# R Capstone: MovieLens

Michael Beller

12/02/2020

## 1. Executive Summary

This project involved analysis of a large (~10 million row) data set from the MovieLens database. Each record in this data set contains a rating of a movie given by a particular reviewer (user). The project was to develop an algorithm to predict a user's rating for a movie they had not previously reviewed.

The MovieLens data set included unique identifiers for the movie and the user, the rating given by the user for that movie, the movie title, a "genres" field listing all applicable genres to which the movie falls, and a time stamp for the review.

Before analysis began, a **validation** set of ~1 million records was split off to be used for the final model accuracy calculation, while the remaining records were split into training (**edx_train**) and test (**edx_test**) sets.

After exploring the training data set and experimenting with different modeling approaches, the model was built on the following key features: 1. The overall mean rating of movie reviews

2. A "movie effect" based on a movie's mean rating
3. A "user/genre effect" based on a user's mean rating for different genres
4. A "user pickiness effect" based on other differences between the user's ratings and the overall mean, accounting for movie effects and user/genre effects.

This model was predicated on breaking down a movie's "genres" field into individual genres, which were used as the basis for the user/genre effect.

The model's effectiveness was judged by its ability to minimize root mean square error (RMSE). A specific target of RMSE $< 0.8649$ was specified. As discussed in ***Section 3*** (Results), RMSE was computed on the **edx_test** set for several iterative versions of the model. The final version was then run against the **validation** set, and the target was achieved. The final result from the validation run can be found in ***Section 3.5***.

## 2. Methods/Analysis

One brief technical note: given the size of the MovieLens data set, and the types of analysis performed, the base memory limit was increased to 10,000.

## 2.1. Initial Setup

The first task was splitting off around ten percent of the data into a validation set, using the provided code:

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

library(tidyverse)
library(caret)
library(data.table)

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

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")

# if using R 4.0 or later:
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, sample.kind="Rounding")
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)
```

Given the large size of the data set, it was necessary to increase the default memory limit.

```
memory.limit(size=10000)
```

```
## [1] 10000
```

Following the split of the validation data, the remaining 90% of the original data set had been stored in **edx**. The **edx** set was then split into training (**edx_train**) and test (**edx_test**) data sets for use in model development.

```
set.seed(99, sample.kind="Rounding")
edx_test_index <- createDataPartition(
  y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train <- edx[-edx_test_index,]
edx_test <- edx[edx_test_index,]
```

The following function was defined to perform the many RMSE calculations that would be required throughout this analysis.

```
RMSE_f <- function(predicted,actual){
    RMSE <- sqrt(mean((actual - predicted)^2))
}
```

## 2.2. Approach/Strategy

The guiding strategy for this analysis was based on the the following two concepts: 1. The size of the **edx_train** data set (7,200,043 rows) meant that simply running a variety of different modeling techniques (logistics regression, LDA/QDA, knn, random forest, etc.) and picking the best performer was not practical. Any such modeling would have to be done on a small subset of the training data.

2. The "basic model" presented in the machine learning class (Predicted rating = Overall Mean Rating + Movie Effect + User-Specific Effect), seemed like a logical way to address modeling such a large data set, but did not perform well enough and appeared incomplete, especially in its handling of user-specific effects.

## 2.3. Exploration - prediction by overall mean

This is the most basic of predictions, using the overall mean rating across all movies and users in the training set to predict ratings in the test set. This is useful only as a starting point.

```
#Calculate overall mean from training set
overallmean <- mean(edx_train$rating)
print(overallmean)
```

```
## [1] 3.512519
```

```
#Predict test set using overall mean
predA <- overallmean
predA_RMSE <- RMSE_f(predA,edx_test$rating)
pred_results <- data.frame(prednum = "A", predtype = "Overall Mean", RMSE = predA_RMSE)
pred_results[1,]
```

```
##   prednum     predtype     RMSE
## 1       A Overall Mean 1.060269
```

The RMSE value from overall mean prediction is 1.0603, leaving plenty of room for improvement. The overall mean was the first component of the eventual model, referenced in the model equation as **OMR**.

Current model status: **Predicted Rating = OMR**

## 2.4 Exploration - prediction by mean movie rating

Still following the machine learning class example at this point, the next step was to calculate mean ratings for each movie across all of its reviews in the training set. These mean ratings were then expressed as differences from the overall mean – for example, if a movie has a mean rating of 3.9 and the overall mean rating is 3.5, the movie would have an effect of 0.4. The resulting movie effects were then used in combination with the overall mean to predict ratings in the test set.

Note that there were some movies in the test set that did not exist in the training set. For any such movies, the movie effect was set to zero, so they were predicted using the overall mean.

```r
#Calculate movie effects as the difference between mean movie rating and overall mean rating
#in the training set.
movie_effects <- edx_train %>%
  group_by(movieId) %>%
  summarize(mcount = n(), meanrating = mean(rating)) %>%
  ungroup() %>%
  mutate(moviediff = meanrating - overallmean)

#Predict test set using movie effects
predB <- left_join(edx_test, movie_effects, by = "movieId")
#If a movie in the test set does not exist in the training set, then set its movie effects
#difference to zero (i.e. use the overall mean)
predB$moviediff[which(is.na(predB$moviediff))] <- 0
#Compute predicted rating
predB <- predB %>% mutate(predrating = overallmean + moviediff)

#Calculate RMSE
predB_RMSE <- RMSE_f(predB$predrating,predB$rating)
pred_results <- bind_rows(pred_results,
                    data_frame(prednum = "B", predtype = "Movie Mean", RMSE = predB_RMSE))
pred_results[2,]
```

```
##   prednum   predtype      RMSE
## 2       B Movie Mean 0.9432967
```

```r
rm(predB)    #cleanup
```

As expected, predicting by mean movie rating led to a significant improvement versus overall mean, illustrating the importance of individual movie quality. However, the resulting improvement still fell far short of the target RMSE. Nevertheless, the magnitude of the movie effect was such that it would be included in the final model, and referenced in the model equation as **ME**.

Current Model: **Predicted Rating = OMR + ME**

At this point, some consideration was given to breaking the overall movie effect into subcomponents – for example, a movie quality effect, a movie genre effect, and a year of release effect. However, the decision was made to focus on user effects first.

## 2.5 Exploration - prediction by mean user rating

To understand if there was potential in user effects, the next step was to make predictions using mean user ratings instead of mean movie ratings. For this step, movie effects were ignored entirely. Mean ratings were calculated by user, then expressed as differences from the overall mean. The resulting user effects were then used in combination with the overall mean to predict ratings in the test set.

Note that there were some users in the test set that did not exist in the training set. For any such users, the user effect was set to zero, so they were predicted using the overall mean.

```r
#Calculate movie effects as the difference between mean movie rating and overall mean rating
#in the training set.
user_effects <- edx_train %>%
                  group_by(userId) %>%
                  summarize(ucount = n(),meanrating = mean(rating)) %>%
                  ungroup() %>%
                  mutate(userdiff = meanrating - overallmean)

#Predict test set using user effects
predC <- left_join(edx_test, user_effects, by = "userId")
#If a user in the test set does not exist in the training set, then set their user effects
#difference to zero (i.e. use the overall mean)
predC$userdiff[which(is.na(predC$userdiff))] <- 0
#Compute predicted rating
predC <- predC %>% mutate(predrating = overallmean + userdiff)
#Calculate and report RMSE
predC_RMSE <- RMSE_f(predC$predrating,predC$rating)
pred_results <- bind_rows(pred_results,
                  data_frame(prednum = "C", predtype = "User Mean", RMSE = predC_RMSE))
pred_results[3,]
```

```
##   prednum  predtype      RMSE
## 3       C User Mean 0.9783654
```

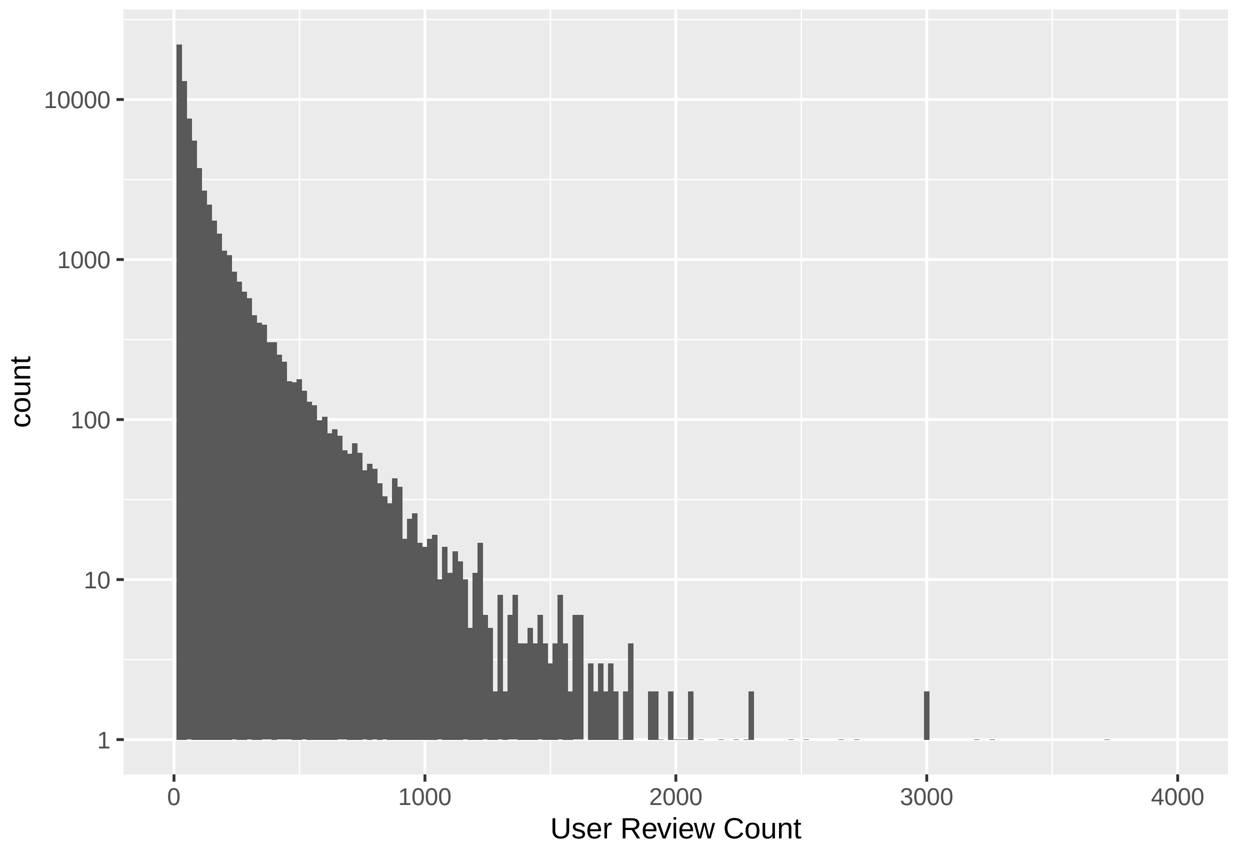
```r
rm(predC)    #cleanup
```

The resulting RMSE of 0.97837 showed that, while not as powerful as movie effects, user effects alone were still a significant improvement over overall mean prediction. The effects were almost as powerful as movie effects alone. This further reinforced the potential predictive improvement that more analysis of user effects could provide.

## 2.6 Analysis of user effects

The machine learning class had indirectly touched on the importance of user effects, treating them as the residual difference between a mean movie rating prediction and the actual rating. However, that had been far from sufficient to meet the target RMSE. To go further, it was necessary to understand some of the characteristics of user data in the training set. One of the more obvious characteristics is the wide range of user activity.

```r
user_effects %>% ggplot(aes(ucount)) + geom_histogram(binwidth = 20) +
  scale_y_log10() + xlim(0,4000) + xlab("User Review Count")
```



```r
user_effectsq <- quantile(user_effects$ucount)
```

As these figures show, the vast majority of users submit less than fifty reviews, while a few very prolific ones submit thousands.

One possibility might be variances in the mean ratings of users based on the number of reviews they have given. This is a look at mean rating by quantile:

```r
lowuser <- user_effects %>% filter(ucount <= user_effectsq[2])
meduser <- user_effects %>% filter(ucount > user_effectsq[2] & ucount <= user_effectsq[3])
highuser <- user_effects %>% filter(ucount > user_effectsq[3] & ucount <= user_effectsq[4])
vhighuser <- user_effects %>% filter(ucount > user_effectsq[4])
user_ratingsq <- data.frame(quantile = c("Low","Medium","High","Very High"),
                           meanrating = c(mean(lowuser$meanrating),
                                          mean(meduser$meanrating),
                                          mean(highuser$meanrating),
                                          mean(vhighuser$meanrating)))
user_ratingsq
```

```
##    quantile meanrating
## 1       Low   3.629459
## 2    Medium   3.661561
## 3      High   3.641356
## 4 Very High   3.523402
```

There was no obvious pattern, although there did seem to be a decrease in mean rating among the most prolific reviewers.

Time-based factors, like movie release year and movie review data, were examined but similarly did not show a clear pattern.

This left genre as a likely candidate for analysis. Many people tend to have certain genres they like or dislike, and it seems likely that these preferences would influence the ratings they would give to particular movies in those genres. Ideally, a "user/genre" effect could be quantified, showing the mean effect that a user's preferences in each genre have on their ratings.

## 2.7 Data cleaning - Genre decomposition

Genre analysis was challenging because of how movie genre data is coded in the MovieLens data set. Each review record in MovieLens has a "genres" field, which is a string containing all of the genres to which a movie belongs. For example, the movie "Toy Story" (movie ID==1) has a genres field equal to "Adventure|Animation|Children|Comedy|Fantasy".

If user/genre effects were coded based on this field as is, we would get mean user/genre ratings for very specific genre combinations. In the Toy Story example, a user's rating for it would be averaged only with other movies that have the "Adventure|Animation|Children| Comedy|Fantasy" genre combination. In this case, there are only two other movies with that combination.

Intuitively, this did not seem like the best approach. In the author's experience, most people would consider their genre preferences in terms of individual genres – for example, "I like action movies", or "I hate romantic comedies." It is much less likely for someone to express a specific preference for "Adventure/Animation/Children/Comedy/Fantasy hybrids". Even if they did, the lesser volume of movies possessing a given genre combination might lead to more variance in ratings.

A more intuitively useful analysis would be averaging ratings of all movies that are included within an individual genre. In the "Toy Story" example, its ratings would be included in individual analyses of the Adventure genre, the Animation genre, the Children genre, the Comedy genre, and the Fantasy genre. This would better reflect the way many people express genre preferences. It also ensures a relatively deep pool for each individual genre (of which there are only 20).

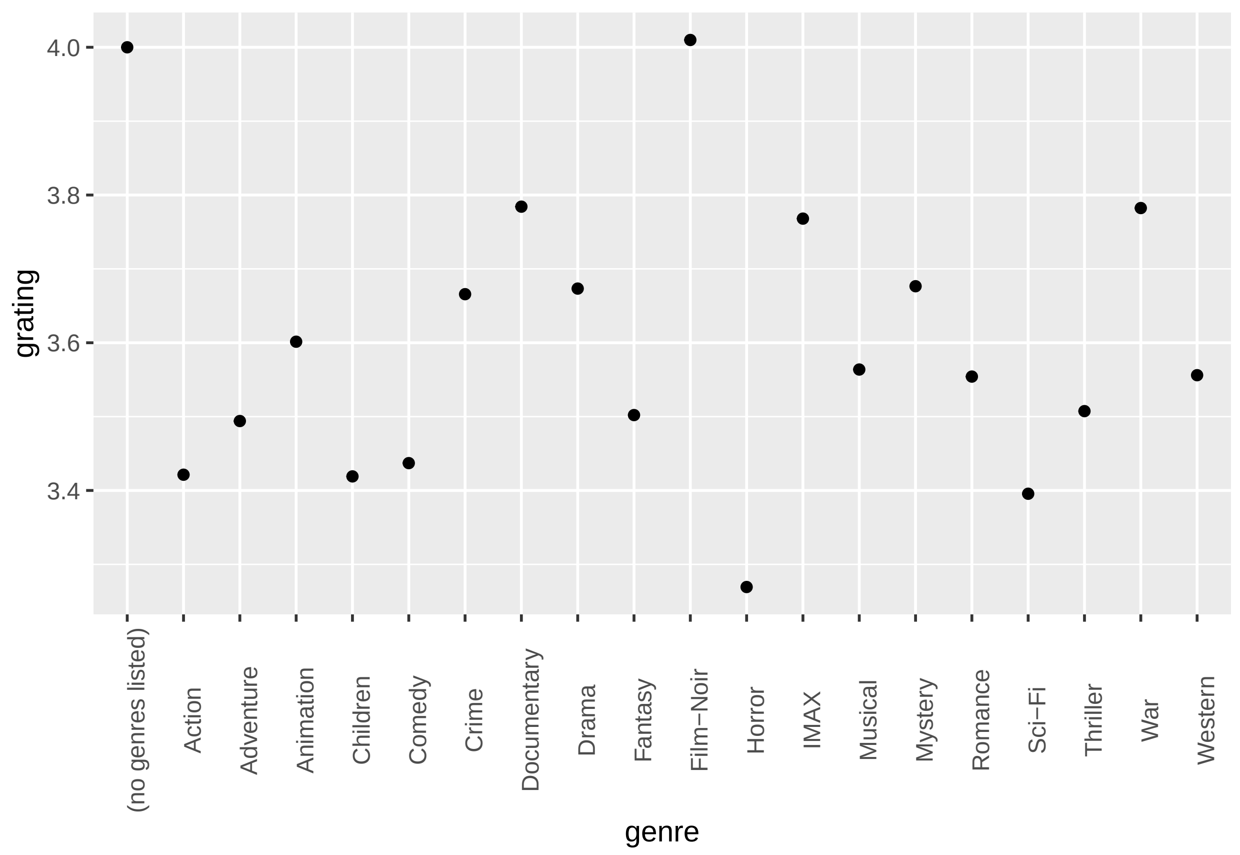
However, this analysis would require some pre-work to decompose the genres field into its individual genre components. Since each individual genre in the genres field was separated by a "|", it was straightforward to split them using functions from the stringr package. A bigger question was how to store this data once split. This was accomplished by creating a **movie_genres** data frame, populated with one record for each genre that a movie belonged in. The genre decomposition was implemented with the following code:

```r
# Create a list of all single genres (from movies that have only one genre)
genre_tab <- edx_train %>% filter(str_detect(genres,"\\|") == FALSE) %>%
                           group_by(genres) %>%
                           summarize(count = n())
genre_list <- genre_tab$genres

# Define function to build movie genres
build_movie_genres <- function (movietab) {
  movgen <- data.frame(movieId = 0, genre = "Dummy")
  #Loop through list of movies
  for (x in 1:nrow(movietab)) {
    mg <- str_split(movietab$genres[x],"\\|",simplify = TRUE)  #| denotes new genre
    mrow <- ""
  #Loop through genres and create rows
    for (y in 1:dim(mg)[2]) {
      mrow <- c(mrow,mg[y])
    }
    mtab <- data.frame(movieId = movietab$movieId[x],genre = mrow)
    mtab <- mtab[-1,]
    movgen <- bind_rows(movgen,mtab)
  }
  movgen
}

#Get list of movies from edx_train, and build movie_genres table from them
moviesum <- edx_train %>%
              group_by(movieId, genres) %>%
              summarize()
movie_genres <- build_movie_genres(moviesum)
movie_genres <- movie_genres[-1,]
```

## 2.8 Movie genre analysis

With the genres field successfully decomposed, the genre analysis began by examining differences in the mean movie ratings for each individual genre.

```
#Get means by genre
genre_means <- inner_join(movie_genres, edx_train,by="movieId") %>%
                group_by(genre) %>%
                summarize(gcount = n(), grating = mean(rating))
genre_means %>% ggplot(aes(genre,grating)) + geom_point() +
                 theme(axis.text.x = element_text(angle=90))
```



As this graph shows, there are significant differences in mean ratings by genre. While we have not yet broken this down to a user level, it still supports the concept that user/genre preferences are a key component of accurate prediction.

### 2.9 User genre analysis

The next step was to look at mean genre ratings for each individual user. These were then expressed as a difference – not from the overall mean, but from the mean for each genre across all users. This seemed appropriate, since any genre effect for the movie itself is already captured in that movie's movie effects difference.

```
#-Build user/genre effects table
usergenre_effects <- inner_join(edx_train, movie_genres, by="movieId") %>%
                     group_by(userId,genre) %>%
                     summarize(ucount = n(), ugrating = mean(rating)) %>%
                     ungroup()
#Link to genre means to calculate difference by genre
usergenre_effects <- inner_join(usergenre_effects, genre_means, by="genre") %>%
                     mutate(ugdiff = ugrating - grating)
```

With these differences calculated, they would now be included in the model, referenced in the model equation as **UGE**.

Current Model: **Predicted Rating = OMR + ME + UGE**

## 2.10 User pickiness

With the user/genre effects calculated, there are still likely to be some differences between user ratings that have nothing to do with the movie itself, or with the genres it belongs to. For example, on a fixed rating scale (0.5 to 5 stars), there are some users who may use a harsher rating scale than others. These effects are collectively referred to here as "user pickiness", and are calculated as the mean residual remaining in the training set when the predicted rating is subtracted from the actual rating (overall mean + movie effects + user/genre effects).

This residual calculation had to address the complication of movies that are assigned to multiple genres. To calculate the user/genre effect for an individual movie rating, the user/genre effects for that user in each of the movie's genres were averaged together. The following example illustrates how this works:

UserID==8 rated the movie "Jumanji" (movieID==2) in the training set. "Jumanji" belongs to the following genres: 1. Adventure (genre mean 3.49, userId8 genre mean 3.49)

2. Children (genre mean 3.42, userId8 genre mean 3.18)
3. Fantasy (genre mean 3.50, userId8 genre mean 3.36)

The mean rating of the three genres "Jumanji" belongs to is 3.47 (the mean of 3.49, 3.42, and 3.50). This is a difference of -0.0408 from the overall mean.

UserId8's mean rating for the three genres "Jumanji" belongs to is 3.34 (the mean of 3.49, 3.18, and 3.36). This is a difference of -0.13 from "Jumanji"'s genre means, which is the user/genre effect from this particular review.

The predicted rating for "Jumanji" from userId8 based on the three main components equals overall mean (3.513) + movie effect (-0.304) + user/genre effect (-0.13), which yields a prediction of 3.079. The actual rating given for "Jumanji" by userId8 is 2.5, leaving a residual of 3.079 - 2.5 = 0.579. This residual will be the user pickiness effect generated from userId8's rating of "Jumanji".

This particular user pickiness effect would be averaged with all of the pickiness effects calculated from all of userId8's ratings in the training set to come up with a mean pickiness rating.

The code for this calculation is shown here.

```r
#Get residuals from overall mean + movie effects
resid_effect <- left_join(edx_train, movie_effects,by="movieId") %>%
                  mutate(resid = rating - overallmean - moviediff)
#Get user/genre effects
resid_effect <- left_join(resid_effect, movie_genres, by="movieId")
resid_effect <- left_join(resid_effect, usergenre_effects, by=c("userId","genre"))
#Calculate mean user/genre effect across all genres in which a movie belongs
resid_effect <- resid_effect %>%
                  group_by(userId,movieId,rating,resid) %>%
                  summarize(ugdiff = mean(ugdiff)) %>%
                  ungroup()
#Assign residual to user pickiness effect
userpicky_effects <- resid_effect %>%
                       mutate(updiff = ifelse((rating + ugdiff - resid > 5) |
                                              (rating + ugdiff - resid < 0.5),
                                               0, resid - ugdiff)) %>%
                       group_by(userId) %>%
                       summarize(updiff = mean(updiff)) %>%
                       ungroup()
```

Note the **ifelse()** clause in the **mutate** statement above. This corrects for situations where a user's actual rating for a movie and that user's genre preferences for that movie are both very high or very low. Such situations could lead to overstatement of the pickiness effect, and were addressed with the **ifelse() clause**. The following example illustrates this:

UserId==1 reviewed the movie "Forrest Gump" (movieId==356). "Forrest Gump" has a mean movie rating of 4.02 in the training set, which is a difference of 0.51 from the overall mean. "Forrest Gump" belongs to the Comedy, Drama, Romance, and War genres. The average of the mean ratings of these four genres is 3.61, which is a difference of 0.0992 from the overall mean.

The mean of UserId1's genre rating for Comedy, Drama, Romance, and War is 1.39. Thus, the predicted rating for "Forrest Gump" from userId1 equals overall mean (3.513) + movie effect (0.51) + user/genre effect (1.39), which yields a prediction of 5.41.

UserId1 gave "Forrest Gump" an actual rating of 5. In fact, userId1 is a very easy to please reviewer, and rated every movie they reviewed a 5.

By the normal logic of our calculation, the residual is equal to the predicted rating minus the actual rating, or 5 - 5.41 = -0.41. Such a residual would imply that userId1 is a fairly harsh reviewer. But in reality userId1 gave the highest rating to every movie they reviewed, and this residual should not be included. For this reason, the user pickiness effect for this review is calculated as zero.

The **ifelse()** logic also accounts for similar situations where a user rates a movie very poorly in genres they dislike, and the residual would seem to imply that they are an overly positive reviewer.

User pickiness effects would now be included in the model, referenced in the model equation as UPE.

Current Model: **Predicted Rating = OMR + ME + UGE + UPE**

## 2.11 Prediction Test - The first full model

Having quantified the main effects above, the first test of the model was performed on the test data set. The specifics of the test are shown in ***Section 3.1***. Overall, the model performed well, clearing the target RMSE threshold with a little room to spare.

This successful result reinforced the appropriateness of the overall project approach. However, there were still potential opportunities to for further improvements.

## 2.12 Further improvements - Regularization

As discussed in section 2.6, the majority of users contributed relatively few reviews to the MovieLens data set. The same is also true when considered from the perspective of reviews per movie:

```r
movie_quantiles <- quantile(movie_effects$mcount)
movie_quantiles
```

```
##      0%     25%     50%     75%    100%
##     1.0    25.0    98.0   453.5 25185.0
```

As this quantile breakdown shows, over half of movies have less than one hundred reviews.

Given that so many movies received relatively few reviews, and so many users gave relatively few reviews, it seemed likely that more extreme ratings might be causing noise. To account for this, experiments were made with regularization, wherein extreme estimates would be adjusted with a penalty term.

Regularization was explore for movie effects and for user/genre effects. Consideration was given as to whether to use one common penalty value for both, or to come up with separate ones for each. Given that movies and users are different entities, the decision was made to optimize the penalty term for each effect separately.

## 2.13 Regularization of Movie Effects

Movie effects were regularized independently of other effects. Therefore, the prediction model used in determining the optimal penalty term was based only on movie effects. Once selected, this penalty term (lambda) would be incorporated into the full model.

Tenfold cross-validation was used to optimize the movie effects lambda as follows: 1. For the current fold, the existing **edx_train** training set into two random sub-partitions, with 80% going to training and 20% going to testing. 2. Within each fold, predictions on the test subset were made using a simple movie effects model from the training subset, across a range of possible lambda values. The prediction accuracy from each lambda value was recorded. 3. The best-performing lambda value from that fold was selected. 4. Steps 1-3 were repeated across all ten folds, resulting in a record of the best-performing lambdas from each of the ten folds.

The initial tenfold cross-validation was conducted using integers between 1 and 10 as potential lambda values. In this first series, all of the best performing lambdas were either 2 or 3.

To refine the estimate further, a second tenfold cross-validation was performed using potential lambda values between 1.5 and 3.5 in increments of 0.2. All of the best-performing lambdas in this series were between 2.1 and 2.5:

One final tenfold cross-validation was performed, using potential lambdas between 2.0 and 2.6 in increments of 0.1. The code from this final refinement is shown here:

```r
kfold <- 10
movie_cvs <- data.frame(fold = 0, bestlambda = 0, bestRMSE = 0)
for (K in 1:kfold) {
  #Create a data partition of the correct size
  set.seed(200+K, sample.kind="Rounding")
  cv_test_index <- createDataPartition(
    y = edx_train$rating, times = 1, p = 0.2, list = FALSE)
  edx_train_k <- edx_train[-cv_test_index,]
  edx_test_k <- edx_train[cv_test_index,]

  #Look at potential lambda penalty terms
#  m_lambda <- seq(1,10,1)
#  m_lambda <- seq(1.5,3.5,0.2)
  m_lambda <- seq (2,2.6,0.1)
  movie_cv <- data.frame(fold = 0, lambda = 0, RMSE = 99999)
  for (L in m_lambda) {
    movie_r <- edx_train_k %>%
                 group_by(movieId) %>%
                 summarize(mdiffr = sum(rating - overallmean)/(n()+L)) %>%
                 ungroup()
    #Use inner join instead of left join --> if a movie doesn't exist in the kfold train,
    #we don't want to consider it for lambda testing
    predk <- left_join(edx_test_k, movie_r, by = "movieId")
    predk$mdiffr[which(is.na(predk$mdiffr))] <- 0
    predk <- predk %>% mutate(predrating = overallmean + mdiffr)
    predk_RMSE <- RMSE_f(predk$predrating,predk$rating)
    movie_cv <- bind_rows(movie_cv, data_frame(fold = K, lambda = L,
                                               RMSE = predk_RMSE))
  }
  movie_cvs <- bind_rows(movie_cvs, data_frame(fold = K,
                 bestlambda = movie_cv$lambda[which.min(movie_cv$RMSE)],
                 bestRMSE = movie_cv$RMSE[which.min(movie_cv$RMSE)]))
}
movie_cvs <- movie_cvs[-1,]
movie_cvs
```

```
##    fold bestlambda  bestRMSE
## 2     1        2.1 0.9436407
## 3     2        2.5 0.9429292
## 4     3        2.2 0.9443214
## 5     4        2.5 0.9446970
## 6     5        2.3 0.9435845
## 7     6        2.1 0.9445292
## 8     7        2.4 0.9440518
## 9     8        2.1 0.9434791
## 10    9        2.6 0.9444988
## 11   10        2.2 0.9443660
```

```
mean(movie_cvs$bestlambda)
```

```
## [1] 2.3
```

From this final run, the mean of the best-performing lambdas (**2.3**) was selected as the penalty term to be used for regularizing movie effects in the full model.

```
#-Regularize movie effects using lambda = 2.3
m_lambda <- 2.3
movie_effectsr <- edx_train %>%
  group_by(movieId) %>%
  summarize(mdiffr = sum(rating - overallmean)/(n()+m_lambda)) %>%
  ungroup()
movie_effects <- inner_join(movie_effects,movie_effectsr,by="movieId")
rm(movie_effectsr)  #cleanup
```

With the regularized movie effects incorporated into the model as term **MEr**, the model was now:

**Predicted Rating = OMR + MEr + UGE + UPE**

A prediction run against the test set was done with this model, as discussed in ***Section 3.2***. The results were a modest improvement from the unregularized version.

## 2.14 Regularization of User/Genre Effects

User/genre effects were regularized independently of other effects. Therefore, the prediction model used in determining the optimal penalty term was based only on user/genre effects. Once selected, this penalty term (lambda) would be incorporated into the full model.

The original intent was to use tenfold cross-validation to optimizing the user/genre effects penalty term, repeating the approach used for movie effects. However, due to the added complexity of calculating average ratings across multi-genre movies, the processing time for user/genre effects was exponentially greater. Therefore, this was scaled back to a onefold model. The existing **edx_train** training set was split into two random sub-partitions, with 80% going to training and 20% going to testing. The penalty term (lambda) was optimized as follows):

1. Predictions on the test subset were made using a simple user/genre effects model from the training subset, across a range of possible lambda values. The prediction accuracy from each lambda value was recorded.

2. The best-performing lambda value was selected.

The initial lambda values considered were integers between 1 and 10. The best performing lambda was 4.

A second pass was done to refine this estimate, considering lambda values between 3.5 and 4.5 in increments of 0.2. The best performing lambda was 4.3.

One final refinement pass was done, considering lambda values between 4.2 and 4.4 in increments of 0.1. The best performing lambda was again 4.3.

The code that generated this final pass is shown here:

```r
#-Regularize user genre means
#Create a data partition for lambda cross-validation
set.seed(400+K, sample.kind="Rounding")
cv_test_index <- createDataPartition(
  y = edx_train$rating, times = 1, p = 0.2, list = FALSE)
edx_train_k <- edx_train[-cv_test_index,]
edx_test_k <- edx_train[cv_test_index,]

#Look at potential lambda penalty terms
#  ug_lambda <- seq(1,10,1)
#  ug_lambda <- seq(3.5,4.5,0.2)
  ug_lambda <- seq(4.2,4.4,0.1)
  userg_cv <- data.frame(fold = 0, lambda = 0, RMSE = 9999)
  for (L in ug_lambda) {
    usergenre_effectsr <- inner_join(edx_train_k, movie_genres, by="movieId")
    usergenre_effectsr <- inner_join(usergenre_effectsr, genre_means, by="genre") %>%
      group_by(userId,genre) %>%
      summarize(ucount = n(), ugdiffr = sum(rating - grating)/(n()+L)) %>%
      ungroup()
    #Use inner join instead of left join --> if a movie doesn't exist in the kfold train,
    #we don't want to consider it for lambda testing
    predk <- inner_join(edx_test_k, movie_genres, by="movieId")
    predk <- inner_join(predk, usergenre_effectsr, by=c("userId","genre"))
    predk <- predk %>%
      group_by(userId,movieId,rating,ugdiffr) %>%
      summarize(ugdiffr = mean(ugdiffr)) %>%
      ungroup()
    predk <- predk %>% mutate(predrating = overallmean + ugdiffr)
    predk_RMSE <- RMSE_f(predk$predrating,predk$rating)
    userg_cv <- bind_rows(userg_cv, data_frame(fold = K, lambda = L,
                                               RMSE = predk_RMSE))
  }
print(userg_cv)
```

```
##   fold lambda         RMSE
## 1    0    0.0 9999.0000000
## 2   10    4.2    0.9729782
## 3   10    4.3    0.9729802
## 4   10    4.4    0.9729856
```

Based on this final results, **4.3** was selected as the penalty term to be used for regularizing user/genre effects in the full model.

```
#-Regularize user/genre combinations using lambda = 4.3
ug_lambda <- 4.3
usergenre_effectsr <- inner_join(edx_train, movie_genres, by="movieId")
usergenre_effectsr <- inner_join(usergenre_effectsr, genre_means, by="genre") %>%
  group_by(userId,genre) %>%
  summarize(ucount = n(), ugdiffr = sum(rating - grating)/(n()+ug_lambda)) %>%
  ungroup()
usergenre_effects <- inner_join(usergenre_effects,usergenre_effectsr,
                                by=c("userId","genre"))
rm(usergenre_effectsr)    #Cleanup
```

With the regularized user/genre effects incorporated into the model as term **UGEr**, the model was now:

**Predicted Rating = OMR + MEr + UGEr + UPE**

A prediction run against the test set was done with this model, as discussed in ***Section 3.3***. The results were a further modest improvement from the previous version.

## 2.15 Adjustment of user pickiness effects

As discussed previously in section 2.10, the user pickiness effect was equal to the residual difference between the predicted rating (Overall Mean + Movie Effects + User/Genre Effects) and the actual rating in the training set. Since the movie effects and user/genre effects have been changed due to regularization, the residual values would now be different, and the user pickiness effect should be adjusted accordingly. The following code calculated the adjusted user pickiness effect:

```
#Adjust user pickiness to account for regularized user/genre effects and movie effects
resid_effectr <- left_join(edx_train, movie_effects,by="movieId") %>%
  mutate(residr = rating - overallmean - mdiffr)
resid_effectr <- left_join(resid_effectr, movie_genres, by="movieId")
resid_effectr <- left_join(resid_effectr, usergenre_effects, by=c("userId","genre"))
resid_effectr$ugdiffr[which(is.na(resid_effectr$ugdiffr))] <- 0
resid_effectr <- resid_effectr %>%
                   group_by(userId,movieId,rating,residr) %>%
                   summarize(ugdiffr = mean(ugdiffr)) %>%
                   ungroup()
userpicky_effectsr <- resid_effectr %>%
                   mutate(updiffa = ifelse((rating + ugdiffr - residr > 5) |
                                   (rating + ugdiffr - residr < 0.5),
                                   0, residr - ugdiffr)) %>%
                   group_by(userId) %>%
                   summarize(updiffa = mean(updiffa)) %>%
                   ungroup()
userpicky_effects <- inner_join(userpicky_effects,userpicky_effectsr,by="userId")
```

These adjusted user pickiness effects were incorporated into the model as term **UPEa**, making the final model:

**Predicted Rating = OMR + MEr + UGEr + UPEa**

A prediction run against the test set was done with this model, as discussed in ***Section 3.4***. The results were again a modest improvement from the previous version.

## 3. Results

This section includes the code and results for the four versions of the model run on the test data set, as well as the final version's results against the validation set. To see the specific results for the final validation set, skip to ***Section 3.5***.

The following abbreviations are used for model components:

**OMR** = Overall Mean Rating (the mean of every rating from every review in the training data set **edx_train**)

**ME** = Movie Effects (the difference between the mean rating of a movie in the training data set and the overall mean)

**UGE** = User/Genre Effects (the difference between a user's mean rating for movies in a particular genre in the training data set and the mean rating of all movies in a particular genre in the training data set)

**UPE** = User Pickiness Effects (the mean residual between the predicted rating from OMR + ME + UGE and the actual rating in the training data set)

**MEr** = Movie Effects, regularized (the movie effects parameter regularized with a penalty term lambda = 2.3)

**UGEr** = User/Genre Effects, regularized (the user/genre effects parameter regularized with a penalty term lambda = 4.3)

**UPEa** = User Pickiness Effects, adjusted (residual now based on regularized movie effects and regularized genre effects)

### 3.1 Results - edx_test, Full Model, no regularization

The first prediction effort used the following formula:

**Predicted Rating = OMR + ME + UGE + UPE**

Here is the code used to generate it:

```r
#Get movie effects
pred1 <- left_join(edx_test, movie_effects, by = "movieId")
#If a movie from the test set does not exist in the training set, set its movie effects
#  parameter to zero.
pred1$moviediff[which(is.na(pred1$moviediff))] <- 0
pred1 <- left_join(pred1,userpicky_effects,by="userId")
#If a user from the test set does not exist in the training set, set its user pickiness
#  effects parameter to zero.
pred1$updiff[which(is.na(pred1$updiff))] <- 0
#Get movie genres
pred1 <- left_join(pred1, movie_genres, by="movieId")
#Get user/genre effects
pred1 <- left_join(pred1, usergenre_effects, by=c("userId","genre"))
#If a user/genre combination from the test set does not exist in the training set, set
#  its user/genre effects parameter to zero.
pred1$ugdiff[which(is.na(pred1$ugdiff))] <- 0
#Add up user/genre effects
pred1 <- pred1 %>%
           group_by(userId,movieId,rating,moviediff,updiff) %>%
           summarize(ugdiff = mean(ugdiff)) %>%
           ungroup()
```

```r
#Calculate predicted rating
pred1 <- pred1 %>% mutate(predrating = overallmean + moviediff + updiff + ugdiff)
#Adjust predictions that fall outside the 0.5 - 5 range
pred1$predrating[which(pred1$predrating>5)] <- 5
pred1$predrating[which(pred1$predrating<0.5)] <- 0.5

#Calculate RMSE
pred1_RMSE <- RMSE_f(pred1$predrating,pred1$rating)
pred_results <- bind_rows(pred_results,
                          data_frame(prednum = "1", predtype = "OMR + ME + UGE + UPE",
                          RMSE = pred1_RMSE))
pred_results[4,]
```

```
##   prednum             predtype      RMSE
## 4       1 OMR + ME + UGE + UPE 0.8544758
```

```r
rm(pred1)    #cleanup
```

Note that this model's formula could generate predictions above the maximum rating of 5 or below the maximum rating of 0.5. To account for these, any predictions above 5 were rounded down to 5, and any predictions below 0.5 were rounded up to 0.5.

The RMSE of 0.85448 was a significant improvement over basic approaches and validated the importance of looking at user/genre preferences as a key to more accurate predictions.

## 3.2 Results - edx_test, Full Model, movie effects regularized

The movie effects in the full model were regularized using the selected best penalty term of 2.3. Thus, the model formula was now the following:

**Predicted Rating = OMR + MEr + UGE + UPE**

Here is the code used to generate it:

```r
#-Prediction 2

#Get regularized movie effects
pred2 <- left_join(edx_test, movie_effects, by = "movieId")
#If a movie from the test set does not exist in the training set, set its regularized movie
#  effects parameter to zero.
pred2$mdiffr[which(is.na(pred2$mdiffr))] <- 0
#Get user pickiness effects
pred2 <- left_join(pred2,userpicky_effects,by="userId")
#If a user from the test set does not exist in the training set, set its user pickiness
#  effects parameter to zero.
pred2$updiff[which(is.na(pred2$updiff))] <- 0
#Get movie genres
pred2 <- left_join(pred2, movie_genres, by="movieId")
#Get user/genre effects
pred2 <- left_join(pred2, usergenre_effects, by=c("userId","genre"))
#If a user/genre combination from the test set does not exist in the training set, set
#  its user/genre effects parameter to zero.
pred2$ugdiff[which(is.na(pred2$ugdiff))] <- 0
```

```r
#Add up user/genre effects
pred2 <- pred2 %>%
  group_by(userId,movieId,rating,mdiffr,updiff) %>%
  summarize(ugdiff = mean(ugdiff)) %>%
  ungroup()
#Calculate predicted rating
pred2 <- pred2 %>% mutate(predrating = overallmean + mdiffr + updiff + ugdiff)
#Adjust predictions that fall outside the 0.5 - 5 range
pred2$predrating[which(pred2$predrating>5)] <- 5
pred2$predrating[which(pred2$predrating<0.5)] <- 0.5

#Calculate RMSE
pred2_RMSE <- RMSE_f(pred2$predrating,pred2$rating)
pred_results <- bind_rows(pred_results,
                          data_frame(prednum = "2", predtype = "OMR + MEr + UGE + UPE",
                                     RMSE = pred2_RMSE))
pred_results[5,]
```

```
##   prednum              predtype      RMSE
## 5       2 OMR + MEr + UGE + UPE 0.8543221
```

```r
rm(pred2)    #cleanup
```

Regularizing the movie effects resulted in a very modest improvement, dropping the RMSE by about .00015 to 0.85432.

### 3.3 Results - edx_test, Full Model, movie effects and user/genre effects regularized

Now, the user genre effects were regularized using the selected best penalty term of 4.3. Thus, the model formula was now the following:

**Predicted Rating = OMR + MEr + UGEr + UPE**

Here is the code used to generate it:

```r
#-Prediction 3

#Get regularized movie effects
pred3 <- left_join(edx_test, movie_effects, by = "movieId")
#If a movie from the test set does not exist in the training set, set its regularized movie
#  effects parameter to zero.
pred3$mdiffr[which(is.na(pred3$mdiffr))] <- 0
#Get user pickiness effects
pred3 <- left_join(pred3,userpicky_effects,by="userId")
#If a user from the test set does not exist in the training set, set its user pickiness
#  effects parameter to zero.
pred3$updiff[which(is.na(pred3$updiff))] <- 0
#Get movie genres
pred3 <- left_join(pred3, movie_genres, by="movieId")
#Get regularized user/genre effects
pred3 <- left_join(pred3, usergenre_effects, by=c("userId","genre"))
```

```
#If a user/genre combination from the test set does not exist in the training set, set
#  its regularized user/genre effects parameter to zero.
pred3$ugdiffr[which(is.na(pred3$ugdiffr))] <- 0
#Add up user/genre effects
pred3 <- pred3 %>%
  group_by(userId,movieId,rating,mdiffr,updiff) %>%
  summarize(ugdiffr = mean(ugdiffr)) %>%
  ungroup()
#Calculate predicted rating
pred3 <- pred3 %>% mutate(predrating = overallmean + mdiffr + updiff + ugdiffr)
#Adjust predictions that fall outside the 0.5 - 5 range
pred3$predrating[which(pred3$predrating>5)] <- 5
pred3$predrating[which(pred3$predrating<0.5)] <- 0.5

#Calculate RMSE
pred3_RMSE <- RMSE_f(pred3$predrating,pred3$rating)
pred_results <- bind_rows(pred_results,
                          data_frame(prednum = "3", predtype = "OMR + MEr + UGEr + UP",
                                     RMSE = pred3_RMSE))
pred_results[6,]
```

```
##   prednum              predtype      RMSE
## 6       3 OMR + MEr + UGEr + UP 0.8527688
```

```
rm(pred3)    #cleanup
```

The user/genre effects regularization had a more substantial impact on RMSE than movie effects regularization had, reducing the overall RMSE to 0.85277.

### 3.4 Results - edx_test, Full Model, movie effects and user/genre effects regularized, user pickiness adjusted

The final version of the model used the regularized movie and user/genre effects, and also adjusted the user pickiness effect to account for the differing residuals after regularization of the other effects. Thus, the model formula was now the following:

**Predicted Rating = OMR + MEr + UGEr + UPEa**

Here is the code used to generate it:

```
#-Prediction 4

#Get regularized movie effects
pred4 <- left_join(edx_test, movie_effects, by = "movieId")
#If a movie from the test set does not exist in the training set, set its regularized movie
#  effects parameter to zero.
pred4$mdiffr[which(is.na(pred4$mdiffr))] <- 0
#Get adjusted user pickiness effects
pred4 <- left_join(pred4,userpicky_effects,by="userId")
#If a user from the test set does not exist in the training set, set its adjusted user
#  pickiness effects parameter to zero.
pred4$updiffa[which(is.na(pred4$updiffa))] <- 0
```

```r
#Get movie genres
pred4 <- left_join(pred4, movie_genres, by="movieId")
#Get regularized user/genre effects
pred4 <- left_join(pred4, usergenre_effects, by=c("userId","genre"))
#If a user/genre combination from the test set does not exist in the training set, set
#  its regularized user/genre effects parameter to zero.
pred4$ugdiffr[which(is.na(pred4$ugdiffr))] <- 0
#Add up user/genre effects
pred4 <- pred4 %>%
           group_by(userId,movieId,rating,mdiffr,updiffa) %>%
           summarize(ugdiffr = mean(ugdiffr)) %>%
           ungroup()
#Calculate predicted rating
pred4 <- pred4 %>% mutate(predrating = overallmean + mdiffr + updiffa + ugdiffr)
#Adjust predictions that fall outside the 0.5 - 5 range
pred4$predrating[which(pred4$predrating>5)] <- 5
pred4$predrating[which(pred4$predrating<0.5)] <- 0.5

#Calculate RMSE
pred4_RMSE <- RMSE_f(pred4$predrating,pred4$rating)
pred_results <- bind_rows(pred_results,
                          data_frame(prednum = "4", predtype = "OMR + MEr + UGEr + UPEa",
                                     RMSE = pred4_RMSE))
pred_results[7,]
```

```
##   prednum                predtype     RMSE
## 7       4 OMR + MEr + UGEr + UPEa 0.851391
```

```r
rm(pred4)
```

The adjustment to user pickiness had a similar impact to the user/genre effects regularization, reducing the overall RMSE on the test set to 0.85139.

## 3.5 Final Result - Validation set, Full Model, movie effects and user/genre effects regularized, user pickiness adjusted

The model from 3.4 was the best performer on the **edx_test** data set:

**Predicted Rating = OMR + MEr + UGEr + UPEa**

See the beginning of ***Section 3*** for explanations of these terms.

This model was run against the validation set, using the final script (included as a separate document in the project submission.) The resulting RMSE was **0.85170**.

## 4. Conclusions

### 4.1 Key success factors

The single biggest contributor to the success of the final model was the incorporation of user/genre effects. This would not have been possible without the work to split the "genres" field into its individual components, and accommodate this in a workable data structure. User/genre effects combined with movie effects reduced the test set RMSE to 0.85448, an improvement of 9.4% over movie effects alone.

Regularization was an important fine-tuning component. When regularized movie effects and genre effects were incorporated into the model, and user pickiness adjusted accordingly, the test set RMSE was reduced by a further 0.4%. With more time, it might have been possible to perform a full tenfold cross validation on the user/genre effects penalty term, and perhaps gain a little more improvement.

## 4.2 Limitations/areas for improvement

There are a number of areas where the model could potentially be improved. Matrix factorization was not incorporated into this model in any way, which limited its ability to detect patterns that are not attributed solely to individual columns in the data set. While the decomposition of the genres field into individual genres may have captured much of these effects, there are likely others that matrix factorization could provide some further insight into, and result in improved performance.

The final model does not include any time-related effects, such as year of movie release or year of review. There does appear to be some sort of nonlinear correlation between a movie's ratings and the year of release. There were some brief attempts to incorporate a time effect into the model, but they made performance moderately worse, so those efforts were abandoned. A more thorough investigation might be able to make a time effect a useful parameter.

The fundamental approach of this project was reliance on estimating effects via means. Because of the size of the MovieLens data set, more advanced modeling algorithms like k-nearest neighbors or random forests were not considered. An alternate approach would be to break the data into small subsets, run a suite of modeling algorithms against them, and extrapolate the results to the larger data set.