HarvardX: PH125.9x Data Science: Capstone - MovieLens Project

Martin Weihrauch

January 30, 2019

Contents

1 Overview

This is my project report for the MovieLens project of the HarvardX: PH125.9x Data Science: Capstone course. The report begins by introducing the general idea and problem of the project and by stating its aims. After dataset preparation and setup, exploratory data analysis is carried out to gain the necessary insights into the data in order to develop a machine learning algorithm that predicts movie ratings. Next, the general modelling approach is broken down step-by-step and ends with the final model and its results in terms of prediction accuracy. This is then followed by a discussion section where strengths and weaknesses of the generated modelling approach are broken down and alternative approaches are briefly discussed. Finally, the report ends with some concluding remarks.

1.1 Introduction

In 2006, Netflix offered a million US-Dollar price to whomever was able to surpass the performance of their movie recommendation algorithm by at least 10%. This goal was achieved by Team BellKor's Pragmatic Chaos in 2009, after a nearly 3-year long contest. Recommendation systems are a common occurrence nowadays and companies, big and small, employ them to increase sales numbers. It doesn't matter whether it is a new book on Amazon you might like to read, or the next popular song you might want to listen to on Spotify. Movie recommendation systems work by knowing what a user likes (e.g. movie genres, actors) to then recommend more similar movies. It can become extremely challenging to recommend a movie to a user who really loves war dramas such as "Saving Private Ryan", but at the same time also really likes pacifist movies (or really hates one or a few particular war dramas, but loves all the rest). The more a recommendation system knows about what users like, the better a recommendation it can make. It is not the purpose of this project to give a "Top 10" best movies to watch recommendation list for a particular user, but instead to make predictions about the star rating users would give to different movies.

1.2 Aim of the Project

The aim of this project is to build a machine learning algorithm that predicts user ratings (from 0.5 to 5 stars) for movies based on the dataset MovieLens with 10 million user ratings of over 10500 different movies. The algorithm is to be trained on a provided training subset (`edx`) of MovieLens and evaluation is to be carried out on the provided testing subset (`validation`). The metric used to evaluate algorithm performance is the residual mean squared error, or RMSE. This represents the typical error in star ratings the predicted rating would be off by.

1.3 Dataset Preparation

The "MovieLens" dataset is automatically downloaded and prepared for analysis by splitting it into a training (`edx`, 90% of the data) and a test set (`validation`, 10% of the data). All necessary R packages are downloaded and loaded to execute the analysis. Algorithm development is to be carried out on the `edx` subset only, as `validation` will only be used to test the final algorithm.

2 Methods and Analysis

2.1 Exploratory Data Analysis

As a first step, we want to get familiar with the dataset. A quick glance at the first few entries reveals that each row represents a single rating of a user for a single movie, including the movie title, genre and timestamp at which the rating was given.

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

A quick summary of the dataset tells us that there are no missing values to deal with (e.g. via a value imputation technique).

```
##      userId         movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08  
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09  
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##     title              genres         
##  Length:9000055     Length:9000055    
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
##                                       
##                                       
##                                       
```

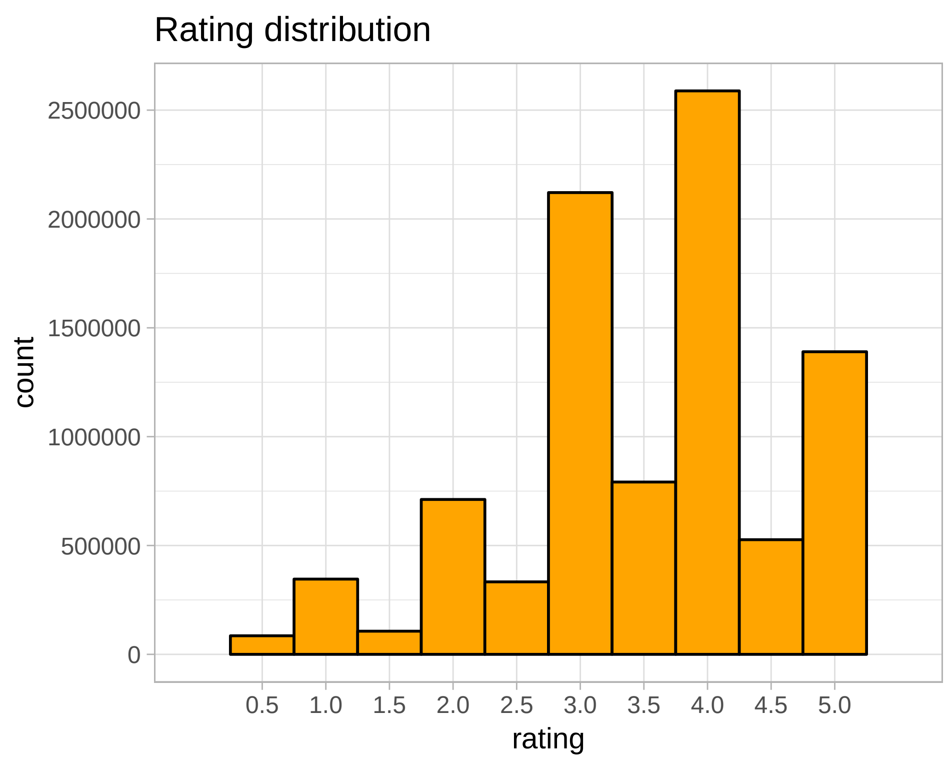
We are dealing with over 9 million movie ratings of ~70000 unique users giving ratings to ~ 10700 different movies.

```
##   n_users n_movies
## 1   69878    10677
```

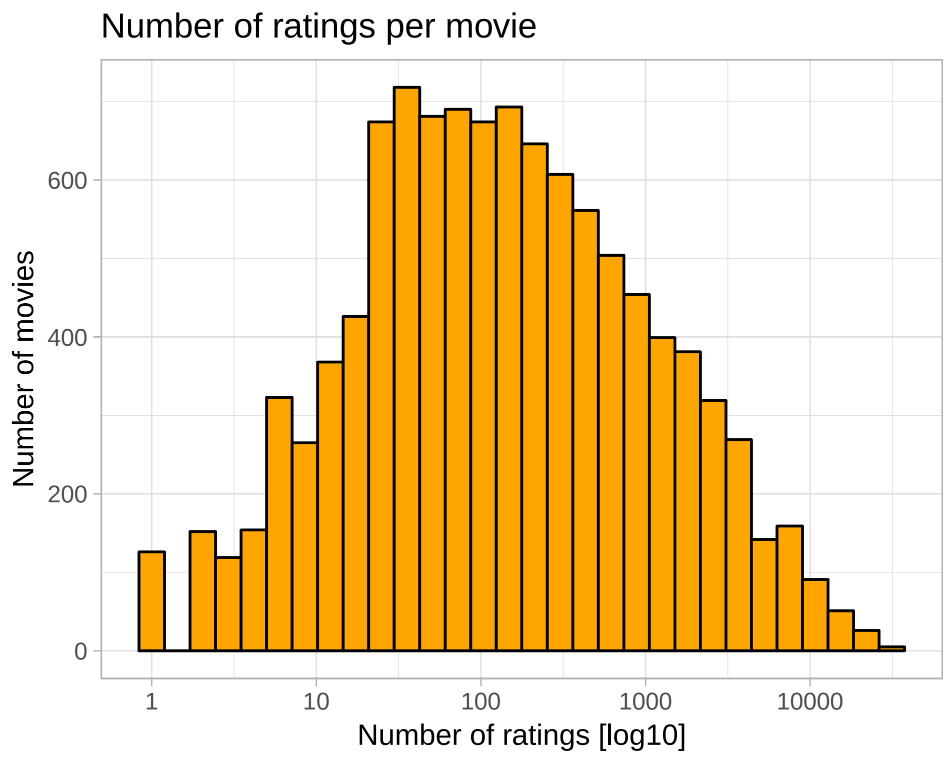
We take a look at the distribution of ratings: 4 is the most common rating, followed by 3 and 5. 0.5 is the least common rating. It becomes apparent that half-star ratings are much less common than full-star ratings.

```
##    rating       n
## 1     4.0 2588430
## 2     3.0 2121240
## 3     5.0 1390114
## 4     3.5  791624
## 5     2.0  711422
## 6     4.5  526736
## 7     1.0  345679
## 8     2.5  333010
## 9     1.5  106426
## 10    0.5   85374
```

To further strengthen our intuition about the rating distribution, we take a look at a histrogram of it.



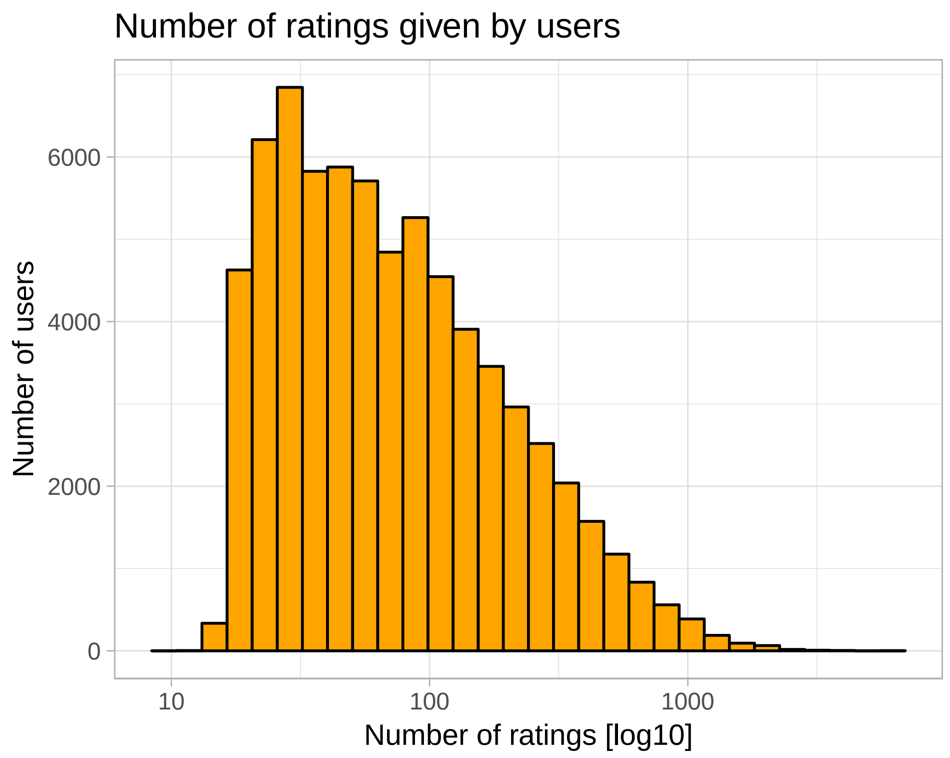
We take a look at the number of times different movies have been rated. We can observe that some movies have been rated much more often than others, while some have gotten very few and sometimes only a single rating. This will be important to consider for our modelling approach, as very low rating numbers might give untrustworthy estimates for predictions.



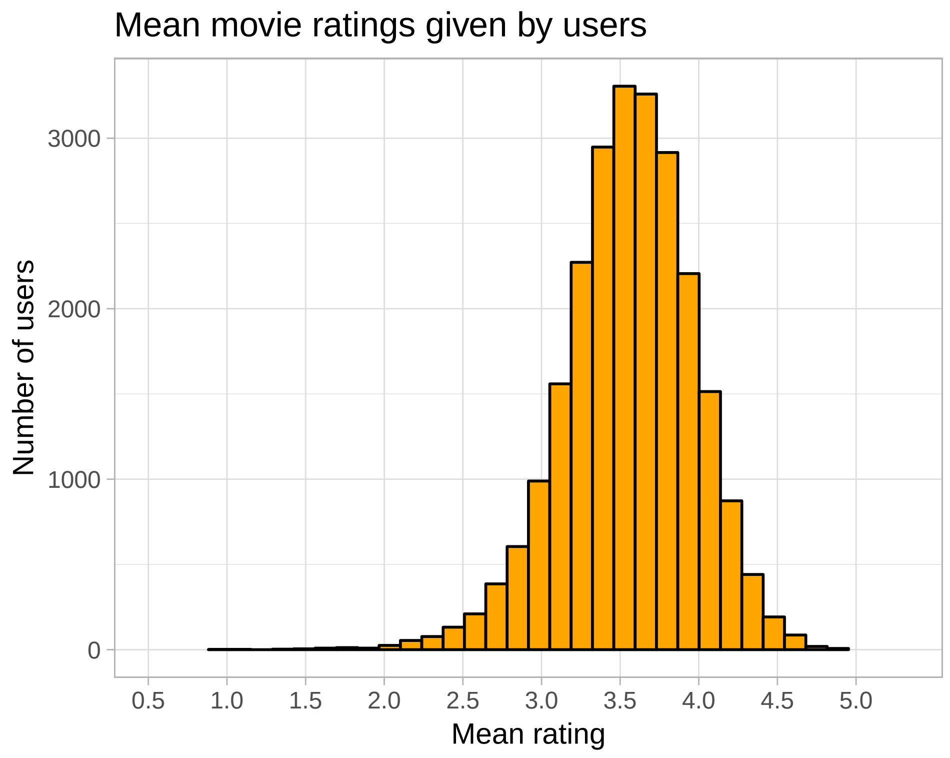
What are some of these movies with very low ratings? We take a look at 20 movies that only received a single rating. As expected, these movies appear to be rather obscure, and predictions of future ratings for these will be difficult.

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di…) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1

Next we will explore user rating behaviour. Some users were very active and rated a large amount of movies, while others only rated a few.



Furthermore, users differ vastly in how critical they are with their ratings. Some users tend to give much lower star ratings and some users tend to give higher star ratings than average. The visualization below includes only users that have rated at least 100 movies.



We take a look at the representation of movie genres, their average ratings and the difference of their average rating from μ ("norm_avg_rating"). We can see that dramas and comedies are quite overrepresented. Some genres clearly get rated better or worse than others, for example Film-Noir (on average getting a 4-star rating), war movies, documentaries and IMAX movies are highly appraised, while horror movies are frowned upon. However, modelling these differences between genres is potentially difficult, as most movies tend to have more than just one genre associated with them and certain effects might cancel eachother out, or worse, mislead the prediction. While the Film-Noir genre has a mean rating of 4, there are still many movies getting as low as 0.5 stars as well.

```
##                genres   count avg_rating norm_avg_rating
## 1               Drama 3910127   3.673131      0.16066549
## 2              Comedy 3540930   3.436908     -0.07555710
## 3              Action 2560545   3.421405     -0.09106058
## 4            Thriller 2325899   3.507676     -0.00478946
## 5           Adventure 1908892   3.493544     -0.01892130
## 6             Romance 1712100   3.553813      0.04134824
## 7              Sci-Fi 1341183   3.395743     -0.11672204
## 8               Crime 1327715   3.665925      0.15346009
## 9             Fantasy  925637   3.501946     -0.01051897
## 10           Children  737994   3.418715     -0.09374974
## 11             Horror  691485   3.269815     -0.24265024
## 12            Mystery  568332   3.677001      0.16453609
## 13                War  511147   3.780813      0.26834736
## 14          Animation  467168   3.600644      0.08817846
## 15            Musical  433080   3.563305      0.05083950
## 16            Western  189394   3.555918      0.04345262
## 17          Film-Noir  118541   4.011625      0.49915947
## 18        Documentary   93066   3.783487      0.27102179
## 19               IMAX    8181   3.767693      0.25522823
## 20 (no genres listed)       7   3.642857      0.13039194
```

2.2 Modelling Approach

We begin by writing a loss-function that computes the Residual Mean Squared Error (RMSE, or "typical error") as our measure of model accuracy. The value is the typical error in star rating we would make upon predicting a movie rating. $y_{u,i}$ here represents the rating for movie i given by user u and N stands for the number of user/movie combinations. We denote our rating predictions from user u for movie i as $\hat{y}_{u,i}$. For the remainder of the report, we will drop the "hat" notation (as in $\hat{y}$, denoting estimation) from the presented equations.

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

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

2.2.1 I. The Simplest Model: Average Movie Rating

After splitting `edx` into training and testing subsets, we begin by building a very simple model: We predict a new movie rating to be the average rating of all movies in our training dataset, completely disregarding the user who gives it. This gives us our baseline RMSE to compare future modelling approaches against. We observe that the mean movie rating μ is a pretty generous > 3.5 stars, quite a bit above the actual average (i.e. 2.5 stars). Our simplest model predicts $Y_{u,i}$ as μ plus the independent errors $\epsilon_{u,i}$, which are sampled from the same distribution and are centered at 0. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone.

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

We create a table to record our approaches and the RMSEs they generate. We can see that by predicting a new rating to be the average rating we are typically off by over one star in rating (RMSE > 1.06)!

Method	RMSE
Average rating	1.061202

But can we do better than simply predicting the average rating? We will begin by incorporating some of the insights we gained during the exploratory data analysis.

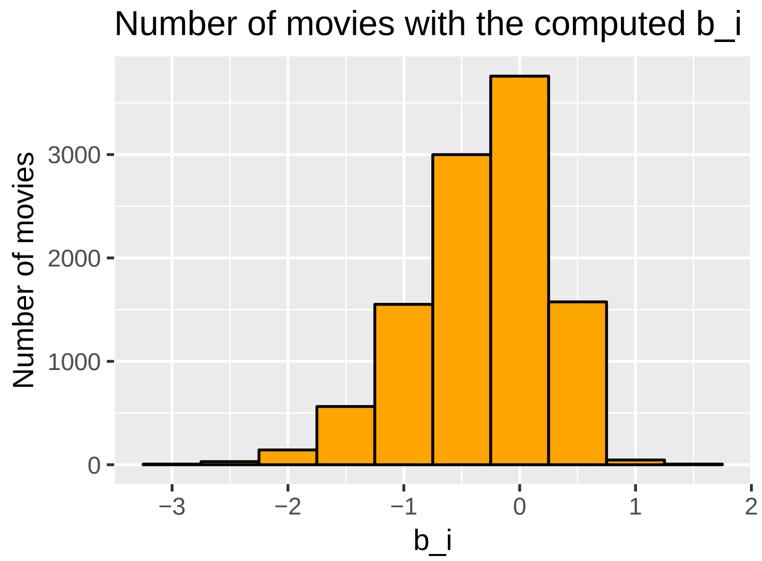
2.2.2 II. Improving the Simplest Model by Incorporating the "Movie-effect"

To improve upon the model, we utilize the fact that movies are actually rated differently: Some movies are simply better or more popular than others, which is reflected in their rating being higher or lower than the average movie rating μ. We compute the estimated deviation of each movies' mean rating from the total mean of all movies μ. We call the resulting variable "b" (as in "bias") for each movie "i": $\hat{b}_i$, which represents the average ranking of movie i. We add this term to our model equation from above.

Our model then becomes:

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

When plotting our computed b_i, we can see that there are large differences in how movies are rated. A very small amount of movies has a large positive b_i, denoting that these movies are on average one or more star ratings above μ. Some movies got a large negative b_i, denoting that on average these movies received 2 or even 3 stars less than μ. Expectedly, most movies received a more or less neutral b_i (around 0), which denotes that these movies were estimated to be just about average.



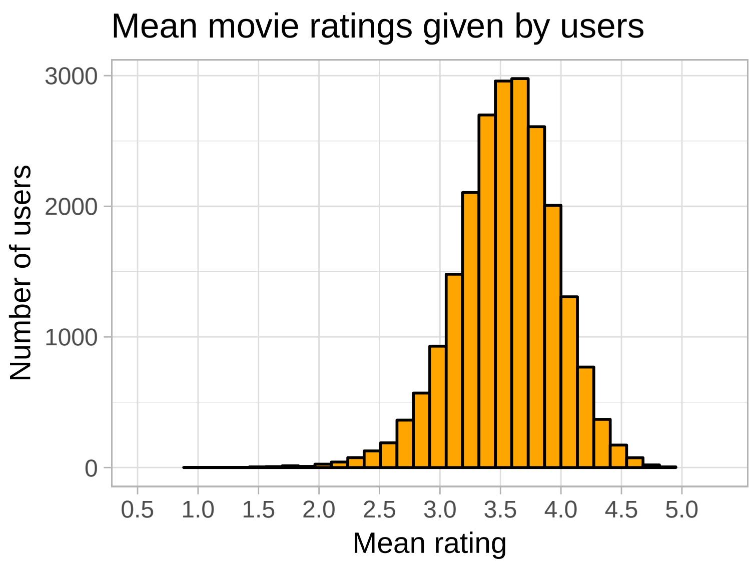
We then predict movie ratings based on the fact that movies are rated differently by adding the computed b_i to μ. If an individual movie is on average rated worse than the average rating of all movies μ, we predict that it will be rated lower than μ by b_i, the difference of the individual movie average from the total average.

Method	RMSE
Average rating	1.0612018
Movie Effect	0.9429615

We can see that the RMSE improves when we take into account that different movies are rated differently. But this model is still in total disregard of individual user rating patterns. We will use some of the insights gained about user rating distribution to further improve upon our model.

2.2.3 III. Augmenting the Model with the "User-effect"

Do users rate different movies differently? We compute b_u, the "User-effect". We can see that some users rate movies generally higher/lower than others, while most fall in-between, but also that user rating of movies is generally higher than a 2.5 "true average" rating. This was reflected previously in the high mean rating of > 3.5. We then predict the ratings taking into account movie and user effects together.



Our model then becomes:

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

And the corresponding RMSE value:

Method	RMSE
Average rating	1.0612018
Movie Effect	0.9429615
Combined Movie & User Effects	0.8646843

We can see that including the user-effect b_u in our rating predictions further reduced the RMSE. It appears that we are still off by ~0.865 stars on average. Are we correctly predicting the best and worst movies with our model? The top ten best and worst movies according to our predictions so far appear to include some "obscure" entries. This might be due to an overall low amount of ratings associated with these movies. We take a look at the amount of ratings given to them. We look at the top ten best movies according to our predictions so far (largest positive b_i), as well as at the number of ratings they received.

title	b_i	n
Hellhounds on My Trail (1999)	1.487544	1
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487544	1
Shadows of Forgotten Ancestors (1964)	1.487544	1
Fighting Elegy (Kenka erejii) (1966)	1.487544	1
Sun Alley (Sonnenallee) (1999)	1.487544	1
Blue Light, The (Das Blaue Licht) (1932)	1.487544	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237544	4
Life of Oharu, The (Saikaku ichidai onna) (1952)	1.237544	2
Human Condition II, The (Ningen no joken II) (1959)	1.237544	4
Human Condition III, The (Ningen no joken III) (1961)	1.237544	4

The top ten worst movies according to our predictions so far (largest negative b_i).

title	b_i	n
Besotted (2001)	-3.012456	1
Hi-Line, The (1999)	-3.012456	1
Accused (Anklaget) (2005)	-3.012456	1
Confessions of a Superhero (2007)	-3.012456	1
War of the Worlds 2: The Next Wave (2008)	-3.012456	2
SuperBabies: Baby Geniuses 2 (2004)	-2.767775	47
Disaster Movie (2008)	-2.745789	30
From Justin to Kelly (2003)	-2.638139	183
Hip Hop Witch, Da (2000)	-2.603365	11
Criminals (1996)	-2.512456	1

Indeed, some of the best and worst movies we predict were rated sparsely. Larger estimates of b_i are likely for movies with very few ratings. The same holds true for the user-effect b_u, in those cases where users only rated a very small number of movies. We can penalize these by making use of regularization. We determine the value `Lambda` that minimizes RMSE, employing 3-way cross-validation. This shrinks the b_i and b_u in case of small number of ratings. Essentially, by shrinking our estimates when we are rather unsure, we are being more conservative in our estimations.

We then calculate the regularized movie-effect b_i using the optimised `Lambda` value. We also apply the same approach to the user-effect b_u.

We then predict the ratings with the regularized movie- and user-effects.

Method	RMSE
Average rating	1.0612018
Movie Effect	0.9429615
Combined Movie & User Effects	0.8646843
Combined, regularized Movie & User Effects on `test_set`	0.8641562

Regularization had little impact on the RMSE, but did it improve the best and worst movies we predict? We take a look at the 10 best movies after regularization.

title	b_i	n
Shawshank Redemption, The (1994)	0.9440266	25188
Godfather, The (1972)	0.9040680	15975
Usual Suspects, The (1995)	0.8539975	19457
Schindler's List (1993)	0.8515375	20877
More (1998)	0.8394262	6
Rear Window (1954)	0.8122348	7115
Casablanca (1942)	0.8070083	10141
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.8028477	2633
Double Indemnity (1944)	0.7940640	1950
Seven Samurai (Shichinin no samurai) (1954)	0.7928834	4648

And at the 10 worst movies after regularization.

title	b_i	n
SuperBabies: Baby Geniuses 2 (2004)	-2.641328	47
From Justin to Kelly (2003)	-2.606097	183

title	b_i	n
Disaster Movie (2008)	-2.554222	30
PokÃ©mon Heroes (2003)	-2.428075	124
Barney's Great Adventure (1998)	-2.344312	186
Glitter (2001)	-2.322023	311
Carnosaur 3: Primal Species (1996)	-2.312408	61
Gigli (2003)	-2.296558	281
Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie) (2002)	-2.283005	188
Faces of Death 6 (1996)	-2.204774	73

Indeed, the movies we predict to be the best/worst make much more sense now and are based on a larger number of ratings. Before the final evaluation, we predict ratings on `validation` and calculate the RMSE of the algorithm on it. We find that the algorithm is doing a little bit worse on `validation` than it did when using it on the `test_set` we had generated from `edx`.

Method	RMSE
Average rating	1.0612018
Movie Effect	0.9429615
Combined Movie & User Effects	0.8646843
Combined, regularized Movie & User Effects on `test_set`	0.8641562
Combined regularized Movie & User Effect on `validation`	0.8652517

3 Results

While the predicted ratings have a similar mean compared to the original ratings, we observe values below 0.5 and above 5 due to the way we calculated them. Furthermore, we predicted numeric values and not categorical ratings from 0.5 to 5. In order to predict star ratings from 0.5 to 5, we have to round our predictions and substitute values above 5 and below 0.5 accordingly. Here is the accuracy of our predictions in star ratings from 0.5 to 5 after rounding.

```
## [1] 0.2478142
```

Here is the final RMSE value of the predicted ratings without rounding.

```
## [1] 0.8652517
```

Method	RMSE
Average rating	1.0612018
Movie Effect	0.9429615
Combined Movie & User Effects	0.8646843
Combined, regularized Movie & User Effects on `test_set`	0.8641562
Combined regularized Movie & User Effect on `validation`	0.8652517

We arrive at an accuracy of almost 24.8% and an RMSE of 0.8648490 when modelling with the regularized movie and user effects and predicting on `validation`.

4 Discussion

4.1 The Final Model

Equation of the final model:

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

The final model presented here performs reasonably well for the average user, that is when a user does not rate a particularly good/popular movie with a large positive b_i unexpectedly low (dislikes a particular movie or genre/actor etc…). The model is performing well when predicting a "critical" user (negative b_u) rating a good/popular movie (positive b_i). However, if a usually "critical" user (negative b_u) simply loves a particular average movie (neutral b_i), our prediction will be quite far off (the user might give a 5, but we predict a rating lower than μ because the movie is just average in terms of its b_i). Some users simply love every movie they rate, giving a full 5-star rating to every single one, resulting in a large b_u in our predictions. Nevertheless, if this user rates below average movies (low or negative b_i), our prediction might not quite reach the full 5-star rating. Incorporating individual user preferences into the model would most likely lead to large improvements in prediction accuracy.

4.2 Other Approaches

In the beginning the accuracy metric for this project was set on predicting the correct star ratings between 0.5 and 5 stars. Modelling the movie- and user-effects as done here only returned about 25% accuracy. By making use of the fact that half-star ratings are much less common than full star ratings, we could round to full-star ratings only and forfeit prediction of any half-star ratings. This would bring up the prediction accuray to around 37% (but never predict a half-star rating at all). Incorporating genres and release year information in analogue fashion to user- and movie-effects yielded only negligible improvements of RMSE and star rating accuracy and were therefore removed from the modelling process of this report. By utilizing the "recommenderlab" package, a user-based collaborative filtering model was built by generating a sparse rating matrix as input. However, once the built model was used to make predictions, RAM became a limiting factor and the predictions could not be finished. By utilizing "ff", "ffbase", and "biglm" R-packages, it was attempted to run a linear model, but RAM issues persisted and these models could never be run successfully. Principal component analysis (PCA) and singular value decomposition (SVD) were attempted, but failed due to RAM constraints.

As predicting star ratings from 0.5 to 5 can be seen as a classification rather than a regression problem, it was attempted to train different random forrest classifier algorithms on the dataset. However, once again the size of the dataset made running them almost impossible. Almost, because the "ranger" R-package actually successfully trained a random forrest with default parameters (running for many hours). The prediction accuracy was at around 33%, not really improving upon the model from above. It was attempted to split the training dataset into many smaller parts, training different algorithms and combining results later, but no such approach yielded appreciable accuracy values.

An approach using naive bayes yielded approximately 34% accuracy when modelling with userId and movieId. Naive bayes assumes independence of the predictors, which in this scenario is not the case. In order to further improve prediction accuracy, it would have been necessary to incorporate user preferences into the equation. Due to any matrix factorization attempts failing, user preference were not successfully modeled and the approach was stopped.

5 Conclusion

For this capstone project, we've built a machine learning algorithm to predict movie ratings with the MovieLens dataset. The ~25% prediction accuracy value of the model presented here can be explained by several facts. User preference for certain types of movies, genres, or even particular actors and directors most likely plays a major role in determining the ratings given. In fact, many modelling approaches in the Netflix challenge incorporated more user information than what was given here and it was found that e.g. user profession has a large predictive value. Other possible approaches would include item- and/or user-based collaborative filtering, to incorporate individual user preferences into the model and thus most likely lead to a large increase in prediction accuracy. By far the largest challenge was the sheer size of the dataset itself. The usual approaches, for example fitting a simple linear model with the `lm()` function or fitting a random forest of decision trees with the `train()` function from the `caret` package will most likely fail due to RAM constraints. As every unique movieId and userId can be seen as a predictor, these kinds of models quickly turn too huge to handle for regular personal computers. It would certainly be possible to work with this dataset in a more big data environment, for example when working with Apache Spark via the `sparklyr` interface.

6 Acknowledgements

I would like to thank the entire edx course staff for providing a very stimulating learning environment throughout all of the courses and for making these learning materials available and exciting. I thank all the active edx forum users for providing help and ample discussion. And finally, a big thank you to my peers for spending their time and effort to review my capstone projects. Thank you very much, everyone.

```r
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 16299)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=German_Switzerland.1252  LC_CTYPE=German_Switzerland.1252
## [3] LC_MONETARY=German_Switzerland.1252 LC_NUMERIC=C
## [5] LC_TIME=German_Switzerland.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] bindrcpp_0.2.2  vtreat_1.3.4    caret_6.0-81    lattice_0.20-35
##  [5] forcats_0.3.0   stringr_1.3.1   dplyr_0.7.8     purrr_0.2.5
##  [9] readr_1.3.0     tidyr_0.8.2     tibble_1.4.2    ggplot2_3.1.0
## [13] tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.0         lubridate_1.7.4    wrapr_1.8.0
##  [4] class_7.3-14       assertthat_0.2.0   digest_0.6.18
##  [7] ipred_0.9-8        foreach_1.4.4      R6_2.3.0
## [10] cellranger_1.1.0   plyr_1.8.4         backports_1.1.3
## [13] stats4_3.5.1       evaluate_0.12      highr_0.7
## [16] httr_1.4.0         pillar_1.3.1       rlang_0.3.0.1
## [19] lazyeval_0.2.1     readxl_1.1.0       rstudioapi_0.8
## [22] data.table_1.11.8  rpart_4.1-13       Matrix_1.2-14
## [25] rmarkdown_1.11     labeling_0.3       splines_3.5.1
## [28] gower_0.1.2        munsell_0.5.0      broom_0.5.1
## [31] compiler_3.5.1     modelr_0.1.2       xfun_0.4
## [34] pkgconfig_2.0.2    htmltools_0.3.6    nnet_7.3-12
## [37] tidyselect_0.2.5   prodlim_2018.04.18 codetools_0.2-15
## [40] crayon_1.3.4       withr_2.1.2        MASS_7.3-50
## [43] recipes_0.1.4      ModelMetrics_1.2.2 grid_3.5.1
## [46] nlme_3.1-137       jsonlite_1.6       gtable_0.2.0
## [49] magrittr_1.5       scales_1.0.0       cli_1.0.1
## [52] stringi_1.2.4      reshape2_1.4.3     timeDate_3043.102
## [55] xml2_1.2.0         generics_0.0.2     lava_1.6.4
## [58] iterators_1.0.10   tools_3.5.1        glue_1.3.0
## [61] hms_0.4.2          parallel_3.5.1     survival_2.42-3
## [64] yaml_2.2.0         colorspace_1.3-2   rvest_0.3.2
## [67] knitr_1.21         bindr_0.1.1        haven_2.0.0
```