HarvardX MovieLens Capstone

matt_173

2022-06-15

Introduction

This report documents analysis of a movie review database called movielens. The goal of the analysis is to produce a movie rating prediction model. The quality metric chosen for the prediction model is root mean squared error (RMSE) as calculated against a set of ratings that were not available to the model development process.

The movielens dataset consists of individual user ratings of the following form:

- userId: unique numeric id for user in the database
- movieId: unique numeric id for movie in the database
- rating: discreet movie rating ranging from 0.5 to 5 in increments of 0.5
- timestamp: system representation of datetime that review was submitted
- title: movie title with release year appended. ex:"Boomerang (1992)"
- genres: pipe-delimited set of genre classifiers for the movie. ex:"Action|Crime|Thriller"

The path of analysis starts from and extends the regularized bias effect framework developed in the course materials. The analysis will further explore:

1. Time-dependence effects
2. Additional bias effects
3. Principal component analysis on residuals

Analysis

Data Sourcing and Partitioning

The analysis begins with sourcing the movie review data from the grouplens website and then partitioning the data so that 10% is reserved for model validation. This validation set will not be used at all in the development of the prediction model. This allows us to gain an "out of sample" measure of prediction quality from our model. The resulting data sets are as follows:

- edx: dataframe with 90% of the total reviews, further partitioned into train_set and test_set as:
  - train_set: 80% of edx reviews
  - test_set: 20% of edx reviews, further filtered to ensure movie/userIds are in train_set
- validation: dataframe with 10% of total reviews. Filtered to contain movie/userIds found in edx. Removed reviews are added back to edx.

**NOTE: Successful knitr execution required pre-processing and saving code chunks. To run the full .RMD, toggle all the eval=F/T tags in document.

```r
#define all libraries here for transparency
quiet_load <- suppressPackageStartupMessages
quiet_load(library(tidyverse))
quiet_load(library(caret))
quiet_load(library(data.table))
quiet_load(library(caret))
quiet_load(library(ggrepel))
quiet_load(library(lubridate))
quiet_load(library(gam))
quiet_load(library(irlba)) #fast truncated SVD and PCA for large dense and sparse matrices
```

```r
dl <- tempfile()
download.file("https://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") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

Data Validation

Note that the data sets are pre-configured in the project setup and we are not meant to modify the existing data in the validation set used for model testing. However we can do some inspection on the edx/validation sets to see if they have data integrity relative to the description above. Since genres and title just map to a movieId we don't explicitly check here.

userIds

```r
length(unique(edx$userId))
```

```
## [1] 69878
```

```r
length(unique(validation$userId))
```

```
## [1] 68534
```

```r
mean(unique(validation$userId) %in% unique(edx$userId))
```

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

movieIds

```r
length(unique(edx$movieId))
```

```
## [1] 10677
```

```r
length(unique(validation$movieId))
```

```
## [1] 9809
```

```r
mean(unique(validation$movieId) %in% unique(edx$movieId))
```

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

ratings

```r
levels(factor(unique(edx$rating)))
```

```
##  [1] "0.5" "1"   "1.5" "2"   "2.5" "3"   "3.5" "4"   "4.5" "5"
```

```r
levels(factor(unique(validation$rating)))
```

```
##  [1] "0.5" "1"   "1.5" "2"   "2.5" "3"   "3.5" "4"   "4.5" "5"
```

timestamps: Note that while the review timestamps looks as expected, we see something unusual with the relationship of review timestamp to movie release year. In some cases the review appears in a year prior to the year associated with the movie title. The number is small and further inspection indicates that the review may typically be the year before the movie year so we disregard in this analysis.

```r
min(as_datetime(edx$timestamp))
```

```
## [1] "1995-01-09 11:46:49 UTC"
```

```r
max(as_datetime(edx$timestamp))
```

```
## [1] "2009-01-05 05:02:16 UTC"
```

```r
min(as_datetime(validation$timestamp))
```

```
## [1] "1995-01-09 11:46:49 UTC"
```

```r
max(as_datetime(validation$timestamp))
```

```
## [1] "2009-01-05 04:50:28 UTC"
```

```r
#check if there are reviews with timestampp year prior to movie year
edx %>% mutate(mov_yr = as.numeric(str_match(title, "\\((\\d{4})\\)$")[,2]),
       rev_yr = as.numeric(year(as_datetime(timestamp)))) %>%
  filter(rev_yr < mov_yr) %>% summarize(n=n()) %>% pull(n)
```

```
## [1] 175
```

```
edx %>% mutate(mov_yr = as.numeric(str_match(title, "\\((\\d{4})\\)$")[,2]),
        rev_yr = as.numeric(year(as_datetime(timestamp)))) %>%
  filter(rev_yr == (mov_yr-1)) %>% summarize(n=n()) %>% pull(n)
```

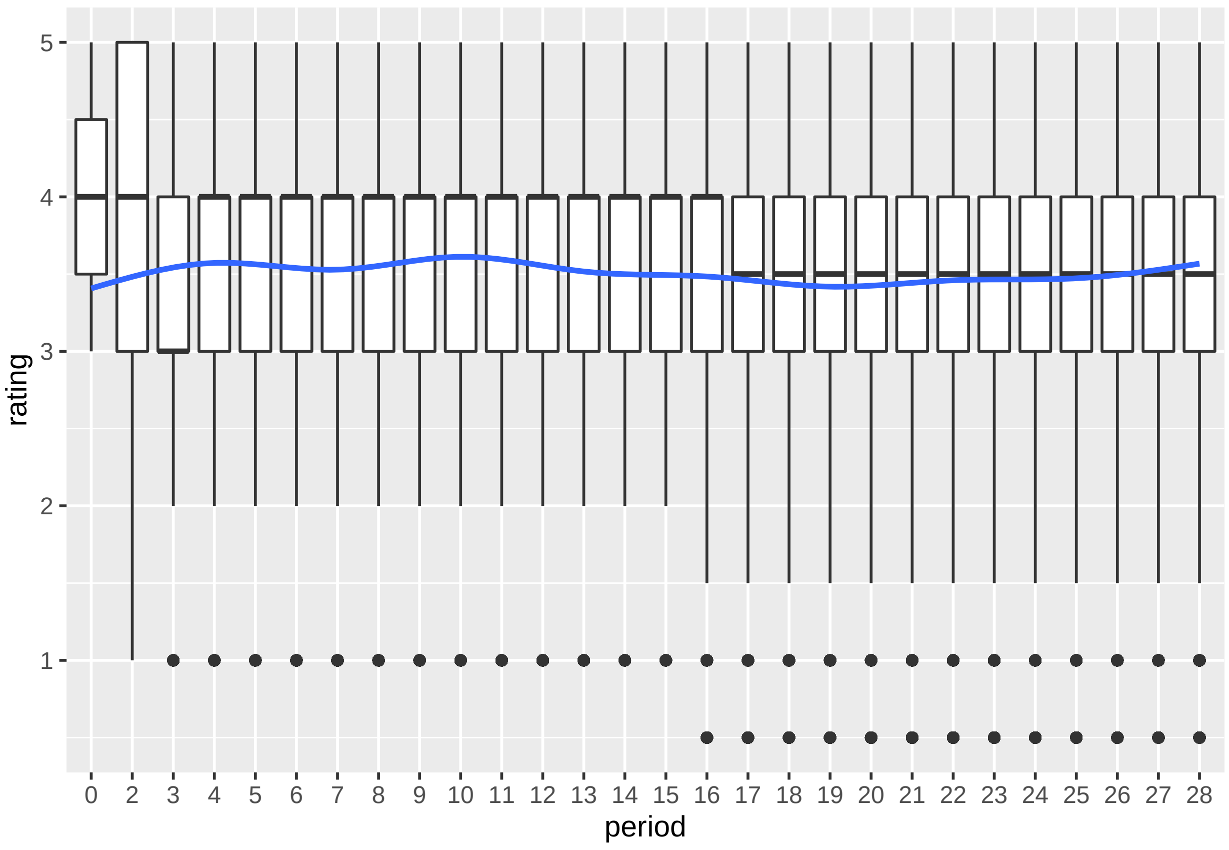
```
## [1] 172
```

Data Exploration: Time Effects

Similar to other regression approaches there is an implicit assumption in the user/movie effects model that the means are stable over time. We'll take a look at three plausible sources of time variation in this section to determine if any are candidates for improving the rating predictions.

Overall Average Rating It is possible that the overall average rating changes over time in a systematic manner. To explore this we'll calculate an average rating for each six month period and plot it. Note that the mean rating doesn't have much time variation.

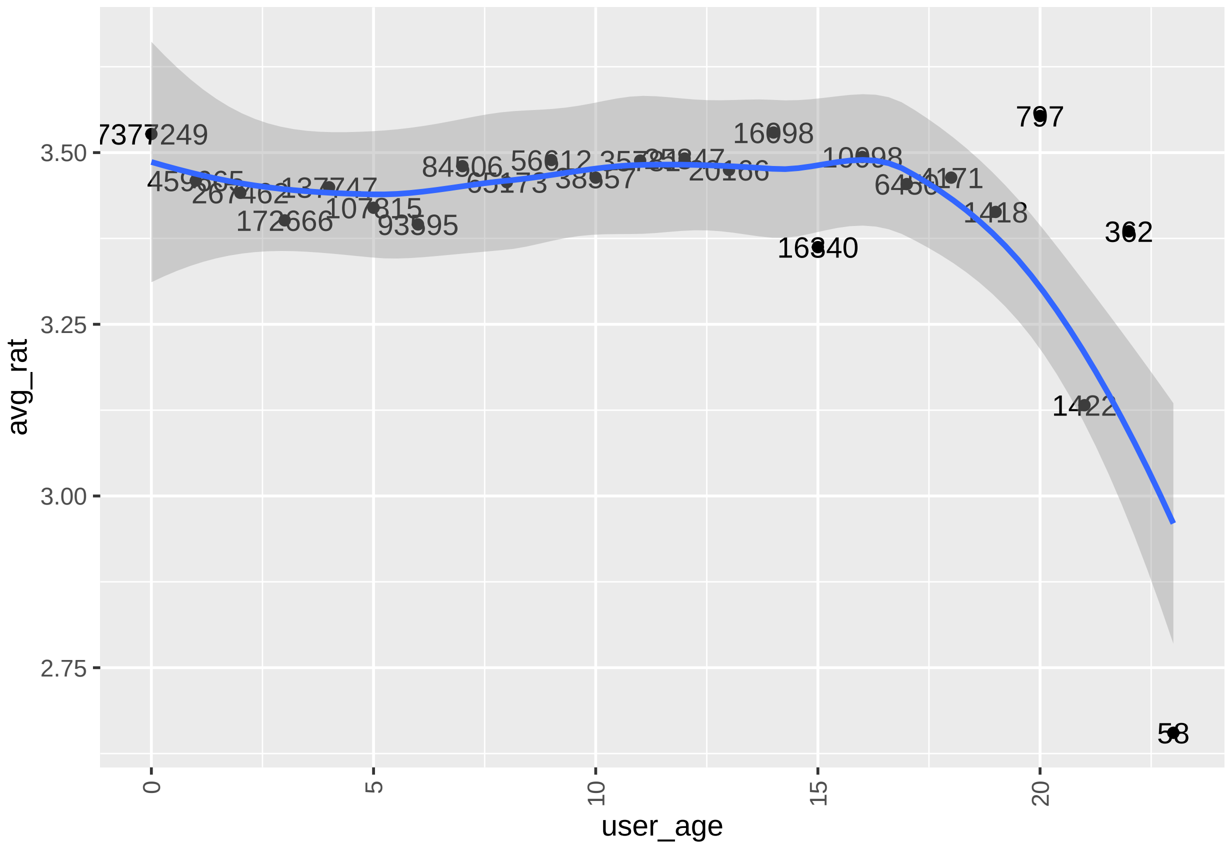
```
base_date = min(edx$timestamp)
#note 86,400 seconds in a day going to "bin" by ~half years
edx %>%
  mutate(period = factor(round((timestamp - base_date)/(182*86400),0))) %>%
  group_by(period) %>%
  ggplot(aes(period, rating)) + geom_boxplot() +
  geom_smooth(aes(x=as.numeric(period)),method = "gam",formula=y ~ s(x, bs = "cs"))
```



User-specific Trend

Another possibility is that each user displays a time-varying trend in their rating pattern. For instance, perhaps enthusiastic new users rate movies higher and then ratings relax to a lower level over time. To study this we, assign a user age to each rating measuring the time since the user's first rating in the data set. Again we group by this measure and calculate a mean rating to plot against user age. Note that there is little variation of average rating with user age until very long tenures, perhaps over 17 years. There are relatively few ratings from these users so modeling time dependence is unlikely to have a material impact on prediction quality.

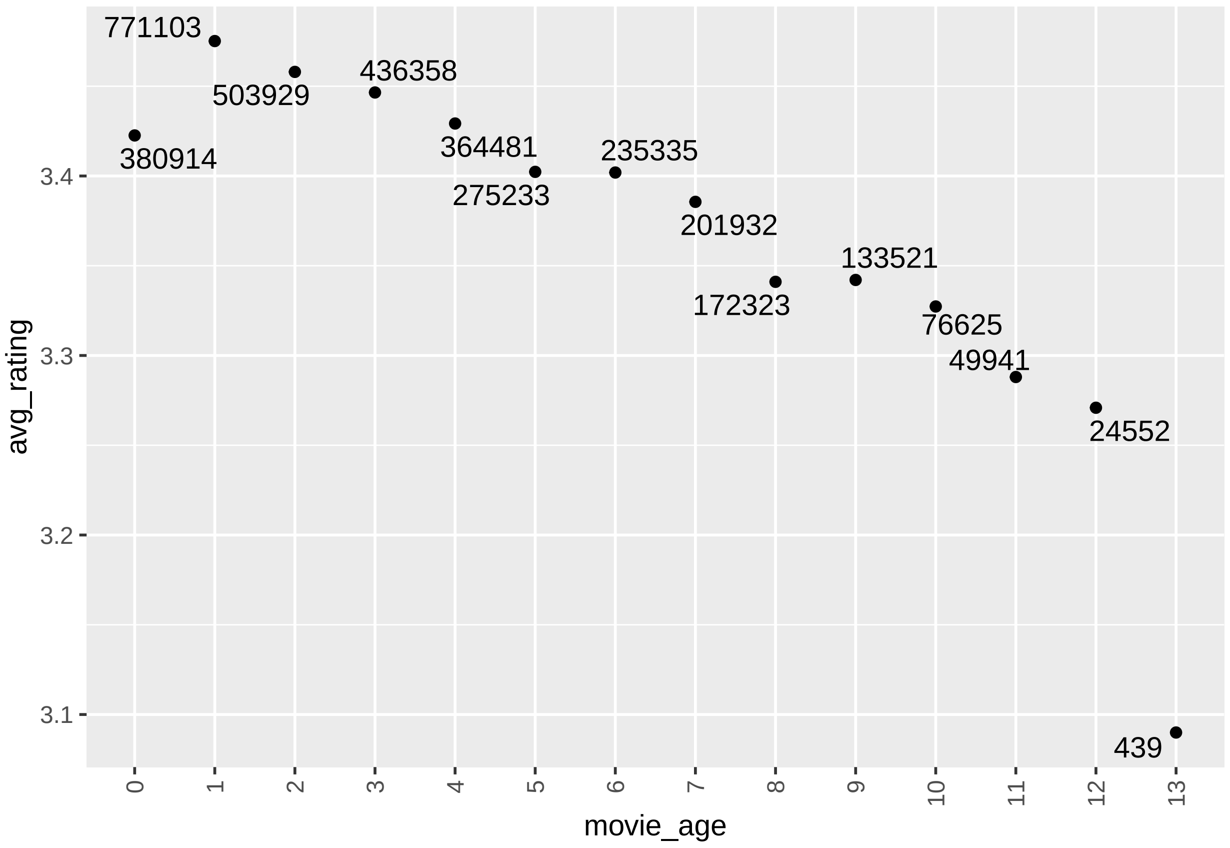
```
user_t <- edx %>% group_by(userId) %>%
  mutate(minstamp = min(timestamp),
         user_age = round((timestamp - minstamp)/(182*86400),0)) %>%
  ungroup()  %>%  group_by(user_age) %>%
  summarize(avg_rat = mean(rating), n = n())

user_t_plot <- user_t %>% ggplot(aes(user_age,avg_rat)) + geom_point() +
  geom_text(aes(label=n)) +
  geom_smooth(method="loess", formula = 'y~x') +
  theme(axis.text.x = element_text(angle=90, vjust=.5, hjust=1))

user_t_plot
```



Movie-specific Trend

Movies might experience different rating trends after their initial release. For example press coverage could shape reviews at that time while later reviews would be less impacted by this coverage. To check for this trend, we first limit the analysis to movies released after ratings started in the database. Although we observe an apparent trend in decreasing rating with time, replacing the static mean with a time-dependent mean in the regularized model did not have a material impact on prediction RMSE so it was not used in the model.

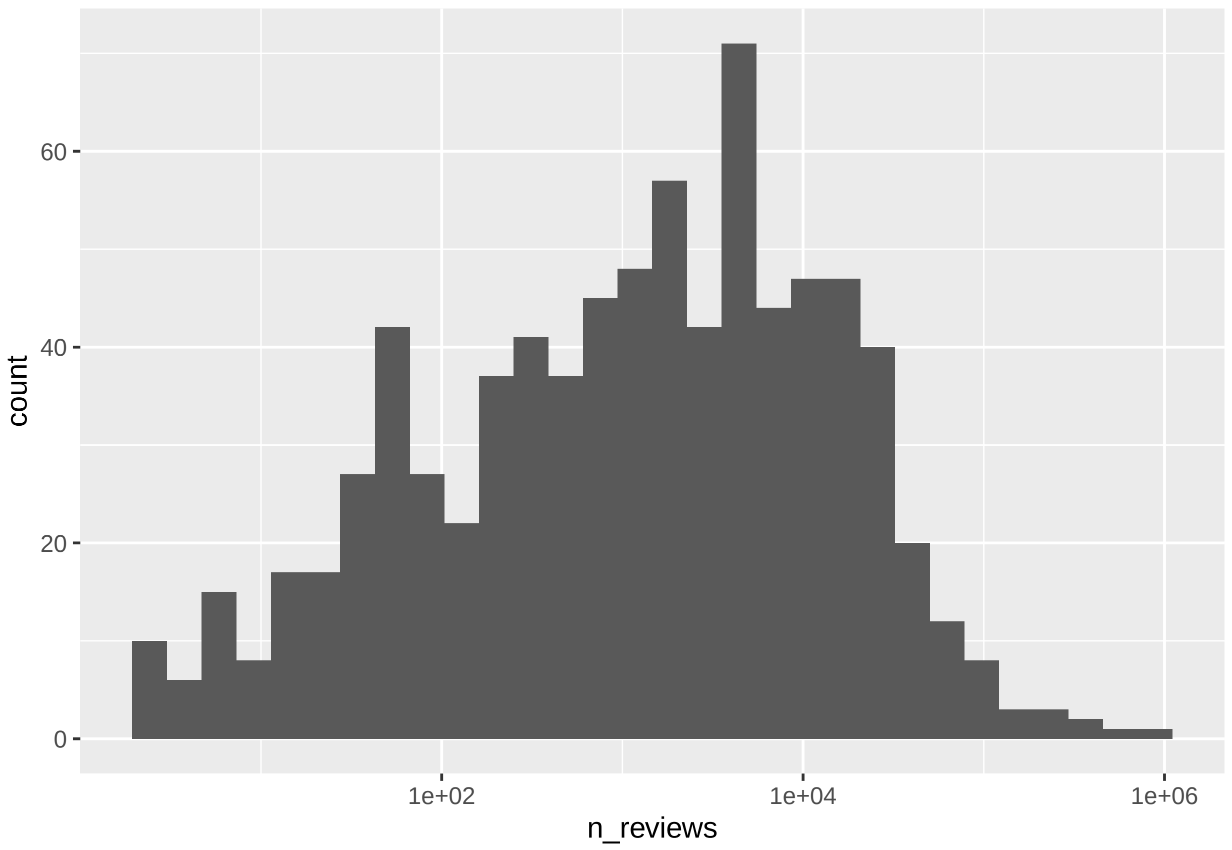
```
mu_t <- edx %>% mutate(rev_yr = year(as_datetime(timestamp)),
                 mov_yr = as.numeric(str_match(edx$title, "\\((\\d{4})\\)$")[,2]),
                 movie_age = factor(rev_yr - mov_yr)) %>%
  filter(mov_yr >= 1996 & (rev_yr - mov_yr) >= 0) %>% group_by(movie_age) %>%
  summarize(avg_rating = mean(rating), n = n())

mu_t %>% ggplot(aes(movie_age,avg_rating)) +
  geom_point() + geom_text_repel(aes(label=n)) +
  theme(axis.text.x = element_text(angle=90, vjust=.5, hjust=1))
```



Data Exploration: Genre Effect

Although it seems clear that there should a strong genre rating variation by user, that approach would be fairly complex and we would need many reviews for each user to properly model this effect. However, we can examine broad genre effects by considering each genre combination as a category and considering the average genre effect. First we look at a histogram of movies by genre combination.

```
edx %>% group_by(genres) %>% summarize(n_reviews = n()) %>% ggplot(aes(n_reviews)) +
  geom_histogram(bins = 30) + scale_x_log10()
```

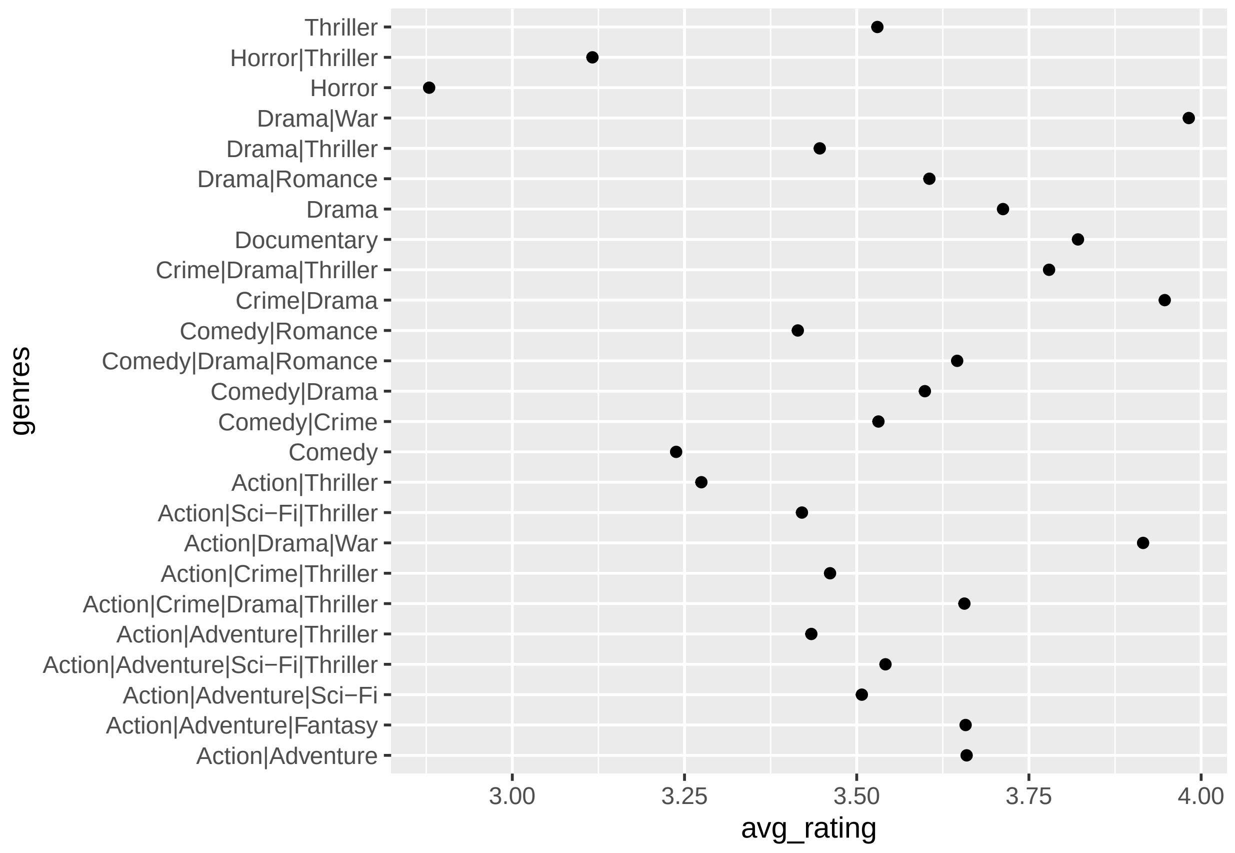


```
edx %>% group_by(genres) %>% summarize(n=n()) %>% arrange(desc(n)) %>% head()
```

```
## # A tibble: 6 x 2
##   genres                      n
##   <chr>                   <int>
## 1 Drama                  733296
## 2 Comedy                 700889
## 3 Comedy|Romance         365468
## 4 Comedy|Drama           323637
## 5 Comedy|Drama|Romance 261425
## 6 Drama|Romance          259355
```

Most important is the amount of rating variability by genres category. We limit to the most populous 25 categories here and see that the variability is quite high so we will include in our regularize effects model.

```
edx %>% group_by(genres) %>% summarize(n=n(), avg_rating=mean(rating)) %>%
  slice_max(order_by = n, n=25) %>% ggplot(aes(avg_rating,genres)) + geom_point()
```



Modeling Approach

The predictive model will extend the regularized effects model from the coursework:

1. Addition of a genre effect
2. Addition of top principal components over the residuals from above model

$$Y = \mu + b_g + b_i + b_u + \sum_{i=1}^{n} U_i d_{i,i} V_i^T$$

Split edx into train/test sets

Model optimization steps should follow a train/test approach to avoid overfitting. We reserve 20% of the edx data for a test set here.

```
set.seed(11, sample.kind="Rounding") #for reproducibility
```

```
## Warning in set.seed(11, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index = createDataPartition(edx$rating, times=1, p=0.2, list=FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Define RMSE function

This function will calculate the root mean squared error for a set of predictions against the reference values specified.

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

K-fold cross validation for lambda

We run a 5-fold cross validation on the training set to determine the optimal lambda value for regularization. This code creates the 5 non-overlapping sets from train_set and validates they are the same size with no overlap.

```
set.seed(7, sample.kind="Rounding")
```

```
## Warning in set.seed(7, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
train_folds <- createFolds(y = train_set$rating, k=5, list = TRUE, returnTrain = FALSE)
sum(sapply(train_folds,function(x){length(x)}))
```

```
## [1] 7200043
```

```
length(train_set$rating) #lengths match
```

```
## [1] 7200043
```

```
#check if folds overlap
sum(train_folds[[1]] %in% train_folds[[2]])
```

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

The cross-validation will run five independent passes, holding out one of the folds for validation in each pass. In each pass, RMSE results are calculated for a range of lambdas. When completed, the RMSE values are averaged for each lambda and the lambda with lowest RMSE is selected for the model.

```
lambdas <- seq(3, 7, 0.25)
#define a dataframe where cols are the lambas and rows are the fold iterations
fold_rmses <- data.frame(matrix(data=0,nrow = 0, ncol = length(lambdas)))
#the zero creates numeric rather than logical
colnames(fold_rmses) <- lambdas
#we'll rbind vector of rmses ordered by lambda to this dataframe

for(i in 1:5){ #sets up the iteration through folds
  #build the loop train/test sets
  l_train_set <- train_set[-train_folds[[i]]]
  temp <- train_set[train_folds[[i]]]
  l_test_set <- temp %>%
    semi_join(l_train_set, by = "genres") %>%
    semi_join(l_train_set, by = "movieId") %>%
    semi_join(l_train_set, by = "userId")
  # Add rows removed from test set back into train set
  removed <- anti_join(temp, l_test_set)
  l_train_set <- rbind(l_train_set, removed)
  rmses <- sapply(lambdas, function(l){
    mu <- mean(l_train_set$rating)

    b_g <- l_train_set %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - mu)/(n()+l))

    b_i <- l_train_set %>%
      left_join(b_g, by = "genres") %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - mu - b_g)/(n()+l))

    b_u <- l_train_set %>%
      left_join(b_g, by = "genres") %>%
      left_join(b_i, by = "movieId") %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - mu - b_g - b_i)/(n()+l))

    predicted_ratings <-
      l_test_set %>%
      left_join(b_g, by = "genres") %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_g + b_i + b_u) %>%
      .$pred
    return(RMSE(predicted_ratings, l_test_set$rating))
  }) #sapply
   #record the vector of rmses for lambda values in this fold
  fold_rmses <- rbind(fold_rmses,rmses)
} #for i=1:5
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
#reassign column namnes
colnames(fold_rmses) <- lambdas
#next average the RMSEs by lambda
cv_rmses <- colMeans(fold_rmses)
```

We can plot the RMSE achieved across the range of lambda values and see that lambda = 5 achieves the lowest RMSE value in the CV. We will use this value of lambda to regularize parameters across the entire edx data set for our final model.

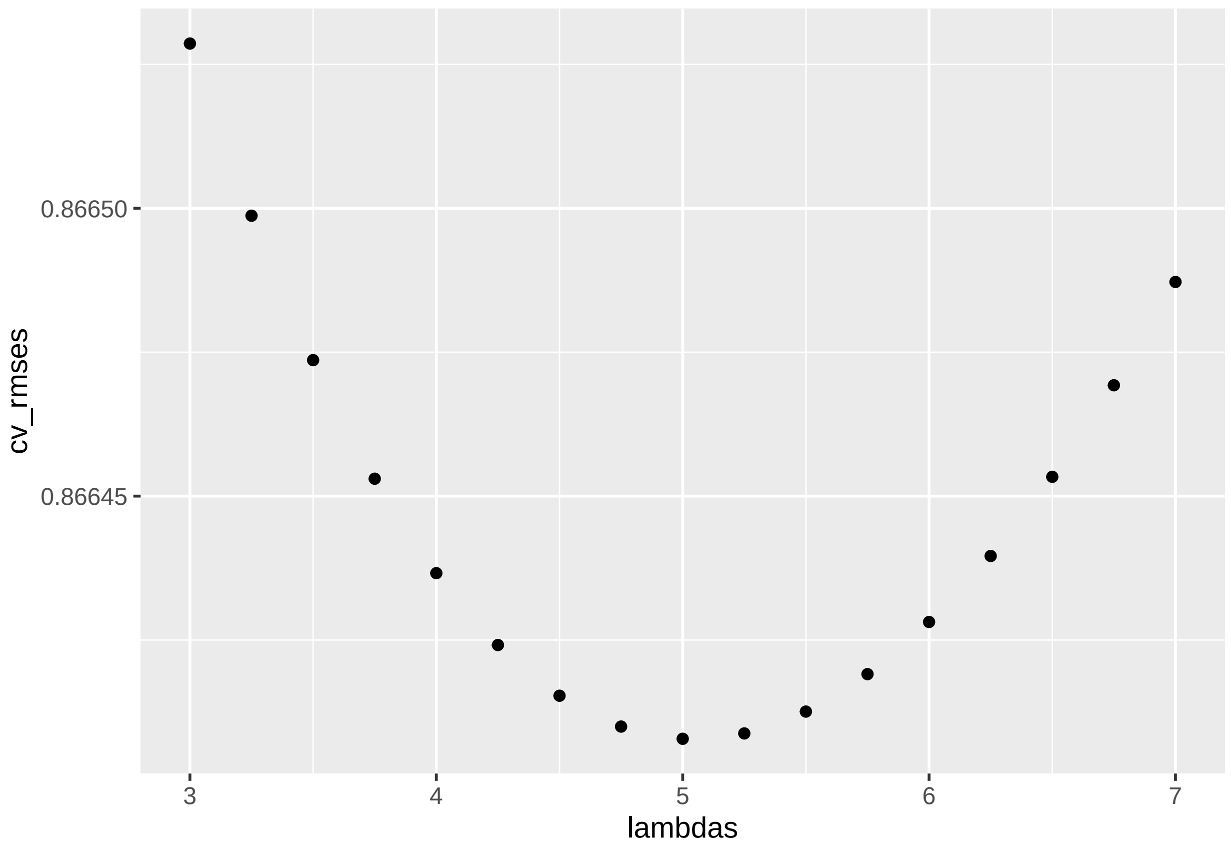
```
cv_lambda <- lambdas[which.min(cv_rmses)]
cv_lambda #5
```

```
## [1] 5
```

```
min(cv_rmses)
```

```
## [1] 0.8664078
```

```
qplot(lambdas,cv_rmses)
```



Final estimation of model effects

With lambda selected through cross-validation, we use the full edx set to estimate the effects/means for the final model.

```
mu <- mean(edx$rating)

b_g <- edx %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu)/(n()+cv_lambda))

b_i <- edx %>%
  left_join(b_g, by = "genres") %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu - b_g)/(n()+cv_lambda))

b_u <- edx %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_i, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_g - b_i)/(n()+cv_lambda))
```

Principal Component Analysis of Residuals

There may be additional systematic variation of ratings still present in the residual errors from this regularized model. Principal component analysis can uncover these latent factors by mapping the variation into a new variable space defined by the principal components of the data. The calculations order the PCs by total variance explained which allows to achieve a large share of the potential explanation with relatively few PCs.

To avoid over-fitting, we'll calculate the principal components on the training data set and then check the prediction quality on the test set.

```
train_err <- train_set %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_g + b_i + b_u, err = rating-pred)

y_train <- train_err %>%
  select(userId, movieId, err) %>%
  spread(movieId, err) %>%
  as.matrix()
rm(train_err, train_set)

rownames(y_train)<- y_train[,1]
y_train <- y_train[,-1]
y_train <- sweep(y_train, 1, rowMeans(y_train, na.rm=TRUE))
y_train <- sweep(y_train, 2, colMeans(y_train, na.rm=TRUE))
y_train[is.na(y_train)] <- 0
pca_train <- prcomp_irlba(y_train, n = 10, retx = TRUE)
```

We can explore the PCA results and prepare to utilize them in our prediction model. The choice of 10 principal components seems appropriate as we see diminishing proportion of variance explained and we become more likely to overfit as we include additional PCs.

```
rownames(pca_train$x) <- rownames(y_train)
rownames(pca_train$rotation) <- colnames(y_train)
summary(pca_train)
```

```
## Importance of components:
##                            PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     0.89442 0.72210 0.68154 0.61120 0.58307 0.56130 0.54598
## Proportion of Variance 0.01064 0.00693 0.00618 0.00497 0.00452 0.00419 0.00396
## Cumulative Proportion  0.01064 0.01757 0.02374 0.02871 0.03323 0.03742 0.04138
##                            PC8     PC9    PC10
## Standard deviation     0.51172 0.50645 0.49214
## Proportion of Variance 0.00348 0.00341 0.00322
## Cumulative Proportion  0.04486 0.04827 0.05150
```

Next we can define a full matrix of corrections based on the PCA for use in the prediction model. Recall that we defined the error as true rating - model prediction. So we would want to "add back" portions of the error to the prediction to get closer to the actual ratings.

```
y_train_hat <- pca_train$x  %*% t(pca_train$rotation)
pca_train_df <- data.frame(userId = rownames(y_train_hat), y_train_hat)
colnames(pca_train_df) <- c("userId",colnames(y_train_hat))
pca_train_df <- gather(pca_train_df, key = "movieId", value = "pca_rat_adj", -userId)
pca_train_df <- pca_train_df %>% mutate(userId = as.numeric(userId),
               movieId = as.numeric(movieId))
```

With the full set of user/movie error correction terms we can now test the contribution of PCA to the rating prediction model for test_set. The reduction in RMSE relative to the regularized effects model indicates that PCA is additive to the prediction model so we will include it.

```
pred_rating_pca_test <- test_set %>%
  left_join(b_g, by = "genres") %>%
```

```r
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(pca_train_df, by = c("userId","movieId")) %>%
  mutate(pred = mu + b_g + b_i + b_u + pca_rat_adj)
```

```r
RMSE(pred_rating_pca_test$pred,test_set$rating)
```

```
## [1] 0.8392949
```

Last, we run the PCA analysis for the entire edx set and generate the error correction elements for the final prediction model.

```r
edx_err <- edx %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_g + b_i + b_u, err = rating-pred)

y_edx <- edx_err %>%
  select(userId, movieId, err) %>%
  spread(movieId, err) %>%
  as.matrix()
rm(edx_err, edx)

rownames(y_edx)<- y_edx[,1]
y_edx <- y_edx[,-1]
y_edx <- sweep(y_edx, 1, rowMeans(y_edx, na.rm=TRUE))
y_edx <- sweep(y_edx, 2, colMeans(y_edx, na.rm=TRUE))
y_edx[is.na(y_edx)] <- 0
pca_edx <- prcomp_irlba(y_edx, n = 10, retx = TRUE)
rownames(pca_edx$x) <- rownames(y_edx)
rownames(pca_edx$rotation) <- colnames(y_edx)
rm(y_edx)
y_edx_hat <- pca_edx$x  %*% t(pca_edx$rotation)
pca_edx_df <- data.frame(userId = rownames(y_edx_hat), y_edx_hat)
colnames(pca_edx_df) <- c("userId",colnames(y_edx_hat))
rm(y_edx_hat, pca_edx)
pca_edx_df <- gather(pca_edx_df, key = "movieId", value = "pca_rat_adj", -userId)
pca_edx_df <- pca_edx_df %>%
        mutate(userId = as.numeric(userId), movieId = as.numeric(movieId))
```

Results

With the final effects and pca models completed, we are now ready to calculate the predictions and associated RMSE for the validation set of movie reviews. **We achieve a prediction RMSE of 0.841 with this final prediction model.**

```r
pred_rating_pca_v <- validation %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(pca_edx_df, by = c("userId","movieId")) %>%
  mutate(pred = mu + b_g + b_i + b_u + pca_rat_adj) %>% .$pred
```

```r
pred_rating_pca_v <- ifelse(pred_rating_pca_v > 5 , 5, pred_rating_pca_v)
pred_rating_pca_v <- ifelse(pred_rating_pca_v < 0.5 , 0.5, pred_rating_pca_v)
```

```r
RMSE(pred_rating_pca_v, validation$rating)
```

```
## [1] 0.8409254
```

Conclusion

This report outlined the development of a user-specific movie rating prediction model based on a historical database of ratings. The analysis extends the regularized effects framework from the course which includes a mean, user effect and movie effect. We saw that time-dependency effects were not material improvements to RMSE, but a genre-specific effect gave a meaningful improvement. We further applied principal components analysis to the model prediction residuals to further improve prediction RMSE in the final model.

This is a topic that has received a great deal of attention and many further enhancements are clearly possible. Future work might include a k-nearest neighbor enhancement for user's with a sufficient number of ratings. The genre factor could be further decomposed from the aggregate used here to individual genre factors to evaluate by user or across the entire population.