Prediction of MovieLens Ratings

Paul Gallagher

February 25, 2021

Executive Summary

MovieLens is a public domain dataset of movie rankings used for the study of recommender systems. The goal of this project was to train a model on a ninety percent partition of this dataset and validate this model on the 10% withheld partition. The root mean square error on the test data was the measure of the model accuracy; RMSE below 0.86490 was sought as one of the project success criteria.

Exploratory data analysis indicated that the training data contained 10 million ratings of 10,677 movies by 69,878 users. Hence, the ratings matrix representing these data was over 98% sparse. The median movie was rated by 122 users and the median user has rated 62 movies.

Simple models incorporating the overall mean rating and user and movie effects were constructed based on code supplied in the course materials. The simplest model had RMSE 1.06, far above the target, while a regularized user+movie effects model approached the target with RMSE 0.8654. Based on a sampling of the literature on recommender systems, matrix factorization was selected as a single model likely to improve substantially on the regularized user+movie model. With the use of the R package `recosystem`, a matrix factorization model was tuned and trained. This model with 35 latent factors gave RMSE of 0.7829 on the `validation` partition of the MovieLens 10M dataset, significantly better than the project success threshold. No improvement was found with modifications of this model including a weighted average with the regularized user+movie model and a factorization of the residual matrix of the regularized user+movie model. It is surmised that additional improvement could be achieved by using known attributes of the ratings such as quantization to whole and half numbers between 0 and 5, and the user preference for whole numbers over half numbers as model refinements. Step improvement might require incorporating time effects, genre effects and implicit user information (e.g. movies rated vs movies not rated).

The RMSE achieved is only slightly higher than reported in the recent literature with similar models with many more factors - 512 and 64 - than the 35 used here. The RMSE achieved here is actually better- 0.78 vs 0.82 – than the older published results for similar models. Thus, the matrix factorization model was very successful in the task of predicting witheld ratings in the MovieLens 10M dataset.

Introduction

This report describes an analysis of a large dataset of user ratings of movies – the MovieLens10M data – with the goal of developing a prediction model for unknown ratings, or ratings witheld from the model training.

This introduction provides brief information on the MovieLens organization and the datasets it maintains. The field of recommender systems is described briefly to give context to the current project. The goals of the project are summarized and the approach to project execution is outlined.

The Methods and Analysis section provides the details of the methods used for data preparation and exploratory analysis. It also describes the rationale behind the development of the prediction model. The results of that model are presented and discussed in the Results and Discussion section. In the final section, Conclusions from the data analysis and modeling are presented.

The MovieLens 10M dataset

GroupLens is a research lab at University of Minnesota, Twin Cities that maintains datasets related to recommender systems among other topics (GroupLens 2013). Among these are the MovieLens datasets containing user ratings of movies. The focus of the current project is the MovieLens 10M dataset, containing 10 million ratings from approximately 72,000 users of about 10,000 movies. This is a stable dataset intended for benchmarking, educational and research purposes. The project assignment information included a script to download the dataset and partition it into a 9 million rating training dataset `edx` and a one million rating `validation` dataset to be used only to assess the final model accuracy.

Recommender Systems

Recommender systems suggest products or service offerings to users or prospective customers based on a dataset of known user ratings of products. In commercial settings additional information is combined with ratings to form recommendations but in academic settings the predictions are often based on the known ratings alone. These systems have been widely-used by service providers (notably streaming media services) and retailers.

A typical use case for a recommender system is to recommend a list of new items that the user would likely rate highly. This helps the user deal with the overwhelming number of choices available online. For algorithm evaluation, the existing ratings dataset is usually split into a training set and a test set, and the algorithms are evaluated by their accuracy on the test set, for example by calculating the root mean square error. Other approaches to algorithm evaluation exist such as providing ("given n") or withholding a pre-determined number of ratings per user ("all but n") from the algorithm. In the current project, the partitioned dataset approach was used.

The Netflix Prize

During and after the Netflix competition that ran from 2006-2009, there was considerable effort invested by academic and other third party teams to improve recommender system algorithms and prediction accuracy. The Netflix competition awarded a $1,000,000 prize to the winning team for developing an algorithm that improved on the Netflix proprietary recommendation system's predictions by 10% as measured by root mean square error on a test dataset. The winning team used an ensemble of many different models to produce the winning final model (Koren 2009). Early on in the Netflix Competition, Matrix Factorization algorithms emerged as a computationally efficient and fairly accurate method to build an accurate rating prediction model (Chen 2011)). These algorithms seek to decompose the ratings matrix into two narrower matrices corresponding to latent factors representing users and movies. This decomposition or factorization is often labelled a singular value decomposition but is not in fact the classical SVD of linear algebra. There is no diagonal matrix and the factor matrices are not orthonormal. The technique emerged as a breakthrough early in the Netflix competition (Funk 2006)and remains the basis of many sophisticated models.

Goals

The goals of this project were to develop and evaluate a machine learning algorithm for ratings prediction in the MovieLens 10M dataset. A script to download the dataset and partition it into training and test sets was provided by the course staff. The required outputs of the project were this report, in PDF and Rmarkdown formats, to include reporting of the final test RMSE figure of merit, and an R script containing the code required to train and evaluate the model.

Approach

The approach to the project was developed through study of the literature related to recommender systems and in particular the publications emerging from the Netflix prize. These suggested that while construction

of a prize-winning model would be a large undertaking, a simple model based on one or two algorithms would be likely sufficiently accurate to beat the metrics stated in the project rubric(Bell et al. 2007). In particular, the literature suggested that matrix factorization, perhaps with an approach similar to the Funk SVD algorithm that emerged from the Netflix prize, would be computationally feasible with a single PC and would improve prediction accuracy over a simple lumped statistical model(Funk 2006).

Methods and Analysis

The analysis was conducted mainly in Microsoft Open R 4.0.2 running on a notebook PC with 16Gb RAM and an Intel i7-8565U CPU with four cores and hyperthreading capability. The early stages of the analysis were completed with R 3.6. The reason to switch to Microsoft Open R was to take advantage of the claimed parallel computing advantages, including use of the Intel MKL linear algebra functions.

The dataset was downloaded and split into the `edx` and `validation` datasets using the code provided in the course materials. The `validation` data was not used until the final calculation of predicted ratings and root mean square error of the model predictions. The `edx` data were used for exploratory data analysis, for model training, and for initial evaluation of models.

The dataset was further partitioned into a training part, `edx_train` and an internal test part, `edx_test`

```
# partition edx
set.seed(1748, sample.kind="Rounding")
test_index = createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train <- edx[-test_index,]
edx_test <- edx[test_index,]

edx_test <- edx_test %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")
```

Exploratory Data Analysis of `edx` dataset

We see that `edx` contains 9000055 movie ratings while `edx_train` and `edx_test` contain 7200043 and 1799976 ratings, respectively. The function `n_distinct` shows that there are ratings from 69878 distinct users in dataset `edx` and there are ratings for 10677 distinct movies. Thus, a fully dense matrix of ratings would have over 746 million entries so the ratings matrix for edx is actually 98.8% sparse (empty).

```
# How many movies?
n_distinct(edx$movieId)
```

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

```
# How many users?
n_distinct(edx$userId)
```

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

The datasets contain a `genres` field that lists each genre that the movie belongs to. We can use function `str_detect` to find out the number of movies in some common genres:

```
# How many movies of common genres
sum(str_detect(edx$genres, "Drama"))
```

```
## [1] 3910127
```

```
sum(str_detect(edx$genres, "Comedy"))
```

```
## [1] 3540930
```

```
sum(str_detect(edx$genres, "Thriller"))
```

```
## [1] 2325899
```

```
sum(str_detect(edx$genres, "Romance"))
```

```
## [1] 1712100
```

The sum of these is greater than the number of movies since each movie can belong to more than one genre.

```
# The most rated movies
modern <- edx %>% group_by(movieId, title) %>% summarize(count = n()) %>%
            arrange(desc(count))
quantile(modern$count)
```

```
##    0%   25%   50%   75%  100%
##     1    30   122   565 31362
```

Grouping by movieID and then counting the rows for each movie, we can find the number of times each movie was rated. The quantile function shows that the least rated movie has just a single rating while the most rated movie has 31362 ratings. The median movie has 122 ratings, and the mean number of ratings is 842.93856 per movie. The top six most rated movies are listed in the table.

Table 1: Most-rated Movies in edx Dataset

movieId	title	count
296	Pulp Fiction (1994)	31362
356	Forrest Gump (1994)	31079
593	Silence of the Lambs, The (1991)	30382
480	Jurassic Park (1993)	29360
318	Shawshank Redemption, The (1994)	28015
110	Braveheart (1995)	26212

Similarly, the number of ratings per user can be analyzed by grouping by `userId`.

```
# Ratings by user
user_count <- edx %>% group_by(userId) %>% summarize(count = n()) %>%
  arrange(desc(count))
quantile(user_count$count)
```

```
##   0%  25%  50%  75% 100%
##   10   32   62  141 6616
```

```
mean(user_count$count)
```

```
## [1] 128.8
```

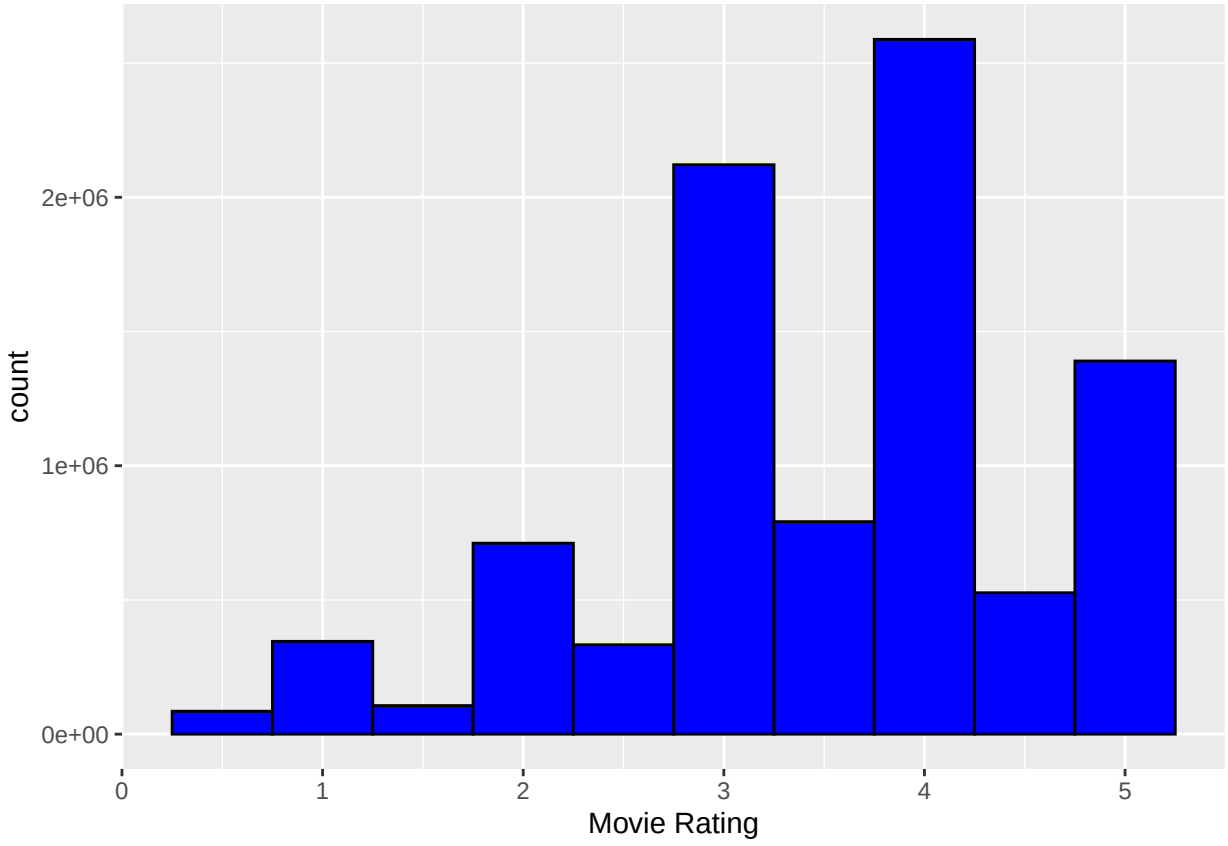
Grouping `edx` by `rating` shows that the lowest rating is 0.5, the highest rating is 5. Users favored whole number ratings over the half-star ratings, as is clear from the histogram of ratings distribution. The mean rating overall is 3.51247, the median rating is 4 and the standard deviation is 1.06033.

```
# Table and histogram of movie ratings
rate_count <- edx %>% group_by(rating) %>% summarize(count = n()) %>%
    arrange(desc(count))
knitr::kable(rate_count, caption = 'Distribution of Star Ratings in edx Dataset')
```

Table 2: Distribution of Star Ratings in edx Dataset

rating	count
4.0	2588430
3.0	2121240
5.0	1390114
3.5	791624
2.0	711422
4.5	526736
1.0	345679
2.5	333010
1.5	106426
0.5	85374

```
rate_count <- rate_count %>% arrange(rating)
edx %>% ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.5, fill = "blue", col = "black") +
  xlab("Movie Rating")
```



Simple Models - Overall Average, User and Movie Effects

For all model development and evaluation, `edx_train` was used as the training data set and `edx_test` was used for model evaluation. The final preferred model would then be trained on the full `edx` dataset before final evaluation on the `validation` data. To provide context, a very simple global mean was used as the starting point. That model may be written as $r_{ij} = \mu_{edx\_train}$ where r_{ij} is the rating by the i-th user of the j-th movie, μ is the average rating and the subscript edx_train denotes that the average is taken over the edx_train dataset. The code for the models in this section is adapted from the code snippets provided in the course textbook (Irizarry 2021)

```
# Try zeroth model -- just the average rating
tr_avg <- mean(edx_train$rating)

naive <- RMSE(edx_test$rating, tr_avg)
naive
```

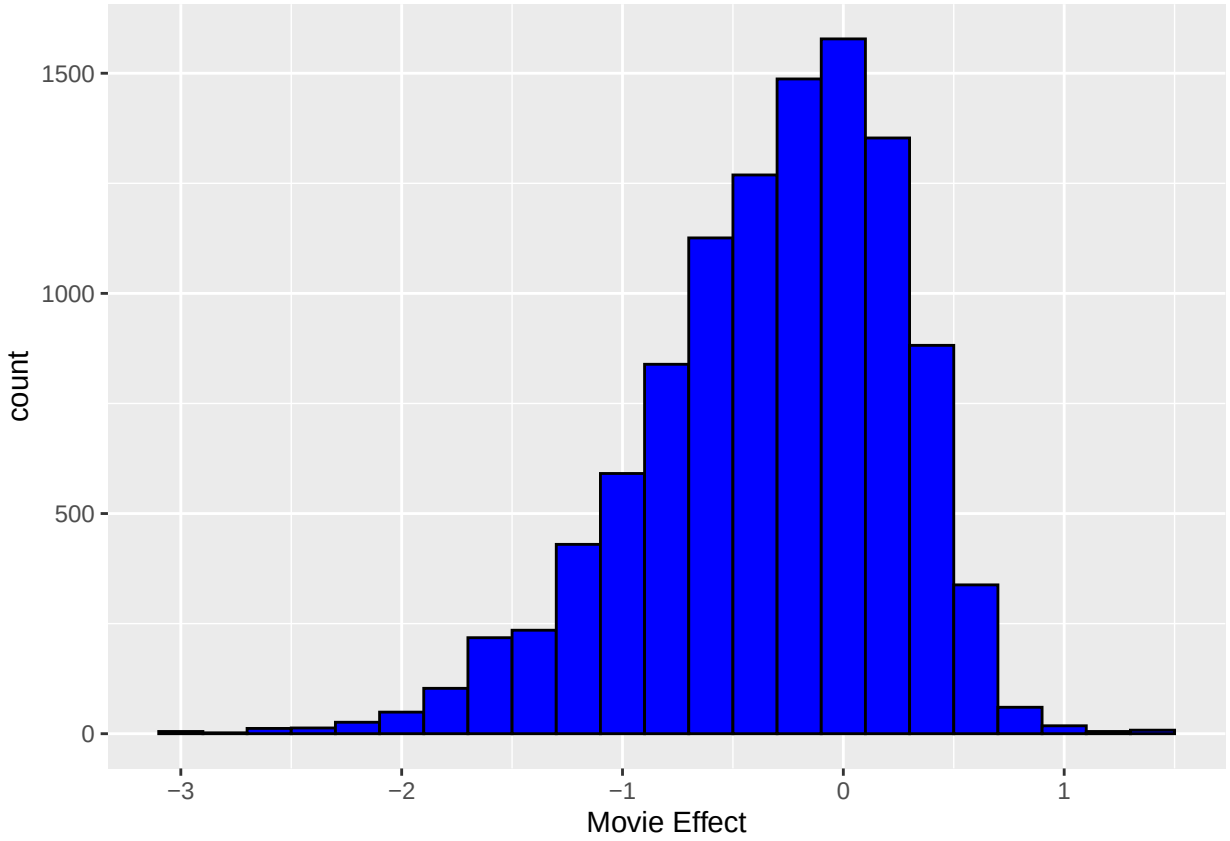
```
## [1] 1.0602
```

We see that this simple model, evaluated on `edx_test` has root mean square error(RMSE) approximately equal to the variance of the full `edx` dataset. The next step is to add terms for the user effect and the movie effect. The movie effect or bias for the j-th movie, $b_{movie,j}$ is calculated as the difference between μ_{edx_train}, the overall average rating and μ_{moviej}, the average rating for the j-th movie: $b_{movie,j} = \mu_{edx\_train} - \mu_{moviej}$. The code to calculate the movie effect is adapted from that provided in the course materials

```
# Add movie effect

movie_avgs <- edx_train %>% group_by(movieId) %>% summarize(b_i = mean(rating - tr_avg))
predicted_ratings <- tr_avg + edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
movie <- RMSE(predicted_ratings, edx_test$rating)
movie
```

```
## [1] 0.94396
```

```
movie_avgs %>% ggplot(aes(b_i)) +
  geom_histogram(binwidth = 0.2, fill = "blue", col = "black") +
  xlab("Movie Effect")
```




```
quantile(movie_avgs$b_i)
```

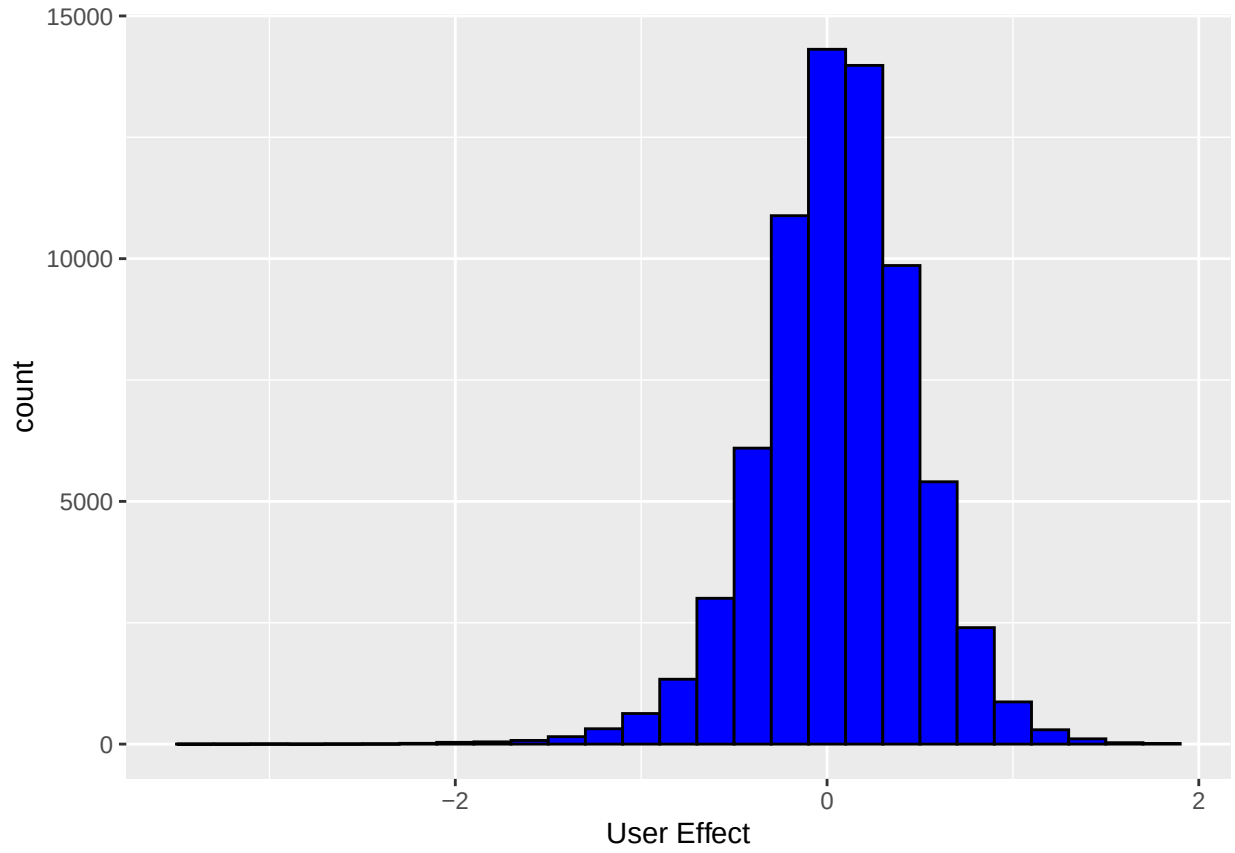
```
##       0%      25%      50%      75%     100%
## -3.01242 -0.67909 -0.24679  0.10042  1.48758
```

The model RMSE is improved by about 11% to 0.944 by accounting for the movie effect. The histogram indicates a wide distribution of movie effects from about -3 to 1.5. The median effect is -0.247. The next stage in model development is to add a user effect. This is calculated similarly to the movie effect as the mean of the difference between the overall average rating and the movie model prediction, averaged over all users.

```
# Add user effect

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

mov_us_predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = tr_avg +b_i +b_u) %>%
  pull(pred)
mov_us <- RMSE(mov_us_predicted_ratings, edx_test$rating)
mov_us
```

```
## [1] 0.8660774
```

```
user_avgs %>% ggplot(aes(b_u)) +
  geom_histogram(binwidth = 0.2, fill = "blue", col = "black") +
  xlab("User Effect")
```




```
quantile(user_avgs$b_u)
```

```
##          0%         25%         50%         75%        100%
## -3.40697186 -0.18290266  0.07248844  0.32511715  1.87688962
```

```
median(user_avgs$b_u)
```

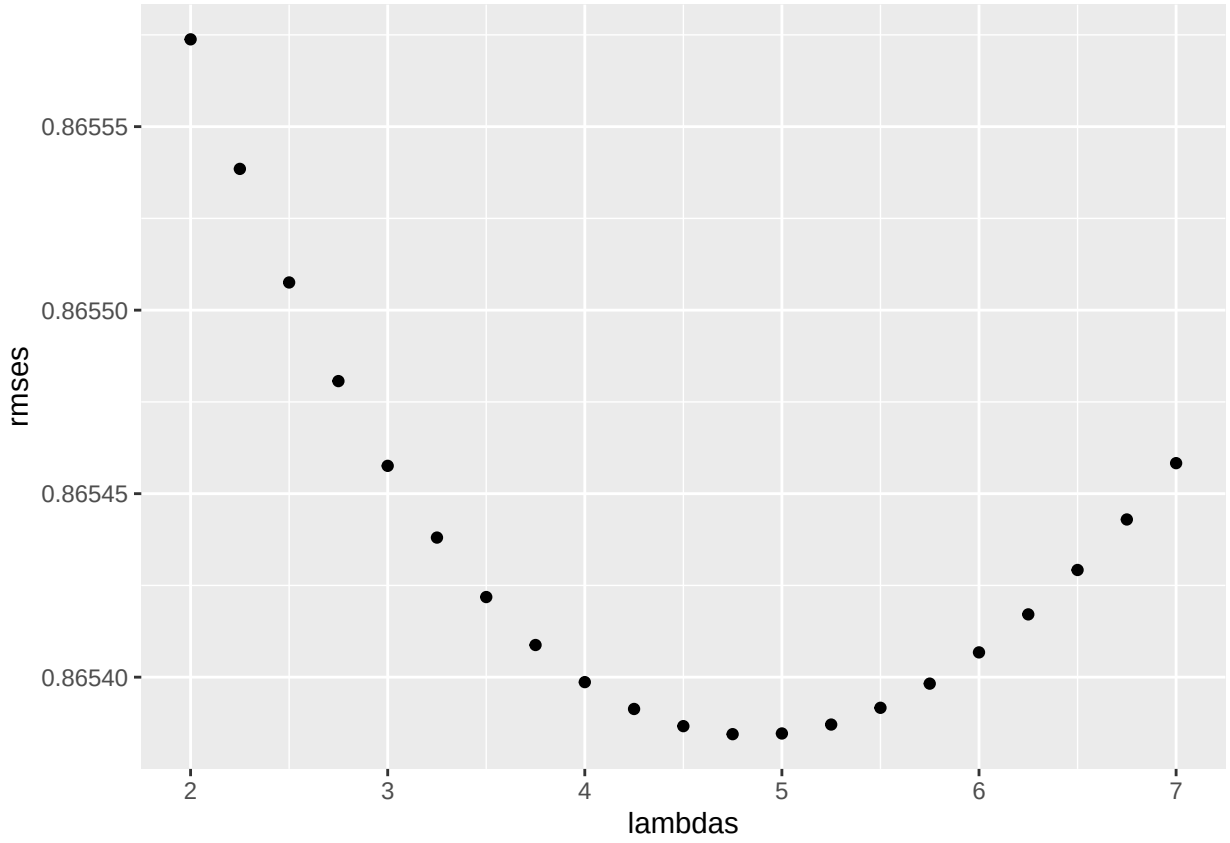
```
## [1] 0.07248844
```

Incorporating the user effect further improves the model RMSE by about 9% over the movie effect model, leading to a cumulative improvement of 18% over the overall mean model. The user effect distribution is somewhat narrower and centered closer to zero than the movie effect. The median is 0.072, compared to -0.247 for the movie effect. The course materials indicate that a further improvement should be available from regularization. The regularization calculation effectively reduces the weighting of ratings from users that rated few movies and movies that received few ratings. This may improve the model's overall performance by placing more weight on the more complete and more certain information from more-active users and more-rated movies. A single regularization parameter was applied to both the movie and user effects at once. The approximate optimization method taught in the course materials was used to find the optimum for λ

```
# Optimize lambda for user + movie regularization model

lambdas <- seq(2, 7, 0.25)

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

    tr_avg <- mean(edx_train$rating)

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

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

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

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


lambda_opt <- lambdas[which.min(rmses)]
lambda_opt
```

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

```
qplot(lambdas, rmses)
```




```
best_rmse <- min(rmses)
best_rmse
```

```
## [1] 0.8653845
```

This form of regularization results in a small improvement in RMSE of the model predictions for `edx_test`, from 0.8661 to 0.8654. It is possible that applying this model to the full `edx` set and then to the `validation` set would result in a slightly better RMSE, but it also seems that we are seeing diminishing returns with this simple model considering only base user and movie effects. With limited time available to invest in this project, I sought a single step change in model sophistication.

Matrix Factorization Model

Clearly there is additional information available in the data file, including genre, movie release year and review timestamp. Further, there is likely implicit information in the movies that users chose to rate. The publications from the Netflix prize indicate that all of these factors can be used to improve model accuracy, but also indicate that no single additional factor would be likely to provide a step-change improvement(Koren 2009). On the other hand, singular value decomposition(SVD) and related matrix factorization approaches did yield single models with step improvements in accuracy. For this reason, I selected singular value decomposition approaches and in particular the "Funk SVD" approach as the next direction for improved modeling efforts.

The SVD decomposes a rectangular matrix X of dimensions i by j into the product UDV^T where U has dimensions i by k, D is a k by k diagonal matrix and V has dimensions j by k. In the current context, it is expected that $k << i, j$. In the Funk SVD, D is incorporated implicitly in U and V and the matrices are

transposed so that we seek the decomposition $X = U^T V$ so that both U and V are matrices assembled from k column vectors. These vectors may be conceptualized as representing latent factors that the factorization has identified for users and for movies. The Funk SVD algorithm takes advantage of the fact that if X is the ratings matrix for the training dataset, then it is a sparse matrix and the algorithm need only consider the non-zero elements when iteratively refining its estimates of the columns of U and V. The algorithm uses a steepest gradient approach to find estimates of these columns one by one. If the algorithm is successful, each successive column will represent a smaller part of the total variance, and the matrix will be factorized accurately with a limited number of columns. Conceptually, this approach seemed to offer a plausible route to modeling the finer structure of the information contained in the training data.

A note in the course textbook recommended investigation of the R package `recommenderlab` to follow up on application of matrix factorization models. This package has functions for training and applying several different models including the Funk SVD. The package seems to be focused on prototyping these models on smaller data sets. Thus, attempting to use the Funk SVD function from `recommenderlab` on the full `edx` dataset resulted in an error message that indicated that a greater than 2Gb vector could not be allocated. Study of the source code for the function showed that it calculated the full ratings matrix X as the product $U^T V$. This is a dense matrix of dimensions 10677 x 69878, containing an estimated rating by every user of every movie in the dataset. Note that at 4 bytes per real number, the approximate memory requirement would be 4x10677x69878 or $2.98x10^9$ bytes, i.e. over 2Gb. Thus, it seemed that `recommenderlab` could not be used to train a Funk SVD model for `edx` or `edx_train`.

It seemed feasible to modify the source code of the `recommenderlab` function and run it as a local function to calculate the SVD. In this code, only the entries in X corresponding to known entries in `edx` or `edx_train` would be calculated. This code ran without error but executed too slowly to be a viable route to a working model. Compiling the function and using Microsoft Open R to take advantage on Intel MKL low level linear algebra functions improved performance by about 30% but order-of-magnitude improvement was required. This would require structuring the inner computation loops to use fast low level calculations and this task was outside my capability and outside the scope of this project.

The next option was to search for an efficient R package for sparse matrix factorization that could be run on a single personal computer. Fortunately, this led to identification of the R package `recosystem` which has these capabilities (Chin et al. 2015; Qiu 2021). This package is an R wrapper for LIBMF which is a set of C++ routines for matrix factorization, taking advantage of parallel processing and other techniques to achieve very fast computation. Much of the computational detail of the LIBMF algorithm is beyond the scope of this report, but can be found in papers published by the developers. The algorithm approximates the matrix factorization by minimizing a square error cost function with regularization terms. Experience has shown that a pure square error cost function is likely to result in over-fitting. The algorithm uses a learning rate parameter to control the step size and assist with convergence. The learning rate and the regularization coefficients are tunable and user-specifiable. The number of columns or latent factors in the factor matrices is also selectable. Several details of computation are also user-specifiable: these include the number of threads for parallel computation, the number of iterations and the number of blocks into which the X matrix is split

The `recosystem` package contains functions for tuning and training a matrix factorization model as well as for predicting ratings based on the trained model. Initially these functions were used to train a model on `edx_train` and evaluate it on `edx_test`. The package requires data in a three column file, the columns being userId, movieId and rating. The data file may be on disk or in memory and the function outputs may also be written to disk or output to memory. Initial tuning runs with `edx_train` showed that the algorithm is very efficient. By trial and error, I found that using 45 factors and 40 iterations gave low final error. More factors sometimes resulted in a higher RMSE solution. Additional iterations above forty gave limited or no improvement in RMSE. Of note, the tuning algorithm used five-fold cross-validation and reported RMSE in the range 0.80 to 0.84 in these initial tests. After optimization of the regularization coefficients and the learning rate, the model was trained with the optimized parameters and then a prediction on edx_test was evaluated.

All calculations were run on a Windows 10 laptop PC with an Intel i7-8565U processor. This processor has four cores with hyperthreading capability and a base clock speed of 2 Gigahertz. Empirically, the training

functions seemed to run fastest on this setup with 8 threads and 32 blocks.

```
# Investigate advanced algorithms - matrix factorization etc
# Matrix algebra and Matrix factorization libraries
library(recosystem)
library(Matrix)

# select user, movie and rating columns from edx and partitions
edx_trips <- edx %>% select(userId, movieId, rating)
edxtr_trips <- edx_train %>% select(userId, movieId, rating)
edxtst_trips <- edx_test %>% select(userId, movieId) %>% mutate(pred = 0)

# Create Recommender System Object

r <- Reco()

# assemble recosystem data files for edx_train and edx_test partition
reco_data <- data_memory(edxtr_trips$userId, edxtr_trips$movieId, edxtr_trips$rating, index1 = TRUE)
reco_test <- data_memory(edxtst_trips$userId, edxtst_trips$movieId, edxtst_trips$rating, index1 = TRUE)

# tune parameters for matrix factorization algorithm

set.seed(1916, sample.kind = "Rounding")

# recosystem tuning function
 res =r$tune(reco_data, opt = list(dim =35L,
                                costp_l1 = 0,
                                costp_l2 = c(0.05,0.06, 0.07),
                                costq_l1 = 0,
                                costq_l2 = c(0.05,0.06,0.07),
                                lrate    = 0.1,
                                nthread = 8,
                                niter = 40,
                                nbin = 32,
                                verbose  = FALSE)
        )
res$min
```

```
## $dim
## [1] 35
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.05
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.07
##
## $lrate
```

```
## [1] 0.1
##
## $loss_fun
## [1] 0.799021
```

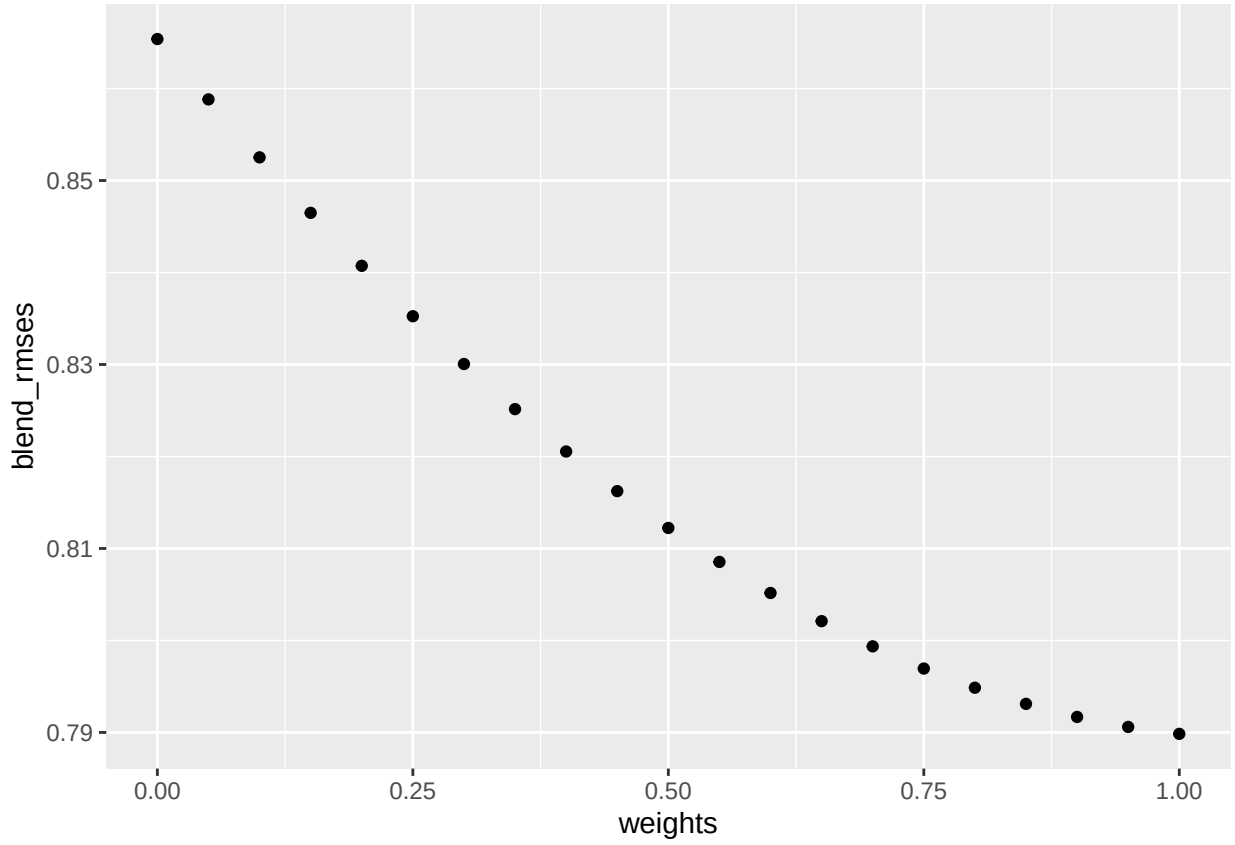
```
# recosystem training function for matrix factorization model
r$train(reco_data, opts = c(res$min, nthread = 8, nbin = 32, niter = 40, verbose = FALSE))
result =  r$output(out_memory(),out_memory())

# recosystem prediction of internal test partition of edx set
pred_test = r$predict(reco_test, out_memory())
edxtr_MF_error <- RMSE(pred_test, edx_test$rating)
edxtr_MF_error
```

```
## [1] 0.789922
```

The RMSE for this model applied to the edx_test dataset is 0.78992, almost a 9% improvement over the regularized model with user and movie effects, and over 25% improvement on the global mean model. Two variants of this model were evaluated to see if a further improvement would be readily available. First a weighted combination of regularized user+movie model and the matrix factorization model was assesses. Simple linear combinations of the two models' predictions were evaluated by calculating RMSE on `edx_test`. None of these combinations improved on the RMSE from the matrix factorization model alone.

```
# Function to calculate user and movie effects for regularized user-model
reg_user_item <- function(data, l) {

  avg <- mean(data$rating)

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

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

  predicted_ratings <-
    data %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = avg + b_i + b_u)
  }

# Extract training data user and movie effects
edxtr_pred <- reg_user_item(edx_train, lambda_opt)
edx_pred <- reg_user_item(edx, lambda_opt)
# Form b_u sparse vectors
edxtr_pred_u <- edxtr_pred %>% select(userId, b_u)%>% distinct(userId, .keep_all = TRUE)
edxtr_pred_i <- edxtr_pred %>% select(movieId, b_i) %>% distinct(movieId, .keep_all = TRUE)

# form regularized base model predictions for edx_test
edx_test_trip <- edx_test %>% select(userId, movieId, rating)
```

```
edx_test_pred <-
  edx_test_trip %>%
  left_join(edxtr_pred_i, by = "movieId") %>%
  left_join(edxtr_pred_u, by = "userId")
edx_test_pred <- edx_test_pred %>% mutate(pred = tr_avg + b_u + b_i)
# test weighted average of models

weights <- seq(0, 1, 0.05)

blend_rmses <- sapply(weights, function(w) {
  blend_pred <- w*pred_test + (1-w)*edx_test_pred$pred
  RMSE(blend_pred, edx_test$rating)
})
qplot(weights, blend_rmses)
```




```
min(blend_rmses)
```

```
## [1] 0.7898479
```

The second modification was to subtract the regularized user+movie predictions from the ratings matrix before running the matrix factorization. The idea was that the regularized model may be capturing the main user and movie effects well and that factorizing the residual matrix would allow the remaining factors to be extracted more accurately.

```
# Matrix Factorization of Residuals of Regularized User+Movie model

# Calculate residuals for edx and edxtra
edxtr_res <- edxtr_pred %>% select(userId, movieId, rating, pred) %>%
  mutate(resid = rating-pred) %>% select(userId, movieId, resid)
edx_res <- edx_pred %>% select(userId, movieId, rating, pred) %>%
  mutate(resid = rating-pred) %>% select(userId, movieId, resid)

r_resid <- Reco()

# assemble recosystem data matrix for residuals in edx_train partition
reco_resid <- data_memory(edxtr_res$userId, edxtr_res$movieId, edxtr_res$resid, index1 = TRUE)
# tune model for residual matrix
resid_rec =r_resid$tune(reco_resid, opt = list(dim =35L,
                                               costp_l1 = 0,
                                              costp_l2 = c(0.05,0.06, 0.07),
                                              costq_l1 = 0,
                                              costq_l2 = c(0.05,0.06,0.07),
                                              lrate    = 0.1,
                                              nthread = 8,
                                              niter = 40,
                                              nbin = 32,
                                              verbose  = FALSE)
)
resid_rec$min
```

```
## $dim
## [1] 35
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.07
##
## $costq_l1
## [1] 0
##
## $costq_l2
## [1] 0.06
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.802357
```

```
# recosystem training function for matrix factorization model

r_resid$train(reco_resid, opts = c(resid_rec$min, nthread = 8, nbin = 32, niter = 40, verbose = FALSE))
result =  r_resid$output(out_memory(),out_memory())

pred_test_res = r_resid$predict(reco_test, out_memory())
```

```
edx_test_pred_res <- edx_test_pred %>% mutate(pred_res = pred + pred_test_res)
test_pred_res_error <- RMSE(edx_test_pred_res$pred_res, edx_test$rating)
test_pred_res_error
```

```
## [1] 0.802196
```

Again, this modification to the model did not provide an improvement over the matrix factorization model. Therefore the final model selected for validation testing was the matrix factorization model. We proceeded with tuning the model parameters for the full `edx` dataset, training the model on `edx` and running the model evaluation on the `validation` data.

```
# select user, movie and rating columns from edx and partitions
edx_trips <- edx %>% select(userId, movieId, rating)

# Create Recommender System Object

r_edx <- Reco()

# assemble recosystem data matrix for full edx dataset
reco_data_edx <- data_memory(edx_trips$userId, edx_trips$movieId, edx_trips$rating, index1 = TRUE)

# tune parameters for matrix factorization algorithm

set.seed(1916, sample.kind = "Rounding")

# recosystem tuning function

res_edx =r_edx$tune(reco_data_edx, opt = list(dim =35L,
                              costp_l1 = 0,
                               costp_l2 = c(0.01,0.04,0.07,0.1),
                               costq_l1 = 0.01,
                               costq_l2 = c(0.01,0.04,0.07,0.1),
                               lrate    = 0.1,
                               nthread = 8,
                               niter = 40,
                               nbin = 32,
                               verbose  = FALSE)
        )
res_edx$min
```

```
## $dim
## [1] 35
##
## $costp_l1
## [1] 0
##
## $costp_l2
## [1] 0.01
##
## $costq_l1
## [1] 0.01
##
```

```
## $costq_l2
## [1] 0.1
##
## $lrate
## [1] 0.1
##
## $loss_fun
## [1] 0.790939
```

```
# Train matrix factorization model on full edx dataset

r_edx$train(reco_data_edx, opts = c(res_edx$min, nthread = 8, nbin = 32, niter = 40, verbose = FALSE))
result =  r_edx$output(out_memory(),out_memory())

# Predict ratings for validation dataset
valid_trip <- validation %>% select(userId, movieId) %>% mutate(pred = 0)
valid_reco <- data_memory(valid_trip$userId, valid_trip$movieId, valid_trip$rating, index1 = TRUE)
pred_valid = r_edx$predict(valid_reco, out_memory())
valid_error <- RMSE(pred_valid, validation$rating)
valid_error
```

```
## [1] 0.782886
```

Results and Discussion

The root mean square error of 0.78289 of the matrix factorization model on the `validation` dataset represents a substantial improvement over all of the simpler models evaluated in this project. This was expected given the good performance of the model when tested on the `edx_test` partition of the the training data, and given the substantial literature confirming that matrix factorization (MF) is an effective modeling technique for this application.

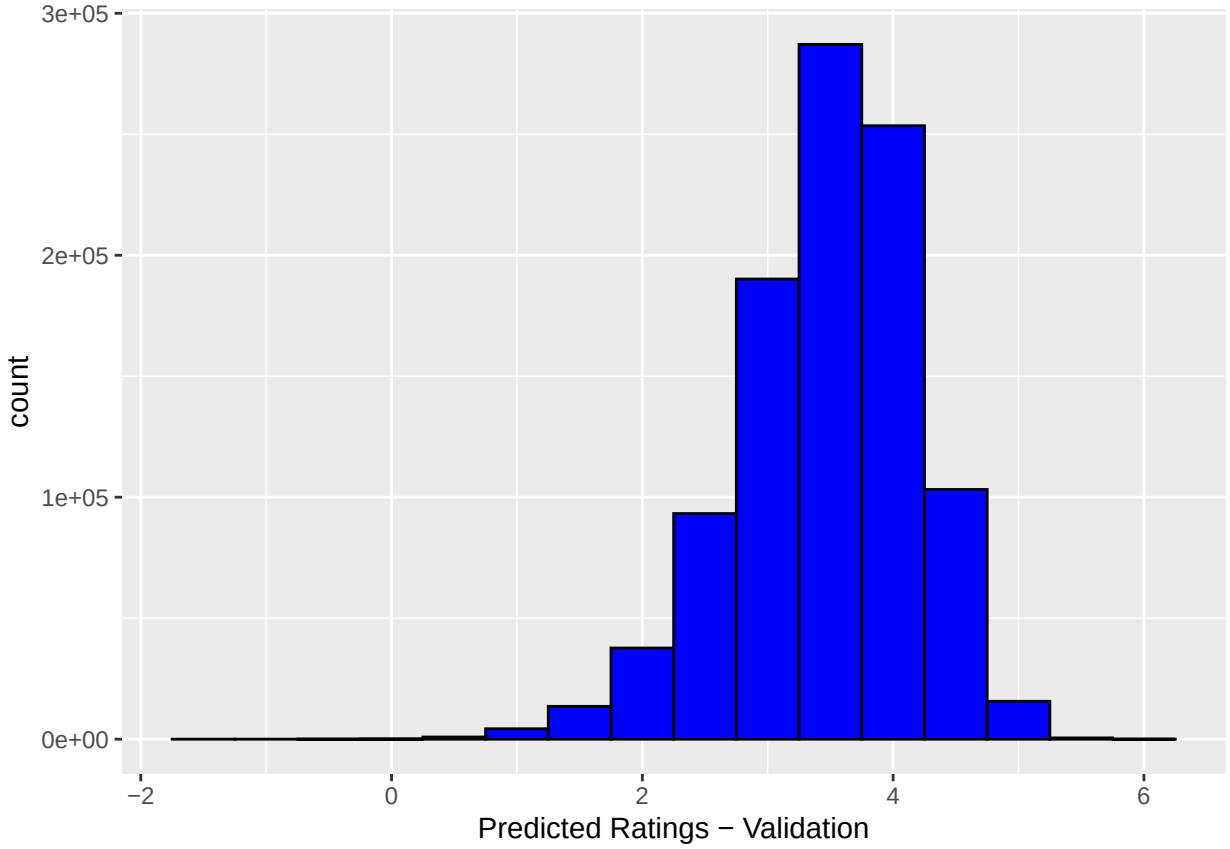
Rendle, Zhang, and Koren (2019) list RMSE they observed for several recommender system models, using 10-fold cross-validation on Movielens 10M as a benchmark test. They quote a value of 0.7720 for a 512 factor model and 0.7763 for a 64 factor case. Thus, a result of 0.7828 for a 35 factor model as we used here seems roughly in line with these expert results. Rendle et al also quote a historic result of 0.8256 for regularized SVD which is essentially our model. Clearly the current model outperforms that result. Note that our result is for a single 90:10 partition of the MovieLens data as opposed to the average of a 10-fold cross-validation, so our result is from similar but not identical methodology to Rendle et al.

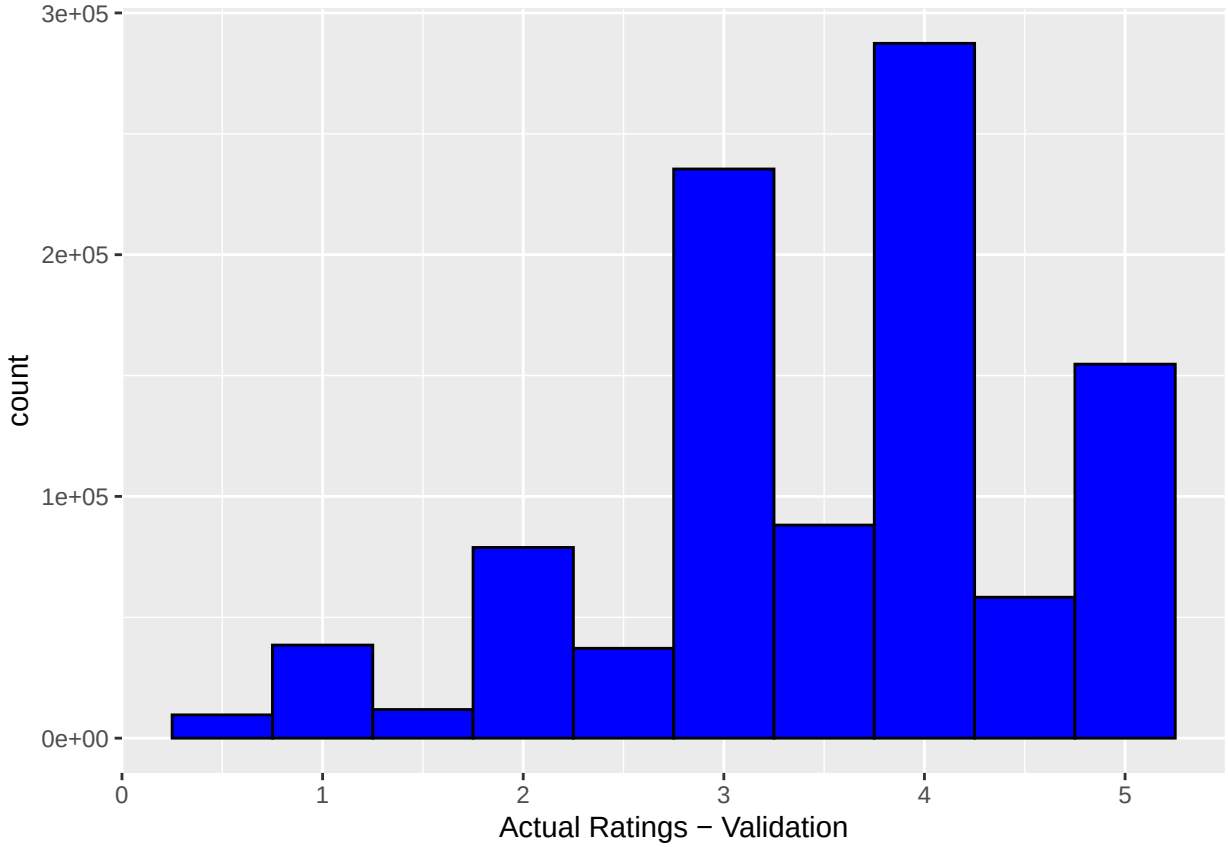
Neither a weighted average of the regularized user+movie effects and the MF model nor application of the to a residuals matrix rather than to the ratings matrix produced an improvement in the model performance. Note that the objective function for training the MF model included regularization terms. This may explain why averaging with the simpler regularized model did not improve performance. The residuals matrix model performed similarly to the ratings matrix MF model, indicating that there was no advantage to treating the main user and movie effects separately. Note that the factorization algorithm used here does not estimate the effect vectors sequentially. Rather it decomposes the matrix into blocks and iteratively calculates updates for the blocks in parallel computation threads. The original Funk SVD algorithm solved for the effect vectors by gradient descent and so would tend to find bigger effects first. Perhaps the blocked and parallel approach, together with a fairly large number of latent effects (35) hints at a reason why modeling the residuals is no more accurate than modeling the ratings.

There are additional steps that could be taken to improve the accuracy somewhat. No constraint was placed on the predicted ratings although we know that actual values below 0.5 or greater than 5 are not present. Thus clipping the predictions to force them to fall in the 0.5 to 5 range would improve the accuracy. This

could be done in various ways, from simply forcing predictions below 0.5 to 0.5 and above 5 to 5, to binning all the predictions in the half-start and whole-star levels that we know are the actual ratings.

The plots below show the distribution of predicted and actual ratings for `validation`.





The plots show that while the model has some error due to a few predictions lying above 5 or below 0, the greater issue appears to be that the model has not captured the user tendency to award many more whole star ratings than half-star ratings. One way forward might be to use unequal bin widths with wider bins around the whole numbers capturing more ratings and narrower bins capturing fewer predictions at the half-star levels.

The biggest weakness of the MF model is that doesn't generate ratings for movies or users for which it has no training ratings. Because of this issue, referred to as the "cold-start" problem, such matrix factorization models are used in concert with other models that may draw on movie attributes and user data to generate estimated ratings in the absence of a rating history.

Conclusions

The MovieLens 10M dataset was used to train a matrix factorization model for predicting withheld ratings. several simpler models were developed and tested to provide comparative prediction accuracy data. The root mean square error on test data was used as the measure of model accuracy. The principal conclusions from this work are:

1. The matrix factorization model was much more accurate, with RMSE 0.78289, than the best simple model tested, the global mean with user and movie effects and regularization, with RMSE 0.86538. The table below summarizes the models developed and their RMSE values.

Table 3: Model RMSEs

Model_Name	Test_Dataset	RMSE
Just the Mean	edx_test	1.06018
Mean + Movie	edx_test	0.94396
Mean + User + Movie	edx_test	0.86608
Mean + User + Movie + Regularization	edx_test	0.86538
Matrix Factorization	edx_test	0.78992
Residuals Matrix Factorization	edx_test	0.80220
Matrix Factorization - Final	validation	0.78289

2. The matrix factorization model result as measured by RMSE, was not improved by taking a weighted average with the User+Movie_Regularized model, nor by factorizing the residuals matrix for the latter model, rather than the raw ratings matrix.

3. Further refinement to the matrix factorization model could address unused information from the training data such as the quantized nature of the ratings, the preference of users for whole star ratings over half-stars, the genre data, the movie release year and the review timestamp information.

References

Bell, Robert M, Yehuda Koren, Chris Volinsky, and T Labs. 2007. "The BellKor Solution to the Netflix Prize," 15.

Chen, Edward. 2011. "Winning the Netflix Prize: A Summary." http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/.

Chin, Wei-Sheng, Yong Zhuang, Yu-Chin Juan, and Chih-Jen Lin. 2015. "A Fast Parallel Stochastic Gradient Method for Matrix Factorization in Shared Memory Systems." *ACM Trans. Intell. Syst. Technol.* 6 (1): 1–24. https://doi.org/10.1145/2668133.

Funk, Simon. 2006. "Netflix Update: Try This at Home." https://sifter.org/~simon/journal/20061211.html.

GroupLens. 2013. "What Is GroupLens?" *GroupLens*. https://grouplens.org/about/what-is-grouplens/.

Irizarry, Rafael A. 2021. *Chapter 33 Large Datasets | Introduction to Data Science*.

Koren, Yehuda. 2009. "The BellKor Solution to the Netflix Grand Prize," 10.

Qiu, Yixuan. 2021. "Yixuan/Recosystem."

Rendle, Steffen, Li Zhang, and Yehuda Koren. 2019. "On the Difficulty of Evaluating Baselines: A Study on Recommender Systems." *arXiv:1905.01395 [Cs]*, May. http://arxiv.org/abs/1905.01395.