MovieLens Capstone Project
Data Science Professional Certificate/HarvardX

Paw Hermansen

March 04, 2019

Executive Summary

The Goal

The goal of this MovieLens Project is to predict the ratings that anonymized users give to movies. The data given is the MovieLens 10M dataset that contains around 10 million ratings. For the purpose of this course the course staff has provided code that splits the original data into a learning dataset and a validation dataset. A model must be created and trained using the learning dataset and finally the model must be validated using the validation dataset.

The predictions are validated using root mean square error (RMSE) as the metric. Specifically a RMSE below 0.87750 is wanted.

The Data

Each rating in both datasets contains the Id of the user giving the ration, the Id of the movie being rated, and the rating itself. It also contains a timestamp for the time of the rating, the titel and the release year of the movie and finally a list of genres of the movie.

I analysed the learning dataset and found, among other, that the learning dataset has 69878 different users giving 9000055 ratings to 10677 different movies in the period from 1995 to May 2009, though only two ratings were given in 1995. I also showed that the average rating is nearly constant over the period. A closer look at the rating patterns, however, show several changes over time, where the largest change happened in May 14, 2003 where ratings change from integer only 1, 2, . . . , 5 to include half-integer ratings 0.5, 1, 1.5, . . . , 4.5, 5.

The Models

I built and trained four models where each model expands the previous one.

Model 1: Average

Model 1 is very simple and always predicts the average of all the ratings:

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

where $Y_{u,i}$ is the rating that user u has given or would give to movie i.

Model 2: Movie Popularity

Model 2 builds on model 1 and also takes into account that some movies on the average are rated higher than other movies:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

where b_i is a measure for the popularity of movie i, i.e. the bias of movie i.

Model 3: User Mildness

Model 3 builds on model 2 and also takes into account that some different users have different average ratings:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

where b_u is a measure for the mildness of user u, i.e. the bias of user u.

Model 4: Genre Popularity

Model 4 builds on Model 3 and also takes into account that different users like or dislike different genres:

$$Y_{u,i} = \mu + b_i + b_u + b_{u,g} + \epsilon_{u,i}$$

where $b_{u,g}$ is a measure for how much a user u likes the genre g.

The Result

Model Name	Model Size (number of floating point numbers)		Validation RMSE
1: Average	One number =	1	1.0612
2: Movie Popularity	+ One number for each movie =	10678	0.9439
3: User Mildness	+ One number for each user =	80556	0.8653
4: Genre Popularity	+ One number per genre for each user =	1478116	0.8498

Conclusion

Model 3 and 4 both have RMSE lower than the wanted 0.87750. Model 4, Genre Popularity, predicts best but is much larger than model 3. Unless the increased prediction is very important, I will recommend to use model 3. Model 3 predicts well and it has a reasonable size.

Analysis

The Assignment and the Data

The goal of this MovieLens Project is to predict the ratings that anonymized users give to movies. The data given is a list movie ratings from MovieLens. For the purpose of this Capstone Project the course staff have written code that parts the original data into a training set `edx` and a validation set `validation`. The validation set must not be used for training but must be used to make a final validation of the developed model.

```
edx <- read.csv(stringsAsFactors = FALSE, file = 'data/edx.csv')
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
```

```r
validation <- read.csv(stringsAsFactors = FALSE, file = 'data/validation.csv')
str(validation)
```

```
## 'data.frame':    999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : int  231 480 586 151 858 1544 590 4995 34 432 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Romanc
```

As seen, `validation` has the same fields as `edx` and they can all be used as predictors. `validation` also contains the actual rating which would give us the possibility to build a model that matches the `validation` exactly but this would be cheating. The model(s) must be fittet using the `edx` only and the ratings in `validation` are only allowed to be used to make the final validation of the model(s).

The metric to use for validation is the root mean square error (RMSE) which is defined as:

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

where N is the number of user u, movie i combinations, $y_{u,i}$ is user u's rating of movie i and $\hat{y}_{u,i}$ is the prediction of $y_{u,i}$.

Data Exploration

In this section I explore the data in `edx`, try to find patterns and investigate if I need to clean or change the data before building a predictive model. Also I check a couple of facts about `validation`.

Basic Information

First a couple of basic sizes of the data in `edx`:

```r
cat('Number of ratings:',  nrow(edx))
```

```
## Number of ratings: 9000055
```

```r
cat('Number of different users:',  length(unique(edx$userId, incomparables = FALSE)))
```

```
## Number of different users: 69878
```

```r
cat('Number of different movies:', length(unique(edx$movieId, incomparables = FALSE)))
```

```
## Number of different movies: 10677
```

It is seen that all users in `edx` have rated at least one movie and that all movies have received at least one rating:

```r
n <- edx %>%
  group_by(userId) %>%
  summarize(numberOfRatings = n()) %>%
  filter(numberOfRatings == 0) %>%
  count()
cat('Number of users that has given no ratings:', toString(n))
```

```
## Number of users that has given no ratings: 0
```

```r
n <- edx %>%
  group_by(movieId) %>%
  summarize(numberOfRatings = n()) %>%
  filter(numberOfRatings == 0) %>%
  count()
cat('Number of movies with no ratings:', toString(n))
```

```
## Number of movies with no ratings: 0
```

Data Cleaning

First I check for missing values in `edx` and `validation` and it turns out that there are no missing values:

```r
cat('Missing values in edx:',  any(is.na(edx)))
```

```
## Missing values in edx: FALSE
```

```r
cat('Missing values in validation:',  any(is.na(validation)))
```

```
## Missing values in validation: FALSE
```

The timestamp field is a number that represents dates and times, actually it is the number of seconds since the start of January 1, 1970. I use the method `as_datetime` from the `lubridate` package to handle conversions between timestamps and datetimes. Because it is so easy I decide to transformation between timestamp and dateTime on scratch where needed.

```r
cat('Date of earliest rating:', format(as_datetime(min(edx$timestamp)), "%Y-%m-%d"))
```

```
## Date of earliest rating: 1995-01-09
```

```r
cat('Date of newest rating:', format(as_datetime(max(edx$timestamp)), "%Y-%m-%d"))
```

```
## Date of newest rating: 2009-01-05
```

`validation` is created to contain only users and movies also present in `edx` which means that all userIds and movieIds in 'validation' already are know in the model(s). This is clearly seen from the following two code-blocks because they remove all rows from `validation` that do not have a `userId` or a `movieId`, respectively, that also exists in `edx`. However the resulting number of rows are exactly the same as the original `validation`, i.e. 999999 meaning that no rows where removed:

```r
nrow(semi_join(validation, edx, by = "userId"))
```

```
## [1] 999999
```

```r
nrow(semi_join(validation, edx, by = "movieId"))
```

```
## [1] 999999
```

All in all it is seen that both `edx` and `validation` are in good formats and that no preliminary data cleaning needs to be done.

Genres

Each movie is assigned one or more genres, and the genres are encoded into one field `genres`. The different genres and the number of movies they are assigned to are:

```r
genres <- edx %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  dplyr::summarize(count = n()) %>%
  arrange(desc(count))

genres
```

```
## # A tibble: 20 x 2
##    genres               count
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

As seen the data has 19 different genres and a pseudo-genre called `(no genres listed)` indicating that the movie has not been assigned any genres.

Actually only one movie has no genres as seen here:

```r
ratingsForMoviesWithPseudoGenre <- edx %>% filter(grepl('(no genres listed)', genres))

ratingsForMoviesWithPseudoGenre
```

```
##   userId movieId rating  timestamp                title             genres
## 1   7701    8606    5.0 1190806786 Pull My Daisy (1958) (no genres listed)
## 2  10680    8606    4.5 1171170472 Pull My Daisy (1958) (no genres listed)
## 3  29097    8606    2.0 1089648625 Pull My Daisy (1958) (no genres listed)
## 4  46142    8606    3.5 1226518191 Pull My Daisy (1958) (no genres listed)
## 5  57696    8606    4.5 1230588636 Pull My Daisy (1958) (no genres listed)
## 6  64411    8606    3.5 1096732843 Pull My Daisy (1958) (no genres listed)
## 7  67385    8606    2.5 1188277325 Pull My Daisy (1958) (no genres listed)
```

Because of the low number of movies, 1, and ratings, 6, I decide to ignore the issue and I handle the pseudo-genre `(no genres listed)` the same as the real genres.

The Highest Rated Movies

```r
edx %>%
  group_by(title) %>%
  summarize(numberOfRatings = n(), averageRating = mean(rating)) %>%
  arrange(desc(averageRating)) %>%
  top_n(10, wt=averageRating)
```

```
## # A tibble: 10 x 3
##    title                                     numberOfRatings averageRating
##    <chr>                                               <int>         <dbl>
##  1 Blue Light, The (Das Blaue Licht) (1932)                1          5
##  2 Fighting Elegy (Kenka erejii) (1966)                    1          5
##  3 Hellhounds on My Trail (1999)                           1          5
##  4 Satan's Tango (Sátántangó) (1994)                       2          5
##  5 Shadows of Forgotten Ancestors (1964)                   1          5
##  6 Sun Alley (Sonnenallee) (1999)                          1          5
##  7 Constantine's Sword (2007)                              2          4.75
##  8 Human Condition II, The (Ningen no joken~               4          4.75
##  9 Human Condition III, The (Ningen no joke~               4          4.75
## 10 Who's Singin' Over There? (a.k.a. Who Si~               4          4.75
```

Oddly enough, these highest rated movies are movies that I have never heard of. But as seen the number of ratings for these movies is extremely low, in some cases only a single rating. To find a more fair list of highest rated movies I need to also take the number of ratings into account. The following shows a list of the highest rated movies that also have more than 100 ratings.

```r
edx %>%
  group_by(title) %>%
  summarize(numberOfRatings = n(), averageRating = mean(rating)) %>%
  filter(numberOfRatings > 100) %>%
  arrange(desc(averageRating)) %>%
  top_n(10, wt=averageRating)
```

```
## # A tibble: 10 x 3
##    title                                         numberOfRatings averageRating
##    <chr>                                                   <int>         <dbl>
##  1 Shawshank Redemption, The (1994)                        28015          4.46
##  2 Godfather, The (1972)                                   17747          4.42
##  3 Usual Suspects, The (1995)                              21648          4.37
##  4 Schindler's List (1993)                                 23193          4.36
##  5 Casablanca (1942)                                       11232          4.32
##  6 Rear Window (1954)                                       7935          4.32
##  7 Sunset Blvd. (a.k.a. Sunset Boulevard) (~                2922          4.32
##  8 Third Man, The (1949)                                    2967          4.31
##  9 Double Indemnity (1944)                                  2154          4.31
## 10 Paths of Glory (1957)                                    1571          4.31
```

Number of Ratings, Rating Users and Rated Movies over Time

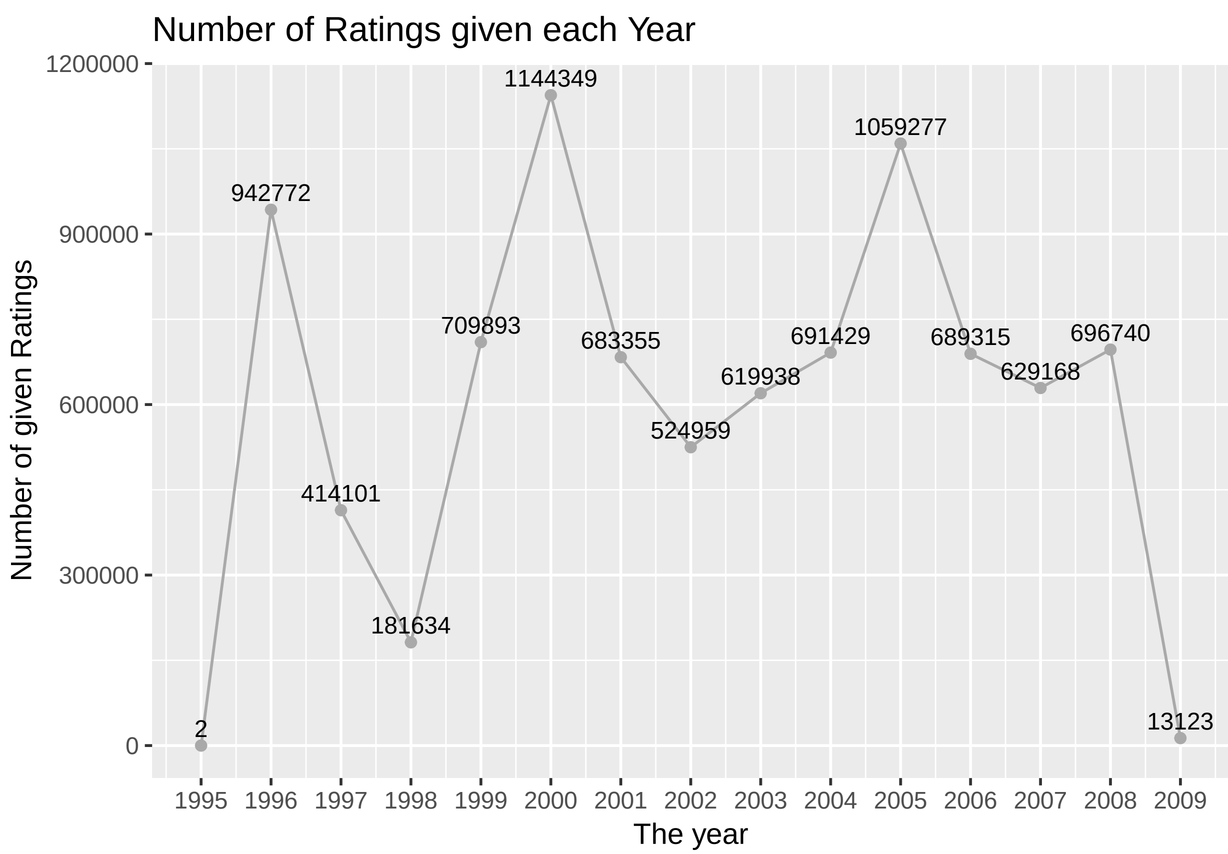
As seen above the 9000055 ratings are given between 1995-01-09 11:46:49 and 2009-01-05 05:02:16. The following table and graphs shows that the number of ratings given each year and the number of active, i.e. rating, users jump up and down. The number of different movies rated each year, however, is rising steadily. Note that the first and last year, 1995 and 2009 are not full years and therefor not comparable to the other years.

```r
edxRatingsGivenPerYear <- edx %>%
    mutate(tsYear = year(as_datetime(timestamp))) %>%
    group_by(tsYear) %>%
    summarize(nRatings = n(), differentUsers = n_distinct(userId), differentMovies = n_distinct(movieId)

edxRatingsGivenPerYear
```

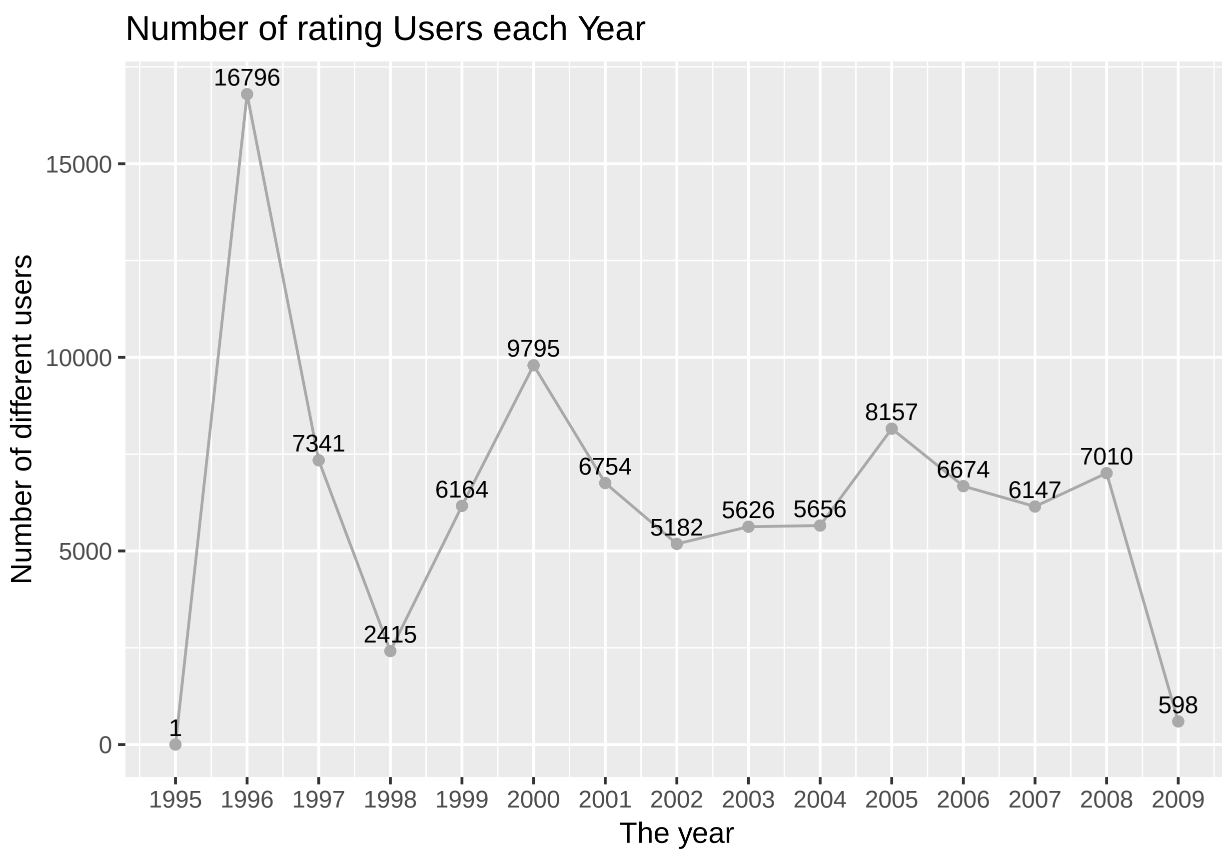
```
## # A tibble: 15 x 4
##    tsYear nRatings differentUsers differentMovies
##     <dbl>    <int>          <int>           <int>
##  1   1995        2              1               2
##  2   1996   942772          16796            1385
##  3   1997   414101           7341            1664
##  4   1998   181634           2415            2261
##  5   1999   709893           6164            3009
##  6   2000  1144349           9795            3810
##  7   2001   683355           6754            4655
##  8   2002   524959           5182            5676
##  9   2003   619938           5626            6743
## 10   2004   691429           5656            7830
## 11   2005  1059277           8157            8281
## 12   2006   689315           6674            8490
## 13   2007   629168           6147            9162
## 14   2008   696740           7010            9590
## 15   2009    13123            598            3452
```

```r
edxRatingsGivenPerYear %>%
    ggplot(aes(x = tsYear, y = nRatings)) +
        geom_point(color = "darkgrey") +
        geom_line(color = "darkgrey") +
```

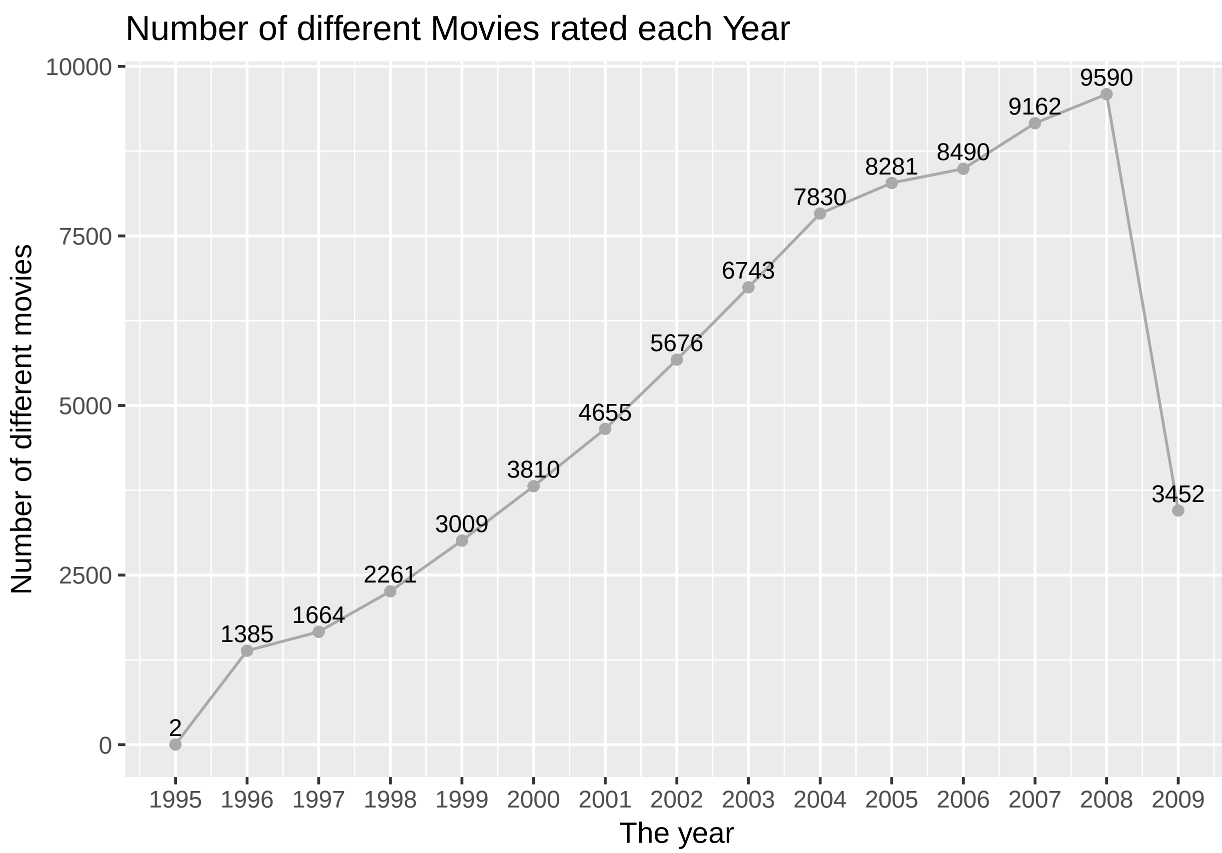
```r
    geom_text(aes(label = nRatings), hjust="middle", vjust=-0.5, size=3) +
    labs(title = "Number of Ratings given each Year",
       x = "The year",
       y = "Number of given Ratings") +
    scale_x_continuous(breaks=seq(1995,2009,1)) +
    theme_grey()
```




```r
edxRatingsGivenPerYear %>%
  ggplot(aes(x = tsYear, y = differentUsers)) +
    geom_point(color = "darkgrey") +
    geom_line(color = "darkgrey") +
    geom_text(aes(label = differentUsers), hjust="middle", vjust=-0.5, size=3) +
    labs(title = "Number of rating Users each Year",
       x = "The year",
       y = "Number of different users") +
    scale_x_continuous(breaks=seq(1995,2009,1)) +
    theme_grey()
```




```r
edxRatingsGivenPerYear %>%
    ggplot(aes(x = tsYear, y = differentMovies)) +
        geom_point(color = "darkgrey") +
        geom_line(color = "darkgrey") +
        geom_text(aes(label = differentMovies), hjust="middle", vjust=-0.5, size=3) +
        labs(title = "Number of different Movies rated each Year",
           x = "The year",
           y = "Number of different movies") +
        scale_x_continuous(breaks=seq(1995,2009,1)) +
        theme_grey()
```



Ratings over Time

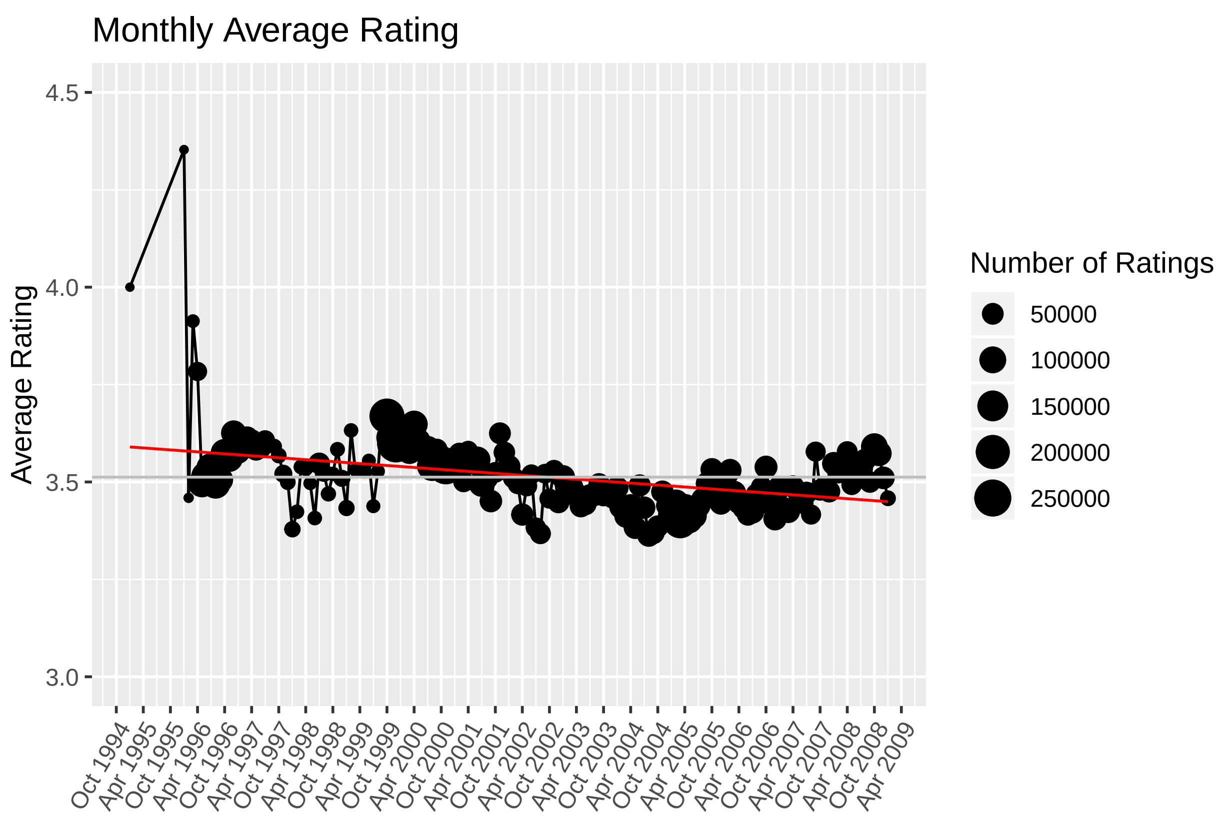
The mean of all the ratings in `edx` is 3.5124652. The following graph shows the mean of the ratings for each month in the dataset.

The size of the dots illustrates how many ratings where given each month. The gray horizontal line show the overall mean while the red line is the linear regression line best fitting through all the ratings.

```
mu = mean(edx$rating)

# Convert the timestamp to Date and add it to edx
edx <- edx %>% mutate(ratingDate = as.Date(as_datetime(timestamp)))

# Compute the monthly averages of the ratings
monthlyAvg = edx %>%
  mutate(month = as.Date(cut(ratingDate, breaks = "month"))) %>%
  group_by(month) %>%
  summarise(averageRating = mean(rating), count = n())

# Fit best linear regression line to the monthly average of ratings
lm_fit <- lm(rating ~ ratingDate, data=edx)
fittedLine = data.frame(ratingDate = monthlyAvg$month)
monthlyAvg$fittedline = predict(lm_fit, fittedLine)

head(monthlyAvg)
```

```
## # A tibble: 6 x 4
##   month      averageRating  count fittedline
##   <date>             <dbl>  <int>      <dbl>
## 1 1995-01-01          4          2       3.59
## 2 1996-01-01          4.35      17       3.58
## 3 1996-02-01          3.46     307       3.58
## 4 1996-03-01          3.91    5948       3.58
## 5 1996-04-01          3.78   31478       3.58
## 6 1996-05-01          3.50 110958       3.58
```

```r
ggplot(monthlyAvg, aes(x=month, y=averageRating)) +
  geom_line() +
  geom_point(aes(size = count)) +
  geom_line(aes(x=month, y=fittedline), color = "red") +
  ylim(3, 4.5) +
  geom_hline(aes(yintercept = mu), color="gray") +
  labs(title = "Monthly Average Rating",
       x = "",
       y = "Average Rating",
       size = "Number of Ratings") +
  scale_x_date(date_labels = "%b %Y", date_breaks="6 month") +
  theme(axis.text.x=element_text(angle=60, hjust=1))
```



It is seen that in 1995 the averages are few and varies wildly. However we have seen earlier that only two ratings were given in 1995 so the start of the graph has no significant influence on the rest.

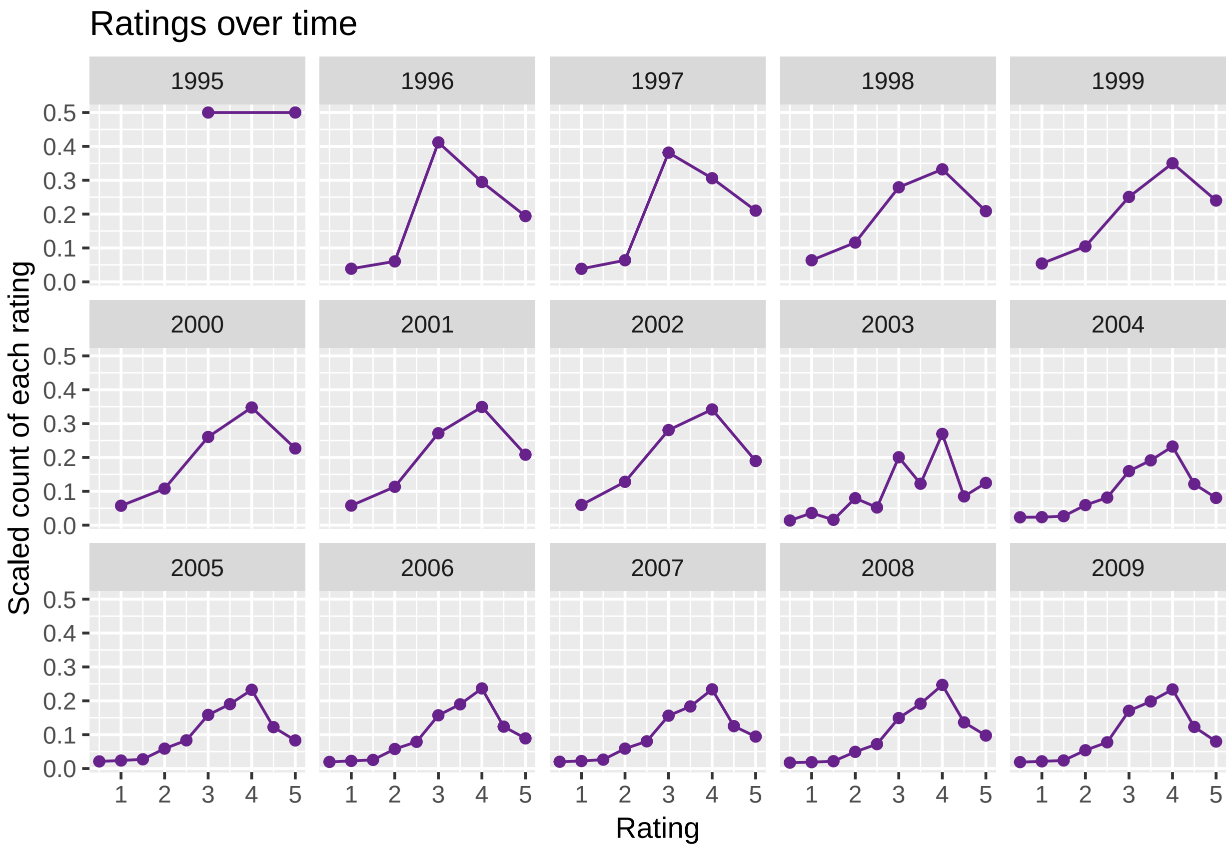
The monthly averages of the ratings do decrease over time as shown by the fitted linear regression, the red line. The average dots themselves, however, show a clear tendency to decrease between Oct 1999 and Oct 2004 whereafter they begin to increase again. A linear fit might not be appropriate here. Also the fluctuations are small compared to the half-integer steps between the ratings. All in all it seems to be reasonable to assume that the average of the ratings are constant over time.

To see if the ratings have the samme pattern over time I draw the following images, one for each year with ratings. Because the number of given ratings differs from year to year I divide the number of each rating by the total number of ratings that year. This way I get a comparable ratings pattern for each year.

```
edxRatingsOverTime <- edx %>%
    mutate(tsYear = year(ratingDate)) %>%
    group_by(tsYear, rating) %>%
    summarize(count = n()) %>%
    group_by(tsYear) %>%
    mutate(sum = sum(count)) %>%
    ungroup() %>%
    mutate(rate = count / sum)

head(edxRatingsOverTime)
```

```
## # A tibble: 6 x 5
##   tsYear rating  count    sum   rate
##    <dbl>  <dbl>  <int>  <int>  <dbl>
## 1   1995      3      1      2 0.5
## 2   1995      5      1      2 0.5
## 3   1996      1  36507 942772 0.0387
## 4   1996      2  56953 942772 0.0604
## 5   1996      3 388303 942772 0.412
## 6   1996      4 277971 942772 0.295
```

```
edxRatingsOverTime %>%
    ggplot(aes(x = rating, y = rate)) +
        geom_point(color = "darkorchid4") +
        geom_line(color = "darkorchid4") +
        facet_wrap( ~ tsYear, ncol=5) +
        labs(title = "Ratings over time",
           y = "Scaled count of each rating",
           x = "Rating") +
        theme_grey()
```



From the above images we see that the ratings pattern do change over time. The changes are not happening gradually but seems to appear in a few abrupt changes. I guess that the ratings are given using different guidelines of how to rate movies or perhaps the data is collected from totally different sources with each their scale of meaning for the rating values.

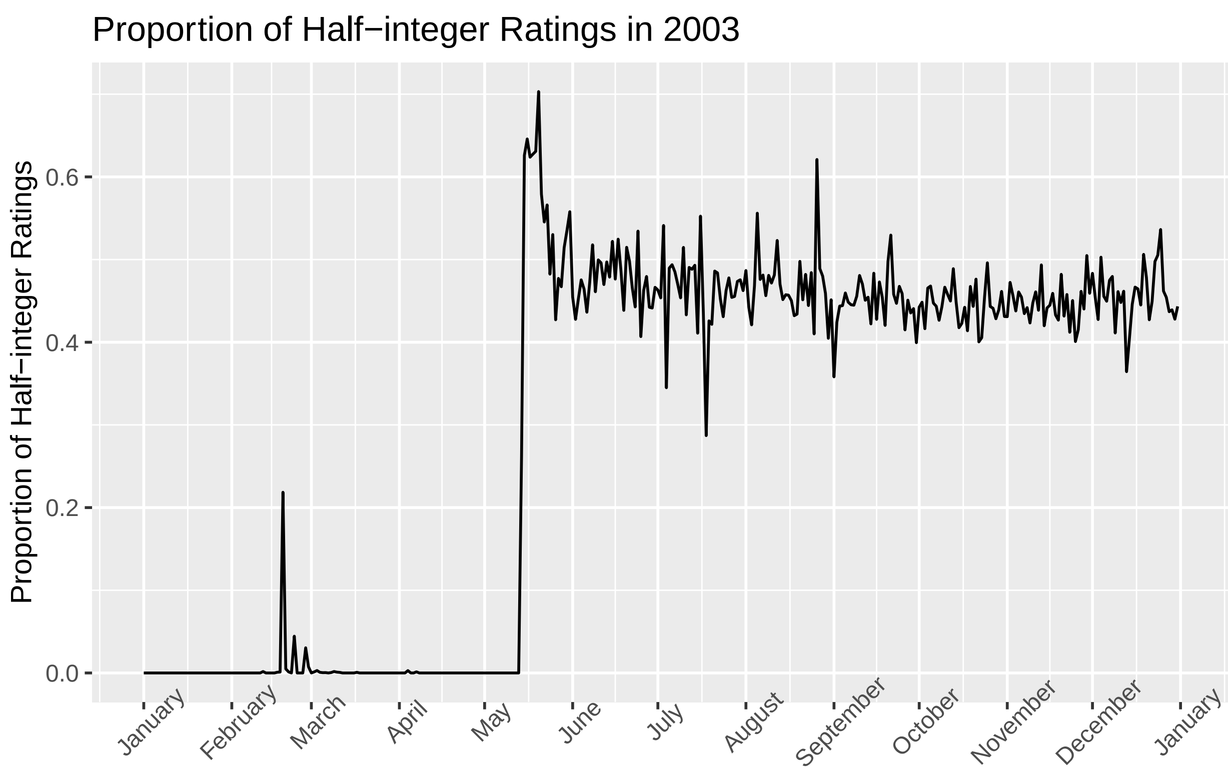
We only have two ratings in 1995 as seen earlier which means that the ratings pattern for 1995 tells us nearly nothing. The ratings pattern for 1996 and 1997 seems very similar with a mode rating of 3. In 1998 the ratings pattern has changed to have the mode rating 4, and this rating pattern is seen until somewhere around 2003 where half-integer ratings are also given which again changes the ratings pattern.

It is also seen that before 2003 all rating where integers but in and after 2003 both integer and half-integer ratings were given. In 2003 the number of half-integer ratings are less than the number of the neighbouring integer ratings. This is probably because the half-integer ratings were not introduced from the beginning of 2003.

To find the exact timestamp when half-integer ratings where introduced, I look at the following graph that shows the proportion of half-integer ratings to integer ratings during 2003.

```
edxHalfintRatingsProportion <- edx %>%
    filter(as.Date("2003-01-01") <= ratingDate & ratingDate <= as.Date("2003-12-31")) %>%
    mutate(ratingType = if_else(round(2*rating) %% 2 == 0, "int", "half")) %>%
    group_by(ratingDate) %>%
    count(ratingType) %>%
    spread(ratingType, n) %>%
    replace_na(list(half = 0, int = 0)) %>%
    mutate(proportion = half / (half + int))

ggplot(data = edxHalfintRatingsProportion, aes(x = ratingDate, y = proportion)) +
```

```
    scale_x_date(date_breaks = "1 month", date_labels = "%B") +
    labs(title = "Proportion of Half-integer Ratings in 2003",
       y = "Proportion of Half-integer Ratings",
       x = "") +
    theme(axis.text.x = element_text(angle = 45)) +
    geom_line()
```



Half-integer rating seems to have been introduced in February 2003 but were very uncommon until the middle of May where half-integer ratings get nearly as commonly used as integer ratings. The half-integer rating was probably being tested by only a few users during part of february.

To find the exact time when the big change in May happened I find the first half-integer rating in May and if it is close the middle of the month it will be it:

```
edx %>%
    filter(as.Date("2003-05-01") <= ratingDate & ratingDate <= as.Date("2003-05-31")) %>%
    filter(round(2*rating) %% 2 != 0) %>%
    arrange(timestamp) %>%
    top_n(-5, wt=timestamp)
```

```
##   userId movieId rating  timestamp                 title
## 1  71331    6220    4.5 1052944896        Willard (2003)
## 2  71331    6281    3.5 1052944933    Phone Booth (2002)
## 3  71331    5219    2.5 1052945349  Resident Evil (2002)
## 4  71331    5528    4.5 1052945382 One Hour Photo (2002)
## 5  71331    5901    3.5 1052945418         Empire (2002)
```

```
##                            genres ratingDate
## 1                         Horror 2003-05-14
## 2                 Drama|Thriller 2003-05-14
## 3 Action|Horror|Sci-Fi|Thriller 2003-05-14
## 4                 Drama|Thriller 2003-05-14
## 5                    Crime|Drama 2003-05-14
```

This shows that the first half-integer rating, except for a few test ratings in February 2003, happened at the timestamp 1052944896 corresponding to May 14, 2003.

The Models

Model 1: Average

As seen earlier the average of the ratings are close to being constant over time and this gives the first, primitive, model. Model 1 simply always predicts the average of all the ratings:

$$Y_{u,i} = \mu + \epsilon_{u,i} \qquad\qquad \text{model 1}$$

where $Y_{u,i}$ is the rating that user u has given or would give to movie i.

Building the model looks like this in R:

```
mu <- mean(edx$rating)

mu
```

```
## [1] 3.512465
```

Model 2: Movie Popularity

Model 2 builds on model 1 and also takes into account that some movies on the average are rated higher than other movies. This gives following model 2:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i} \qquad\qquad \text{model 2}$$

where b_i is a measure for the popularity of movie i, i.e. the bias of movie i.

I compute b_i for each movie i as the mean of the still unexplained residues $\epsilon_{u,i}$. In R I approximate b_i by:

```
moviePopularity <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

glimpse(moviePopularity)
```

```
## Observations: 10,677
## Variables: 2
## $ movieId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,...
## $ b_i     <dbl> 0.41517246, -0.30706581, -0.36548171, -0.64816590, -0....
```

Model 3: User Mildness

Model 3 builds on model 2 and also takes into account that some different users have different average ratings. This gives following model 3:

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i} \qquad \text{model 3}$$

where b_u is a measure for the mildness of user u, i.e. the bias of user u.

I compute b_u for each user u as the mean of the still unexplained residues $\epsilon_{u,i}$. In R I approximate b_u by:

```
userMildness <- edx %>%
  left_join(moviePopularity, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

glimpse(userMildness)
```

```
## Observations: 69,878
## Variables: 2
## $ userId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, ...
## $ b_u    <dbl> 1.679234705, -0.236408562, 0.264330314, 0.652078062, 0....
```

Model 4: Genre Popularity

Model 4 builds on Model 3 and also takes into account that different users like or dislike different genres. This gives the following model 4:

$$Y_{u,i} = \mu + b_i + b_u + b_{u,g} + \epsilon_{u,i} \qquad \text{model 4}$$

where $b_{u,g}$ is a measure for how much a user u likes the genre g.

I compute $b_{u,g}$ for each user u and genre g as the mean of the still unexplained residues $\epsilon_{u,i}$. In R I approximate $b_{u,g}$ by:

It is a bit more complex to build than the other models. First I split each movie into one row for each genre. Then I compute $b_{u,g}$ for each user u and genre g as the mean of the part of the rating that is not explained by the overall mean μ, the movie popularity b_i and the user mildness b_u.

```
genrePopularity <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  left_join(moviePopularity, by='movieId') %>%
  left_join(userMildness, by='userId') %>%
  group_by(userId, genres) %>%
  summarize(b_ug = mean(rating - mu - b_i - b_u))

glimpse(genrePopularity)
```

```
## Observations: 1,100,988
## Variables: 3
## $ userId <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2...
## $ genres <chr> "Action", "Adventure", "Animation", "Children", "Comedy...
## $ b_ug   <dbl> 0.06513179, -0.19180025, -0.28374731, -0.07270949, 0.18...
```

For the next I need a list of all the genres:

```
genresList <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  distinct(genres) %>%
  .$genres

genresList
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

The validation will be much easier with a row for all combinations of user u and genre i where $b_{u,g} = 0$ for all combinations not rated by the user.

I find that by using the R crossing method to generate all user, genre combinations and set $b_{u,g} = 0$ for all rows. Then I remove all user, genre combinations that already exist in `genrePopularity` and add them again from `genrePopularity` including the earlier computed values of $b_{u,g}$.

```
genrePopularity <- crossing(userId = edx$userId, genres = genresList) %>%
  mutate(b_ug = 0.0) %>%
  anti_join(genrePopularity, by=c('userId', 'genres')) %>%
  bind_rows(genrePopularity)

glimpse(genrePopularity)
```

```
## Observations: 1,397,560
## Variables: 3
## $ userId <int> 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3...
## $ genres <chr> "(no genres listed)", "Documentary", "Film-Noir", "Horr...
## $ b_ug   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
```

Results

The Validation Metric

As mentioned earlier, the metric to use for validation is the root mean square error (RMSE) for which I use:

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

An earlier version of this project used accuracy as its metric for validation. In the RMSE metric we predict real-number values where a close-by prediction is better than a farther-away prediction. The accuracy metric, on the other hand, requires us to predict a categorical rating, i.e. one of the possible ratings 0.5, 1.0, 1.5, . . . , 5.0, and it is only counted as correct if it is the precise same rating as the given rating.

Out of personal interest I decide to also calculate the accuracy metric for my model(s) but I only optimize my model(s) for the RMSE metric.

The functions necessary for this are as follows:

```
roundRatings <- function(ratings, timestamps) {
  isIntOnly <- validation$timestamp < 1052944896
  mult <- ifelse(isIntOnly, 1, 2)  # 1: only integer ratings, 2: also half-integer ratings
  minValue <- ifelse(isIntOnly, 1.0, 0.5) # 1: only integer ratings, 0.5: also half-integer ratings
  rounded_ratings <- round(mult * ratings, 0) / mult
  rounded_ratings <- ifelse(rounded_ratings < minValue, minValue, rounded_ratings)
  rounded_ratings <- ifelse(5.0 < rounded_ratings, 5.0, rounded_ratings)
  return(rounded_ratings)
}
```

The `roundRatings` function rounds a list of real-numbered predictions to the actually allowed predictions like 0.5, 1.0, . . . , 5.0. In the function I make use of the previously demonstrated fact that half-integer rating were only given after a certain date, as I showed earlier. If the resulting prediction is lower, or higher, than the allowed lowest, or highest, rating I change the prediction to the lowest, or highest, allowed rating.

Then the function `accuracy` just has to compare the predicted rating with the true rating and count how many are equal:

```
accuracy <- function(true_ratings, predicted_ratings) {
  mean(true_ratings == predicted_ratings)
}
```

Validate Model 1

To predict the ratings in `validation` I use:

```
# Predict ratings in the validation dataset
predicted_ratings1 <- validation %>%
  mutate(pred = mu) %>%
  .$pred

head(predicted_ratings1)
```

```
## [1] 3.512465 3.512465 3.512465 3.512465 3.512465 3.512465
```

Model 1 is, as expected, seen to predict the same value μ all the time.

For the accuracy metric I also compute the ratings rounded to categorial ratings:

```
predicted_rounded_ratings1 <- roundRatings(predicted_ratings1, validation$timestamp)

head(predicted_rounded_ratings1)
```

```
## [1] 4 4 4 4 4 4
```

And finally I compute the RMSE and accuracy:

```r
cat("Model 1: RMSE = ", RMSE(validation$rating, predicted_ratings1), "\n")
```

```
## Model 1: RMSE =  1.061202
```

```r
cat("Model 1: Accuracy = ", accuracy(validation$rating, predicted_rounded_ratings1), "\n")
```

```
## Model 1: Accuracy =  0.2672923
```

Validate Model 2

Predicting with model 2 is nearly as simple as with model 1:

```r
# Predict ratings in the validation dataset
predicted_ratings2 <- validation %>%
  left_join(moviePopularity, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

head(predicted_ratings2)
```

```
## [1] 2.935121 3.663522 3.055652 3.530058 4.415366 2.945278
```

Continuing the same way as for model 1:

```r
predicted_rounded_ratings2 <- roundRatings(predicted_ratings2, validation$timestamp)

head(predicted_rounded_ratings2)
```

```
## [1] 3 4 3 4 4 3
```

And finally I compute the RMSE and accuracy:

```r
cat("Model 2: RMSE = ", RMSE(validation$rating, predicted_ratings2), "\n")
```

```
## Model 2: RMSE =  0.9439087
```

```r
cat("Model 2: Accuracy = ", accuracy(validation$rating, predicted_rounded_ratings2), "\n")
```

```
## Model 2: Accuracy =  0.3223333
```

Validate Model 3

I predict and validate with model 3 as with the other models except for the extra joining of `userMildness`:

```r
# Predict ratings in the validation dataset
predicted_ratings3 <- validation %>%
  left_join(moviePopularity, by='movieId') %>%
  left_join(userMildness, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

head(predicted_ratings3)
```

```
## [1] 4.614356 5.342757 4.734887 3.293650 4.178957 2.708870
```

```r
predicted_rounded_ratings3 <- roundRatings(predicted_ratings3, validation$timestamp)

head(predicted_rounded_ratings3)
```

```
## [1] 5 5 5 3 4 3
```

And finally I compute the RMSE and accuracy:

```r
cat("Model 3: RMSE = ", RMSE(validation$rating, predicted_ratings3), "\n")
```

```
## Model 3: RMSE =  0.8653488
```

```r
cat("Model 3: Accuracy = ", accuracy(validation$rating, predicted_rounded_ratings3), "\n")
```

```
## Model 3: Accuracy =  0.3592804
```

Validate Model 4

Predicting with model 4 is requires just a little more work the the other models because we need to take the mean of the genre popularities for the genres of a movie:

```r
# Predict ratings in the validation dataset
predicted_ratings4 <- validation %>%
  separate_rows(genres, sep = "\\|") %>%
  left_join(genrePopularity, by=c("userId", "genres")) %>%
  group_by(userId, movieId) %>%
  summarize(b_ug = mean(b_ug)) %>%
  left_join(moviePopularity, by='movieId') %>%
  left_join(userMildness, by='userId') %>%
  mutate(pred = mu + b_i + b_u + b_ug) %>%
  .$pred

head(predicted_ratings4)
```

```
## [1] 4.801687 5.292735 4.792198 3.351059 4.290540 2.694559
```

```r
predicted_rounded_ratings4 <- roundRatings(predicted_ratings4, validation$timestamp)

head(predicted_rounded_ratings4)
```

```
## [1] 5 5 5 3 4 3
```

And finally I compute the RMSE and accuracy:

```r
cat("Model 4: RMSE = ", RMSE(validation$rating, predicted_ratings4), "\n")
```

```
## Model 4: RMSE =  0.8497552
```

```r
cat("Model 4: Accuracy = ", accuracy(validation$rating, predicted_rounded_ratings4), "\n")
```

```
## Model 4: Accuracy =  0.3666484
```

Conclusion

I have created four models of increasing complexity and size and with decreasing RMSE, i.e. increasing prediction precision. Model 3, User Mildness, predicts better than asked for.

Model 4, Genre Popularity, predicts yet better but is much larger than model 3. Unless the increased prediction is very important, I will recommend to use model 3. Model 3 predicts well and it has a reasonable size.

Model 4 can be made a little smaller by including the user mildness effect into the genre popularity effect. This would decrease the size of model 4 by 69878 out of 1478116, i.e. by 4.7%.

In the analysis I showed that the average rating value is nearly constant over time. The models build on the assumptions that also the movie popularity, the user mildness and the genre popularity are constant over time. Further work could be done to investigate if this is true, and if not, the changes over time could be build into the models.

The accuracy metric of the presented models is not impressive. The best model, model 4, only predicts a little more than every third rating correct. More work could be done to investigate how far off the predictions are. Also analysing the worst predictions might give inspiration to new and better models.