Movie Rating Predictions

Nick Aristidou

15/11/2021

Movie Recommender

Abstract

This research project investigates the creation of a recommendation system for movies, where Naive Bayes models and a Matrix Factorization model are employed. To compare the models the Residual Mean Square Error is calculated. The best Naive Bayes model achieved a RMSE of 0.84 on the test set and a RMSE of 0.825 on the validation set, whilst the matrix factorization model achieved a RMSE of 0.79 on the test set.

Introduction

The aim of this of this research capstone is to generate a movie recommendation system, using the MovieLens dataset. The data used herein is produced by a research lab from the University of Minnesota, GroupLens, whom have compiled a collection of millions of movie ratings. Specifically, for this project the 10M version is utilized. The project will explore the data and then will aim to create a model that given a user will predict the rating for a movie. To evaluate model performance the metric Root Mean Squared Error (RMSE) will be employed, which is determined by the square root of the residual sum of squares from comparing predictions $\hat{y}$ with observed outcomes y for each user u and each item (movie) i:

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

Data loading and partitioning into a train, test and validation split.

To create a model that can predict ratings, the data set needs to broken down into a train, test and validation split. Machine learning requires this split so that a model can be fitted on a sample of the data, the train set, has a subset of data to provide an unbiased evaluation of a model fit and tune hyperparameters, the validation set. Finally there also needs to be a sample of data to provide an unbiased evaluation of the final model, the test set.

Special consideration to the creation of these sets is required for a recommendation system as the model needs to learn user and movie features, therefore each set must contain the userId and movieId for each user and movie in the original data. The following code, loads the data creates the first partition into the train and validation sets, whilst ensuring each set contains an occurrence of every userId and movieId:

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
```

```
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Further partitioning of the train data to create the final data sample, the test set.

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

# Make sure userId and movieId in  test set are also in train set
test <- temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

# Add rows removed from testset back into train set
removed <- anti_join(temp, test)
train <- rbind(train, removed)

rm(test_index, temp, removed)
```

The final sets of data are of the following form: 1. Train set - 6 columns and 7200089 rows. 2. Validation set - 6 columns and 999993 rows. 3. Test set - 6 columns and 1799972 rows.

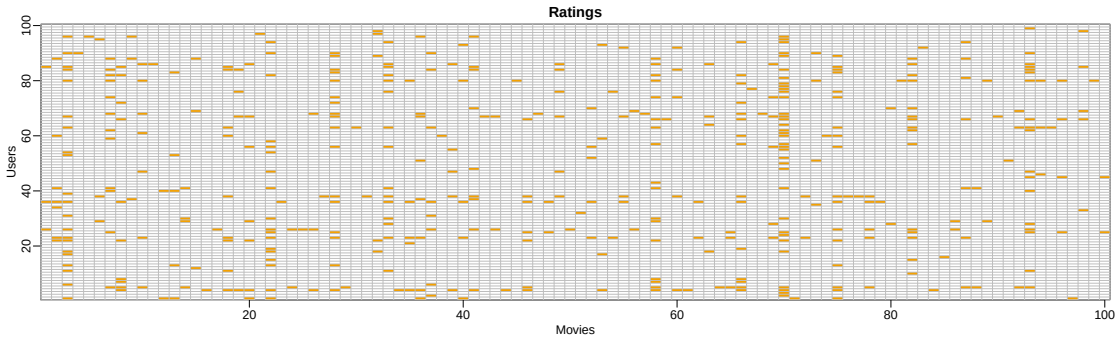
Exploratory Data Analysis and Visualisation

To begin the following table shows the data is comprised of the columns userId, movieId, rating, timestamp, title and genres and depicts the type of information stored within. It is noted that the data contains 69878

unique user ids and 10677 unique movie ids. Furthermore the ratings range from 0.5 up to 5, with increments of 0.5.

```
## Rows: 7,200,089
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, ~
## $ movieId   <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 370, 377, 420, ~
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 3, ~
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 838983392, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumber (1994)~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy", "Action~
```

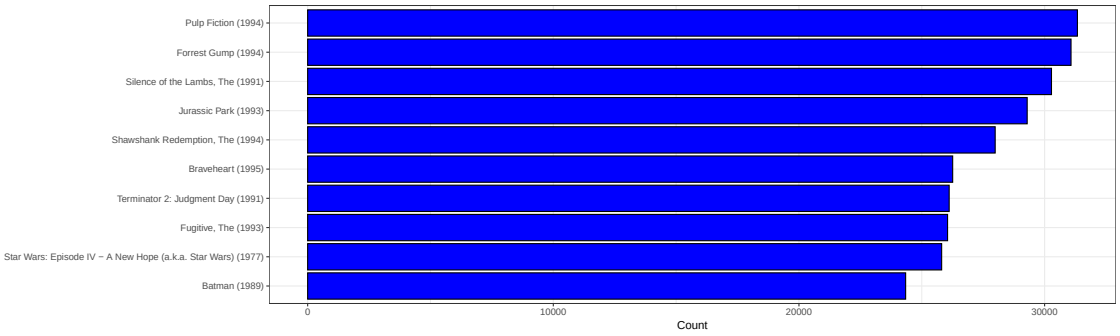
A useful way to visualize the task of a recommendation system is to generate a rating matrix grid, which transforms the data into users as rows and movies as columns. Each cell is then occupied by the rating that a given user has reviewed a given movie, whilst unseen/unrated movies remain blank. The figure below takes a subset of 100 users to generate this rating matrix grid and shows the issue of sparsity. Most users will only have rated a small number of movies out of the entire movie library.



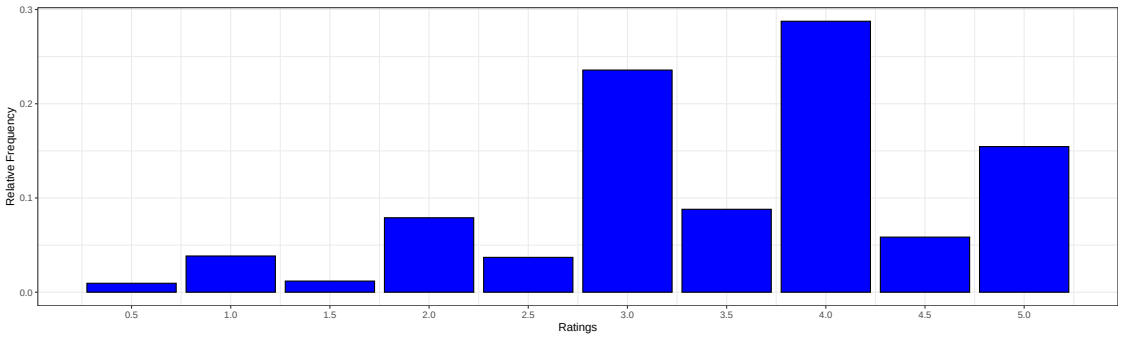
To further explore the data the following sections will examine effects and trends in the data associated with the movies, users, time and the genre of the movie.

Exploring Movie Effect

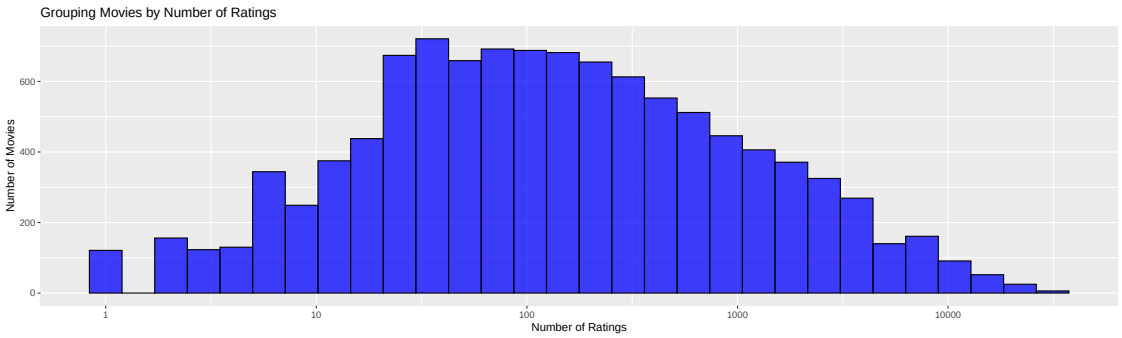
The top 10 most rated movies in the data set are:



The distribution of movie ratings is as follows and shows that a rating of 4.0 is the most frequently given value.



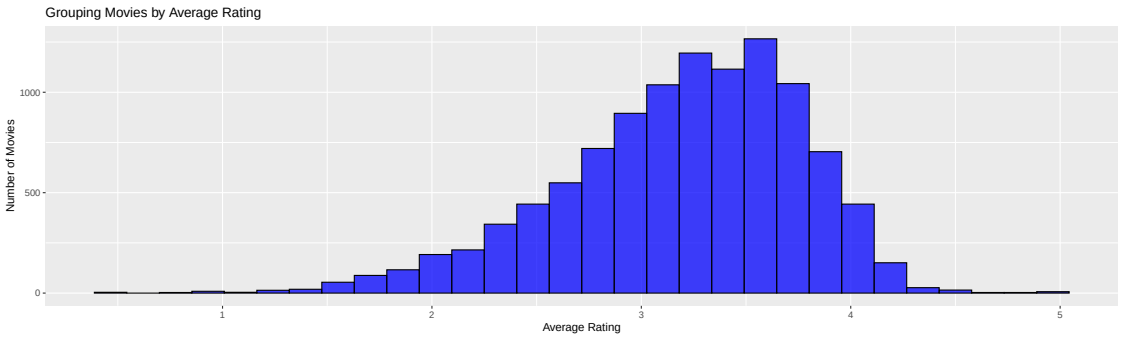
The number of ratings for a movie follows an approximately normal distribution, as shown below.



The top 10 most rated films and their average rating

movieId	title	No_Ratings	Average
296	Pulp Fiction (1994)	31336	4.161731
356	Forrest Gump (1994)	31076	4.010265
593	Silence of the Lambs, The (1991)	30280	4.205086
480	Jurassic Park (1993)	29291	3.658189
318	Shawshank Redemption, The (1994)	27988	4.456928
110	Braveheart (1995)	26258	4.083194
589	Terminator 2: Judgment Day (1991)	26115	3.928394
457	Fugitive, The (1993)	26050	4.005969
260	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25809	4.218567
592	Batman (1989)	24343	3.389291

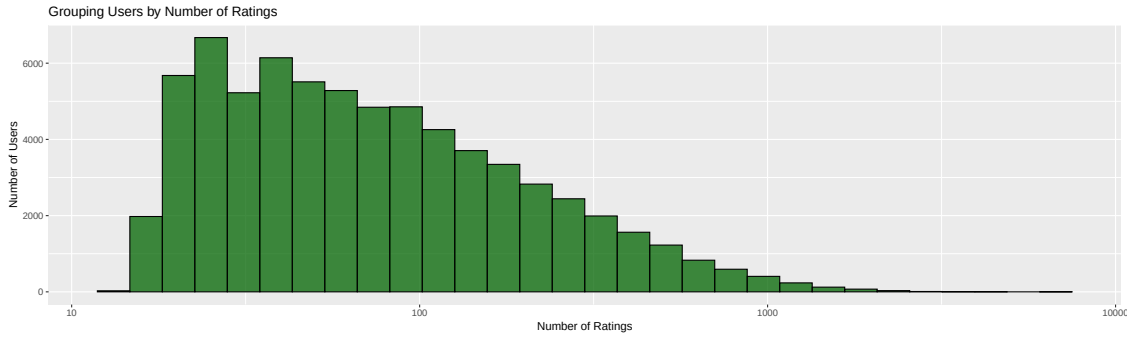
The average rating distribution for movies is also approximately normal, with the average movie rating at 3.512464.



From the above visuals it is noted that the movieId will be critical in generating rating predictions as each movie will effect the outcome. For example a movie that is always consistently rated highly should therefore obtain a higher rating from the model. However, the visuals also highlighted that the range in the number of ratings a movie will receive varies, with some movies receiving many and others receiving very few. A range of factors could be at play here, a movie could be very new in the data and thus has not been watched/rated by many users yet and contributes to a well known issue in recommendation systems known as the "cold-start" problem. Further more a film may not be attractive to the majority of audiences and covers a niche topic. It is likely these types of films will also receive higher ratings as they are more likely watched by users that will enjoy the content. The range in the number of ratings for a given movie is highly likely to generate a source of error in the models deployed and will contribute to larger values of RMSE.

User Effects Investigation

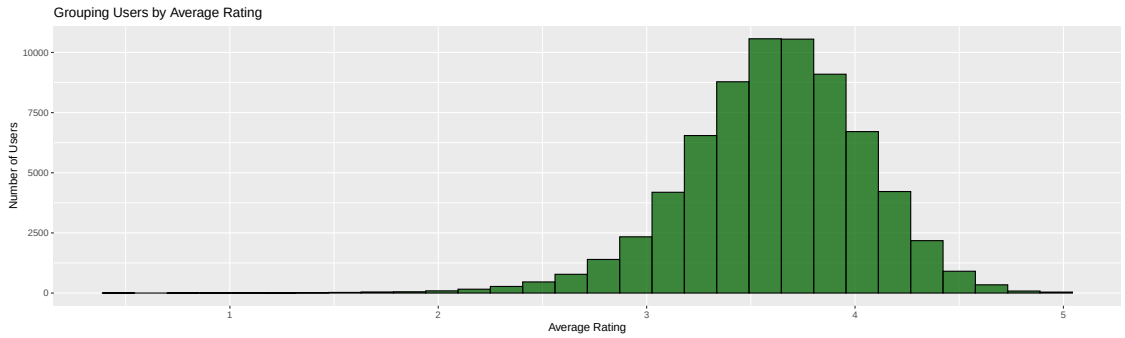
The figure below shows the distribution of the number of ratings given by users and exhibits a right skew.



The table below shows the Top 10 users by rating and their average movie rating.

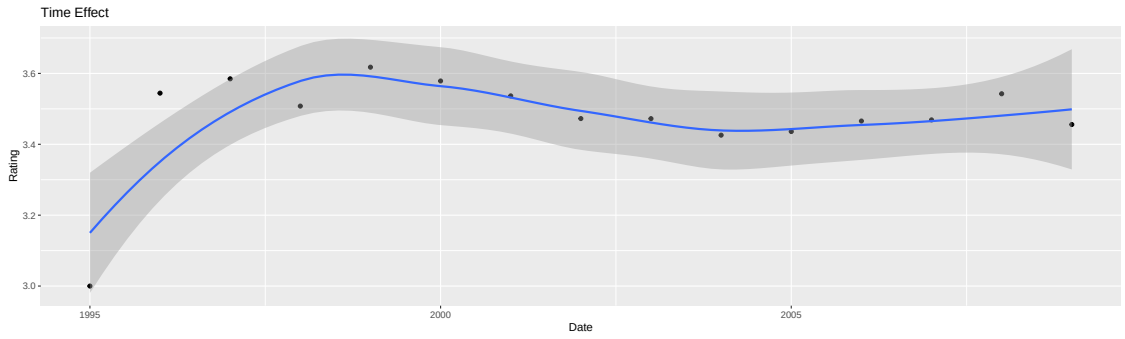
userId	No_Ratings	Average
59269	6637	3.264653
67385	6376	3.196910
14463	4637	2.406081
68259	4056	3.560651
27468	4018	3.831011
19635	3740	3.499064
3817	3736	3.108538
63134	3390	3.266814
58357	3318	2.998945
27584	3139	3.003026

The average user rating distribution is displayed below and also shows a normal distribution.

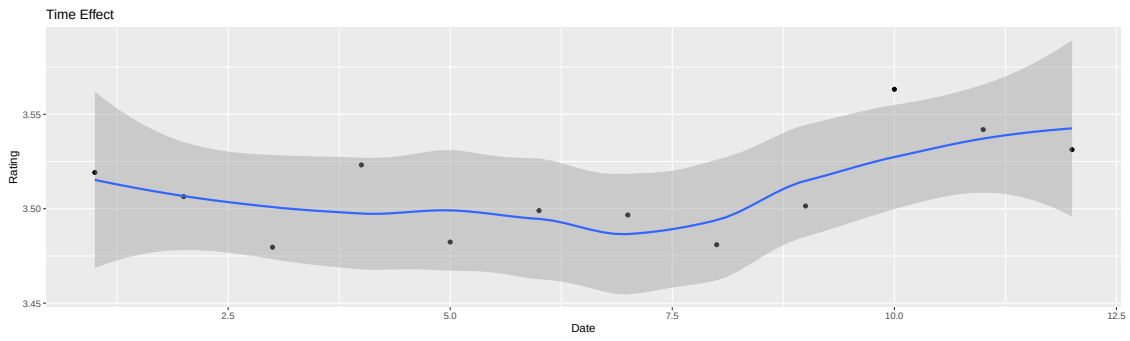


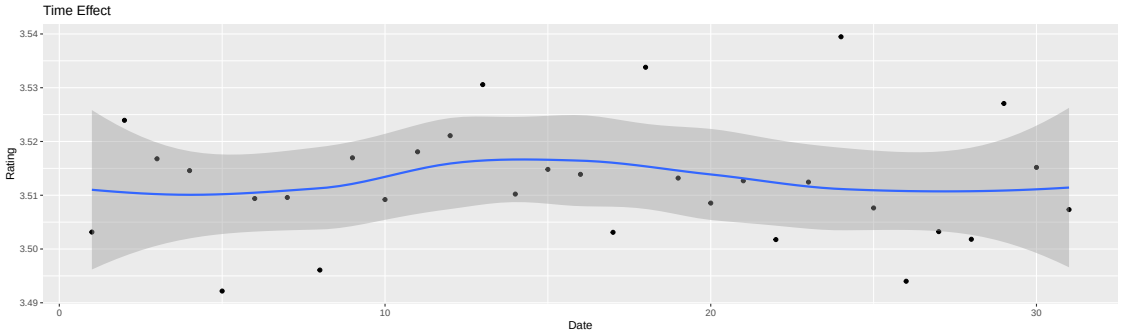
Time Effect

The timestamp field represents the number of seconds past midnight from 1 January 1, 1970. The following figure, shows average rating as function of the year a rating was given. As there is only one rating in 1995, which is a 5, this value can be treated as an outlier and if excluded it is noted that the avg rating by year is consistent and unlikely to prove useful as a predictor for rating.



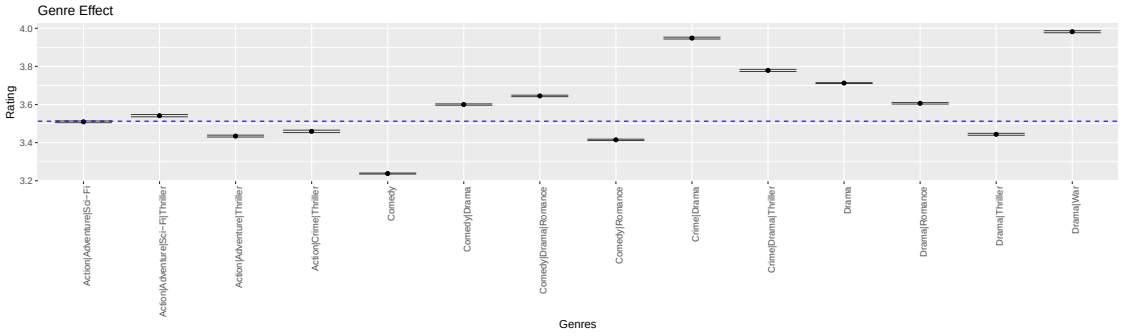
Likewise, further investigation shows that neither month nor the day of the rating considerable impact the average rating given. Both of these features are shown in the figures below, respectively.





Genre Effect

In contrast to the time effect, it is noted that the genre of the movie can impact the rating of the movie. This is show in the figure below, where the ratings as a function genre are displayed. This may indicate that genre could be a useful feature to include to help improve accuracy of the prediction model.



Model Building

Initial Prediction

```
## [1] "The average movie rating is 3.51237713867148"
```

Employing the simple approximation that a movie will be rated with the average movie rating, 3.5123771, generates a model that defaults to this value for every prediction. This model can be represented by the equation $\hat{Y}_{u,i} = \mu + \epsilon_{u,i}$, where $\epsilon_{u,i}$ is th error distribution. This model generates the following RMSE evaluation:

Method	RMSE
Naive Bayes	1.059643

Add movie effect term

To enhance the model, the effect of a movie can be included as this was noted to earlier to show a strong relationship to the rating. This movie effect $\hat{b}_i$ can be calculated as the mean of the difference between the observed rating y and the mean μ and takes the form:

$\hat{b}_i = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{\mu})$

With the overal model represented by: $\hat{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$

Method	RMSE
Naive Bayes	1.0596429
Movie Effect	0.9431724

Add user effect

Additionally from the previous EDA, it was shown that a given user had an impact on ratings, where a unique user has there own preferences and tastes and will rate a movie accordingly. As an example one user may consistently given ratings of 4 or 5, whilst another may given lower scores of 1 to 2. To account for this effect the model can be expanded to include the user term $\hat{b}_u$.

$\hat{b}_u = \frac{1}{N} \sum_{u,i=1}^{N} (y_{u,i} - \hat{b}_i - \hat{\mu})$

This generates the model of the form:

$\hat{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$

Method	RMSE
Naive Bayes	1.0596429
Movie Effect	0.9431724
Movie + User Effects	0.8655154

Add genres effect

Expanding the model to also include a genre effect as per the above logic leads to a model of the form:

$\hat{Y}_{u,i} = \mu + b_i + b_u + b_g + \epsilon_{u,i}$

Method	RMSE
Naive Bayes	1.0596429
Movie Effect	0.9431724
Movie + User Effects	0.8655154
Movie + User + Genres Effects	0.8662891

It is worth noting this has had little to no impact on the model, which may seem suprising since there appeared to be correlation between genre and rating. The lack of any significant impact on RMSE, may arise as this factor may be captured in the user term. A user is likely to have a favored genre and will rate thesse higher than films in other genres.

Add time

Here for research purposes only, the year of the review is also considered but noted that this will likely have no effect as per the fact that the timestamp had very little to no correlation to ratings.

Including a year effect modifies the model to the following:

$\hat{Y}_{u,i} = \mu + b_i + b_u + b_g + b_{yr} + \epsilon_{u,i}$

Method	RMSE
Naive Bayes	1.0596429
Movie Effect	0.9431724
Movie + User Effects	0.8655154
Movie + User + Genres Effects	0.8662891
Movie + User + Genres + Year Effects	0.8651462

Validate Model

To validate the model, the validation data set is employed and predicted ratings are produced. These predications are then compared to the actual ratings contained in this dataset and the RMSE is calculated to determine the success of the model. (A lower RMSE is a better model)

Method	RMSE
Naive Bayes	1.0596429
Movie Effect	0.9431724
Movie + User Effects	0.8655154
Movie + User + Genres Effects	0.8662891
Movie + User + Genres + Year Effects	0.8651462
Model Validation	0.8651755

Model Tuning - Regularization

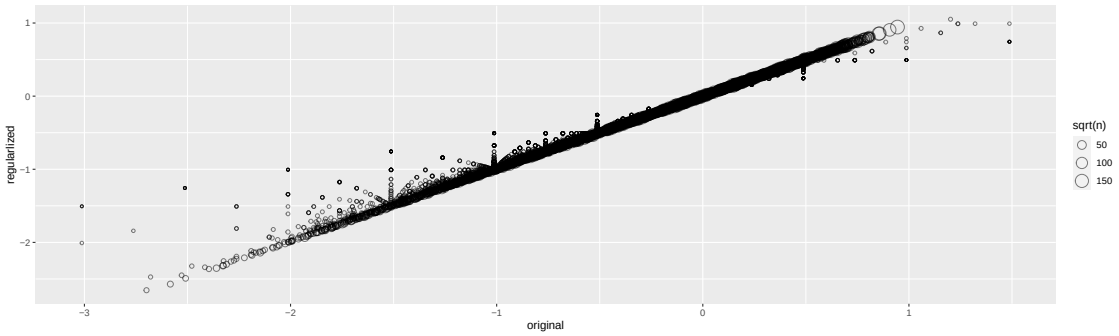
As previously described some users have only rated very few films, and some movies have very few ratings leading to small sample sizes for these instances. Statistically speaking this introduces a larger estimated error. The user and movie effect terms, $b_u and b_i$, can strongly influence predictions made by the model and error in these will cause the model performance to deteriorate. To combat this a technique known as regularization can be employed. This introduces a tuning parameter, λ, that can penalize distortions to these terms in essence by penalizing small sample sizes. For example the addition of λ modifies the movie effect term b_i to the following:

$\hat{b}_i = \frac{1}{n_i+\lambda} \sum_{u=1}^{n_i} (y_{u,i} - \hat{\mu})$

The following set of movies shown highlight this issue where some films have a low number of ratings and their movie effect term is significant.

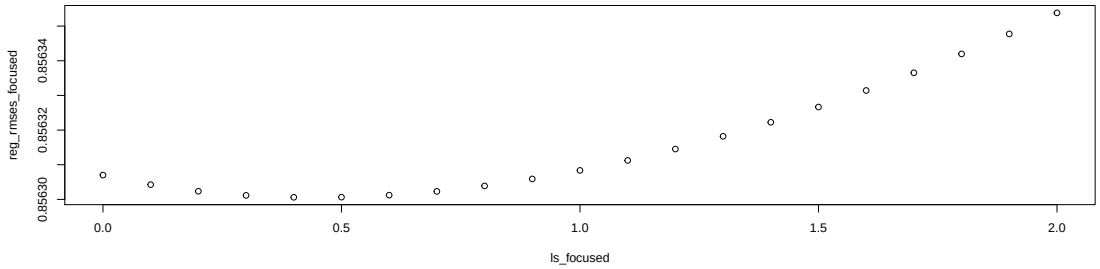
title	b_i	n
Hellhounds on My Trail (1999)	1.487623	1
Satan's Tango (Sátántangó) (1994)	1.487623	2
Shadows of Forgotten Ancestors (1964)	1.487623	1
Fighting Elegy (Kenka erejii) (1966)	1.487623	1
Sun Alley (Sonnenallee) (1999)	1.487623	1
Blue Light, The (Das Blaue Licht) (1932)	1.487623	1
Constantine's Sword (2007)	1.487623	1
Human Condition II, The (Ningen no joken II) (1959)	1.320956	3
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237623	4
Human Condition III, The (Ningen no joken III) (1961)	1.237623	4

To optimize the value of λ, multiple simulations a run against multiple values of λ:




```
ls_focused <- seq(0, 2, 0.1)

reg_rmses_focused <- sapply(ls_focused, function(l) {

   # Calculate the regularized avges for movie, user, genre, and time

   movie_avgs_reg <- edx %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu) / (n() + l))

   user_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - mu - b_i) / (n() + l))

    genre_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + l))

    year_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      left_join(genre_avgs_reg, by='genres') %>%
      group_by(Year) %>%
      summarize(b_yr = sum(rating - mu - b_i - b_u - b_g) / (n() + l))

   #predict the movie ratings on edx set
   pred_movie_user_genre_year_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      left_join(genre_avgs_reg, by='genres') %>%
      left_join(year_avgs_reg, by='Year') %>%
      mutate(pred = mu + b_i + b_u + b_g + b_yr) %>%
      pull(pred)

   #calculate RMSE
   RMSE(edx$rating, pred_movie_user_genre_year_reg)
})

# pick the final lambda
```

```
plot(ls_focused, reg_rmses_focused)
```



Method	RMSE
Naive Bayes	1.0596429
Movie Effect	0.9431724
Movie + User Effects	0.8655154
Movie + User + Genres Effects	0.8662891
Movie + User + Genres + Year Effects	0.8651462
Model Validation	0.8651755
Regularized Model	0.8253597

Model Evaluation - Test Set

Method	RMSE
Naive Bayes	1.0596429
Movie Effect	0.9431724
Movie + User Effects	0.8655154
Movie + User + Genres Effects	0.8662891
Movie + User + Genres + Year Effects	0.8651462
Model Validation	0.8651755
Regularized Model	0.8253597
Model Evaluation	0.8396412

Matrix Factorisation

One of the most widely used tools in recommendation systems is matrix factorisation, and was popularized in industrial applications after emerging as the winner from the Netflix prize. This method aims to fill in the rating matrix grid shown earlier by representing this large sparse matrix by the product two lower dimension matrices. The R library recosystem provides a method to decompose the matrix into these lower dimensions and then allow predictions to be made.

To demonstrate matrix factorization, the following code breaks down the steps the recosystem library is performing. It is worth noting there are many methods employed to solve the dimensionality reduction of the matrix and the example below is for illustrative purposes and may not replicate the method employed by the recosystem library. This example is also only run on a sample of 10 users and 10 movies only as this

operation is computational expensive on large datasets and to achieve the results that the library delivers requires the use of C programming languages to handle memory.

```
#Generate random sample of 10 users
users <- sample(unique(train$userId), 10)

# R is the rating matrix grid and
R<-
  train %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 10)) %>%
  as.matrix() %>% replace_na(0)
R
```

```
##       6365 2302 2580 1079 31 1394 39292 1372 442 3094
##  [1,]  3.5    0    0    0  0  2.5     4    3   0    0
##  [2,]  0.0    0    0    0  0  0.0     0    0   0    0
##  [3,]  0.0    0    0    0  0  0.0     0    0   4    0
##  [4,]  0.0    3    4    3  0  0.0     0    0   0    5
##  [5,]  0.0    0    0    0  0  0.0     0    0   0    0
##  [6,]  0.0    0    0    0  0  0.0     0    0   0    0
##  [7,]  0.0    0    0    0  3  0.0     0    0   3    0
##  [8,]  0.0    0    0    0  3  0.0     0    0   0    0
##  [9,]  0.0    0    0    0  0  0.0     0    0   0    0
## [10,]  0.0    0    0    0  0  0.0     0    0   0    0
```

Decompose matrix R into two lower dimension matricies P and Q:

```
N <- nrow(R)
M <- ncol(R)
#latent features:
K = 3

P <- matrix(rexp(N*K), N)
Q <- matrix(rexp(M*K), M)

output <- matrix_fact(R,P,Q,K)
output
```

```
## $P
##            [,1]         [,2]      [,3]
##  [1,] 0.5712096  2.448962777 0.2316313
##  [2,] 0.2615477  1.002701883 0.1315450
##  [3,] 4.4493609 -0.004207592 0.1911209
##  [4,] 0.7650730  1.423052327 0.7969167
##  [5,] 1.1038533  1.133030338 1.3304720
##  [6,] 1.2255110  0.775746243 0.2312279
##  [7,] 1.1329914  0.927088628 0.8383349
##  [8,] 0.9642716  0.147470631 1.6240200
##  [9,] 1.2423227  0.057992818 0.4042737
## [10,] 0.1170080  0.105293988 0.6991641
##
## $Q
```

```
##             [,1]        [,2]       [,3]
##  [1,] 1.3239552  1.10403803 0.12667328
##  [2,] 0.5559391  1.48385225 0.57094877
##  [3,] 1.8030361  1.30683795 0.94440788
##  [4,] 1.2290065  0.09982665 2.40132274
##  [5,] 2.3192046 -0.02992719 0.46826925
##  [6,] 0.9448293  0.78990168 0.07876483
##  [7,] 0.4062831  1.49892465 0.34965728
##  [8,] 1.0393880  0.92111056 0.61190647
##  [9,] 0.8393748  0.93071972 1.39823451
## [10,] 0.9198639  2.43249996 1.03085815
```

Combine P and Q to generate filled in matrix

```
mf_example <- output$P %*% t(output$Q)
mf_example
```

```
##             [,1]      [,2]     [,3]      [,4]       [,5]     [,6]      [,7]
##  [1,] 3.4893455 4.0837063 4.449063 1.5027135  1.3599272 2.492380 3.9838751
##  [2,] 1.4699616 1.7083715 1.906181 0.7374222  0.6381730 1.049515 1.6552326
##  [3,] 5.9103192 2.5764504 8.197356 5.9268164 10.4086006 4.215616 1.8682201
##  [4,] 2.6849744 2.9919320 3.991768 2.9959925  2.1049445 1.909704 2.7225322
##  [5,] 2.8808962 3.0545562 4.727483 4.6646422  3.1491725 2.042730 2.6120133
##  [6,] 2.5082655 1.9644216 3.441789 2.1388540  2.9272719 1.788875 1.7415401
##  [7,] 2.6297656 2.4841830 4.046109 3.4981145  2.9924602 1.868824 2.1430811
##  [8,] 1.6451855 1.6821331 3.465074 5.0996136  2.9924083 1.155475 1.1806650
##  [9,] 1.7600166 1.0075281 2.697539 2.5034034  3.0687740 1.251434 0.7330188
## [10,] 0.3597273 0.6204769 1.008868 1.8332333  0.5956113 0.248794 0.4498339
##             [,8]     [,9]    [,10]
##  [1,] 2.9912106 3.082632 6.721316
##  [2,] 1.2759420 1.336702 2.815265
##  [3,] 4.7376847 3.997997 4.279590
##  [4,] 2.5936347 3.080922 4.986846
##  [5,] 3.0051025 3.841392 5.143019
##  [6,] 2.1298193 2.073976 3.252669
##  [7,] 2.5445513 2.986053 4.161545
##  [8,] 2.1318373 3.217400 2.919855
##  [9,] 1.5920507 1.662019 1.700584
## [10,] 0.6464271 1.173808 1.084498
```

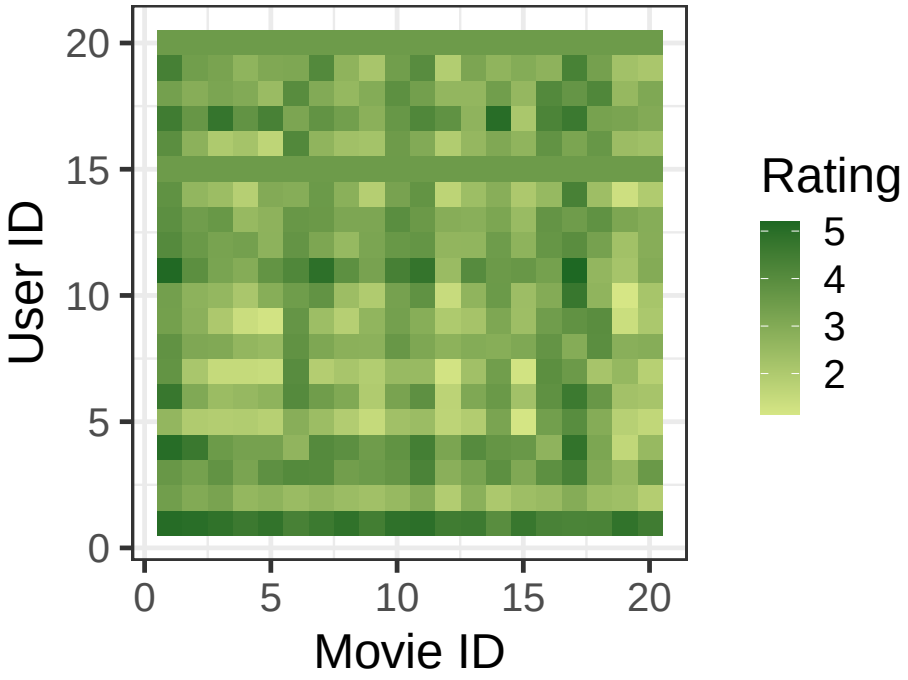
Only values that were previously blank are needed for the recommendation system to truly work as these are what would be used to decided if a user would like the movie/item.

Generate predictions against the test data set:

```
test_set= data_memory(test$userId, test$movieId, index1 = TRUE)
pred = r$predict(test_set, out_memory())
mf_rmse <- RMSE(pred,test$rating)

result <-
  result %>%
  add_row("Method" = "Matrix Factorization", "RMSE" = mf_rmse)

result %>% kbl(.,booktabs = T) %>% kable_styling(latex_options = c("striped","hold_position"))
```

Method	RMSE
Naive Bayes	1.0596429
Movie Effect	0.9431724
Movie + User Effects	0.8655154
Movie + User + Genres Effects	0.8662891
Movie + User + Genres + Year Effects	0.8651462
Model Validation	0.8651755
Regularized Model	0.8253597
Model Evaluation	0.8396412
Matrix Factorization	0.7941279

Heatmap to show how the rating matrix has now been populated with predictions:



Conclusion

In summary, it was noted by exploration of the MovieLens data that movie and the user had the greatest impact on the variability of the rating from the mean value. The initial approach to build a recommendation system was built upon deploying the average movie rating. This yielded and RMSE of 1.06, which fails to account for biases exhibited by users and biases fro movies. The basis model was expanded to include these terms in addition to the genre of the movie and the time of the rating. These additions improved the model performance as noted by the improved RMSE of ca. 0.865. It is noted that the majority of this improvement derives from the user and movie effect terms, however for exploratory purposes the genre and time terms were still included.

To enhance the model further, regularization was applied to help reduce the error introduced from small samples that occur when a movie has very few ratings or where a user has rated very few movies. The tuning parameter λ was optimized against a simulation aimed at minimizing RMSE. With the optimal

tuning parameter selected the model was deployed on the test set and a final RMSE value of 0.84 was achieved. This model reported a RMSE of 0.825 on the validation dataset.

Expanding beyond this, the project explored the use of matrix factorization to fill in the sparse rating matrix grid. Using this technique, a RMSE of 0.79 can be achieved.

Additional Comments

There are many challenges that affect recommendation systems, one of which has been the issue of the cold-start problem. Here recommenders struggle to deal with new users or new movies. The models employed herein would fail to predict a rating for these, and alternative methods of handling this issue need to be explored.

Furthermore, recommendation systems face ethical challenges and content based challenges were companies need to be careful of the recommendations made. For example the content of the item may not be suitable for the age of the user and filtering of their respective item libraries may be required. Diversity of recommendations is also another issue, were if a recommendation system keeps recommending very similar types of movies a user my never see new types of content. Here a potential solution is to ensure that in a top n recommendation a handful of the recommendations are diverse and are items unlikely to be seen. In a similar vein the preferences of a user may change with time, something they were interested in a year ago may not interest them today. Again solutions to this may involve the application of filtering the user catalogs to only look at recent recommendations and drop historical ones.

It is also worth noting that user/item databases can get very large very quickly and the concepts and application of Big Data may need to be applied to run and generate models. The computing power of one laptop/computer may be insufficient to compute the matrix factorization of a dataset of the likes of Amazon or Netflix.