Movie Lens In House Recommender Systems

Anil Akyildirim & Sie Siong Wong
DATA 612
Summer 2020

Introduction

What is requested?

- Create Multiple Recommender Systems using Movie Lens Data Set
- Create Recommendation and Evaluation to pick a system with the best performance.

Our approach solving it

- Data Collection and Preparation
- Data Exploration
- Non-Distributed Recommender System
  - Recommender Development & Evaluation
- Distributed Recommender Systems
  - Recommender Development
  - Recommender Evaluation
- Conclusion

Data Collection

```r
# Load movies and ratings datasets
movies <- fread("https://raw.githubusercontent.com/SieSiongWong/DATA-612/master/movies.csv")

ratings <- fread("https://raw.githubusercontent.com/SieSiongWong/DATA-612/master/ratings_1m.csv")

head(movies)
```

```
##    movieId                              title
## 1:       1                   Toy Story (1995)
## 2:       2                     Jumanji (1995)
## 3:       3            Grumpier Old Men (1995)
## 4:       4           Waiting to Exhale (1995)
## 5:       5 Father of the Bride Part II (1995)
## 6:       6                        Heat (1995)
##                                          genres
## 1: Adventure|Animation|Children|Comedy|Fantasy
## 2:                   Adventure|Children|Fantasy
## 3:                               Comedy|Romance
## 4:                         Comedy|Drama|Romance
## 5:                                       Comedy
## 6:                        Action|Crime|Thriller
```

```
##    userId movieId rating
## 1:      1    1193      5
## 2:      1     661      3
## 3:      1     914      3
## 4:      1    3408      4
## 5:      1    2355      5
## 6:      1    1197      3
```

- The ratings dataset has 1 million ratings from 6000 users on 9743 movies.

Data Exploration and Preparation

```r
# Summary of movies and ratings datasets
str(movies)
```

```
## Classes 'data.table' and 'data.frame':   9742 obs. of  3 variables:
##  $ movieId: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ title  : chr  "Toy Story (1995)" "Jumanji (1995)" "Grumpier Old Men (1995)" "Waiting to Exhale (1995)"
...
##  $ genres : chr  "Adventure|Animation|Children|Comedy|Fantasy" "Adventure|Children|Fantasy" "Comedy|Romanc
e" "Comedy|Drama|Romance" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```r
str(ratings)
```

```
## Classes 'data.table' and 'data.frame':   1000209 obs. of  3 variables:
##  $ userId : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId: int  1193 661 914 3408 2355 1197 1287 2804 594 919 ...
##  $ rating : int  5 3 3 4 5 3 5 5 4 4 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

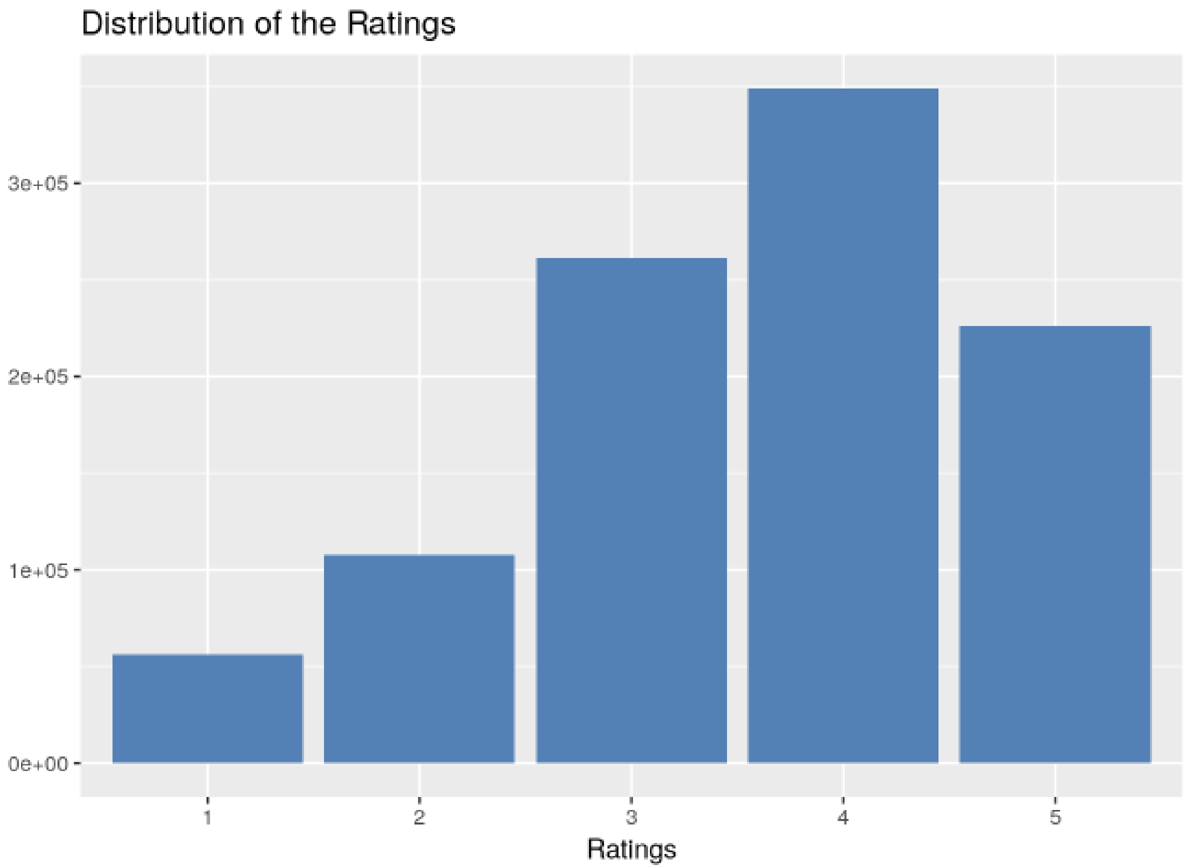
```r
# Statistical summary of rating variable
describe(ratings$rating)
```

```
## ratings$rating
##        n  missing distinct     Info     Mean      Gmd
##  1000209        0        5    0.927    3.582    1.219
##
## lowest : 1 2 3 4 5, highest: 1 2 3 4 5
##
## Value           1      2      3      4      5
## Frequency   56174 107557 261197 348971 226310
## Proportion  0.056  0.108  0.261  0.349  0.226
```

```r
# Convert to rating matrix
ratings_matrix <- dcast(ratings, userId~movieId, value.var = "rating", na.rm = FALSE)

# Remove user Id column
ratings_matrix <- as.matrix(ratings_matrix[,-1])

# Convert rating matrix into a recommenderlab sparse matrix
ratings_matrix <- as(ratings_matrix, "realRatingMatrix")

ratings_matrix
```

```
6040 x 3706 rating matrix of class ‘realRatingMatrix’ with 1000209
ratings.
```

Data Exploration



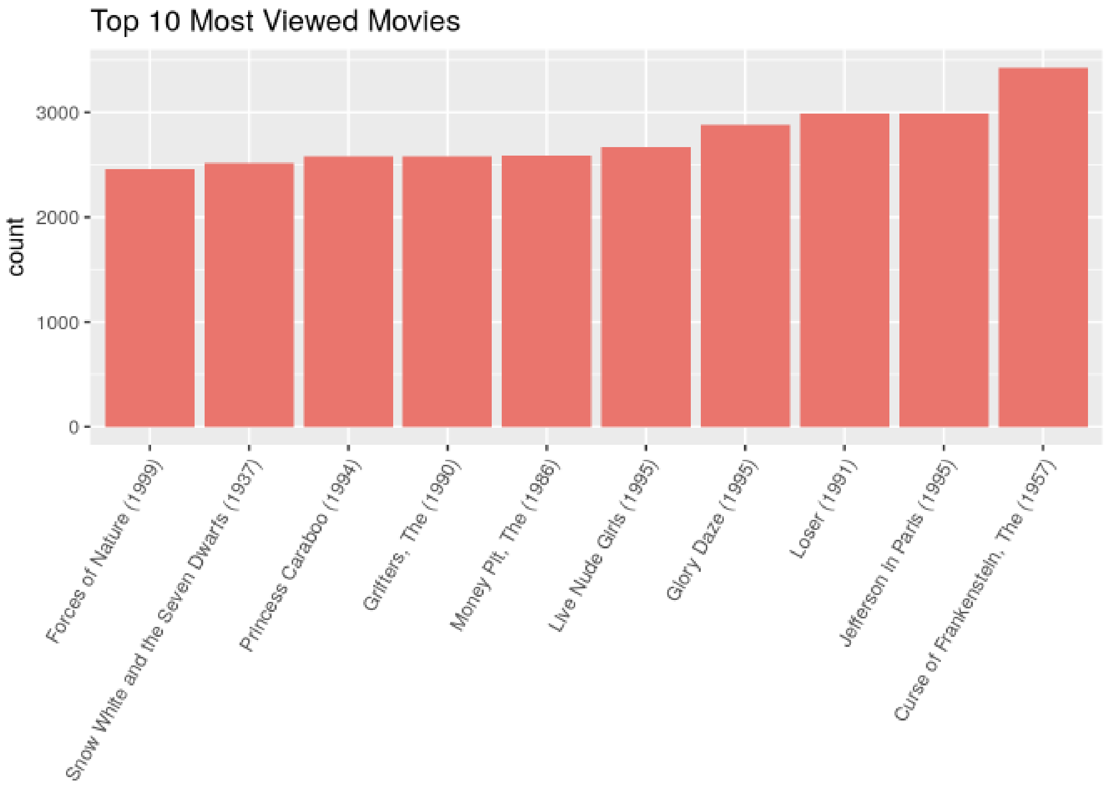

```r
# Convert the ratings matrix into a vector
vec_ratings <- as.vector(ratings_matrix@data)

# Unique ratings
unique(vec_ratings)
```

```r
# Count the occurrences for each rating
table_ratings <- table(vec_ratings)

table_ratings
```

```r
# Remove zero rating and convert the vector to factor
vec_ratings <- vec_ratings[vec_ratings != 0] %>% factor()

# Visualize through qqplot
qplot(vec_ratings, fill = I("steelblue")) +
  ggtitle("Distribution of the Ratings") +
  labs(x = "Ratings")
```

- Majority of the movies get rating 4.

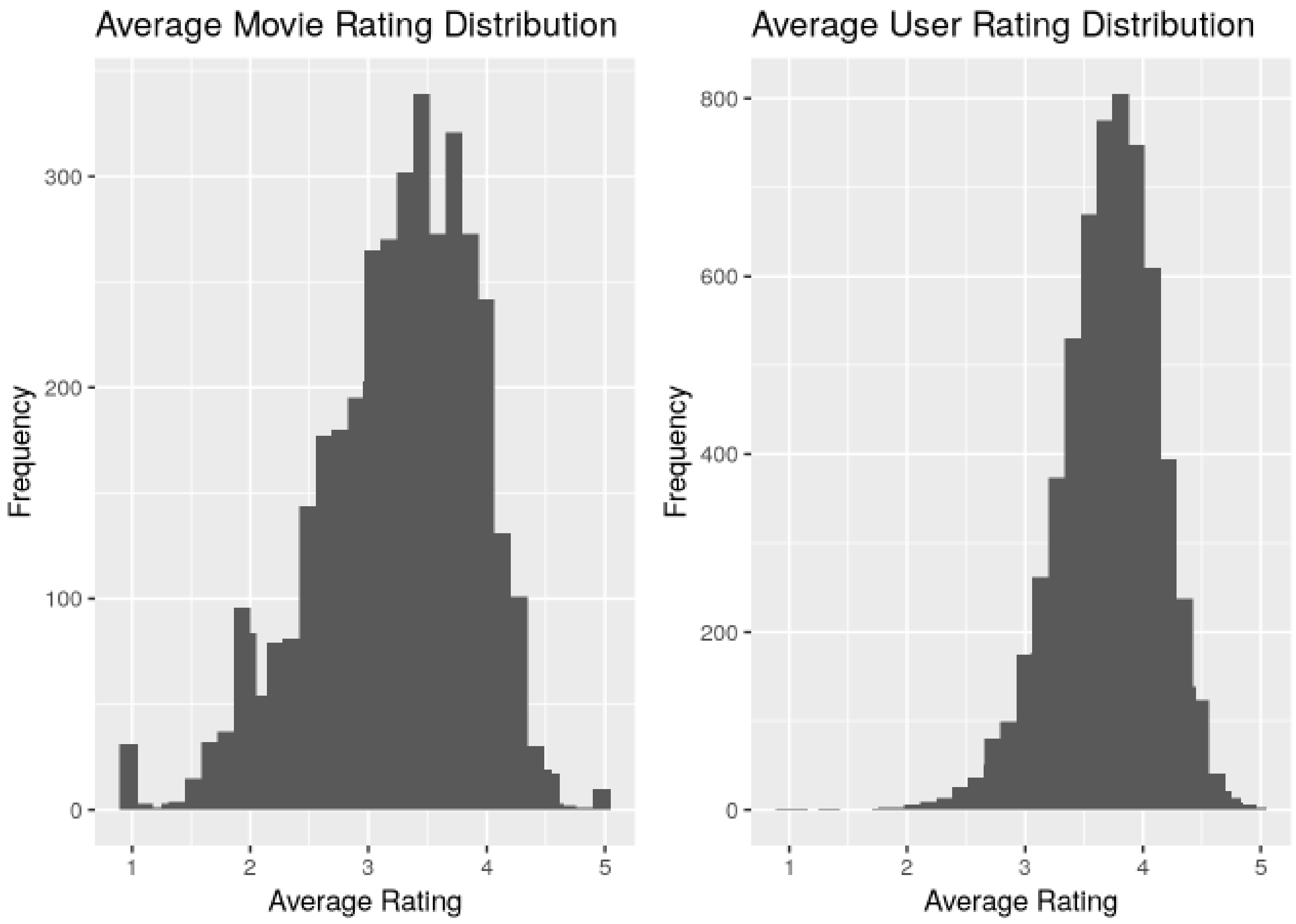
Data Exploration




```r
# Search for the top 10 most viewed movies
most_views <- colCounts(ratings_matrix) %>% melt()

most_views <- tibble::rowid_to_column(most_views, "movieId")
names(most_views)[2] <- 'count'
most_views <- most_views %>%
  merge(movies, by = "movieId") %>%
  top_n(count, n = 10)

# Visualize the top 10 most viewed movies
ggplot(most_views, aes(x = reorder(title, count), y = count, fill = 'lightblue')) +
  geom_bar(stat = "identity") +
  theme(axis.text.x =element_text(angle = 60, hjust = 1)) +
  ggtitle("Top 10 Most Viewed Movies") +
  theme(legend.position = "none", axis.title.x = element_blank())
```

- Curse of Frankenstein is the highest Viewed Movies

Data Exploration




```r
# Average rating for each movie
avg_ratings_mv <- colMeans(ratings_matrix)

# Average rating for each user
avg_ratings_us <- rowMeans(ratings_matrix)

# Visualize the distribution of the average movie rati
avg1 <- qplot(avg_ratings_mv) +
  stat_bin(binwidth = 0.1) +
  ggtitle("Average Movie Rating Distribution") +
  labs(x = 'Average Rating', y = 'Frequency')

# Visualize the distribution of the average user ratin
avg2 <- qplot(avg_ratings_us) +
  stat_bin(binwidth = 0.1) +
  ggtitle("Average User Rating Distribution") +
  labs(x = 'Average Rating', y = 'Frequency')

# Compare the average rating distribution plots
grid.arrange(avg1, avg2, nrow = 1)
```

- Some movies have only few ratings and some users only rated few movies.
- To avoid bias we remove the least watched movies and least rated users with a threshold of minimum number such as 50

Data Exploration


```r
# Filter users and movies more than 50
ratings_matrix <- ratings_matrix[rowCounts(ratings_matrix) > 50, colCounts(ratings_matrix) > 50]

# Average rating for each movie
avg_ratings_mv2 <- colMeans(ratings_matrix)

# Average rating for each user
avg_ratings_us2 <- rowMeans(ratings_matrix)

# Visualize the distribution of the average movie rating
avg3 <- qplot(avg_ratings_mv2) +
  stat_bin(binwidth = 0.1) +
  ggtitle("Average Movie Rating Distribution") +
  labs(x = 'Average Rating', y = 'Frequency')

# Visualize the distribution of the average user rating
avg4 <- qplot(avg_ratings_us2) +
  stat_bin(binwidth = 0.1) +
  ggtitle("Average User Rating Distribution") +
  labs(x = 'Average Rating', y = 'Frequency')

# Compare the average rating distribution plots
grid.arrange(arrangeGrob(avg1, avg2, ncol = 1, top=textGrob("Before")), arrangeGrob(avg3, avg4, ncol = 1, top=
textGrob("After")), ncol = 2)
```

- After filtering and removing the least watched and least rated movies we see the curve is narrower and there is less variance.

Define Recommender Method

```r
knitr::opts_chunk$set(cache=TRUE)

# Setup the evaluation scheme
evaluation <- evaluationScheme(ratings_matrix,
                                method     = "cross",
                                k          = 5,
                                train      = 0.8,
                                given      = 10,
                                goodRating = 3
                              )

evaluation
```

- Focus on UBCF, SVDF and ALS with Recommenderlab

```r
# Set up list of algorithms
algorithms <- list(
  "User-Based CF"              = list(name  = "UBCF", parameter = list(method = "cosine", nn = 25
  "Funk SVD"                   = list(name  = "SVDF", parameter = list(k = 10)),
  "Alternating Least Squares"  = list(name  = "ALS")
                 )

# Estimate the models with top N recommendation lists
results <- evaluate(evaluation,
                    algorithms,
                    type  = "topNList",
                    n     = c(1, 3, 5, 10, 15, 20)
                   )
```

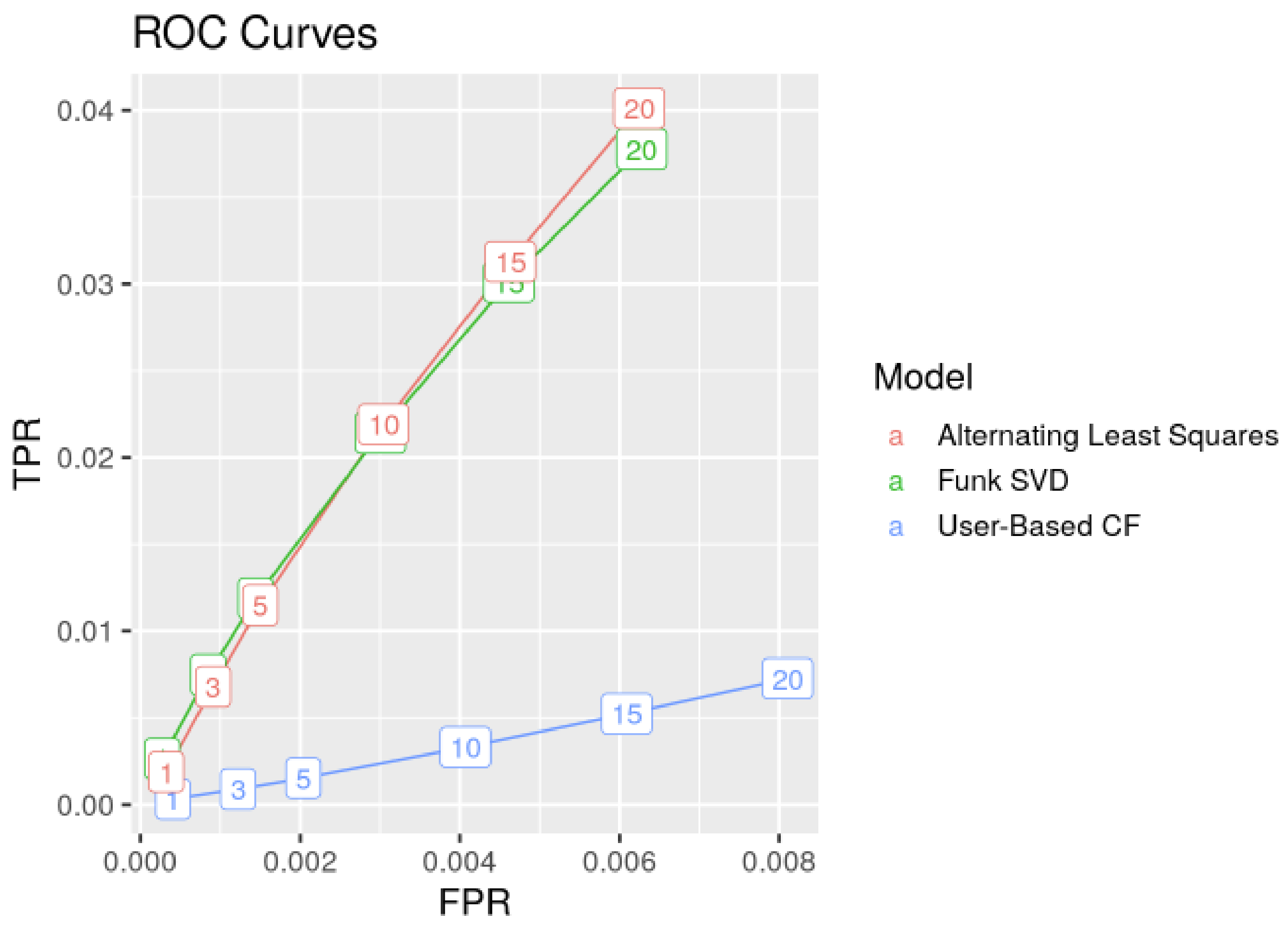
Recommender Method Evaluation

```r
# Create a function to get average of precision, recall, TPR, FPR
avg_cf_matrix <- function(results) {
avg <- results %>%
  getConfusionMatrix()  %>%
  as.list()
  as.data.frame( Reduce("+", avg) / length(avg)) %>%
  mutate(n = c(1, 3, 5, 10, 15, 20)) %>%
  select('n', 'precision', 'recall', 'TPR', 'FPR')
}

# Using map() to iterate the avg function across both models
results_tbl <- results %>% map(avg_cf_matrix) %>% enframe() %>% unnest()
```

```r
results_tbl
```

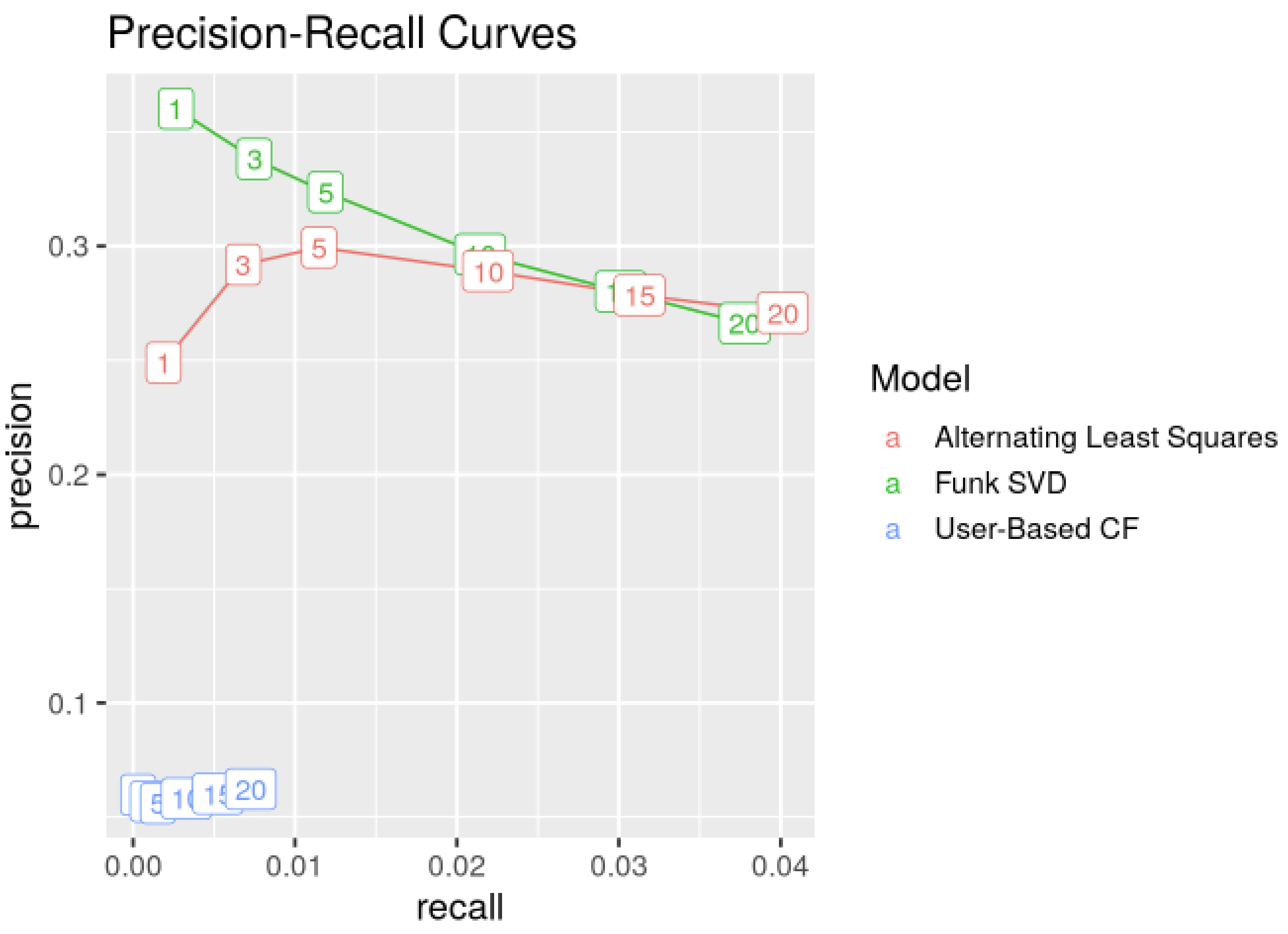
```
## # A tibble: 18 x 6
##    name                          n precision   recall      TPR      FPR
##    <chr>                     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
##  1 User-Based CF                 1    0.0597 0.000314 0.000314 0.000406
##  2 User-Based CF                 3    0.0568 0.000894 0.000894 0.00122
##  3 User-Based CF                 5    0.0560 0.00152  0.00152  0.00204
##  4 User-Based CF                10    0.0581 0.00331  0.00331  0.00407
##  5 User-Based CF                15    0.0602 0.00524  0.00524  0.00609
##  6 User-Based CF                20    0.0622 0.00726  0.00726  0.00810
##  7 Funk SVD                      1    0.360  0.00265  0.00265  0.000272
##  8 Funk SVD                      3    0.338  0.00748  0.00748  0.000846
##  9 Funk SVD                      5    0.324  0.0119   0.0119   0.00144
## 10 Funk SVD                     10    0.296  0.0214   0.0214   0.00300
## 11 Funk SVD                     15    0.280  0.0301   0.0301   0.00461
## 12 Funk SVD                     20    0.266  0.0378   0.0378   0.00627
## 13 Alternating Least Squares     1    0.249  0.00188  0.00188  0.000322
## 14 Alternating Least Squares     3    0.292  0.00680  0.00680  0.000910
## 15 Alternating Least Squares     5    0.299  0.0115   0.0115   0.00150
## 16 Alternating Least Squares    10    0.289  0.0219   0.0219   0.00304
## 17 Alternating Least Squares    15    0.279  0.0313   0.0313   0.00463
## 18 Alternating Least Squares    20    0.271  0.0401   0.0401   0.00624
```

Recommender Method Evaluation




```r
# Plot ROC curves for each model
results_tbl %>%
  ggplot(aes(FPR, TPR, color = fct_reorder2(as.factor(name), FPR, TPR))) +
  geom_line() +
  geom_label(aes(label = n))  +
  labs(title = "ROC Curves", color = "Model") +
  theme_grey(base_size = 14)
```

Recommender Method Evaluation




```r
# Plot Precision-Recall curves for each model
results_tbl %>%
  ggplot(aes(recall, precision, color = fct_reorder2(as.factor(name), recall, precision))) +
  geom_line() +
  geom_label(aes(label = n))  +
  labs(title = "Precision-Recall Curves", colour = "Model") +
  theme_grey(base_size = 14)
```

- ALS has the highest precision and sensitivity.

Recommender Model Improvement– ALS Optimization

```
## ALS run fold/sample [model time/prediction time]
##   1  [0.001sec/26.108sec]
##   2  [0.001sec/26.172sec]
##   3  [0sec/26.28sec]
##   4  [0sec/26.225sec]
##   5  [0sec/25.273sec]
## ALS run fold/sample [model time/prediction time]
##   1  [0sec/26.708sec]
##   2  [0.005sec/28.541sec]
##   3  [0.001sec/28.169sec]
##   4  [0.001sec/28.367sec]
##   5  [0sec/27.338sec]
## ALS run fold/sample [model time/prediction time]
##   1  [0.001sec/28.21sec]
##   2  [0.001sec/27.624sec]
##   3  [0.006sec/27.646sec]
##   4  [0.007sec/27.343sec]
##   5  [0.007sec/26.892sec]
## ALS run fold/sample [model time/prediction time]
##   1  [0.007sec/35.459sec]
##   2  [0.006sec/36.177sec]
##   3  [0.006sec/35.912sec]
##   4  [0.007sec/36.003sec]
##   5  [0.006sec/27.868sec]
## ALS run fold/sample [model time/prediction time]
##   1  [0.001sec/72.424sec]
##   2  [0.001sec/74.817sec]
##   3  [0.006sec/42.183sec]
##   4  [0.006sec/41.797sec]
##   5  [0.006sec/40.444sec]
## ALS run fold/sample [model time/prediction time]
##   1  [0.006sec/1322.911sec]
##   2  [0sec/2708.106sec]
##   3  [0.006sec/2355.103sec]
##   4  [0.007sec/2348.893sec]
##   5  [0.006sec/2348.442sec]
```

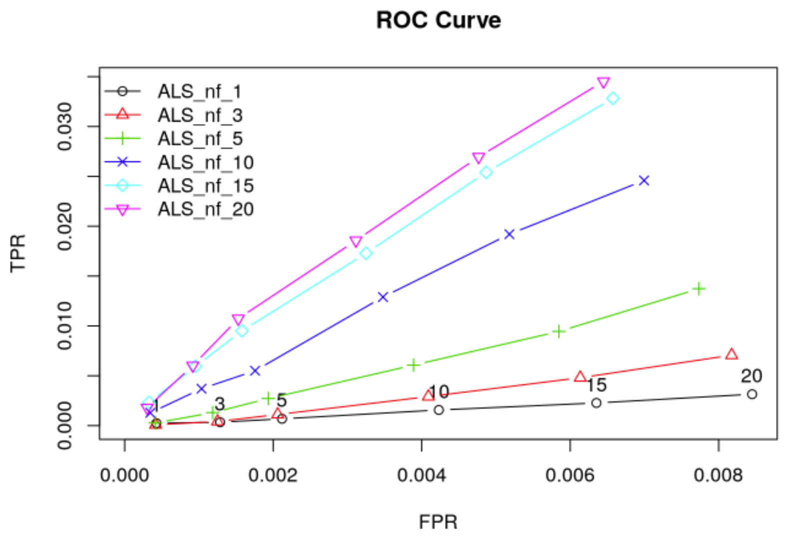
- Default n value is 10

```
## $normalize
## NULL
##
## $lambda
## [1] 0.1
##
## $n_factors
## [1] 10
##
## $n_iteratio
## [1] 10
##
## $min_item_n
## [1] 1
##
## $seed
## NULL
```

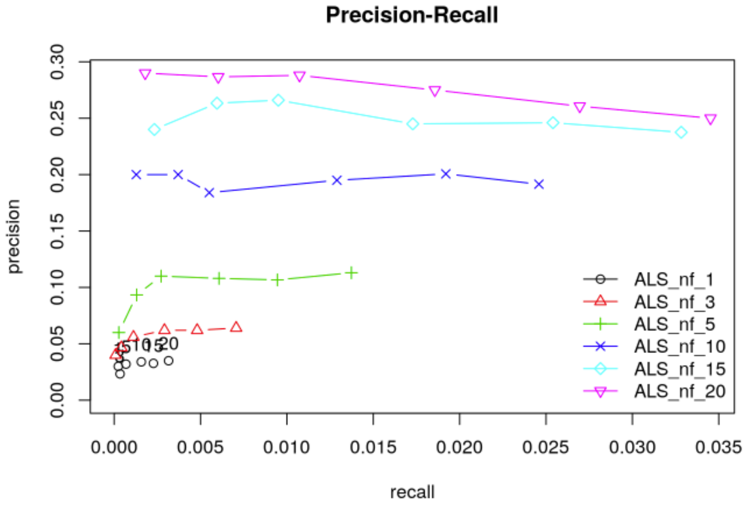
```
knitr::opts_chunk$set(cache=TRUE)

# Default parameter values for the ALS method
rec <-  recommenderRegistry$get_entries(dataType = "realRatingMatrix")
rec$ALS_realRatingMatrix$parameters
```

```
# Random select 100,000 rows to optimize,
ratings_matrix_opt <- ratings_matrix[sample(nrow(ratings_matrix), 100,000), ]

# Setup a new evaluation scheme for paratmeter optimization
evaluation_opt <- evaluationScheme(ratings_matrix_opt,
                                   method     = "cross",
                                   k          = 5,
                                   train      = 0.8,
                                   given      = 10,
                                   goodRating = 3
                                   )


# Let set the n_factors ranging from 1 to 20
nf <- c(1, 3, 5, 10, 15, 20)

# Using lapply to define a list of models to evaluate
als_models <- lapply(nf, function(n){
                        list(name = "ALS",
                             param = list(n_factors = n))
                        })

names(als_models) <- paste0("ALS_nf_", nf)

list_results <- evaluate(evaluation_opt,
                         method = als_models,
                         n = c(1, 3, 5, 10, 15, 20)
                         )
```

Recommender Method Improvement– ALS Optimization




```r
# Plot ROC curve
plot(list_results, annotate = 1, legend = "topleft")
title("ROC Curve")
```




```r
# Plot Precision-Recall curve
plot(list_results, "prec/rec", annotate = 1, legend = "bottomright")
title("Precision-Recall")
```

- n_factors =20 has the highest AUC.

Model Development - ALS

```r
knitr::opts_chunk$set(cache=TRUE)

set.seed(123)

# Create an item-based CF recommender using training data
tic()
rec_als <- Recommender(data = train, method = "ALS", parameter=list(n_factors = 20))
train_time_rec <- toc(quiet = TRUE)

# Create predictions for the test items using known ratings with type as ratings
tic()
pred_als_acr <- predict(object = rec_als, newdata = test_known, type = "ratings")
predict_time_rec <- toc(quiet = TRUE)

# Create predictions for the test items using known ratings with type as top n recommendation list
tic()
pred_als_n <- predict(object = rec_als, newdata = test_known, n = 5)
top_n_time_rec <- toc(quiet = TRUE)
```

```r
knitr::opts_chunk$set(cache=TRUE)

evaluation <- evaluationScheme(ratings_matrix, method = "split", train = 0.8, given = 10)

evaluation
```

```
## Evaluation scheme with 10 items given
## Method: 'split' with 1 run(s).
## Training set proportion: 0.800
## Good ratings: NA
## Data set: 4247 x 2499 rating matrix of class 'realRatingMatrix' with 918946 ratings.
```

```r
train <- getData(evaluation, "train")
train
```

```
## 3397 x 2499 rating matrix of class 'realRatingMatrix' with 736876 ratings.
```

```r
test_known <- getData(evaluation, "known")
test_known
```

```
## 850 x 2499 rating matrix of class 'realRatingMatrix' with 8500 ratings.
```

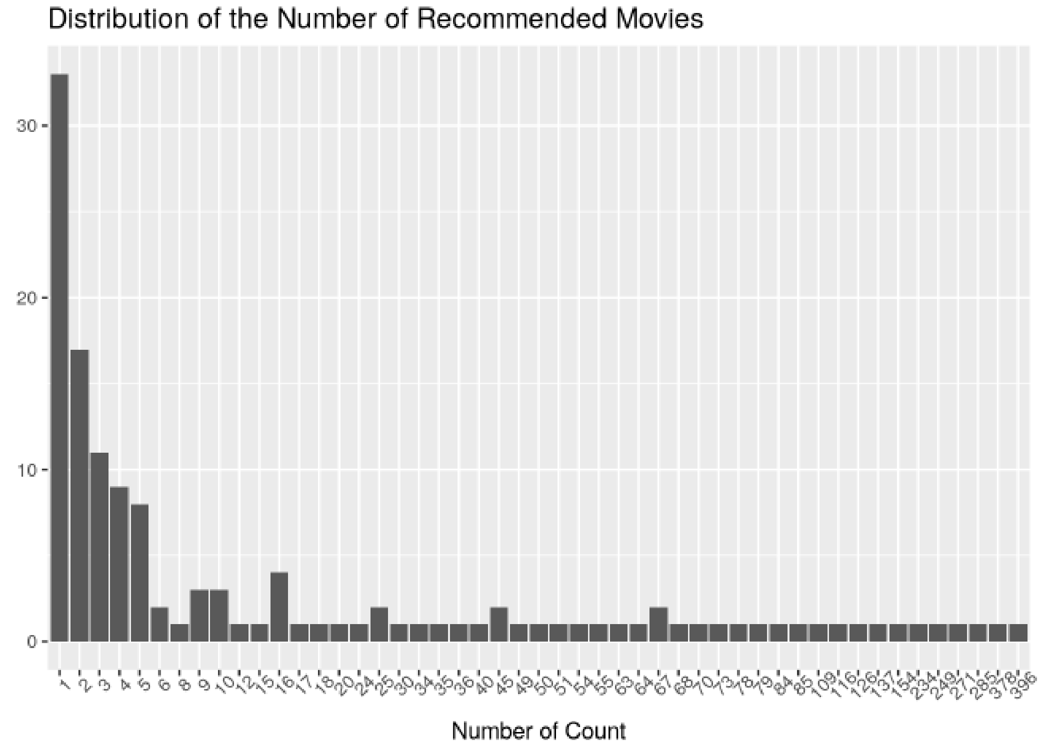
```r
test_unknown <- getData(evaluation, "unknown")
test_unknown
```

```
## 850 x 2499 rating matrix of class 'realRatingMatrix' with 173570 ratings.
```

- Split Test and Train
- Create ALS Model with n_factors=20

Model Evaluation

```r
# Define a matrix with the recommendations to the test set users
rec_matrix <- as.matrix(data.frame(pred_als_n@items))

# Define a vector with all recommendations
num_of_items <- factor(table(rec_matrix))

# Visualize the distribution of the number of recommended movies
qplot(num_of_items) +
  ggtitle("Distribution of the Number of Recommended Movies") +
  labs(x = "Number of Count") +
  theme(axis.text.x = element_text(angle=45))
```




```r
# Recommendations for the first user.
first_user_rec <- pred_als_n@items[1:1] %>% data.frame()
colnames(first_user_rec) <- c("movieId")
first_user_rec <- first_user_rec %>%
  merge(movies, by = "movieId") %>%
  select(-movieId)

first_user_rec
```

```
##                                                        title
## 1                                           To Die For (1995)
## 2 To Wong Foo, Thanks for Everything! Julie Newmar (1995)
## 3                                                Rudy (1993)
##                    genres
## 1 Comedy|Drama|Thriller
## 2                  Comedy
## 3                   Drama
```

- Majority of the movies have been recommended many times, few movies have been recommended small amount of times.

Model Evaluation

```
# Top 10 most recommended movies
top10_rec <- num_of_items %>% data.frame()
top10_rec <- cbind(movieId = rownames(top10_rec), top10_rec)
rownames(top10_rec) <- 1:nrow(top10_rec)
colnames(top10_rec)[2] <- "count"
top10_rec <- top10_rec %>%
  mutate_if(is.factor, ~ as.integer(levels(.x))[.x]) %>%
  merge(movies, by = "movieId") %>%
  top_n(count, n = 10)

top10_rec <- top10_rec[order(top10_rec$count, decreasing = TRUE),] %>%
  select(title:genres)

top10_rec
```

```
##                                            title                 genres
## 4                           Boys on the Side (1995)          Comedy|Drama
## 6                                       Rudy (1993)                 Drama
## 10   Rosencrantz and Guildenstern Are Dead (1990)          Comedy|Drama
## 8                          Cold Comfort Farm (1995)                Comedy
## 5                                Paper, The (1994)          Comedy|Drama
## 3                                   Mallrats (1995)        Comedy|Romance
## 1                                 To Die For (1995) Comedy|Drama|Thriller
## 9                                  Barb Wire (1996)         Action|Sci-Fi
## 7                   Terminator 2: Judgment Day (1991)         Action|Sci-Fi
## 2  Once Upon a Time... When We Were Colored (1995)         Drama|Romance
```

Content Based Recommender System

```r
# Map movie Id for ratings and movies dataset
movies_new <- data.frame(movieId = unique(ratings$movieId)) %>% merge(movies, by = "movieId")
ratings_new <- merge(ratings, movies, by = "movieId") %>% select(-c(title, genres))

# Convert to data frame
genres <- as.data.frame(movies_new$genres, stringsAsFactors = FALSE)

# Split the genres fro each row and transpose
genres_2 <- as.data.frame(tstrsplit(genres[, 1], '[|]', type.convert = TRUE), stringsAsFactors = FALSE)

# Name the columns from 1 to 7
colnames(genres_2) <- c(1:7)

# Create a matrix with columns representing every unique genre, and indicate whether a genre was present or no
t in each movie

## Find unique genres
genre_list <- str_c(c(movies$genres),collapse = ',')
genre_list <- gsub("\\|", ",", genre_list)
genre_list <- unique(strsplit(genre_list, ",")[[1]])
genre_list
```

```
##   Adventure Animation Children Comedy Fantasy Romance Drama Action Crime
## 1         1         1        1      1       1       0     0      0     0
## 2         1         0        1      0       1       0     0      0     0
## 3         0         0        0      1       0       1     0      0     0
## 4         0         0        0      1       0       1     1      0     0
## 5         0         0        0      1       0       0     0      0     0
## 6         0         0        0      0       0       0     0      1     1
##   Thriller Horror Mystery Sci-Fi War Musical Documentary IMAX Western
## 1        0      0       0      0   0       0           0    0       0
## 2        0      0       0      0   0       0           0    0       0
## 3        0      0       0      0   0       0           0    0       0
## 4        0      0       0      0   0       0           0    0       0
## 5        0      0       0      0   0       0           0    0       0
## 6        1      0       0      0   0       0           0    0       0
##   Film-Noir (no genres listed)
## 1         0                  0
## 2         0                  0
## 3         0                  0
## 4         0                  0
## 5         0                  0
## 6         0                  0
```

```r
# Empty matrix
genre_matrix <- matrix(0, length(movies_new$movieId) + 1, length(genre_list))

# Set first row to genre list
genre_matrix[1, ] <- genre_list

# Set column names to genre list
colnames(genre_matrix) <- genre_list

# Iterate through matrix
for (i in 1:nrow(genres_2)) {
 for (c in 1:ncol(genres_2)) {
 genmat_col = which(genre_matrix[1, ] == genres_2[i, c])
 genre_matrix[i + 1, genmat_col] <- 1
 }
}

# Convert into dataframe

## Remove first row (genre list)
genre_matrix_2 <- as.data.frame(genre_matrix[-1, ], stringsAsFactors = FALSE)

## Convert from characters to integers
for (c in 1:ncol(genre_matrix_2)) {
  genre_matrix_2[, c] <- as.integer(genre_matrix_2[, c])
}

head(genre_matrix_2)
```

Content Based Recommender System

```r
tic()

# Convert the ratings into a binary format, where ratings of 4 and 5 are mapped to 1, and ratings of 3 and bel
ow are mapped to -1

binary_ratings <- ratings_new
binary_ratings <- binary_ratings %>%
  mutate(rating = ifelse(rating==4|rating==5, 1,
                         ifelse(rating==1|rating==2|rating==3, -1, NA)))


# Transform from a long format to a wide format
binary_ratings_2 <- dcast(binary_ratings, movieId~userId, value.var = "rating", na.rm = FALSE)

# Convert NA to 0
binary_ratings_2[is.na(binary_ratings_2)] <- 0

# Remove movie Id column
binary_ratings_2 = binary_ratings_2[, -1]

# Calculate dot product of the movie genre matrix and the binary ratings matrix
x <- as.matrix(genre_matrix_2)
y <- as.matrix(binary_ratings_2)
result = t(x) %*% y


# Convert to binary scale
result <- ifelse(result > 0, 1, 0)

prof_mtx_time <- toc(quiet = TRUE)
```

```r
tic()

# First user's profile
result_2 <- result[1, ]

# Calculate Jaccard Distance to measure the similarity between user profiles and the movie genre matrix
sim_mtx <- rbind.data.frame(result_2, genre_matrix_2)
sim_mtx <- data.frame(lapply(sim_mtx, function(x){as.integer(x)}))
sim_results <- dist(sim_mtx, method = "Jaccard")
sim_results <- as.data.frame(as.matrix(sim_results[1:nrow(binary_ratings_2)]))
rows <- which(sim_results == min(sim_results))

sim_dist_time <- toc(quiet = TRUE)
```

```r
# Movies recommended to the first user
movies_rec <- movies_new[rows, ]

kable(movies_rec, format = "html", row.names = FALSE) %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

movieId	title	genres
1907	Mulan (1998)	Adventure|Animation|Children|Comedy|Drama|Musical|Romance

Distributed System – Spark – Data Preparation

```r
# Connect to your Spark cluster
spark_conn <- spark_connect(master = "local")

# Copy ratings matrix to Spark
ratings_tbl <- copy_to(spark_conn, ratings, overwrite=TRUE)

# Remove least-watched movies and least-rated users less than 50
ratings_tbl <- ratings_tbl %>%
  group_by(userId) %>%
  dplyr::mutate(count = n()) %>%
  filter(count > 50)
```

```r
ratings_tbl <- ratings_tbl %>%
  select(-count) %>%
  group_by(movieId) %>%
  dplyr::mutate(count = n()) %>%
  filter(count >50)
ratings_tbl <- ratings_tbl %>% select(-count)
```

```r
# Split the dataset into training and test set
partitions <- ratings_tbl %>% sdf_random_split(training = 0.8, test = 0.2, seed = 123)
```

- Connect to spark
- Copy ratings data into spark
- Remove movies that have only few ratings and remove users who only rated few movies
- Split, test and train

Distributed System – Spark – Model Development - ALS

```
set.seed(456)

# Train the ALS model
tic()
als_model <- ml_als(partitions$training, rating_col = "rating", user_col = "userId", item_col = "movieId")
train_time_sp <- toc(quiet = TRUE)

# Predict rating using test set
tic()
als_pred <- ml_predict(als_model, partitions$test)
predict_time_sp <- toc(quiet = TRUE)

# Return the top 5 recommendations
tic()
als_rec <- ml_recommend(als_model, type = "item", 5) %>% select(-recommendations)
top_n_time_sp <- toc(quiet =TRUE)
```

```
# Recommendations for the first user.
first_user_sp <- als_rec %>%
  filter(userId==13) %>%
  select(-c(userId, rating)) %>%
  merge(movies, by = "movieId") %>%
  select(-movieId)

first_user_sp
```

```
##                                                                       title
## 1                                      Star Wars: Episode IV - A New Hope (1977)
## 2                                              Shawshank Redemption, The (1994)
## 3                                                       Schindler's List (1993)
## 4 Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
## 5                                                       Sixth Sense, The (1999)
##                    genres
## 1 Action|Adventure|Sci-Fi
## 2             Crime|Drama
## 3               Drama|War
## 4        Action|Adventure
## 5    Drama|Horror|Mystery
```

- Create the Recommender System
- Recommendations for the first user.

Distributed System – H2O – Data Preparation

```r
# Specify H2O configuration needed the start and run of H2O-3 cluster
h2oConf <- H2OConf()

# Create H2O Context
hc <- H2OContext.getOrCreate(h2oConf)
```

```r
# Covert the ratings_tbl_2 Spark dataframe into an H2O frame
ratings_tbl_hf <- hc$asH2OFrame(ratings_tbl)
```

Split Dataset

```r
# Split the dataset into training and test set
splits_2 <- h2o.splitFrame(ratings_tbl_hf, ratios = 0.8, seed = 123)
```

- Connect to H2O
- Convert the Spark ratings dataframe to H2O frame
- Split, test and train

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         7 seconds 246 milliseconds
##     H2O cluster timezone:       Etc/UTC
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.30.0.6
##     H2O cluster version age:    14 days, 10 hours and 25 minutes
##     H2O cluster name:           sparkling-water-rstudio_local-1594793430674
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   1.78 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          172.31.35.234
##     H2O Connection port:        54325
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         XGBoost, Algos, Amazon S3, Sparkling Water REST API Extensions, AutoML, Cor
e V3, TargetEncoder, Core V4
##     R Version:                  R version 3.6.0 (2019-04-26)
##
## Reference class object of class "H2OContext"
## Field "jhc":
## <jobj[94]>
##   org.apache.spark.h2o.H2OContext
##
## Sparkling Water Context:
##  * Sparkling Water Version: 3.30.0.6-1-2.4
##  * H2O name: sparkling-water-rstudio_local-1594793430674
##  * cluster size: 1
##  * list of used nodes:
##   (executorId, host, port)
##   ------------------------
##   (0,172.31.35.234,54325)
##   ------------------------
##
##   Open H2O Flow in browser: http://localhost:54324 (CMD + click in Mac OSX)
##
##
```

Distributed System – H2O – Model Development - GLM

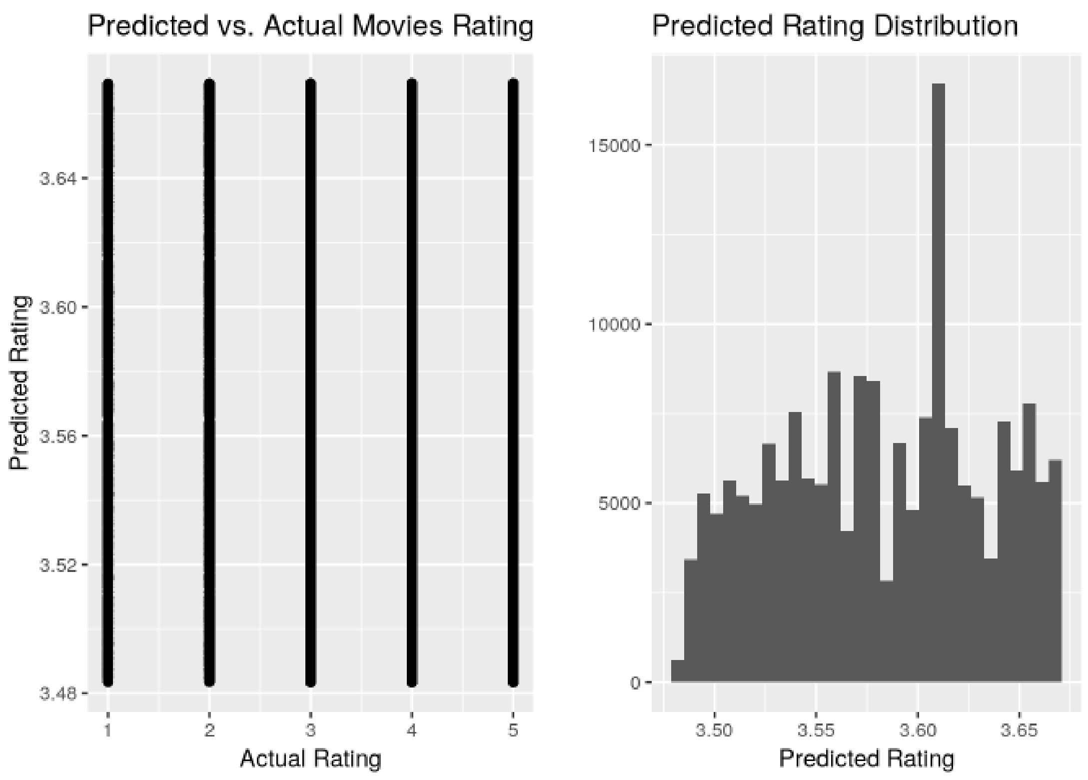
```
y = "rating"
x = setdiff(names(ratings_tbl_hf), y)

# Train the model
tic()

glm_model <-  h2o.glm(x = x,
                      y = y,
                      training_frame = splits_2[[1]],
                      validation_frame = splits_2[[2]],
                      lambda_search = TRUE,
                      seed = 455)
```

```
train_time_h2o <- toc(quiet =TRUE)

# Get top level summary information on our model
summary(glm_model)
```

- Create the recommender system
- Summary of the model

```
## Model Details:
## ==============
##
## H2ORegressionModel: glm
## Model Key:  GLM_model_R_1594793458141_1
## GLM Model: summary
##     family     link                              regularization
## 1 gaussian identity Elastic Net (alpha = 0.5, lambda = 0.03036 )
##                                                              lambda_search
## 1 nlambda = 100, lambda.max = 0.1345, lambda.min = 0.03036, lambda.1se = -1.0
##   number_of_predictors_total number_of_active_predictors
## 1                          2                           1
##   number_of_iterations  training_frame
## 1                   17 RTMP_sid_8b90_1
##
## H2ORegressionMetrics: glm
## ** Reported on training data. **
##
## MSE:  1.229628
## RMSE:  1.108886
## MAE:  0.9259446
## RMSLE:  0.2835432
## Mean Residual Deviance :  1.229628
## R^2 :  0.003459827
## Null Deviance :905653
## Null D.o.F. :733977
## Residual Deviance :902519.6
## Residual D.o.F. :733976
## AIC :2234667
##
##
## H2ORegressionMetrics: glm
## ** Reported on validation data. **
##
## MSE:  1.226579
## RMSE:  1.10751
## MAE:  0.9245265
## RMSLE:  0.2832008
## Mean Residual Deviance :  1.226579
## R^2 :  0.003648224
## Null Deviance :225385.5
## Null D.o.F. :183080
## Residual Deviance :224563.3
## Residual D.o.F. :183079
## AIC :556957.8
##
```

Distributed System – H2O – Model Development - GLM




```r
# Convert from H2O Frame to Spark Data Frame
predicted <- hc$asSparkFrame(glm_pred)

# Extract the true 'rating' values from our test dataset
actual <- hc$asSparkFrame(splits_2[[2]]) %>%
  select(rating) %>%
  collect() %>%
  `[[`("rating")

# Produce a data frame housing the predicted + actual 'rating' values
predict_actual <- data.frame(predicted = predicted, actual = actual)
names(predict_actual) <- c("predicted", "actual")

# Plot predicted vs. actual values
point_plot <- ggplot(predict_actual, aes(x = actual, y = predicted)) +
  geom_point() +
  theme(plot.title = element_text(hjust = 0.5)) +
  labs(
    x = "Actual Rating",
    y = "Predicted Rating",
    title = "Predicted vs. Actual Movies Rating")

# Plot predicted rating distribution
dist_plot <- qplot(predict_actual$predicted) +
  ggtitle("Predicted Rating Distribution") +
  labs(x = 'Predicted Rating')

grid.arrange(point_plot, dist_plot, nrow = 1)
```

- Rating prediction mostly between 3.3 and 3.8
- Improvement could done by tuning its hyperparameters such as alpha and lambda

Distributed System – Model Evaluation - Accuracy

	RMSE	MSE	MAE
Recommenderlab ALS	0.9404515	0.8844490	0.7491568
Spark ALS	0.8654353	0.7489783	0.6941973
H2O GLM	1.1075102	1.2265789	0.9245265

```r
# Evaluate the accuracy for the Recommenderlab ALS model
acr_als <- calcPredictionAccuracy(pred_als_acr, test_unknown)

# Remove NaN values due to dataset splitting in Spark
als_pred <- als_pred %>% filter(!is.na(prediction))

# Evaluate the accuracy for the Spark ALS model
spark_mae <- als_pred  %>%
  data.frame() %>%
  mutate(error = abs(rating - prediction)) %>%
  summarize(mean(error))

spark_mse <- als_pred  %>%
  data.frame() %>%
  mutate(error = (rating - prediction)^2) %>%
  summarize(mean(error))

spark_rmse <- als_pred  %>%
  data.frame() %>%
  mutate(error = (rating - prediction)^2) %>%
  summarize(sqrt(mean(error)))

Spark_ALS <- data.frame(RMSE = spark_rmse, MSE = spark_mse, MAE = spark_mae)
colnames(Spark_ALS) <- c("RMSE", "MSE", "MAE")

# Evaluate the accuracy for the H2O GLM model
h2o_mse <- glm_perf@metrics$MSE %>% data.frame()

h2o_rmse <- glm_perf@metrics$RMSE %>% data.frame()

h2o_mae <- glm_perf@metrics$mae %>% data.frame()

H2O_GLM <- data.frame(RMSE = h2o_rmse, MSE = h2o_mse, MAE = h2o_mae)
colnames(H2O_GLM) <- c("RMSE", "MSE", "MAE")

# Combine the RMSE, MSE, and MAE for both models
acr <- rbind("Recommenderlab ALS" = acr_als,
             "Spark ALS" = Spark_ALS,
             "H2O GLM" = H2O_GLM)

# Update column names to RMSE, MSE, and MAE
colnames(acr) <- c("RMSE", "MSE", "MAE")

kable(acr, format = "html") %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

Distributed System – Model Evaluation - Performance

Method	Training	Predicting	Top_N
Recommenderlab	6644.53	416.14	412.45
Spark	7.22	0.07	3.57
H2O	2.23	1.06	NA

Method	Profile	Similarity
Content-Based	5.85	0.41

```r
# Set up data frame for running time performance
runtime <- data.frame(Method=character(), Training=double(), Predicting=double(), Top_N=double())
runtime_2 <- data.frame(Method=character(), Profile=double(), Similarity=double())

# Combine the running time for the Recommenderlab ALS model
runtime <- rbind(runtime, data.frame(Method  = "Recommenderlab",
                                     Training = round(train_time_rec$toc - train_time_rec$tic, 2),
                                     Predicting = round(predict_time_rec$toc - predict_time_rec$tic, 2),
                                     Top_N = round(top_n_time_rec$toc - top_n_time_rec$tic, 2)))

# Combine the running time for the Content-Based model.
runtime_2 <- rbind(runtime_2, data.frame(Method  = "Content-Based",
                                     Profile = round(prof_mtx_time$toc - prof_mtx_time$tic, 2),
                                     Similarity = round(sim_dist_time$toc - sim_dist_time$tic, 2)))


# Combine the running time for the Spark ALS model
runtime<- rbind(runtime, data.frame(Method  = "Spark",
                                    Training = round(train_time_sp$toc - train_time_sp$tic, 2),
                                    Predicting = round(predict_time_sp$toc - predict_time_sp$tic, 2),
                                    Top_N = round(top_n_time_sp$toc - top_n_time_sp$tic, 2)))

# Combine the running time for the H2O GLM model
runtime<- rbind(runtime, data.frame(Method  = "H2O",
                                    Training = round(train_time_h2o$toc - train_time_h2o$tic, 2),
                                    Predicting = round(predict_time_h2o$toc - predict_time_h2o$tic, 2),
                                    Top_N = NA))


# Remove row names
rownames(runtime) <- NULL
rownames(runtime_2) <- NULL

kable(runtime, format = "html", row.names = FALSE) %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

Distributed System –Model Evaluation – 1st User

Evaluate the 1st User

```
first_user_all <- cbind(first_user_rec$title, movies_rec$title, first_user_sp$title)
```

```
colnames(first_user_all) <- c("Recommenderlab", "Content-Based", "Spark")

kable(first_user_all, format = "html", row.names = FALSE) %>%
  kable_styling(bootstrap_options = c("striped", "hover"))
```

Recommenderlab	Content-Based	Spark
Mallrats (1995)	Mulan (1998)	Star Wars: Episode IV - A New Hope (1977)
Boys on the Side (1995)	Mulan (1998)	Shawshank Redemption, The (1994)
Paper, The (1994)	Mulan (1998)	Schindler's List (1993)
Cold Comfort Farm (1995)	Mulan (1998)	Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)
Mallrats (1995)	Mulan (1998)	Sixth Sense, The (1999)

Conclusion

- We created 4 different recommender systems, Distributed and Non-distributed.
    - Recommenderlab, ALS method
    - Content-Based , movie genres
    - Spark, ALS method
    - H2O, GLM method
- Spark ALS approach performs the best in term of accuracy and speed.
- Recommenderlab approach took the longest time to run, ~ 2 hours just for training a million rating data.
    - AWS EC2 Instance Type T2, 8 vCPU, 32gb Memory
    - Could be run faster on Instance Type G4, which designed to accelerate computing for machine learning inference for applications like recommender systems.
- We can further improve the distributed recommender systems by optimizing the algorithms hyperparameters.
- The content-based recommender engine can also be improved to include several other attributes such as movie overview, actors, directors, and keywords . This could be done with methods such as the Term Frequency–Inverse Document Frequency algorithm (TFIDF).

References

Gorakala, K.G. & Usuelli, M. (2015, Sept). Building a Recommendation System with R (pp. 50-92). Packt Publishing Ltd.

Hashler, M. & Vereet, B. (2019, Aug 27). Package ‘recommenderlab’. CRAN. Retrieved from https://cran.r-project.org/web/packages/recommenderlab/recommenderlab.pdf.

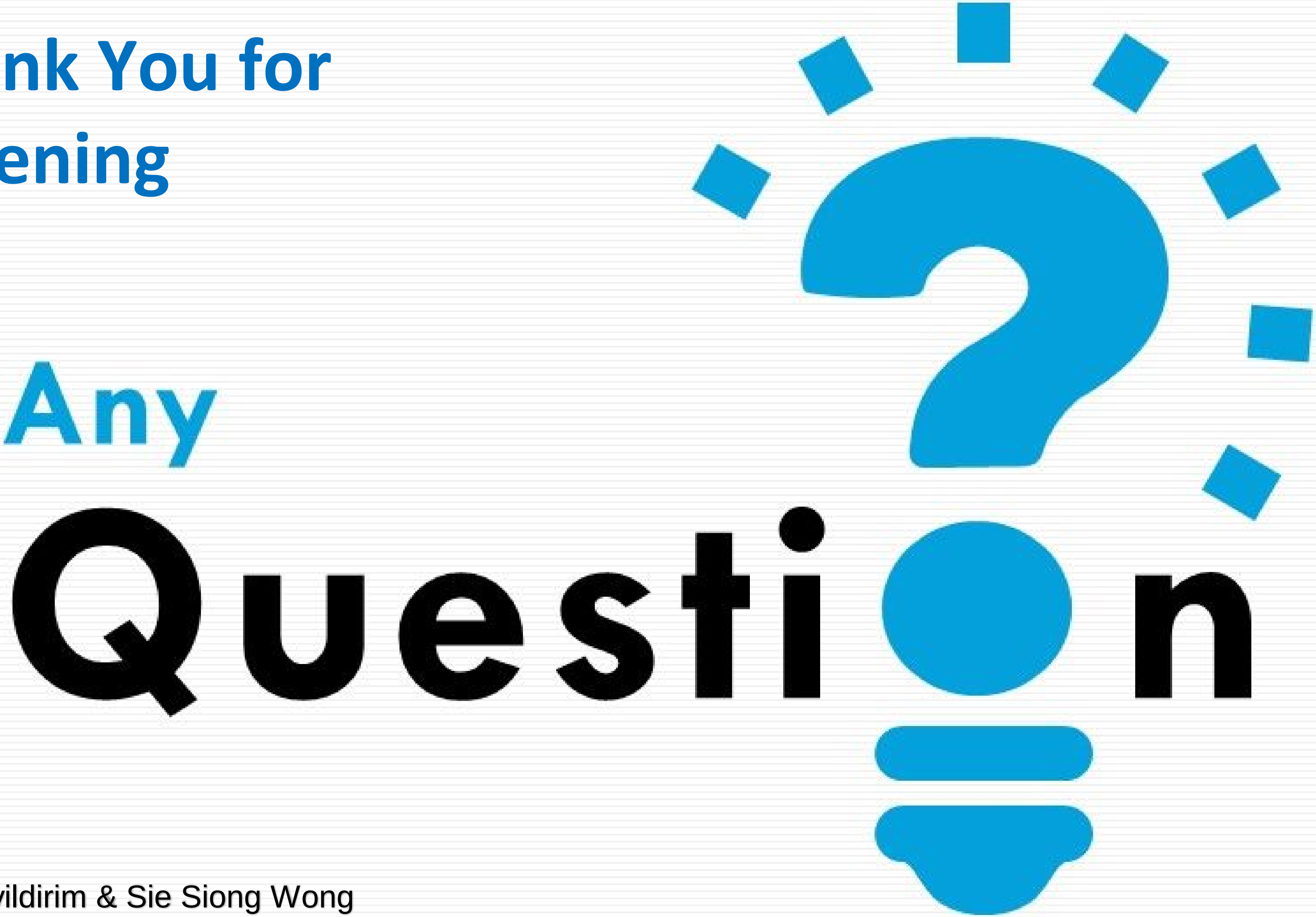
Raela. (2015, Jun 7). Building a Movie Recommendation Engine with R. Muffynomster. Retrieved from https://muffynomster.wordpress.com/2015/06/07/building-a-movie-recommendation-engine-with-r/.

Boehmke, B. & Greenwell, B. (2020, Feb 1st). Hands-On Machine Learning with R. Taylor & Francis Group. Retrieved from https://bradleyboehmke.github.io/HOML/GLRM.html.

Generalized Linear Model (GLM). (2020, Jun 30). H2O.ai. Retrieved from https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/glm.html.

Usai, D. (2019, Mar 25). Market Basket Analysis with recommenderlab. Medium. Retrieved from https://towardsdatascience.com/market-basket-analysis-with-recommenderlab-5e8bdc0de236

Thank You for Listening



Anil Akyildirim & Sie Siong Wong