Recommender Systems: Predicting movie ratings with MovieLens

Alex V. Hadar

7/28/2021

Introduction

Recommender systems have become in recent years part of everyday life for an increasing number of people. Although many of us use them for shopping, movie selection or music choice, the inner workings of such a system are not always obvious.

The general approach includes user and item data, ratings and similarities. In more detail, the recommender systems literature mentions different types of recommenders: non-personalized (based on item groupings), personalized (based on user-item groupings), content-based (using descriptions or reviews).

Models and methods used to predict user ratings or create lists of recommendations range from linear regression, collaborative filtering, matrix factorization to neural networks.

In this project we will use the Movielens-10M data to predict user ratings based on other ratings only. Alternative approaches could add genre or temporal data to rating data.

Analysis

We start the analysis with an overview of the data, followed by a short visual exploration. After this, we present different recommendation algorithms and their performance. Finally, the best model is used for predictions. Recommender problems can be regarded as regression, which means we can use RMSE and MAE as performance metrics:

RMSE (Root Mean Squared Error): $\sqrt{\frac{1}{N}\sum(y-\hat{y})^2} = \sqrt{\frac{1}{N}\sum_{u,i}(y_{u,i}-\hat{y}_{u,i})^2}$

MAE (Mean Absolute Error): $\frac{1}{N}\sum|y-\hat{y}| = \frac{1}{N}\sum_{u,i}|y_{u,i}-\hat{y}_{u,i}|$

where u is the user, i is the movie, N is the total number of ratings, $y_{u,i}$ is the rating given by user u for movie i, and $\hat{y}_{u,i}$ is the predicted rating given by user u for movie i.

Exploratory data analysis (EDA)

The Movielens-10M dataset can be downloaded and unpacked using the following code:

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

```
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))
# if using R 4.0 or later:
# movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
#                                            title = as.character(title),
#                                            genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

According to the Moveielens site, the data includes 10 million ratings and additional data, generated for 10000 movies by 72000 users. Ratings range from 0.5 to 5.0 stars in increments of 0.5. This means half-star ratings are allowed. Also, the total number of ratings is significantly lower than the number of users times the number of movies, which means sparse rating data.

For the purpose of analysis and modeling we split the data in two parts: edx and validation dataset, with 90% allocated to edx, 10% to the validation. The edx part will be used for training/test of various models. The validation data is the final holdout dataset and will be used on the chosen algorithm for performance evaluation using RMSE. Users and movies included in the validation data are also present in the edx data, to avoid predictions for new users without available data.

```
# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

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

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

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

# store the edx and validation data for later use
saveRDS(edx, file = "edx.Rda")
saveRDS(validation, file = "vali.Rda")
```

The edx data consists of 9000055 ratings created by 69878 users for 10677 movies. It also includes genre information for the movies and timestamps of the ratings.

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

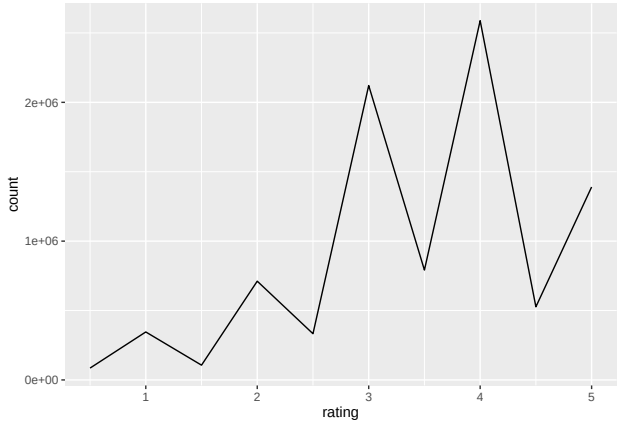
Only the edx data will be used for training and tuning the models. For this reason, we split this data in two parts: training and test data, with 90% allocated to training, 10% to test. We are interested in the available ratings, not genre or time features.

```
edx <- readRDS("edx.Rda")
# keep only the rating information
edx.user.ratings <- edx %>% select(userId, movieId, rating)
# head(edx.user.ratings)

# test set will be 10% of edx data
# set.seed(1, sample.kind="Rounding") # set before, so not needed again;
test_index <- createDataPartition(y = edx.user.ratings$rating, times = 1, p = 0.1,
                                  list = FALSE)
train <- edx.user.ratings[-test_index,]
temp <- edx.user.ratings[test_index,]

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

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

rm(test_index, temp, edx.user.ratings, removed)
```

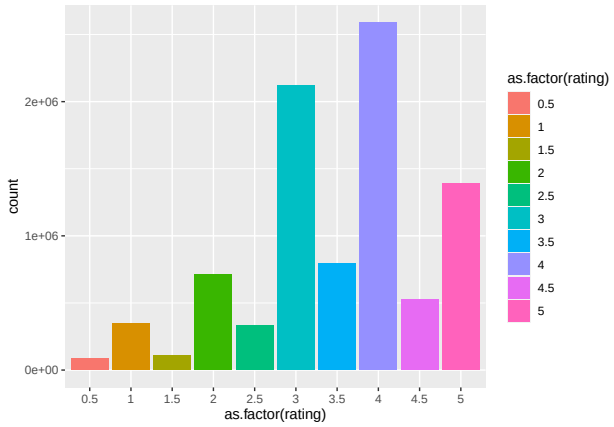
The next step is a short visual exploration of the data.

Rating distribution (number of ratings per class/number of stars) There are less half-star ratings than full-star ratings. Most ratings are at least 3 stars.

```
edx %>% group_by(rating) %>% summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) + geom_line()
```

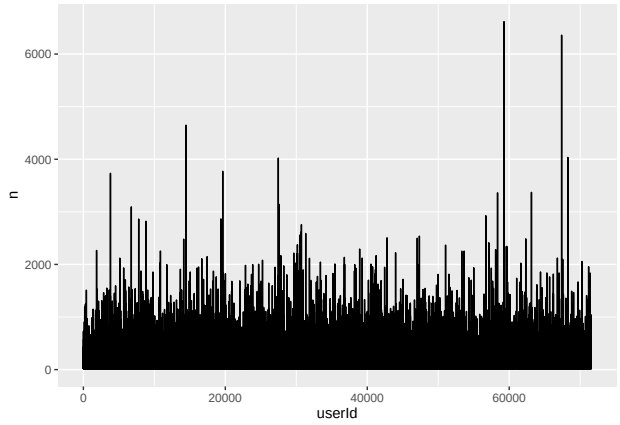



```
edx %>% ggplot(aes(x = as.factor(rating))) + geom_bar(aes(fill = as.factor(rating)))
```



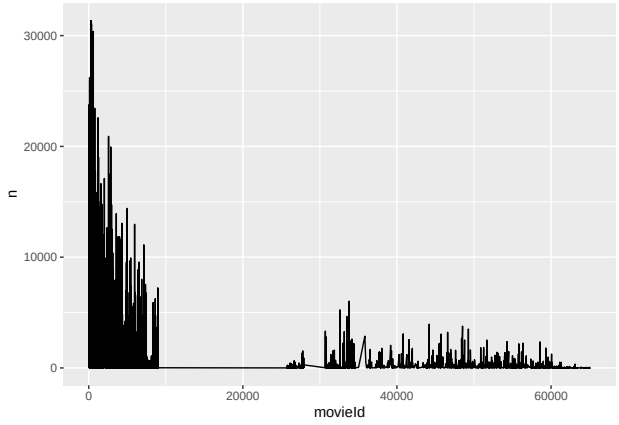
Rating distribution (number of ratings per user) There are only a few users who rated many thousand movies. This means that overall the ratings matrix is sparse, because each user rates only a small part of the available movies.

```
user.ratings <- edx %>% group_by(userId) %>%
                  summarize(n= n(), avg.rating = mean(rating))
user.ratings %>% ggplot(aes(x = userId, y = n)) + geom_line()
```



Rating distribution (number of ratings per movie) Some movies are rated and watched by tens of thousands of users - the blockbuster movies, while others are only rated hundreds of times or less.

```
movie.ratings <- edx %>% group_by(movieId) %>%
                  summarize(n= n(), avg.rating = mean(rating))
movie.ratings %>% ggplot(aes(x = movieId, y = n)) + geom_line()
```



Genre (selection, non-exclusive) Drama and comedy are among the most popular genres.

```
##     Drama    Comedy  Thriller   Romance Adventure    Sci-Fi
##   3910127   3540930   2325899   1712100   1908892   1341183
```

Top movie counts Most-rated movies from the dataset are shown below:

```
# explore the movie top counts
most_rated <- edx %>% group_by(movieId, title) %>%
         summarize(count = n()) %>% arrange(desc(count))
head(most_rated, n=10)
```

```
## # A tibble: 10 x 3
## # Groups:   movieId [10]
##    movieId title                                                  count
##      <dbl> <chr>                                                  <int>
##  1     296 Pulp Fiction (1994)                                    31362
##  2     356 Forrest Gump (1994)                                    31079
##  3     593 Silence of the Lambs, The (1991)                       30382
##  4     480 Jurassic Park (1993)                                   29360
##  5     318 Shawshank Redemption, The (1994)                       28015
##  6     110 Braveheart (1995)                                      26212
##  7     457 Fugitive, The (1993)                                   25998
##  8     589 Terminator 2: Judgment Day (1991)                      25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                       24284
```

Models

Same rating for all users and movies

For a first simple model, we assume that the predicted value is always the mean value of all ratings from the dataset. There are no differences between users or movies in this setting.

$Y_{u,i} = \mu + \varepsilon_{u,i}$

```
# Root Mean Square Error
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

# Mean Absolute Error
MAE <- function(true_ratings, predicted_ratings){
  mean(abs(true_ratings - predicted_ratings))
}


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

```
## [1] 3.512509
```

Model performance is determined using RMSE and MAE, then the results are stored:

```
rmse.model1 <- RMSE(test$rating, mu)
rmse.model1
```

```
## [1] 1.061135
```

```
mae.model1 <- MAE(test$rating, mu)
mae.model1
```

```
## [1] 0.8563163
```

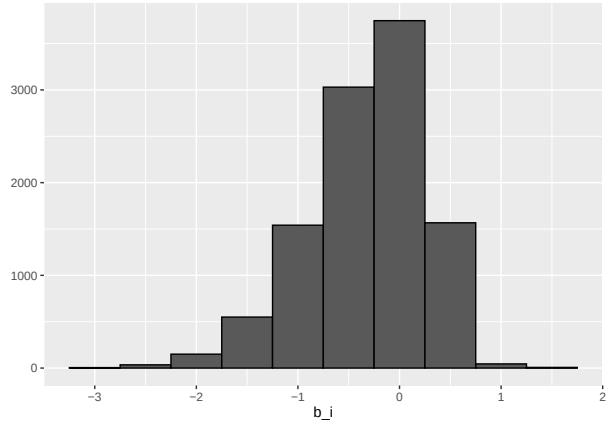
```
rmse_results <- data_frame(method = "Mean value", RMSE = rmse.model1, MAE = mae.model1)
```

Model with movie effect

There are differences between movies, which are accounted for using the movie effect variable. For example, some movies are blockbusters and are rated consistently high by a large number of users, while other, lesser known movies, may have mixed reviews.

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

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

movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



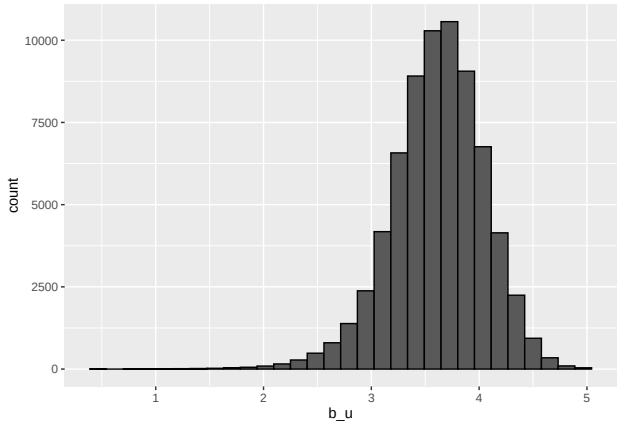

```
predicted_ratings <- mu + test %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

rmse.model2 <- RMSE(test$rating, predicted_ratings)
mae.model2 <- MAE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effects Model",
                                     RMSE = rmse.model2, MAE = mae.model2 ))
```

Model with movie effect and user effect

In addition to movie differences. , there are also differences between users. Users which give consistently high or consistently low ratings are considered when we add a user effect variable to the model.

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

```
train%>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



```
# lm is VERY slow here -> do not run this code
# lm(rating ~ as.factor(movieId) + as.factor(userId))
user_avgs <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- test%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

rmse.model3 <- RMSE(test$rating, predicted_ratings)
mae.model3 <- MAE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = rmse.model3, MAE = mae.model3 ))
```

Model using regularization

In order to avoid overfitting and poor performance, regularization can be used. In this case, a penalty term is applied to the model and affects the error weight: higher errors have higher impact. For our model the terms are squared. Regularizaton can be applied to variables(movie effect) or to groups of variables(movie and user effects).

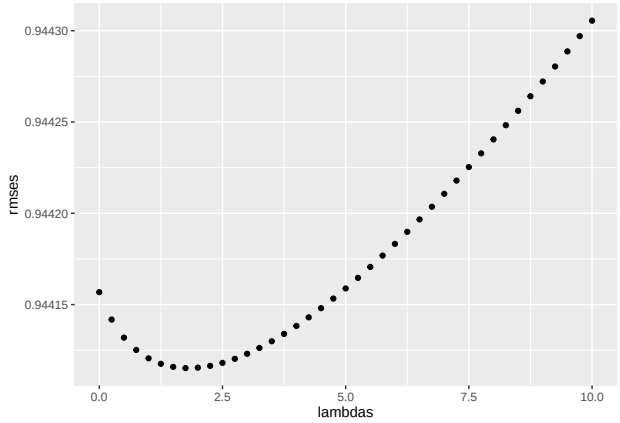
To determine the regularization parameter *lambda* we use cross-validation. We also take into consideration the different number of ratings per movie and per user.

Regularization for movie effects:

$$Y_{u,i} = \mu + \frac{1}{\lambda + n_i} b_i + \varepsilon_{u,i}$$

In this case we minimize $\sum_{u,i}(y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$.

```
# for movie effects
# choose lambda using cross validation
lambdas <- seq(0, 10, 0.25)
mu <- mean(train$rating)
just_the_sum <- train %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(test$rating, predicted_ratings))
})
qplot(lambdas, rmses)
```




```
lambda <- lambdas[which.min(rmses)]
lambda # 1.75 for movie effects
```

```
## [1] 1.75
```

```
# penalty term  with value 1.75 -> see above how it is chosen (cross validation)
# lambda <- 1.75 # use lambda from above directly
mu <- mean(train$rating)
movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

rmse.model4 <- RMSE(test$rating, predicted_ratings)
mae.model4 <- MAE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effects Model",
                                     RMSE = rmse.model4, MAE = mae.model4 ))
```

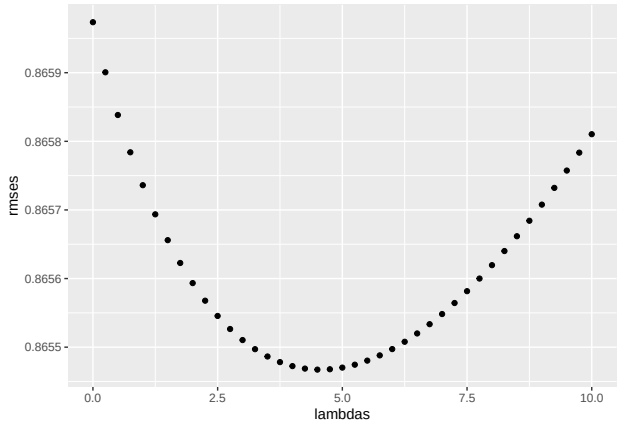
Regularization for movie and user effects

$$Y_{u,i} = \mu + \frac{1}{\lambda + n_i} b_i + \frac{1}{\lambda + n_u} b_u + \varepsilon_{u,i}$$

In this case we minimize $\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$.

```
# for movie and user effects
# choose lambda using cross validation
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train$rating)
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
```

```
  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  predicted_ratings <-
    test%>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred
  return(RMSE(test$rating, predicted_ratings))
})

qplot(lambdas, rmses)
```




```
lambda <- lambdas[which.min(rmses)]
lambda # 4.5 for movie and user effects
```

```
## [1] 4.5
```

```
# penalty term -> see above how it is chosen (cross validation)
# use lambda from above directly
mu <- mean(train$rating)
movie_reg_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

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

predicted_ratings <- test %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(user_reg_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

```
rmse.model5 <- RMSE(test$rating, predicted_ratings)
mae.model5 <- MAE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = rmse.model5, MAE = mae.model5 ))
```

Model using matrix factorization (and regularization)

With matrix factorization the rating matrix is approximated using two smaller matrices each with a reduced number of features. An user-feature and an item-feature matrix of lower dimension are used for prediction. Lower dimension means also less computational power is needed and results are generated faster.

This decomposition may detect latent features that are present in the rating data, but are not fully covered by user or movie effects (e.g. user-item interactions).

$R \approx P'Q$, where R is an $m \times n$ matrix, P is an $k \times n$ matrix and Q is an $k \times n$ matrix. The number of latent features is k.

The predicted rating for user u on item i is given by $p'_u q_i$, with the corresponding model:

$Y_{u,i} = p'_u q_i + \varepsilon_{u,i}$

The algorithm for matrix factorization makes use of stochastic gradient descent. Several parameters of the algorithm can be tuned for optimal performance: number of dimensions (latent features), learning rate, number of iterations and regularization parameters.

We will use the *recosystem* library (Qiu et al. (2021)) for this purpose. It was developed specifically for recommender systems applications using matrix factorization. Being a wrapper of C-language implementations, it is relatively fast and suitable for parallel computations, even in case of memory limitations.

```
# use recosystem library for matrix factorization and predictions
train_set = data_memory(user_index = train$userId, item_index = train$movieId,
                        rating = train$rating, index1 = TRUE)
test_set  = data_memory(user_index = test$userId, item_index = test$movieId,
                        rating = NULL, index1 = TRUE)
r = Reco()
# dim  -> number of latent factors: use 20, then tune
# r$train(train_set, opts = c(dim = 20, nthread = 1, niter = 20))
# Return results as R vector
# predicted_ratings <- r$predict(test_set, out_memory())
# get RMSE
# rmse.model6.notune <- RMSE(test$rating, predicted_ratings)
# rmse.model6.notune

set.seed(1, sample.kind="Rounding")
# model tuning for: number of latent factors (dim), learning rate (lrate)
opts = r$tune(train_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                     costp_l1 = 0, costq_l1 = 0,
                                     nthread = 1, niter = 10))
# opts
# opts$min
# use the tuned values for training
r$train(train_set, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9820   1.1047e+07
##    1       0.8758   8.9901e+06
##    2       0.8428   8.3399e+06
##    3       0.8209   7.9568e+06
##    4       0.8049   7.6980e+06
##    5       0.7927   7.5057e+06
##    6       0.7823   7.3610e+06
##    7       0.7732   7.2407e+06
##    8       0.7654   7.1409e+06
##    9       0.7588   7.0615e+06
##   10       0.7530   6.9947e+06
##   11       0.7479   6.9343e+06
##   12       0.7435   6.8868e+06
##   13       0.7394   6.8422e+06
##   14       0.7356   6.8030e+06
##   15       0.7323   6.7685e+06
##   16       0.7291   6.7370e+06
##   17       0.7263   6.7105e+06
##   18       0.7237   6.6879e+06
##   19       0.7212   6.6615e+06
```

```
# Return results as R vector
predicted_ratings <- r$predict(test_set, out_memory())
head(predicted_ratings, 10)
```

```
##  [1] 4.683247 4.160217 4.010381 3.944696 4.218243 3.765348 3.277071 3.947889
##  [9] 4.196535 4.102006
```

```
# r$model
# check for NA values
# sum(is.na(predicted_ratings))

# get RMSE
rmse.model6 <- RMSE(test$rating, predicted_ratings)
mae.model6 <- MAE(test$rating, predicted_ratings)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix Factorization Model",
                                     RMSE = rmse.model6, MAE = mae.model6))
```

Model using collaborative filtering

Collaborative filtering uses similarities measures between users or between items to select of predict ratings. For the prediction, only ratings of users/items from a neighborhood are considered. The main types of filtering are user-user and item-item respectively. Item-item is considered more stable and more efficient when the number of users is significantly lower than the number of items.

Due to high computational impact with large datasets (such as Movielens-10M) we only mention this approach here, but will not implement it or provide results for it. Helpful are the *recommenderlab* package and collaborative filtering literature (see Hahsler (2015), Falk (2019)).

Choosing the model for final validation

When we compare the RMSE for the used models, the best performing model turns out to be matrix factorization. We will use this for the final predictions on the validation (final holdout) data.

method	RMSE	MAE
Mean value	1.0611350	0.8563163
Movie Effects Model	0.9441568	0.7385547
Movie + User Effects Model	0.8262583	0.6368427
Regularized Movie Effects Model	0.9441152	0.7385385
Regularized Movie + User Effect Model	0.8654673	0.6699602
Matrix Factorization Model	0.7857092	0.6056591

Predictions on the final holdout data

Before making the predictions, the chosen model is re-trained on the entire edx data, in order to increase the information obtained from available data.

```
# evaluate the chosen model on the final hold-out validation set

# tune the model: retrain on full edx dataset, then test on validation data for final RMSE
# load the datasets
edx <- readRDS("edx.Rda")
validation <- readRDS("vali.Rda")
# create the training and test datasets
train_set = data_memory(user_index = edx$userId, item_index = edx$movieId,
                        rating = edx$rating, index1 = TRUE)
test_set  = data_memory(user_index = validation$userId, item_index = validation$movieId,
                        rating = NULL, index1 = TRUE)
```

```
r = Reco()
set.seed(1, sample.kind="Rounding")
# model tuning for: number of latent factors (dim), learning rate (lrate)
opts = r$tune(train_set, opts = list(dim = c(10, 20, 30), lrate = c(0.1, 0.2),
                                     costp_l1 = 0, costq_l1 = 0,
                                     nthread = 1, niter = 10))
# opts$min
# use the tuned values for training
r$train(train_set, opts = c(opts$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9725   1.2031e+07
##    1       0.8727   9.8837e+06
##    2       0.8391   9.1663e+06
##    3       0.8175   8.7558e+06
##    4       0.8021   8.4829e+06
##    5       0.7903   8.2813e+06
##    6       0.7803   8.1294e+06
##    7       0.7717   8.0050e+06
##    8       0.7645   7.9023e+06
##    9       0.7584   7.8225e+06
```

```
##   10       0.7531   7.7555e+06
##   11       0.7485   7.6946e+06
##   12       0.7444   7.6470e+06
##   13       0.7407   7.6025e+06
##   14       0.7373   7.5632e+06
##   15       0.7343   7.5298e+06
##   16       0.7314   7.4974e+06
##   17       0.7288   7.4711e+06
##   18       0.7264   7.4487e+06
##   19       0.7241   7.4222e+06
```

```
# Return results as R vector
predicted_ratings <- r$predict(test_set, out_memory())
# head(predicted_ratings, 10)
# save to file as well

# check for NA values
# sum(is.na(predicted_ratings))
# get RMSE
rmse.final.retrained <- RMSE(validation$rating, predicted_ratings)
mae.final.retrained <- MAE(validation$rating, predicted_ratings)
```

Now the RMSE and MAE can be computed for the matrix factorization model on the validation data:

```
rmse.final.retrained
```

```
## [1] 0.7825732
```

```
mae.final.retrained
```

```
## [1] 0.603092
```

Results

From the models listed and used, an RMSE of ~0.86 or below can be achieved using the user and movie effects model and, more clearly, the matrix factorization model. On the final validation data, the matrix factorization model generates an RMSE of 0.7825732.

Matrix factorization is effective, however it might be less easy to explain the detected features and translate them into known, business relevant factors.

Conclusion

We explored the Movielens-10M data and we tested different approaches for rating prediction. The Matrix factorization model proved very effective and delivered the lowest RMSE, with the (regularized) movie and user effects model as valuable alternative.

Although only briefly mentioned in this report, collaborative filtering could provide similar performance. The same might be true for other approaches like neural networks or models which include additional information about user preferences or movie content. Other topics related to recommender systems like data collection, real-time systems or cold start for new users were not discussed here, but they are important for any real-world implementation. Further suggestions on these can be found in Falk (2019).

References

Falk, K. 2019. *Practical Recommender Systems*. Manning Publications.

Hahsler, Michael. 2015. "Recommenderlab: A Framework for Developing and Testing Recommendation Algorithms."

Irizarry, R. A. 2019. *Introduction to Data Science: Data Analysis and Prediction Algorithms with r*. Chapman & Hall/CRC Data Science Series. CRC Press.

Qiu, Yixuan, Chih-Jen Lin, Yu-Chin Juan, Wei-Sheng Chin, Yong Zhuang, Bo-Wen Yuan, Meng-Yuan Yang, and Maintainer Yixuan Qiu. 2021. "Package 'Recosystem'."