MovieLens

Herdiantri Sufriyana

9/5/2020

Introduction

In this project, the dataset consisted of a training set (edx) and a validation set (validation). The dataset contains data about user rating on movies between 1 to 5 (with half). Here are the dimension of the datasets:

```
## edx dataset has 9,000,055 rows and 6 columns
```

```
## validation dataset has 999,999 rows and 6 columns
```

This project goal was to predict user rating with target root mean squared error (RMSE) <0.86490.

Key steps that were performed:

1. Data preprocessing
2. Exploratory data analysis
3. Model development

1. Baseline model
2. User-centered model
3. Movie k-neighbor model
4. User implicit neighbor rating model
5. Standard singular value decomposition (SVD) model
6. Ensembl model: majority vote, average rating, and stacking by elastic net regression

4. Model validation

———————————— NOTE ————————————

We commented some codes and substitute these with the note of the processing. If you want to run it by yourself, you need to commented of those parts (FYI, you can commented on/off multiple lines by selecting the lines and use Ctrl+Shift+C on Windows). Any variable, of which the code is commented, is loaded from RDS file (automatically downloaded, please make sure you have internet connection). We did this since these codes would take a fairly or even very long time to finish.

Methods

Programming environment

We used devtools and BiocManager to control versions of the package. If you want to run this code, make sure you also run the package installation and synchronization. First, this code will install and load devtools and BiocManager to control versions. This code was developed using R 4.0.2. If you run it, make sure your R version is 3.5 above since the random seed was based on the later version. if not, you need to change all of the seed; otherwise the results is possibly different.

All packages is installed using BiocManager with Bioconductor version 3.11.

Then, load the packages.

We will use theme set from DS labs.

Data retrieval

Here we retrieved edx and validation set. These dataset was saved to RDS files. We turned off this code block. You can just run the next block, but make sure you load the session file (.RData) and the RDS files.

Data preprocessing

For data preprocessing, we checked for missing values. There was no missing values for both edx and validation sets.

```
# Load variables to RDS on 2020-09-05
if(!all(edx_files %in% list.files())) edx_files=download_from_my_github(edx_files)
edx=readRDS_split(edx_files)

validation_file='validation.rds'
if(!'validation.rds' %in% list.files()) validation_file=download_from_my_github(validation_file)
validation=readRDS(validation_file)

# Convert to list and do is.na function
cat('edx missing values =',mean(edx %>% as.list() %>% sapply(is.na)),'\n')
```

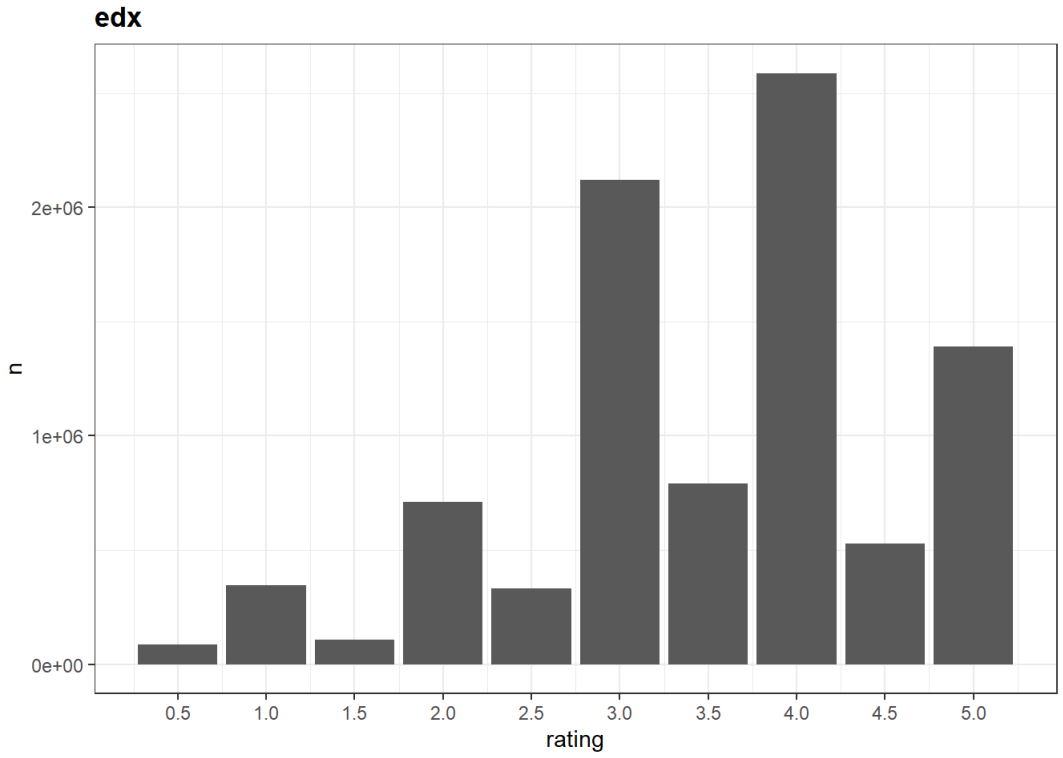
```
## edx missing values = 0
```

```
cat('validation missing values =',mean(validation %>% as.list() %>% sapply(is.na)))
```

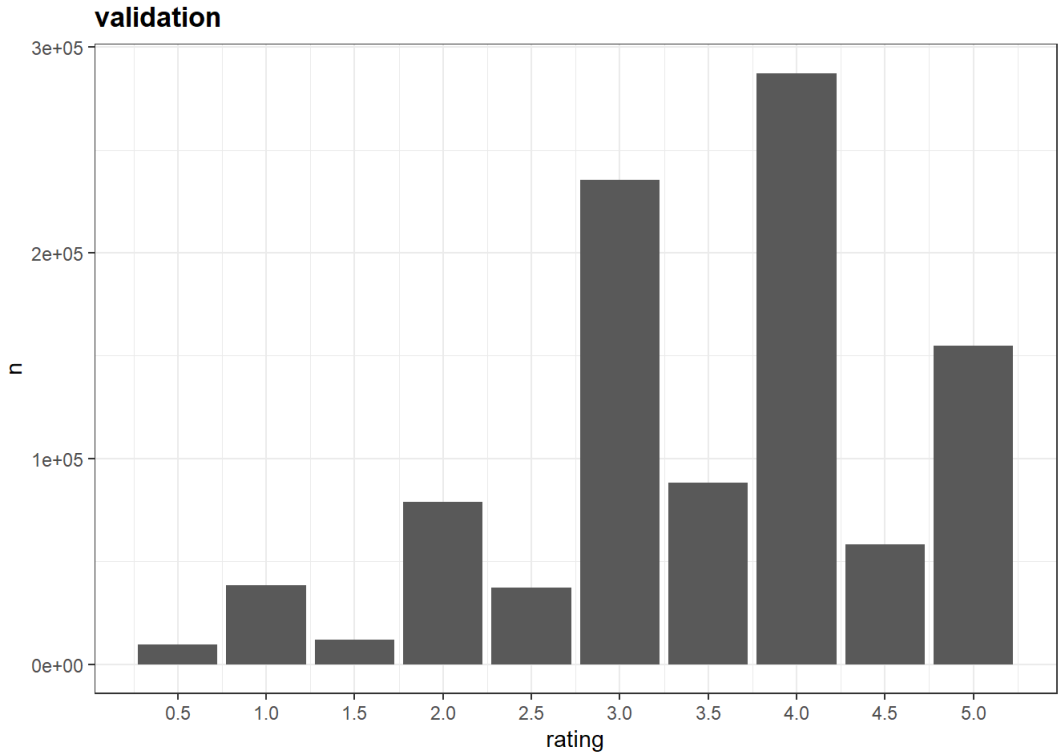
```
## validation missing values = 0
```

Let's check for the ratings distribution. There were more rounded rating instead of half rating. But, this was because the half rating was stacked with rounded ones.

```
# Plot frequency per rating number
edx %>%
  group_by(rating) %>%
  summarize(n=n()) %>%
  ggplot(aes(rating,n)) +
  geom_col() +
  scale_x_continuous(breaks=seq(0,5,0.5)) +
  ggtitle('edx')
```




```
validation %>%
  group_by(rating) %>%
  summarize(n=n()) %>%
  ggplot(aes(rating,n)) +
  geom_col() +
  scale_x_continuous(breaks=seq(0,5,0.5)) +
  ggtitle('validation')
```



For data preprocessing, we build functions with two stages:

1. We created unique ID for each rating and select only that ID, movie ID, user ID, and rating for light computation.
2. We computed days since user first rating, since movie first rating, and number of users rating each movie.

This was the finalized preprocessing methods that were concluded after exploratory data analysis below.

Datasets

To develop prediction models, we only used edx as training set; thus, an internal validation index was maded to subset this training set to evaluate predictive performances without using validation set. We did not make a different dataset, but, only use the index if we need an internal validation set.

```
cat('########### Preprocess train set ###########','\n')
```

```
## ########### Preprocess train set ###########
```

```
train_set=edx %>% data_preprocessing$rating()
```

```
## Create IDs for every combination of userId and movieId
## Select only IDs, movieId, userId, and rating
## Done.
```

```
cat('')

# suppresWarnings(set.seed(33)) # if using R 3.5 or earlier
suppressWarnings(set.seed(33,sample.kind = 'Rounding')) # if using R 3.6 or later

# Create validation index (this is NOT from test_set or validation, BUT from train_set or edx)
val_index=
  train_set %>%
  pull(rating) %>%
  createDataPartition(times=10,p=0.1,list=F)
```

We also build a function to prevent the same movie or user between the training and internal validation.

Add more features by second data preprocessing.

```
# Preprocess user-centered data
cat('########### Preprocess train set ###########','\n')
```

```
## ########### Preprocess train set ###########
```

```
train_set=
  train_set %>%
  rownames_to_column(var='id') %>%
  left_join(
    edx %>%
      data_preprocessing$user_movie() %>%
      rownames_to_column(var='id'),
    by='id'
  )
```

```
## Create IDs for every combination of userId and movieId
## Compute number of days since user first rating
## Compute number of days since movie first rating
## Compute cumulative number of users rating each movie every day since movie first rating
## Done.
```

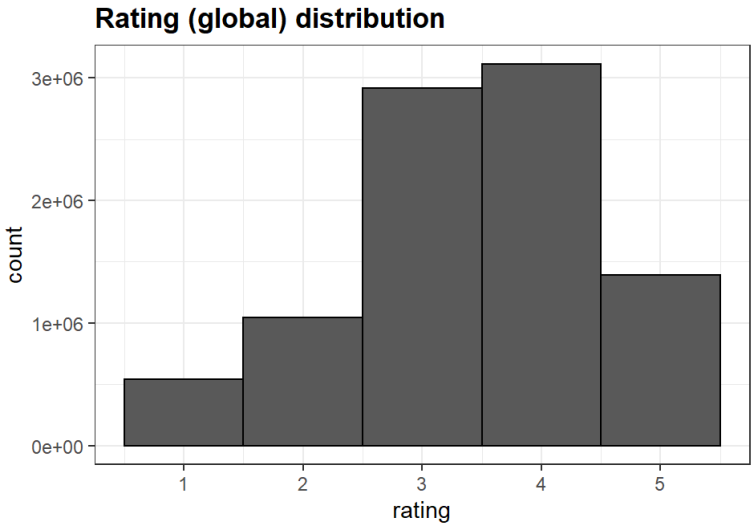
```
cat('')
```

Exploratory data analysis

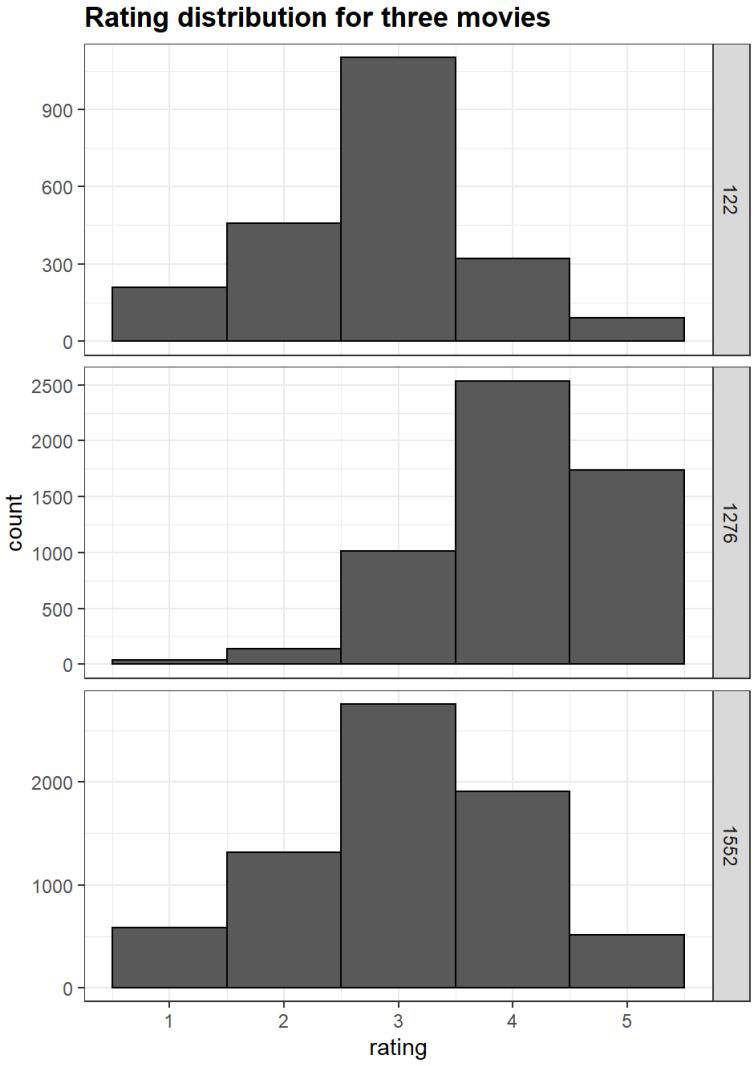
We will only explore edx that is already rename as train_set. By summary table, we can see at glance that number of days and number of movie or the rating is not scaled. We will scale them later.

```
## # A tibble: 8 x 2
##   variable                                              value
##   <chr>                                                 <chr>
## 1 rating (mean)                                         3.51
## 2 rating (SD)                                           1.06
## 3 latest number of days since user first rating (mean)  505
## 4 latest number of days since user first rating (SD)    856
## 5 latest number of days since movie first rating (mean) 3,696
## 6 latest number of days since movie first rating (SD)   1,068
## 7 latest number of movie rating (mean)                  6,787
## 8 latest number of movie rating(SD)                     6,854
```

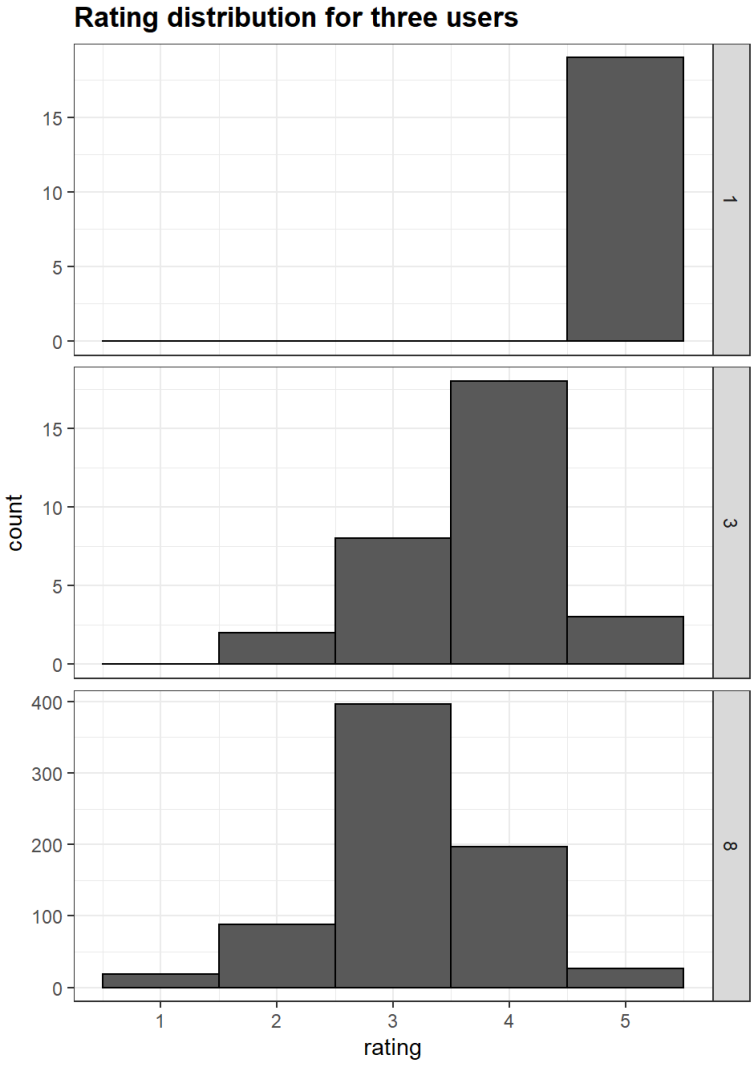
Here is the rating (global) average.



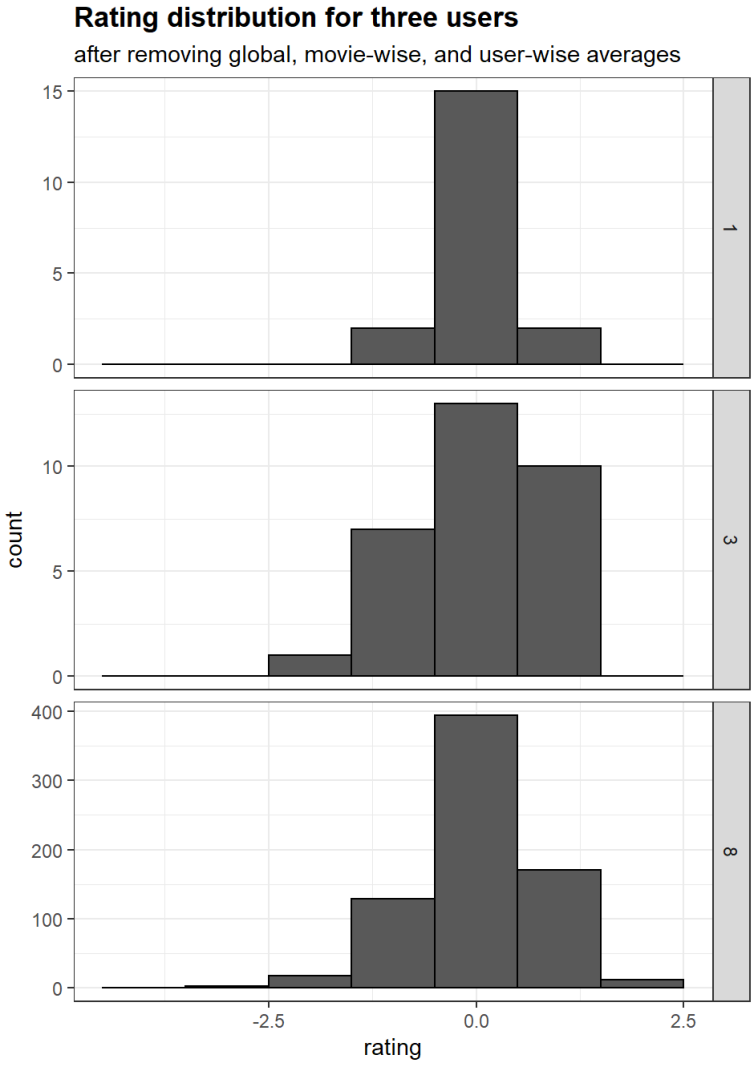
We can see below that the average rating is different among movies. Note, the y scale is free up to each movie. We just want to show the different in the distribution by rating number (axis x).



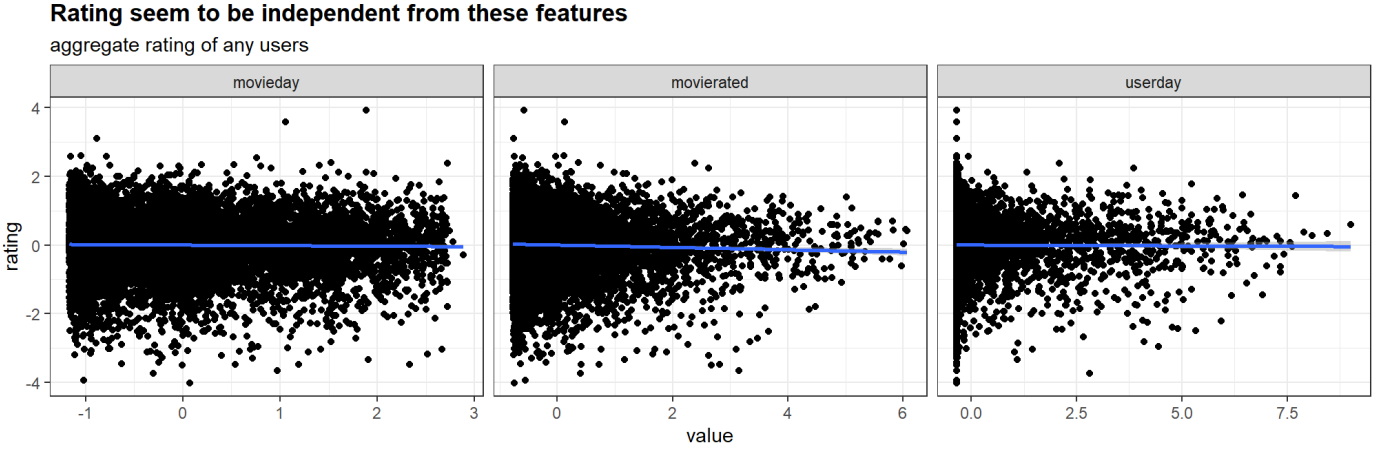
The average rating for each user was also different.



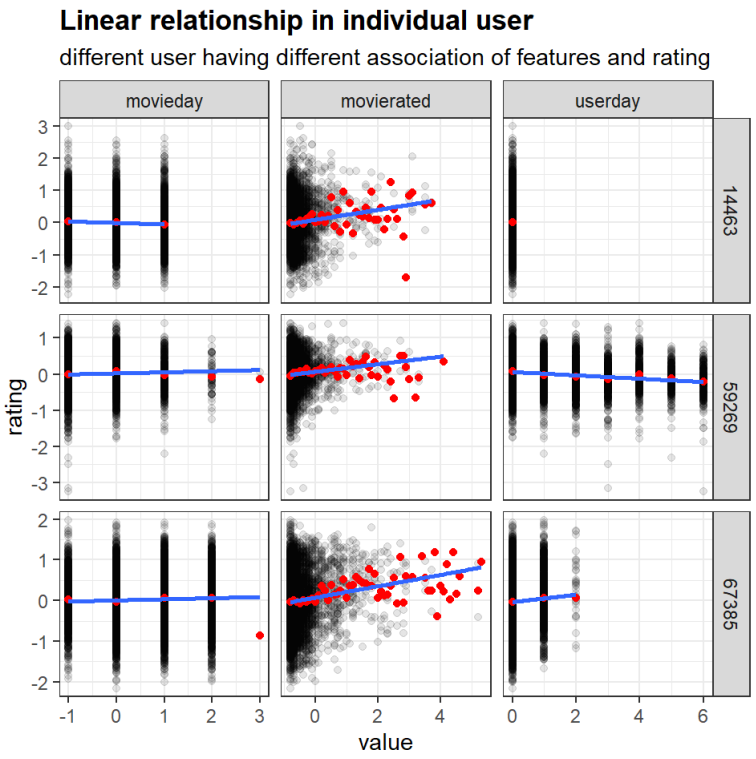
If we removed global, movie-wise, and user-wise average; thus, we got similar distribution that is more predictable.



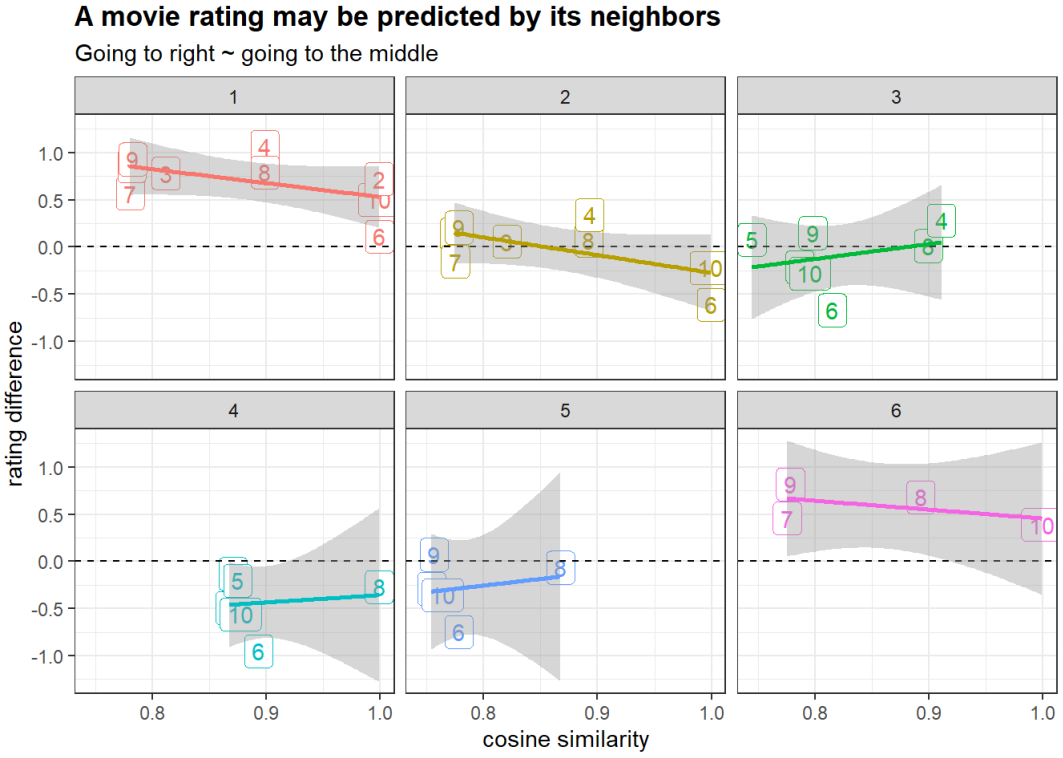
We also scaled the number of days since user/movie first rating and the number of movie rated. Note, if we visualized rating against these features, we could not see linear relationship.



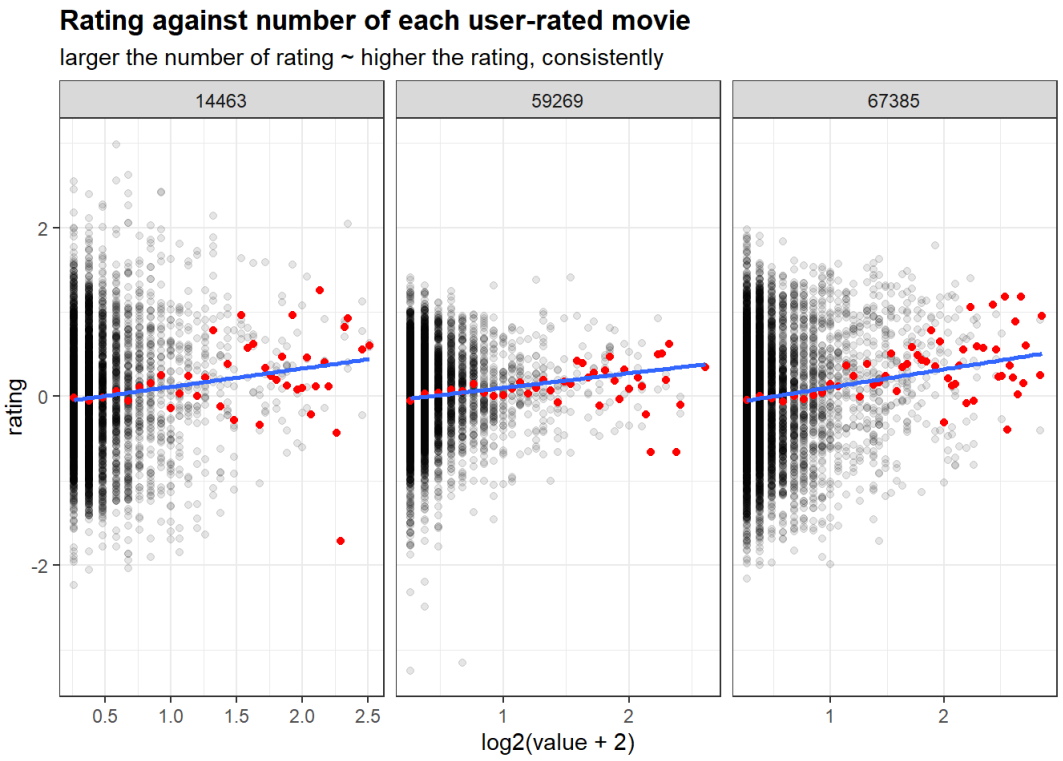
But, if we visualized it for each user (e.g., we take 3 users), then can see clear linear relationship. This means we need to make the linear regression for each user if using these features (after removing biases). In the figure below, the black and red points are individual and average rating for each rounded feature. This visualization was intended to demonstrate conditional probabilities between each feature and rating.



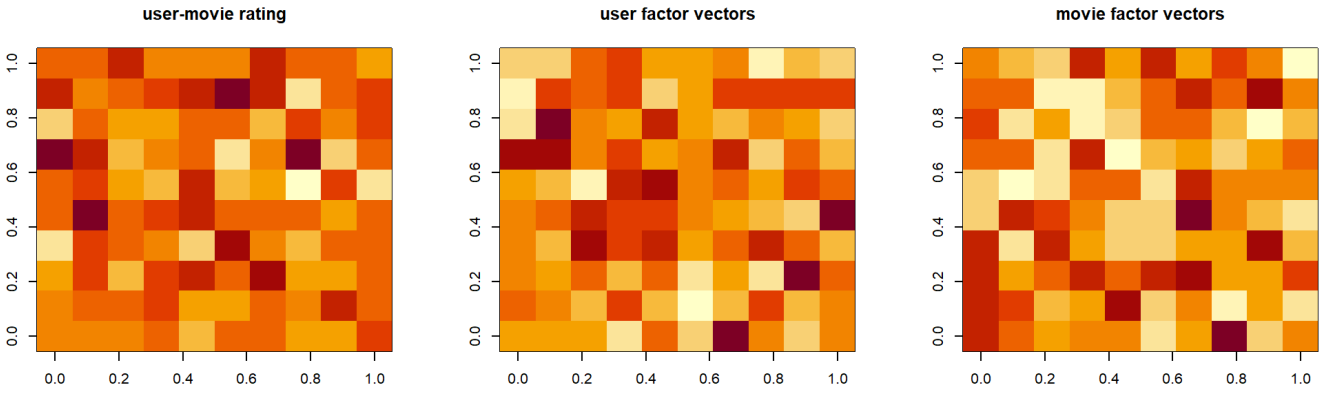
Let's take some movies that have sufficient ratings (but not too large, just for samples). We created movie-rating matrix to compute cosine similarity matrix. Then, we depicted rating difference of a movie compared to others that were similar to it. We could see below that more similar 'neighbor' movie tend to have smaller difference of rating.



Since above we filter movie based on number of rating, we need to find out how this number may affect the rating for each user. Say later we want to adjust user effect on association of a movie and its neighbors' rating. We could see the effect is consistent. We would like to adjust the user effect if many of the neighbors of the movie was also rated by the user. This number of ratings per user may implicit user likes of the neighbor movie such the user may rated more or less (e.g. the movie is not as good the similar one the user expect).



We also may represent new features using matrix factorization by standard singular value decomposition (SSVD). We selected users and movies that have many ratings as sample. Below is the represented features.



If we do inference using linear regression, we can find three significant vectors. This means we may use matrix factorization to improve our prediction.

```
## # A tibble: 4 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    3.35      0.176     19.1  1.77e-30
## 2 ssvd_u3        0.767     0.297      2.58 1.18e- 2
## 3 ssvd_u6        0.683     0.332      2.06 4.30e- 2
## 4 ssvd_u8        0.788     0.304      2.59 1.16e- 2
```

Model development

Based on visualization and insights gained above, we developed a prediction model with boosting method. This means we will build multiple models sequentially whereas a model would predict error or residual of prediction from previous model.

1. Baseline model should use global, movie-wise, and user-wise average
2. The residuals would be predicted by number of days since user/movie first rating and number of rating per movie
3. Residuals of k similar movies based on residual similarity were used to predict a movie
4. Then we will used number of available residuals that implicit a user likes similar movies
5. Using standard SVD, we got either user or movie factor vectors as dot products for predicting the residual
6. Finally, we ensemble 5 models above either by majority voting (rating being rounded), averaging, or stacking into a linear model

Baseline model

To compute the average, we do regularization by multiple lambda as denominator with sample size. We identified the lambda using 10-fold cross validation (shuffle split). This process need 40 lambda x 10 folds = 400 loops; thus we used parallelism. Beware if you want to run this code since we takes of processing core leaving only 1 core.

The reasons for using global, movie-wise, and user-wise average to predict rating:

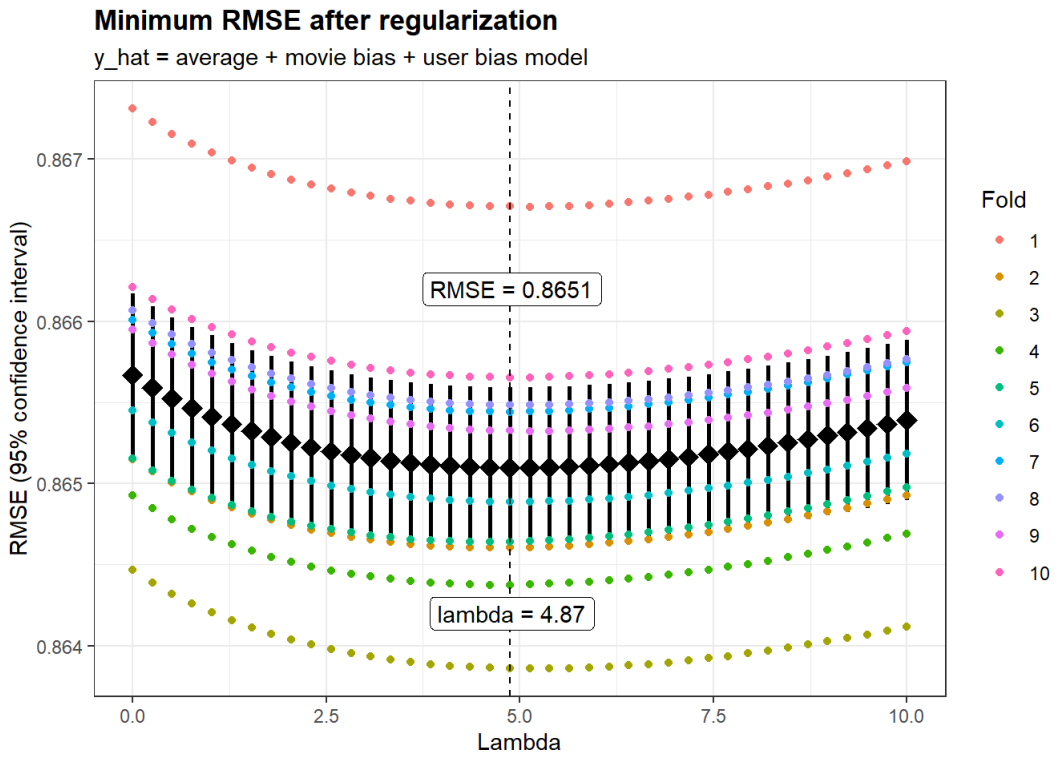
1. Global average rating estimated how arbitrary user rating in the population.
2. Some movie may be rated higher because of the artist, based on famous comic books, etc., or may be lower somehow
3. Some users rated higher than the population, even, some others would rate differently to common people

```
##
## Regularization of average + movie bias + user bias model for each fold by parallel computing
##
## Started: 2020-09-19 22:40:56
##
##   |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=24m 14s
##
## End: 2020-09-19 23:05:19
##
##
```

We save the regularization results into reg.rds. We commented to run off the saving code. Instead we make it to read the saved files

```
reg_files=
  seq(10) %>%
  lapply(function(x)paste0('reg',x,'.',1:2,'.rds'))

# if(!all(unlist(reg_files) %in% list.files())){
#   reg_files=
#     seq(length(reg)) %>%
#     lapply(function(x)saveRDS_split_l(reg[[x]],paste0('reg',x,'.'),2))
# }

if(!all(unlist(reg_files) %in% list.files())){
  reg_files=
    reg_files %>%
    lapply(function(x)download_from_my_github(x))
}

reg=
  reg_files %>%
  lapply(function(x)readRDS_split_l(x))
```

The regularization is plotted below. We can see the lowest RMSE by cross validation using the best lambda. By different folds, we may estimate the confidence interval that will fall on the true RMSE, including later using validation set. Seeing the interval, we may not achieve the RMSE goal by baseline model except very few chance.



The best lambda was used to compute the averages and the baseline model is computed below.

```
mu=
  train_set %>%
  summarize(mu=mean(rating))

movie_b=
  train_set %>%
  cbind(mu) %>%
  group_by(movieId) %>%
  summarize(movie_b=sum(rating-mu)/(n()+reg_dt$best_lambda[1]))

user_b=
  train_set %>%
  cbind(mu) %>%
  left_join(movie_b,by='movieId') %>%
  group_by(userId) %>%
  summarize(user_b=sum(rating-(mu+movie_b))/(n()+reg_dt$best_lambda[1]))

rmse=list()
rmse[[1]]=
  reg_dt %>%
  filter(lambda==reg_dt$best_lambda[1]) %>%
  select(mean_RMSE,lb_RMSE,ub_RMSE) %>%
  .[1,] %>%
  mutate(model='Baseline model')
```

The baseline model is saved as RDS files.

```
saveRDS(mu,'mu.rds')
saveRDS(movie_b,'movie_b.rds')
saveRDS(user_b,'user_b.rds')
```

User-centered model

For second model, we developed it to predict the residual. Below the training set is updated for the residuals.

```
train_set=
  train_set %>%
  cbind(mu) %>%
  left_join(movie_b,by='movieId') %>%
  left_join(user_b,by='userId') %>%
  mutate(residual=rating-(mu+movie_b+user_b))
```

We will also used movie bias as one more additional features; thus this model had four features:

1. Number of days since user first rating, maybe one give lower rating through time or not
2. Number of days since movie first rating, older movies may be rated higher or not by a user
3. Number of movie ratings, in which some users tend to rate higher for more-rated movies or a person don't like being common
4. Movie overall rating (average) may cause a user rate higher, or lower since he/she may not like being common

So there would be one linear model with four features for each user. Since user numbers is quite large (even more than the movies), this would like to give big impact to improve the prediction. Below you will see the results.

```
tidy_strat_lm=function(i,formula,data,strata){

  result=
    data %>%
    rename_at(strata,function(column)'strata')

  result2=
    result %>%
    filter(strata==i) %>%
    select_at(c(
      as.character(formula)[2],
      str_split(as.character(formula)[3],' \\+ ')[[1]]
    )) %>%
    setNames(c('y',colnames(.)[-1]))

  # suppressWarnings(set.seed(33)) # if using R 3.5 or earlier
  suppressWarnings(set.seed(33,sample.kind = 'Rounding')) # if using R 3.6 or later
  result3=
    train(
      y~.,data=result2,method='glmnet',
      trControl=trainControl('cv',number=2),
      tuneGrid=data.frame(alpha=0,lambda=10^seq(-3,3,len=5))
    )

  result4=
    coef(result3$finalModel,result3$bestTune$lambda) %>%
    as.matrix()
  rm(result3)

  data.frame(
      strata=as(i,class(result$strata)),
      term=rownames(result4),
      estimate=result4[,1]
    ) %>%
    rename_at('strata',function(column)strata)
}
```

We used caret and glmnet package to do ridge regression (L2-norm regularization). The reason why we used this regularization was because the weights of the features were quite balanced to each other as shown by preliminary run using simple linear model. The reasoning for each feature is also quite make sense; thus, no need to remove any of them. Ridge regression did not removed features. Instead, the other kind regularized linear regression such lasso or elastic net doing this. Usually, L2-norm regularization is quite effective to prevent overfitting and improve model generalizability for future, unobserved (yet) data.

As you can see in the code above, ridge regression using zero alpha. We then tuned the lambda. We only used 2-fold cross validation for this. Because this would be computationally expensive (the longest one), we also applied parallelism. Beware to run the code below

```
# {
#   cat('Conduct ridge regression for each user by parallel computing\n')
#   cat('Started:',as.character(now()),'\n')
#   cl=makeCluster(detectCores()-1)
#   clusterEvalQ(cl,{
#     library('tidyverse')
#     library('pbapply')
#     library('caret')
#     library('glmnet')
#   })
#
#     user_ridge=
#       train_set %>%
#       pull(userId) %>%
#       .[!duplicated(.)] %>%
#       pblapply(
#         FUN=tidy_strat_lm,
#         formula=residual~userday_z+movieday_z+movierated_z+movie_b_z,
#         data=
#           train_set %>%
#           mutate(
#             userday_z=scale(userday),
#             movieday_z=scale(movieday),
#             movierated_z=scale(movierated),
#             movie_b_z=scale(movie_b)
#           ),
#         strata='userId',
#         cl=cl
#       )
#
#   stopCluster(cl)
#   rm(cl)
#   gc()
#   cat('End:',as.character(now()))
# }

cat('
Conduct ridge regression for each user by parallel computing\n
Started: 2020-09-19 23:09:21 \n
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05h 26m 25s\n
End: 2020-09-20 04:35:54\n
    ')
```

```
##
## Conduct ridge regression for each user by parallel computing
##
## Started: 2020-09-19 23:09:21
##
##   |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05h 26m 25s
##
## End: 2020-09-20 04:35:54
##
##
```

Let save the most expensive variable :D into RDS files. You can just load the previous one, instead running the code above.

```
# if(!'user_ridge.rds' %in% list.files()) saveRDS(user_ridge,'user_ridge.rds')
user_ridge_file='user_ridge.rds'
if(!'user_ridge.rds' %in% list.files()) user_ridge_file=download_from_my_github(user_ridge_file)
user_ridge=readRDS(user_ridge_file)
```

Below we tidy up the ridge regression results, incorporate it into training set, then estimate the RMSE using 10-fold cross validation. We just used the weights and computed manually instead of using 'predict' function. Therefore, just by subsetting the training set using the index, we can used the internally-validated prediction (y_hat) to estimate the RMSE.

Also save average and standard deviation since we scale the features. These numbers would be used for validation set. We would centerize the validation residuals (after bias removed by baseline model) using this average and scale it using the training standard deviation. Finally we updated the residual to boost using the next model.

```
# Tidy up user ridge regression table
user_ridge=
  user_ridge %>%
  do.call(rbind,.) %>%
  mutate(
    term=
      term %>%
      str_to_lower() %>%
      str_replace('\\(','') %>%
      str_replace('\\)','') %>%
      str_replace('ib','b') %>%
      str_replace('\\^','') %>%
      str_replace_all('\\`','')
  ) %>%
  select(userId,term,estimate) %>%
  spread(term,estimate,fill=0) %>%
  rename_at(colnames(.)[-1],function(x)paste0('ur_',x))

# Incorporate additional features and the weights
train_set=
  train_set %>%
  select(-residual) %>%
  mutate(
    userday_z=scale(userday),
    movieday_z=scale(movieday),
    movierated_z=scale(movierated),
    movie_b_z=scale(movie_b)
  ) %>%
  left_join(user_ridge,by='userId') %>%
  mutate(
    y_hat=
      mu+movie_b+user_b
      +ur_intercept
      +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
      +ur_movierated_z*movierated_z+ur_userday_z*userday_z
  ) %>%
  as.data.table()

# Estimate RMSE using the best user-centered model with 10-fold cross validation
rmse[[2]]=
  val_index %>%
  as.data.table() %>%
  gather(fold,seq) %>%
  left_join(
    train_set %>%
      mutate(seq=seq(nrow(.))) %>%
      select(seq,y_hat,rating),
    by='seq'
  ) %>%
  group_by(fold) %>%
  summarize(RMSE=RMSE(y_hat,rating)) %>%
  summarize(
    mean_RMSE=mean(RMSE),
    sd_RMSE=sd(RMSE),
    n_RMSE=n(),
    lb_RMSE=mean_RMSE-qnorm(0.975)*sd_RMSE/sqrt(n_RMSE),
    ub_RMSE=mean_RMSE+qnorm(0.975)*sd_RMSE/sqrt(n_RMSE)
  ) %>%
  select(mean_RMSE,lb_RMSE,ub_RMSE) %>%
  mutate(model='Baseline + user-centered model')

# Save average and standard deviation of the features of training set to scale those of validation set later
ur_mus=
  train_set %>%
  select(movie_b,movieday,movierated,userday) %>%
  cbind(
    summarize_all(.,mean) %>% rename_all(function(colname)paste0('m_',colname)),
    summarize_all(.,sd) %>% rename_all(function(colname)paste0('s_',colname))
  ) %>%
  select(-movie_b,-movieday,-movierated,-userday) %>%
  .[1,]
saveRDS(ur_mus,'ur_mus.rds')
```

```
# Update the residual
train_set=
  train_set %>%
  cbind(ur_mus) %>%
  mutate(residual=rating-y_hat) %>%
  select(-movie_b_z,-movieday_z,-movierated_z,-userday_z,-y_hat)
```

Movie k-neighbors model

We had exploited the user part, but not yet the movie. Neighborhood model is widely used in the NetFlix challenge because of practicality. This means every time a new movie having sufficient number of ratings. One can compute its similarity to all available movies, then use k most similar movies that a user rated to predict how that user rated the new movie. There is no need to retrain all models.

However, since we had approximately 7 to 10 thousand movies, then we need to square this number to know how large the similarity matrix. The user similarity matrix is squares of the movie number. This is why we did not developed user k-neighbors model, which has number about 6 times larger than the movie numbers.

In the code below, we did some tricks:

1. First, we scale it before filtering because we the predicted residual should be equivalent among movies
2. We filter only movie with >50 ratings and then user with >50 rating. It still be a large matrix, though.
3. Since this should be movie-user sparse matrix, we scale down the 'width' into half-rounded rating group, instead of all user rating.
4. We computed the average for each group.
5. Therefore, the similarity was based on distribution of ratings among users for each movie.

By these tricks, we could reduce the very much computational burden.

```
# Get average residual per movie and group of rounded residual (no decimal)
m_res=
  train_set %>%
  group_by(userId) %>%
  mutate(residual=scale(residual)) %>%
  ungroup() %>%
  group_by(movieId) %>%
  mutate(residual=scale(residual)) %>%
  ungroup() %>%
  group_by(movieId) %>%
  filter(n()>50) %>%
  ungroup() %>%
  group_by(userId) %>%
  filter(n()>50) %>%
  ungroup() %>%
  select(movieId,userId,residual) %>%
  mutate(res_group=round(residual)) %>%
  group_by(movieId,res_group) %>%
  summarize(residual=mean(residual))

# # Create a movie-movie combination table
# m_sim=
#   m_res$movieId %>%
#   .[!duplicated(.)] %>%
#   pblapply(X=seq(length(.)-1),movieId=.,FUN=function(X,movieId)
#     data.table(movieId1=movieId[X],movieId2=movieId[(X+1):length(movieId)])
#   ) %>%
#   do.call(rbind,.)

cat('|++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05s  ')
```

```
## |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05s
```

We also need to tradeoff among time, memory, CPU, and other procesess. Somehow we used chunked codes below to compute similarity, instead using code pipeline as most codes in this markdown. We clear the memory for each step. We computed cosine similarity as the metric, manually. We have tried even multiplication using matrix to expect a cheaper computation, but it did not work. A data.table operation seem cheaper. Here is the equation for cosine similarity:

```
 sum(AxB)/sqrt(sum(A^2)*sum(B^2))
```

where A and B were same-length vectors of averages from all rating groups.

Value of 1 denoted perfect similarity, value of 0 denoted no similarity, and value of -1 denoted the opposite (a very different). It is actually using somekind of angle, just like cosine things.

```
# cat('Started:',as.character(now()),'\n')
#
# # Join average residual of each movie and the group for each movie in pair,
# # then remove residual table and clear memory garbage by gc() (also do this for next steps)
# m_sim=
#   m_sim %>%
#   left_join(rename_all(m_res,function(column)paste0(column,'1')),by='movieId1') %>%
#   left_join(rename_all(m_res,function(column)paste0(column,'2')),by='movieId2')
# gc()
# rm(m_res)
#
# # Unselect residual group name freeing up memory for next step
# m_sim=
#   m_sim %>%
#   select(-res_group1,-res_group2)
# gc()
#
# # Split paired residual table into two for cheaper computation each run
# set=split(1:nrow(m_sim),1:2)
# m_sim1=m_sim[set[[1]],]
# m_sim2=m_sim[set[[2]],]
# rm(m_sim,set)
# gc()
#
# # Multiply residual to each other and to themselves (squares) for each table partition
# m_sim1=
#   m_sim1 %>%
#   mutate(
#     AB=residual1*residual2,
#     A2=residual1^2,
#     B2=residual2^2
#   ) %>%
#   select(-residual1,-residual2)
# gc()
#
# m_sim2=
#   m_sim2 %>%
#   mutate(
#     AB=residual1*residual2,
#     A2=residual1^2,
#     B2=residual2^2
#   ) %>%
#   select(-residual1,-residual2)
# gc()
#
# # Grouped multiple-row residuals of all groups, by each movie pair for each table partition
# m_sim1=
#   m_sim1 %>%
#   group_by(movieId1,movieId2)
# gc()
#
# m_sim2=
#   m_sim2 %>%
#   group_by(movieId1,movieId2)
# gc()
#
# # Sum all results of between- and self-movie multiplication by each movie pair for each table partition
# m_sim1=
#   m_sim1 %>%
#   summarize(
#     sum_AB=sum(AB),
#     sum_A2=sum(A2),
#     sum_B2=sum(B2)
#   )
# gc()
#
# m_sim2=
#   m_sim2 %>%
#   summarize(
#     sum_AB=sum(AB),
#     sum_A2=sum(A2),
#     sum_B2=sum(B2)
```

```
#   )
# gc()
#
# # Bind all table partitions
# m_sim=rbind(m_sim1,m_sim2)
# rm(m_sim1,m_sim2)
#
# # Sum of the total multiplication from two table partitions
# m_sim=
#   m_sim %>%
#   group_by(movieId1,movieId2) %>%
#   summarize(
#     sum_AB=sum(sum_AB),
#     sum_A2=sum(sum_A2),
#     sum_B2=sum(sum_B2)
#   )
# gc()
#
# cat('End:',as.character(now()))

cat('
Started: 2020-09-20 08:04:24 \n
            used    (Mb)  gc trigger    (Mb)   max used    (Mb)\n
Ncells   12965541   692.5    69136052  3692.3  263733105 14084.9\n
Vcells 6605857393 50398.7 11233526534 85705.1 9069728953 69196.6\n
            used    (Mb)  gc trigger    (Mb)   max used    (Mb)\n
Ncells   12958474   692.1    55308842  2953.9  263733105 14084.9\n
Vcells 4141624758 31598.1 11233526534 85705.1 9069728953 69196.6\n
            used    (Mb)  gc trigger    (Mb)   max used    (Mb)\n
Ncells   12958490   692.1    44247074  2363.1  263733105 14084.9\n
Vcells 4141626799 31598.2 11233526534 85705.1 9069764213 69196.9\n
            used    (Mb)  gc trigger    (Mb)   max used    (Mb)\n
Ncells   12958491   692.1    35397660  1890.5  263733105 14084.9\n
Vcells 4757643377 36298.0 11233526534 85705.1 9069764213 69196.9\n
            used    (Mb)  gc trigger    (Mb)   max used    (Mb)\n
Ncells   12958492   692.1    35397660  1890.5  263733105 14084.9\n
Vcells 5373657882 40997.8 11233526534 85705.1 9069764213 69196.9\n
            used    (Mb)  gc trigger    (Mb)   max used    (Mb)\n
Ncells   37714670  2014.2    57188134  3054.2  263733105 14084.9\n
Vcells 5815035295 44365.2 11233526534 85705.1 9069764213 69196.9\n
            used    (Mb)  gc trigger    (Mb)   max used    (Mb)\n
Ncells   62470852  3336.4   109516542  5848.9  263733105 14084.9\n
Vcells 6256414525 47732.7 11233526534 85705.1 9069764213 69196.9\n
            used    (Mb)  gc trigger    (Mb)   max used    (Mb)\n
Ncells   37721721  2014.6   316181388 16886.0  305537481 16317.5\n
Vcells 3462384004 26415.9 10784249472 82277.3 9069764213 69196.9\n
           used   (Mb) gc trigger    (Mb)   max used    (Mb)\n
Ncells  12972585  692.9  252945111 13508.8  316181388 16886.0\n
Vcells 668353733 5099.2 8627399578 65821.9 9069764213 69196.9\n
           used   (Mb) gc trigger    (Mb)   max used    (Mb)\n
Ncells  12965516  692.5  202356089 10807.0  316181388 16886.0\n
Vcells 556938556 4249.2 6901919663 52657.5 9069764213 69196.9\n
End: 2020-09-20 08:18:39\n
    ')
```

```
##
## Started: 2020-09-20 08:04:24
##
##              used    (Mb)  gc trigger    (Mb)   max used    (Mb)
##
## Ncells   12965541   692.5    69136052  3692.3  263733105 14084.9
##
## Vcells 6605857393 50398.7 11233526534 85705.1 9069728953 69196.6
##
##              used    (Mb)  gc trigger    (Mb)   max used    (Mb)
##
## Ncells   12958474   692.1    55308842  2953.9  263733105 14084.9
##
## Vcells 4141624758 31598.1 11233526534 85705.1 9069728953 69196.6
##
##              used    (Mb)  gc trigger    (Mb)   max used    (Mb)
##
## Ncells   12958490   692.1    44247074  2363.1  263733105 14084.9
##
## Vcells 4141626799 31598.2 11233526534 85705.1 9069764213 69196.9
##
##              used    (Mb)  gc trigger    (Mb)   max used    (Mb)
##
## Ncells   12958491   692.1    35397660  1890.5  263733105 14084.9
##
## Vcells 4757643377 36298.0 11233526534 85705.1 9069764213 69196.9
##
##              used    (Mb)  gc trigger    (Mb)   max used    (Mb)
##
## Ncells   12958492   692.1    35397660  1890.5  263733105 14084.9
##
## Vcells 5373657882 40997.8 11233526534 85705.1 9069764213 69196.9
##
##              used    (Mb)  gc trigger    (Mb)   max used    (Mb)
##
## Ncells   37714670  2014.2    57188134  3054.2  263733105 14084.9
##
## Vcells 5815035295 44365.2 11233526534 85705.1 9069764213 69196.9
##
##              used    (Mb)  gc trigger    (Mb)   max used    (Mb)
##
## Ncells   62470852  3336.4   109516542  5848.9  263733105 14084.9
##
## Vcells 6256414525 47732.7 11233526534 85705.1 9069764213 69196.9
##
##              used    (Mb)  gc trigger    (Mb)   max used    (Mb)
##
## Ncells   37721721  2014.6   316181388 16886.0  305537481 16317.5
##
## Vcells 3462384004 26415.9 10784249472 82277.3 9069764213 69196.9
##
##             used   (Mb) gc trigger    (Mb)   max used    (Mb)
##
## Ncells  12972585  692.9  252945111 13508.8  316181388 16886.0
##
## Vcells 668353733 5099.2 8627399578 65821.9 9069764213 69196.9
##
##             used   (Mb) gc trigger    (Mb)   max used    (Mb)
##
## Ncells  12965516  692.5  202356089 10807.0  316181388 16886.0
##
## Vcells 556938556 4249.2 6901919663 52657.5 9069764213 69196.9
##
## End: 2020-09-20 08:18:39
##
##
```

After the code above , we can be more relax. Below is the last steps of computing cosine similarity. Let save the results.

```
# # Multiply sum of squares for each movie pair
# m_sim=
#   m_sim %>%
#   mutate(ss_A2B2=sum_A2*sum_B2) %>%
#   select(-sum_A2,-sum_B2)
#
# # Square root the sum of squares for each movie pair
# m_sim=
#   m_sim %>%
#   mutate(sqrt_ss_A2B2=sqrt(ss_A2B2)) %>%
#   select(-ss_A2B2)
#
# # Get cosine similarity by dividing sum of between-movie residual multiplication with the root of sum of squares
# m_sim=
#   m_sim %>%
#   mutate(cosine_sim=sum_AB/sqrt_ss_A2B2) %>%
#   select(-sum_AB,-sqrt_ss_A2B2)

# Save movie-movie cosine similarity
m_sim_files=paste0('m_sim',1:11,'.rds')
# if(!all(m_sim_files %in% list.files())) m_sim_files=saveRDS_split(m_sim,'m_sim',11)
if(!all(m_sim_files %in% list.files())) m_sim_files=download_from_my_github(m_sim_files)
m_sim=readRDS_split(m_sim_files)
```

After getting the similarity matrix, we can find out the k most similar movies for each movie. Below we constructed (again) the training subset but this time we did not group the ratings. Instead, we just compute sum and the number of instances. We will use these to compute average but after determining how much the k.

For example, we tried k=5 then we summed of sums of residuals from the 5 most similar movies then divided by sums of the number of instances from those movies. This time, we used validation IDs instead of index since need to be specific when taking neighbors.

We would find k that minimizes the RMSE of (internal) validation set. For this part, we just used mean or average of neighbor ratings to predict a movie rating. But, below these parts, we would weighted each neighbors. To make it clear, there is two phases of neighborhood modeling here:

1. Use average ratings of k neighbors to find the best k similar neighbors
2. Use linear regression with lasso regularization to weight which neighbors should contributed more to the prediction

This is because some movies can be predicted by 1-2 only, while others may not require any neighbors, or even some movies required a lot neighbors. For the first phase, we set the same number of neighbors, but we wanted to know how much the initial k. Later, in lasso regularization, as described previously, we would remove some features, which are the neighbor ratings. This is why we choose lasso instead of ridge regression. We did not want elastic net since it required more steps that are still not be efficient for this part.

```
# Reconstruct residual table but no residual-value grouping.
# sum and count all residuals for each movie to compute the average or k-mean later (depending k most similar neighbor
s)
m_res=
  train_set %>%
  group_by(movieId) %>%
  filter(n()>50) %>%
  ungroup() %>%
  group_by(userId) %>%
  filter(n()>50) %>%
  ungroup() %>%
  select(movieId,userId,residual) %>%
  group_by(movieId) %>%
  summarize(
    sum_residual=sum(residual),
    n_residual=n()
  ) %>%
  rename(movieId2=movieId)

# Get IDs of validation index (of edx) among which have no common movie-user with the training set (for getting k-mea
n)
val_id=
  seq(1,dim(val_index)[2]) %>%
  pbsapply(function(X)diff_mu(train_set,val_index[,X])$id) %>%
  `dimnames<-`(dimnames(val_index))

# # Construct cross-validation set with pre-computed baseline and user-centered prediction
# m_cv=
#   train_set %>%
#   mutate(
#     userday_z=(userday-m_userday)/s_userday,
#     movieday_z=(movieday-m_movieday)/s_movieday,
#     movierated_z=(movierated-m_movierated)/s_movierated,
#     movie_b_z=(movie_b-m_movie_b)/s_movie_b
#   ) %>%
#   mutate(
#     y_hat=
#       mu+movie_b+user_b
#       +ur_intercept
#       +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
#       +ur_movierated_z*movierated_z+ur_userday_z*userday_z
#   ) %>%
#   mutate(seq=seq(nrow(.))) %>%
#   select(id,movieId,seq,y_hat,rating) %>%
#   as.data.table()
m_cv_files=paste0('m_cv',1:7,'.rds')
# if(!all(m_cv_files %in% list.files())) m_cv_files=saveRDS_split(m_cv,'m_cv',7)
if(!all(m_cv_files %in% list.files())) m_cv_files=download_from_my_github(m_cv_files)
m_cv=readRDS_split(m_cv_files)

cat('|++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=06s  ')
```

```
## |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=06s
```

We just used k=3 to k=21 as the candidates. Say larger k can minimize the RMSE, however, we might not be able to do the computation. Meanwhile, later we would do lasso regression. The most expensive part was to construct tables of thousand rows or more for the predicted movie, multiplied with the number of neighbors. We may not achieve the most optimal k here, but we can pursue it later using lasso and the next models.

Please kindly beware to run the code below. It also applied paralellism.

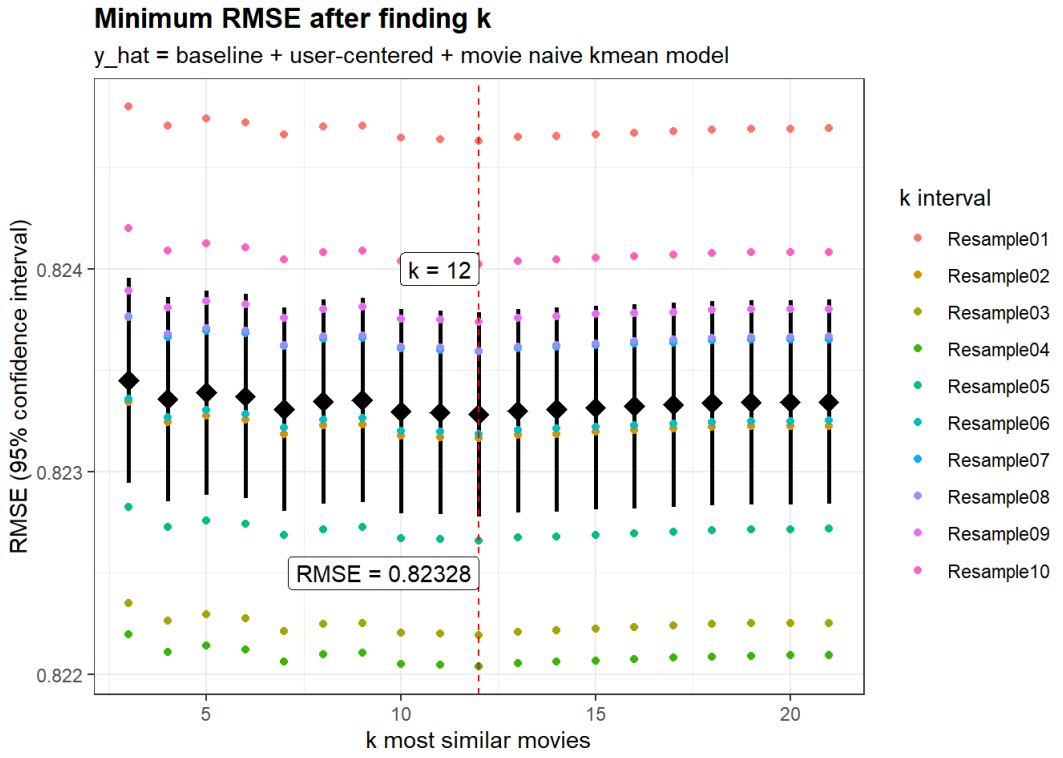
```
# {
#   cat('Find k most similar neighbors of which average minimize RMSE using 10-fold CV by parallel computing\n')
#   cat('Started:',as.character(now()),'\n')
#   cl=makeCluster(detectCores()-1)
#   clusterEvalQ(cl,{
#     library('tidyverse')
#     library('data.table')
#     library('pbapply')
#     library('caret')
#   })
#
#     find_k=
#       seq(3,21) %>% # (k = 3~21 trading off k computation and next lasso regression)
#       pblapply(m_sim=m_sim,m_res=m_res,m_cv=m_cv,val_id=val_id,cl=cl,FUN=function(X,m_sim,m_res,m_cv,val_id){
#         m_kmean=
#           m_sim %>%
#           rename(movieId=movieId1) %>%
#           group_by(movieId) %>%
#           slice_max(order_by=cosine_sim,n=X) %>%
#           select(-cosine_sim) %>%
#           left_join(m_res,by='movieId2') %>%
#           group_by(movieId) %>%
#           summarize(kmean=sum(sum_residual)/sum(n_residual))
#         gc()
#         rm(m_sim)
#
#         m_cv=
#           m_cv %>%
#           left_join(m_kmean,by='movieId') %>%
#           mutate(y_hat=y_hat+ifelse(is.na(kmean),0,kmean))
#         gc()
#         rm(m_kmean)
#
#         result=
#           val_id %>%
#           as.data.table() %>%
#           gather(fold,id) %>%
#           left_join(m_cv,by='id')
#         gc()
#         rm(m_cv,val_id)
#
#         result %>%
#           group_by(fold) %>%
#           summarize(RMSE=RMSE(y_hat,rating)) %>%
#           mutate(k=X)
#       }) %>%
#       do.call(rbind,.)
#
#   stopCluster(cl)
#   rm(cl)
#   gc()
#   cat('End:',as.character(now()))
# }

cat('
Find k most similar neighbors of which average minimize RMSE using 10-fold CV by parallel computing\n
Started: 2020-09-20 08:20:25 \n
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=07m 07s\n
End: 2020-09-20 08:27:42\n
    ')
```

```
##
## Find k most similar neighbors of which average minimize RMSE using 10-fold CV by parallel computing
##
## Started: 2020-09-20 08:20:25
##
##   |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=07m 07s
##
## End: 2020-09-20 08:27:42
##
##
```

Save k comparisons.

```
# if(!'find_k.rds' %in% list.files()) saveRDS(find_k,'find_k.rds')
find_k_file='find_k.rds'
if(!'find_k.rds' %in% list.files()) find_k_file=download_from_my_github(find_k_file)
find_k=readRDS(find_k_file)
```

We can see the RMSE plot below showing further RMSE decreasing is not significant; thus better to get not too large k.



Here below also need time to insert residuals from each neighbor to training set.

```
# Construct movie-movie matrix of outcome and features (residuals of its neighbor),
# as much as k (best value) neighbors
m_kmean=
  m_sim %>%
  rename(movieId=movieId1) %>%
  group_by(movieId) %>%
  slice_max(order_by=cosine_sim,n=12) %>%
  group_by(movieId) %>%
  mutate(rank=paste0('neighbor',rev(rank(cosine_sim)))) %>%
  select(-cosine_sim) %>%
  spread(rank,movieId2)

# # For each movie as outcome, construct a training set based on matrix above
# m_res=
#   train_set %>%
#   group_by(movieId) %>%
#   filter(n()>50) %>%
#   ungroup() %>%
#   group_by(userId) %>%
#   filter(n()>50) %>%
#   ungroup() %>%
#   select(userId,movieId,residual) %>%
#   spread(movieId,residual) %>%
#   column_to_rownames(var='userId') %>%
#   pblapply(X=seq(nrow(m_kmean)),m_kmean=m_kmean,m_res=.,function(X,m_kmean,m_res){
#     m_res %>%
#       select_at(as.character(m_kmean[X,] %>% .[!is.na(.)])) %>%
#       setNames(c('residual',colnames(.)[-1])) %>%
#       filter(!is.na(residual)) %>%
#       filter_all(function(column)mean(is.na(column))<1) %>%
#       mutate_all(function(column)ifelse(is.na(column),0,column))
#   })
m_res2_files=paste0('m_res2_',1:13,'.rds')
# if(!all(m_res2_files %in% list.files())) m_res2_files=saveRDS_split_l(m_res,'m_res2_',13)
if(!all(m_res2_files %in% list.files())) m_res2_files=download_from_my_github(m_res2_files)
m_res=readRDS_split_l(m_res2_files)

cat('|++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=07m 53s')
```

```
## |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=07m 53s
```

We conducted lasso regression. One can see the code below that uses caret and glmnet with alpha=1, which is the setting for lasso. We will make a model for each movie, not user. It was simply like user ridge regression. But, now we need different number of neighbors. Say a neighbor rating did not have significant association, the weights would small or even zero. Then the neighbor simply did not affect a movie to predict.

```
tidy_kmean_lm=function(i,kmean_data_list,id,strata){

  if(dim(kmean_data_list[[i]])[1]>50){
    # suppressWarnings(set.seed(33)) # if using R 3.5 or earlier
    suppressWarnings(set.seed(33,sample.kind = 'Rounding')) # if using R 3.6 or later
    result=
      suppressWarnings(train(
        residual~.,data=kmean_data_list[[i]],method='glmnet',
        trControl=trainControl('cv',number=2),
        tuneGrid=data.frame(alpha=1,lambda=10^seq(-3,3,len=5))
      ))

    result2=
      coef(result$finalModel,result$bestTune$lambda) %>%
      as.matrix()
    rm(result)

    data.frame(
        strata=as(id[i],class(id[i])),
        term=rownames(result2),
        estimate=result2[,1]
      ) %>%
      rename_at('strata',function(column)strata)
  }

}
```

Beware to run the code below. It also applied parallelism. We applied only 2-fold cross validation for this part.

```
# {
#   cat('Conduct lasso regression for each movie by parallel computing\n')
#   cat('Started:',as.character(now()),'\n')
#   cl=makeCluster(detectCores()-1)
#   clusterEvalQ(cl,{
#     library('tidyverse')
#     library('pbapply')
#     library('caret')
#     library('glmnet')
#   })
#
#     movie_lasso=
#       m_res %>%
#       pblapply(
#         X=seq(1,length(.)),
#         FUN=tidy_kmean_lm,
#         kmean_data_list=.,
#         id=m_kmean$movieId,
#         strata='movieId',
#         cl=cl
#       )
#
#   stopCluster(cl)
#   rm(cl)
#   gc()
#   cat('End:',as.character(now()))
# }

cat('
Conduct lasso regression for each movie by parallel computing\n
Started: 2020-09-20 08:39:03 \n
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=03h 21m 12s\n
End: 2020-09-20 12:00:26\n
    ')
```

```
##
## Conduct lasso regression for each movie by parallel computing
##
## Started: 2020-09-20 08:39:03
##
##   |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=03h 21m 12s
##
## End: 2020-09-20 12:00:26
##
##
```

Save the lasso regression results.

```
# if(!'movie_lasso.rds' %in% list.files()) saveRDS(movie_lasso,'movie_lasso.rds')
movie_lasso_file='movie_lasso.rds'
if(!'movie_lasso.rds' %in% list.files()) movie_lasso_file=download_from_my_github(movie_lasso_file)
movie_lasso=readRDS(movie_lasso_file)
```

To insert residuals to different number of neighbors, we build a function as seen below.

```
# Tidy up movie k-neighbors lasso regression table
movie_lasso=
  movie_lasso %>%
  do.call(rbind,.) %>%
  mutate(
    term=
      term %>%
      str_to_lower() %>%
      str_replace('\\(','') %>%
      str_replace('\\)','') %>%
      str_replace('ib','b') %>%
      str_replace('\\^','') %>%
      str_replace_all('\\`','')
  ) %>%
  left_join(
    filter(.,term=='intercept') %>%
      select(movieId,estimate) %>%
      setNames(c('movieId','intercept')),
    by='movieId'
  ) %>%
  filter(term!='intercept') %>%
  select(movieId,intercept,term,estimate) %>%
  mutate(term=as(term,class(movieId))) %>%
  group_by(movieId) %>%
  mutate(seq=seq(n())) %>%
  rename(i_neighbor=intercept,neighborId=term,w_neighbor=estimate) %>%
  pivot_wider(
    names_from='seq',names_sep='',
    values_from=c('neighborId','w_neighbor')
  )

# Create a function to get movie k-neighbor residuals for training set (later)
insert_neighbor=function(set,mkl,cl=NULL){

  set=set %>%
    left_join(mkl,by='movieId')

  small_set=select(set,userId,movieId,residual)

  new_set=
    pblapply(X=1:length(colnames(mkl) %>% .[str_detect(.,'neighborId')]),cl=cl,FUN=function(X){
      set %>%
        left_join(
          small_set %>%
            rename_at('movieId',function(colname)paste0('neighborId',X)) %>%
            rename_at('residual',function(colname)paste0('r_neighbor',X)),
          by=c('userId',paste0('neighborId',X))
        ) %>%
        select(paste0('r_neighbor',X))
    }) %>%
    do.call(cbind,.)

  new_set

}
```

Need to use parallel computing to insert the residuals to training set.

```
# {
#   cat('Get movie k-neighbor residuals for training set  by parallel computing\n')
#   cat('Started:',as.character(now()),'\n')
#   cl=makeCluster(detectCores()-1)
#   clusterEvalQ(cl,{
#     library('tidyverse')
#     library('data.table')
#     library('pbapply')
#   })
#
#     train_neighbors=
#       train_set %>%
#       insert_neighbor(movie_lasso,cl=cl)
#
#   stopCluster(cl)
#   rm(cl)
#   gc()
#   cat('End:',as.character(now()))
# }

cat('
Get movie k-neighbor residuals for training set  by parallel computing\n
Started: 2020-09-20 12:03:50 \n
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05m 52s\n
End: 2020-09-20 12:12:08\n
    ')
```

```
##
## Get movie k-neighbor residuals for training set  by parallel computing
##
## Started: 2020-09-20 12:03:50
##
##   |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=05m 52s
##
## End: 2020-09-20 12:12:08
##
##
```

Again, save the expensive parts.

```
# if(!'train_neighbors.rds' %in% list.files()) saveRDS(train_neighbors,'train_neighbors.rds')
train_neighbors_file='train_neighbors.rds'
if(!'train_neighbors.rds' %in% list.files()) train_neighbors_file=download_from_my_github(train_neighbors_file)
train_neighbors=readRDS(train_neighbors_file)
```

Below we incorporated the lasso weights and the neighbor residuals. After computing the prediction, we used 10-fold cross validation to estimate the RMSE. Finally, we update the residual for the next model.

```
# Incorporate k-neighbors and the weights
train_set=
  train_set %>%
  left_join(movie_lasso,by='movieId') %>%
  mutate_at(colnames(.) %>% .[str_detect(.,'i_neighbor')],function(column)ifelse(is.na(column),0,column)) %>%
  mutate_at(colnames(.) %>% .[str_detect(.,'w_neighbor')],function(column)ifelse(is.na(column),0,column)) %>%
  cbind(train_neighbors) %>%
  mutate_at(colnames(.) %>% .[str_detect(.,'r_neighbor')],function(column)ifelse(is.na(column),0,column)) %>%
  select(-residual) %>%
  mutate(
    userday_z=(userday-m_userday)/s_userday,
    movieday_z=(movieday-m_movieday)/s_movieday,
    movierated_z=(movierated-m_movierated)/s_movierated,
    movie_b_z=(movie_b-m_movie_b)/s_movie_b
  ) %>%
  mutate(
    y_hat=
      mu+movie_b+user_b
      +ur_intercept
      +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
      +ur_movierated_z*movierated_z+ur_userday_z*userday_z
      +i_neighbor
      +w_neighbor1*r_neighbor1+w_neighbor2*r_neighbor2
      +w_neighbor3*r_neighbor3+w_neighbor4*r_neighbor4
      +w_neighbor5*r_neighbor5+w_neighbor6*r_neighbor6
      +w_neighbor7*r_neighbor7+w_neighbor8*r_neighbor8
      +w_neighbor9*r_neighbor9+w_neighbor10*r_neighbor10
      +w_neighbor11*r_neighbor11+w_neighbor12*r_neighbor12
  ) %>%
  as.data.table()

# Estimate RMSE using the best movie k-neighbors model with 10-fold cross validation
rmse[[3]]=
  val_index %>%
  as.data.table() %>%
  gather(fold,seq) %>%
  left_join(
    train_set %>%
      mutate(seq=seq(nrow(.))) %>%
      select(seq,y_hat,rating),
    by='seq'
  ) %>%
  group_by(fold) %>%
  summarize(RMSE=RMSE(y_hat,rating)) %>%
  summarize(
    mean_RMSE=mean(RMSE),
    sd_RMSE=sd(RMSE),
    n_RMSE=n(),
    lb_RMSE=mean_RMSE-qnorm(0.975)*sd_RMSE/sqrt(n_RMSE),
    ub_RMSE=mean_RMSE+qnorm(0.975)*sd_RMSE/sqrt(n_RMSE)
  ) %>%
  select(mean_RMSE,lb_RMSE,ub_RMSE) %>%
  mutate(model='Baseline + user-centered + movie neighbors model')

# Update the residual
train_set=
  train_set %>%
  mutate(residual=rating-y_hat) %>%
  select(-movie_b_z,-movieday_z,-movierated_z,-userday_z,-y_hat)
```

User's implicit neighbor-rating Model

When we used movie k-neighbors model, this mean we used ratings of the same user to a set of movies that is similar to one that the user not rate it yet. Say a movie have 12 neighbors, if 1 movies are already rated, we may think that the user like this kind of movies. If we looked back to user-centered model, number of movie rated contributed to the prediction. The difference here that we may think if similar movies rated, then it is very likely a user give similar rating. The model is specific to the pair of user and the a set of k movies.

In this model, we just count the number of neighbors that already rated. The maximum number, of course, is the k itself. So, we will use a simple linear model using 2-fold corss validation. We used caret and lm as shown below.

```
count_ui_neighbor=function(set){

  mutate(
    set,
    ui_neighbor=
      as.integer(r_neighbor1!=0)+as.integer(r_neighbor2!=0)+
      as.integer(r_neighbor3!=0)+as.integer(r_neighbor4!=0)+
      as.integer(r_neighbor5!=0)+as.integer(r_neighbor6!=0)+
      as.integer(r_neighbor7!=0)+as.integer(r_neighbor8!=0)+
      as.integer(r_neighbor9!=0)+as.integer(r_neighbor10!=0)+
      as.integer(r_neighbor11!=0)+as.integer(r_neighbor12!=0)
  )

}

tidy_implicit_lm=function(i,formula,data,strata){

  result=
    data %>%
    rename_at(strata,function(column)'strata')

  result2=
    result %>%
    filter(strata==i) %>%
    select_at(c(
      as.character(formula)[2],
      str_split(as.character(formula)[3],' \\+ ')[[1]]
    )) %>%
    setNames(c('y',colnames(.)[-1]))

  # suppressWarnings(set.seed(33)) # if using R 3.5 or earlier
  suppressWarnings(set.seed(33,sample.kind = 'Rounding')) # if using R 3.6 or later
  result3=
    train(
      y~.,data=result2,method='lm',
      trControl=trainControl('cv',number=2),
      tuneGrid=data.frame(intercept=0)
    )

  result4=
    coef(result3$finalModel,result3$bestTune$intercept) %>%
    as.matrix()
  intercept=result3$bestTune$intercept
  rm(result3)

  data.frame(
      strata=as(i,class(result$strata)),
      intercept=intercept,
      term=rownames(result4),
      estimate=result4[,1]
    ) %>%
    rename_at('strata',function(column)strata)
}
```

We fitted one model for a user with number of rating >50 only and all of the ratings is not the same (non-zero absolute standard deviation). Please beware to run the code below since we applied parallelism.

```
# {
#   cat('Fit implicit neighbor rating of each user using linear regression by parallel computing\n')
#   cat('Started:',as.character(now()),'\n')
#   cl=makeCluster(detectCores()-1)
#   clusterEvalQ(cl,{
#     library('tidyverse')
#     library('pbapply')
#     library('caret')
#   })
#
#     user_implicit=
#       train_set %>%
#       count_ui_neighbor() %>%
#       group_by(userId) %>%
#       filter(n()>50 & abs(sd(ui_neighbor))>0) %>%
#       ungroup() %>%
#       pull(userId) %>%
#       .[!duplicated(.)] %>%
#       pblapply(
#         FUN=tidy_implicit_lm,
#         formula=residual~ui_neighbor,
#         data=
#           train_set %>%
#           count_ui_neighbor() %>%
#           group_by(userId) %>%
#           filter(n()>50 & abs(sd(ui_neighbor))>0) %>%
#           ungroup() %>%
#           select(userId,residual,ui_neighbor),
#         strata='userId',
#         cl=cl
#       )
#
#   stopCluster(cl)
#   rm(cl)
#   gc()
#   cat('End:',as.character(now()))
# }

cat('
Fit implicit neighbor rating of each user using linear regression by parallel computing\n
Started: 2020-09-20 13:54:57 \n
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=21m 58s\n
End: 2020-09-20 14:17:12\n
    ')
```

```
##
## Fit implicit neighbor rating of each user using linear regression by parallel computing
##
## Started: 2020-09-20 13:54:57
##
##   |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=21m 58s
##
## End: 2020-09-20 14:17:12
##
##
```

Save user implicit models.

```
# if(!'user_implicit.rds' %in% list.files()) saveRDS(user_implicit,'user_implicit.rds')
user_implicit_file='user_implicit.rds'
if(!'user_implicit.rds' %in% list.files()) user_implicit_file=download_from_my_github(user_implicit_file)
user_implicit=readRDS(user_implicit_file)
```

Tidy up the implicit regression table, incorporate to training set, and update the residuals for next model.

```
# Tidy up user implicit regression table
user_implicit=
  user_implicit %>%
  do.call(rbind,.) %>%
  mutate(
    term=
      term %>%
      str_to_lower() %>%
      str_replace('\\(','') %>%
      str_replace('\\)','') %>%
      str_replace('ib','b') %>%
      str_replace('\\^','') %>%
      str_replace_all('\\`','')
  ) %>%
  select(userId,term,estimate) %>%
  spread(term,estimate,fill=0) %>%
  rename_at(colnames(.)[-1],function(colname)paste0('w_',colname))

# Incorporate implicit neighbor rating and the weights
train_set=
  train_set %>%
  left_join(user_implicit,by='userId') %>%
  mutate(w_ui_neighbor=ifelse(is.na(w_ui_neighbor),0,w_ui_neighbor)) %>%
  count_ui_neighbor() %>%
  select(-residual) %>%
  mutate(
    userday_z=(userday-m_userday)/s_userday,
    movieday_z=(movieday-m_movieday)/s_movieday,
    movierated_z=(movierated-m_movierated)/s_movierated,
    movie_b_z=(movie_b-m_movie_b)/s_movie_b
  ) %>%
  mutate(
    y_hat=
      mu+movie_b+user_b
      +ur_intercept
      +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
      +ur_movierated_z*movierated_z+ur_userday_z*userday_z
      +i_neighbor
      +w_neighbor1*r_neighbor1+w_neighbor2*r_neighbor2
      +w_neighbor3*r_neighbor3+w_neighbor4*r_neighbor4
      +w_neighbor5*r_neighbor5+w_neighbor6*r_neighbor6
      +w_neighbor7*r_neighbor7+w_neighbor8*r_neighbor8
      +w_neighbor9*r_neighbor9+w_neighbor10*r_neighbor10
      +w_neighbor11*r_neighbor11+w_neighbor12*r_neighbor12
      +w_ui_neighbor*ui_neighbor
  ) %>%
  as.data.table()

# Estimate RMSE using the best user implicit model with 10-fold cross validation
rmse[[4]]=
  val_index %>%
  as.data.table() %>%
  gather(fold,seq) %>%
  left_join(
    train_set %>%
      mutate(seq=seq(nrow(.))) %>%
      select(seq,y_hat,rating),
    by='seq'
  ) %>%
  group_by(fold) %>%
  summarize(RMSE=RMSE(y_hat,rating)) %>%
  summarize(
    mean_RMSE=mean(RMSE),
    sd_RMSE=sd(RMSE),
    n_RMSE=n(),
    lb_RMSE=mean_RMSE-qnorm(0.975)*sd_RMSE/sqrt(n_RMSE),
    ub_RMSE=mean_RMSE+qnorm(0.975)*sd_RMSE/sqrt(n_RMSE)
  ) %>%
  select(mean_RMSE,lb_RMSE,ub_RMSE) %>%
  mutate(model='Baseline + user-centered + movie neighbors + user implicit model')

# Update the residual
```

```
train_set=
  train_set %>%
  mutate(residual=rating-y_hat) %>%
  select(-movie_b_z,-movieday_z,-movierated_z,-userday_z,-y_hat)
```

Matrix factorization

Either users or movies may have a particular pattern that was not captured by features we aware of. We need to identify latent factors that may define some characteristics of users or movies that may improve the prediction. In previous exploratory data analysis, three singular values decomposed from user-wise rating distribution showed a significant association with rating; however, we only learn these factors from a very small subset of users or movies.

To greatly reduce computational price, we also applied the similar tricks for data processing with those in movie k-neighbor model (see step 1 to 4).

```
# Create a table of movies, users, and ratings with similar tricks with those in movie k-neighbor model
m_u=
  train_set %>%
  group_by(userId) %>%
  mutate(residual=scale(residual)) %>%
  ungroup() %>%
  group_by(movieId) %>%
  mutate(residual=scale(residual)) %>%
  ungroup() %>%
  group_by(movieId) %>%
  filter(n()>50) %>%
  ungroup() %>%
  group_by(userId) %>%
  filter(n()>50) %>%
  ungroup()  %>%
  select(movieId,userId,residual) %>%
  mutate(res_group=round(residual))

# For movie, we grouped based on it and take the average rating in each rating group/bin,
# and also made it re-centered by the average and re-scaled by the standard deviation,
# then conduct singular value decomposition and take the latent factor vectors
m_v=
  m_u %>%
  group_by(movieId,res_group) %>%
  summarize(residual=mean(residual)) %>%
  spread(movieId,residual) %>%
  column_to_rownames(var='res_group') %>%
  t()
m_v[is.na(m_v)]=0
m_v=
  m_v %>%
  sweep(1,rowMeans2(.),'-') %>%
  sweep(1,rowSds(.),'/')
m_v=
  svd(m_v)$u %>%
  `rownames<-`(rownames(m_v))

# For user, we grouped based on it and take the average rating in each rating group/bin,
# and also made it re-centered by the average and re-scaled by the standard deviation,
# then conduct singular value decomposition and take the latent factor vectors
u_v=
  m_u %>%
  group_by(userId,res_group) %>%
  summarize(residual=mean(residual)) %>%
  spread(userId,residual) %>%
  column_to_rownames(var='res_group') %>%
  t()
u_v[is.na(u_v)]=0
u_v=
  u_v %>%
  sweep(1,rowMeans2(.),'-') %>%
  sweep(1,rowSds(.),'/')
u_v=
  svd(u_v)$u %>%
  `rownames<-`(rownames(u_v))

# Create a function to incorporate a pre-defined number of factor into a given dataset by user ID or movie ID
insert_ssvd=function(set,u_v_mat,m_v_mat,range){

  set=mutate(set,ssvd=0)
  for(latent in range){
    set=
      set %>%
      left_join(
        data.table(userId=as(rownames(u_v_mat),class(.$userId)),u_v_mat[,latent,drop=F]) %>%
          setNames(c('userId','ssvd_u')),
        by='userId'
      ) %>%
      left_join(
        data.table(movieId=as(rownames(m_v_mat),class(.$movieId)),m_v_mat[,latent,drop=F]) %>%
          setNames(c('movieId','ssvd_m')),
        by='movieId'
```

```
      ) %>%
      mutate_at(colnames(.) %>% .[str_detect(.,'ssvd_')],function(x)ifelse(is.na(x),0,x)) %>%
      mutate(ssvd=ssvd+(ssvd_u*ssvd_m)) %>%
      select(-ssvd_u,-ssvd_m)
  }
  set

}
```

We identified one or more latent factors from both user and movie. We used these rules or patterns to identify the factors:

1. For simplicity, we take the same factor index for both user and movie
2. We may take one or more factors at a time with f1 to f2 subsetting, e.g 1 to 1 (one factor) or 3 to 6 (more factors)
3. We got the dot products of user against movie factor vectors, this means we multiply them to each other and sum the multiplies
4. The dot products was used to predict the residuals and RMSEs were computed
5. A set of user-movie factor vectors that minimize RMSE would be taken

```
# {
#   cat('Determine number of latent factor for standard SVD for each fold by parallel computing\n')
#   cat('Started:',as.character(now()),'\n')
#   cl=makeCluster(detectCores()-1)
#   clusterEvalQ(cl,{
#     library('tidyverse')
#     library('data.table')
#     library('caret')
#     library('pbapply')
#   })
#
#     fac_ssvd=pblapply(X=1:dim(val_index)[2],train_set=train_set,val_index=val_index,diff_mu=diff_mu,
#                       u_v=u_v,m_v=m_v,insert_ssvd=insert_ssvd,cl=cl,
#                       FUN=function(X,train_set,val_index,diff_mu=diff_mu,u_v=u_v,m_v=m_v,insert_ssvd=insert_ssvd){
#
#       f1=c(1:15,1:14,1:13,1:12,1:11,1:10,1:9,1:8,1:7,1:6,1:5,1:4,1:3,1:2)
#       f2=c(1:15,2:15,3:15,4:15,5:15,6:15,7:15,8:15,9:15,10:15,11:15,12:15,13:15,14:15)
#
#       lapply(X=seq(length(f1)),X2=X,f1=f1,f2=f2,FUN=function(X,X2,f1,f2){
#
#         rmse=
#           train_set %>%
#           diff_mu(val_index[,X2]) %>%
#           select(-residual) %>%
#           insert_ssvd(u_v,m_v,f1[X]:f2[X]) %>%
#           mutate(
#             userday_z=(userday-m_userday)/s_userday,
#             movieday_z=(movieday-m_movieday)/s_movieday,
#             movierated_z=(movierated-m_movierated)/s_movierated,
#             movie_b_z=(movie_b-m_movie_b)/s_movie_b
#           ) %>%
#           mutate(
#             y_hat=
#               mu+movie_b+user_b
#               +ur_intercept
#               +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
#               +ur_movierated_z*movierated_z+ur_userday_z*userday_z
#               +i_neighbor
#               +w_neighbor1*r_neighbor1+w_neighbor2*r_neighbor2
#               +w_neighbor3*r_neighbor3+w_neighbor4*r_neighbor4
#               +w_neighbor5*r_neighbor5+w_neighbor6*r_neighbor6
#               +w_neighbor7*r_neighbor7+w_neighbor8*r_neighbor8
#               +w_neighbor9*r_neighbor9+w_neighbor10*r_neighbor10
#               +w_neighbor11*r_neighbor11+w_neighbor12*r_neighbor12
#               +w_ui_neighbor*ui_neighbor
#               +ssvd
#           ) %>%
#           as.data.table() %>%
#           pull(y_hat) %>%
#           RMSE(diff_mu(train_set,val_index[,X2])$rating,na.rm=T)
#
#         list(f1=f1[X],f2=f2[X],RMSE=rmse)
#
#       })
#
#     })
#
#   suppressWarnings(stopCluster(cl))
#   rm(cl)
#   suppressWarnings(gc())
#   cat('End:',as.character(now()))
# }

cat('
Determine number of latent factor for standard SVD for each fold by parallel computing\n
Started: 2020-09-20 14:26:27 \n
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=35m 25s\n
End: 2020-09-20 15:02:11\n
    ')
```

```
##
## Determine number of latent factor for standard SVD for each fold by parallel computing
##
## Started: 2020-09-20 14:26:27
##
##   |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=35m 25s
##
## End: 2020-09-20 15:02:11
##
##
```

We save the comparison of the standard SVD models.

```
# if(!'fac_ssvd.rds' %in% list.files()) saveRDS(fac_ssvd,'fac_ssvd.rds')
fac_ssvd_file='fac_ssvd.rds'
if(!'fac_ssvd.rds' %in% list.files()) fac_ssvd_file=download_from_my_github(fac_ssvd_file)
fac_ssvd=readRDS(fac_ssvd_file)
```

Below we can see the set of factor vectors from both user and movie that have minimum RMSE on (internal) validation set.

```
## # A tibble: 10 x 9
##       f1    f2 mean_RMSE  sd_RMSE n_RMSE lb_RMSE ub_RMSE f      rank
##    <int> <int>     <dbl>    <dbl>  <int>   <dbl>   <dbl> <fct> <int>
##  1     7     9     0.821 0.000782     10   0.821   0.822 7:9       1
##  2     7     8     0.821 0.000782     10   0.821   0.822 7:8       2
##  3     7    10     0.821 0.000782     10   0.821   0.822 7:10      3
##  4     7     7     0.821 0.000782     10   0.821   0.822 7:7       4
##  5     6     9     0.821 0.000782     10   0.821   0.822 6:9       5
##  6     6     8     0.821 0.000782     10   0.821   0.822 6:8       6
##  7     6    10     0.821 0.000782     10   0.821   0.822 6:10      7
##  8     6     7     0.821 0.000782     10   0.821   0.822 6:7       8
##  9     7    12     0.821 0.000782     10   0.821   0.822 7:12      9
## 10     1     1     0.821 0.000782     10   0.821   0.822 1:1      10
```

Let's tidy up the results, incorporate it into training set, and compute RMSE for this model. Finally, we updated the residual for last model, which is ensemble.

```
# Tidy up selected user and movie factor vectors
user_ssvd=
  data.table(u_v[,7:9],keep.rownames=T) %>%
  setNames(c('userId',paste0('ssvd_u',7:9))) %>%
  mutate(userId=as(userId,class(train_set$userId)))
movie_ssvd=
  data.table(m_v[,7:9],keep.rownames=T) %>%
  setNames(c('movieId',paste0('ssvd_m',7:9))) %>%
  mutate(movieId=as(movieId,class(train_set$movieId)))

# Incorporate standard SVD factor vectors
train_set=
  train_set %>%
  left_join(user_ssvd,by='userId') %>%
  left_join(movie_ssvd,by='movieId') %>%
  mutate_at(colnames(.) %>% .[str_detect(.,'ssvd_')],function(x)ifelse(is.na(x),0,x)) %>%
  select(-residual) %>%
  mutate(
    userday_z=(userday-m_userday)/s_userday,
    movieday_z=(movieday-m_movieday)/s_movieday,
    movierated_z=(movierated-m_movierated)/s_movierated,
    movie_b_z=(movie_b-m_movie_b)/s_movie_b
  ) %>%
  mutate(
    y_hat=
      mu+movie_b+user_b
      +ur_intercept
      +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
      +ur_movierated_z*movierated_z+ur_userday_z*userday_z
      +i_neighbor
      +w_neighbor1*r_neighbor1+w_neighbor2*r_neighbor2
      +w_neighbor3*r_neighbor3+w_neighbor4*r_neighbor4
      +w_neighbor5*r_neighbor5+w_neighbor6*r_neighbor6
      +w_neighbor7*r_neighbor7+w_neighbor8*r_neighbor8
      +w_neighbor9*r_neighbor9+w_neighbor10*r_neighbor10
      +w_neighbor11*r_neighbor11+w_neighbor12*r_neighbor12
      +w_ui_neighbor*ui_neighbor
      +ssvd_u7*ssvd_m7+ssvd_u8*ssvd_m8+ssvd_u9*ssvd_m9
  ) %>%
  as.data.table()

# Estimate RMSE using the best standard SVD model with 10-fold cross validation
rmse[[5]]=
  val_index %>%
  as.data.table() %>%
  gather(fold,seq) %>%
  left_join(
    train_set %>%
      mutate(seq=seq(nrow(.))) %>%
      select(seq,y_hat,rating),
    by='seq'
  ) %>%
  group_by(fold) %>%
  summarize(RMSE=RMSE(y_hat,rating)) %>%
  summarize(
    mean_RMSE=mean(RMSE),
    sd_RMSE=sd(RMSE),
    n_RMSE=n(),
    lb_RMSE=mean_RMSE-qnorm(0.975)*sd_RMSE/sqrt(n_RMSE),
    ub_RMSE=mean_RMSE+qnorm(0.975)*sd_RMSE/sqrt(n_RMSE)
  ) %>%
  select(mean_RMSE,lb_RMSE,ub_RMSE) %>%
  mutate(model='Baseline + user-centered + movie neighbors + user implicit + standard SVD model')

# Update the residual
train_set=
  train_set %>%
  mutate(residual=rating-y_hat) %>%
  select(-movie_b_z,-movieday_z,-movierated_z,-userday_z,-y_hat)
```

Ensemble model

To ensemble previous models, we applied three strategies:

1. Majority vote
2. Average rating
3. Stacking previous models of which results as features in an elastic net regression model

Before we did ensemble, we had compiled previous models as a list. We used whole training set but later we subset it into multiple (internal) validation sets; thus, a 10-fold cross validation was conducted.

```
# Create an empty list to store the models
ensemble=list()

# Baseline model using global, movie-wise, and user-wise average as fixed biases
ensemble$baseline=
  train_set %>%
  mutate(
    y_hat=mu+movie_b+user_b
  ) %>%
  select(id,movieId,userId,y_hat,rating)

# User-centered ridge regression model
ensemble$user_ridge=
  train_set %>%
  mutate(
    userday_z=(userday-m_userday)/s_userday,
    movieday_z=(movieday-m_movieday)/s_movieday,
    movierated_z=(movierated-m_movierated)/s_movierated,
    movie_b_z=(movie_b-m_movie_b)/s_movie_b
  ) %>%
  mutate(
    y_hat=
      mu+movie_b+user_b
      +ur_intercept
      +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
      +ur_movierated_z*movierated_z+ur_userday_z*userday_z
  ) %>%
  select(id,movieId,userId,y_hat,rating)

# Movie k-neighbors model
ensemble$movie_neighbors=
  train_set %>%
  mutate(
    userday_z=(userday-m_userday)/s_userday,
    movieday_z=(movieday-m_movieday)/s_movieday,
    movierated_z=(movierated-m_movierated)/s_movierated,
    movie_b_z=(movie_b-m_movie_b)/s_movie_b
  ) %>%
  mutate(
    y_hat=
      mu+movie_b+user_b
      +ur_intercept
      +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
      +ur_movierated_z*movierated_z+ur_userday_z*userday_z
      +i_neighbor
      +w_neighbor1*r_neighbor1+w_neighbor2*r_neighbor2
      +w_neighbor3*r_neighbor3+w_neighbor4*r_neighbor4
      +w_neighbor5*r_neighbor5+w_neighbor6*r_neighbor6
      +w_neighbor7*r_neighbor7+w_neighbor8*r_neighbor8
      +w_neighbor9*r_neighbor9+w_neighbor10*r_neighbor10
      +w_neighbor11*r_neighbor11+w_neighbor12*r_neighbor12
  ) %>%
  select(id,movieId,userId,y_hat,rating)

# User implicit neighbor rating model
ensemble$user_implicit=
  train_set %>%
  mutate(
    userday_z=(userday-m_userday)/s_userday,
    movieday_z=(movieday-m_movieday)/s_movieday,
    movierated_z=(movierated-m_movierated)/s_movierated,
    movie_b_z=(movie_b-m_movie_b)/s_movie_b
  ) %>%
  mutate(
    y_hat=
      mu+movie_b+user_b
      +ur_intercept
      +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
      +ur_movierated_z*movierated_z+ur_userday_z*userday_z
      +i_neighbor
      +w_neighbor1*r_neighbor1+w_neighbor2*r_neighbor2
      +w_neighbor3*r_neighbor3+w_neighbor4*r_neighbor4
```

```
      +w_neighbor5*r_neighbor5+w_neighbor6*r_neighbor6
      +w_neighbor7*r_neighbor7+w_neighbor8*r_neighbor8
      +w_neighbor9*r_neighbor9+w_neighbor10*r_neighbor10
      +w_neighbor11*r_neighbor11+w_neighbor12*r_neighbor12
      +w_ui_neighbor*ui_neighbor
  ) %>%
  select(id,movieId,userId,y_hat,rating)

# Standard SVD model
ensemble$standard_svd=
  train_set %>%
  mutate(
    userday_z=(userday-m_userday)/s_userday,
    movieday_z=(movieday-m_movieday)/s_movieday,
    movierated_z=(movierated-m_movierated)/s_movierated,
    movie_b_z=(movie_b-m_movie_b)/s_movie_b
  ) %>%
  mutate(
    y_hat=
      mu+movie_b+user_b
      +ur_intercept
      +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
      +ur_movierated_z*movierated_z+ur_userday_z*userday_z
      +i_neighbor
      +w_neighbor1*r_neighbor1+w_neighbor2*r_neighbor2
      +w_neighbor3*r_neighbor3+w_neighbor4*r_neighbor4
      +w_neighbor5*r_neighbor5+w_neighbor6*r_neighbor6
      +w_neighbor7*r_neighbor7+w_neighbor8*r_neighbor8
      +w_neighbor9*r_neighbor9+w_neighbor10*r_neighbor10
      +w_neighbor11*r_neighbor11+w_neighbor12*r_neighbor12
      +w_ui_neighbor*ui_neighbor
      +ssvd_u7*ssvd_m7+ssvd_u8*ssvd_m8+ssvd_u9*ssvd_m9
  ) %>%
  select(id,movieId,userId,y_hat,rating)
```

For ensemble model by majority vote, we treat the prediction as classification task since rating is half-rounded. Each model prediction then being half-rounded. Then, we took a majority class of half-rounded rating as the classification result. Commonly, differences of rating prediction among non-baseline models was not much. Even if we got the differences, these were quite unevenly distributed; yet, the predicted classes are adjacent. Therefore, for simplicity, we used median that tend to take mode of number (equivalent to majority) in this kind of situation. The median function is faster than if we applied a custom function using sapply/lapply.

```
ensemble$majority=
  matrix(c(
    ensemble$baseline$y_hat,
    ensemble$user_ridge$y_hat,
    ensemble$movie_neighbors$y_hat,
    ensemble$user_implicit$y_hat,
    ensemble$standard_svd$y_hat
  ),nrow=length(ensemble$baseline$y_hat)) %>%
  data.table(
    id=ensemble$baseline$id,
    movieId=ensemble$baseline$movieId,
    userId=ensemble$baseline$userId,
    y_hat=rowMedians(round(.*2)/2),
    rating=ensemble$baseline$rating
  ) %>%
  select(id,movieId,userId,y_hat,rating)
```

For ensemble model by average, the procedure is straightforward. We just compute the average predicted rating. No rounding or weighting was applied.

```
ensemble$average=
  matrix(c(
    ensemble$baseline$y_hat,
    ensemble$user_ridge$y_hat,
    ensemble$movie_neighbors$y_hat,
    ensemble$user_implicit$y_hat,
    ensemble$standard_svd$y_hat
  ),nrow=length(ensemble$baseline$y_hat)) %>%
  data.table(
    id=ensemble$baseline$id,
    movieId=ensemble$baseline$movieId,
    userId=ensemble$baseline$userId,
    y_hat=rowMeans2(.),
    rating=ensemble$baseline$rating
  ) %>%
  select(id,movieId,userId,y_hat,rating)
```

For ensemble model by stacking previous models, we will use the predicted ratings from previous models in parallel as features in a linear regression model. Below we just prepare the training set compiling the predicted ratings of previous models.

```
ensemble$stacking=
  data.table(
    id=ensemble$baseline$id,
    movieId=ensemble$baseline$movieId,
    userId=ensemble$baseline$userId,
    rating=ensemble$baseline$rating,
    baseline=ensemble$baseline$y_hat,
    user_ridge=ensemble$user_ridge$y_hat,
    movie_neighbors=ensemble$movie_neighbors$y_hat,
    user_implicit=ensemble$user_implicit$y_hat,
    standard_svd=ensemble$standard_svd$y_hat
  )
```

We applied elastic net regression that regularized the regression using both L1-norm (lasso) and L2-norm (ridge) parameterization. We did not apply grid search for lambda that minimize the (internal) validation RMSE. This means we did not determine by ourself what were the pair of alpha and lambda values to apply. In caret package, we used argument of tuneLength=10 in the train function. The alpha value in either user-centered ridge regression or movie k-neighbor lasso regression were exactly 0 or 1. In this elastic net, instead of grid search, we did random search of 10 sets of alpha and lambda values, randomly determined by the caret. A set that minimize the RMSE would be applied for the final model. We also applied only 2-fold cross validation to find the best tuning parameters.

Below you can see the training function. We did not apply parallelism in this part beause we cannot determine the loops. Fortunately, the computation is fairly affordable with single core processing unit.

```
# Create a function to train an elastic net regression model
tidy_stack_lm=function(data){

  # suppressWarnings(set.seed(33)) # if using R 3.5 or earlier
  suppressWarnings(set.seed(33,sample.kind = 'Rounding')) # if using R 3.6 or later
  result=
    suppressWarnings(train(
      rating~.,data=data,method='glmnet',
      trControl=trainControl('cv',number=2),
      tuneLength=10
    ))

  result2=
    coef(result$finalModel,result$bestTune$lambda) %>%
    as.matrix()
  rm(result)

  data.frame(
    term=rownames(result2),
    estimate=result2[,1]
  )

}

# # Train an elastic net regression model.
# cat('Train an elastic net regression model\n')
# cat('Started:',as.character(now()),'\n')
# stack_elnet=pblapply(X=1,function(X) tidy_stack_lm(select(ensemble$stacking,-id,-movieId,-userId)))[[1]]
# cat('End:',as.character(now()))

cat('
Train an elastic net regression model\n
Started: 2020-09-20 15:10:48 \n
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=10m 59s\n
End: 2020-09-20 15:21:47\n
    ')
```

```
##
## Train an elastic net regression model
##
## Started: 2020-09-20 15:10:48
##
##   |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=10m 59s
##
## End: 2020-09-20 15:21:47
##
##
```

Save the training results.

```
# if(!'stack_elnet.rds' %in% list.files()) saveRDS(stack_elnet,'stack_elnet.rds')
stack_elnet_file='stack_elnet.rds'
if(!'stack_elnet.rds' %in% list.files()) stack_elnet_file=download_from_my_github(stack_elnet_file)
stack_elnet=readRDS(stack_elnet_file)
```

Tidy up the regression table and incorporate it into the stacking part of the ensemble list.

```
# Tidy up elastic net regression table
stack_elnet=
  stack_elnet %>%
  mutate(
    term=
      term %>%
      str_to_lower() %>%
      str_replace('\\(','') %>%
      str_replace('\\)','') %>%
      str_replace('ib','b') %>%
      str_replace('\\^','') %>%
      str_replace_all('\\`','')
  ) %>%
  select(term,estimate) %>%
  spread(term,estimate,fill=0) %>%
  rename_at(colnames(.)[-2],function(colname)paste0('w_',colname))

# Incorporate stacking elastic net and the weights
ensemble$stacking=
  ensemble$stacking %>%
  cbind(stack_elnet) %>%
  mutate(
    y_hat=
      intercept
      +w_baseline*baseline
      +w_user_ridge*user_ridge
      +w_movie_neighbors*movie_neighbors
      +w_user_implicit*user_implicit
      +w_standard_svd*standard_svd
  ) %>%
  as.data.table()
```

Finally, we computed RMSEs of all ensemble models using 10-fold cross validation (shuffle split).

```
# Estimate RMSE using the ensemble majority vote model with 10-fold cross validation
rmse[[6]]=
  val_index %>%
  as.data.table() %>%
  gather(fold,seq) %>%
  left_join(
    ensemble$majority %>%
      mutate(seq=seq(nrow(.))) %>%
      select(seq,y_hat,rating),
    by='seq'
  ) %>%
  group_by(fold) %>%
  summarize(RMSE=RMSE(y_hat,rating)) %>%
  summarize(
    mean_RMSE=mean(RMSE),
    sd_RMSE=sd(RMSE),
    n_RMSE=n(),
    lb_RMSE=mean_RMSE-qnorm(0.975)*sd_RMSE/sqrt(n_RMSE),
    ub_RMSE=mean_RMSE+qnorm(0.975)*sd_RMSE/sqrt(n_RMSE)
  ) %>%
  select(mean_RMSE,lb_RMSE,ub_RMSE) %>%
  mutate(model='Ensemble majority vote model')

# Estimate RMSE using the ensemble average model with 10-fold cross validation
rmse[[7]]=
  val_index %>%
  as.data.table() %>%
  gather(fold,seq) %>%
  left_join(
    ensemble$average %>%
      mutate(seq=seq(nrow(.))) %>%
      select(seq,y_hat,rating),
    by='seq'
  ) %>%
  group_by(fold) %>%
  summarize(RMSE=RMSE(y_hat,rating)) %>%
  summarize(
    mean_RMSE=mean(RMSE),
    sd_RMSE=sd(RMSE),
    n_RMSE=n(),
    lb_RMSE=mean_RMSE-qnorm(0.975)*sd_RMSE/sqrt(n_RMSE),
    ub_RMSE=mean_RMSE+qnorm(0.975)*sd_RMSE/sqrt(n_RMSE)
  ) %>%
  select(mean_RMSE,lb_RMSE,ub_RMSE) %>%
  mutate(model='Ensemble average model')

# Estimate RMSE using the ensemble stacking elastic net model with 10-fold cross validation
rmse[[8]]=
  val_index %>%
  as.data.table() %>%
  gather(fold,seq) %>%
  left_join(
    ensemble$stacking %>%
      mutate(seq=seq(nrow(.))) %>%
      select(seq,y_hat,rating),
    by='seq'
  ) %>%
  group_by(fold) %>%
  summarize(RMSE=RMSE(y_hat,rating)) %>%
  summarize(
    mean_RMSE=mean(RMSE),
    sd_RMSE=sd(RMSE),
    n_RMSE=n(),
    lb_RMSE=mean_RMSE-qnorm(0.975)*sd_RMSE/sqrt(n_RMSE),
    ub_RMSE=mean_RMSE+qnorm(0.975)*sd_RMSE/sqrt(n_RMSE)
  ) %>%
  select(mean_RMSE,lb_RMSE,ub_RMSE) %>%
  mutate(model='Ensemble stacking elastic net regression model')
```

Model validation

We will use only the best model among 5 + 3 (ensemble) models for testing by (external, but not independent?) ‘validation’ set. Suppose we compared all models using this dataset, we might be tempted to change our model; however, we realized that this kind of selection may not estimate the future (unobserved) dataset. So, just use one best model from selection using (internal) cross validation to avoid that temptation :))

Below you will find that we used ‘validation’ for test set. Note, in movie k-neighbor components, we used neighbor ratings (residuals) in training set since we don’t know yet the residuals of the test set. We cannot compute the residuals of test set as the neighbor ones since this part will take true prediction. If we did this, then it would break the rule of validation itself. Therefore, we just use neighbor residuals of training set in a belief that this dataset can estimate the distribution of any other dataset for test set.

Also note, we used first day of user or movie rating, and number of movie rating, based on training set. This is because the first days of a user and movie might be recorded in either edx or validation set. So, we just define those in the training set since we need to train models using features that could only be extracted if these information were available. A similar manner was also applied for number of movie rating. For each day since the first day, this number cumulatively increases, but only counting rating recorded in the edx. If many movies, or rating by some users, in (external) ‘validation’ set was not recorded in edx, then at some points, this will led to higher error when validating our model. If there were negative days, then we just make it to be zero day.

```
test_set=

  ################################### Preprocess data ###################################
  validation %>%
  data_preprocessing$rating() %>%
  rownames_to_column(var='id') %>%

  ################################### Baseline components ###################################
  lapply(X=seq(nrow(.)),Y=.,function(X,Y){
    if(X==1){
      cat('Bind the baseline components','\n')
      Y
    }
  }) %>%
  .[[1]] %>%
  cbind(mu) %>%
  left_join(movie_b,by='movieId') %>%
  left_join(user_b,by='userId') %>%

  ################################### User-centered components ###################################
  lapply(X=seq(nrow(.)),Y=.,function(X,Y){
    if(X==1){
      cat('Bind the user-centered components.','\n')
      Y
    }
  }) %>%
  .[[1]] %>%
  left_join(
    validation %>%
      data_preprocessing$user_movie(
        verbose=F,
        validation_mode=T,
        training_set=edx  #>>>>>>>>>>>> First days and number of rating from training set, read the reason above
      ) %>%
      rownames_to_column(var='id'),
    by='id'
  ) %>%
  cbind(ur_mus) %>%
  mutate(
    userday_z=(userday-m_userday)/s_userday,
    movieday_z=(movieday-m_movieday)/s_movieday,
    movierated_z=(movierated-m_movierated)/s_movierated,
    movie_b_z=(movie_b-m_movie_b)/s_movie_b
  ) %>%
  left_join(user_ridge,by='userId') %>%

  ################################### Movie k-neighbor components ###################################
  lapply(X=seq(nrow(.)),Y=.,function(X,Y){
    if(X==1){
      cat('Bind the movie k-neighbor components.','\n')
      Y
    }
  }) %>%
  .[[1]] %>%
  left_join(movie_lasso,by='movieId') %>%
  mutate_at(colnames(.) %>% .[str_detect(.,'i_neighbor')],function(column)ifelse(is.na(column),0,column)) %>%
  mutate_at(colnames(.) %>% .[str_detect(.,'w_neighbor')],function(column)ifelse(is.na(column),0,column)) %>%
  left_join(
    train_set %>% #>>>>>>>>>>>>>>>>> We use neighbors' residuals of training set, nothing from test set yet
      select(movieId,userId) %>%
      cbind(readRDS('train_neighbors.rds')) %>%
      mutate_at(colnames(.) %>% .[str_detect(.,'r_neighbor')],function(column)ifelse(is.na(column),0,column)),
    by=c('movieId','userId')
  ) %>%
  mutate_at(colnames(.) %>% .[str_detect(.,'r_neighbor')],function(column)ifelse(is.na(column),0,column)) %>%

  ################################### User implicit rating components ###################################
  lapply(X=seq(nrow(.)),Y=.,function(X,Y){
    if(X==1){
      cat('Bind the user implicit neighbor rating components.','\n')
      Y
    }
```

```
}) %>%
.[[1]] %>%
left_join(user_implicit,by='userId') %>%
mutate(w_ui_neighbor=ifelse(is.na(w_ui_neighbor),0,w_ui_neighbor)) %>%
count_ui_neighbor() %>%

################################### Standard SVD components ###################################
lapply(X=seq(nrow(.)),Y=.,function(X,Y){
  if(X==1){
    cat('Bind the standard SVD components.','\n')
    Y
  }
}) %>%
.[[1]] %>%
left_join(user_ssvd,by='userId') %>%
left_join(movie_ssvd,by='movieId') %>%
mutate_at(colnames(.) %>% .[str_detect(.,'ssvd_')],function(x)ifelse(is.na(x),0,x)) %>%

################################### Ensemble stacking components ###################################
lapply(X=seq(nrow(.)),Y=.,function(X,Y){
  if(X==1){
    cat('Bind the ensemble stacking elastic net regression components.','\n')
    Y
  }
}) %>%
.[[1]] %>%
cbind(stack_elnet) %>%

################################### Validate the prediction model ###################################
lapply(X=seq(nrow(.)),Y=.,function(X,Y){
  if(X==1){
    cat('Predict the validation set.','\n')
    Y
  }
}) %>%
.[[1]] %>%
mutate(

  ############## Compute baseline prediction ##############
  baseline=
    mu+movie_b+user_b,

  ############## Compute user-centered prediction ##############
  user_ridge=
    baseline
    +ur_intercept
    +ur_movie_b_z*movie_b_z+ur_movieday_z*movieday_z
    +ur_movierated_z*movierated_z+ur_userday_z*userday_z,

  ############## Compute movie k-neighbor prediction ##############
  movie_neighbors=
    user_ridge
    +i_neighbor
    +w_neighbor1*r_neighbor1+w_neighbor2*r_neighbor2
    +w_neighbor3*r_neighbor3+w_neighbor4*r_neighbor4
    +w_neighbor5*r_neighbor5+w_neighbor6*r_neighbor6
    +w_neighbor7*r_neighbor7+w_neighbor8*r_neighbor8
    +w_neighbor9*r_neighbor9+w_neighbor10*r_neighbor10
    +w_neighbor11*r_neighbor11+w_neighbor12*r_neighbor12,

  ############## Compute user implicit neighbor rating prediction ##############
  user_implicit=
    movie_neighbors
    +w_ui_neighbor*ui_neighbor,

  ############## Compute standard SVD prediction ##############
  standard_svd=
    user_implicit
    +ssvd_u7*ssvd_m7+ssvd_u8*ssvd_m8+ssvd_u9*ssvd_m9,

  ############## Ensemble-stacking all predictions weighted by elastic net regression ##############
  y_hat=
```

```
      intercept
      +w_baseline*baseline
      +w_user_ridge*user_ridge
      +w_movie_neighbors*movie_neighbors
      +w_user_implicit*user_implicit
      +w_standard_svd*standard_svd

  ) %>%

  lapply(X=seq(nrow(.)),Y=.,function(X,Y){
    if(X==1){
      cat('Done.','\n')
      Y
    }
  }) %>%
  .[[1]] %>%
  as.data.table()
```

```
## Create IDs for every combination of userId and movieId
## Select only IDs, movieId, userId, and rating
## Done.
## Bind the baseline components
## Bind the user-centered components.
## Bind the movie k-neighbor components.
## Bind the user implicit neighbor rating components.
## Bind the standard SVD components.
## Bind the ensemble stacking elastic net regression components.
## Predict the validation set.
## Done.
```

Results

Remember that our goal was:

"... to predict user rating with target root mean squared error (RMSE) <0.86490."

Look at test_set that was constructed by the code above and that being presented by the code below. The best model can achieve the goal. The RMSE was 0.85706. This is lower than the target of <0.86490 by 0.00784. So, we did it, we have achieved the goal :))

```
cat('RMSE = ',round(RMSE(test_set$y_hat,test_set$rating),5),'\n',
    '25 points: RMSE <0.86490 :))',sep='')
```

```
## RMSE = 0.85706
## 25 points: RMSE <0.86490 :))
```

Just for sanity check. Let's use the original 'validation' set instead of processing it into test_set. But, stick to use prediction from test_set.

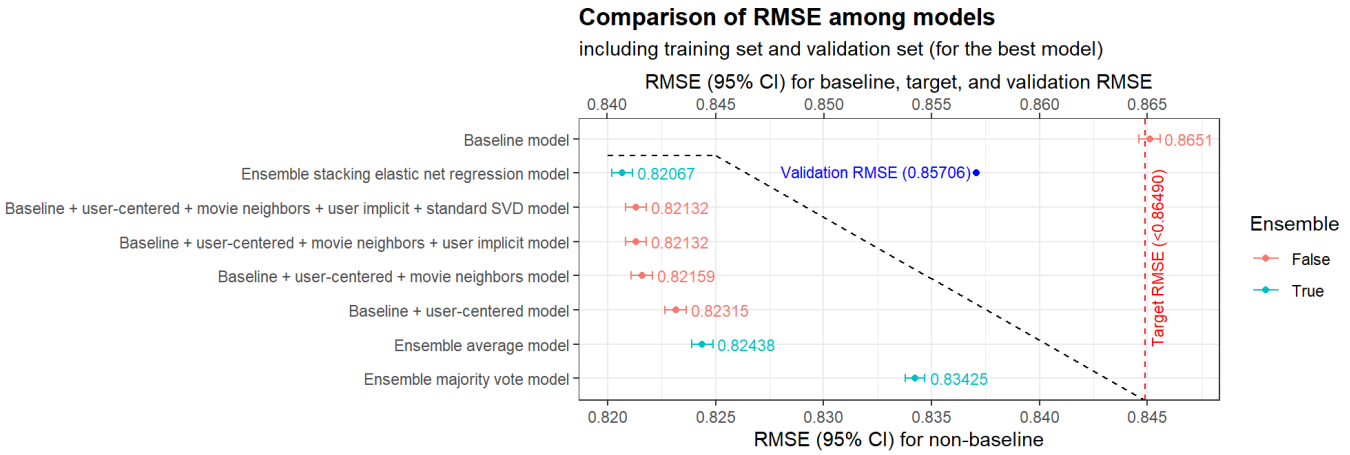
```
cat('RMSE = ',round(RMSE(test_set$y_hat,validation$rating),5),'\n',
    '25 points: RMSE <0.86490 :))',sep='')
```

```
## RMSE = 0.85706
## 25 points: RMSE <0.86490 :))
```

The same RMSE, isn't it?

So, what is the best model? The best model is the ensemble model by stacking previous models using an elastic net regression. Below you can see that majority vote did not work well, so was the ensemble model by average. It is because this kind of ensemble also included baseline model which has the highest RMSE. Note, in the plot below, the points with 95% confidence intervals were RMSEs from (internal) cross validation using 10 shuffle splits.

For comparison, we also plot validation RMSE using the 'validation' set that was used to test the best model only, which was the ensemble stacking elastic net regression model. Except the baseline model, any RMSEs achieved the target RMSE. Note, we have different x scale for RMSE of non-baseline (scale at the bottom) with baseline, target, and validation RMSE (scale at the top). This is because the non-baseline mostly relatively far below the baseline etc, which makes the plot is too concentrated on the left side and make us difficult to compare RMSE among them. For better visualization, we used two scale, please beware to interpret this plot. We drawed a black, dashed-line to separate points that use different scales.



From this comparison, we learn that the RMSE was getting lower as we applied boosting in which next model predict errors or residuals of previous one. However, the decreasing magnitude was also more tiny which become not so significant. For example, we clearly see the model is significantly improve after boosting by user-centered model and continuously improved after more boosting by movie k-neighbor model. The interval estimate of RMSE was not overlapped among baseline and the two models.

Boosting by the user implicit neighbor rating model did not improve the RMSE significantly. So was the standard SVD model. But, the RMSE interval estimate decreases fairly significant compared to the movie k-neighbor model after boosting by the ensemble stacking elastic net regression model. The RMSE point estimate of this model is not within RMSE interval estimate of the movie k-neighbor model.

But, these are RMSEs of cross (internal) validation. Although we used 95% confidence interval to approach covering true RMSE, the 'validation' RMSE is not covered by the interval from the corresponding model. This may be due to two things.

First, we need to understand that the 'validation' (test set) is also random variable that estimate true RMSE of the population (arbitrary dataset for test set). We may be 'not lucky' this time because getting that (random) samples which is lower than the expected value estimated by cross validation. In this reason, we assume cross validation is robust to estimate the true RMSE.

Second, we assume that cross validation is not that robust. A guideline for risk of bias assessment of a multivariable predictive modeling (PROBAST) recommended bootstrapping for internal validation. This is reasonable since boostrapping typically runs for >200 repetitions that is belief to be better (such we believe large sample is better than the small one for inference statistics). However, such bootstrapping is almost impossible, if not all, for big data. We even used only 2-fold instead of 5- or 10-fold cross validation to pick the best tuning parameters, although we chose the model using 10-fold cross validation.

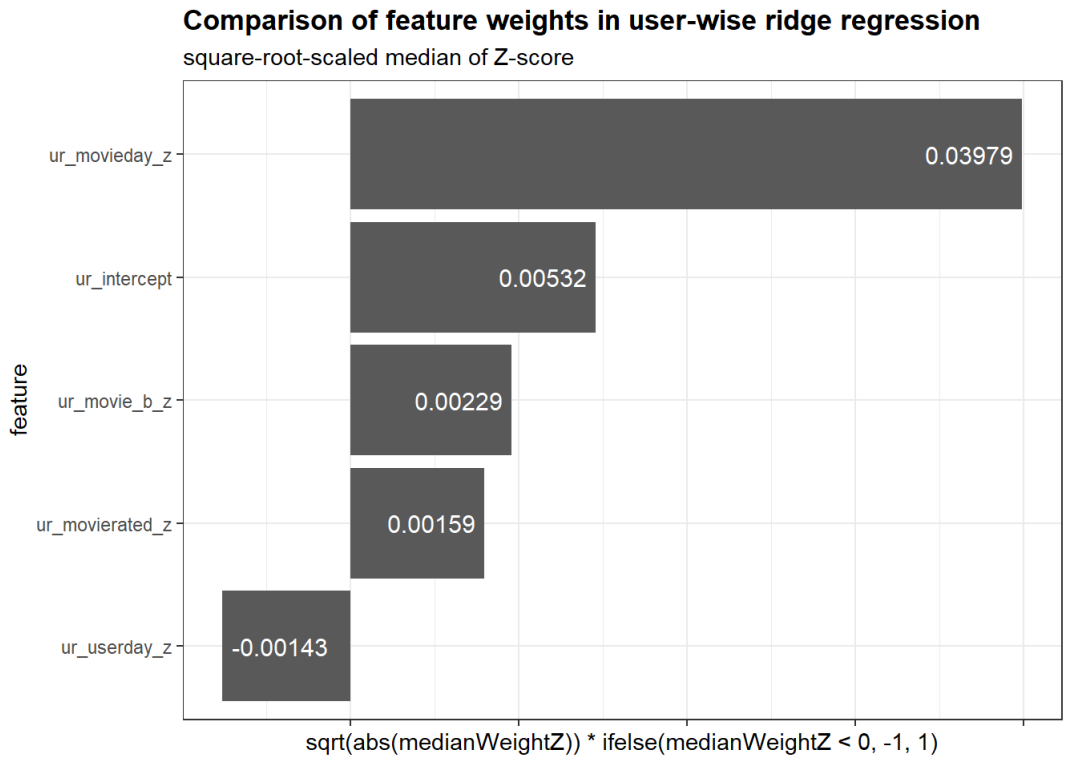
Anyway, we can achieve the target RMSE without 'sneaking a peek' at the 'validation' data :D. This was why we did not preprocess 'validation' data until the phase we validate the best model.

Now, let's briefly look at the best model as a prediction function. Below you can see the best model:

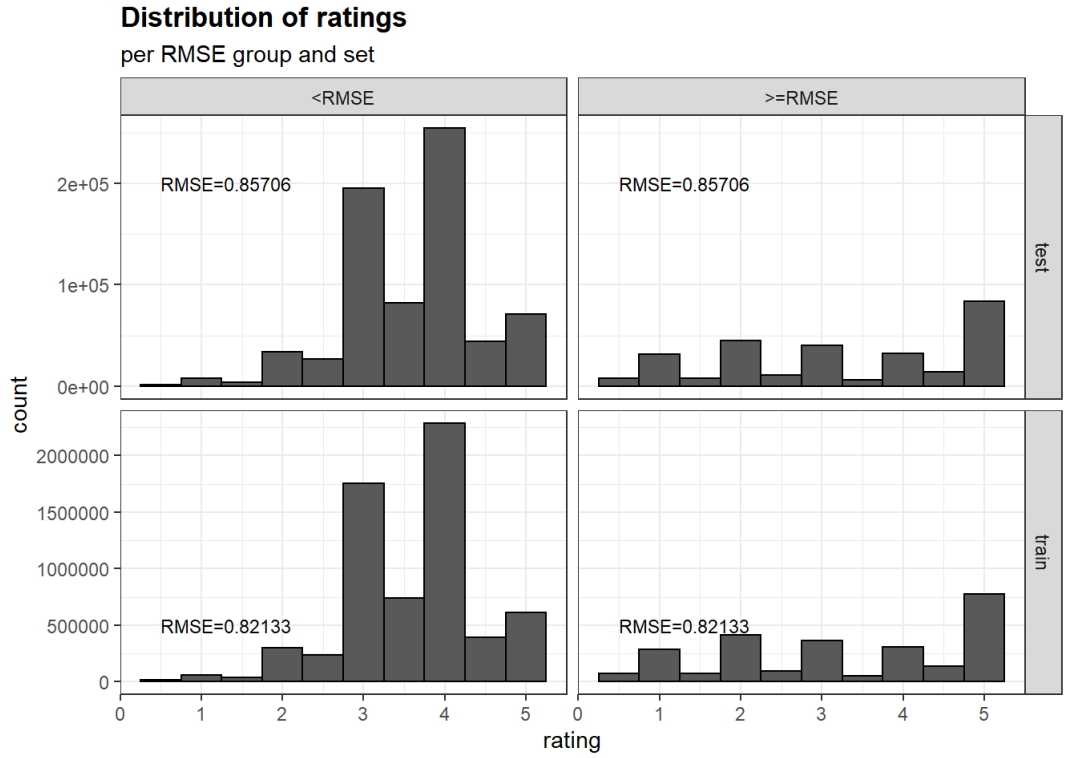
```
## y_hat
## =0.007367358
##   - 0.1391493 x  basePred
##   + 0.1339218 x (basePred + userWs_ridgeR)
##   + 0.3011024 x (basePred + userWs_ridgeR + movieWs_kN_lassoR)
##   + 0.4125280 x (basePred + userWs_ridgeR + movieWs_kN_lassoR + userWs_kNRNum_linearReg)
##   + 0.2902015 x (basePred + userWs_ridgeR + movieWs_kN_lassoR + userWs_kNRNum_linearR + userWs_movieWs_stdSVD)
```

We could multiply each weight with corresponding model prediction within each pair of parentheses. This implied user-wise ridge regression being heaviest-weighted, even compared to baseline prediction using rating averages. The weights then were followed by movie k-neighbor, baseline, user implicit neighbor rating, and standard SVD models. Well, the user-wise ridge regression is the most expensive computation that payoffs.

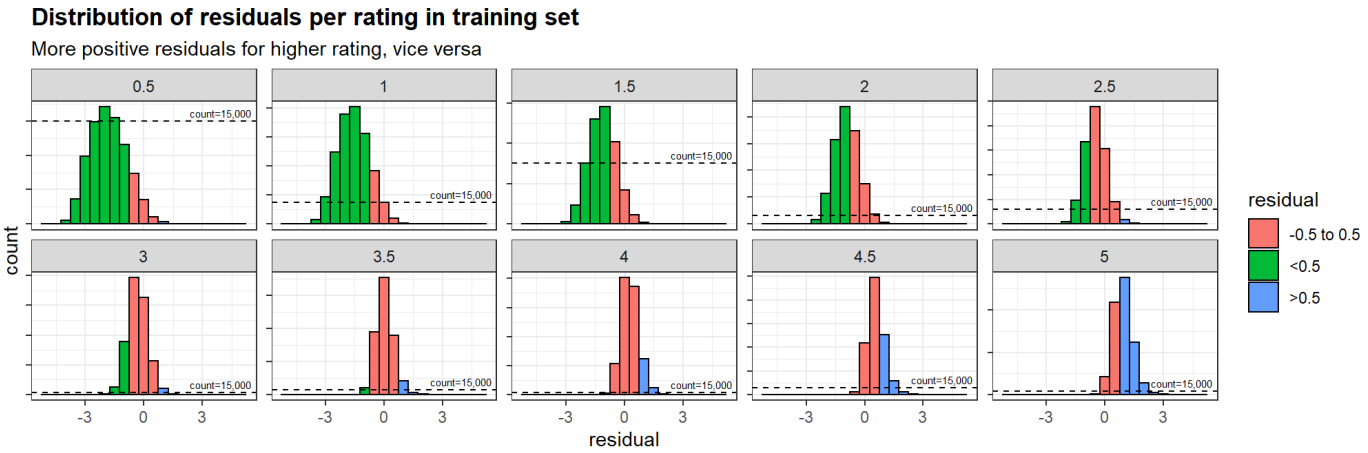
We depicted the magnitude of weights among users in the user ridge regression model. Number of days since movie first rating was the most important to increase the rating, while those since user first rating was the opposite. We showed the real median Z-score of weight for each feature in the model, but for visualization purpose, the bar length is square-root-scaled (absolute value applied to avoid error).



What makes higher RMSE? Below we compared distribution of ratings between predicted ones less RMSE and more RMSE for either training or test set. The less- and more-RMSE groups were classified by individual residual against RMSE as the cutoff. We found rating of 5 in more-RMSE group has higher counts against other ratings compared to those in less-RMSE group. This applied for both training and test sets. We thought these might be due to much prediction were out of range of rating, especially more than the upper bound.



Below we depicted distribution of residuals per rating in training set only. The y-scale were removed to make plot area wider, but we provide a reference line to get a sense how much counts corresponding to the bar height. This figure demonstrated a bias of extreme rating prediction. Either lower or higher rating have higher residuals. Because lower ratings have less counts that higher ones, the magnitude of this bias is not much for lower rating. Meanwhile, higher rating have much counts. This is why the figure above showed higher relative counts of rating of 5 compared to those of rating of 0.5.



Conclusion

In conclusion, ensemble staking of multiple prediction models by an elastic net regression could achieved predictive performance of 0.85706, as shown by the 'validation' set. The prediction models consisted of baseline model by rating averages, user-wise ridge regression, movie k-neighbor lasso regression, user-wise movie neighbor residual (rating) regression, and standard user- and movie standard SVD model.

Some of limitations of this work are related to both modeling and output processing. In the modeling, we still did not include other features such time and genre. We also did not apply Restricted Boltzmann Machines that was shown in NetFlix challenge could give significant improvement, particularly for movies or users that have low numbers of ratings. We only applied SVD as matrix factorization and this technique was found in that challenge useful when movies or users have high numbers of ratings. In the output processing, we did not scale the predicted rating. This seem contributing higher RMSE in both training and validation sets; however, we did not apply optimization for this part since this may be overfitting because we already used validation set to make this conclusion.

Future work need to apply more features and algorithms that have different ways of handling problems. The predicted ratings should also be scaled. Alternatively, treating the prediction as a classification problem may also be better approach.

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18363)
##
## Matrix products: default
##
## Random number generation:
##  RNG:     Mersenne-Twister
##  Normal:  Inversion
##  Sample:  Rounding
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
## system code page: 950
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] pbapply_1.4-3       glmnet_4.0-2        Matrix_1.2-18
##  [4] dslabs_0.7.3        zeallot_0.1.0       broom_0.7.0
##  [7] matrixStats_0.56.0  data.table_1.13.0   caret_6.0-86
## [10] lattice_0.20-41     forcats_0.5.0       stringr_1.4.0
## [13] dplyr_1.0.0         purrr_0.3.4         readr_1.3.1
## [16] tidyr_1.1.0         tibble_3.0.3        ggplot2_3.3.2
## [19] tidyverse_1.3.0     lubridate_1.7.9     Biobase_2.48.0
## [22] BiocGenerics_0.34.0 usethis_1.6.1
##
## loaded via a namespace (and not attached):
##  [1] colorspace_1.4-1     ellipsis_0.3.1       class_7.3-17
##  [4] rprojroot_1.3-2      fs_1.4.2             rstudioapi_0.11
##  [7] farver_2.0.3         remotes_2.2.0        prodlim_2019.11.13
## [10] fansi_0.4.1          xml2_1.3.2           codetools_0.2-16
## [13] splines_4.0.2        knitr_1.29           pkgload_1.1.0
## [16] jsonlite_1.7.0       pROC_1.16.2          dbplyr_1.4.4
## [19] BiocManager_1.30.10  compiler_4.0.2       httr_1.4.2
## [22] backports_1.1.8      assertthat_0.2.1     cli_2.0.2
## [25] htmltools_0.5.0      prettyunits_1.1.1    tools_4.0.2
## [28] gtable_0.3.0         glue_1.4.1           reshape2_1.4.4
## [31] Rcpp_1.0.5           cellranger_1.1.0     vctrs_0.3.2
## [34] nlme_3.1-148         iterators_1.0.12     timeDate_3043.102
## [37] gower_0.2.2          xfun_0.16            ps_1.3.3
## [40] testthat_2.3.2       rvest_0.3.6          lifecycle_0.2.0
## [43] MASS_7.3-51.6        scales_1.1.1         ipred_0.9-9
## [46] hms_0.5.3            yaml_2.2.1           memoise_1.1.0
## [49] rpart_4.1-15         stringi_1.4.6        desc_1.2.0
## [52] foreach_1.5.0        pkgbuild_1.1.0       lava_1.6.7
## [55] shape_1.4.4          rlang_0.4.7          pkgconfig_2.0.3
## [58] evaluate_0.14        recipes_0.1.13       labeling_0.3
## [61] processx_3.4.3       tidyselect_1.1.0     plyr_1.8.6
## [64] magrittr_1.5         R6_2.4.1             generics_0.0.2
## [67] DBI_1.1.0            mgcv_1.8-31          pillar_1.4.6
## [70] haven_2.3.1          withr_2.2.0          survival_3.1-12
## [73] nnet_7.3-14          modelr_0.1.8         crayon_1.3.4
## [76] utf8_1.1.4           rmarkdown_2.3        grid_4.0.2
## [79] readxl_1.3.1         blob_1.2.1           callr_3.4.3
## [82] ModelMetrics_1.2.2.2 reprex_0.3.0         digest_0.6.25
## [85] stats4_4.0.2         munsell_0.5.0        sessioninfo_1.1.1
```