Movielens Project

edX / HarvardX PH125.9x 'Data Science: Capstone' project

jrollb

5 Jan, 2021

Contents

Introduction

This is my submission of the 'Movielens Project', one of the two final projects of the edx course HarvardX PH125.9x 'Data Science: Capstone'. The task is to build a movie recommendation system using the database 'MovieLens' (see https://grouplens.org/datasets/movielens/), which contains movie ratings for a large number of movies and viewers ('users'). The goal of the movie recommendation system is to predict ratings for user/movie pairs for which no rating exists yet. Given a specific user, the ratings predicted to be high can be considered a good movie recommendation for this user.

The modeling approach contains two steps. First, I set up a basic model similar to the one discussed in the lecture, which predicts the overall average rating plus a movie effect and a user effect. Secondly, I extend the model by applying singular value decomposition (SVD) to the unexplained residuals. Although the SVD method was described in the course material, its application was not worked out in detail there. Here, I show an implementation of the SVD approach in detail.

For a stable estimation of the movie and user effects I used regularization, as suggested in the course material. This contains a parameter, which has to be chosen as part of the model building process. Similarly, the SVD requires a determination of the number of singular values to be taken into account. In order to determine these parameters, I have split the complete 'training' sample into a training and testing set. By performing various fits with different parameter values on the training set and testing them on the test set it is possible to determine an optimal choice. Once these parameters have been determined, the final model can be fitted on the validation set provided with the problem.

There are a few interesting takeaways from this exercise. As expected, the first two or three simplest steps result in considerable improvements. After that it becomes harder to further improve the performance. As the SVD decomposition and the mapping of the predictions to the test set (or later to the validation set) take quite a bit of computing time, the best choice of the number of factors is partly limited by computing power. Further improvement could be made with more processing power, but there is a cost and as a result a tradeoff. On the computer used for the assignment, the number of factors seems to be limited below the number the data itself would allow. However, the results are satisfying enough.

Data set

The MovieLens database is provided by the research lab 'GroupLens' of the University of Minnesota. The data set used for this project is available from the web link http://grouplens.org/datasets/movielens/10m/. It is a 10M sized subset of a much larger database, see https://grouplens.org/datasets/movielens/latest/ for the latest version of the complete set. The 10M subset has been posted in the year 2009.

Reference:

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872

Goals

The goal of the project is to create a successful recommendation system. Success is measured in terms of the root-mean-square-error (RMSE) of the residuals (i.e., the unexplained deviations between predicted and actual values), when the system is applied to a validation set. Definition and details on the RMSE will follow below.

Key steps

The steps to be performed can be grouped into intermediate milestones:

- Data pre-processing: A larger piece of code has been provided, which creates consistent data sets for all students. More explicitly there is a training set 'edx' and a test set 'validation'. However, these are meant for the final model training and its evaluation. For finding a good model, I will subdivide the set 'edx' further into training and test sets ('edx_train' and 'edx_test'). Further, the column containing the film genres contains strings of all genres that fit a movie separated by the character '|'. This results in a very large number of different entries. For the application of tree models, it is advantageous to transform this into a 'tidy' format.
- Data exploration and visualization: In order to understand the data structure, contents, and possible features, several simple analysis steps and graphical representations are used. Interesting statistics are the number and range of ratings, but (for the SVD) also the size of the data set.
- Using insights to outline possible modeling approaches: The data exploration steps provide a modeling guideline. The first insight is that there are many movie/user pairs, making it hard to apply the standard machine learning algorithms. This motivates to first introduce movie and user biases as shown in the course material. Beyond that, the insights on genre would at first sight motivate a 'drama bias'; however, as the movie ID already implies the genre effect, it should only be applied if there are no (or very little) movie ratings.
- Model building and testing on the test subset (not yet the validation set): I will first work with one test set and play with the seed and the split ratio between train and test set. These preliminary steps are not included here. In order to determine best parameters, a simplified 'cross validation' is used:

The model is fit for various parameters, such that an optimal choice can be made. For various reasons (to be discussed below) I only fit once on the same test set (in contrast to real cross-validation, where different samples of training and test sets are used).

- Final model choice and evaluation on the validation set: In this step I will consider the final model as my best prediction model and evaluate it once on the test set 'validation'.
- Interpretation of the results

Methods

Data cleaning

For this project, the data download and preparation has been provided with the problem description. I have put these steps in a separate R script, provided with the submission or on github. I added two lines to store the resulting data sets 'edx' and 'validation' as 'rds' files. The latter will be loaded below for the use in this Rmd file. Probably the most important data cleaning step is a semi-join that excludes observations from the validation set, for which no user/movie pair is in the training set. In other words, no predictions are provided for new users or new movies.

For some investigations regarding the effect of the film genre, I wrote a little function, which supplements the field 'genres' by separate columns for the main genres ("drama", "action", etc.) filled with a one if the genre is contained in the 'genres' field and zero otherwise.

```
#----------------------------------------------------------
# Define a function, which adds columns for the main genres
#----------------------------------------------------------

extend_genre_column <- function(x){
  x %>% mutate(genre.Action = as.numeric(str_detect(genres,"Action")),
               genre.Adventure = as.numeric(str_detect(genres,"Adventure")),
               genre.Comedy = as.numeric(str_detect(genres,"Comedy")),
               genre.Drama = as.numeric(str_detect(genres,"Drama")),
               genre.Thriller = as.numeric(str_detect(genres,"Thriller")))
}
```

It is possible to extend this function to other genres. As this will not be used in the final model, the reason of which I explain below, I only create these few extra columns during run time.

Data exploration and visualization

First load the data set 'edx', previously stored as an rds-file (see R-script):

```
# Load the training set 'edx'
edx <- readRDS("edx.rds")
```

Structure of the data set In order to understand the structure of the 'edx' dataset, let's look at a random sample of rows:

```
# Show sample output
set.seed(10)
edx %>% slice_sample(n = 7) %>% kable()
```

userId	movieId	rating	timestamp	title	genres
38340	1617	3.0	1068866925	L.A. Confidential (1997)	Crime|Film-Noir|Mystery|Thriller
37100	3160	4.0	1068420183	Magnolia (1999)	Drama
7972	1515	3.0	879125432	Volcano (1997)	Action|Drama|Thriller

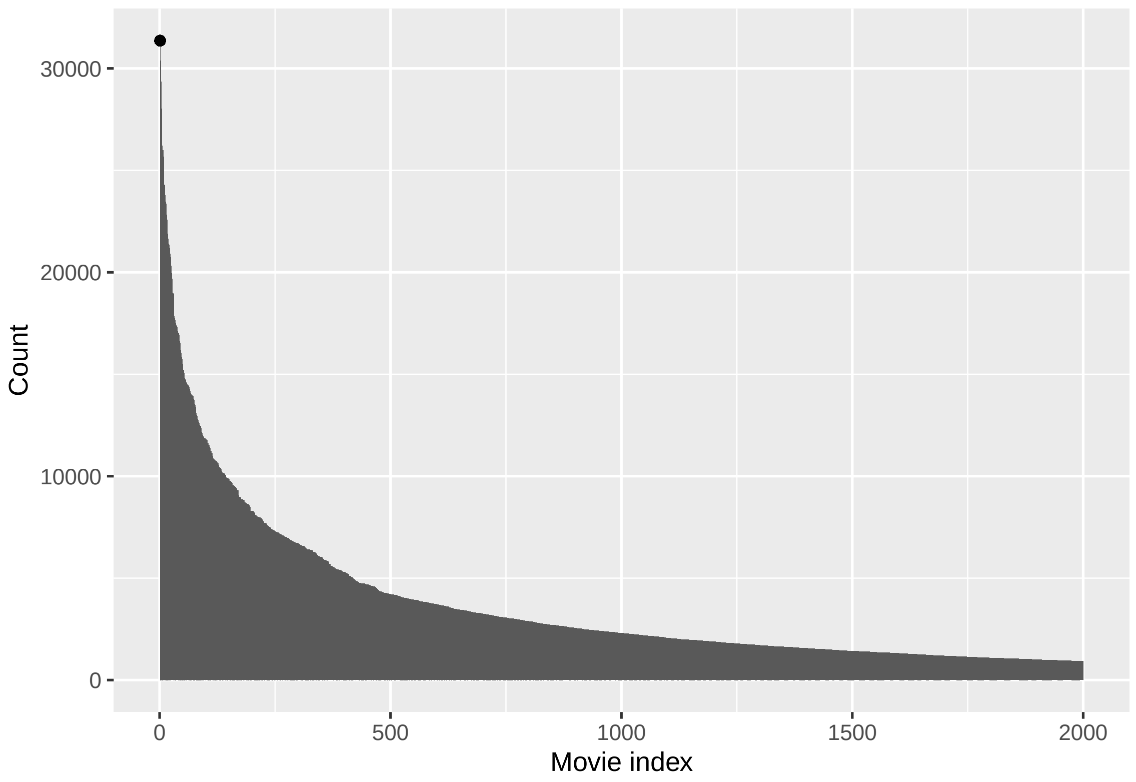
userId	movieId	rating	timestamp	title	genres
38170	6459	2.5	1103689475	Desert Fox, The (1951)	Drama|War
12729	2908	4.0	1112547801	Boys Don't Cry (1999)	Drama
49194	3021	2.0	994876441	Funhouse, The (1981)	Horror
57122	1220	4.5	1078678324	Blues Brothers, The (1980)	Action|Adventure|Comedy|Musical

There are 9000055 observations and 6 columns (user ID, movie IDs, rating, timestamp, title of the movie, genres of the movie). The genre column contains all genres into which the movie fits. These genre types are separated by a 'pipe' symbol. The column 'timestamp' is presumably the time of the rating, but I will not use it here. (It can be converted into a more readable format with the 'as_datetime' function from the 'lubridate' library.)

There are 69878 users and 10677 movies. The number of ratings is far smaller than the product of user and movie numbers. This indicates that there are many user/movie pairs without ratings. These are the gaps, the recommendation system is supposed to fill.

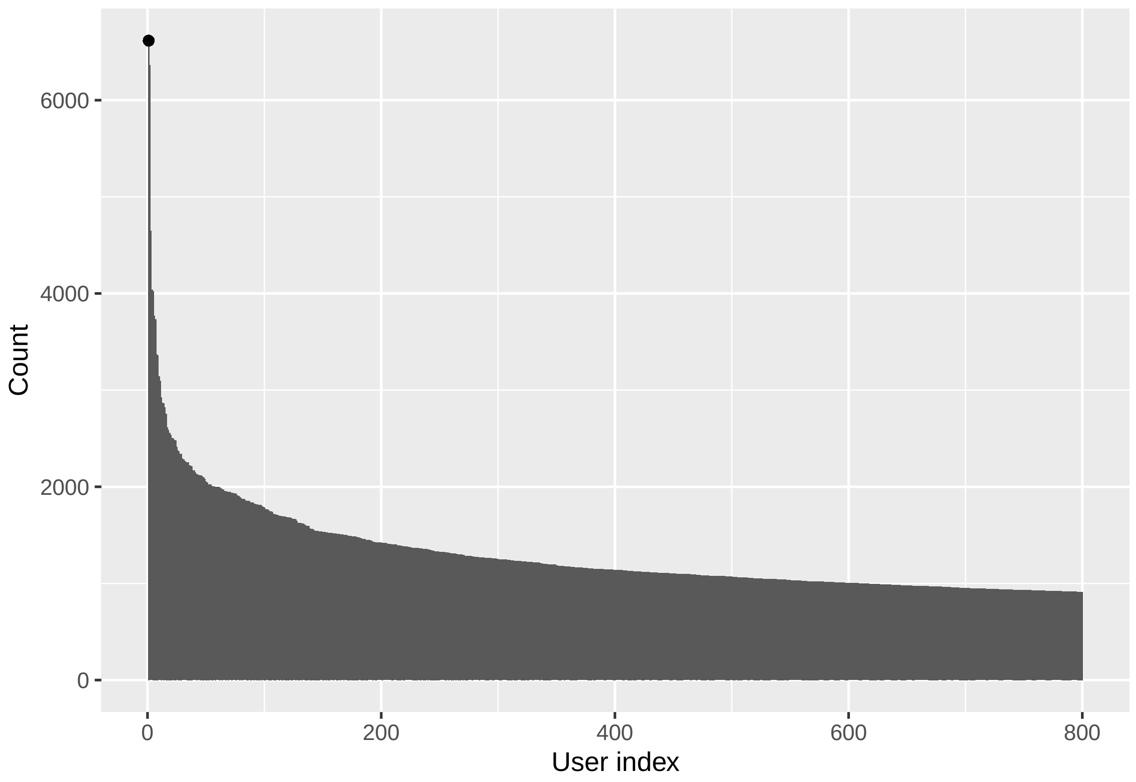
Rating counts by movie and by user The range of rating counts per movie can be illustrated as follows (showing only the first 2000 movies ordered by number of ratings):

```r
# Plot, illustrating the range of rating counts per movie
edx %>%
  group_by(movieId) %>%
  summarize(n = n()) %>%
  arrange(desc(n)) %>%
  mutate(index = 1:nrow(.)) %>%
  filter(index <= 2000) %>%
  ggplot(aes(x = index,y = n)) +
  geom_col() +
  labs(x="Movie index",y="Count") +
  geom_point(aes(x = 1,y = max(n)))
```



The dot marks the maximum number of ratings of over 30’000. The ratings spread over an enormous range. Similarly, the number of ratings per user is spread quite a bit (showing only the first 800 users ordered by number of ratings):

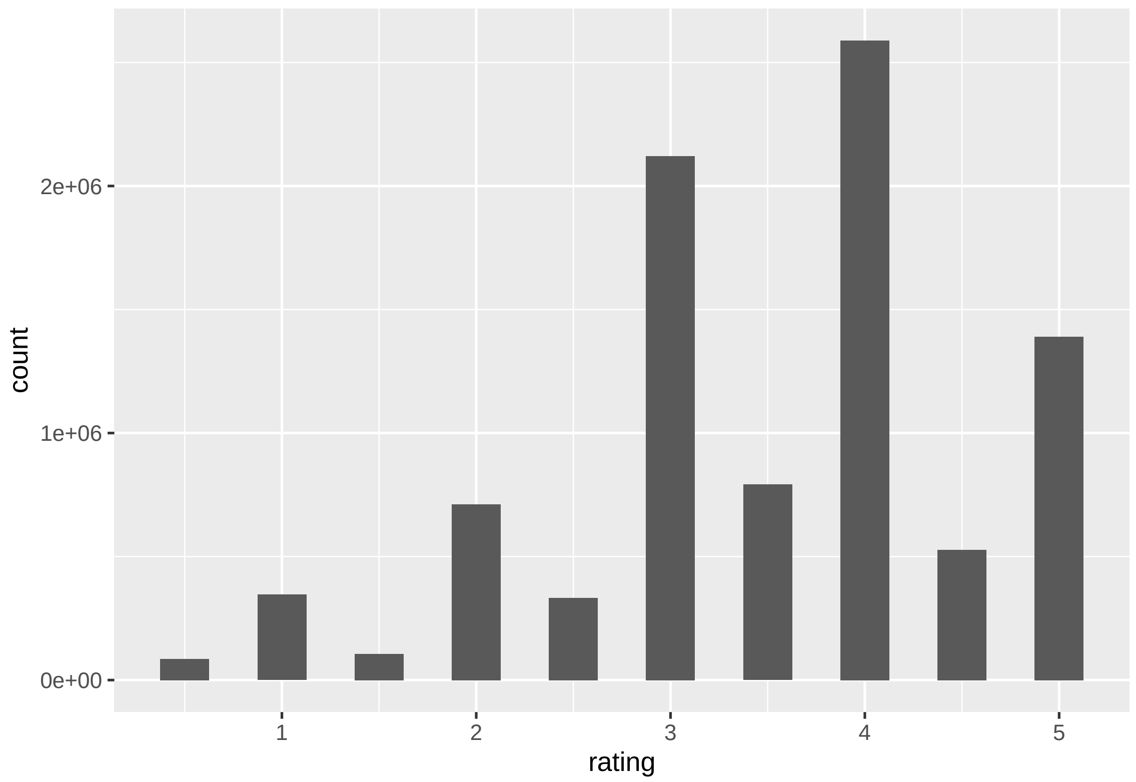
```r
# Plot, illustrating the range of rating counts per user
edx %>%
  group_by(userId) %>%
  summarize(n = n()) %>%
  arrange(desc(n)) %>%
  mutate(index = 1:nrow(.)) %>%
  filter(index <= 800) %>%
  ggplot(aes(x = index,y = n)) +
  geom_col() +
  labs(x="User index",y="Count") +
  geom_point(aes(x = 1,y = max(n)))
```



Again, the maximum number of ratings for one user (over 6'000) is marked by a dot.

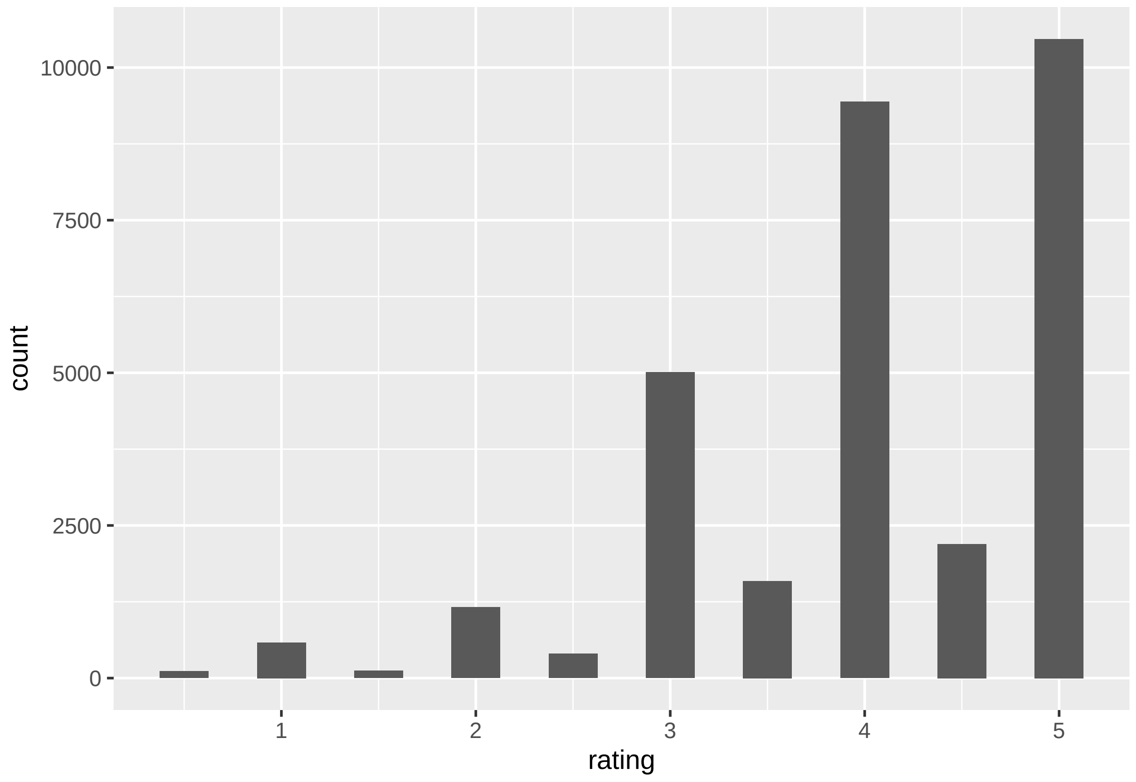
Distributions of ratings The histogram of ratings looks as follows:

```
# Histogram of ratings (using geom_col instead of geom_histogram for better control)
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(rating,count)) +
  geom_col(width = 0.25)
```

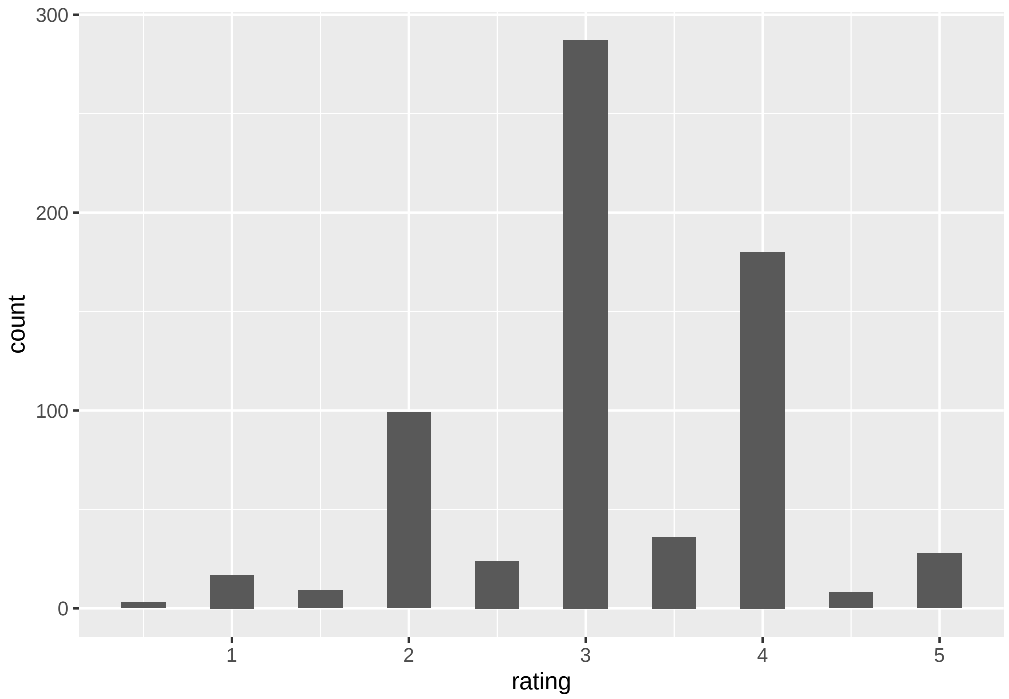


The half-integer ratings are less frequent.

Here is an example distribution of ratings for a highly rated movie (“Forest Gump”):

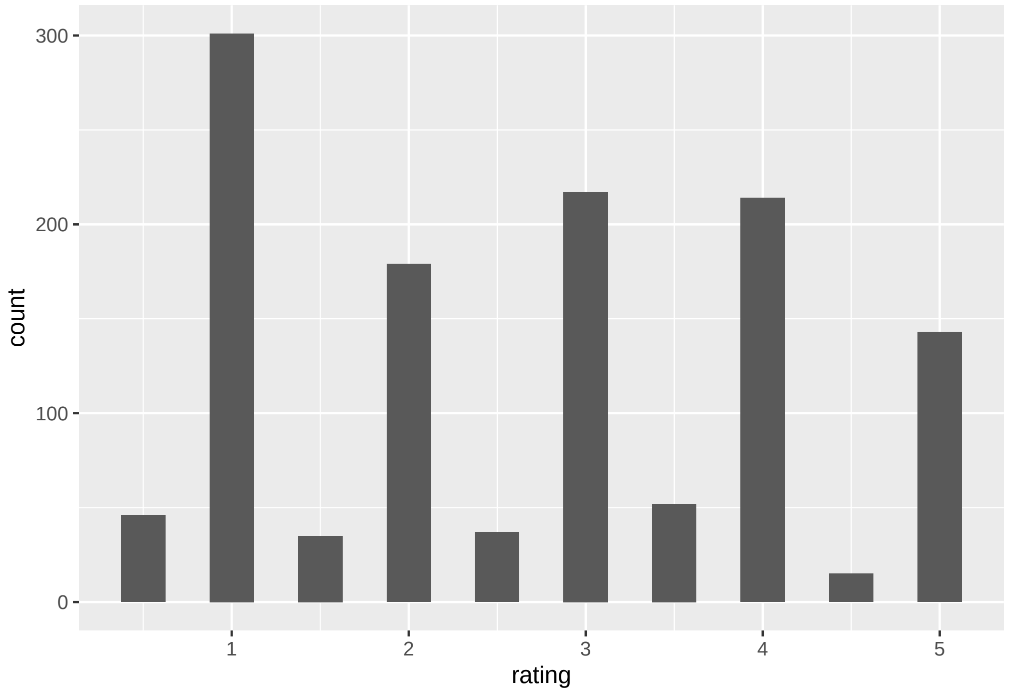


This distribution is very skewed, as the rating is limited on the high side. A movie with a more average rating looks more symmetric



A movie with very low ratings is skewed opposite to the one with high ratings. It is conceivable that the skewed distributions for highly and lowly rated movies distorts the statistics somewhat. A numerical transformation assuming a distribution with boundaries could change this. However, it seems to produce considerable effort, while the potential for improvement is unclear.

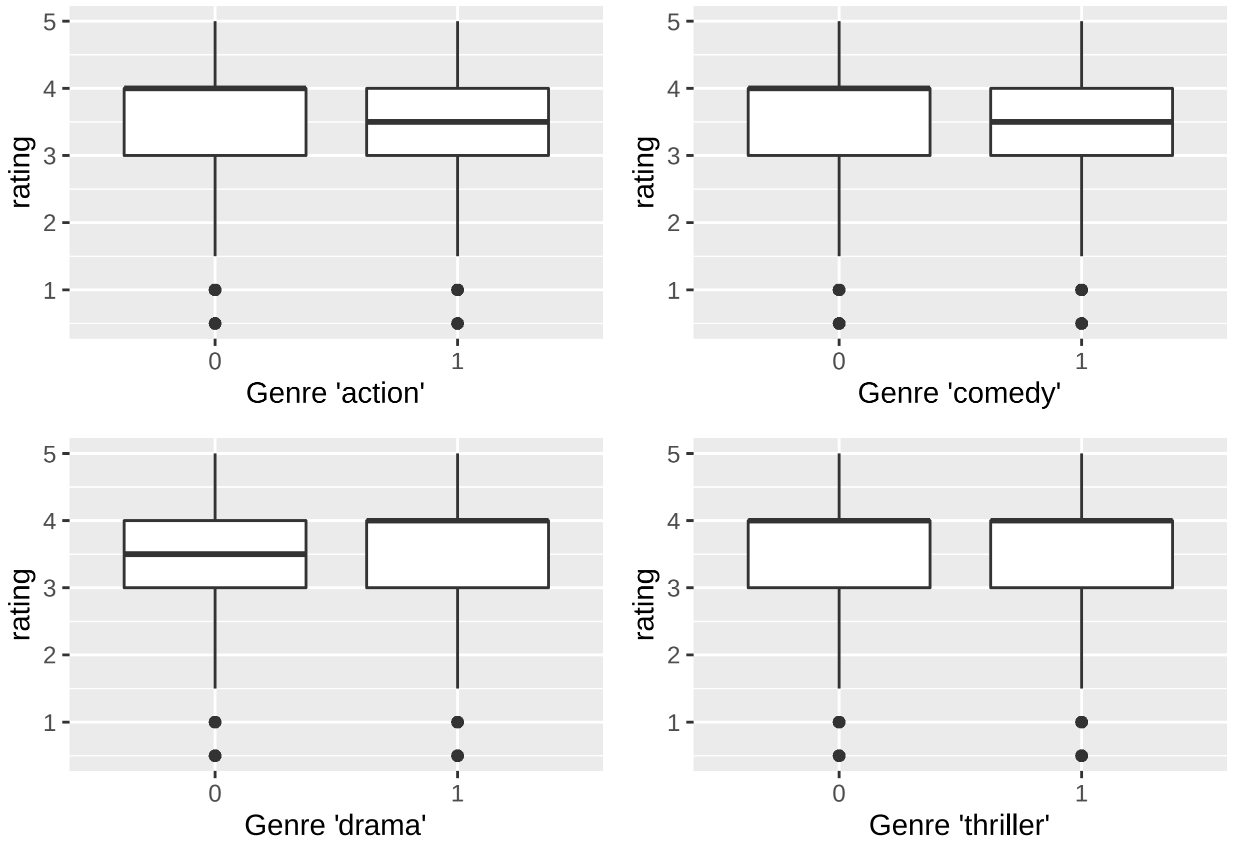
There are exceptions, where a ‘bad’ movie is considered a cult movie by some. One such example is “Plan 9 from Outer Space”:



This is a very untypical (almost uniform) distribution.

Genre The data set contains the movie genre as a variable. This column is condensing all possible genre labels for a given movie into one long string. One can either consider the combination as a whole as genre or split it up into individual types of genre.

How big is the genre effect? Usually, a boxplot for a few of the major genres would illustrate the explanatory power. Here I randomly select 50'000 observations and display boxplots for four genres.



Since the rating is not continuous, it is hard to see the effect on the mean value. (The median difference can be larger than the mean difference here.) So, in this case it is better to look at numbers directly. The mean values (for the 50'000 observations) from a direct calculation are:

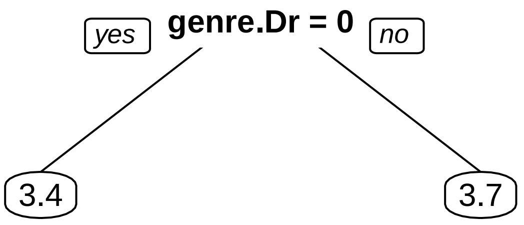
```r
tmpEdx %>%
  summarize(avg.rtg.action = sum(rating * genre.Action)/sum(genre.Action),
            avg.rtg.adventure = sum(rating * genre.Adventure)/sum(genre.Adventure),
            avg.rtg.comedy = sum(rating * genre.Comedy)/sum(genre.Comedy),
            avg.rtg.drama = sum(rating * genre.Drama)/sum(genre.Drama),
            avg.rtg.thriller = sum(rating * genre.Thriller)/sum(genre.Thriller),
            avg.rtg.overall = mean(rating))
```

```
##   avg.rtg.action avg.rtg.adventure avg.rtg.comedy avg.rtg.drama
## 1       3.409582          3.492923       3.444286      3.673146
##   avg.rtg.thriller avg.rtg.overall
## 1         3.513647         3.51041
```

The genre "drama" is the only one with a notable improvement.

Further, I apply a CART model to demonstrate that the main effect of the genre is whether the movie is in the genre 'drama' or not.

```r
# Apply a CART model for main genre relevance
tmpEdx2 <- tmpEdx %>%
  select(-userId,-movieId,-genres)

CART.model <- rpart(rating ~ .,data = tmpEdx2)
prp(CART.model,)
```



The CART model nicely illustrates that only the genre ‘drama’ is relevant for prediction, when looking at overall averages.

Insights gained

- There are too many movie/user combinations for the standard regression and tree methods. The run times would be enormous. So the approach of determining average ratings plus movie and user biases is the method of choice. In order to achieve a further improvement, I will use singular value decomposition, as R provides a fast-enough algorithm for it.
- The half-integer ratings are less likely than their integer neighboring ratings. This could possibly lead to a prevalence problem and it might be an improvement to attribute the half-integer ratings to one of the neighboring integer ratings. However, I will not do so here.
- The ratings are distributed over a finite range and therefore skewed, when close to the borders. It might improve results if the ratings were numerically transformed first. However, as the focus is on average values, the benefit seems small compared to the effort.
- The movie genre is not very powerful in explaining ratings. Only if ‘Drama’ is contained in the entry in the ‘genres’ column, the rating improves slightly on average. Thus it suffices to add a column ‘Drama’ with entries zero and one to cover the genre effect. It can be added as a third bias to the model above. However, Below, I will make the point that there is limited value in using the genre information if single movie averages are also used.

Modeling approach

Performance and the RMSE The model performance of the recommendation system will be measured in terms of the root-mean-square error

$$\mathrm{RMSE}(y_{\mathrm{pred}}, y_{\mathrm{act}}) = \sqrt{\frac{1}{N_{\mathrm{obs}}} \sum_{i,u} \left(y_{\mathrm{pred},i,u} - y_{\mathrm{act},i,u}\right)^2}$$

where $y_{\mathrm{pred},i,u}$ is the predicted movie rating for movie i and user u and $y_{\mathrm{act},i,u}$ is the corresponding actual rating. The sum is over all movie/user pairs for which an actual rating is available. As in this problem for many movie/user pairs there is no actual rating, the sum will not extend over the full grid spread by movie IDs and user IDs. Accordingly, the number of observation is not necessarily the product of number of movies and number of users. (I will comment on this further below.)

The R code implementing this function is

```r
#-----------------------------------------------------------
# Define a function for calculating the RMSE
#-----------------------------------------------------------
```

```r
RMSE <- function(y_pred,y_actual){
  sqrt(mean((y_pred - y_actual)^2))
}
```

Method overview The choice of methods selected below is based on a few boundary conditions for the problem:

- As stated above the large amount of movie/user combinations rules out simple regression or tree methods. Instead I follow the approach via average ratings plus movie and user biases.
- Since an average value is determined for each movie to capture the movie effect, the genre is implicitly taken into account. For this reason I will not include genre as a variable. However, the genre would become important, if a new movie would have to be rated without already existing user ratings (or too few ratings).
- It turns out that the movie/user pair predictions can be improved using SVD.

Accordingly, I will take two steps:

- Firstly, follow the approach taken in the lecture to cover the overall mean, the movie effect, and the user effect.
- Secondly, use SVD to further improve the model.

I will explain these two steps in more detail in the next three subsections.

Basic models covered in the lecture As a basic model, the approach from the course material is repeated, before further improving it later. The approach consists of the following parts:

- Baseline model: For each yet unrated movie/user combination predict a rating equal to the overall average rating μ, where
$$\mu = \frac{1}{N_{\text{obs}}} \sum_{i,u} y_{i,u}$$
is determined from the training set. As there are many gaps in the data, a more proper mathematical notation would likely be
$$\mu = \frac{\sum_{i,u} y_{i,u} 1_{i,u}}{\sum_{i,u} 1_{i,u}}$$
with the sums covering the whole index range (i.e., all movies and all users). The indicator $1_{i,u}$ equals one if there is a rating for movie i and user u and zero otherwise. A similar notation would be appropriate for the RMSE, for the movie effect below, and for the user effect below. But I consider the original notation self-explaining.
- Movie effect: For each unrated movie/user combination predict $\mu + b_i$, where b_i is the mean rating of movie i across all users. Two versions were provided in the lecture:
  - without regularization, $b_i = \frac{1}{N_{\text{user}}} \sum_u (Y_{i,u} - \mu)$,
  - with regularization, $b_i = \frac{1}{N_{\text{user}}+\lambda} \sum_u (Y_{i,u} - \mu)$. The parameter λ is to be determined later, by running for several values and minimizing.
- User effect: For each unrated movie/user combination predict $\mu + b_i + b_u$, where b_u is the mean rating of the user u across all movies the user rated. Again this was treated in the lecture.
  - without regularization, $b_u = \frac{1}{N_{\text{movie}}} \sum_i (Y_{i,u} - \mu - b_i)$,
  - with regularization, $b_u = \frac{1}{N_{\text{movie}}+\lambda} \sum_i (Y_{i,u} - \mu - b_i)$ with the regularized b_u. Note that b_i depends on λ from the first bullet point above, where λ is to be determined later, by minimizing the RMSE. So the parameter λ is shared. The formula for b_u is not completely exact, as both the averages

should actually be determined at the same time. However, the approach above simplifies the formula and the code considerably.

In principle, the analogous method can be applied to include the movie genre. However, as each movie already got its average, the genre is already taken care of. It would be different, if no movie ratings (or a very small number) were available. In that case, the genre could be used to match preferences of the user.

Options for advanced approaches Further improvements of the model suggested in course material (including problem sets) are

- Include a time effect: In one of the problems a weak dependency of the mean over the time of the rating was observed. It is in principle possible to fit this curve and work with time-dependent averages. However, the noise exceeded the variations of this function considerably, so I will not treat it here.
- Include a genre effect: In one of the problems a slight dependency on the film genre was observed. There are several ways to approach this. In fact, I have tried a few things, including clustering. However, this did not lead to the desired effect, because it is already contained in the movie effect: As each movie has a fixed genre combination, the genre effect does not add benefit. It would only be valuable if (1) there were no movie rating yet, but the genre was known or (2) if the number of movies would not allow for determining a mean for each movie, in which case the movies could be grouped by genre combination.
- Matrix factorization
  - Singular value decomposition (SVD). This will be worked out below.
  - Principal component analysis (PCA). Given the similarities of the approaches and the success of the SVD approach, I will not implement this. It would probably lead to similar results.

Singular value decomposition (SVD) The idea behind the SVD is to detect broad features, which explain the largest proportion of the variance. For the MovieLens application, it couples certain user groups prefering certain movie types. The SVD is an exact decomposition of a matrix into a product of the form

$$M = UDV^{\mathrm{T}}$$

with a diagonal matrix D. The latter contains the ordered singular values as diagonal elements. Their squares sum up to the total variance of the matrix elements. The matrix dimensions, as it is structured in the R base package function 'svd', is as follows: If the number of rows of M is larger than the number of columns, then the matrix U has the same dimensions as M, while V is quadratic with the same dimensions as D. If the number of columns is larger, then U is a square matrix and V is rectangular. The usefulness lies in the expansion in terms of the ordered singular values,

$$M = \sum_n d_n \mathbf{u_n} \mathbf{v_n}^{\mathrm{T}}$$

where the first few terms cover the largest contributions to the total variance. Effectively, each term couples certain user groups to certain movie groups.

The SVD algorithm requires a matrix 'movie × user' with entries for all movies and user selected. On a small laptop, the algorithm 'svd' manages a few thousands movies times a few thousands users (e.g. 2000 x 2000) in a few minutes, but beyond that, run times become large. (Roughly speaking, a cubic scaling with size.)

Model training

Preparations

Sub-split of 'edx' into training and test set The following parameter determines the proportion of observations selected for the test set:

```r
# Percentage of data in the test set (should be <= 0.5)
p_testset <- 0.35
```

The reason for the value is that the data set is fairly large, such that the test set can be chosen higher than 20%-30%. The training set is then still large enough and a larger test set reduces the risk of overfitting.

For training the models, I create test and training sets

```r
# Split 'edx' into training and test set
set.seed(123, sample.kind="Rounding")
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```r
indices_test <- createDataPartition(y = edx$rating, times = 1, p = p_testset, list = FALSE)
edx_train <- edx[-indices_test,]
edx_test <- edx[indices_test,]

# Make sure userId and movieId in the validation set are also present in the edx set
edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
```

Step-by-step model training

Baseline model The baseline model assumes that the ratings are randomly spread around a common mean value μ, i.e.,

$$Y = \mu + \epsilon_{i,u}$$

Here, $\epsilon_{i,u}$ is an error term with expected value zero. The baseline model predicts the average rating regardless of the user and movie:

```r
#-----------------------------------------------------------
# Baseline model Y = mu + epsilon_{i,u}
#-----------------------------------------------------------

# Estimate parameter mu (aka mu_hat) as average rating
mu <- mean(edx_train$rating)

# Predict rating using the baseline model
y_hat <- rep(mu,nrow(edx_test))

# Calculate RMSE
rmse <- RMSE(y_hat,edx_test$rating)

# Store result in a tibble
rmse_results <- tibble(method = "Baseline model", RMSE = rmse)
```

The average rating is 3.5122233, which is the value the baseline model would predict. A movie recommendation based on this would have to be a random movie, which the user has not rated yet.

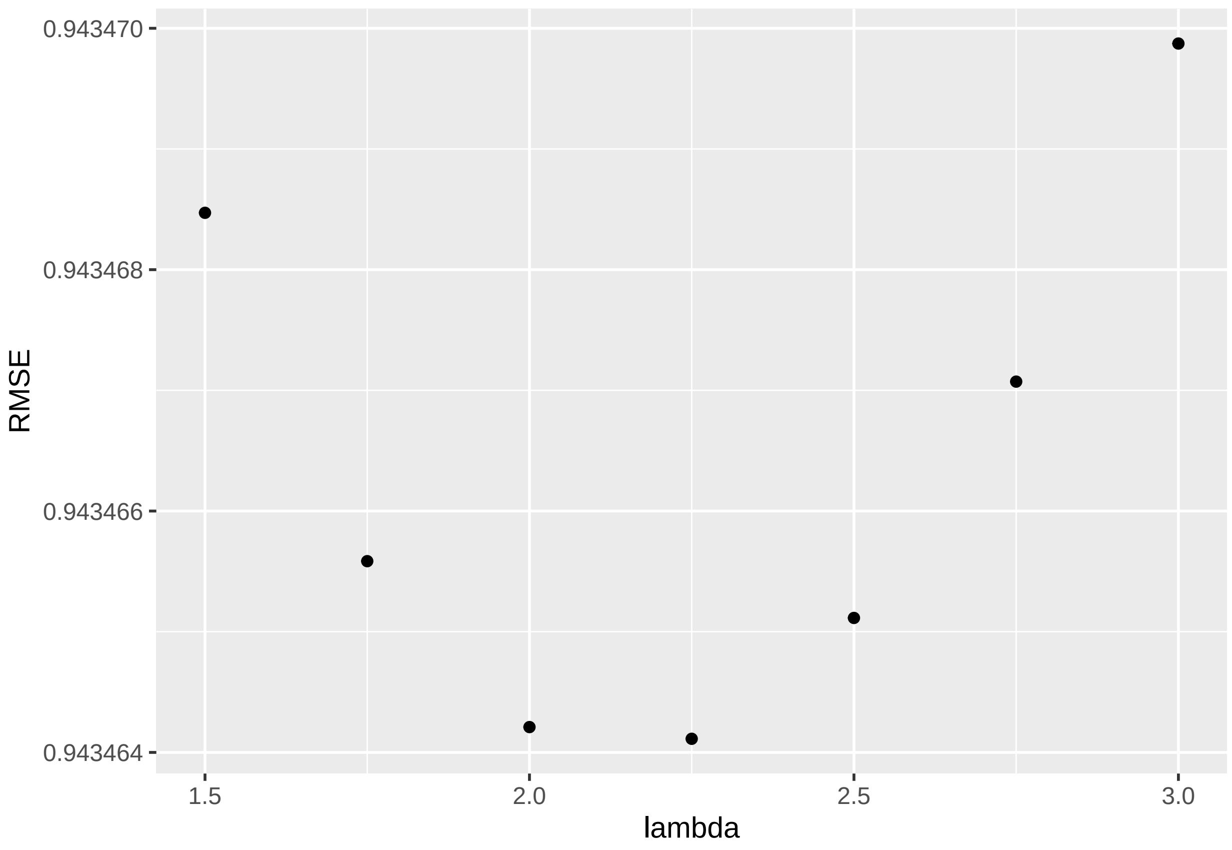
Model with movie effect The baseline model can be improved by introducing the average deviation b_i from the mean μ for each movie.

```r
#-----------------------------------------------------------
# Model with movie effect: Y = mu + b_i + epsilon
#-----------------------------------------------------------
```

```r
# Estimate model parameter b_i (aka b_i_hat)
movie_averages <- edx_train %>%
  group_by(movieId) %>%
  summarize(title = first(title), b_i = mean(rating - mu))

# Predict rating using the model including the movie effect
y_hat <- edx_test %>%
  left_join(movie_averages, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)

# Calculate RMSE
rmse <- RMSE(y_hat,edx_test$rating)

# Append result to tibble
rmse_results <- rmse_results %>%
  bind_rows(tibble(method = "Movie effect", RMSE = rmse))
```

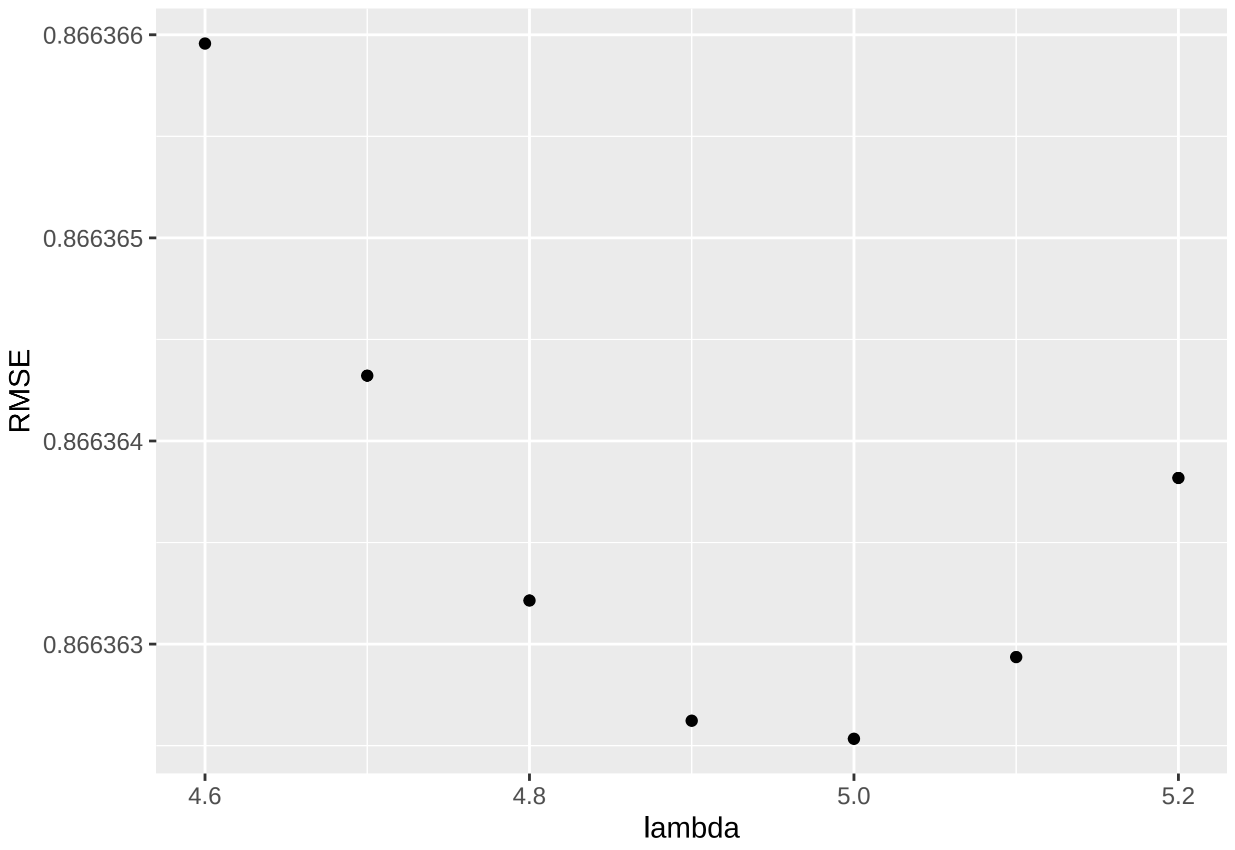
The same can be done using regularization. The regularization parameter can be chosen to cover a wide range first, after which the range is refined close to the minimum.

```r
#----------------------------------------------------------
# Movie effect: regularization
#----------------------------------------------------------

lambdas <- seq(0, 10, 1) # start with a broad range
lambdas <- seq(1.5, 3, 0.25) # refine

rmses <- sapply(lambdas, function(lam){

  movie_averages <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lam))

  y_hat <- edx_test %>%
    left_join(movie_averages, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

  return(RMSE(y_hat, edx_test$rating))
})

# Plot the RMSE values versus lambda values
qplot(lambdas, rmses, xlab = "lambda",ylab = "RMSE")
```



```r
# Pick the optimal lambda
lambda <- lambdas[which.min(rmses)]
rmse <- rmses[which.min(rmses)]

# Store the lambda for later purpose
lambda.movieEffect <- lambda

# Update the movie averages with the best regularized result for later use
movie_averages <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# Append result to tibble
rmse_results <- rmse_results %>%
  bind_rows(tibble(method = "Movie effect, regularized", RMSE = rmse))
```

Model with movie and user effect The parameter estimation is performed as described earlier and similar to the movie effect. A plot of the RMSE for various values of the regularization parameter λ illustrates how the optimal value is found.



The movie and user averages are stored in the variables 'movie_averages' and 'user_averages' for later use.

Genre effect As discussed above, I will not include the genre effect given that it is already included in the movie effect. However, if the task were to predict ratings for unrated movies, the genre would be an important piece of information.

Singular value decomposition (SVD) As the 'edx' dataset is very large, the SVD algorithm (at least on the computer I use) cannot handle the whole set. I reduce the dataset by requiring that for each movie and user there are at least

```
nFilter <- 350 # Aiming for 350! During development: 450 is reasonably fast.
```

observations. For narrowing down the matrix, I use the same procedure that was used in the course (for the purpose of applying PCA). I only include those movies and users for which at least 350 ratings exist.

```
train_small <- edx_train %>%
  group_by(movieId) %>%
  filter(n() >= nFilter) %>% ungroup() %>%
  group_by(userId) %>%
  filter(n() >= nFilter) %>% ungroup()
```

For this reduced dataset, with minimal rating number 350, there are now only 976896 observations. This is only a fraction of the overall 9000055 observations in the 'edx' set.

The reduced set 'train_small' is a tidy table, which I now have to transform into a matrix. This is the format needed to apply a matrix decomposition.

```
y <- train_small %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()

rownames(y) <- y[,1]
y <- y[,-1]
```

The matrix is large, but just so that SVD can finish in a reasonable amount of time:

```
print(dim(y))
```

```
## [1] 1956 2735
```

The goal is to explain everything in addition to the above movie and user effect. For that purpose I subtract the column means representing the movie effect and then the row means representing the user effect.

```
# Subtract means
y <- sweep(y, 2, colMeans(y, na.rm=TRUE))
y <- sweep(y, 1, rowMeans(y, na.rm=TRUE))
```

Finally, set the residuals to zero where ratings are missing.

```
# Set the residuals to zero for the missing ratings (but see limitations in report!)
y[is.na(y)] <- 0
```

It should be noted that this trains the model as if all unknown ratings had obtained the average rating. This is different from predicting the average where we do not know the rating. The prediction does not influence the numeric parameters derived in model fitting. But here, we replace missing values with average values for model fitting and therefore influence the result somewhat.

Apply SVD to the matrix y:

```
#----------------------------------------------------------
# Apply SVD
#----------------------------------------------------------
# print("Start SVD algorithm...")
tmp <- Sys.time()

# Apply SVD algorithm to y
s <- svd(y)

time.SVD <- Sys.time() - tmp
# print("... Finished SVD.")
# print(time.SVD)
```

The runtime of the SVD algorithm was

```
## Time difference of 52.96592 secs
```

Optionally, a few outputs can be produced to understand the output. The output contains three elements:

```
names(s)
```

```
## [1] "d" "u" "v"
```

The correctness of the decomposition can be tested (suppressed, if there are more than 1000 singular values for timing reasons):

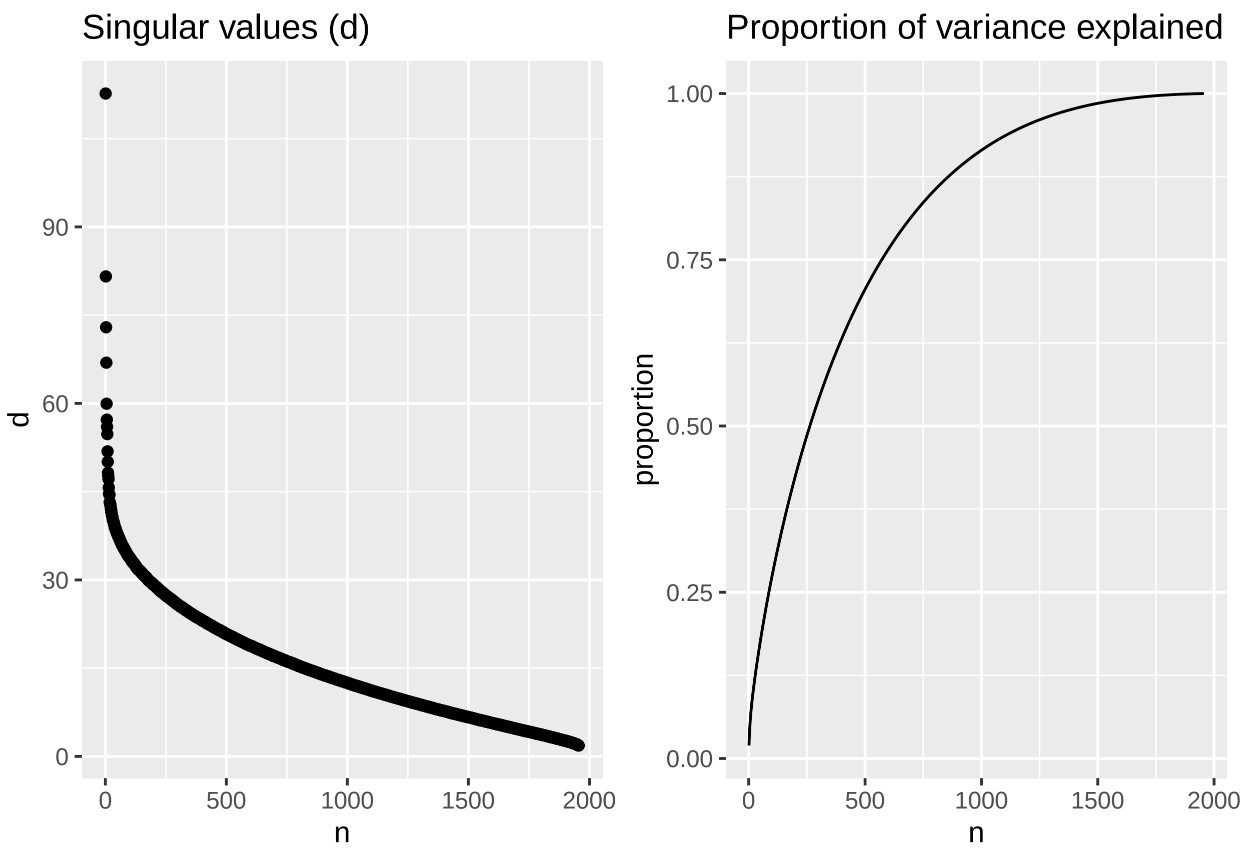
```
if(length(s$d < 1000)){
  y_svd <- s$u %*% diag(s$d) %*% t(s$v)
  print(paste0("Check SVD: Error of composing y from U D V^T: ",max(abs(y - y_svd))))
}
```

```
## [1] "Check SVD: Error of composing y from U D V^T: 1.27826189303543e-13"
```

Apart from numerical noise, the difference to the original matrix is zero.

The plot of singular values (only shown if there are less than 1000 values) demonstrates that they are ordered with large values first. I also show the contribution of the variance explained by the first n 'factors', i.e., the sum of indices 1 through n.

```
if(length(s$d < 1000)){
  p1 <- data.frame(n=1:length(s$d),d=s$d) %>%
    ggplot(aes(n,d)) +
    geom_point() +
    ggtitle("Singular values (d)")
  p2 <- data.frame(n=1:length(s$d),d=s$d) %>%
    mutate(propVar = cumsum(d^2)/sum(d^2)) %>%
    ggplot(aes(n,propVar)) +
    geom_line() +
    labs(y = "proportion") +
    ggtitle("Proportion of variance explained")
  grid.arrange(p1,p2,ncol=2)
}
```



The largest gain comes from the first factor, so let's start with one factor. The following piece of code determines the rating difference additionally explained by SVD. I then add the "delta" to the predictions for movie/user pairs.

```r
# First factor:
pq1 <- with(s,(u[,1, drop=FALSE]*d[1]) %*% t(v[,1, drop=FALSE]))
colnames(pq1) <- colnames(y)
rownames(pq1) <- rownames(y)
delta_y_pq1 <- pq1 %>%
  as_tibble() %>%
  mutate(userID = rownames(y)) %>%
  gather("movie","pq1",-userID) %>%
  mutate(movieId = as.integer(movie),userId = as.integer(userID))

# Predict rating using the model including the user effect
y_hat <- edx_test %>%
  left_join(movie_averages, by='movieId') %>%
  left_join(user_averages, by='userId') %>%
  left_join(delta_y_pq1, by=c('movieId','userId')) %>%
  mutate(pred = mu + b_i + b_u + if_else(is.na(pq1),0,pq1)) %>%
  pull(pred)

# Calculate RMSE
rmse <- RMSE(y_hat,edx_test$rating)

# Append result to tibble
rmse_results <- rmse_results %>%
  bind_rows(tibble(method = "SVD first factor", RMSE = rmse))
```

Due to the reduced dataset used for SVD, there are user/movie pairs without an SVD delta to the prediction. For these pairs I just predict the movie+user effect as before the SVD. Adding the first SVD correction indeed improves the result:

```r
# Intermediate results
kable(rmse_results,caption = "RMSE for each method")
```

Table 2: RMSE for each method

method	RMSE
Baseline model	1.0599209
Movie effect	0.9435537
Movie effect, regularized	0.9434641
Movie and user effect	0.8672370
Movie + user, regularized	0.8663625
SVD first factor	0.8644670

In order to understand the first factor better, let's filter for the movies with the highest contributions from the first factor. Note that the factor describes the relationship between movies and users. I.e., we show a few movie titles that are favored by this group of users.

```r
tmpTitleMapping <- edx_train %>%
  select(movieId,title) %>%
  distinct()

tmpMoviesPQ1 <- delta_y_pq1 %>%
  select(-movie,-userID) %>%
  group_by(movieId) %>%
  summarize(meanPQ1 = mean(pq1)) %>%
  left_join(tmpTitleMapping, by = "movieId") %>%
  select(title,meanPQ1)

tmpMoviesPQ1%>%
  arrange(desc(meanPQ1)) %>%
  head() %>%
  kable()
```

title	meanPQ1
2001: A Space Odyssey (1968)	0.0082104
Clockwork Orange, A (1971)	0.0079548
Fargo (1996)	0.0071483
Royal Tenenbaums, The (2001)	0.0070196
Being John Malkovich (1999)	0.0067921
Taxi Driver (1976)	0.0067449

Similarly, the lowest contributions are:

```
tmpMoviesPQ1%>%
  arrange(meanPQ1) %>%
  head() %>%
  kable()
```

title	meanPQ1
Armageddon (1998)	-0.0084810
Independence Day (a.k.a. ID4) (1996)	-0.0075067
Twister (1996)	-0.0069324
Batman & Robin (1997)	-0.0057863
Ghost (1990)	-0.0055734
Star Wars: Episode I - The Phantom Menace (1999)	-0.0054146

Thus, the component corresponding to the first singular value describes critically acclaimed movies (here the positive values) versus blockbusters (here the negative values). In other words, the largest component of the remaining variance reflects the difference between users who like the more sophisticated film versus those who like the mainstream blockbusters.

Including more factors will reduce the RMSE, but too many factors will lead to overfitting. This is because including all the factors 'explains' every data point including noise. However, the purpose of applying SVD is to filter out the major coarse grained averages from the noise. This is why the factors should be cut off once a good amount of variation is explained. To find a cutoff, more factors are added.

method	RMSE
Baseline model	1.0599209
Movie effect	0.9435537
Movie effect, regularized	0.9434641
Movie and user effect	0.8672370
Movie + user, regularized	0.8663625
SVD first factor	0.8644670
SVD with 2 factor(s)	0.8635810
SVD with 3 factor(s)	0.8628779
SVD with 18 factor(s)	0.8601089

method	RMSE
SVD with 21 factor(s)	0.8600540
SVD with 24 factor(s)	0.8600253
SVD with 27 factor(s)	0.8600256
SVD with 30 factor(s)	0.8600198
SVD with 33 factor(s)	0.8600530
SVD with 36 factor(s)	0.8600891

Now we have to pick the optimal number of factors, keeping in mind that this depends on the number of observations that we take into account. For the given size of the matrix, we can pick the value with the minimum RMSE above. But if we pick 'nFilter' larger, then there is more data to fit. Then it can be expected that a few more factors (not describing noise) can be used. As we pick 'nFilter' such that run times are limited, the number of factors should then also be determined by the amount of data that can be handled within that run time. The number of factors has to give reasonable run times and robust predictions.

Final model choice

Fixing the model assumptions From the earlier checks it seems reasonable to pick 20 to 30 singular value 'factors' So my final choice will be 25 factors. Further, I do not determine the regularization constants for movie and user effect again, but take the values obtained above (2.25 for movie effect only, 5 for movie+user effect).

Refitting on the whole 'edx' set I repeat all the modeling steps for the complete 'edx' set. For later reference I determine the "in sample" RMSE. After that I apply the resulting model fit to the validation set. The really interesting result will be the RMSE on the validation set.

The filter parameter for the SVD has to be set larger, because the 'edx' set is larger than the training sets before.

```
# Restrict the data set by requiring a minimum
nFilter <- 500 # Aiming for 525! 550 is well feasible!!!
```

[1] 2371 2851

The runtime for the SVD algorithm was

```
## Time difference of 1.33382 mins
```

The results will be shown in the following section.

Results

In-sample results

The results obtained from refitting on the whole 'edx' data set are as follows:

method	RMSE
Baseline model (in sample)	1.0603313
Movie effect, regularized	0.9423884
Movie and user effect (regul., in sample)	0.8570452
SVD with 20 factor(s)	0.8426393
SVD with 25 factor(s)	0.8412347
SVD with 30 factor(s)	0.8399808

Needless to say that the in-sample RMSE does not approach a minimum. Taking all factors replicates the matrix completely, yielding a zero RMSE. However, this is over-fitting, because it fits the noise realization of the specific training set (here the ‘edx’ set). The number of factors was determined before and fixed to 25.

Out-of-sample results

In order to assess the performance of the final model, let’s make a prediction on the ‘validation’ set. For this final performance measure I add on little step. I cut off all predicted ratings at a maximum level of 5.0 and a minimum level of 0.5. (I do that only for the final prediction, not for intermediate steps.)

The results for the validation set are as follows, where I have fixed the SVD with 25 factors upfront.

method	RMSE
Baseline model (out of sample)	1.0612018
Movie effect, regularized (out of sample)	0.9438521
Movie and user effect (regul., out of sample)	0.8648177
SVD with 20 factor(s), out of sample	0.8542713
SVD with 25 factor(s), out of sample	0.8539054
SVD with 30 factor(s), out of sample	0.8536523

The final RMSE for the 25 factor SVD correction is 0.8539054.

The other values are only for comparison. The last improvements are less impressive than the first ones. Note that I have picked the number of factors before. So I have to stick with that choice, because the assumption is that the validation set corresponds to the unknown future sample for which the prediction is made. Only in retrospect the factors could have been chosen more ambitiously. This is probably a good sign, because then it is more likely that the factors explain real features and are robust.

Conclusion

Summary

Based on 9000055 ratings for 10677 movies and 69878 users, I set up a recommendation system and trained it on a given test set. I applied it on a validation set to check for robustness and performance on an unknown set. The root-mean-square error (RMSE) on the validation was similar to the in-sample values, which is a good sign that there is no overfitting or other unwanted effects.

The recommendation system predicts the overall mean rating plus a movie and user effect plus combined movie/user group features extracted from the training set using the singular value decomposition (SVD). For the movie and user effect, using regularization prevented very high or very low ratings in cases, where only a very small number of ratings exist. The number of singular value contributions was determined by a performance minimum on the training set when the number was varied, but then combined with some caution to prevent overfitting. This reflects the tradeoff between explaining more of the variance and the risk to fit noise.

In each step the performance of the model improved. It is likely that the SVD could be pushed more, if more computer power would be applied. However, with increasing complexity, the improvements become smaller and smaller. For single-user practical purposes, there is not so much difference between an RMSE of (e.g.) 0.866 and 0.865; in both cases the rating is uncertain by almost one rating point. Only applied to a large number of users it might have a certain importance for the provider of the recommendation system to reduce variance further.

Limitations

A few of the limitations are:

- Not very many variables are available in the given dataset. It would be favorable to have more information, like budget, actors, director, producers, etc.
- The setup ensures that all movie/user pairs are included in the test set. This does not entail the case, where there are no ratings for new users and new movies. In reality, new users probably sign up to streaming services continuously and new movies are released frequently. With the model trained here, the average rating can be assigned, but it would be favorable to use information like genre, actors, director, producers, etc. (see first bullet point)
- For non-existent ratings, the residuals were replaced by zero for the SVD analysis. This is not expected to do much harm, but it could possibly add artificial bias to the estimate. Further, the SVD analysis did not include all data, as this is extremely costly in terms of runtime and memory.

Future work

There are further steps that could potentially improve the model on the given data set. For example, more features like names of actors, . . . , producer, budget etc might be available and could be considered as additional features. The runs above show that there is potential in working with a larger matrix in the SVD. However, instead of brute-force calculation, it should be considered to make the methods more efficient, such that they can be tested on larger data sets.

There are a few minor corrections that can be made to the approach. The regularization formula, averaging by movie first and by user second, contains a simplification. However, while the extra effort is probably large, the size of the improvement is unclear. Further, setting the excess ratings to zero for non-existing ratings for use in the SVD means that the algorithm will treat the zeros like actual ratings. This might influence results negatively. However, due to lack of convincing alternatives, I would expect large effort for little improvement.

Apart from this, the algorithm should be made flexible for practical applications. For example, if a user filters for a certain genre and other available features, then the model should minimize RMSE conditional on these selections. Here again, the genre would acquire importance for a different purpose. There is probably much more to gain here, in particular because it would be very useful for practical applications.

Further, fitting might be more efficient, if the half-integer rating were allocated to the neighboring integer ratings using appropriate weights. This would remove the different counts between integer and half-integer ratings.