(eval_sets, "known")  # known validation set
validation.unknown.all.data.S <- getData(eval_sets, "unknown")  # unknown validation set
```

- 89% users are present in training. 414 out of 466 users.
- 11% users are present in testing. 52 out of 466 users. 15 ratings of each user is kept.
- Same 10% users as above are kept in validation. Each user has 20+ ratings. Remaining ratings post previous selection is kept here.
- So we will be predicting at least 5 ratings for each of the 26 users

5. Memory-Based Collaborative Filtering

Model 9 : User Based Collaborative Filtering (cont.)

```
# 5. Run User Based Collaborative Filtering (UBCF) model on training.all.data.S - model.UBCF
model.UBCF <- Recommender(data = training.all.data.S, method = ”UBCF”, parameter = list(method = ”Cosine”))

# 6. Predict unknown ratings on validation.known.all.data.S using model.UBCF - prediction.UBCF
items_to_recommend <- 10 # upto 10 unknown ratings will be predicted
prediction.UBCF <- predict(object = model.UBCF, newdata = validation.known.all.data.S,
                           n = items_to_recommend, type = ”ratings”)

# 7. Measure RMSE by comparing the unknown ratings with ratings on validation.unknown.all.data.S - rmse.UBCF
rmse.UBCF <- calcPredictionAccuracy(x = prediction.UBCF, data = validation.unknown.all.data.S,
                                    byUser = FALSE)[1]

rmse.results[nrow(rmse.results)+1,] <- c(9, ’Memory Based Collaborative Filtering’,
’UBCF - Movies (50+ ratings) & Users (20+ ratings)’, round(rmse.UBCF,5))
```

- RMSE of 0.9081786 can be improved further if we select users with higher number of movies rated
- RMSE drops to 0.8717059 if we select only users with minimum of 50 movie rated
- RMSE should improve on larger datasets where we have more users and ratings
- UBCF RMSE is not better than our earlier model, hence we will not analyze this technique further

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818

5. Memory-Based Collaborative Filtering

Model 10 : Item Based Collaborative Filtering

We will utilize the training, validation known & validation unknown datasets from UBCF technique

```
# 1. Run Item Based Collaborative Filtering (UBCF) model on training.all.data.S - model.IBCF
model.IBCF <- Recommender(data = training.all.data.S, method = "IBCF", parameter = list(method = "Cosine"))

# 2. Predict unknown ratings on validation.known.all.data.S using model.UBCF - prediction.UBCF
items_to_recommend <- 10 # upto 10 unknown ratings will be predicted
prediction.IBCF <- predict(object = model.IBCF, newdata = validation.known.all.data.S,
                           n = items_to_recommend, type = "ratings")

# 3. Measure RMSE by comparing the unknown ratings with ratings on validation.unknown.all.data.S - rmse.IBCF
rmse.IBCF <- calcPredictionAccuracy(x = prediction.IBCF, data = validation.unknown.all.data.S, byUser = FALSE)[1]
rmse.results[nrow(rmse.results)+1,] <- c(10, 'Memory Based Collaborative Filtering',
'IBCF - Movies (50+ ratings) & Users (20+ ratings)', round(rmse.IBCF,5))
```

- RMSE has increased to 1.262425
- We can see that CF models can perform better only if we have users with several rated movies
- In our case linear models with regularization have done better than CF models

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242

6. Model-Based Collaborative Filtering

Model 11 : Slope One

- Slope One needs input in data table format with names as user_id, item_id & rating. Also user_id & item_id should be character.
- We will use the function - data.prepare.for.slope.one to get datasets ready for Slope One model
- Other functions used to build the SlopeOne model are kept in file ExternalFunctions.R

```
# 1. Getting the data ready for Slope One model
edxS.SO <- data.prepare.for.slope.one(edxS[,1:3])
validationS.SO <- data.prepare.for.slope.one(validationS[,1:3])
# 2. Normalize ratings
edxS.SO.norm <- normalize_ratings(edxS.SO)
# 3. Build Slope One model
model.SO <- build_slopeone(edxS.SO.norm$ratings)
# 4. Make predictions on validation dataset (this step will take time to run)
predictions.SO <- predict_slopeone(model.SO, validationS.SO[ , c(1, 2), with = FALSE], edxS.SO.norm$ratings)
unnormalized.predictions.SO <- unnormalize_ratings(normalized = edxS.SO.norm, ratings = predictions.SO)
# 5. Calculating RMSE
rmse.SO <- rmse(validationS.SO$rating,unnormalized.predictions.SO$predicted_rating)
rmse.results[nrow(rmse.results)+1,] <- c(11, 'Model Based Collaborative Filtering', 'Slope One', round(rmse.SO,5))
```

- RMSE of 0.8771416 is not lower than linear model with regularized user, movie & genre effects
- This can improve on the larger datasets with movies & users having more ratings

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242
11	Model Based Collaborative Filtering	Slope One	0.83247

6. Model-Based Collaborative Filtering

Model 12 : SVD with Approximation

```
# 1. Make union of edxS & validationS to make one single dataset - 'all.data.S'
all.data.S <- rbind(edxS[,1:3], validationS[,1:3])

# 2. Create a selection from this dataset for popular movies & users - 'popular.all.data.S'
minColCnt <- 4 # Selecting movies which has been rated by at least 50 users
popular.all.data.S <- all.data.S %>% group_by(movieId) %>% filter(n()>minColCnt) %>% ungroup()

# 3. Convert this dataset to format required by SVDApproximation package - 'all.data.S'
all.data.S <- create.sequenced.data(popular.all.data.S)[,c(1,2,5)]
colnames(all.data.S) <- c('user', 'item', 'rating')
all.data.S <- data.table(all.data.S)

# 4. Create training, test & validation matrices from this
if(as.numeric(R.Version()$major)==3 & as.numeric(R.Version()$minor) >5)
  set.seed(1, sample.kind="Rounding") else set.seed(1)
mtx <- split_ratings(ratings_table = all.data.S, proportion = c(0.7, 0.15, 0.15))

# 5. Run SVD Approximation model - 'model.svdApprox'
model.svdApprox <- svd_build(mtx)

# 6. Fine tune model parameter r by comparing performance with validation matrix - 'model_tunes'
model_tunes <- svd_tune(model.svdApprox, r = 2:50)

# 7. Measure RMSE by comparing ratings in test matrix - 'rmse.svdApprox'
rmse.svdApprox <- svd_rmse(model.svdApprox, r = model_tunes$r_best, rmse_type = c("test"))
rmse.results[nrow(rmse.results)+1,] <- c(12, 'Model Based Collaborative Filtering',
'SVD Approx. - Movies (3+ ratings)', round(rmse.svdApprox,5))
```

6. Model-Based Collaborative Filtering

Model 12 : SVD with Approximation (contd.)

- RMSE is similar to what we achieved in Slope One.

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242
11	Model Based Collaborative Filtering	Slope One	0.83247
12	Model Based Collaborative Filtering	SVD Approx. - Movies (3+ ratings)	0.88233

6. Model-Based Collaborative Filtering

Model 13 : Matrix Factorization

- Steps involved:
- 1. Convert the data in to the format that can be accepted by recosystem package
- 2. Build the model object
- 3. Fine tune the model for best model parameters
- 4. Train the model
- 5. Run the model on test data to predict the ratings
- 6. Calculate the RMSE on test data by comparing with actual

```
# 1. Convert the data in to the format that can be accepted by recosystem package
train_data <-  with(edxS, data_memory(user_index = userId, item_index = movieId, rating = rating))
test_data  <-  with(validationS,  data_memory(user_index = userId, item_index = movieId, rating = rating))

# 2. Build the model object
model.MF <-  recosystem::Reco()

# 3. Fine tune the model for best model parameters.
if(as.numeric(R.Version()$major)==3 & as.numeric(R.Version()$minor) >5)
  set.seed(1, sample.kind="Rounding") else set.seed(1)

opts <- model.MF$tune(train_data, opts = list(dim = c(300), lrate = c(0.01),costp_l2 = c(0.01),costq_l2 = c(0.1),
                                       nthread  = 4, niter = 10))

# 4. Train the model
model.MF$train(train_data, opts = c(opts$min, nthread = 4, niter = 100))

# 5. Run the model on test data to predict the ratings
predicted.ratings.MF <-  model.MF$predict(test_data, out_memory())
rmse.MF <-  rmse(validationS$rating, predicted.ratings.MF)
rmse.results[nrow(rmse.results)+1,] <- c(13, 'Model Based Collaborative Filtering',
'Matrix Factorization', round(rmse.MF,5))
```

6. Model-Based Collaborative Filtering

Model 13 : Matrix Factorization

- This is the best performance till now on the complete 100K dataset

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242
11	Model Based Collaborative Filtering	Slope One	0.83247
12	Model Based Collaborative Filtering	SVD Approx. - Movies (3+ ratings)	0.88233
13	Model Based Collaborative Filtering	Matrix Factorization	0.85363

7. Ensemble Methods

Model 14 : Ensemble Methods - Data Preparation

- 1. Select variables for model building
- 2. Create derived variables

```
# 1. Selecting variables for model building. Removing timestamp, title, genres, (no genres listed), date
edxS.selected <- edxS %>% select(-c("timestamp","title","genres", "(no genres listed)", "date" ))
validationS.selected <- validationS %>% select(-c("timestamp","title","genres", "(no genres listed)", "date" ))

# 2. Creating derived variables
# Number of movies rated by every user
edxS.selected <- edxS.selected %>% group_by(userId) %>%  mutate(n.movies_byUser = n())
validationS.selected <- validationS.selected %>% group_by(userId) %>%  mutate(n.movies_byUser = n())

# Number of users ratings by every movie
edxS.selected <- edxS.selected %>% group_by(movieId) %>% mutate(n.users_bymovie = n())
validationS.selected <- validationS.selected %>% group_by(movieId) %>% mutate(n.users_bymovie = n())

# Count of distinct genres present in a movie
edxS.selected$Cnt.Genres.in.Movie <- edxS[,7:26] %>% rowSums()
validationS.selected$Cnt.Genres.in.Movie <- validationS[,7:26] %>% rowSums()

# Count of distinct genres watched by user
x <-  aggregate(edxS[,7:26],by = list(userId=edxS$userId),
                function(genre) if(any(genre==T)) return(1) else return(0))
x$Cnt.Genres.by.Users <- rowSums(x[,-1])
edxS.selected <- edxS.selected %>% left_join(x[,c('userId','Cnt.Genres.by.Users')], by = 'userId')

x <-  aggregate(validationS[,7:26],by = list(userId=validationS$userId),
                function(genre) if(any(genre==T)) return(1) else return(0))
x$Cnt.Genres.by.Users <- rowSums(x[,-1])
validationS.selected <- validationS.selected %>% left_join(x[,c('userId','Cnt.Genres.by.Users')], by = 'userId')
```

7. Ensemble Methods

Model 14 : Ensemble Methods - Data Preparation (cont.)

- 2. Creating derived variables continued
- 3. Converting userId, movieId & date variables to factors

```
# Count of movies watched by users in every genre
x <-  aggregate(edxS[,7:25],by = list(userId=edxS$userId),sum)
colnames(x)[-1] <- paste0('Cnt.',colnames(x)[-1])
edxS.selected <- edxS.selected %>% left_join(x, by = 'userId')

x <-  aggregate(validationS[,7:25],by = list(userId=validationS$userId),sum)
colnames(x)[-1] <- paste0('Cnt.',colnames(x)[-1])
validationS.selected <- validationS.selected %>% left_join(x, by = 'userId')

# 3. Converting userId, movieId & date variables to factors
edxS.selected <- edxS.selected %>% mutate(userId = as.factor(userId),
                                          movieId = as.factor(movieId),
                                          yr = as.factor(yr),
                                          mnth = as.factor(mnth),
                                          dt = as.factor(dt),
                                          day = as.factor(day),
                                          hr = as.factor(hr))

validationS.selected <- validationS.selected %>% mutate(userId = as.factor(userId),
                                                        movieId = as.factor(movieId),
                                                        yr = as.factor(yr),
                                                        mnth = as.factor(mnth),
                                                        dt = as.factor(dt),
                                                        day = as.factor(day),
                                                        hr = as.factor(hr))
```

7. Ensemble Methods

Model 14 : Ensemble Methods - Building Gradient Boosting Model

- H2O instance is used to run the ensemble models
- Only the best model iteration is kept here. Refer to the code for all iterations

```
# 1. Initializing H2O instance
h2o.init(nthreads=-1, max_mem_size = ”10G”)  # initializes with all available threads and 10Gb memory
h2o.removeAll() # frees up the memory

# 2. Selected Model iteration
model.gbm3 <- h2o.gbm(y = ”rating”, training_frame = as.h2o(edxS.selected), ntrees = 50,
max_depth=10, nfolds = 3, seed = 1,keep_cross_validation_predictions = TRUE, fold_assignment = ”Random”)

# 3. Predicting the ratings on validation data
predicted.ratings <- h2o.predict(model.gbm3,as.h2o(validationS.selected))
rmse.gbm3 <- rmse(as.h2o(validationS.selected$rating), predicted.ratings)
rmse.results[nrow(rmse.results)+1,] <- c(14, 'Ensemble Model', 'GBM only', round(rmse.gbm3,5))
```

- Inspite of putting so many variables, we get RMSE of 0.93777. We will see if this improves with Random Forest

```
view.rmse()
```

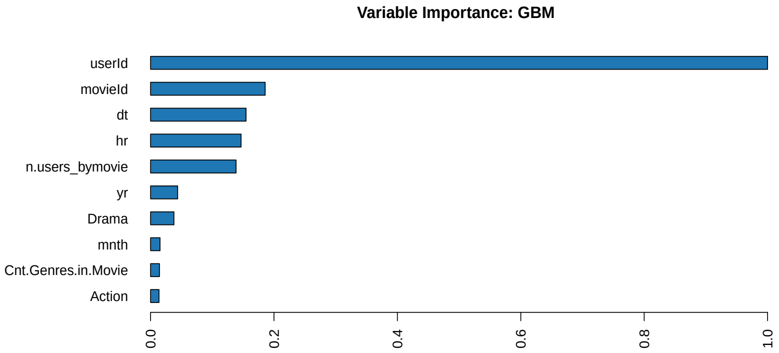
SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242
11	Model Based Collaborative Filtering	Slope One	0.83247
12	Model Based Collaborative Filtering	SVD Approx. - Movies (3+ ratings)	0.88233
13	Model Based Collaborative Filtering	Matrix Factorization	0.85363
14	Ensemble Model	GBM only	0.93777

7. Ensemble Methods

Model 14 : Ensemble Methods - GBM variable importance

- Before we move to the next model, lets see what factors impact the ratings the most
- While the model has not shown great performance, but it can tells us what are the key variables that impact the ratings
- While we would have guessed that users plays the most important role in a movie rating prediction, it is interesting to see that date of rating, hour of the day when the rating was given, number of users who have rated the movie and certain genre of movie like - drama & action plays a key role in determining the ratings. The date of rating might indicate that variables like - time lapsed since movie release might play an important role.

```
h2o.varimp_plot(model.gbm3)
```



Plot of top 10 most important variables, sorted in the order of relative variable importance. The most important variable is given a value 1.

7. Ensemble Methods

Model 15 : Ensemble Methods - Random Forest Model

Important note - In order to ensure that later we can combine both Random Forest & GBM models, it is key that we keep the seed & number of folds the same and keep cross validation predictions in both types of model iterations.

```
# 1. Selected Model iteration
model.rf1 <- h2o.randomForest(y = ”rating”, training_frame = as.h2o(edxS.selected), ntrees = 50,
max_depth=20, nfolds = 3, seed = 1,keep_cross_validation_predictions = TRUE, fold_assignment = ”Random”)

# 2. Predicting the ratings on validation data
predicted.ratings <- h2o.predict(model.rf1,as.h2o(validationS.selected))
rmse.rf1 <- rmse(as.h2o(validationS.selected$rating), predicted.ratings)
rmse.results[nrow(rmse.results)+1,] <- c(15, ’Ensemble Model’, ’Random Forest only’, round(rmse.rf1,5))
```

- Random Forest model too gives a similar performance as GBM
- Next we will build the ensemble model of Random Forest & GBM to see if there is an improvement in performance

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242
11	Model Based Collaborative Filtering	Slope One	0.83247
12	Model Based Collaborative Filtering	SVD Approx. - Movies (3+ ratings)	0.88233
13	Model Based Collaborative Filtering	Matrix Factorization	0.85363
14	Ensemble Model	GBM only	0.93777
15	Ensemble Model	Random Forest only	0.95412

7. Ensemble Methods

Model 16 : Ensemble Methods - Ensemble Model

We will build the ensemble model using the GBM & Random Forest model

```
# 1. Building the ensemble model
model.ensemble <- h2o.stackedEnsemble(y = ”rating”,training_frame = as.h2o(edxS.selected),
                                      base_models = list(model.gbm3@model_id, model.rf1@model_id))

# 2. Predicting the ratings on validation data
pred.ratings.ensemble <- h2o.predict(model.ensemble,as.h2o(validationS.selected))
rmse.ensemble <- RMSE(pred.ratings.ensemble, as.h2o(validationS.selected$rating))
rmse.results[nrow(rmse.results)+1,] <- c(16, ’Ensemble Model’, ’GBM + Random Forest’, round(rmse.ensemble,5))
```

- The final RMSE of ensemble models is 0.9174705
- This is the last of all the models that we wanted to try on the 100K dataset

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
1	Baseline model	Random rating with 0.1 prob	1.94563
2	Baseline model	Random rating with existing prob	1.48903
3	Baseline model	Mean rating	1.05498
4	Linear model	Movie effect only	0.97328
5	Linear model	Movie & user effect	0.88639
6	Linear model	Movie, user & genre effect	0.88152
7	Linear model	Movie & user effect with regularization	0.86788
8	Linear model	Movie, user & genre effect with reg.	0.85533
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242
11	Model Based Collaborative Filtering	Slope One	0.83247
12	Model Based Collaborative Filtering	SVD Approx. - Movies (3+ ratings)	0.88233
13	Model Based Collaborative Filtering	Matrix Factorization	0.85363
14	Ensemble Model	GBM only	0.93777
15	Ensemble Model	Random Forest only	0.95412
16	Ensemble Model	GBM + Random Forest	0.91747

8. Model Selection on 100K dataset

- We can see that Model Based Collaborative Filtering techniques have given the best RMSE
- Slope One method has best RMSE, however it was difficult to run on 10Mn dataset using local machine with limited RAM.
- Hence we will select Matrix Factorization as the final modeling technique which we will run on the larger dataset (10 Mn ratings). The recosystem package supports high performance multi-core parallel computing. Most importantly unlike most other R packages for modeling that store the whole dataset and model object in memory, recosystem can significantly reduce memory use, by storing the constructed model on hard disk, and output result can also be directly written into a file rather than be kept in memory
- It is also interesting to see that linear model that captured the effect of movie, user & genre with regularization has also performed than better than the target RMSE

```
rmse.results[nrow(rmse.results)+1,] <- c(17, 'Target RMSE', 'Target RMSE', 0.86490)
rmse.results <- rmse.results %>% arrange(RMSE)

view.rmse()
```

SNo	ModelType	Algorithm	RMSE
11	Model Based Collaborative Filtering	Slope One	0.83247
13	Model Based Collaborative Filtering	Matrix Factorization	0.85363
8	Linear model	Movie, user & genre effect with reg.	0.85533
17	Target RMSE	Target RMSE	0.8649
7	Linear model	Movie & user effect with regularization	0.86788
6	Linear model	Movie, user & genre effect	0.88152
12	Model Based Collaborative Filtering	SVD Approx. - Movies (3+ ratings)	0.88233
5	Linear model	Movie & user effect	0.88639
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
16	Ensemble Model	GBM + Random Forest	0.91747
14	Ensemble Model	GBM only	0.93777
15	Ensemble Model	Random Forest only	0.95412
4	Linear model	Movie effect only	0.97328
3	Baseline model	Mean rating	1.05498
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242
2	Baseline model	Random rating with existing prob	1.48903
1	Baseline model	Random rating with 0.1 prob	1.94563

Outline

Model Training & Validation on 10Mn Movie Dataset using selected model

Model 17 : Matrix Factorization modeling on 10Mn Dataset

```r
# 1. Convert the data in to the format that can be accepted by recosystem package
train_data <-  with(edxB, data_memory(user_index = userId, item_index = movieId, rating = rating))
test_data  <-  with(validationB,  data_memory(user_index = userId, item_index = movieId, rating = rating))

# 2. Build the model object
model.MF.10Mn <-  recosystem::Reco()

# 3. Fine tune the model for best model parameters.
# Checking for R Version and using set.seed function appropriately
if(as.numeric(R.Version()$major)==3 & as.numeric(R.Version()$minor) >5)
  set.seed(1, sample.kind="Rounding") else set.seed(1)

opts <- model.MF.10Mn$tune(train_data, opts = list(dim = c(10,20,30), lrate = c(0.1),
                                       costp_l1 = 0, costq_l1 = 0,
                                       costp_l2 = 0.01,costq_l2 = 0.1,
                                       nthread  = 4, niter = 10))

# 4. Train the model
model.MF.10Mn$train(train_data, opts = c(opts$min, nthread = 4, niter = 100))

# 5. Run the model on test data to predict the ratings
predicted.ratings.MF.10Mn <-  model.MF.10Mn$predict(test_data, out_memory())
rmse.MF.10Mn <-  rmse(validationB$rating, predicted.ratings.MF.10Mn)
rmse.results[nrow(rmse.results)+1,] <- c(18, '10Mn dataset', 'Matrix Factorization', round(rmse.MF.10Mn,5))
```

Model Training & Validation on 10Mn Movie Dataset using selected model

Model 17 : Matrix Factorization modeling on 10Mn Dataset (cont)

- The RMSE on the 10Mn dataset is better than the target
- The selected model of Matrix Factorization has worked quite well on 10Mn Dataset

```
view.rmse()
```

SNo	ModelType	Algorithm	RMSE
18	10Mn dataset	Matrix Factorization	0.78238
11	Model Based Collaborative Filtering	Slope One	0.83247
13	Model Based Collaborative Filtering	Matrix Factorization	0.85363
8	Linear model	Movie, user & genre effect with reg.	0.85533
17	Target RMSE	Target RMSE	0.8649
7	Linear model	Movie & user effect with regularization	0.86788
6	Linear model	Movie, user & genre effect	0.88152
12	Model Based Collaborative Filtering	SVD Approx. - Movies (3+ ratings)	0.88233
5	Linear model	Movie & user effect	0.88639
9	Memory Based Collaborative Filtering	UBCF - Movies (50+ ratings) & Users (20+ ratings)	0.90818
16	Ensemble Model	GBM + Random Forest	0.91747
14	Ensemble Model	GBM only	0.93777
15	Ensemble Model	Random Forest only	0.95412
4	Linear model	Movie effect only	0.97328
3	Baseline model	Mean rating	1.05498
10	Memory Based Collaborative Filtering	IBCF - Movies (50+ ratings) & Users (20+ ratings)	1.26242
2	Baseline model	Random rating with existing prob	1.48903
1	Baseline model	Random rating with 0.1 prob	1.94563

Outline

Key learnings from the entire exercise

- We tried broadly 5 types of models - (1) Linear Models, (2) Linear Models with Regularization (3) Memory Based Collaborative Filtering (4) Model Based Collaborative Filtering & (5) Ensemble Models
- We realized that Model Based Collaborative Filtering works the best. Both Slope One and Matrix Factorization performed better than target RMSE.
- Genre played a significant role in determining the ratings. It was interesting to see how a simpler linear model with user, movie & genre effects with regularization performed better than target RMSE.
- We also realized that Memory Based Collaborative Models, although easy to implement, is unable to predict for new users or movies (where we have less number of observations) and works only for popular movies or users.
- Ensemble models highlighted that traditional techniques might not work so well for recommendation engines and makes us realize why we need specialized techniques like Collaborative Filtering. However it helped us realize the aspects that impacts the ratings most. Users preferences define the ratings more than anything else. Even the number of movies rated by users matter. Genres play any important role, especially the popular ones like - Drama & Action. Variables extracted from timestamp like date & hour also plays a role. Date indicates that new variables like - Days lapsed since movie release can also play a role in rating prediction.
- We chose the Matrix Factorization (MF) technique to predict on 10Mn dataset. Slope One required memory beyond the local machine capacity. Thanks to recosystem package which utilizes parallel computing & disk storage, we could easily see the implementation of MF technique on 10Mn dataset. The final performance of MF technique on 10Mn dataset was way better than its performance on 100K dataset. It is possible that even other models perform better on the larger dataset.

Outline

Action points to improve current recommendation engine

- Implement currently used techniques with genre & time variables - Currently all techniques used in Collaborative Filtering (both Memory Based & Model Based) consider only userID & movieID. It will be interesting to figure out how can they be used alongwith other variables and measure the model performance with Genre & Time variables.
- Create new variables - Enrich the models by adding new derived variables like - Days since movie release. We must add other aspects of a movie like - Actors & Director from other data sources to improve model performance.
- Try other techniques - We should try other advanced techniques like Deep Learning and see the model performance.
- Try mix of techniques - We know that memory based collaborative filtering works best with popular movies and users. Can we think of finding a way to apply both memory based & model based collaborative filtering together to get the best of both modeling techniques.

References

I would like to thank each of them listed below. Without the learnings drawn from here, this work would not be possible.

- EDX Data Science Program by Rafael Irizarry
- Recosystem Github
- Article on recommendation engine techniques by James Le
- Capstone Project by Louis Mono
- Movie recommendation work on Movielens data using R by Taras Hnot
- Slope One & SVD Approximation code
- Capstone Project by Victor Ivamoto
- Movie recommendation work on Movielens data using R by Kumudini Bhave