MovieLens Recommendation System - J. Finelli, July 2021

Project Overview

Recommendation systems have become a prominent feature of many leading consumer-facing platforms, with online retail experiences (such as Amazon) and streaming content services (such as Netflix) at the top of this list. In this project, we attempt to develop a recommendation system similar to ones widely used by these types of companies. Here, we will examine movie rating data.

Our dataset is a list of ten million ratings that a variety of viewers gave to a variety of movies between 1995 and 2009. Each row of data identifies the viewer, the movie, the date of the rating, and the rating given. Ratings were given on a scale of 1 (lowest) to 5 (highest), with half-point ratings allowed (e.g. 3.5).

Our goal is to build a model that will predict with the greatest possible accuracy the rating that any given viewer gave any given movie in our dataset. Once done, we would then be able to generate movie recommendations (i.e. movies we predict would earn high ratings) both for these viewers and, going forward, for new viewers too.

To achieve our goal, we will build several candidate models and will judge their performance when applied to a portion of our data that we reserve in advance.

Each model will be assessed by comparing the rating it predicts for given viewer/movie combinations to the rating that the viewer in question actually gave to the movie in question (which the model has not seen).

Below, we develop **linear models**, **regularization techniques**, **matrix factorization models**, and also **ensemble models** in order to identify the best-performing solution. In the end, we wind up with a model that predicts ratings with a typical error of .81 stars.

Methods and Analysis

Data preparation and cleaning:

After loading required packages and libraries, we create the MovieLens dataset by downloading its component pieces and then joining them together.

```
#download movielens files
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")

#create data frames
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")
```

Once in hand, we see that MovieLens dataset has just over 10 million rows. We also see that it is in tidy format, meaning that each row contains exactly one observation (here, a rating given by a specific user of a specific movie).

```
str(movielens, width=80,strict.width="cut")
```

```
## Classes 'data.table' and 'data.frame':   10000054 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392..
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)"..
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|"..
##  - attr(*, ".internal.selfref")=<externalptr>
```

In order to facilitate data analysis, we begin by breaking off a "validation" test set (1M observations) to be used only at the very end of the analysis.

```
# 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)

#check various dataset sizes
length(validation$rating)
```

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

```
length(edx$rating)
```

```
## [1] 9000055
```

We then break off a second "rehearsal" test set (900k) as well. We will use this rehearsal test set before finalizing our model.

```
# rehearsal/debug ("rd") set will be 10% of edx data, rest becomes "edx_trim"
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_trim <- edx[-test_index,]
temp <- edx[test_index,]

#make sure userId and movieId in rd set are also in edx_trim set
rd <- temp %>%
  semi_join(edx_trim, by = "movieId") %>%
  semi_join(edx_trim, by = "userId")

#add rows removed from rd set back into newly created edx_trim set
removed <- anti_join(temp, rd)
edx_trim <- rbind(edx_trim, removed)
```

```
#check lengths
length(rd$rating)
```

```
## [1] 899988
```

```
length(edx_trim$rating)
```

```
## [1] 8100067
```

We divide the remaining 8.1M observations (named "edx_trim") into a traditional "train" and "test" set combination, with 87.5% of these observations being assigned to the train set.

```
#separate edx_trim into edx_test and edx_trim
# edx test set will be 12% of currently remaining edx data
test2_index <- createDataPartition(y = edx_trim$rating, times = 1, p = 0.12, list = FALSE)
edx_train <- edx_trim[-test2_index,]
temp <- edx_trim[test2_index,]

#make sure userId and movieId in edx_test set are also in edx_train set
edx_test <- temp %>%
    semi_join(edx_train, by = "movieId") %>%
    semi_join(edx_train, by = "userId")

#add rows removed from rd set back into edx_train set
removed <- anti_join(temp, edx_test)
edx_train <- rbind(edx_train, removed)
```

As a wholly separate exercise, in order to facilitate k-fold cross-validation, the "edx_trim" set is also divided into unique train / test set combinations five different times, with each observation appearing in a test set exactly once among these five iterations.

For simplicity, we show the creation of only the first of these five separate train/test sets here.

```
#Create 5 folds of edx_trim for regularization tuning
movie_folds <- createFolds(edx_trim$rating, k = 5, list = TRUE, returnTrain = FALSE)

#fold1 - create, ensure structure, verify combined length
mfold_1_train <- edx_trim[-movie_folds$Fold1,]
temp <- edx_trim[movie_folds$Fold1,]

mfold_1_test <- temp %>%
  semi_join(mfold_1_train, by="movieId") %>%
  semi_join(mfold_1_train, by="userId")

removed <- anti_join(temp, mfold_1_test)
mfold_1_train <- rbind(mfold_1_train, removed)
```

Data exploration and visualization:

Once our datasets are formed, we check to see whether any rows in our training set appear to be missing any critical components, such as the rating, movieId, or userId, etc. There appear to be no such holes.

```
#core elements check for "NA"
NAs <- sum(edx_train$rating=="NA") +
sum(edx_train$userId=="NA") +
sum(edx_train$movieId=="NA")
NAs
```

```
## [1] 0
```

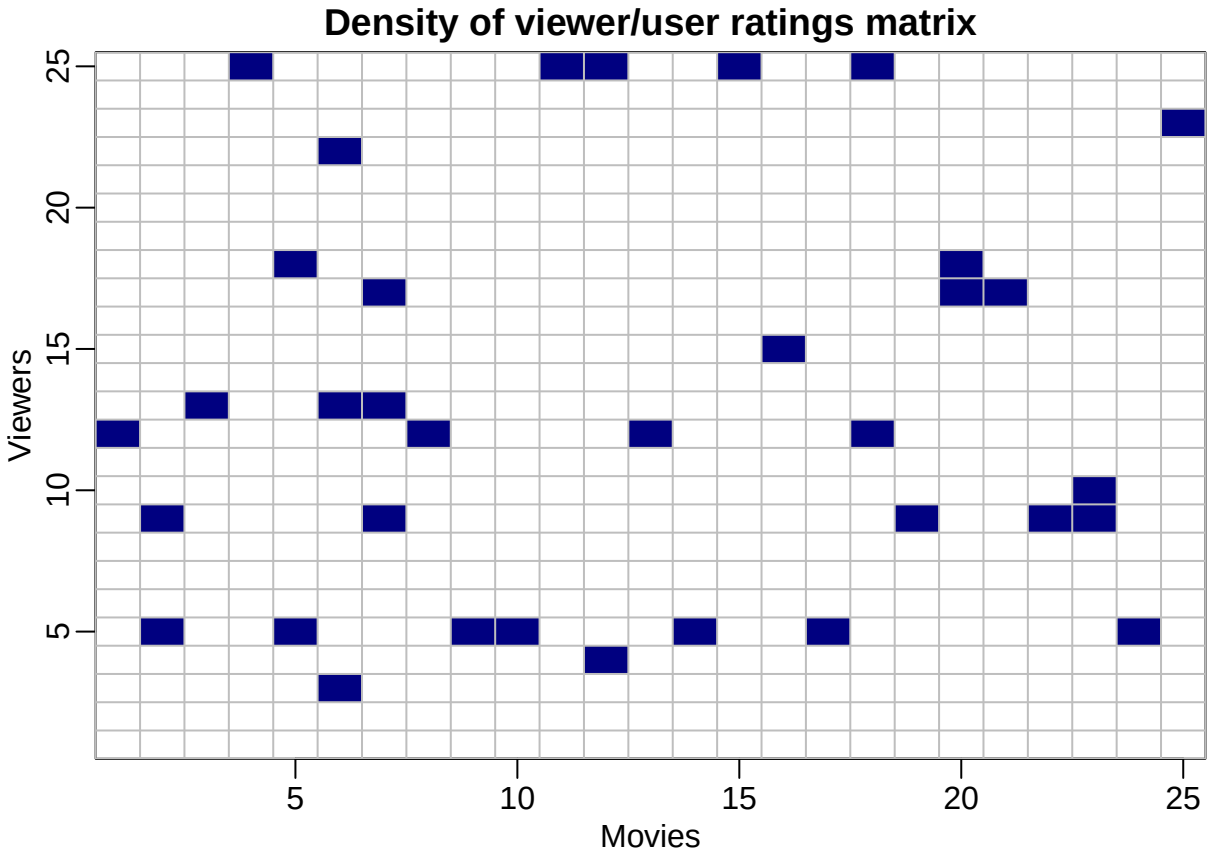
We also look at how many distinct viewers (which we will use interchangably with the term "users") and movies are within the dataset.

```
#number of unique viewers and movies
edx_train %>%
  summarize(num_users = n_distinct(userId),
            num_movies = n_distinct(movieId))
```

```
##   num_users num_movies
## 1     69878      10677
```

Given the number of users (69,878) and movies (10,677) included in the trainig set, compared to the number of ratings, it becomes clear that not every viewer rated every movie – far from it, in fact.

If one were to imagine this dataset as a matrix of viewers and movies, with each cell within as a possible rating, this dataset would only fill about 1% of cells with ratings. Below, we see a visualization of 25 randomly selected viewers and movies, with a colored cell representing a viewer/movie combination for which we have a rating in hand.



This "sparse matrix" setup makes our challenge both more difficult and more interesting.

Beyond the size and structure of the dataset, we are also interested in understanding the basic viewing and rating patterns within the data. This will help us understand if the dataset is believable, for instance. Having no way of identifying users (understandably), we focus our efforts on movie metrics.

Which movies received the highest avg. rating (min 5k views)?

```
edx_train %>%
  group_by(title) %>%
```

```
  filter(n() >= 5000) %>%
  summarize(Ratings = n(), AvgRating = mean(rating)) %>%
  arrange(desc(AvgRating)) %>%
  slice(1:7)
```

```
## # A tibble: 7 x 3
##   title                              Ratings AvgRating
##   <chr>                                <int>     <dbl>
## 1 Shawshank Redemption, The (1994)   22041      4.46
## 2 Godfather, The (1972)              14178      4.42
## 3 Schindler's List (1993)            18366      4.37
## 4 Usual Suspects, The (1995)         17246      4.37
## 5 Casablanca (1942)                   8835      4.32
## 6 Rear Window (1954)                  6260      4.32
## 7 Godfather: Part II, The (1974)      9479      4.31
```

What about the lowest average rating (min 5k views)?

```
## # A tibble: 7 x 3
##   title                                  Ratings AvgRating
##   <chr>                                    <int>     <dbl>
## 1 Judge Dredd (1995)                        6289      2.53
## 2 Ace Ventura: When Nature Calls (1995)     8219      2.59
## 3 Coneheads (1993)                          5942      2.60
## 4 Congo (1995)                              6557      2.60
## 5 Honey, I Shrunk the Kids (1989)           6597      2.67
## 6 Cable Guy, The (1996)                     5934      2.69
## 7 Johnny Mnemonic (1995)                    5275      2.71
```

Which movies were rated most frequently?

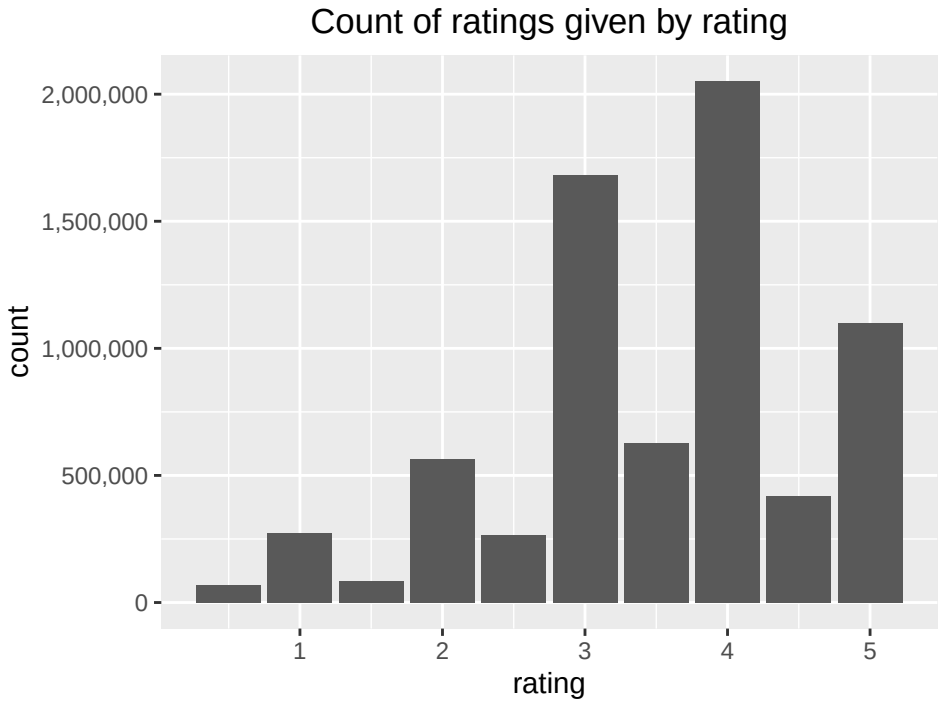
```
##                                title     n
## 1:              Pulp Fiction (1994) 24769
## 2:              Forrest Gump (1994) 24554
## 3: Silence of the Lambs, The (1991) 24019
## 4:             Jurassic Park (1993) 23321
## 5: Shawshank Redemption, The (1994) 22041
## 6:                Braveheart (1995) 20722
## 7:             Fugitive, The (1993) 20652
```

Seeing these results gives us comfort in the integrity and credibility of the data, as we are seeing many familiar and popular (or less so) movie titles in the tables above.

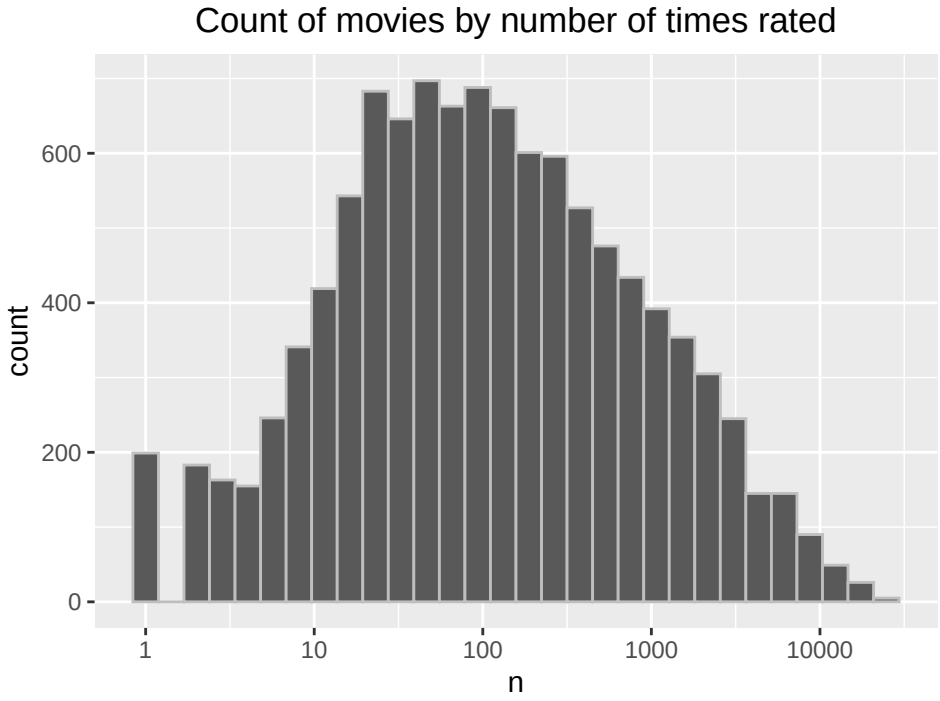
With this in mind, we next look to understand various distributions within the dataset.

Distribution of ratings given by users (shows half-point ratings were less common)

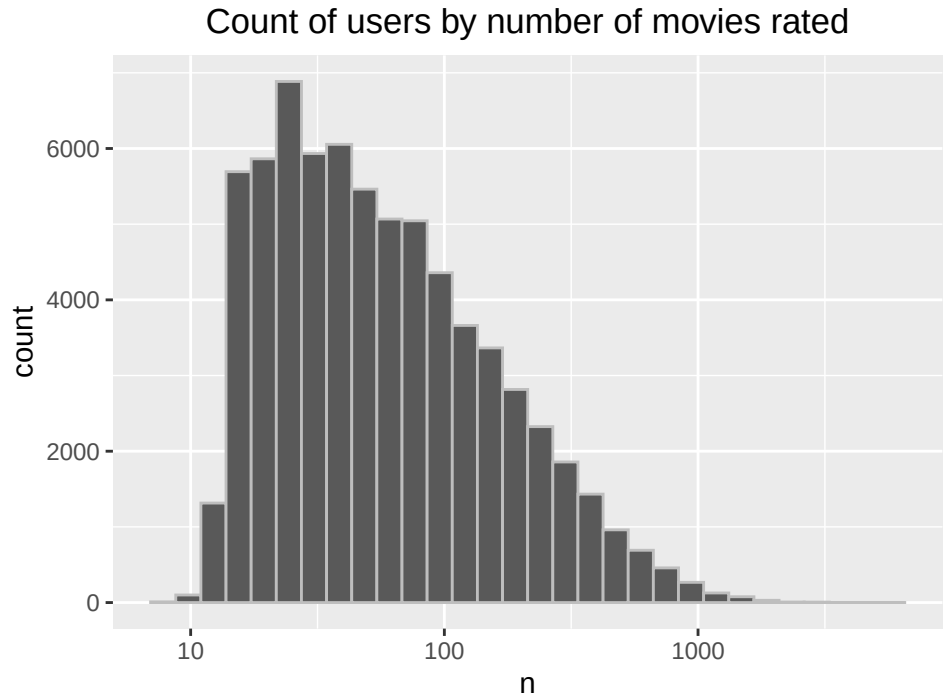
```
#frequency of ratings
edx_train %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  scale_y_continuous(labels = comma) +
  geom_col() +
  ggtitle("Count of ratings given by rating") +
  theme(plot.title = element_text(hjust = 0.5))
```



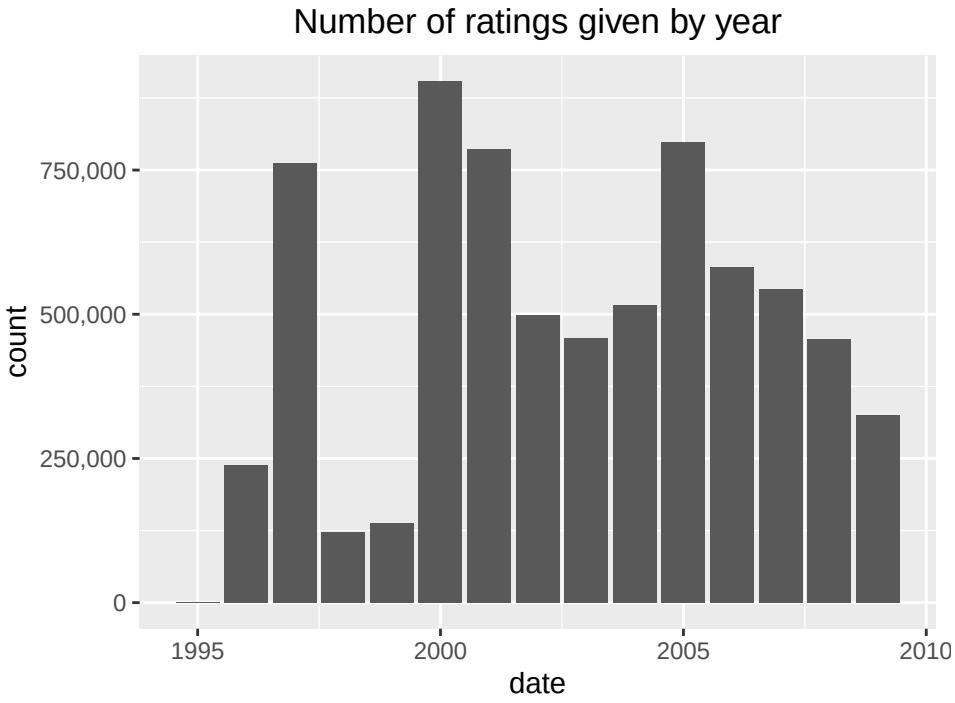
Distribution of number of ratings by movie (shows most commonly occurring ratings counts for movies were between 50-150)



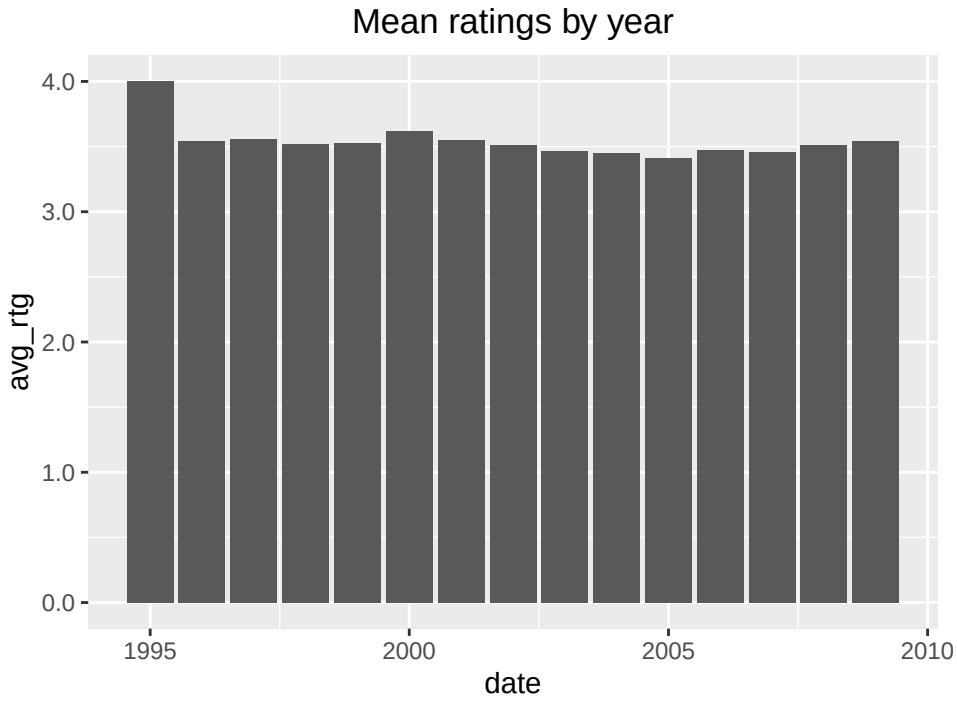
Distribution of number of ratings by user (shows most commonly occurring ratings counts for users were between 30-60):



Number of ratings over time (shows a clear dip from 1997 to 1998):



Average rating over time (very consistent from 1996 onwards)



With these baseline distributions now internalized, we are ready to design our modeling approach.

Modeling approach:

Defining success:

To quantify each model's performance, we use the root mean squared error ("RMSE"), where error is defined by the difference between the predicted and observed rating for any viewer/movie combination. The formula is given by:

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

where $\hat{y}_{u,i}$ represents the predicted rating for movie i by user u, and $y_{u,i}$ represents the actual rating given.

Linear models

Our first approach is to understand each rating given as a simple combination of the predictors provided to us: movie, user, and date of rating.

For movie and user effects, we calculate the average difference from the mean (of all ratings) by movie and by user, and we combine these with the aforementioned mean, plus an error term as well. Our formula becomes:

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

where $Y_{u,i}$ is the rating given, μ is the mean rating for all movies, and b_i and b_u are the movie and user effects, respectively.

Because the error term $\epsilon_{u,i}$ is assumed to be random and centered at zero, the predictors that minimize the overall RMSE for this equation will be the those that minimize the RMSE of each element of it, i.e. the average of each (average rating overall, then average residual difference by movie, then average remaining residual difference by user).

Later, we will add in the effect of the date of the rating as well. This effect will prove to be better seen as a linear function of t, and thus our rating will be fully modeled as follows:

$$Y_{u,i} = \mu + b_i + b_u + f(t) + \epsilon_{u,i}$$

Regularization

One shortcoming of the strict linear approach above is that it weighs all averages equally. Put another way, the effect of a viewer who has rated merely two movies is given the same weight as the effect of a viewer who has rated hundreds. As we know, small sample sizes can yield averages that wind up being outliers, and these outlier means can therefore affect our models adversely.

In order to reduce the potential harm of these kinds of outlier means, we introduce a penalty term to our model that remains large when the sample size for any given viewer or movie is small, but becomes small (or near zero) when the sample size gets large. This penalty term, lambda (λ), winds up in the denominator of the equation that will yield our improved b_i and b_u terms.

The example for our improved b_i is below - note how λ becomes less influential the greater that n_i (the number of movies present in the data set) becomes.

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

New factor: date of rating

Next, we improve our model further by considering the impact on rating of the date a movie was rated.

Our graphs above show that mean ratings over time have been stable and that the number of ratings given each year has fluctuated a bit. However, what will be more important to us will be whether there is a relationship between date (as a year) of rating and the residuals (i.e. the differences between predicted and actual rating) after our regularized linear model described above has been applied. In our case, we find there is indeed a slight one.

Matrix factorization Next, we examine a model that considers each given rating as a cell within a larger matrix (as visualized earlier), with viewers on one axis and movies on the other.

Using the matrix factorization approach, each value (i.e. rating) in this large matrix can be thought of as the product of two simpler matrices: one that describes viewer preferences (such as to what degree the viewer favors a certain movie category, or an actor/actress), and the second that describes to what degree each movie fits each of those descriptions.

Generating these two sub-matrices will be helped by identifying the "principal components" of this type of model, which is another term for the aspects of a movie (genre, cast, etc.) that can be uniquely identified and modeled.

From there, we multiply these two matrices, in essence, in order to generate credible predictions for each cell of the original matrix.

Ensembles

Last, we examine whether we can achieve an even lower RMSE by generating new predictions through a combination of our existing predictions.

For example, if one top-performing model tended to over-estimate ratings a bit, but another strong model tended to slightly under-estimate them, we could improve our overall RMSE by creating a new set of predictions that was simply the average of these models' predictions for each given viewer/movie combination.

Below, we wind up exploring two such ensembles: one that includes all established models, and another that uses only the two best-performing ones.

Results

Linear models

With our first model, we begin by predicting each rating as simply the average of all ratings.

$\hat{y}_{u,i} = \mu$,

This approach will certainly need to be improved upon, but it gives us a starting point.

This preliminary model shows us that the mean rating (mu) is just over 3.5, and it gives us a starting RMSE of 1.06.

```
#first model
  #just the train set average
  mu_train <- mean(edx_train$rating)
  mu_train
```

```
## [1] 3.5125
```

```
  #calculate rmse of first model
  model_1_rmse <- RMSE(mu_train, edx_test$rating)

  #create a data frame to keep track of all results, add first result to it
  rmse_results <- tibble(Method = "Simple Average", RMSE = model_1_rmse)
  rmse_results %>% knitr::kable()
```

Method	RMSE
Simple Average	1.0603

Because we saw earlier that some movies are clearly more popular than others, we next model the predicted rating for any movie as the average rating of all movies combined with the difference between that movie's mean rating from the overall mean.

We call this movie effect (i.e. this difference) b_i, and we include it as follows:

$\hat{y}_{u,i} = \mu + b_i + \epsilon_{u,i}$.

Put another way, we now predict each movie' rating simply as the average rating that each movie earned in our existing data set, plus the error term.

```
#second model
  #prediction = average + movie effect
  mu <- mean(edx_train$rating)
  movie_avgs <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))

  #assign each movie its predicted rating from this second model
  predicted_ratings_bi <- mu + edx_test %>%
    left_join(movie_avgs, by='movieId') %>%
    .$b_i

  #calculate the RMSE of this new model
  model_2_rmse <- RMSE(predicted_ratings_bi, edx_test$rating)
```

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402

This yields a noticeable improvement in RMSE.

We then repeat this process to incorporate the effect of ratings given by users, some who are stingy with their ratings, others who are generous.

Thus, we calculate b_u as the average difference from zero of the actual rating once the mean rating and movie effects have been taken into consideration. We include it as follows: $\hat{y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$.

```
#third model
  #prediction = average + movie effect + user effect
  mu <- mean(edx_train$rating)
  user_avgs <- edx_train %>%
    left_join(movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = mean(rating - mu - b_i))

  #assign each movie its predicted rating from this third model
  predicted_ratings_biu <- edx_test %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  #calculate the RMSE of this third model
  model_3_rmse <- RMSE(predicted_ratings_biu, edx_test$rating)
```

We can now see that our RMSE has fallen from over 1 to under .9.

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597

Incorporating regularization

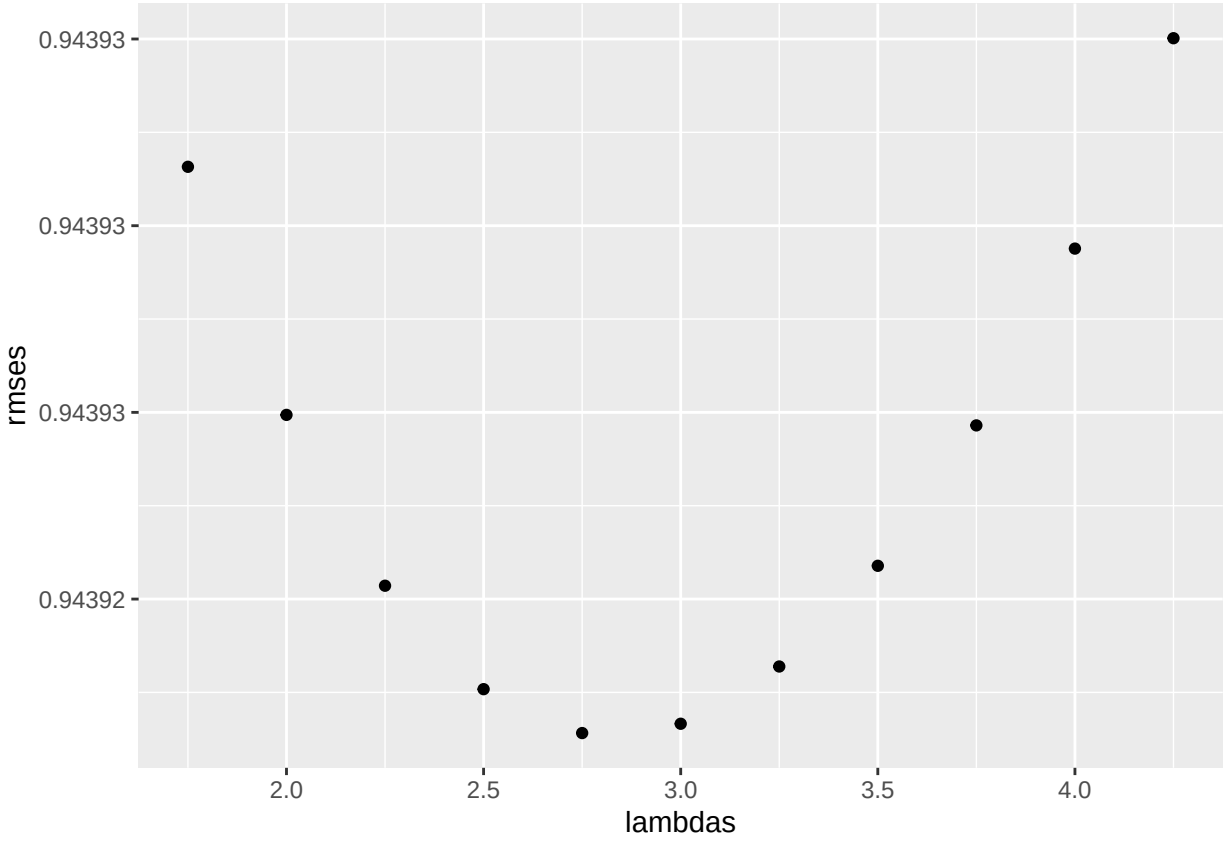
Our regularization process, designed to reduce the potentially harmful effects of outlier mean ratings (by a small number of users, or of a small number of movies) begins by calculating an updated b_i that will minimize our RMSE given the penalty term of lambda.

```
#fourth model
  #prediction = regularized version of movie effects only
  #calculate new b_i using concept of lambda
  #(set at 2 for now, will use cv later to pick final)
  lambda <- 2
  mu <- mean(edx_train$rating)
  movie_reg_avgs <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))
```

That said, since lambda is a tune-able parameter, we need to try out various values for lambda and find the one that yields the lowest RMSE.

```
#tune for best lambda
  lambdas <- seq(1.75, 4.25, .25)
  mu <- mean(edx_train$rating)
  sum1 <- edx_train %>%
    group_by(movieId) %>%
    summarize(ss = sum(rating-mu), n_i = n())
```

```
  rmses <- sapply(lambdas, function(l){
    predicted_ratings <-
      edx_test %>%
      left_join(sum1, by = "movieId") %>%
      mutate(b_i = ss/(n_i+l)) %>%
      mutate(prediction = mu + b_i) %>%
      .$prediction
    return(RMSE(predicted_ratings, edx_test$rating))
  })

  #verify with graph
  qplot(lambdas, rmses)
```



As this graph shows, our best result comes when lambda is set at 2.75.

```
#select best performing lambda for use
  best_lambda1 <- lambdas[which.min(rmses)]
  best_lambda1
```

```
## [1] 2.75
```

With our first regularized model tuned, we then re-generate our predictions with our best-performing lambda of 2.75 in place, and calculate the resulting RMSE.

```
#re-run first regularized model using best_lambda from above
  lambda <- best_lambda1
  mu <- mean(edx_train$rating)
  movie_reg_avgs <- edx_train %>%
    group_by(movieId) %>%
```

```
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  #assign each movie its predicted rating from this regularized (movie effects only) model
  predicted_ratings_r1 <- edx_test %>%
    left_join(movie_reg_avgs, by = "movieId") %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)

  #calculate the RMSE of this fourth model
  model_4_rmse <- RMSE(predicted_ratings_r1, edx_test$rating)
```

This regularization process appears to have produced an improvement, although slight, in comparison to its un-regularized equivalent (comparing the second and fourth lines below):

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597
Regularized Movie Effect Model	0.94392

We then repeat this full regularization process for the user effect (b_u). In addition, we use full cross-validation when calculating the RMSE that each candidate lambda will produce.

Specifically, rather than calculating RMSE for each lambda using only one sample (`edx_train`) of our larger dataset (`edx`), we instead create five different and equally-sized samples of `edx_train` (k-fold cross-validation, rather than bootstrapping), and calculate an RMSE for each, with our final comparison set of RMSEs being their respective averages.

First, we calculate our initial regularized b_u using a temporary lambda of 2.75.

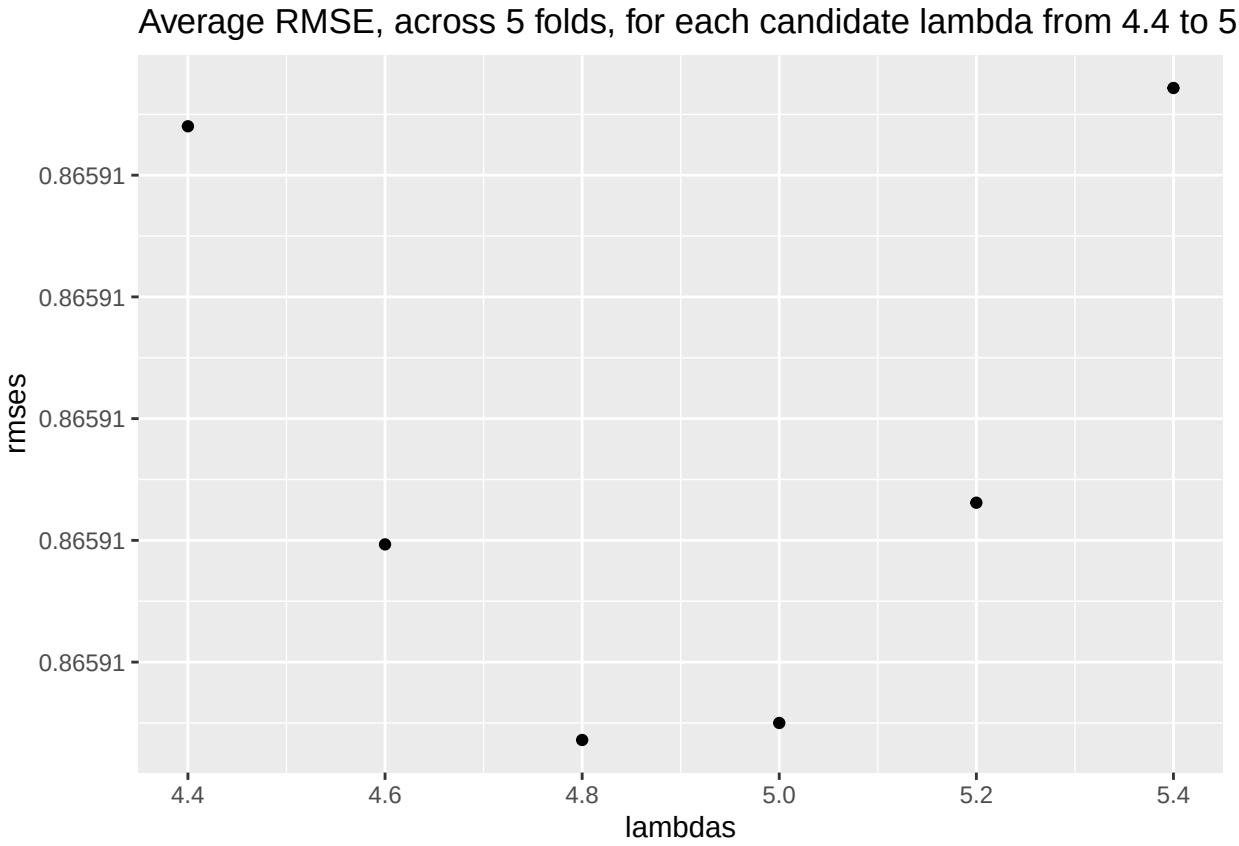
```
#fifth model
  #prediction = regularized version of movie and user effects

  #calculate b_i and b_u using concept of lambda
  #(set at 2.75 for now, will use cv later to pick final)
  lambda <- 2.75
  mu <- mean(edx_train$rating)
  movie_reg2_avgs <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

  user_reg2_avgs <- edx_train %>%
    left_join(movie_reg2_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda), n_u = n())
```

We then set our series of candidate lambdas, and run the first (4.4 in this case) through all five folds, generating the resultant predictions and RMSEs. For simplicity, the calculations for only the first fold (and thus first component RMSE) of $\lambda = 4.4$ are displayed here.

```
#tune updated model for lambda with 5-fold cross-validation
  lambdas <- seq(4.4, 5.4, .2)
  mu_f1 <- mean(mfold_1_train$rating)
  mu_f2 <- mean(mfold_2_train$rating)
  mu_f3 <- mean(mfold_3_train$rating)
```

```
mu_f4 <- mean(mfold_4_train$rating)
mu_f5 <- mean(mfold_5_train$rating)

#use full 5-fold cross-validation here when tuning
rmses <- sapply(lambdas, function(l){

  #first fold
  b_i_set <- mfold_1_train %>%
    group_by(movieId) %>%
    summarize(ss1 = sum(rating-mu_f1),
              n_i =n (),
              b_i = (ss1)/(n_i+l))

  b_u_set <- mfold_1_train %>%
    left_join(b_i_set, by="movieId") %>%
    group_by(userId)%>%
    summarize(ss2 = sum(rating-mu_f1),
              n_u = n(),
              b_u = (ss2 - sum(b_i))/(n_u+l))

  predicted_ratings2 <-
    mfold_1_test %>%
    left_join(b_i_set, by = "movieId") %>%
    left_join(b_u_set, by = "userId") %>%
    mutate(prediction = mu_f1 + b_i + b_u) %>%
    .$prediction

  rmsef1 <- RMSE(predicted_ratings2, mfold_1_test$rating)
```

The resultant graph shows that our new best lambda is 4.8.



Thus, like before, we set lambda to this preferred value, and then re-generate the associated new predictions and RMSE.

```
#select best performing lambda for use
  best_lambda2 <- lambdas[which.min(rmses)]

  #re-run second regularized model using best lambda
  lambda <- best_lambda2
  mu <- mean(edx_train$rating)
  movie_reg2_avgs <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i  = sum(rating - mu)/(n()+lambda))

  user_reg2_avgs <- edx_train %>%
    left_join(movie_reg2_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  #assign each movie its predicted rating from this regularized (movie effects only) model
  predicted_ratings_r2 <- edx_test %>%
    left_join(movie_reg2_avgs, by = "movieId") %>%
    left_join(user_reg2_avgs, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  #calculate the RMSE of this fifth model
  model_5_rmse <- RMSE(predicted_ratings_r2, edx_test$rating)
```

This new RMSE is, again, a bit of an improvement on its un-regularized sibling (two lines above it).

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597
Regularized Movie Effect Model	0.94392
Regularized Movie + User Effect Model	0.86532

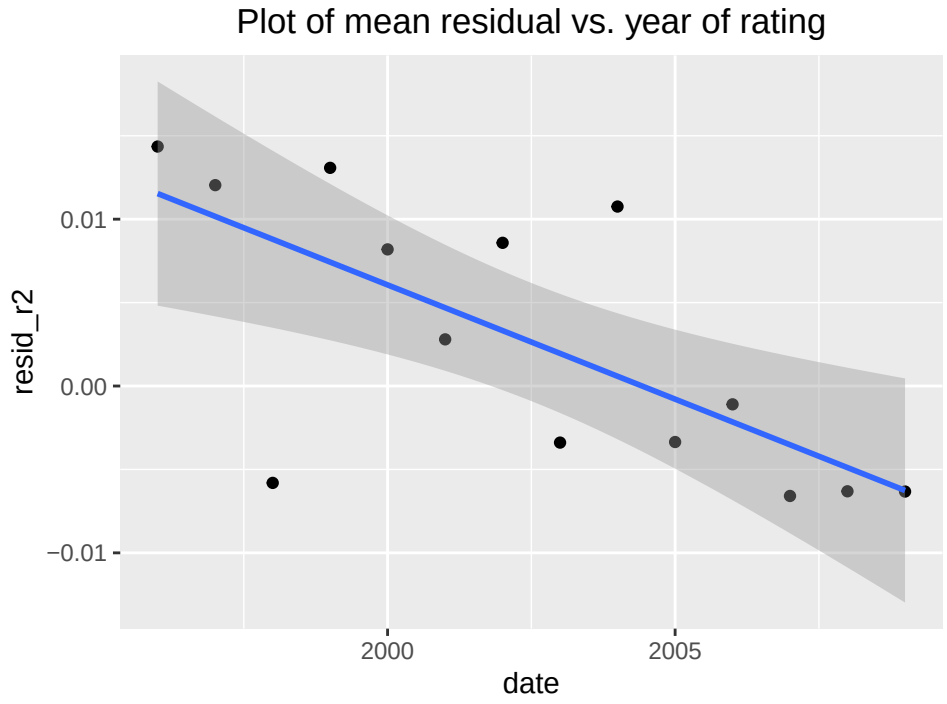
Adding a new factor

We next examine whether we can improve our predictions further by capturing the effect of an additional factor in our data, specifically the date that a rating was given.

As mentioned earlier, the first step here is to examine the relationship between date (by year) of rating and the residuals after our regularized b_i and b_u have been taken into account.

Residual $= Y_{u,i} - (mu + b_i + b_u)$

So, we analyze, then graph:



As this graph shows, the effect appears to be very slight, but present. We decide to incorporate this effect as a simple linear function of the time of rating. We calculate these residuals using our training set, then we fit a linear model to identify coefficients.

```
#build model to capture these residuals
    #calculate *train* set residuals
    predicted_ratings_r2_train <- edx_train %>%
      left_join(movie_reg2_avgs, by = "movieId") %>%
      left_join(user_reg2_avgs, by = "userId") %>%
      mutate(pred_r2_train = mu + b_i + b_u) %>%
      pull(pred_r2_train)
```

```
    edx_train <- mutate(edx_train, resid_r2_train = rating-predicted_ratings_r2_train)

    #for simplicity, regress residuals vs. time on a linear model (as graph suggests)
    fit_time1 <- lm(resid_r2_train ~ timestamp, data = edx_train)
```

```
fit_time1$coef[1]
```

```
## (Intercept)
##    0.043165
```

```
fit_time1$coef[2]
```

```
##   timestamp
## -3.9412e-11
```

Once complete, we incorporate the results of $f(t)$ into our most recent predictions to make a new prediction set. We then calculate the new RMSE.

```
 #incorporate this simple linear function to our regularized  predictions in the test set
  pred_ratings_r2_time <- predicted_ratings_r2 + fit_time1$coef[1] +
                      (fit_time1$coef[2]*edx_test$timestamp)

  #calculate the RMSE of this sixth model
  model_6_rmse <- RMSE(pred_ratings_r2_time, edx_test$rating)
```

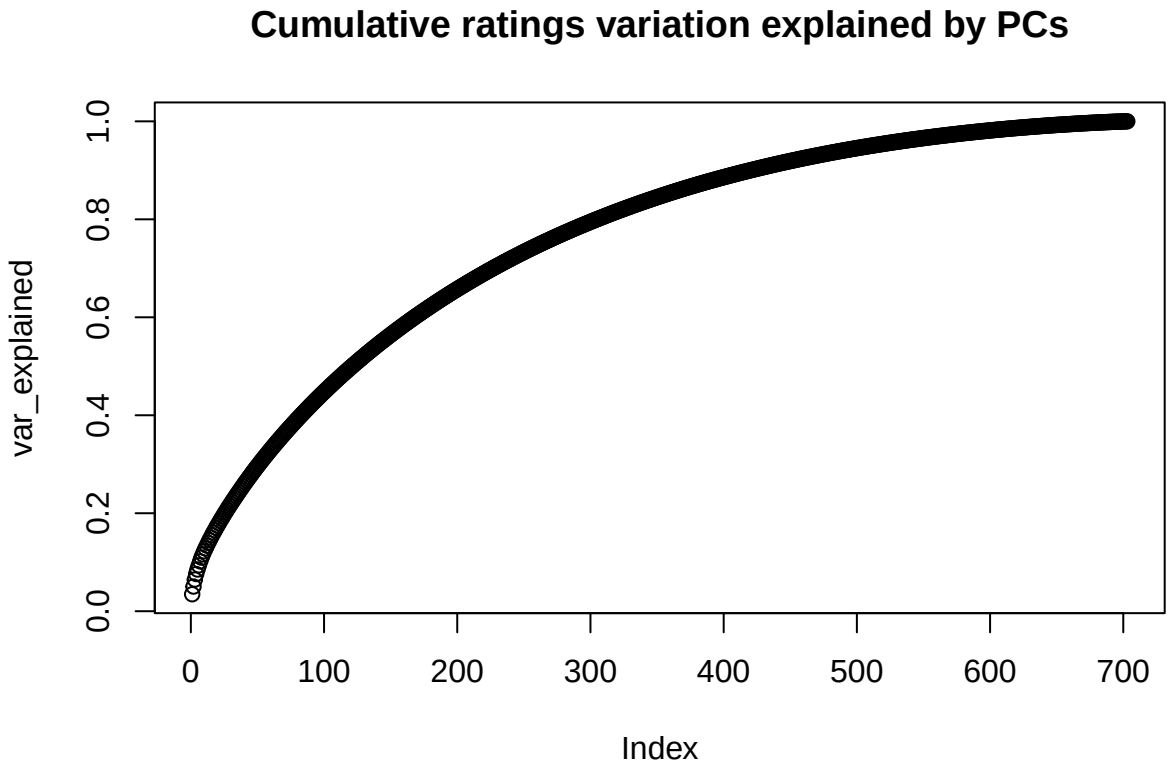
The improvement in RMSE from including this new time factor in our model is small, but it does exist.

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597
Regularized Movie Effect Model	0.94392
Regularized Movie + User Effect Model	0.86532
Reg. M/U Effect + Time Model	0.86529

Matrix factorization

We begin our approach by confirming that matrix factorization models in R can indeed identify underlying "principal components" that contribute to each movie's rating. Recall that these components represent some aspect of a movie (theme, movie star, era, etc.) that appeals to viewers.

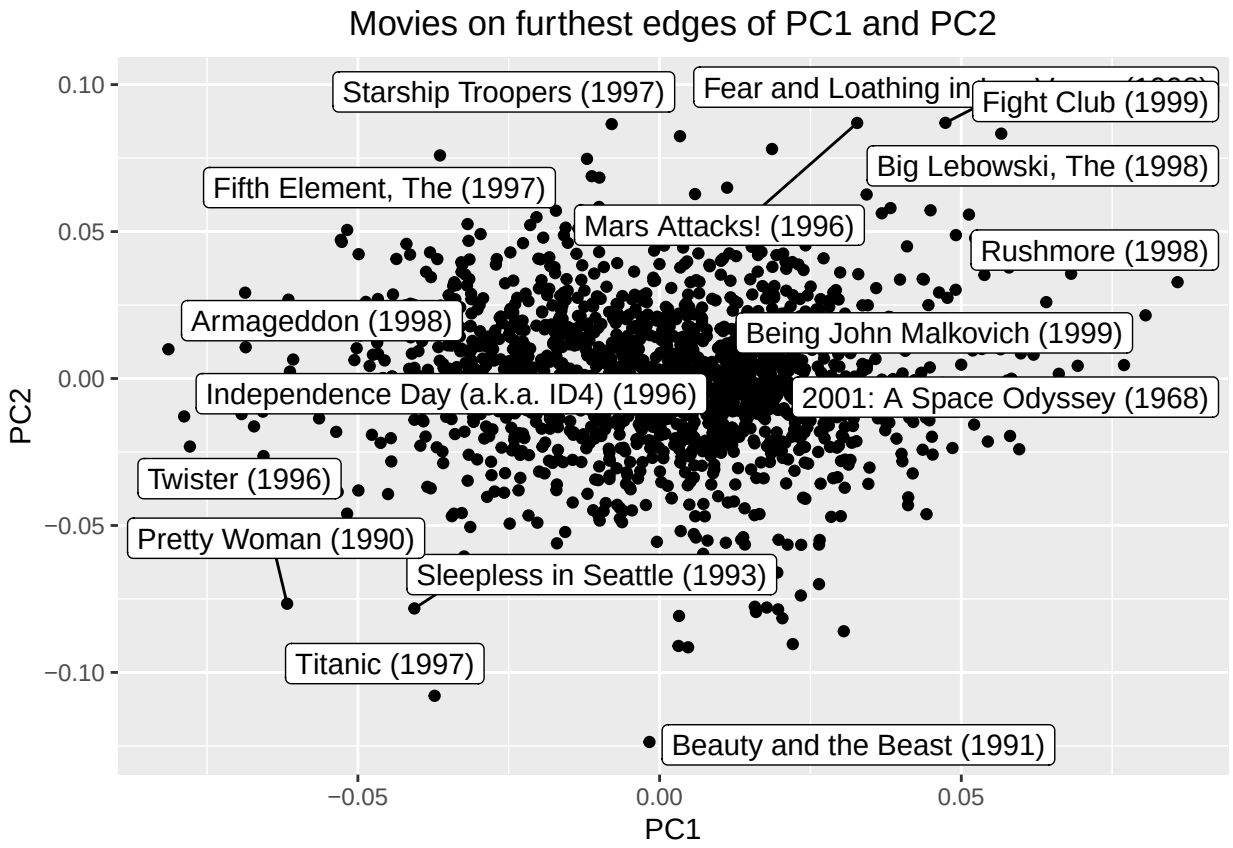
```
#identify principal components
  y[is.na(y)] <- 0
  y <- sweep(y, 1, rowMeans(y))
  pca <- prcomp(y)

  #show that few factors explain lots of variation
  var_explained <- cumsum(pca$sdev^2/sum(pca$sdev^2))
  plot(var_explained, main = "Cumulative ratings variation explained by PCs")
```



As the graph above shows, a very useful percent of rating variation can be explained with a reasonably small number of components.

Next, we attempt to visualize a few of the "principal components" that these powerful models can identify as contributing to underlying ratings.

```
#visualize PCs: movies on one side of a PC vs. the other
  pcs <- data.frame(pca$rotation, name = colnames(y))
  pcs %>%  ggplot(aes(PC1, PC2)) + geom_point() +
    geom_label_repel(aes(PC1, PC2, label=name, max.overlaps = 10),
                     data = filter(pcs,
                                   PC1 < -0.075 |
                                     PC1 > 0.075 |
                                     PC2 < -0.075 |
                                     PC2 > 0.075)) +
    ggtitle("Movies on furthest edges of PC1 and PC2") +
    theme(plot.title = element_text(hjust = 0.5))
```



For example, the matrix above highlights movies with high and low scores for both PC1 and PC2. In it, we can begin to tell that mass appeal movies (left side of PC1 axis) are distinct from niche appeal movies (right side).

Along the same lines, we can look at movies on opposite ends of the 6th principal component, for example, we notice a clear distinction between light comedy (first table) vs. space fantasy movies (second table).

```
#show lists of highest and lowest PC6
  pcs %>% select(PC6) %>% arrange(desc(PC6)) %>% slice(1:6)
```

```
##                                                      PC6
## Austin Powers: International Man of Mystery (1997) 0.12567
## Happy Gilmore (1996)                               0.12481
## Wayne's World (1992)                               0.11598
## Austin Powers: The Spy Who Shagged Me (1999)       0.11043
## Billy Madison (1995)                               0.10798
## Dumb & Dumber (1994)                               0.10768
```

```
  pcs %>% select(PC6) %>% arrange(PC6) %>% slice(1:6)
```

```
##                                                       PC6
## Blade Runner (1982)                                -0.117501
## 2001: A Space Odyssey (1968)                       -0.095319
## Star Wars: Episode I - The Phantom Menace (1999) -0.088969
## Star Trek II: The Wrath of Khan (1982)             -0.086770
## A.I. Artificial Intelligence (2001)                -0.086272
## Apocalypse Now (1979)                              -0.076921
```

Satisfied that these principal components are credible, our next step is to use the `recosystem` package to build a full matrix factorization model and generate resulting predictions.

First, we build and tune the model.

```
#now, build seventh model to capitalize on this approach

  #create object, coerce train + test sets into required format
  recsys_model <- Reco()
  recsys_train <- data_memory(edx_train$userId, edx_train$movieId, rating = edx_train$rating)
  recsys_test <- data_memory(edx_test$userId, edx_test$movieId, rating = edx_test$rating)

  #tune model on train set
  options1 <- recsys_model$tune(recsys_train, opts=list(dim=c(5,10),
                                                        lrate=c(0.1, 0.2),
                                                        costp_l1 = c(0),
                                                        costq_l1 = c(0),
                                                        niter = 10,
                                                        verbose=FALSE))
  #examine optimal settings
  options1$min
```

```
## $dim
## [1] 10
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.2
##
## $loss_fun
## [1] 0.82798
```

Next, we install these preferred settings, run the model to generate predictions, and calculate our RMSE.

```
 #train model with these settings
  recsys_model$train(recsys_train, opts = c(options1$min, nthread = 1, niter = 20))
```

```
#generate predictions
  pred_ratings_mfx1 = recsys_model$predict(recsys_test, out_memory())

  model_7_rmse <- RMSE(pred_ratings_mfx1, edx_test$rating)
```

As we can see, this approach gives us a significantly improved RMSE of just over 0.81.

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597

Method	RMSE
Regularized Movie Effect Model	0.94392
Regularized Movie + User Effect Model	0.86532
Reg. M/U Effect + Time Model	0.86529
Matrix Factorization	0.81016

Ensembles

To build our ensemble models, we start with a data table of all predictions for the `edx_test` set thus far.

```
#assemble data frame of all predictions so far
pred_df <- data.frame(JustAvg = mu_train, Movie = predicted_ratings_bi,
                      MovieUser = predicted_ratings_biu, RegI = predicted_ratings_r1,
                      RegIU = predicted_ratings_r2, RegIU.Time = pred_ratings_r2_time,
                      Mfx1 = pred_ratings_mfx1)
head(pred_df)
```

```
##   JustAvg  Movie MovieUser   RegI  RegIU RegIU.Time   Mfx1
## 1  3.5125 2.8494    4.4497 2.8504 4.0821     4.0922 4.1548
## 2  3.5125 4.2195    3.8698 4.2194 3.9639     3.9729 3.8335
## 3  3.5125 3.2880    2.9383 3.2881 3.0328     3.0417 2.9302
## 4  3.5125 3.0013    2.6516 3.0017 2.7466     2.7556 2.5511
## 5  3.5125 3.3367    3.6538 3.3373 3.5979     3.5963 3.4537
## 6  3.5125 3.4947    4.0885 3.4947 4.0041     4.0140 3.9742
```

The prediction from our first ensemble model will simply be the mean of all existing predictions.

```
#build + test various ensemble approaches
  #ensemble of all models
  predicted_ratings_ensall <- rowMeans(pred_df) #mean of all predictions
  head(predicted_ratings_ensall)
```

```
## [1] 3.7130 3.9416 3.1474 2.8886 3.4983 3.7975
```

```
  #calculate the RMSE of this first ensemble model
  ensall_rmse <- RMSE(predicted_ratings_ensall, edx_test$rating)
```

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597
Regularized Movie Effect Model	0.94392
Regularized Movie + User Effect Model	0.86532
Reg. M/U Effect + Time Model	0.86529
Matrix Factorization	0.81016
Ensemble All	0.86805

We then repeat this process, focusing only on the two top-performing models.

```
 #ensemble of best two models
  #select colums
  cols_selected <- c(6,7) #select columns of current best two models
  predicted_ratings_ens2 <- rowMeans(pred_df[,cols_selected])
  head(predicted_ratings_ens2)
```

```
## [1] 4.1235 3.9032 2.9859 2.6533 3.5250 3.9941
```

```
  #calculate the RMSE of this second ensemble model
  ens2_rmse <- RMSE(predicted_ratings_ens2, edx_test$rating)
```

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597
Regularized Movie Effect Model	0.94392
Regularized Movie + User Effect Model	0.86532
Reg. M/U Effect + Time Model	0.86529
Matrix Factorization	0.81016
Ensemble All	0.86805
Ensemble Top Two	0.82308

As we can see, neither ensemble model out-performed the matrix factorization model.

Rehearsing

As shown above, our best-performing model is indeed our matrix factorization model. So, we apply it to our `rd` set (to which which it has never been exposed), both to make sure that everything runs as planned, and to verify that the resultant RMSE will be close to that achieved using the `edx_test` set.

```
#matrix factorization model
  #do not train further, do not change any parameters

  #generate required dataset
  recsys_test_rehearsal <- data_memory(rd$userId, rd$movieId, rating = rd$rating)

  #calculate predictions for rehearsal set
  pred_ratings_mfx1_rehearsal = recsys_model$predict(recsys_test_rehearsal, out_memory())

  #calculate RMSE for rehearsal set
  model_7_rehearsal_rmse <- RMSE(pred_ratings_mfx1_rehearsal, rd$rating)
```

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597
Regularized Movie Effect Model	0.94392
Regularized Movie + User Effect Model	0.86532
Reg. M/U Effect + Time Model	0.86529
Matrix Factorization	0.81016
Ensemble All	0.86805
Ensemble Top Two	0.82308
Matrix Fac. on Rehearsal	0.81035

Validation set application

As a last step, we use our best model to predict ratings for our validation set itself. Predictions for the first several rows are displayed for convenience.

```
#matrix factorization model
  #do not train further, do not change any parameters

  #generate required dataset
  recsys_test_validation <- data_memory(validation$userId, validation$movieId,
                                        rating = validation$rating)

  #calculate predictions for validation set
  pred_ratings_mfx1_validation = recsys_model$predict(recsys_test_validation, out_memory())

  #look at first several entries
  head(pred_ratings_mfx1_validation)
```

```
## [1] 4.4086 5.2718 4.6287 3.2217 4.0470 2.6820
```

```
  #if desired, full list of predictions is here (commented out for now)
  #pred_ratings_mfx1_validation

  #calculate RMSE for validation set
  model_7_validation_rmse <- RMSE(pred_ratings_mfx1_validation, validation$rating)
```

Method	RMSE
Simple Average	1.06035
Movie Effect Model	0.94402
Movie + User Effects Model	0.86597
Regularized Movie Effect Model	0.94392
Regularized Movie + User Effect Model	0.86532
Reg. M/U Effect + Time Model	0.86529
Matrix Factorization	0.81016
Ensemble All	0.86805
Ensemble Top Two	0.82308
Matrix Fac. on Rehearsal	0.81035
Matrix Fac. on Validation	0.81036

We see, happily, that it performs nearly as well with this independent validation set as it did on our training and rd sets. We see an RMSE of 0.8104.

(Conclusion begins on next page)

Conclusion

Overall, we find that the matrix factorization approach produces the lowest RMSE among the models we examined, and that is gives a final RMSE of 0.8104 when tested on our validation set.

Method	RMSE
Matrix Fac. on Validation	0.81036

This result speaks to the power of machine learning to recognize subtle factors embedded in ratings, factors that we (as viewers) might not even be conscious of ourselves!

Put another way, just because a machine learning algorithm such as matrix factorization can't put a name to a particular factor (preference for slapstick comedy elements, for ex.), that doesn't mean the model can't recognize its existence and apply it to a successful prediction.

Limitations:

One of the limitations of this study was the inability to use linear regression when generating a fundamentally linear model. This was due to the combination of very large data size and the limited computing power on a standard consumer laptop.

Another limitation of this work was that the "result" data (i.e. the rating) we examined was treated only as continuous rather than as categorical in any way.

Future Work

Thus, regarding future work, it would be interesting to treat the ratings as categorical outcomes (there were 10 possible ratings, after all), and to examine other models that do well in predicting such categorical outcomes, such as random forests or k-nearest neighbors.