Movie Lens Report

2023-12-03

Executive Summary

Recommendation systems are ubiquitous within the internet. The ability to predict with accuracy the wants and needs of a given user is central to many commonly used apps today. A more accurate recommendation system would mean better and more relevant suggestions for the user thereby maximizing their engagement with the product recommended. In this project we attempt to create a movie recommendation system based off the movie lens dataset.

This project consists of producing predicted ratings given a training set comprised of 9 million samples from the movielens 10M data set. Within this process we used exploratory statistics to examine the basic structure of the underlying data to establish the key relationships of the different variables provided. We then utilized these variables to construct a machine learning model that predicts the corresponding score for the given conditions. the ultimate goal of this project was to maximize the prediction accuracy through minimization of the RMSE of the combined test dataset.

Methods and Analysis

For this exercise we will construct a combined model consisting of a mix of linear fittings of residual variances and temporal trend fitting and interpolation using LOESS fitting. To prevent over fitting and compensate for different amount of data points for groups (especially those with very low data points) we will compensate for the bias using regularization of the data with the regularization factor lambda to be decided through training - testing minimization of the RMSE.

We first carry out the initial data processing into the final_holdout_test and edx data that will be used for fitting.

```
############################################################
# Create edx and final_holdout_test sets
############################################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)
```

```r
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

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

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

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

From this initial set up we obtain the datasets edx and final_holdout_test. The final_holdout_test is set aside for testing after training is complete.

Initial assessment

We first examine the overall dynamics of the data. The basic table arrangement:

```
head(edx, 10)
```

```
##    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)
## 8       1     356      5 838983653                       Forrest Gump (1994)
## 9       1     362      5 838984885                  Jungle Book, The (1994)
## 10      1     364      5 838983707                    Lion King, The (1994)
## 11      1     370      5 838984596 Naked Gun 33 1/3: The Final Insult (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
## 8                   Comedy|Drama|Romance|War
## 9                 Adventure|Children|Romance
## 10 Adventure|Animation|Children|Drama|Musical
## 11                            Action|Comedy
```

We note 6 columns present. Timestamp in particular needs to be converted to an actual date. Genre seems like a list, but it may be too computationally expensive to split then fit. Will save this to last if the performance is not adequate. It looks prudent to split the title column as it contains both the title and the year of movie release. It's probably unique like the movie ID.

Next we examine some summary statistics:

```
summary(edx)
```

```
##      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
##
##
##
```

userId and movieId should really be factor columns as the numbers are discrete. In this case userId needs to be converted while we will use the movie title instead of the movieId. The average rating is important as we will be using it as a baseline to then estimate fixed effects on the data. We also obtain the maximum and minimum of the ratings allowing constraints to be set for the prediction later.

With the initial raw data examined we can start processing the raw data. First we load the required libraries:

```
#------------------------------------------------------------------------------
#loading essential modules

if(!require(ggplot2)) install.packages("ggplot2", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org")

library(ggplot2)
library(lubridate)
library(stringr)
```

With the libraries loaded we first want to add in some extra columns and change some stuff that may improve the later inference:

- the year of the movie
- the deviation of the ratings from the average rating (used for future fitting)
- a combination of the reviewer and genre as reviewers likely have favorite genres which they may bias themselves to
- convert the userId column to a factor column

```
# prep data make a column that has the rating deviation from mean
edx = edx %>%
  mutate(rating_dev = rating-mean(rating),
         movie_year = str_extract(title, "(\\(\\d{4}\\))"),
         reviewer_genre = paste(genres, userId))
# chop off the brackets and make it into a numeric column
edx$movie_year = gsub("\\(", "", edx$movie_year)
edx$movie_year = gsub("\\)", "", edx$movie_year)
edx$movie_year = as.numeric(edx$movie_year)
edx$userId <- as.factor(edx$userId)
```

next we convert timestamp to dates:

```
# convert to datetime then to date
edx <- mutate(edx, review_date = as.Date(as_datetime(timestamp)))
edx <- mutate(edx, review_year = year(review_date))
```

this completes the initial data preparation, we can now split the data into testing and training which will be used in future optimisation tasks.

```
# First split data into test and validation sets:
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set = edx[-test_index,]
val_set = edx[test_index,]
```

Modelling temporal effects

To start with we will set a baseline performance test by comparing the simple mean to the overall dataset and obtaining the RMSE of that

```
# base case, what is the mean?
mu = mean(edx$rating, na.rm = TRUE)
rmse_base = RMSE(edx$rating, mu)
print(rmse_base)
```

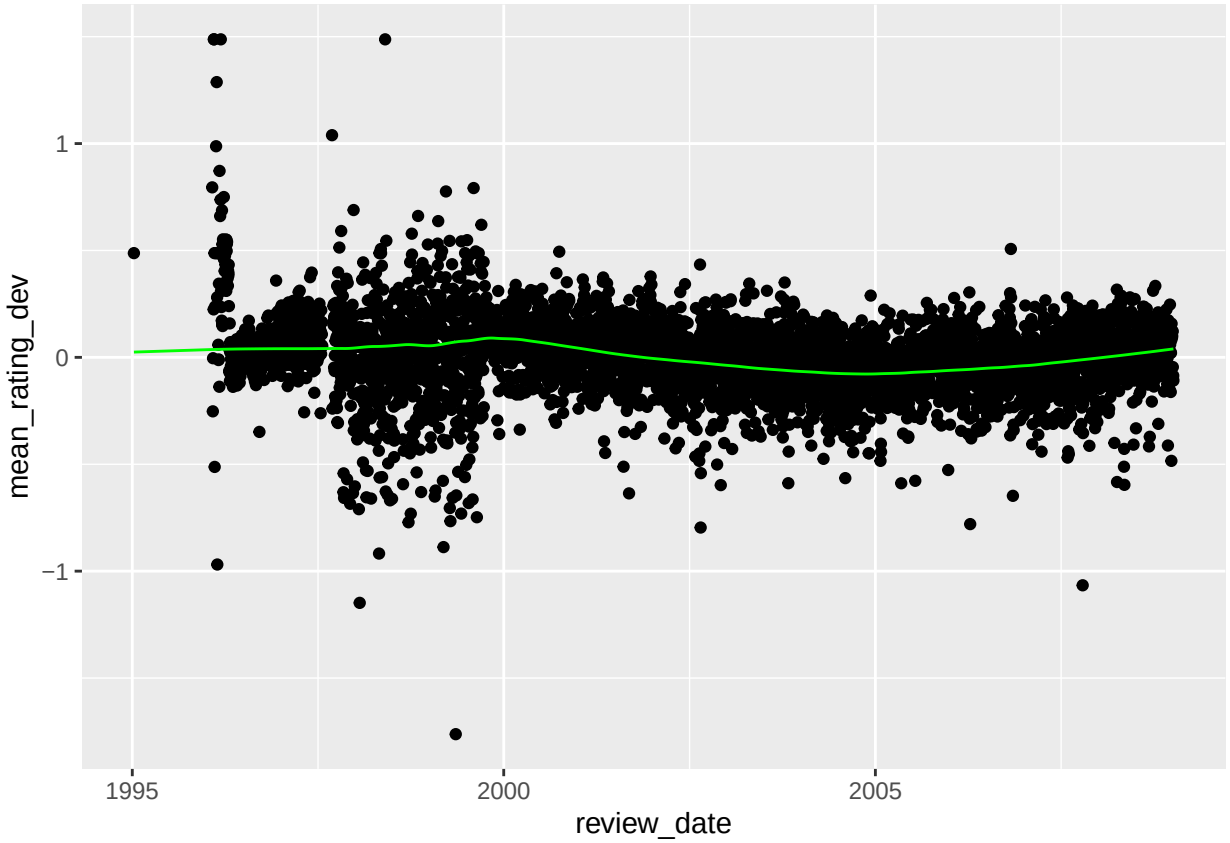
```
## [1] 1.060331
```

In theory any predictive method should show an appreciative gain over this value for it to be a factor affecting the overall results.

To start with, we examine the effect of time on the overall ratings. This can come from two sources, the time in which the review is posted and the time in which the movie comes out. Starting with the time of review, we use LOESS to fit the temporal variance. Because the dataset is too large to reasonably process the data is first compressed through averaging across the different review dates. To compensate for different review dates having differing amount of reviews, the data contribution to the final LOESS calculation is weighted according to the number of reviews.

To first average then view the data with fit:

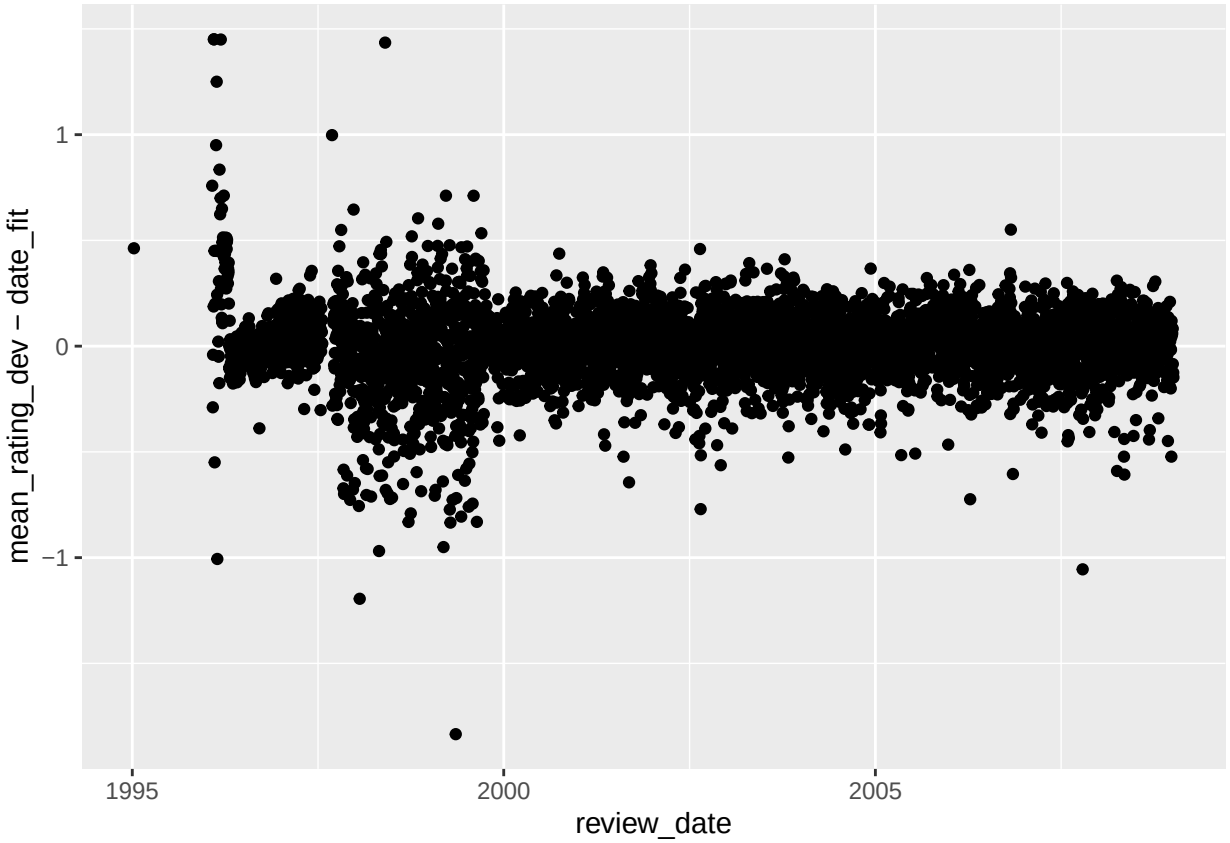
```
# calculate time dependent variation in scores. Will use loess
date_mean = train_set %>% group_by(review_date) %>%
  summarise(mean_rating = mean(rating),
            mean_rating_dev = mean(rating_dev),
            weight = n())

time_deviation = loess(mean_rating_dev ~ as.numeric(review_date),
                       data = date_mean,
                       span=0.3,
                       degree = 1,
                       weights = weight)

time_effect = date_mean %>%
  mutate(date_fit = predict(time_deviation, data = date_mean))

ggplot(data = time_effect,
       aes(x = review_date, y=mean_rating_dev)) +
  geom_point() +
  geom_line(aes(x = review_date, y=date_fit), color='green')
```



We note that the plot shows considerable variation across the dates, but the green loess fit seems to capture the general trend of the data. This fit in this case is based off the difference of actual scores to the overall mean. This will make it easier later when the final adjustments are applied.

to check if we corrected the temporal variance we can re-plot the scatter plot to see how the data distributes around 0

```
# plot correction
ggplot(data = time_effect,
       aes(x = review_date,
           y=mean_rating_dev-date_fit)
       ) + geom_point()
```



there appears to be some error in roughly 1997, but overall the distribution sits well along 0 suggesting the majority of the review date effect is corrected.

To check the effect on RMSE (for the whole edx set):

```
edx$time_bias = predict(
  time_deviation,
  data.frame(review_date = as.numeric(edx$review_date)))

print(RMSE(edx$rating, mu))
```

```
## [1] 1.060331
```

```
print(RMSE(edx$rating, edx$time_bias+mu))
```

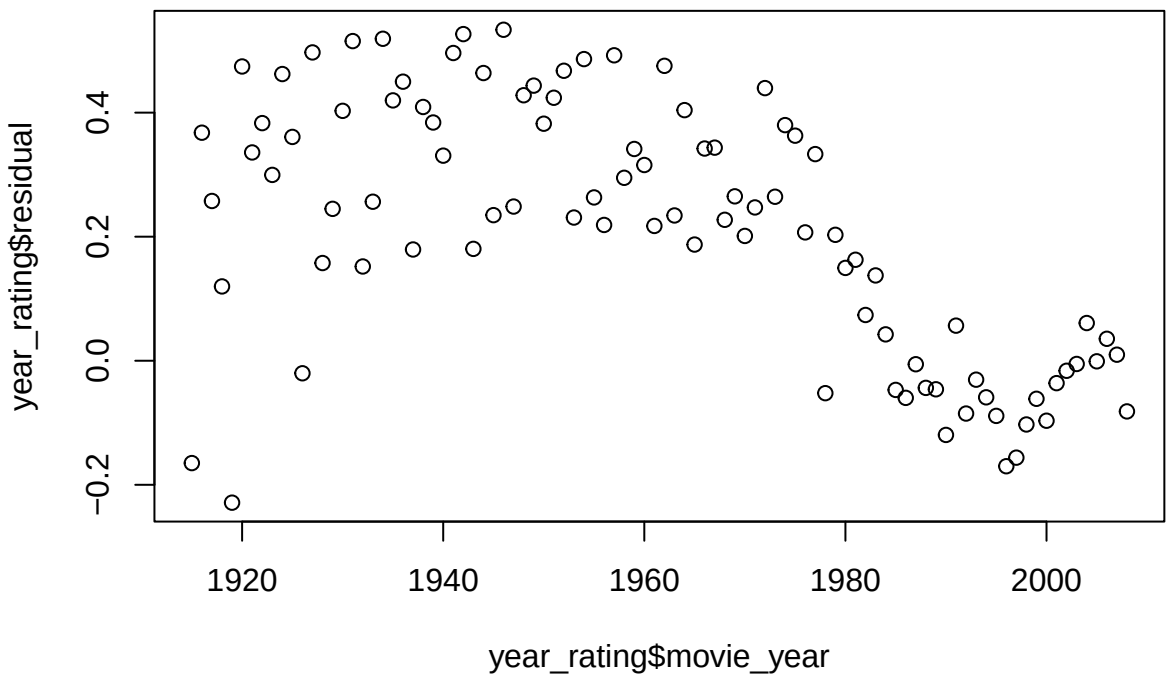
```
## [1] 1.058651
```

It's an improvement, albeit very mild. Next the effect of movie release year on the overall distribution is added. To do so, first, the residual variance after fitting the review date is extracted.

```
# residual variance after review time fitted
edx$residual =  edx$rating-mu-edx$time_bias
```

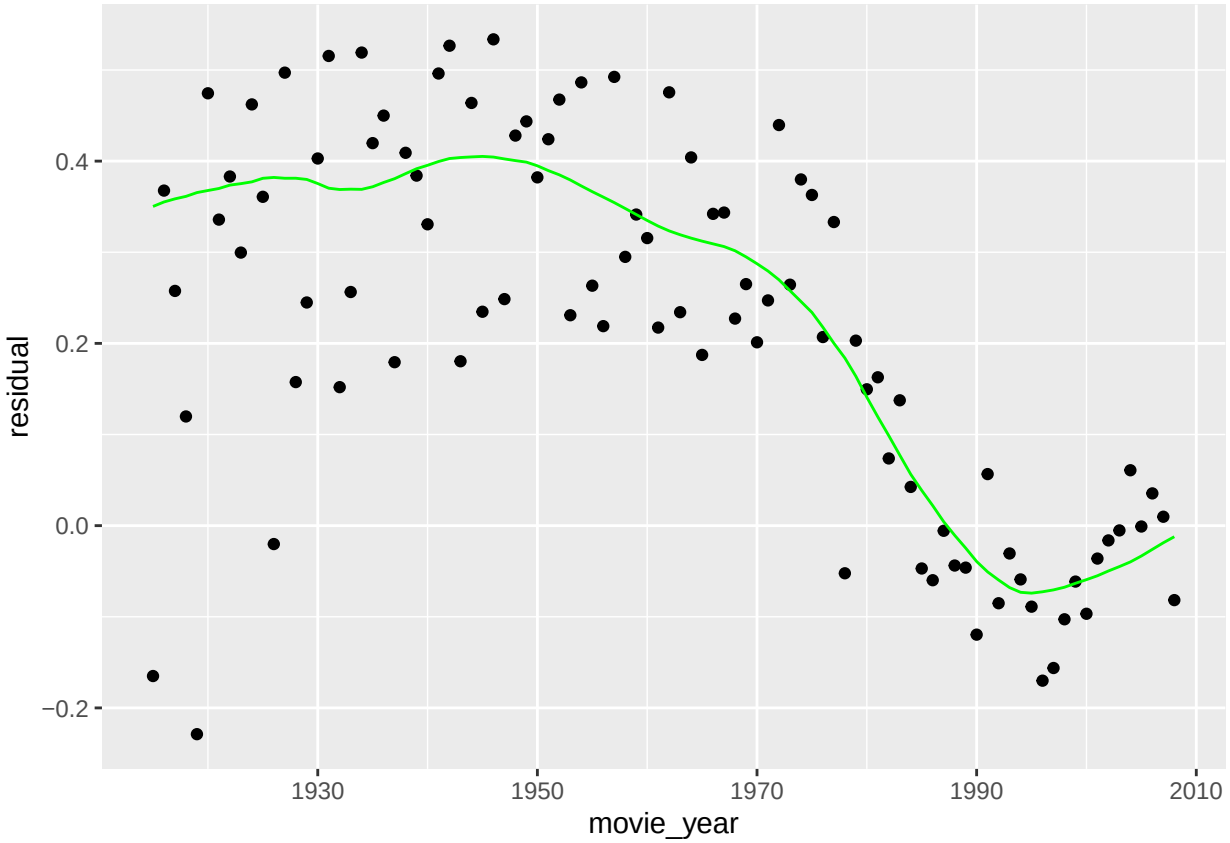
Next we plot of the variance for the review year:

```
year_rating = edx %>%
  group_by(movie_year) %>%
  summarise(residual = mean(residual), weight = n())
plot(year_rating$movie_year, year_rating$residual)
```



Definitely a substantial variance, it looks to be between 0.2 less than average and 0.4 above average. To fit the curve via LOESS

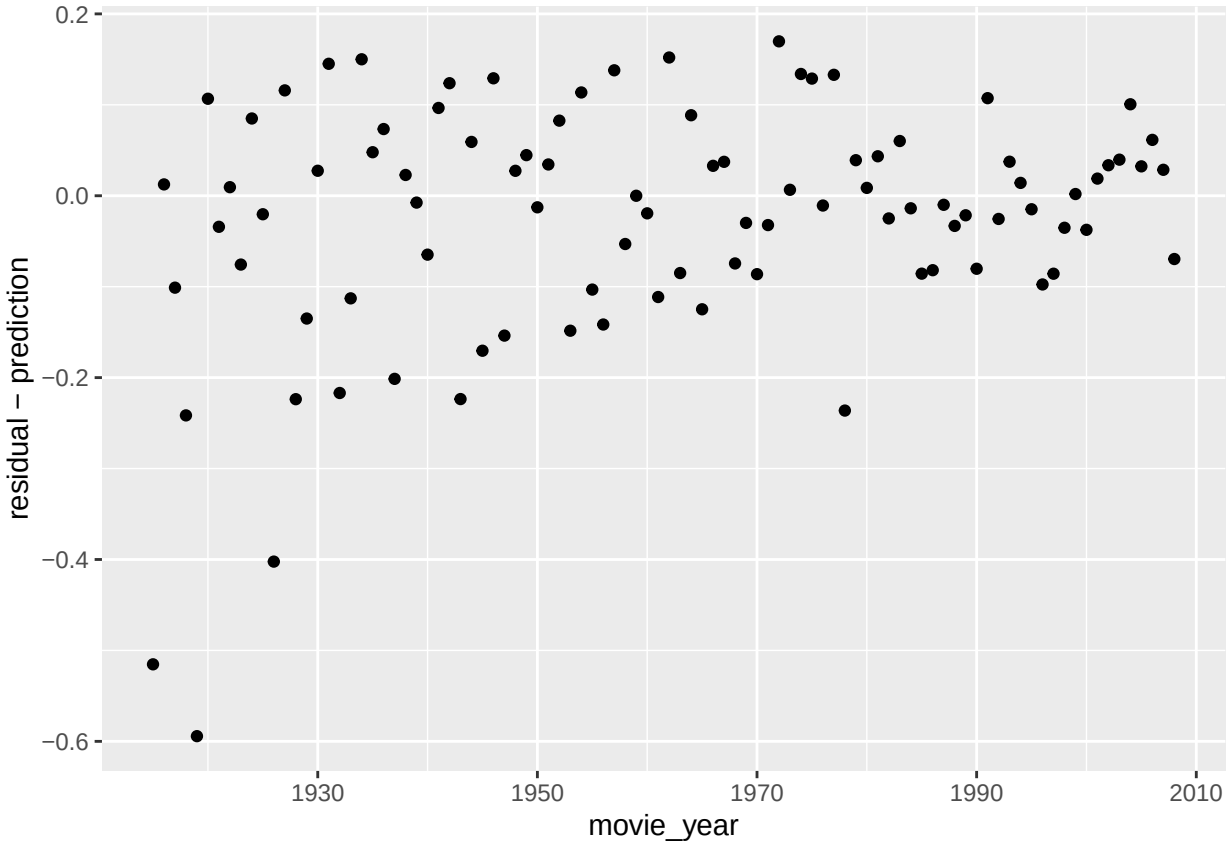
```
release_year_deviation = loess(
  residual ~ movie_year,
  data = year_rating,
  span=0.3,
  degree = 1,
  weights = weight)

year_rating$prediction = predict(release_year_deviation, year_rating)

ggplot(
  data = year_rating,
  aes(x = movie_year, y=residual)
  ) +
  geom_point() +
  geom_line(aes(x = movie_year,
                y=prediction),
            color='green')
```




```
edx$release_bias = predict(release_year_deviation, edx)
```

The fit looks pretty good, some outliers at in the early 1900s but otherwise it conforms well. Checking on the corrected variance:

```
# plot the corrected values
ggplot(data = year_rating, aes(x = movie_year, y=residual-prediction)) + geom_point()
```



and RMSE:

```
# error RMSE:
print(RMSE(edx$rating, edx$release_bias + mu))
```

```
## [1] 1.051103
```

More of an improvement, still pretty small.

For the final time adjusted ratings, final RMSE in comparison with just the mean:

```
edx$mu_t = edx$time_bias + mu + edx$release_bias
print(RMSE(edx$rating, mu))
```

```
## [1] 1.060331
```

```
print(RMSE(edx$rating, edx$mu_t)) #a tiny improvement
```

```
## [1] 1.04949
```

Definitely something. Though just small adjustments to the RMSE. Nevertheless, it's not expected that time is the key determinant for ratings, for that what the movie is and who reviews it will likely have the greater impact.

Modelling discrete effects

For this section we look at the effects of the user, the movie itself, the genre and the possible interaction effects of user favoritism for certain genres. These effects are best modeled for these average data sets with mean effect adjustments with regularization.

In this section we examine 4 different effects. To see the individual contributions for each, similar steps as above can be tested. First to add the time functions to the original data split:

```
# recreate the sets now with the time adjustment in place
train_set = edx[-test_index,]
val_set = edx[test_index,]
val_set_testing = edx[test_index,]
```

the individual effects are then:

```
#-------------------------------------------------------------------------------
# calculate genre dependent variation
genre_bias = train_set %>% group_by(genres) %>% summarise(mean_rating_genre = mean(rating), mean_dev_ge
error = left_join(edx, genre_bias, by='genres')

print('base rmse:' )
```

```
## [1] "base rmse:"
```

```
print(RMSE(error$rating, error$mu_t))
```

```
## [1] 1.04949
```

```
print('genre corrected rmse:' )
```

```
## [1] "genre corrected rmse:"
```

```
print(RMSE(error$rating, replace_na(error$mean_dev_genre, 0)+error$mu_t))
```

```
## [1] 1.008387
```

```
#-------------------------------------------------------------------------------
# calculate user dependent variation
average_user_ratings = train_set %>%
  group_by(userId) %>%
  summarise(mean_rating_user = mean(rating),
            mean_dev_user = mean(rating-mu_t),
            sum_dev_genre = sum(rating-mu_t),
            entries = n())

error = left_join(edx, average_user_ratings, by='userId')
print('user corrected rmse:' )
```

```
## [1] "user corrected rmse:"
```

```
print(RMSE(error$rating, replace_na(error$mean_dev_user, 0)+error$mu_t))
```

```
## [1] 0.9616939
```

```
#-------------------------------------------------------------------------------
# calculate movie dependent variation
average_movie_rating = train_set %>%
  group_by(title) %>%
  summarise(mean_rating_movie = mean(rating),
            mean_dev_movie = mean(rating-mu_t),
            sum_dev_genre = sum(rating-mu_t),
            entries = n())

error = left_join(edx, average_movie_rating, by='title')
print('title corrected rmse:' )
```

```
## [1] "title corrected rmse:"
```

```
print(RMSE(error$rating, replace_na(error$mean_dev_movie, 0)+error$mu_t))
```

```
## [1] 0.9407898
```

```
#-------------------------------------------------------------------------------
# calculate reviewer, genre co-dependent variation
average_rg_rating = train_set %>%
  group_by(reviewer_genre) %>%
  summarise(mean_rating_movie = mean(rating),
            mean_dev_rg = mean(rating-mu_t),
            sum_dev_genre = sum(rating-mu_t),
            entries = n())
error = left_join(edx, average_rg_rating, by='reviewer_genre')
print('reviewer, genre co-dependent corrected rmse:' )
```

```
## [1] "reviewer, genre co-dependent corrected rmse:"
```

```
print(RMSE(error$rating, replace_na(error$mean_dev_rg, 0)+error$mu_t))
```

```
## [1] 0.6908205
```

Each of these effects can lead to an improvement in the estimates as indicated by their lower RMSE. Therefore, correction of each of these factors is carried out in the final calculations. We find that from the scores, the user and title both ahve a significant imapct on the overall scoring, additionally, it's likely that the reviewer genre correction is severely over fitting data considering the very small RMSE for that particular predictor.

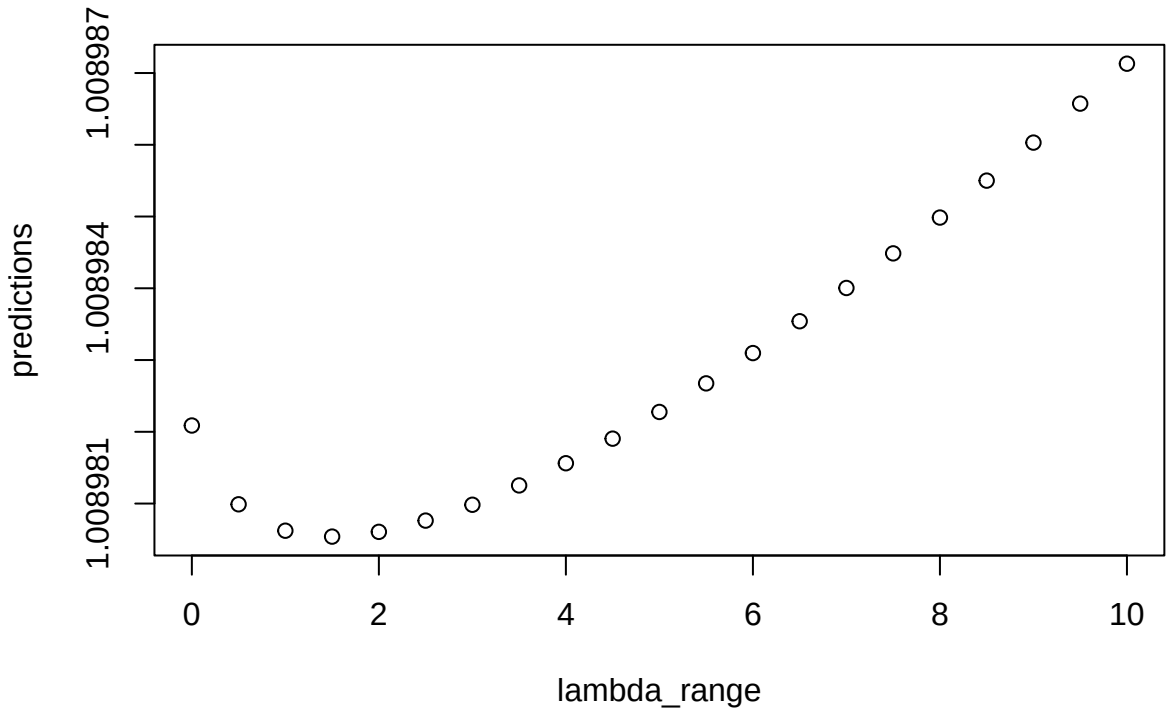
To prevent over fitting of small groups especially in the combined user/genre grouping, regularization can be sued to reduce the amount of dependence on groups with low number of entries. Regularization is carried out through optimization of the regularization factor lambda.

In addition to regularization, the factors are also combined together where each factor is to only fit on the residual variance of the previous factor. With these combined the final model can be created.

Results

First the genre component is added

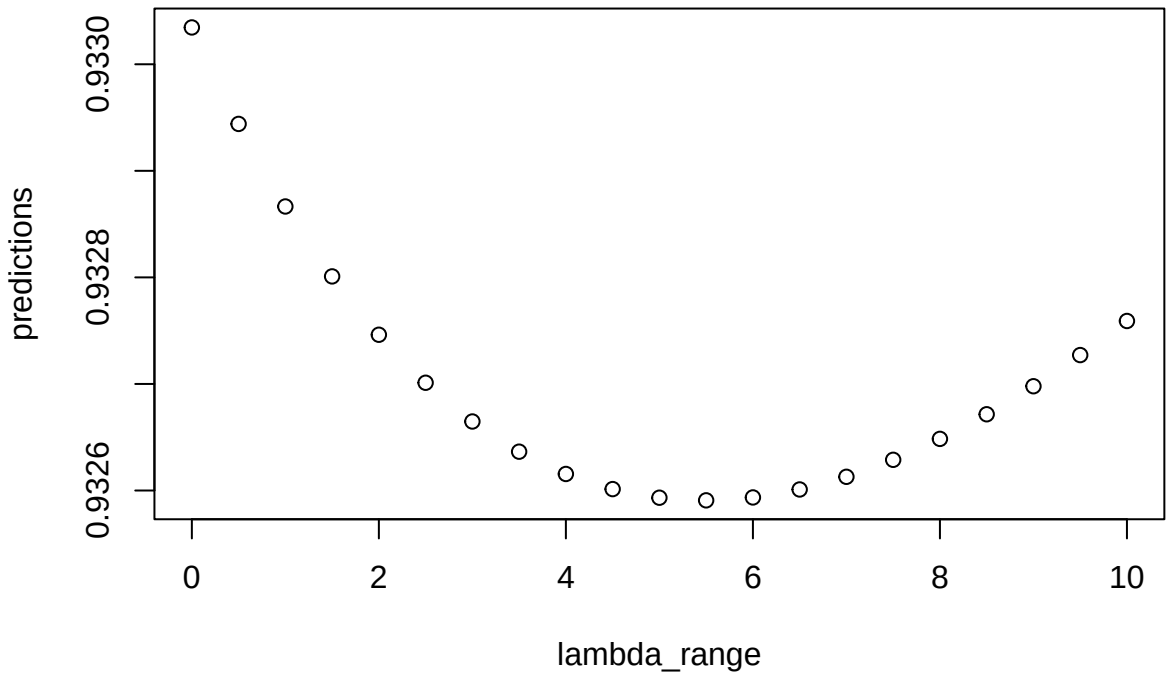
```
# regularised model

# isolated genre effect
lambda_range = seq(0, 10, 0.5)

lambda_search_genre = function(lambda){
  genre_alone_reg = train_set %>%
    group_by(genres) %>%
    summarize(genre_effect_reg = sum(rating-mu_t)/(n()+lambda))

  prediction = val_set %>%
    left_join(genre_alone_reg, by = 'genres')
  rmse = RMSE(prediction$rating, replace_na(prediction$genre_effect_reg, 0) + prediction$mu_t)
  return(rmse)
}

# predict and plot the lambda scores
predictions = sapply(lambda_range, lambda_search_genre)
plot(lambda_range, predictions)
```



```
lambda_t_g = lambda_range[which.min(predictions)] # returns 0, regularization does not appear to help i

# predictor table for genre effects
genre_alone_reg = edx %>%
  group_by(genres) %>%
  summarize(genre_effect_reg = sum(rating-mu_t)/(n()+lambda_t_g))

train_set = train_set %>%
  left_join(genre_alone_reg, by='genres')
val_set = val_set %>%
  left_join(genre_alone_reg, by='genres')
```

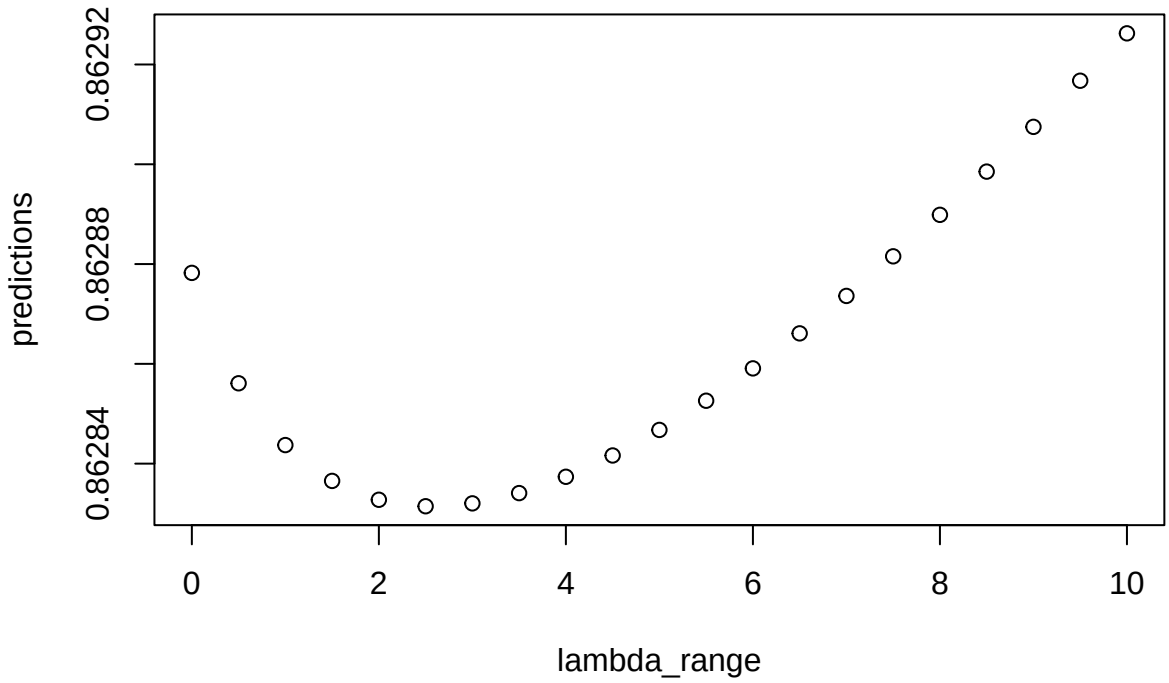
This plot shows that the ideal lambda setting for regularization for this is 1.5. This then continues for the remaining factors to consider.

For user effects



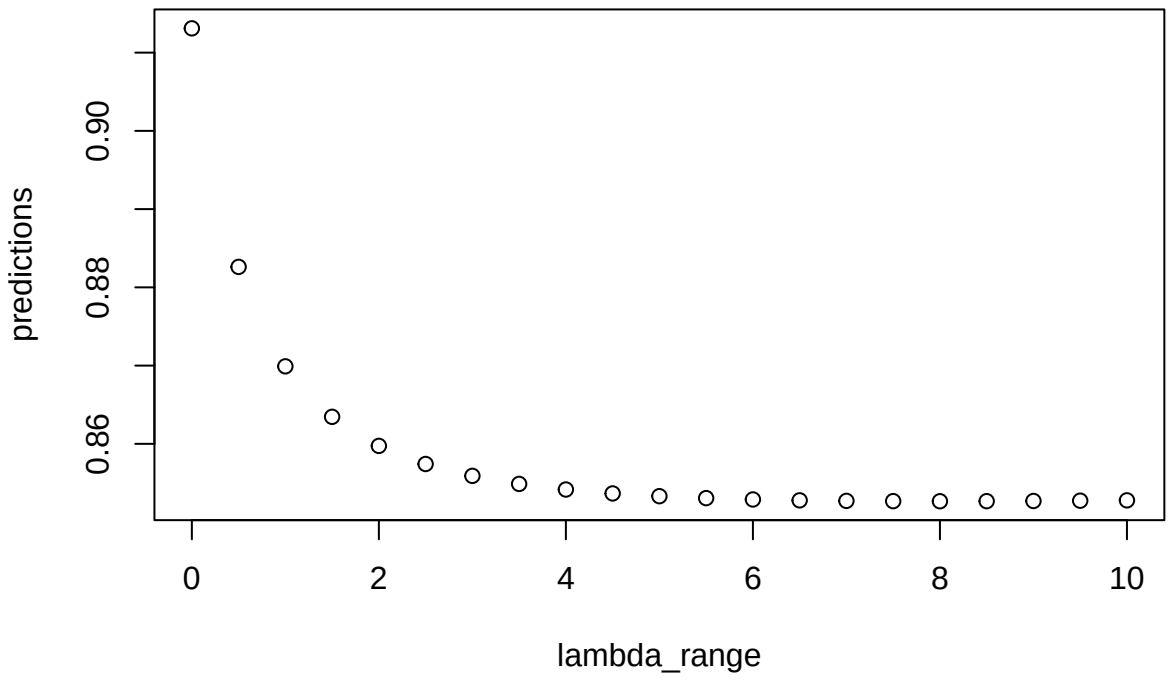

```
## [1] "lambda for :5.5"
```

For title effects




```
## [1] "lambda for title:2.5"
```

For genre/user co-effects




```
##  [1] "lambda for genre/user:0"   "lambda for genre/user:0.5"
##  [3] "lambda for genre/user:1"   "lambda for genre/user:1.5"
##  [5] "lambda for genre/user:2"   "lambda for genre/user:2.5"
##  [7] "lambda for genre/user:3"   "lambda for genre/user:3.5"
##  [9] "lambda for genre/user:4"   "lambda for genre/user:4.5"
## [11] "lambda for genre/user:5"   "lambda for genre/user:5.5"
## [13] "lambda for genre/user:6"   "lambda for genre/user:6.5"
## [15] "lambda for genre/user:7"   "lambda for genre/user:7.5"
## [17] "lambda for genre/user:8"   "lambda for genre/user:8.5"
## [19] "lambda for genre/user:9"   "lambda for genre/user:9.5"
## [21] "lambda for genre/user:10"
```

```
## `summarise()` has grouped output by 'reviewer_genre'. You can override using
## the `.groups` argument.
```

This allows for creation of the final predictor function by combining all of the above models. We note that progressive corrections on prediction variance leads to much better RMSE on the validation data.

For the final prediciton:

```
final_predictor = function(dataset){

  # to make code easier for passing in the final holdout there will be some repeat stuff

  dataset = dataset %>%
```

```
  mutate(rating_dev = rating-mean(rating),
         movie_year = str_extract(title, "(\\(\\d{4}\\))"),
         reviewer_genre = paste(genres, userId))

dataset$movie_year = gsub("\\(", "", dataset$movie_year)
dataset$movie_year = gsub("\\)", "", dataset$movie_year)
dataset$movie_year = as.numeric(dataset$movie_year)

dataset$userId = as.factor(dataset$userId)
dataset = mutate(dataset, review_date = as.Date(as_datetime(timestamp)))
dataset = mutate(dataset, review_year = year(review_date))


# setting the base
dataset$base_prediction = mu


# setting the time bias with base
dataset$time_bias = predict(time_deviation,
                            data.frame(review_date = as.numeric(dataset$review_date)))
dataset$release_bias = predict(release_year_deviation, dataset)
dataset$mu_t = dataset$time_bias + mu + dataset$release_bias

# user adjustment
dataset = left_join(dataset, genre_alone_reg, by='genres')

# genre
dataset = left_join(dataset, user_effect_reg, by='userId')

# title adjustment
dataset = left_join(dataset, title_effect_reg, by='title')

# reviewer genre effects
dataset = left_join(dataset, reviewer_genre_effect_reg, by='reviewer_genre')
View(dataset)

# logic side, favor title means with user adjustment
# if title missing from training, guess using genre prediction
# if user missing, assume the mean title (i.e. 0 adjustment)

dataset$final_prediction = replace_na(dataset$genre_effect_reg, 0) +
  replace_na(dataset$user_effect_reg, 0) +
  replace_na(dataset$title_effect_reg, 0) +
  replace_na(dataset$reviewer_genre_effect_reg, 0) +
  dataset$mu_t

# set constraints

dataset$final_prediction[dataset$final_prediction > 5] = 5
dataset$final_prediction[dataset$final_prediction < 0.5] = 0.5
dataset$prediction_error = (dataset$rating - dataset$final_prediction)^2

#calculate rmse
```

```
  rmse = RMSE(dataset$rating, dataset$final_prediction)
  print(paste0('Final prediction RMSE:',rmse))
  return(dataset)
}
```

using this function, the final predicted output is:

```
output = final_predictor(final_holdout_test)
```

```
## [1] "Final prediction RMSE:0.851159058305698"
```

This predicted values is considerably better than the target RMSE of 0.8649. However, it should be noted, that these predictions are still more than half score off the actual score for users in reality. It's likely that due to computational limitations, more complex models such as KNN, RF and Bayesian predictors could not be used.

Conclusion

In this project a simple score prediction algorithm was created to predict movie scores when given a range of factors. We note that a simple multivariate mean fitting algorithm was sufficient when coupled with time based fitting through LOESS to achieve the desired RMSE. It's notable however, that while this algorithm is shown to be effective up to this level, it can still be further improved likely through more sophisticated modelling that takes into account more potential points of interaction. Especially of interest is the effect of various genres on the overall predictive power.

A major limitation in this project is often the slow nature of the language. While further code optimization is likely possible, it's probably a much better idea to use more efficient languages to carry out the model construction in future development.