Accuracy and Beyond

Data 612 Project 4

Paul Perez & William Outcault

30 June 2020

Contents

Introduction

Assignment

The goal of this assignment is give you practice working with accuracy and other recommender system metrics.

In this assignment you're asked to do at least one or (if you like) both of the following:

- Work in a small group, and/or
- Choose a different dataset to work with from your previous projects.

Deliverables

1. As in your previous assignments, compare the accuracy of at least two recommender system algorithms against your offline data.
2. Implement support for at least one business or user experience goal such as increased serendipity, novelty, or diversity.
3. Compare and report on any change in accuracy before and after you've made the change in #2.
4. As part of your textual conclusion, discuss one or more additional experiments that could be performed and/or metrics that could be evaluated only if online evaluation was possible. Also, briefly propose how you would design a reasonable online evaluation environment.

Our Recommender System

This is an elaboration of a previous project (source code found at the bottom). For this assignment we will pretend we work for a movie streaming platform. We have previous reports that show high customer turnover rates between competing movie streaming platforms.

This recommender system uses the `recommenderlab` package, and we'll load the MovieLense dataset. We can see that using the `class()` function, that this MovieLense dataset is a `realRatingMatrix`.

Our Goal:

Data shows high turnover rates between users and their choice of streaming platforms; meaning one bad recommendation and we may lose them as a customer. Our goal is to minimize these turnover rates by instilling trust through consistent and precise recommendations.

Our Approach:

Our approach will be to maximize precision by optimizing the choice of algorithms, evaluation schema methods, and parameters. This will be done using exploratory analysis and by evaluating a series of different models.

Data Exploration

```
data("MovieLense")
matrix <- as(MovieLense, "realRatingMatrix")
```

Using the `dim()` function, we can see the dimensions of the MovieLense matrix.

```
dim(matrix)
```

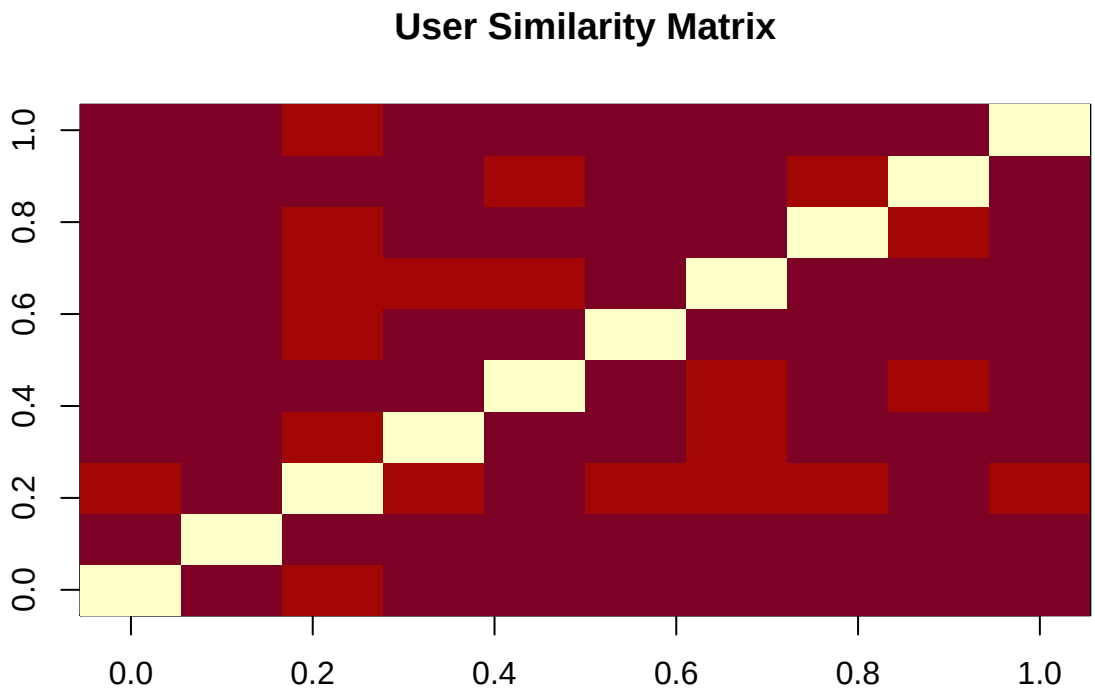
```
## [1]  943 1664
```

There are `943` users and `1664` movies.

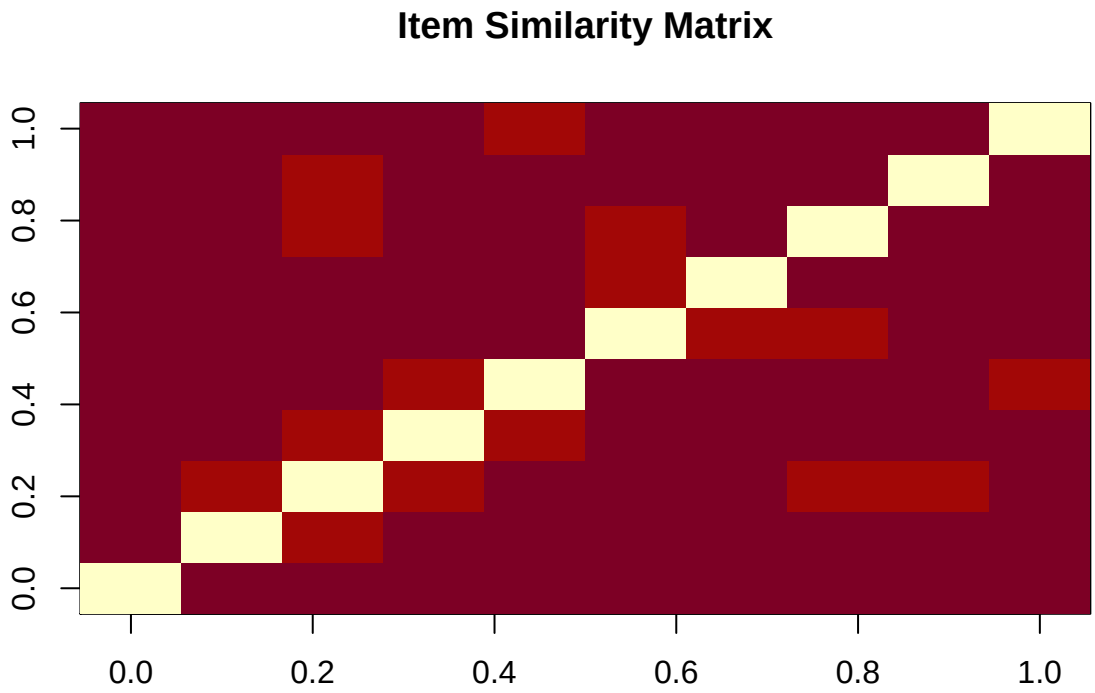
Similarity Matrices

The similarity matrix measures the similarity between item's or user's within the matrix. For the purpose of this project, we'll use the `cosine` method.

```
user_simularity <- similarity(matrix[1:10, ], method = "cosine", which = "users")
image(as.matrix(user_simularity), main = "User Similarity Matrix")
```



```
item_similarity <- similarity(matrix[, 1:10], method = "cosine", which = "items")
image(as.matrix(item_similarity), main = "Item Similarity Matrix")
```



The darker the cell in each of these similarity matrices show's the greater similarity between the two user's or item's.

There are some emperical studies that show greater performance with the `cosine` similarity method for item based recommendation systems where the `pearson` similarity method shows greater performance for user based recommendation systems.

Since we loaded the data as a `realRatingMatrix`, which is an S4 class, we can look further into components of the object using `slotNames()`

```
slotNames(matrix)
```

```
## [1] "data"      "normalize"
```

By adding `@data` to the `matrix` object, we'll be able to take a look at this class, which happens to be a `dgCMatrix` class that inherits from the original matrix.

```
class(matrix@data)
```

```
## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"
```

Table 1: Item Similarity Table

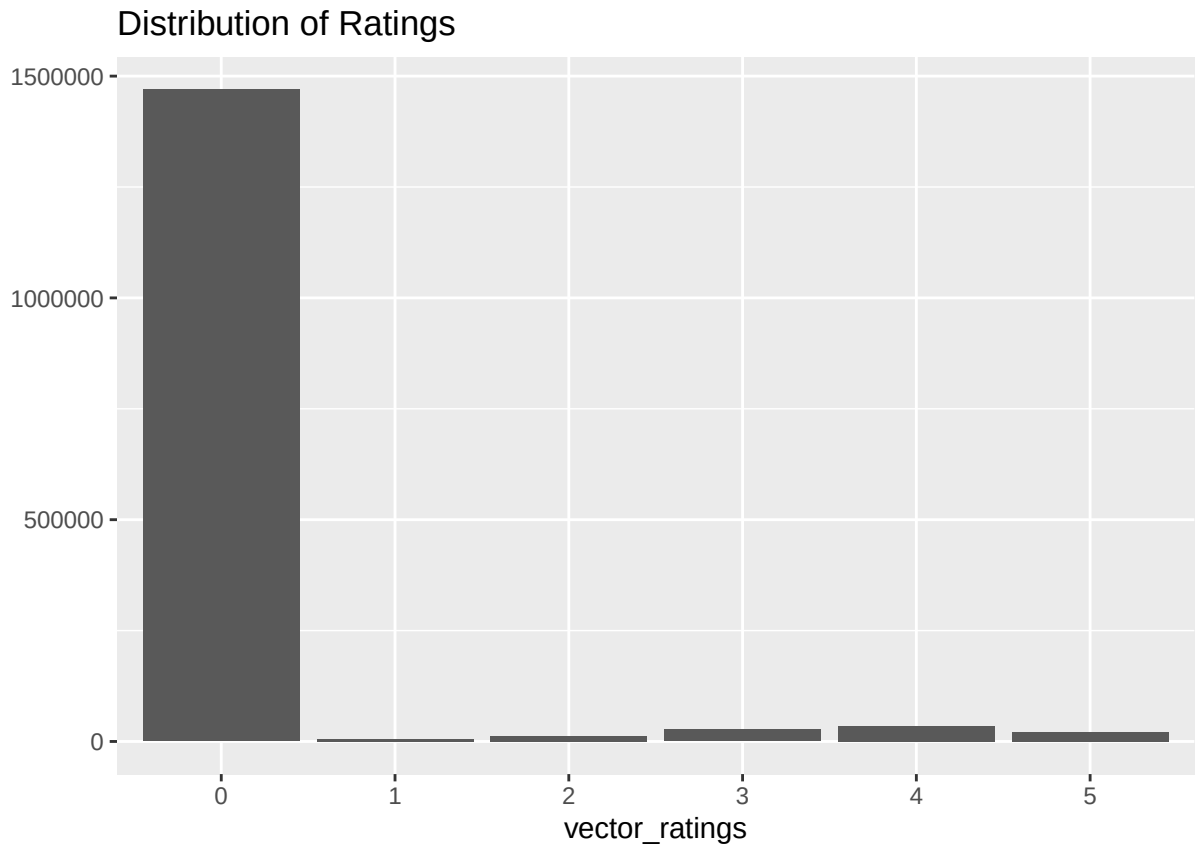
vector_ratings	Freq
0	1469760
1	6059
2	11307
3	27002
4	33947
5	21077

By converting the ratings to a vector, we can count the number of unique ratings available.

```
vector_ratings <- as.vector(matrix@data)
table_ratings <- table(vector_ratings)
table_ratings %>% kable(caption = "Item Similarity Table") %>% kable_styling("striped", full_width = TRU
```

Rating Distributions

```
vector_ratings <- factor(vector_ratings)
qplot(vector_ratings) + ggtitle("Distribution of Ratings")
```



Relevant Data

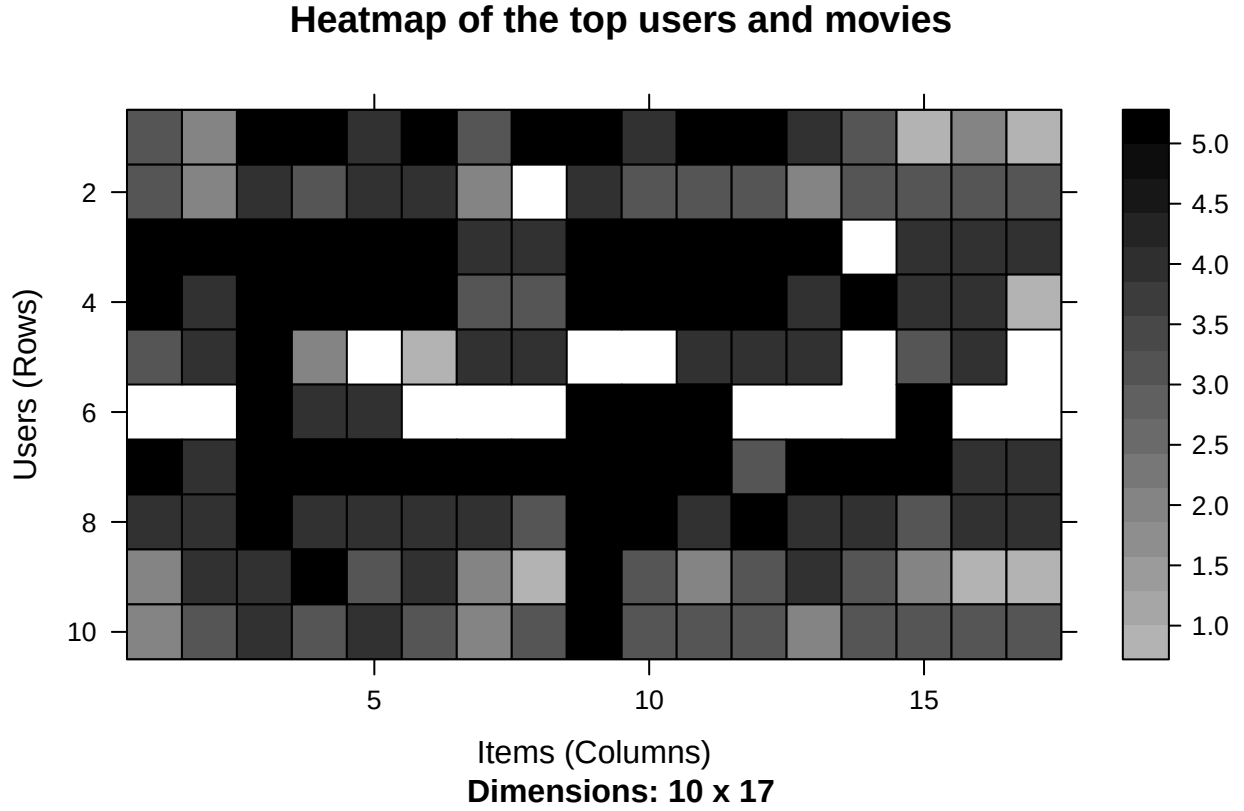
We can subset this data to select users who have rated at least 95 movies, and movies that have been watched at least 50 times. The dimension of the matrix changes from 943 x 1664 to 369 x 691

```
movie_ratings <- matrix[rowCounts(matrix) > 95, colCounts(matrix) > 50]
dim(movie_ratings)
```

```
## [1] 369 591
```

Top 1% of Users

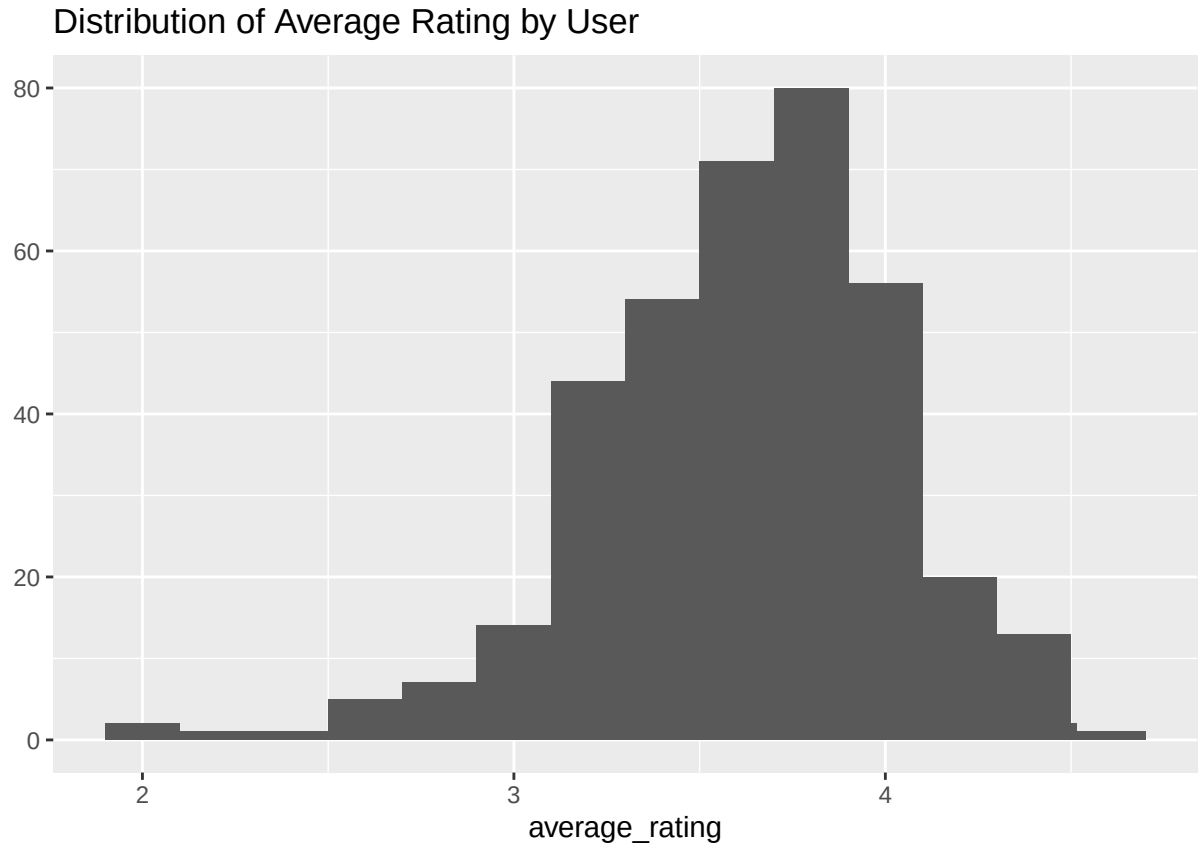
```
min_movies <- quantile(rowCounts(matrix, na.rm = TRUE), 0.99)
min_users <- quantile(colCounts(matrix, na.rm = TRUE), 0.99)
image(matrix[rowCounts(matrix) > min_movies, colCounts(matrix) > min_users], main = "Heatmap of the top
```



Average Rating (User) Distribution

```
average_rating <- rowMeans(movie_ratings)
qplot(average_rating) + stat_bin(binwidth = 0.2) + ggtitle ("Distribution of Average Rating by User")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Evaluation Schemas

We will proceed by testing the performance using algorithms so far discussed this semester, in addition to different evaluation schema methods.

- Split
- Cross-validation
- Bootstrapping

Before discussing algorithms we will go through each evaluation scheme method.

Split

We begin by using `recommenderlab`'s split method to sample 80% of the samples rows for training and 20% for testing.

```
items_to_keep <- 15
rating_threshold <- 3
n_fold <- 4
set.seed(123)
split_eval_sets <- evaluationScheme(data = movie_ratings, method = "split",
                                    given = items_to_keep, goodRating = rating_threshold, k = 1, train=
```

```
users_in_training <- nrow(getData(split_eval_sets, "train")) / nrow(movie_ratings)
print(paste("Percentage of users in the training set: ",round(users_in_training, 2)*100,"%", sep=""))
```

```
## [1] "Percentage of users in the training set: 80%"
```

```
users_in_testing <- nrow(getData(split_eval_sets, "known")) / nrow(movie_ratings)
print(paste("Percentage of users in the testing set: ",round(users_in_testing,2)*100,"%", sep=""))
```

```
## [1] "Percentage of users in the testing set: 20%"
```

Cross-Validation

Next we will use cross-validation method to create models using k-fold validation sets. The samples were 75/25 for training/testing respectively.

```
cv_eval_sets <- evaluationScheme(data = movie_ratings, method = "cross-validation",
                                 given = items_to_keep, goodRating = rating_threshold,
                                 k = n_fold, train=0.80)
```

```
users_in_training <- nrow(getData(cv_eval_sets, "train")) / nrow(movie_ratings)
print(paste("Percentage of users in the training set: ",round(users_in_training,
                                                              2)*100,"%", sep=""))
```

```
## [1] "Percentage of users in the training set: 75%"
```

```
users_in_testing <- nrow(getData(cv_eval_sets, "known")) / nrow(movie_ratings)
print(paste("Percentage of users in the testing set: ",round(users_in_testing,
                                                             2)*100,"%", sep=""))
```

```
## [1] "Percentage of users in the testing set: 25%"
```

Bootstrap

We will implement `recommenderlab`'s bootstrapping method. The difference will be a larger test set if we continue to use 80% of rows for the training, because bootstrapping samples the rows with replacement allowing us to sample one user more than once.

```
bs_eval_sets <- evaluationScheme(data = movie_ratings, method = "bootstrap",
                                 given = items_to_keep, goodRating = rating_threshold,
                                 k = 1, train = 0.80)
```

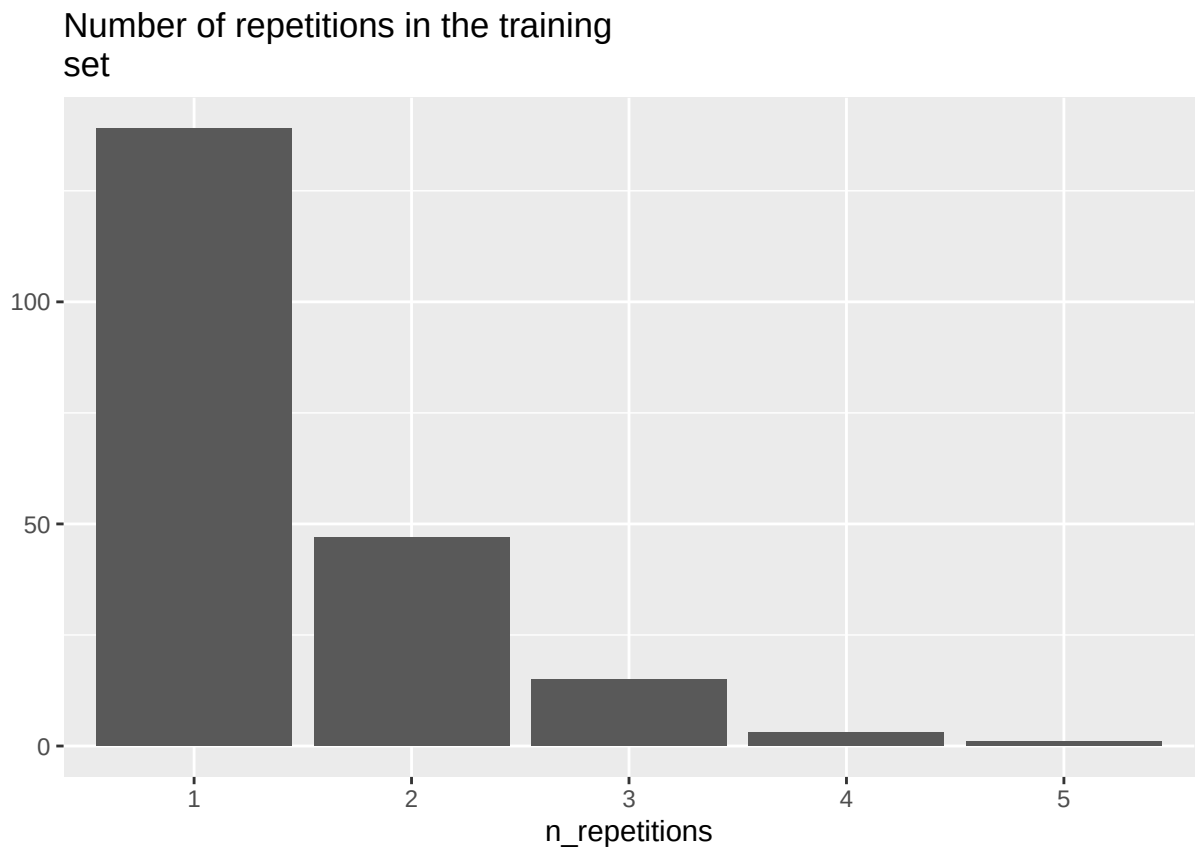
```
users_in_training <- nrow(getData(bs_eval_sets, "train")) / nrow(movie_ratings)
print(paste("Percentage of users in the training set: ",round(users_in_training,
                                                              2)*100,"%", sep=""))
```

```
## [1] "Percentage of users in the training set: 80%"
```

```
users_in_testing <- nrow(getData(bs_eval_sets, "known")) / nrow(movie_ratings)
print(paste("Percentage of users in the testing set: ",round(users_in_testing,
                                                             2)*100,"%", sep=""))
```

```
## [1] "Percentage of users in the testing set: 44%"
```

```
table_train <- table(bs_eval_sets@runsTrain[[1]])
n_repetitions <- factor(as.vector(table_train))
qplot(n_repetitions) + ggtitle("Number of repetitions in the training
set")
```



Algorithms

Next we will use different algorithms with different parameters to optimize model performance. The algorithms will include:

- IBCF using Cosine
- IBCF using Pearson
- UBCF using Cosine
- UBCF using Pearson
- SVD
- Random (Our baseline)

All model performances will be compared with our random algorithm which is the baseline for model performance. The performance for each model will be compared with one another, and overall performance will be compared between the different evaluation scheme methods described earlier.

```
# Algorithms to be tested
algorithms <- list(
  "IBCF_Cos" = list(name = "IBCF", param = list(method = "cosine")),
  "IBCF_Pear" = list(name = "IBCF", param = list(method = "pearson")),
  "UBCF_Cos" = list(name = "UBCF", param = list(method = "cosine")),
  "UBCF_Pear" = list(name = "UBCF", param = list(method = "pearson")),
  "SVD" = list(name = "SVD"),
  "Random" = list(name = "Random", param = NULL)
```

```
  )
# Store esults for each ES method
split_results <- evaluate(x = split_eval_sets, method = algorithms, n = c(10, 20, 30, 50, 70, 100))
cross_val_results <- evaluate(x = cv_eval_sets, method = algorithms, n = c(10, 20, 30, 50, 70, 100))
bootstrap_results <- evaluate(x = bs_eval_sets, method = algorithms, n = c(10, 20, 30, 50, 70, 100))
```

Now that we have our evaluation results list for each method we can sum up indices to take account of all splits at the same time.

```
# Function to convert results to table with consilidated indices.
avg_conf_matr <- function(results) {
  tmp <- results %>%
    getConfusionMatrix() %>%
    as.list()
    as.data.frame(Reduce("+",tmp) / length(tmp)) %>%
    mutate(n = c(10, 20, 30, 50, 70, 100)) %>%
    select('n', 'precision', 'recall', 'TPR', 'FPR')
}
```

Results

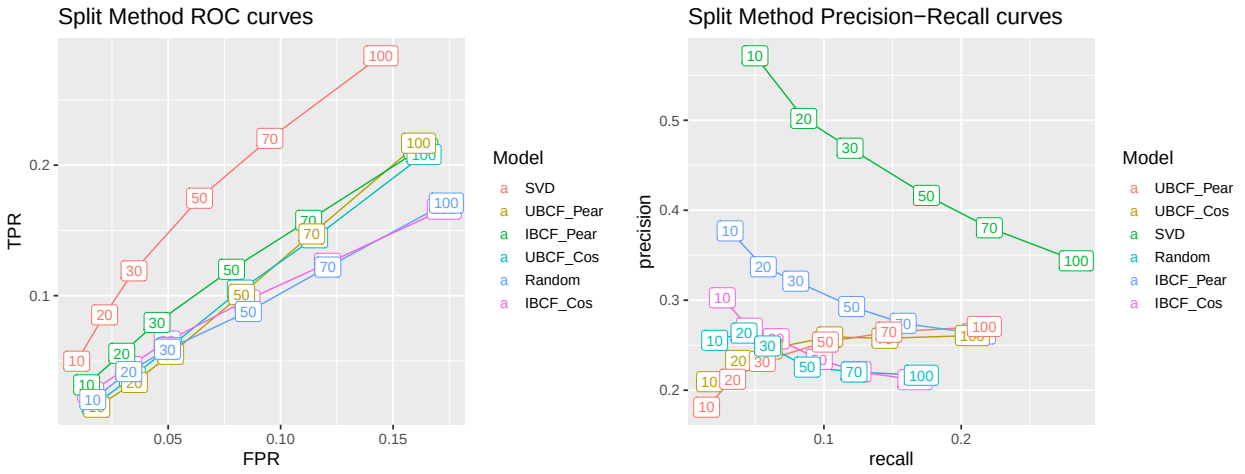
Results from Split Method

```
split_results_tbl <- split_results %>% map(avg_conf_matr) %>% enframe()
split_results_tbl <- unnest(split_results_tbl, colnames(split_results_tbl))
print(head(split_results_tbl))
```

```
## # A tibble: 6 x 6
##   name         n precision recall    TPR    FPR
##   <chr>    <dbl>     <dbl>  <dbl>  <dbl>  <dbl>
## 1 IBCF_Cos    10     0.303 0.0260 0.0260 0.0155
## 2 IBCF_Cos    20     0.269 0.0456 0.0456 0.0325
## 3 IBCF_Cos    30     0.257 0.0652 0.0652 0.0495
## 4 IBCF_Cos    50     0.234 0.0966 0.0966 0.0850
## 5 IBCF_Cos    70     0.220 0.125  0.125  0.121
## 6 IBCF_Cos   100     0.212 0.166  0.166  0.173
```

```
par(mfrow=c(1,2))
# Plot FPR TPR Split Method Results
split_results_tbl %>%
  ggplot(aes(FPR, TPR, color = fct_reorder2(as.factor(name), FPR, TPR))) +
  geom_line() +
  geom_label(aes(label = n)) +
  labs(title = "Split Method ROC curves", color = "Model") +
  theme_grey(base_size = 14)
# Plot Recall Precision Split Method Results
split_results_tbl %>%
  ggplot(aes(recall, precision, color = fct_reorder2(as.factor(name),  precision, recall))) +
  geom_line() +
```

```
  geom_label(aes(label = n)) +
  labs(title = "Split Method Precision-Recall curves", color = "Model") +
  theme_grey(base_size = 14)
```

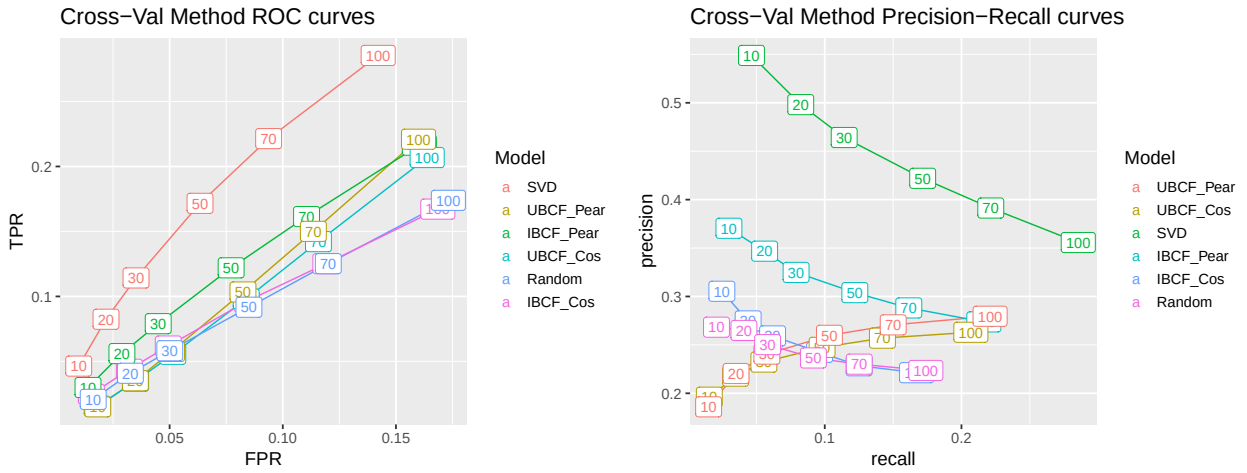


Results from Cross-Val Method

```
cv_results_tbl <- cross_val_results %>% map(avg_conf_matr) %>% enframe()
cv_results_tbl <- unnest(cv_results_tbl, colnames(cv_results_tbl))
print(head(cv_results_tbl))
```

```
## # A tibble: 6 x 6
##   name          n precision recall    TPR    FPR
##   <chr>    <dbl>     <dbl>  <dbl>  <dbl>  <dbl>
## 1 IBCF_Cos    10     0.305 0.0247 0.0247 0.0154
## 2 IBCF_Cos    20     0.275 0.0440 0.0440 0.0322
## 3 IBCF_Cos    30     0.259 0.0618 0.0618 0.0494
## 4 IBCF_Cos    50     0.242 0.0962 0.0962 0.0840
## 5 IBCF_Cos    70     0.228 0.125  0.125  0.119
## 6 IBCF_Cos   100     0.221 0.167  0.167  0.169
```

```
# Plot FPR TPR Cross-Val Method Results
cv_results_tbl %>%
  ggplot(aes(FPR, TPR, color = fct_reorder2(as.factor(name), FPR, TPR))) +
  geom_line() + geom_label(aes(label = n)) +
  labs(title = "Cross-Val Method ROC curves", color = "Model") +
  theme_grey(base_size = 14)
# Plot Recall Precision Cross-Val Method Results
cv_results_tbl %>%
  ggplot(aes(recall, precision, color = fct_reorder2(as.factor(name),  precision, recall))) +
  geom_line() +
  geom_label(aes(label = n)) +
  labs(title = "Cross-Val Method Precision-Recall curves", color = "Model") +
  theme_grey(base_size = 14)
```

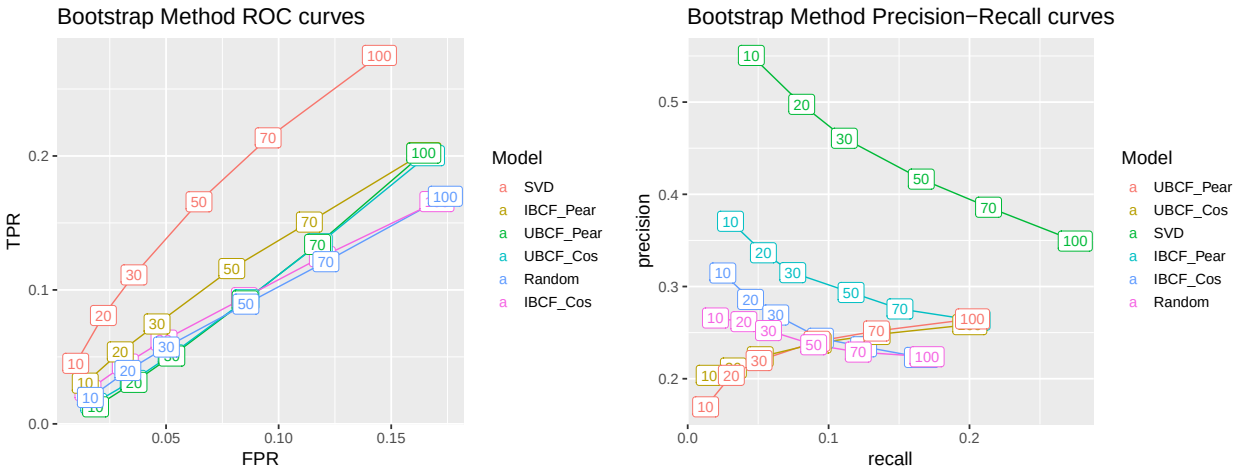


Results from Bootstrap Method

```
bs_results_tbl <- bootstrap_results %>% map(avg_conf_matr) %>% enframe()
bs_results_tbl <- unnest(bs_results_tbl, colnames(bs_results_tbl))
print(head(bs_results_tbl))
```

```
## # A tibble: 6 x 6
##   name          n precision recall    TPR    FPR
##   <chr>    <dbl>     <dbl>  <dbl>  <dbl>  <dbl>
## 1 IBCF_Cos    10     0.315 0.0249 0.0249 0.0153
## 2 IBCF_Cos    20     0.286 0.0446 0.0446 0.0319
## 3 IBCF_Cos    30     0.269 0.0625 0.0625 0.0491
## 4 IBCF_Cos    50     0.244 0.0942 0.0942 0.0848
## 5 IBCF_Cos    70     0.235 0.125  0.125  0.119
## 6 IBCF_Cos   100     0.223 0.166  0.166  0.170
```

```
# Plot FPR TPR Bootstrap Method Results
bs_results_tbl %>%
  ggplot(aes(FPR, TPR, color = fct_reorder2(as.factor(name), FPR, TPR))) +
  geom_line() + geom_label(aes(label = n)) +
  labs(title = "Bootstrap Method ROC curves", color = "Model") +
  theme_grey(base_size = 14)
# Plot Recall Precision Bootstrap Method Results
bs_results_tbl %>%
  ggplot(aes(recall, precision, color = fct_reorder2(as.factor(name),precision, recall))) +
  geom_line() +
  geom_label(aes(label = n)) +
  labs(title = "Bootstrap Method Precision-Recall curves", color = "Model") +
  theme_grey(base_size = 14)
```


Bootstrap Method ROC curves
TPR
FPR
0.0
0.1
0.2
0.05
0.10
0.15
Model
SVD
IBCF_Pear
UBCF_Pear
UBCF_Cos
Random
IBCF_Cos
Bootstrap Method Precision−Recall curves
precision
recall
0.2
0.3
0.4
0.5
0.0
0.1
0.2
Model
UBCF_Pear
UBCF_Cos
SVD
IBCF_Pear
IBCF_Cos
Random

Conclusion

To reiterate, our goal is to maximize precision in order to reduce customer turnover rates. We performed exploratory analysis, split the data using three different evaluation schema methods, created six different models for each schema, and plotted our results. In conclusion, by using User-Based Collaborative Filtering with the pearson method we can maximize precision and minimize customer turnover rates.

Further Experiments

I would love to conduct further experiments to calculate diversity using online evaluations. The main priority in our project was to maximize precision so that we could confidently suggest a movie that the customer would like.

However including a section that would help less popular movies be discovered would be a great feature for a multitude of reasons. True movie lovers could explore outside of the mainstream and it would keep from stagnant recommendations that I personally see all the time from streaming platforms.

Source Code: GitHub Repository - William Outcault

GitHub Repository - Paul Perez

Continuation of: GitHub Repository