Movie Prediction Capstone Project

Steffen Parratt

2/18/2019

Introduction

This report describes the approach and results for the MovieLens Capstone project for the HarvardX course PH125.9x. My project submission includes three files, all with the same file name, but in the following three formats: PDF, RMD and R. The computer code in the R file is identical to the RMD file, except some of the experimental code found at the end of the R file is not executed in the RMD file because it may take hours to run and may not complete execution because of computer memory limitations.

There are four major sections to this report:

- Introduction
- Methods & Analysis
- Results
- Conclusions

Goal

The goal of this project is to create a movie recommendation system using the MovieLens data set. In addition to producing a thoughtful and readable report, a metric for success is producing a system with a reported root mean square error (RMSE) of less than or equal to 0.87750 for the validation data set provided.

As this project is a warm up exercise for creating a data science project of our own choice, we have been encouraged by our teaching assistants to explore the variety of different data science techniques and algorithms that we have encountered in the HarvardX series of courses, particularly those that may be useful in our individual project.

Key Steps

This section provides a high-level overview of the key steps completed in this project. These steps are described in detail in the sections below. In addition, the R code provided with this report is also organized in this same format. The following are the key steps:

- Data Gathering
- Data Preparation
- Model and Parameter Selection
- Parameter Optimization: Regularization
- Model Training and Evaluation
- Other Factors Considered: Exploration & Visualization
- Other Modeling Approaches
- Results
- Conclusions

Data Gathering was provided in the project assignment. We were provided with a script of R code that created a training data set (named "edx") and testing data set (named "validation") from the publicly available MovieLens data set. This script was not modified in any way and appears "as is" in my R and RMD files.

Data Preparation is the process of cleaning and setting up the data to be analyzed. The MovieLens data is used widely and seems very clean. Nevertheless, it is always good practice to check the data and make sure that it is without errors and in the format that is expected.

Model and Parameter Selection describes the activity of reviewing and selecting an approach to modeling the problem and deciding which parameters to include in the model. In this project we use the modeling approach described in section 35.4 of our course textbook.

Parameter Optimization is the activity of optimizing the parameter values that have been included in the model. In this project, the user and movie effect parameters were optimized via **regularization**, which is described in section 35.5 of our textbook.

Model Training and Evaluation is the activity of training the chosen model and parameters on the "edx" data set, and then applying the trained model to the provided "validation" data set, and reporting the RMSE score.

Other Factors Considered: Exploration & Visualization describes through exploration and visualization the parameters that I considered adding to the model, specifically, date and genres, but did not, along with my reasons.

Other Modeling Approaches describes my exploration of algorithms used in the recommenderlab package, including matrix factorization, factor analysis, singular value decomposition (SVD), principal component analysis (PCA) and related methods.

Results simply summarizes the RMSE of the models built and the RMSE of the validation data set.

Conclusions describes some lessons learned in this project, which I am now applying in my individual data science project.

Methods & Analysis

Here we go into detail on each key step of the data analysis project. The code and output shown below is contained in the RMD file and executed with Knit.

Data Gathering

In this project the data was provided to us through the course website page.

The data is the 10M version of the MovieLens data set. This data was split 90/10 into a training data set ("edx") and a test data set ("validation"). The "edx" data set was designed to be partitioned into training and test sets for constructing a prediction system. The "validation" data set is to be used only for the final RMSE calculation.

Data Preparation

A typical data science project may require an extensive data preparation process to remove incomplete or erroneous data elements from the data set. In this case, accurate completion of the MovieLens Quiz ensured that we had obtained the data set as intended by edX course instructors. However, that does not necessarily mean that the data set is pristine and therefore I ran the following checks on it:

- First, I temporarily bound the edx and validation data sets together into a data frame called "data_set" to run my checks on all of the data.
- The columns were checked to verify that they were in the format expected.
- The entire data set was searched for any fields that stored "NA" instead of useful data.
- Since the movieId parameter was given as a numeric field, I used the the modulus function to search for any non-integer values, in case movieId was used for indexing.
- Ratings were tested to ensure that all were with the bounds of 0 to 5.

If any data was considered erroneous, then an error message would be provided and a recommendation to stop the analysis process would be given. Otherwise an affirmative message was given to continue with the analysis. The last step is to remove the temporary "data_set" file, which is no longer needed.

```
# Purpose is to ensure the dataset is complete and error-free
# Temporarily combine edx and validation data sets for checking...
data_set <- rbind(edx, validation)

# Check to see that columns are the types we expect
column_classes_are_incorrect <- !is.integer(data_set$userId) |
                                !is.numeric(data_set$movieId) |
                                !is.numeric(data_set$rating) |
                                !is.integer(data_set$timestamp) |
                                !is.character(data_set$title) |
                                !is.character(data_set$genres)
if (column_classes_are_incorrect) print("Unexpected column classes")

# Check to see that there are no NA values in the data fields
NAs_in_row <- data_set %>% filter(is.na(userId) |
                                  is.na(movieId) |
                                  is.na(rating) |
                                  is.na(timestamp) |
                                  is.na(title)   |
                                  is.na(genres))
NAs_present_in_data_set <- (nrow(NAs_in_row) > 0)
if (NAs_present_in_data_set) print("Investigate NA values in data")

# Check to see that movieId fields are integer values
movieId_not_integer_row <- data_set %>% filter(movieId%%1 !=0)
movieId_not_integer <- (nrow(movieId_not_integer_row) > 0)
if (movieId_not_integer) print("Investigate non-integer movieId fields")

# Check to see that ratings are between 0 and 5
rating_data_error <- data_set %>% filter(rating < 0.0 | rating > 5.0)
ratings_out_of_range <- (nrow(rating_data_error) > 0)
if (ratings_out_of_range) print("Data cleansing required to correct userId rows")

# Determine whether to stop the analysis because of corrupt data
if (column_classes_are_incorrect |
    NAs_present_in_data_set |
    movieId_not_integer |
    ratings_out_of_range) {
  print("DATA PREPARATION SUMMARY: Stop analysis and cleanse data.")
  } else {
  print("DATA PREPARATION SUMMARY: Data clean, continue with analysis.")
```

```
}
```

```
## [1] "DATA PREPARATION SUMMARY: Data clean, continue with analysis."
```

```
# If no data issues, then remove temporary data set
rm(data_set)

# ... and continue to the next phase of analysis...
```

Model and Parameter Selection

The model and parameter selection process began by building the recommendation system described in the course textbook to see how it performed relative to the RMSE levels shown in the rubric.

The "validation" data set should be used only in the final step of the model evaluation. To perform our model analysis and selection process, we need to create training and testing data sets from the "edx" training data set, which we do with the following code:

```
# Partition the data into training and test sets
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# To ensure we do not include users and movies in the test set that do not appear in
# the training set we remove those entries with the semi_join function
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

It is important to partition the test data set using the semi_join function on movieId and userId as was done with the "validation" data set. Otherwise, our trained model will not function as expected in the validation phase.

Before we begin the analysis process, we need to define the RSME metric that will be used to measure the performance of our different models and parameters:

```
# We will test the accuracy of our model with the RMSE function
RMSE <- function(predicted_ratings, true_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Similar to the textbook, our first model predicts the same rating for all movies, regardless of the user. Our estimate, mu, of each movie rating is merely the average of the training set ratings, and it gives a resulting mu similar to that found in our textbook.

```
mu <- mean(train_set$rating)
mu # 3.512482
```

```
## [1] 3.512482
```

If we naively predict all unknown ratings with this mu, we obtain the following RMSE:

```
naive_rmse <- RMSE(mu, test_set$rating)
naive_rmse # 1.059904
```

```
## [1] 1.059904
```

Which we will store in our results table:

```
rmse_results <- tibble(method = "Just the average",
                       RMSE = naive_rmse)
```

Now we will augment our model by adding the term b_i, which represents the average ranking for movie i. We call this the "movie effect". b_i can be estimated by the difference between the rating of each movie and mu, and is computed as follows:

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

We now form new our prediction, compute the RMSE against the test data set, and it add it to our table of results, which shows the improvement over our naive model:

```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- bind_rows(rmse_results, tibble(method="Movie Effect Model",
                                               RMSE = model_1_rmse))
rmse_results
```

```
## # A tibble: 2 x 2
##   method              RMSE
##   <chr>              <dbl>
## 1 Just the average   1.06
## 2 Movie Effect Model 0.944
```

We further refine our model by adding a term b_u that represent the average movie ranking for user u.

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
```

We can now construct predicted ratings using both our b_i and b_u terms and see how much the RMSE improves:

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

model_2_rmse <- RMSE(predicted_ratings, test_set$rating) # 0.866
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
                                 RMSE = model_2_rmse))
rmse_results
```

```
## # A tibble: 3 x 2
##   method                     RMSE
##   <chr>                     <dbl>
## 1 Just the average          1.06
## 2 Movie Effect Model        0.944
```

```
## 3 Movie + User Effects Model 0.866
```

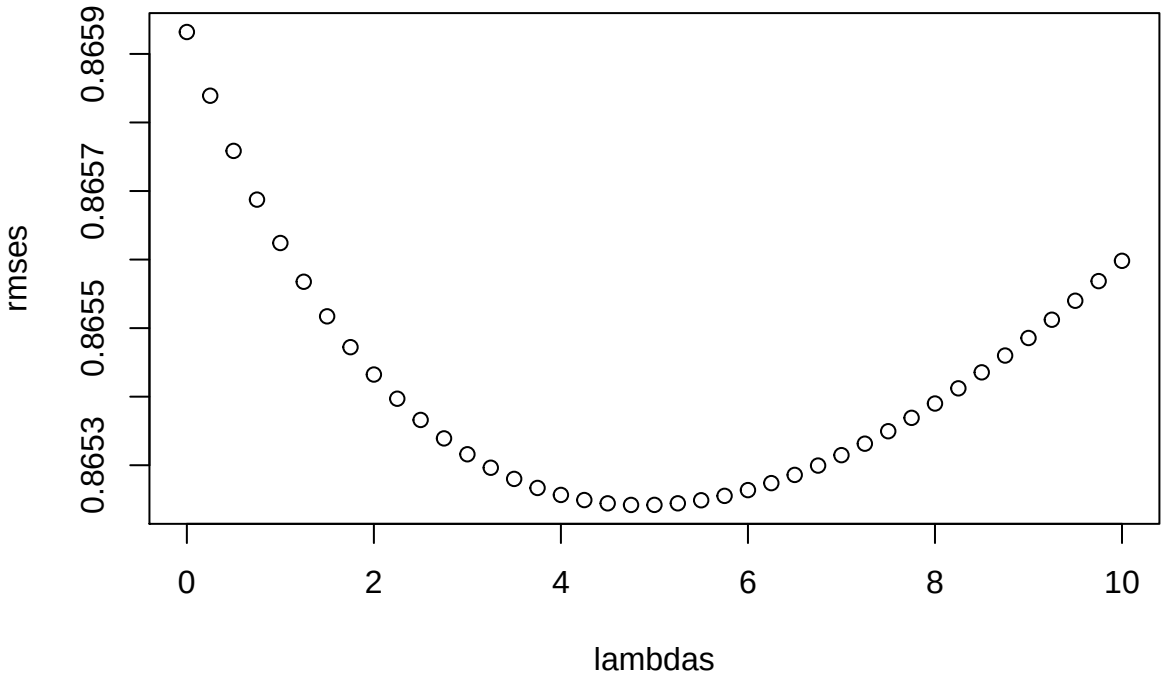
We can see from the table above that the RMSE of 0.866 associated with our current model is significantly better than the target RMSE of 0.87750. Although I have not shown it here, just to satisfy my own curiosity, the above analysis was run with several different set.seed() values and the results were nearly identical in all cases. Hence, from a grading standpoint, this model is considered sufficiently powerful and robust for this problem assignment.

However, similar to our textbook, I explored the possibility of optimizing the b_i and b_u parameters to potentially enhance the model.

Parameter Optimization: Regularization

As we witnessed in our course materials and exercises, small numbers of observations for any parameter can lead to large variability, which degrades a model. The parameters b_i and b_u may be optimized for low numbers of ratings for any particular movie i or user u. In this case, we can regularize the parameters by the number of observed ratings and a constant lambda. The following code computes a lambda value for b_i and b_u that minimizes the RMSE.

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

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

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

  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})

plot(lambdas, rmses)
```




```
lambda <- lambdas[which.min(rmses)]
lambda # 4.75
```

```
## [1] 4.75
```

As we can see from the plot a value of 4.75 minimizes the RMSE, but different values of lambda in the neighborhood of 4.75 also produce acceptable RMSE values.

We record the RMSE associated with this lambda in our results table.

```
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User Effect Model",
                                 RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie + User Effect Model	0.8652421

As we can see from the table, the regularization process adds little additional predictive power to our model.

Model Training and Evaluation

At this point we are satisfied with our model and the parameter optimization. We now train the model with the full training data set:

```
mu <- mean(edx$rating)
lambda <- 4.75

b_i <- edx %>%
```

```
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

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

predicted_ratings <- validation %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
```

We then evaluate our final trained model against the "validation" data set, and recorded the results in our data table.

```
final_rmse <- RMSE(predicted_ratings, validation$rating)

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Final Model with Validation dataset",
                                 RMSE = final_rmse))
final_results <- rmse_results
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie + User Effect Model	0.8652421
Final Model with Validation dataset	0.8648201

We will revisit this table in the Results section below. Before doing so, we will discuss parameters and models that I chose not to use.

Other Factors Considered: Exploration & Visualization

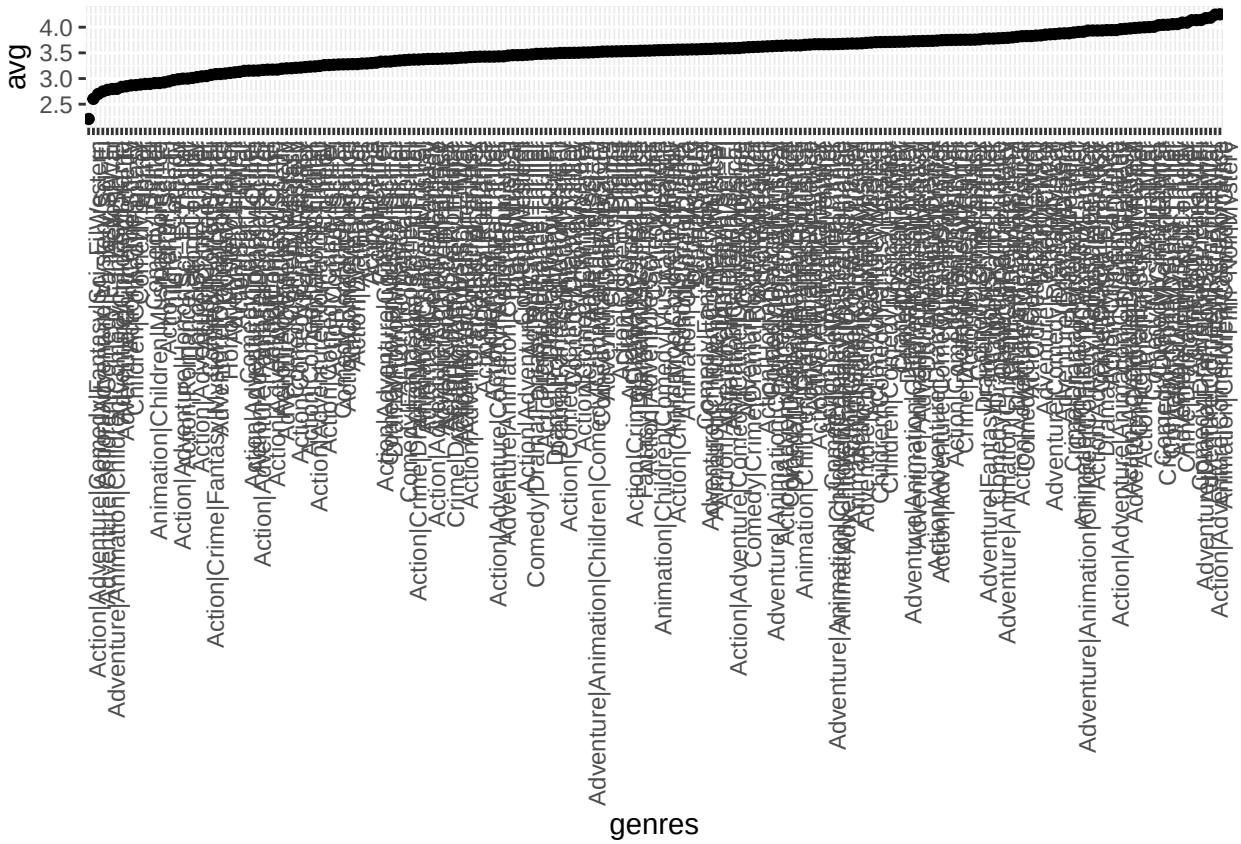
There are two other factors, genres and timestamp, in our data set that may be used to enhance our model. In this section we explore these two factors, and explain why they were not included in the final model.

In studying the MovieLens data set, we saw evidence of a "genre effect", which we can see in the plot below. Along the x-axis are the different genres, and the y-axis summarizes the mean ratings and box plots two standard errors above and below the mean, and then ordered by y-value.

Although the plot is a bit crowded in this document format, the reader can see evidence of a "genre effect".

```
genres_trend <- test_set %>%
  group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >= 1000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
```

```
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

plot(genres_trend)
```



To determine if genres adds predictive power to my model, we need to go back and redefine our training and test sets to ensure that our test set does not include genres that are not in our training set.

```
set.seed(1)
test_index <- createDataPartition(y = 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 = "genres")
```

We now run the same methodology used for the movie and user effects, and then compare the improvement in predictive power over our original naive model.

```
mu <- mean(train_set$rating)
mu # 3.512482
```

```
## [1] 3.512482
```

```
naive_rmse <- RMSE(mu, test_set$rating)
naive_rmse # 1.059909
```

```
## [1] 1.059909
```

```
genres_avgs <- train_set %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu))

predicted_ratings <- mu + test_set %>%
  left_join(genres_avgs, by='genres') %>%
  mutate(pred = mu + b_g) %>%
  pull(b_g)

model_genres_rmse <- RMSE(predicted_ratings, test_set$rating)
model_genres_rmse # 1.01781
```

```
## [1] 1.01781
```

```
genres_rmse_impact <- naive_rmse - model_genres_rmse
genres_rmse_impact # 0.042099
```

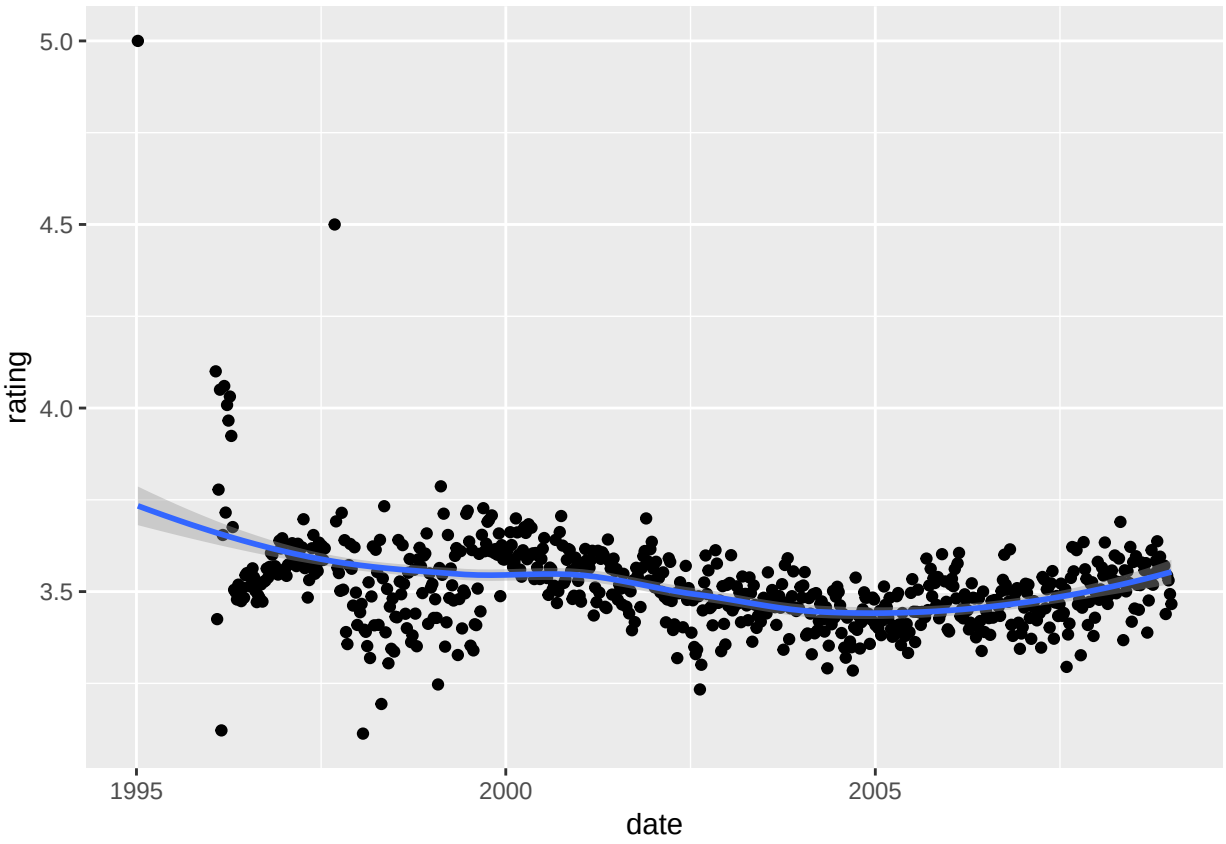
```
## [1] 0.04209938
```

We see that the genres effect improvement in the RMSE is 0.04, which is significantly smaller than the b_i and b_u effects. In addition, since the "validation" data set is not guaranteed to contain all of the genres, the actual improvement may be even smaller in the validation phase. For these reasons I did not include a genres parameter in my final model.

We now turn from the genres factor to the timestamp factor.

In studying the MovieLens data set, we saw some evidence of a "time" or "date effect", which we can see in the following plot, which shows average ratings by week through the years.

```
library(lubridate)
date_trend <- train_set %>% mutate(date = round_date(as_datetime(timestamp), unit = "week")) %>%
  group_by(date) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()

plot(date_trend)
```



Although this seems even weaker than the genres trend, we want to be thorough and explore the impact on our predictive model.

First, we need to add a "date" column to the data set, and then rebuild our training and testing data sets, using a semi_join to ensure that dates in the testing set appear in the training set.

```
edx_with_dates <- edx %>% mutate(date = round_date(as_datetime(timestamp), unit = "week"))

set.seed(1)
test_index <- createDataPartition(y = edx_with_dates$rating, times = 1, p = 0.2, list = FALSE)
train_set <- edx_with_dates[-test_index,]
test_set <- edx_with_dates[test_index,]

test_set <- test_set %>%
  semi_join(train_set, by = "date")
```

We now run our model as we have done several times before, to see if the "date effect" offers any improvement over the naive algorithm.

```
mu <- mean(train_set$rating)
mu # 3.512482
```

```
## [1] 3.512482
```

```
naive_rmse <- RMSE(mu, test_set$rating)
naive_rmse # 1.059909
```

```
## [1] 1.059909
```

```
# We compute term b_d to represent the average ranking for date d.
date_avgs <- train_set %>%
  group_by(date) %>%
  summarize(b_d = mean(rating - mu))

predicted_ratings <- mu + test_set %>%
  left_join(date_avgs, by='date') %>%
  pull(b_d)

model_date_rmse <- RMSE(predicted_ratings, test_set$rating)
model_date_rmse # 1.056202
```

```
## [1] 1.056202
```

```
date_rmse_impact <- naive_rmse - model_date_rmse
date_rmse_impact # 0.003707791
```

```
## [1] 0.003707791
```

As we expected, the date effect is almost negligible and therefore is not included in the model.

At this point in the analysis process, I reviewed the paper Winning the Netflix Prize: A Summary for additional ideas on improving my model, which led me to Matrix Factorization approaches. In particular, our class textbook led me to the recommenderlab package, which is discussed in the next section.

Other Models Considered

Our professor mentioned the recommenderlab package in our course textbook, in section 35.6, as tool for exploring matrix factorization, factor analysis, singular value decomposition (SVD), principal component analysis (PCA) and related methods. Recommenderlab is a framework and software for developing and testing recommendation algorithms.

Recommenderlab can be installed as follows:

```
install.packages("recommenderlab")
library(recommenderlab)
```

Please note that the following code appears in my R and RMD file, but I have the eval flag set to false in the RMD file, otherwise this code can crash the user's computer due to memory constraints.

One of our classmates posted a relevant English/German website on our class discussion board. This website compares recommender systems, and uses the MovieLens data set as an example, which provided an easy way to begin exploring this package. I borrowed code from this website to get started.

Recommenderlab requires that model data be stored in their "realRatingMatrix" format. To get our model data into the realRatingMatrix format requires first coaxing it into sparse matrix format.

```
sparse_ratings <- sparseMatrix(i = edx$userId, j = edx$movieId, x = edx$rating,
                               dims = c(max(edx$userId), max(edx$movieId)),
                               dimnames = list(paste("u", 1:max(edx$userId), sep = ""),
                                               paste("m", 1:max(edx$movieId), sep = "")))
```

The English/German website advocated a slightly different approach, which did not quite work for me. I did get the code to work by changing the matrix dimension parameter. The sparse matrix created by the code above has some rows with no data, which need to be removed with the code below.

```
sparse_ratings <- sparse_ratings[which(rowSums(sparse_ratings) > 0),]
```

Once the zero rows were removed, I was able to create their realRatingMatrix with our "edx" data set.

```
real_ratings <- new("realRatingMatrix", data = sparse_ratings)
```

After setting the seed, this matrix of data is fed into the evaluationScheme module, which allows users to create evaluation schemes using methods such as a simple split, bootstrap sampling, and k-fold cross validation. It also splits the training data from the test data. Testing is performed by withholding items with the "given" parameter, and the user's algorithm is evaluated on how closely it predicts the withheld values. A variety of user algorithms can be chosen. The one below uses the POPULAR option, which generates recommendations solely on the popularity of items. The websites mentioned above describe this in detail.

```
set.seed(1)
e <- evaluationScheme(real_ratings, method="split", train=0.8, given=-5)

model <- Recommender(getData(e, "train"), "POPULAR")

prediction <- predict(model, getData(e, "known"), type="ratings")

rmse_popular <- calcPredictionAccuracy(prediction, getData(e, "unknown"))
rmse_popular # 0.9234480
```

After almost an hour of computation on a MacBook Pro with 16 GB of memory, I was able to generate and evaluate my first model, and computed an RMSE of 0.9234480.

Unfortunately, my first model was the only one that I was able to run. I tried several others, using user-based collaborative filtering (UBCF) with different combinations of parameters, and item-based collaborative filtering (IBCF). All of the combinations failed because of computer memory limitations. (Please see error messages and commentary from other users commented below.)

```
model <- Recommender(getData(e, "train"), method = "UBCF",
                     parameter=list(normalize = "center", method="Cosine", nn=50))
prediction <- predict(model, getData(e, "known"), type="ratings")
# Error in asMethod(object) :
#   Cholmod error 'problem too large' at file ../Core/cholmod_dense.c, line 105
# "The error statement appears to confirm that you have a memory issue.
# You'll most-likely need more memory to analyze that dataset as-is."
rmse_ubcf <- calcPredictionAccuracy(prediction, getData(e, "unknown"))
rmse_ubcf

model <- Recommender(getData(e, "train"), method = "IBCF",
                     parameter=list(normalize = "center", method="Cosine", k=350))
prediction <- predict(model, getData(e, "known"), type="ratings")
# Error in asMethod(object) :
#   Cholmod error 'problem too large' at file ../Core/cholmod_dense.c, line 105
# "The error statement appears to confirm that you have a memory issue.
# You'll most-likely need more memory to analyze that dataset as-is."
rmse_ibcf <- calcPredictionAccuracy(prediction, getData(e, "unknown"))
rmse_ibcf
```

I tried several system changes suggested by users on stackoverflow, but to no avail. In the Conclusion section below I will discuss my plan to overcome these limitations in my own independent capstone project.

Results

Below is the final results table from the previous Methods and Analysis section.

```
final_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0599043
Movie Effect Model	0.9437429
Movie + User Effects Model	0.8659320
Regularized Movie + User Effect Model	0.8652421
Final Model with Validation dataset	0.8648201

Our model includes a regularized movie effect parameter and regularized user effect parameter. The model was trained on the "edx" data frame provided and evaluated on the "valuation" data frame provided. The resulting evaluated RMSE was 0.86482, which is well below the target of 0.87750.

I chose not to include a date effect parameter because it had minimal predictive power. A genres effect parameter had more predictive power, but my model did not require this effect to meet the target RMSE, and I was concerned that the setup we were given did not ensure the genres in the validation set were also in edx set. Hence, it seemed unwise to me to include this effect in my model.

Conclusions

I am disappointed that my exploration of the recommenderlab was stymied by problem-size and computer memory issues. I have a strong professional interest in these matrix factorization approaches, and I would like to consider using them in my independent capstone project. I am pursuing two avenues for continuing to explore this tool: (1) choose a independent data project with far fewer data elements (not 10M like this project), and (2) see if I can borrow some computing resources from an online resource (as opposed to using my MacBook Pro).