Harvard Data Science Professional Capstone

Predicting user ratings from movielens dataset

philk1337

2019-01-31

Contents

This project uses RSME as the grading metric.
Although i began with accuracy in mind and therefore also implemented accuracy as a metric (not to be graded).

1 Management Summary

This report summarizes the approach of predicting movie ratings by user based on a huge set of data from a even bigger dataset from movielens (www.movielens.org). The data to use consists of two datasets - one training dataset and one validation set. Latter doesn't include the rating column and the created system has to tested with this at the end.

The aim of the project is to create a movie recommendation system using the features in the MovieLens dataset to predict the potential movie rating a special user gives for a movie. We had to train a machine learning algorithm using the inputs in one subset to predict movie ratings in the validation set.

In this approach i tried to go for a human-centered way as well as a data-supported. Thinking from the user perspective one has usually films and genres he likes most. So i expect that besides user- and movie-specific preferences there also will be genre-specific preferences. These are influencing the ratings of a user.

I have tested different models like Generalized Linear Model (GLM), Naïve Bayes, Neural Network, Random Forrest and more and ended creating a model After an exploratory analysis, the variables movieId, userId and genre have been selected as potential predictors. Best results for the RSME were reached by using a simple modeling approach.

The quality of the final model is evaluated on the RMSE, with a target value of RMSE <= 0.87750. Also the results were summarized step by step as the model gets more complex.

The final model reached an **RMSE** of around **0.8434** and is therefore ok in this context.

A possible next step to improve results could be to use Matrix Factorization to also include the fact that some groups of movies, users and genres may be similar and could be used for prediction.

2 Analysis and Methods

Before starting with modeling i did some data exploration and visualization to find the right direction of a first approach. As the data is already prepared for machine learning there were no missing values and it is in tidy format already. Only the genres column has to be split as the genres of a movie were put in one column as a string only seperated by a pipe-character.

After exploration it was clear, that there are clearly a **movie-effect** and a **user-effect**. There are blockbuster-movies that are rated much more frequent and there are kind of power-users with significantly more ratings then the average user.

2.1 Explorative Analysis

First we load the dataset and take a quick look:

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Lets check the dimensions of the dataset and look, if there are missing values. We can see this table is in tidy format with 9000055 rows and 6 columns containing 0 missing values.

If there would have been empty values we could probably have imputed them using **KNN** based on similar values:

```
preProcValues <- preProcess(edx, method = c("knnImpute","center","scale"))

library('RANN')
train_processed <- predict(preProcValues, edx)
sum(is.na(train_processed))
```

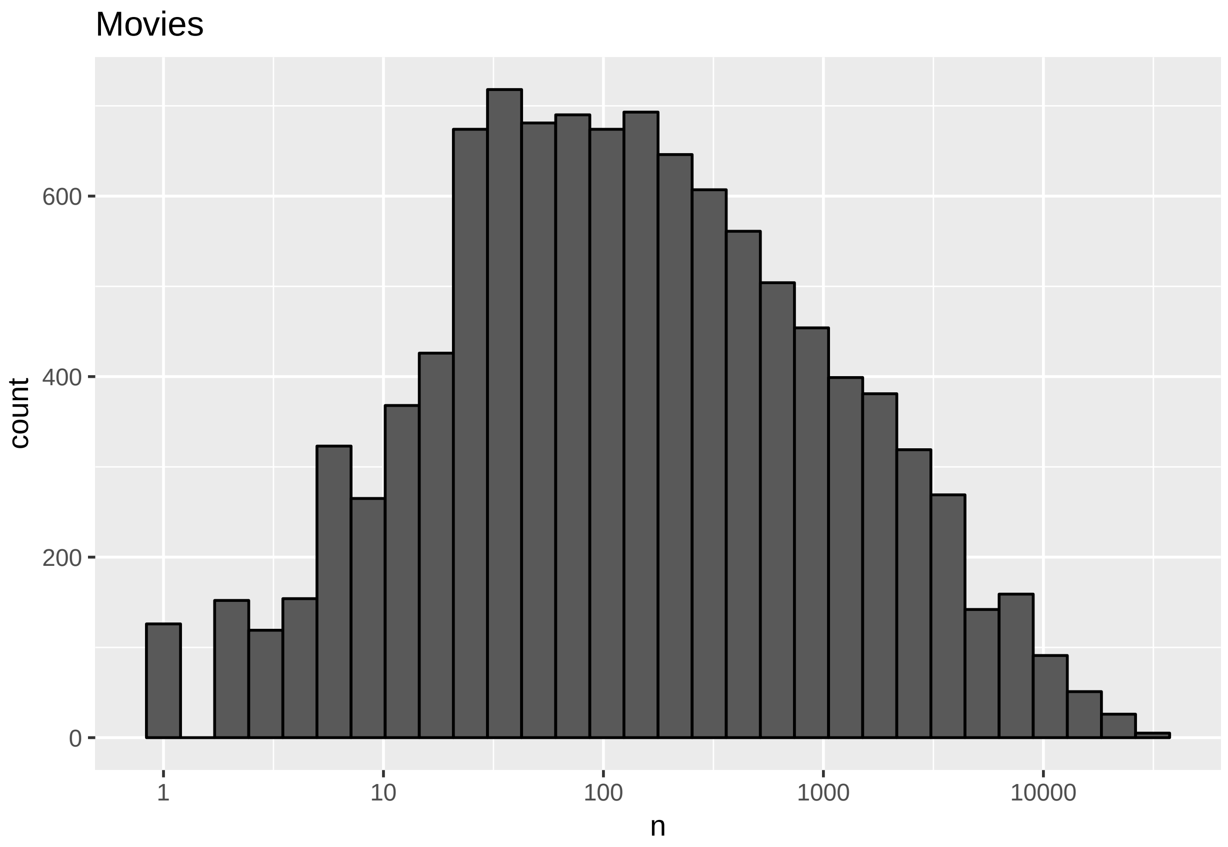
Each row represents a rating given by one user to one movie.

Now lets have a look at how many distinct users and movies are in the dataset:

distinct_users	distinct_movies
69878	10677

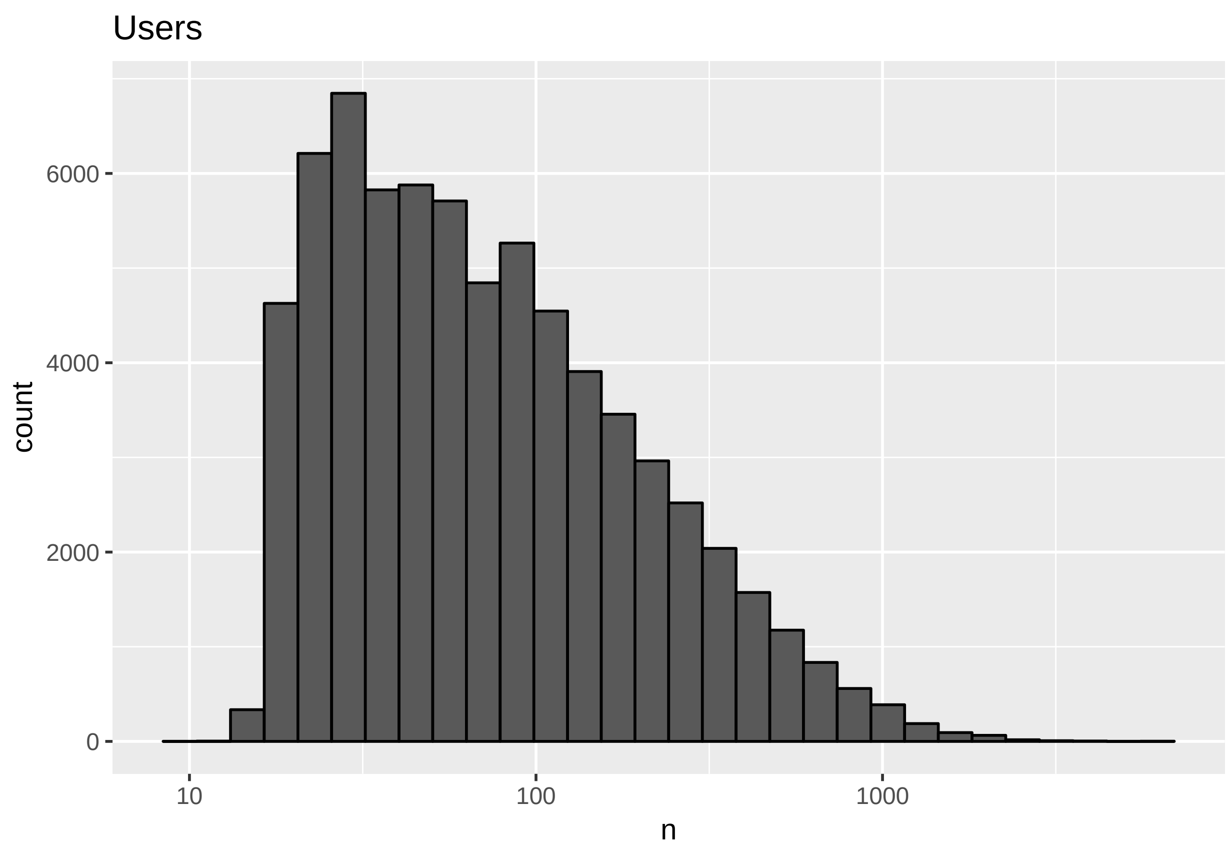
Interesting fact: There are more users than movies, so most movies are rated more than once. Now let's look at some of the general properties of the data to better understand it.

The first thing we notice is that some movies get rated more than others. Here is the distribution:



This is not surpring given that there are blockbuster movies watched by millions and more special movies in art, independent movies watched by just a few users.

Then we get to know that some users are more active than others at rating movies:



And here are the possible values for "rating":

x
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

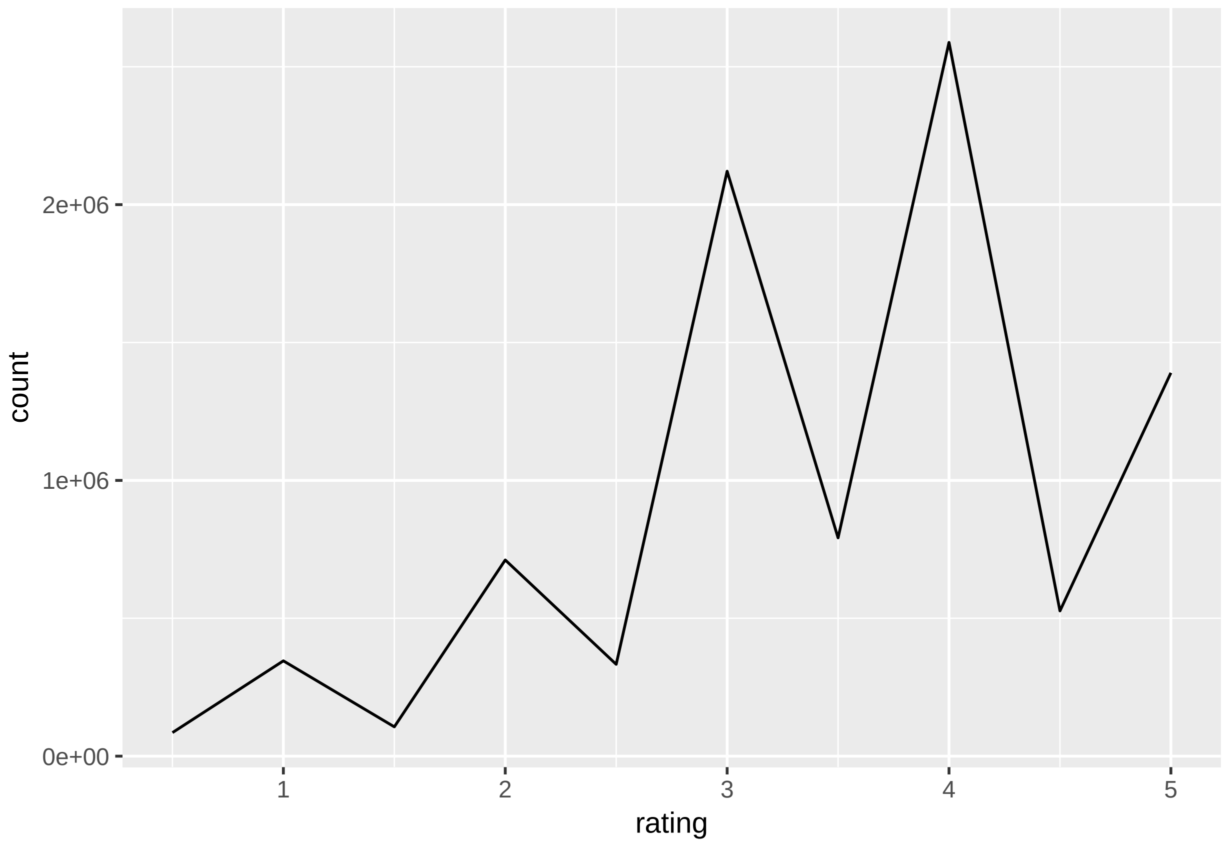
Sum of given ratings by category:

rating	count
4.0	2588430
3.0	2121240

rating	count
5.0	1390114
3.5	791624
2.0	711422
4.5	526736
1.0	345679
2.5	333010
1.5	106426
0.5	85374

As we can see the ratings 4, 3 and 5 are given much more than the other ratings.

Are there more full star ratings than half star ratings (i.e. more 4 than 4.5)?



As we can see there are definitely **more full star ratings than half star ratings**.

2.1.1 Split genre column into seperate values

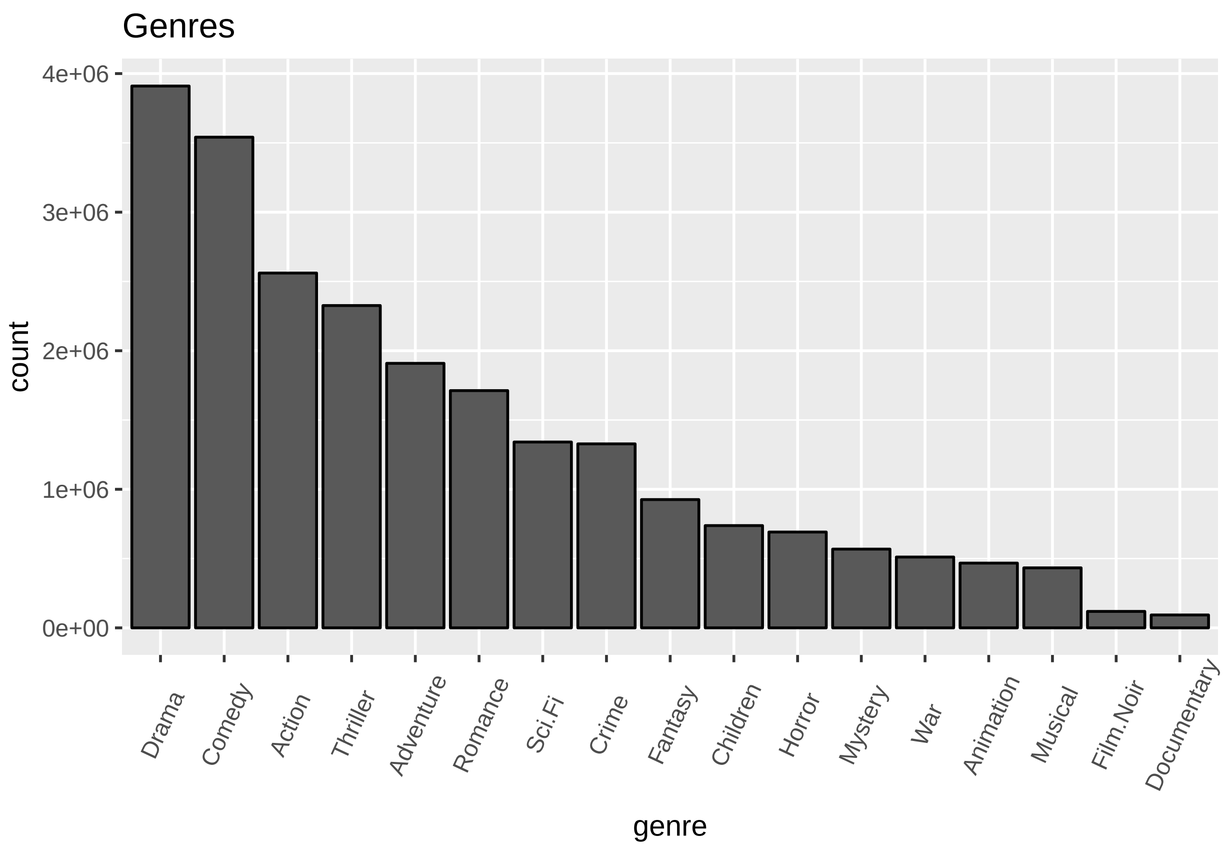
To including the genres in analysis and models we now generate a tidy dataset by splitting the genres colum into separate colums per genre (one-hot) and analyze the genres:

First we save the genres in an array to then create columns per specific genre:

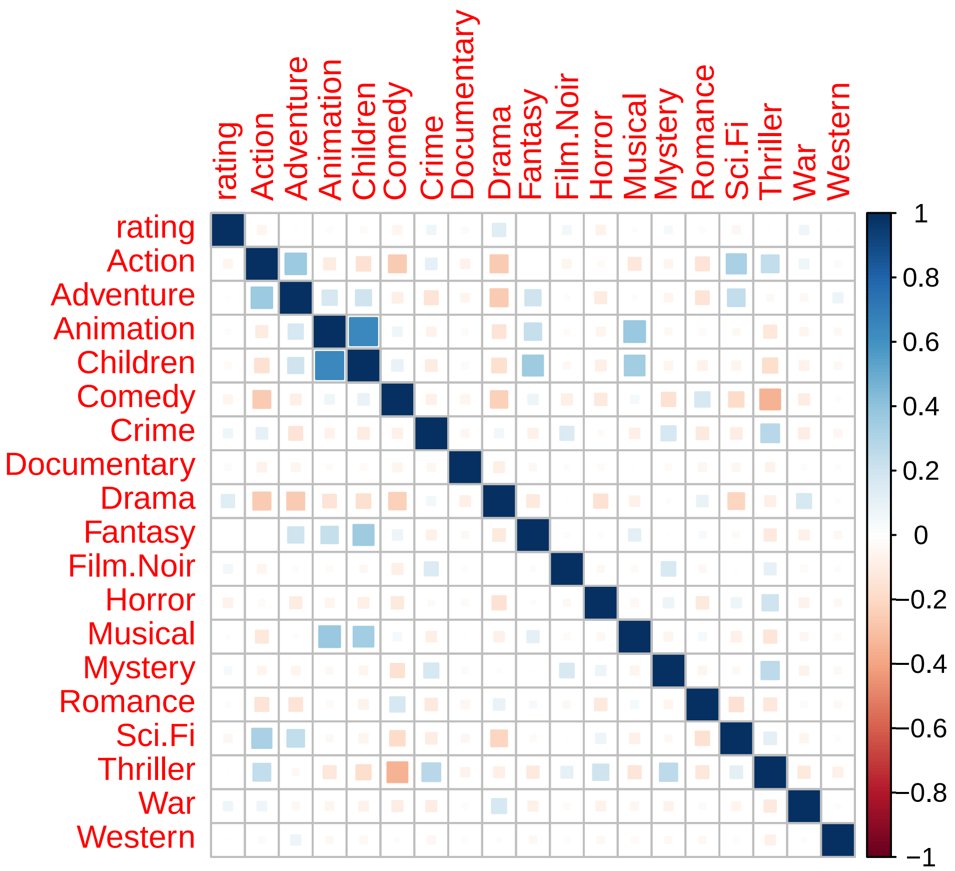
```
# Save genres as array
genres <- c("Action", "Adventure", "Animation", "Children", "Comedy",
    "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir", "Horror",
    "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller", "War", "Western")

# create genre-specific column-copys
for(i in genres) {
  edx_genres[i] <- i
}

# Create new features from genres and remove timestamp and title
edx_genres <- edx_genres %>%
  rowwise() %>% mutate(Action = as.numeric(grepl(Action,genres))) %>%
  rowwise() %>% mutate(Adventure = as.numeric(grepl(Adventure,genres))) %>%
  rowwise() %>% mutate(Animation = as.numeric(grepl(Animation,genres))) %>%
  rowwise() %>% mutate(Children = as.numeric(grepl(Children,genres))) %>%
  rowwise() %>% mutate(Comedy = as.numeric(grepl(Comedy,genres))) %>%
  rowwise() %>% mutate(Crime = as.numeric(grepl(Crime,genres))) %>%
  rowwise() %>% mutate(Documentary = as.numeric(grepl(Documentary,genres))) %>%
  rowwise() %>% mutate(Drama = as.numeric(grepl(Drama,genres))) %>%
  rowwise() %>% mutate(Fantasy = as.numeric(grepl(Fantasy,genres))) %>%
  rowwise() %>% mutate("Film-Noir" = as.numeric(grepl("Film-Noir",genres))) %>%
  rowwise() %>% mutate(Horror = as.numeric(grepl(Horror,genres))) %>%
  rowwise() %>% mutate(Musical = as.numeric(grepl(Musical,genres))) %>%
  rowwise() %>% mutate(Mystery = as.numeric(grepl(Mystery,genres))) %>%
  rowwise() %>% mutate(Romance = as.numeric(grepl(Romance,genres))) %>%
  rowwise() %>% mutate("Sci-Fi" = as.numeric(grepl("Sci-Fi",genres))) %>%
  rowwise() %>% mutate(Thriller = as.numeric(grepl(Thriller,genres))) %>%
  rowwise() %>% mutate(War = as.numeric(grepl(War,genres))) %>%
  rowwise() %>% mutate(Western = as.numeric(grepl(Western,genres))) %>%
  select(-title, -timestamp, -genres)

# convert to factors
```

```
edx_genres$userId <- as.factor(edx_genres$userId)
edx_genres$movieId <- as.factor(edx_genres$movieId)
edx_genres$rating <- as.factor(edx_genres$rating)
```

2.1.2 Analyze Genres



2.1.3 Correlations between genres



As we can clearly see there seems to be a strong correlation between the genres **Chilren** and **Animation**. This finding will later be the basis for including an additional step in the final recommender model and also trying to use the genres as prediction feature.

2.2 Methods

In a first modeling round i tried to use different algorithms for **multiclass classification** like Random Forest with caret. After many hours and days of waiting for the model-calculation on my macbook i decided to abort and thought about it again.

Now i started to use a single Decision tree followed by using the Ranger-package as it cumputes faster - but also with no success. Still the calculation lacks memory.

In a new approach i selected a random smaller representative test set due to massive ressource limitations of my mac. Effect of this approach was bad accuracy cause of too small data.

In a further approach i used the **recommenderlab** package to create a **user based recommender** as well as an **item-based recommender**. Calculating the necessary matrix and creating a model based on the full dataset worked, but predicting did not due to - again - ressource limitations.

2.3 Create Train- and Test-Sets from edx dataset

```
split <- createDataPartition(edx$rating, p = 0.8, list = FALSE)
train <- edx[split,]
test <- edx[-split,]
```

2.4 Create sample from data

One try was to reduce the size of thr dataset for creating the models by taking stratified samples out of the basic edx dataset. Using this samle-training-set the calculation of the models and also the predictions got much more fast, but also the accuracy/RMSE went down a lot.

```
train_s <- sample_n(train, 10000, replace = FALSE)
#table(train_s$movieId)

test_s <- c()
test_s <- test %>%
  semi_join(train_s, by = "movieId") %>%
  semi_join(train_s, by = "userId")

# select matching movieIds for validation set
match <- as.data.frame(test$movieId) == train_s$movieId
test_s <- test[match,]
```

2.5 Final: Recommendation System

In a first step i splitted the edx set containing the data with target column (rating) into two sets: train and test. So by creating the model using the train set i can compare the result of the prediction with an unseen dataset.

```
set.seed(755)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

To make sure to not include users and movies in the test set that do not appear in the training set, i remove these entries using the semi_join function:

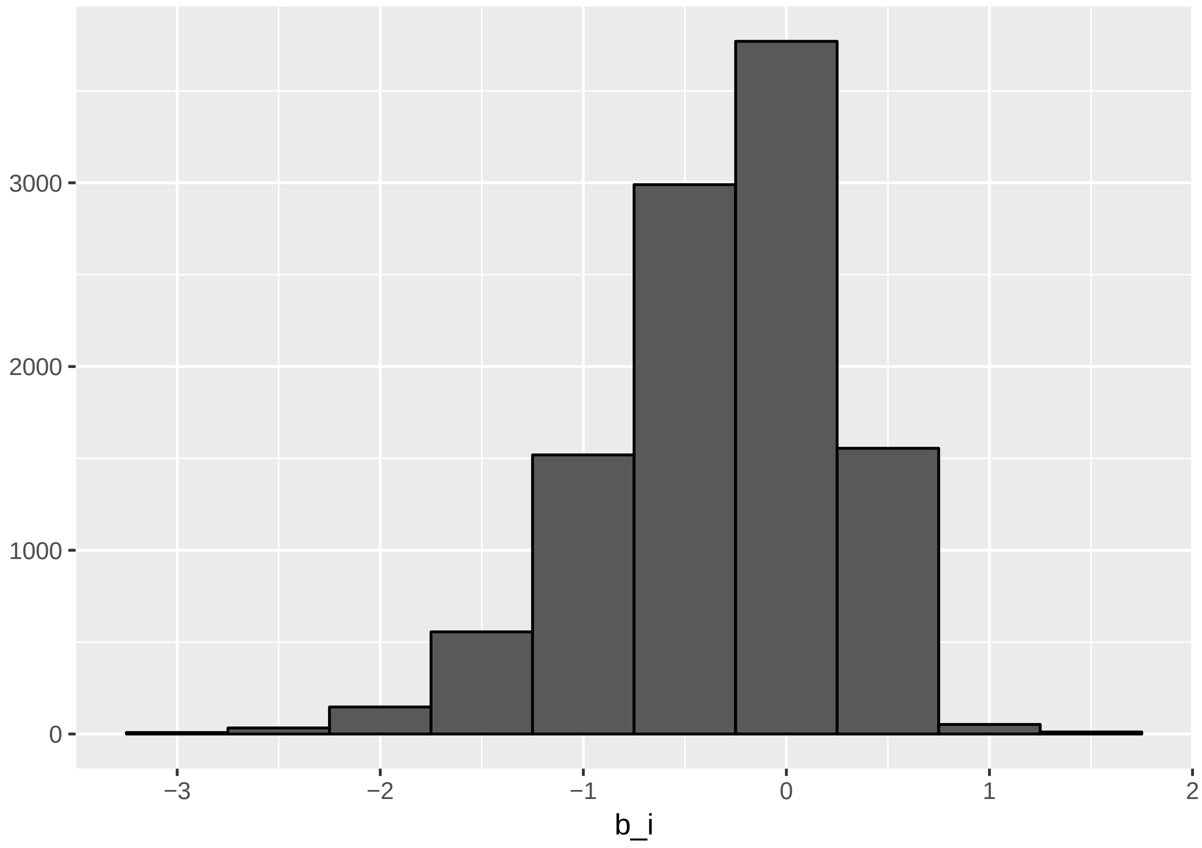
Now i construct a function to calculate the RMSE and create a first model only using mu hat of the ratings for a baseline prediction:

We know that the estimate that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings. We start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. So what number should this prediction be? We can use a model based approach. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

If we predict all unknown ratings with mu_hat, we obtain the following RMSE and accuracy:

method	RMSE
Just the average	1.060561

We know from our analysis that some movies are just generally rated higher than others. So our intuition that different movies are rated differently is confirmed by data. We can augment our previous model by calculating the least squares estimates of the **movie effect**:

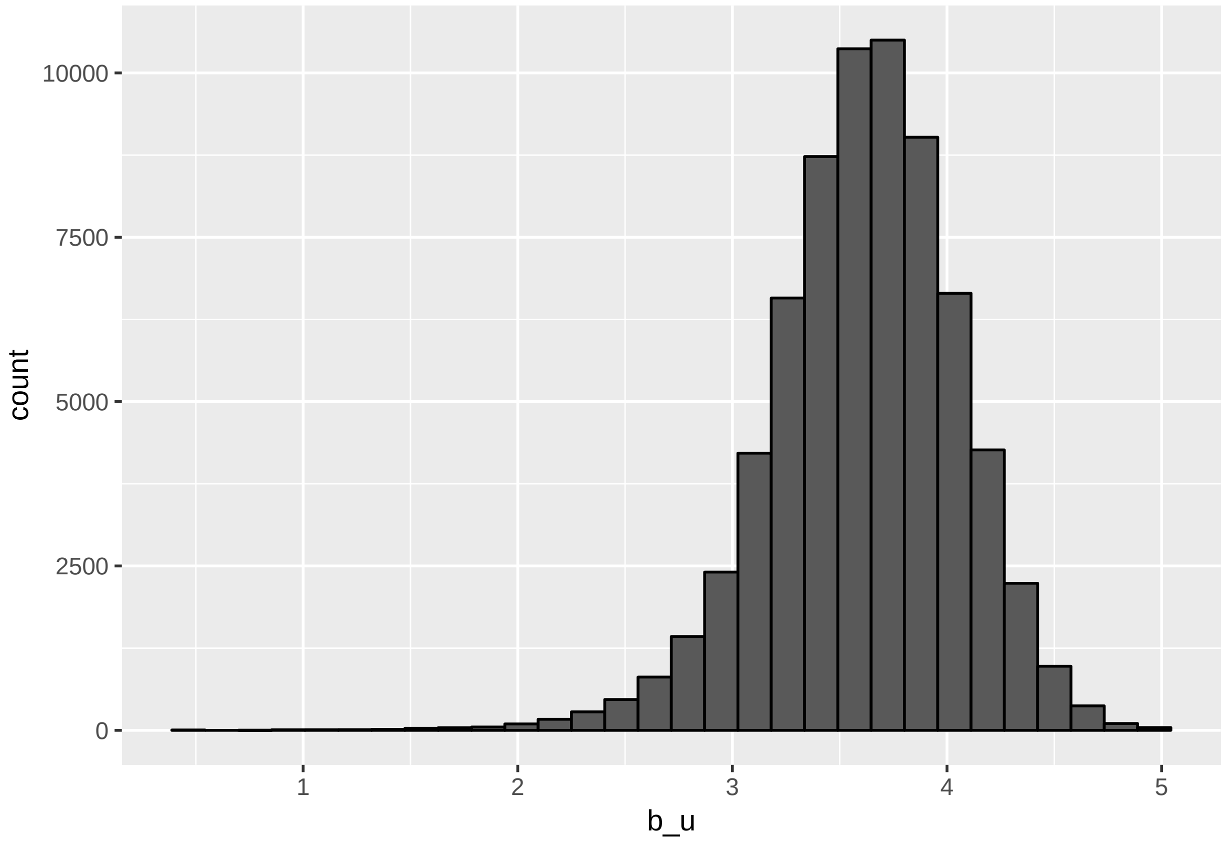


method	RMSE
Just the average	1.0605613

method	RMSE
Movie Effect Model	0.9439868

As we can see there is already an improvement over the naive mu_hat approach.

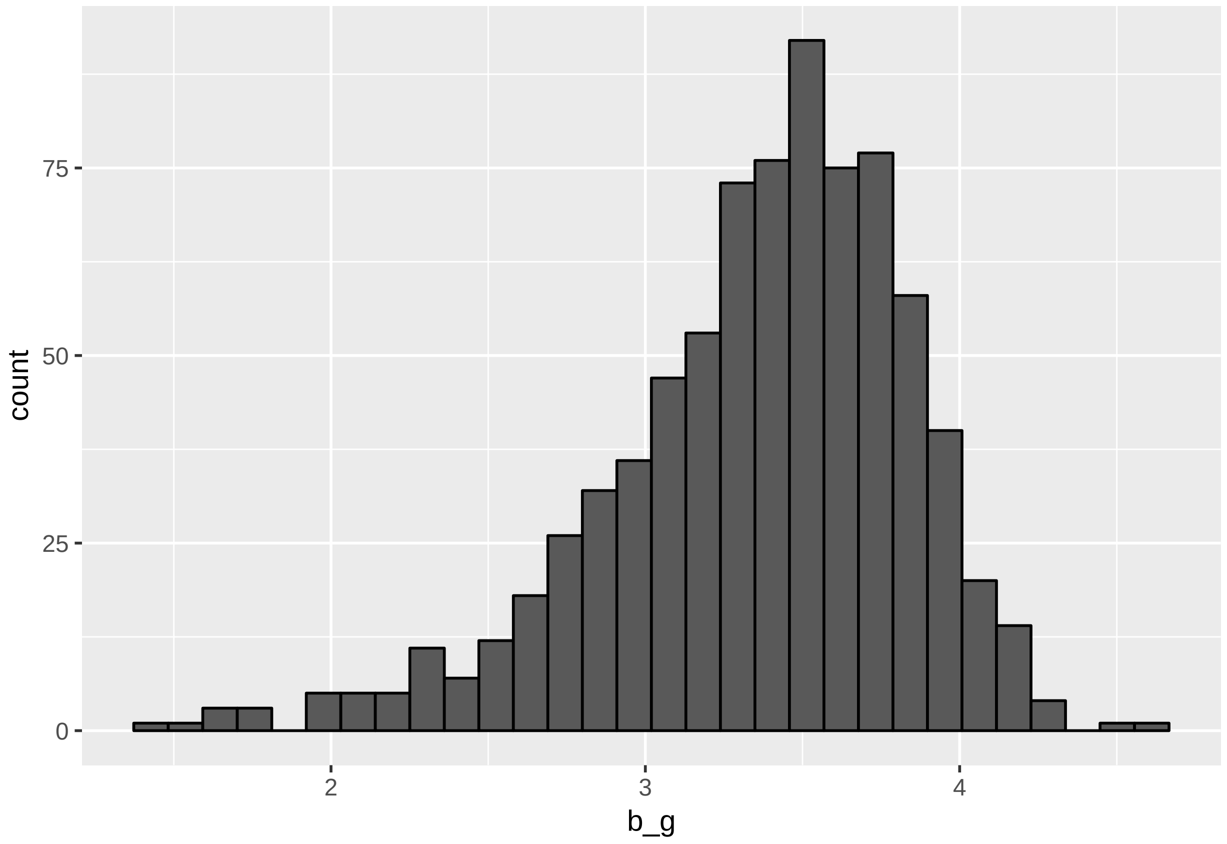
In a next step we try to calculate the **user effect** to augment our model even further:



method	RMSE
Just the average	1.0605613
Movie Effect Model	0.9439868
Movie + User Effects Model	0.8439865

method	accuracy
Just the average	0.0881810
Moviel Effect Model	0.3256925
Movie + User Effects Model	0.3711333

The RMSE sinks again as well as the accuracy rises a bit. This direction is promising. In an additional step we now try to get even better by including the **genre-effect**:



method	RMSE
Just the average	1.0605613
Movie Effect Model	0.9439868
Movie + User Effects Model	0.8439865
Movie + User Effects + Genres Model	0.8434070

method	accuracy
Just the average	0.0881810
Moviel Effect Model	0.3256925
Movie + User Effects Model	0.3711333
Movie + User Effects + Genre Model	0.3717072

As we can see the accuracy as well as the RSME get slightly better. But considering the effort in making the use of genres possible that seems not worth it.

3 Results

After creating and playing around with multiple classification and clustering algorithms in different package variants there was clear, that i don't have enough power for using most ml algorithms on this big data set. I had to realize that i have to choose another way for this problem.

So i created a simple recommender model using only the mean and developed this further also implementing the movie-effect as well as the user-effect seen in the data. In a last step i added a genre-effect to the model without that much impact on RSME and accuracy.

So finally the model reached an **RMSE** of around **0.8434** and an **accuracy** of around **37%**.

4 Conclusion

I have tested different models like Generalized Linear Model (GLM), Naïve Bayes, Neural Network, Random Forrest and more and ended creating a model After an exploratory analysis, the variables movieId, userId and genre have been selected as potential predictors. Best results for the RSME were reached by using a simple modeling approach.

The final model reached an **RMSE** of around **0.8434** and is therefore ok in this context.

A possible next step to improve results could be to use Matrix Factorization to also include the fact that some groups of movies, users and genres may be similar and could be used for prediction.

4.1 Learnings

There were some useful learnings i want to share here: despite the huge amount of time i invested in making the genres usable as predictors by one-hotting them there was a really, really small effect on the resulting RMSE. Another thing i learned the hard way using lots of probing time: sometimes one can't use the existing algorithms cause of hardware limitations when confronted with a big sized dataset. So you have to da kind of an approximation and also can get a really good result.

5 APPENDIX

This appendix is not meant to be rated. It should show the different approaches i took aiming at an optimal accuracy (which was the first mentioned metric). It contains some more explanation of the reasons and backgrounds and also code examples.

5.1 Approach: Random Forest and more

In one of the first steps in the modeling process i wanted to test as many different classification algorithms as possible. Seeing more and more the lack of computing power i concentrated using only some of them.

```
# Train multiple models

models <- c("glm", "lda",  "naive_bayes",  "svmLinear",
            "gamboost",  "gamLoess", "qda",
            "knn", "kknn", "loclda", "gam",
            "rf", "ranger",  "wsrf", "Rborist",
            "avNNet", "mlp", "monmlp",
            "adaboost", "gbm",
            "svmRadial", "svmRadialCost", "svmRadialSigma")

fits <- lapply(models, function(model){
  print(model)
  train(rating ~ ., method = model, data = edx)
})

# Train random forest model
library(randomForest)
model_rf <- randomForest(rating ~ ., data = train_s, method="class")
plot(model_rf)

# Test accuracy
confusionMatrix(as.factor(predict(model_rf, newdata = test_rsf)), as.factor(test_rsf$rating



### Train other models

set.seed(1337)
ctrl <- trainControl(method="repeatedcv",repeats = 3)
ctrl <- trainControl(method="none")

model_gbm<-train(as.factor(rating) ~ .,data=train_s, method='gbm', trControl = ctrl)
model_glm<-train(as.factor(rating) ~ movieId + userId, data=train_s, method='glm')
```

```
model_rbor <- train(as.factor(rating) ~ movieId + userId, data=train_s, method='Rborist', pr
 model_rf <- train(as.factor(rating) ~ userId + movieId, data = train_s, methof = "rf", preP

 # Multinomial Logistic Regression Neural Network
 model_nnet <- train(rating ~ ., data = train_s, method = "nnet", MaxNWts =7000, trControl =

 # Naive Bayes


 # CART
 model_rpart <- train(factor(rating) ~ ., data = train_s, method = "rpart", trControl = ctrl

 library(rpart)
 model_rpart <- rpart(rating ~ ., data = edx_genres_2, method="class")

 model_rf <- randomForest(rating ~ ., data = trsf, method="class")

 # Logistic Regression
 model_glm <- train(rating ~ ., data=e_bin, method='glm')

 # Naive Bayes
 x <- edx[,1:2]
 x$userId <- as.factor(x$userId)
 x$movieId <- as.factor(x$movieId)
 y<- as.factor(edx$rating)

 model_nb <- train(x[1:1000000,],y[1:1000000],'nb',trControl=ctrl)

 model_nb

 model_glm
 plot(model_rf)
 model_nnet


 ### Predict & Evaluate

 confusionMatrix(as.factor(predict(model_rf, newdata = test_rsf)), as.factor(test_rsf$rating
 levels(as.factor(test_s$rating))
 levels(as.factor(predict(model_rf, newdata = test_s)))
 table(predict(model_rf, newdata = test_s))



 ### Train KNN model

 library(kknn)
 set.seed(1337)
 ctrl <- trainControl(method="repeatedcv",repeats = 3)
 knn <- train(rating ~ userId, method = "knn", data = edx_small,
 trControl = ctrl, preProcess = c("center","scale"), tuneLength = 20)
```
```

### Visualize trained KNN model

knn
plot(knn)



### Predict Test KNN


knn_pred <- (predict(knn, newdata = validation_small))
confusionMatrix(as.factor(knn_pred), test_edx_small$rating)$overall["Accuracy"]
```

5.2 Approach: Recommenderlab

```
# If not installed, first install following three packages in R
library(recommenderlab)
library(reshape2)
library(ggplot2)
library(tidyverse)

# Read training file along with header
```

```
tr<-read.csv("train_v2.csv",header=TRUE)
set.seed(1337)
tr <- sample_n(edx, 100000, replace = FALSE)
tr <- tr[,1:3]
tr$userId <- as.integer(tr$userId)
tr$movieId <- as.integer(tr$movieId)
tr$rating <- as.integer(tr$rating)


# Just look at first few lines of this file
head(tr)

# change the rating factors in 1 to 10 for easier calculation of matches
tr$rating <- as.factor(tr$rating)
levels(tr$rating) <- seq(1,10)
tr$rating <- as.integer(tr$rating)

# Using acast to convert above data as follows:
#       m1  m2   m3   m4
# u1    3   4    2    5
# u2    1   6    5
# u3    4   4    2    5
g<-acast(tr, userId ~ movieId, value.var = "rating")
# Check the class of g
class(g)
head(g)

# Convert it as a matrix
R <- as(g, "matrix")
class(R)

# Convert R into realRatingMatrix data structure
#   realRatingMatrix is a recommenderlab sparse-matrix like data-structure
r <- as(R, "realRatingMatrix")
r

# view r in other possible ways
as(r, "list")     # A list
as(r, "matrix")   # A sparse matrix
```

```
# I can turn it into data-frame
head(as(r, "data.frame"))

# normalize the rating matrix
r_m <- normalize(r)
r_m
as(r_m, "list")

# Draw an image plot of raw-ratings & normalized ratings
#  A column represents one specific movie and ratings by users
#   are shaded.
#   Note that some items are always rated 'black' by most users
#    while some items are not rated by many users
#     On the other hand a few users always give high ratings
#      as in some cases a series of black dots cut across items
image(r, main = "Raw Ratings")
image(r_m, main = "Normalized Ratings")

# Can also turn the matrix into a 0-1 binary matrix
r_b <- binarize(r, minRating=1)
as(r_b, "matrix")

# Create a recommender object (model)
#   Run anyone of the following four code lines.
#     Do not run all four
#       They pertain to four different algorithms.
#        UBCF: User-based collaborative filtering
#        IBCF: Item-based collaborative filtering
#      Parameter 'method' decides similarity measure
#        Cosine or Jaccard
rec=Recommender(r[1:nrow(r)],method="UBCF", param=list(normalize = "Z-score",method="Cosine"
rec=Recommender(r[1:nrow(r)],method="UBCF", param=list(normalize = "Z-score",method="Jaccard
rec=Recommender(r[1:nrow(r)],method="IBCF", param=list(normalize = "Z-score",method="Jaccard
rec=Recommender(r[1:nrow(r)],method="POPULAR")

# Depending upon your selection, examine what you got
print(rec)
names(getModel(rec))
```

```
getModel(rec)$nn

############Create predictions#############################
# This prediction does not predict movie ratings for test.
#   But it fills up the user 'X' item matrix so that
#    for any userid and movieid, I can find predicted rating

recom <- predict(rec, r[1:nrow(r)], type="ratings")
recom

########## Examination of model & experimentation  #############

# Convert prediction into list, user-wise
as(recom, "list")
# Study and Compare the following:
as(r, "matrix")     # Has lots of NAs. 'r' is the original matrix
as(recom, "matrix") # Is full of ratings. NAs disappear
as(recom, "matrix")[,1:10] # Show ratings for all users for items 1 to 10
as(recom, "matrix")[5,3]   # Rating for user 5 for item at index 3
as.integer(as(recom, "matrix")[5,3]) # Just get the integer value
as.integer(round(as(recom, "matrix")[6039,8])) # Just get the correct integer value
as.integer(round(as(recom, "matrix")[368,3717]))

# Convert all your recommendations to list structure
rec_list<-as(recom,"list")
head(summary(rec_list))
# Access this list. User 2, item at index 2
rec_list[[2]][2]
# Convert to data frame all recommendations for user 1
u1<-as.data.frame(rec_list[[1]])
attributes(u1)
class(u1)
# Create a column by name of id in data frame u1 and populate it with row names
u1$id<-row.names(u1)
# Check movie ratings are in column 1 of u1
u1
# Now access movie ratings in column 1 for u1
u1[u1$id==3952,1]
```

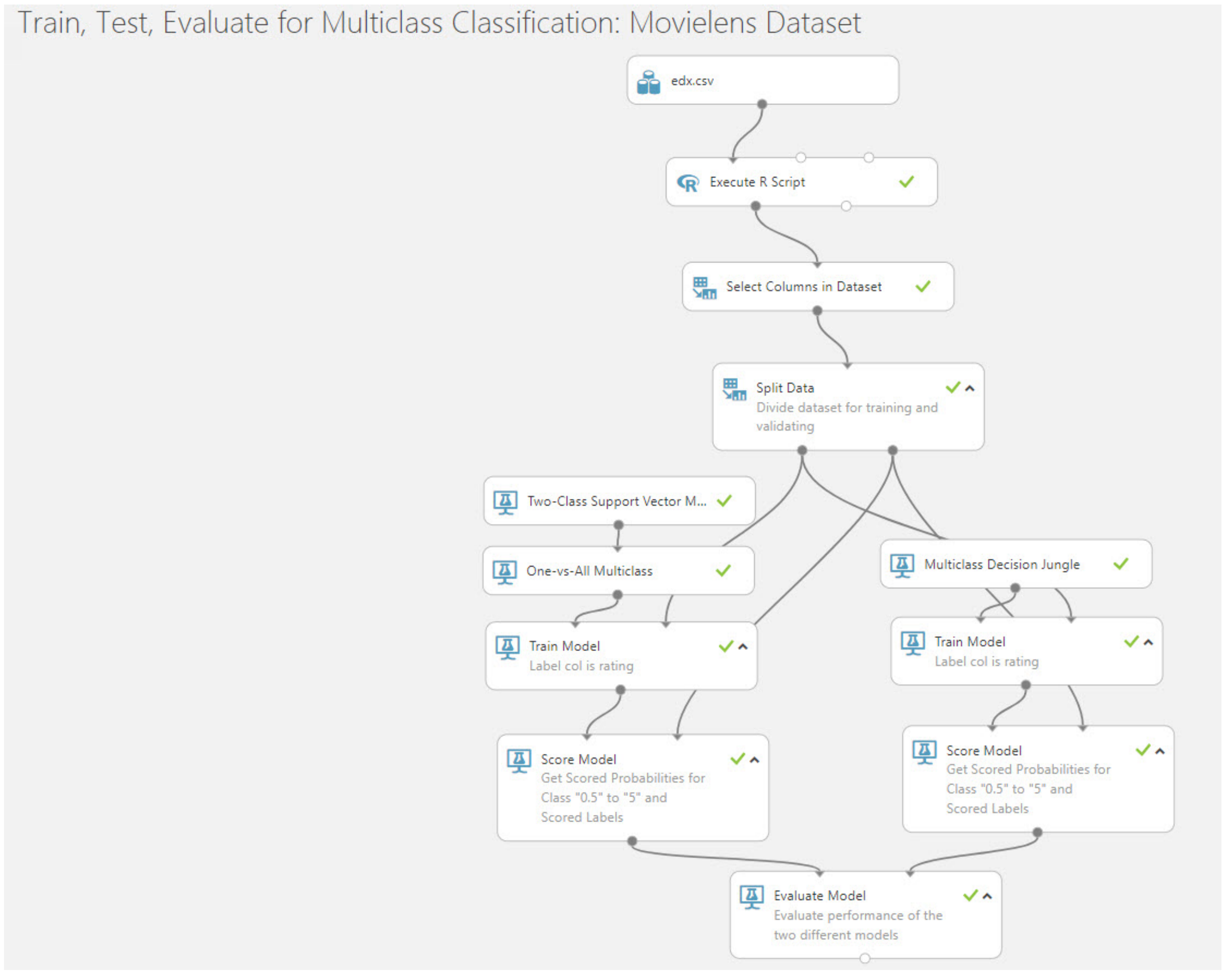
```
# Read test file
#test<-read.csv("test_v2.csv",header=TRUE)
set.seed(1337)
#test <- sample_n(edx, 1000, replace = FALSE)
test <- edx[1:100000,]
test <- tr %>% select(userId,movieId)
# test <- test %>% select(userId,movieId)
test$userId <- as.integer(test$userId)
test$movieId <- as.integer(test$movieId)

match <- as.data.frame(test$movieId) == tr$movieId
test <- test[match,]

head(test)
# Get ratings list
rec_list<-as(recom,"list")
head(summary(rec_list))
ratings<-NULL
# For all lines in test file, one by one
for ( u in 1:length(test[,1])) {
   # Read userid and movieid from columns 1 and 2 of test data
   userId <- test[u,1]
   movieId<-test[u,2]

   # Get as list & then convert to data frame all recommendations for user: userid
   u1<-as.data.frame(rec_list[[userId]])
   # Create a (second column) column-id in the data-frame u1 and populate it with row-names
   # Remember (or check) that rownames of u1 contain are by movie-ids
   # We use row.names() function
   u1$id<-row.names(u1)
   # Now access movie ratings in column 1 of u1
   x= u1[u1$id==movieId,1]
   # print(u)
   # print(length(x))
   # If no ratings were found, assign 0. You could also
   #   assign user-average
   if (length(x)==0)
   {
     ratings[u] <- 0
   }
   else
   {
     ratings[u] <-x
   }

}
length(ratings)

tx<-cbind(test[,1], floor(ratings / 0.5) * 0.5)



# round original ratings to full numbers for having the same factors

tx_2 <- as.data.frame(tx)
confusionMatrix(as.factor(tx_2$V2),as.factor(round(tr$rating)))$overall["Accuracy"]

tx_2$V2[tx_2$V2 == 0] <- 0.5
confusionMatrix(as.factor(tx_2$V2),as.factor(tr$rating))$overall["Accuracy"]
summary(tr$rating)
summary(tx_2$V2)

table(as.factor(tr$rating))
table(as.factor(tx_2$V2))
```

5.3 Approach: Using Azure ML

At some point i decided to do a **more power** cloud approach using **Azure Machine Learning**. There i created multiple classification and also clustering approaches.

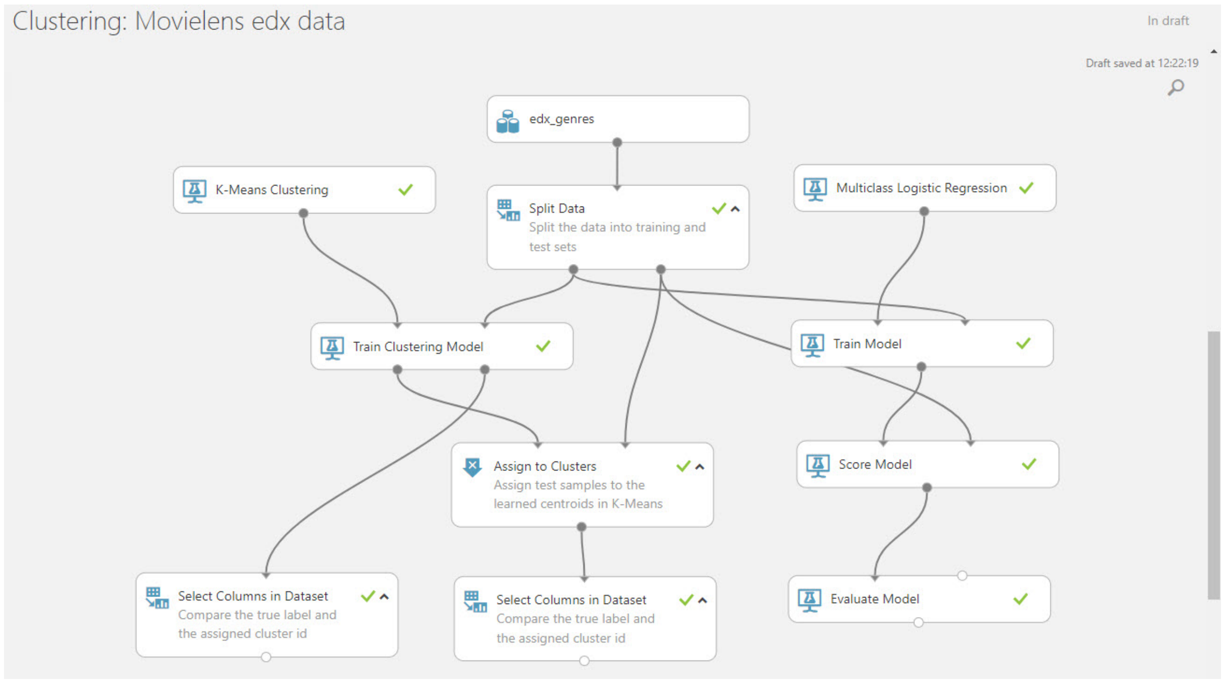
The calculation worked fine, the ressource limitation was gone but still i wasn*t able to get a better accuracy.

5.3.1 Classification


Train, Test, Evaluate for Multiclass Classification: Movielens Dataset
edx.csv
Execute R Script
Select Columns in Dataset
Split Data
Divide dataset for training and validating
Two-Class Support Vector M...
One-vs-All Multiclass
Multiclass Decision Jungle
Train Model
Label col is rating
Train Model
Label col is rating
Score Model
Get Scored Probabilities for Class "0.5" to "5" and Scored Labels
Score Model
Get Scored Probabilities for Class "0.5" to "5" and Scored Labels
Evaluate Model
Evaluate performance of the two different models

bad accuracy: * Two-Class Support Vector Machine with One-vs-All-Multiclass: 0.3316 * Multiclass Decision Jungle: 0.292

5.3.2 Clustering


Clustering: Movielens edx data
In draft
Draft saved at 12:22:19
edx_genres
K-Means Clustering
Split Data
Split the data into training and test sets
Multiclass Logistic Regression
Train Clustering Model
Train Model
Assign to Clusters
Assign test samples to the learned centroids in K-Means
Score Model
Select Columns in Dataset
Compare the true label and the assigned cluster id
Select Columns in Dataset
Compare the true label and the assigned cluster id
Evaluate Model

slight improvement in accuracy:

- K-Means: 0.33
- Multiclass Logistic Regression: 0.38