Movie Recommender Engine

Ningning Zhu

- **Data Cleaning and Exploring**
    - Word Cloud of production companies
    - Number of movie releases per month
    - Movie revenue by month
    - Global map of production countries
    - Most popular movies
    - Relationship between vote average, vote count and popularity
- **Recommend by calculated IMDB Score**
- **Most Popular (item average) Method Recommendation,** *RMSE = 0.9933889*

- **Memory Based Method**
    - **User Based Collaborative Filtering (UBCF),** *RMSE = 0.9650920*
        - Best K Nearest Neighbor (KNN)
        - Best User Similarity calculation method
            - Cosine Similarity
            - Pearson Correlation
            - Euclidean Distance
        - Data transformation
        - Specific user rating prediction
    - **Memory Based Model Comparison**

- **Method Based (Method-Matrix Factorization)**
    - Gradient Descent, *RMSE = 0.9045648*
        - Latent factor, Learning Rate, L1&L2 Regulation
    - Singular Value Decomposition (SVD) Approximation, *RMSE = 0.9860490*
    - Simon Funk SVD, *RMSE = 0.9070839*
    - Alternating Least Square (ALS), *RMSE = 0.9796331*
    - Hybrid, *RMSE = 0.9519534*
        - SVD, UBCF and Popular weight 0.1, 0.6, 0.3
    - Slope One

- **Content Based**
    - Recommend by genre
        - Word Could of Genre
    - Recommend by Natural Language Processing (NLP) to movie overview
        - Clustering by groups
        - Predict movies similar to a specific movie by its overview
        - Searching Engine
    - Recommend by actor, crew and genre

Data Cleaning and Exploring

Look at the dataset

```
## Observations: 4,803
## Variables: 20
## $ budget               <int> 237000000, 300000000, 245000000, 250000000,…
## $ genres               <chr> "[{\"id\": 28, \"name\": \"Action\"}, {\"id…
## $ homepage             <chr> "http://www.avatarmovie.com/", "http://disn…
## $ id                   <int> 19995, 285, 206647, 49026, 49529, 559, 3875…
## $ keywords             <chr> "[{\"id\": 1463, \"name\": \"culture clash\…
## $ original_language    <chr> "en", "en", "en", "en", "en", "en", "en", "…
## $ original_title       <chr> "Avatar", "Pirates of the Caribbean: At Wor…
## $ overview             <chr> "In the 22nd century, a paraplegic Marine i…
## $ popularity           <dbl> 150.43758, 139.08262, 107.37679, 112.31295,…
## $ production_companies <chr> "[{\"name\": \"Ingenious Film Partners\", \…
## $ production_countries <chr> "[{\"iso_3166_1\": \"US\", \"name\": \"Unit…
## $ release_date         <chr> "2009-12-10", "2007-05-19", "2015-10-26", "…
## $ revenue              <dbl> 2787965087, 961000000, 880674609, 108493909…
## $ runtime              <dbl> 162, 169, 148, 165, 132, 139, 100, 141, 153…
## $ spoken_languages     <chr> "[{\"iso_639_1\": \"en\", \"name\": \"Engli…
## $ status               <chr> "Released", "Released", "Released", "Releas…
## $ tagline              <chr> "Enter the World of Pandora.", "At the end …
## $ title                <chr> "Avatar", "Pirates of the Caribbean: At Wor…
## $ vote_average         <dbl> 7.2, 6.9, 6.3, 7.6, 6.1, 5.9, 7.4, 7.3, 7.4…
## $ vote_count           <int> 11800, 4500, 4466, 9106, 2124, 3576, 3330, …
```

```
##      budget             genres             homepage
##  Min.   :        0   Length:4803        Length:4803
##  1st Qu.:   790000   Class :character   Class :character
##  Median : 15000000   Mode  :character   Mode  :character
##  Mean   : 29045040
##  3rd Qu.: 40000000
##  Max.   :380000000
##
##        id           keywords         original_language  original_title
##  Min.   :     5   Length:4803        Length:4803        Length:4803
##  1st Qu.:  9014   Class :character   Class :character   Class :character
##  Median : 14629   Mode  :character   Mode  :character   Mode  :character
##  Mean   : 57166
##  3rd Qu.: 58610
##  Max.   :459488
##
##    overview           popularity      production_companies
##  Length:4803        Min.   :  0.000   Length:4803
##  Class :character   1st Qu.:  4.668   Class :character
##  Mode  :character   Median : 12.922   Mode  :character
##                     Mean   : 21.492
##                     3rd Qu.: 28.314
##                     Max.   :875.581
##
##  production_countries release_date          revenue
##  Length:4803          Length:4803        Min.   :0.000e+00
##  Class :character     Class :character   1st Qu.:0.000e+00
##  Mode  :character     Mode  :character   Median :1.917e+07
##                                          Mean   :8.226e+07
##                                          3rd Qu.:9.292e+07
##                                          Max.   :2.788e+09
##
```

```
##
##     runtime      spoken_languages      status            tagline
##  Min.   :  0.0   Length:4803        Length:4803        Length:4803
##  1st Qu.: 94.0   Class :character   Class :character   Class :character
##  Median :103.0   Mode  :character   Mode  :character   Mode  :character
##  Mean   :106.9
##  3rd Qu.:118.0
##  Max.   :338.0
##  NA's   :2
##     title            vote_average      vote_count
##  Length:4803        Min.   : 0.000   Min.   :    0.0
##  Class :character   1st Qu.: 5.600   1st Qu.:   54.0
##  Mode  :character   Median : 6.200   Median :  235.0
##                     Mean   : 6.092   Mean   :  690.2
##                     3rd Qu.: 6.800   3rd Qu.:  737.0
##                     Max.   :10.000   Max.   :13752.0
##
```

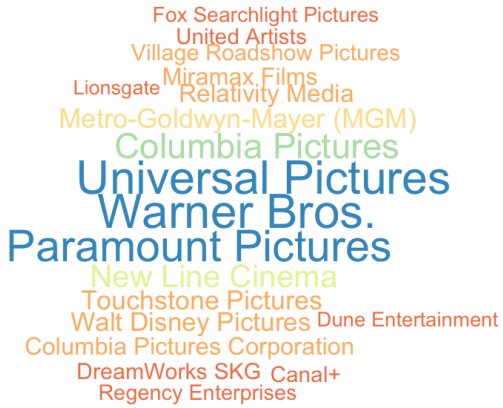
Convert JSon format (keywords, production companies, genres)

```
##       id  title       keyword
## 1  19995 Avatar culture clash
## 2  19995 Avatar        future
## 3  19995 Avatar     space war
## 4  19995 Avatar  space colony
## 5  19995 Avatar       society
## 6  19995 Avatar  space travel
## 7  19995 Avatar    futuristic
## 8  19995 Avatar       romance
## 9  19995 Avatar         space
## 10 19995 Avatar         alien
```

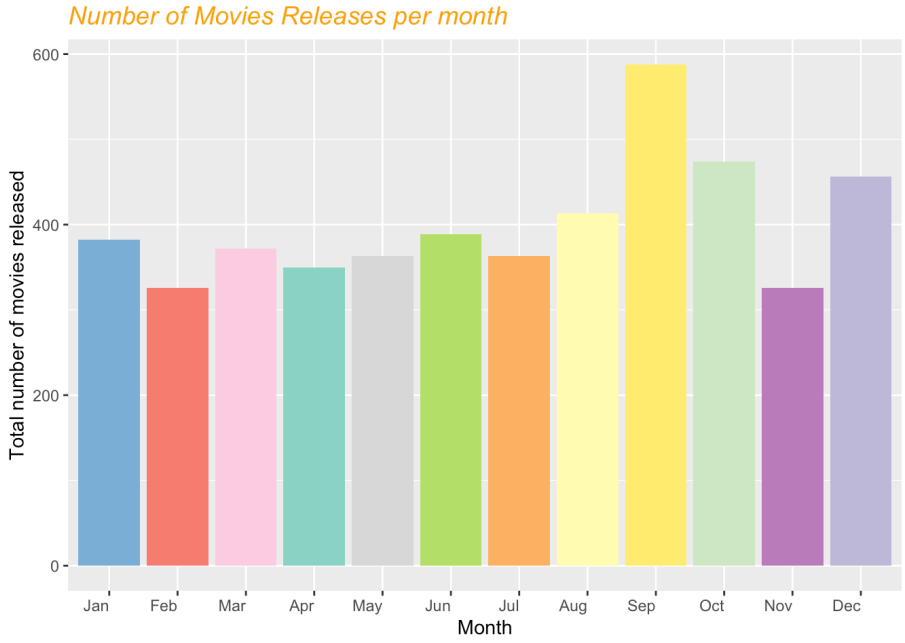
```
##        id                                    title           genre
## 1   19995                                   Avatar          Action
## 2   19995                                   Avatar       Adventure
## 3   19995                                   Avatar         Fantasy
## 4   19995                                   Avatar Science Fiction
## 5     285 Pirates of the Caribbean: At World's End       Adventure
## 6     285 Pirates of the Caribbean: At World's End         Fantasy
## 7     285 Pirates of the Caribbean: At World's End          Action
## 8  206647                                  Spectre          Action
## 9  206647                                  Spectre       Adventure
## 10 206647                                  Spectre           Crime
```

```
##        id                             company
## 1   19995               Ingenious Film Partners
## 2   19995 Twentieth Century Fox Film Corporation
## 3   19995                    Dune Entertainment
## 4   19995              Lightstorm Entertainment
## 5     285                  Walt Disney Pictures
## 6     285               Jerry Bruckheimer Films
## 7     285               Second Mate Productions
## 8  206647                     Columbia Pictures
## 9  206647                               Danjaq
## 10 206647                                  B24
```

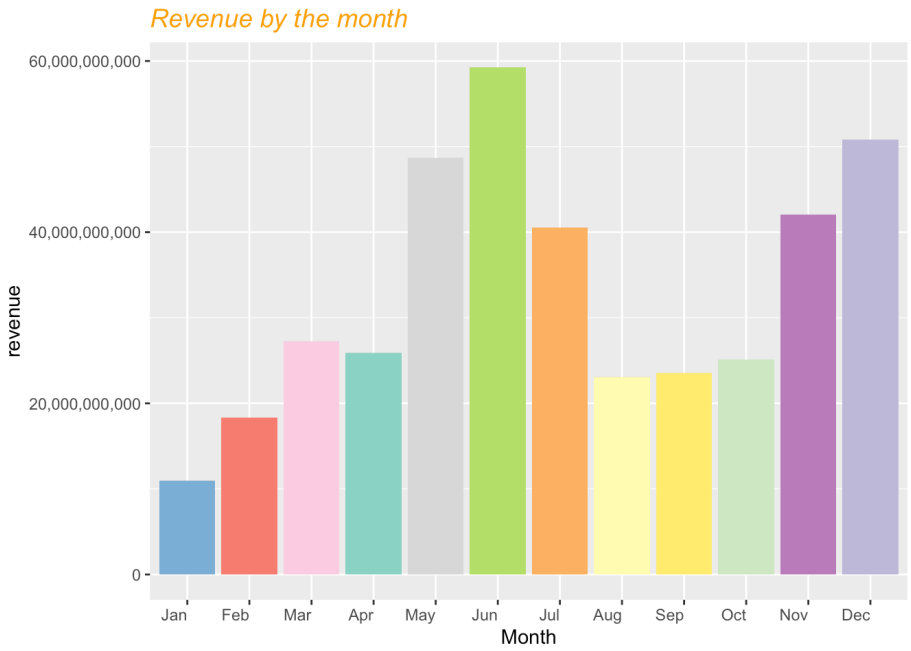
Make a word cloud of production companies. From the word cloud, we can tell Universal Pictures, Warner Bros, and Paramount Pictures produces the most movies among all movie companies.



Convert the char class of the release date into date format, then group the months that the movie releases in the descending order. From Wikipedia, we know that in North America, at least, there are two dump "seasons" — late summer (August and September), and winter (January, February, and sometimes early March). In the histogram below, movies released at August, September, December and January are high compared to other month's. Our finding accords to the movie production phenomenon, hence this dataset is convincing.

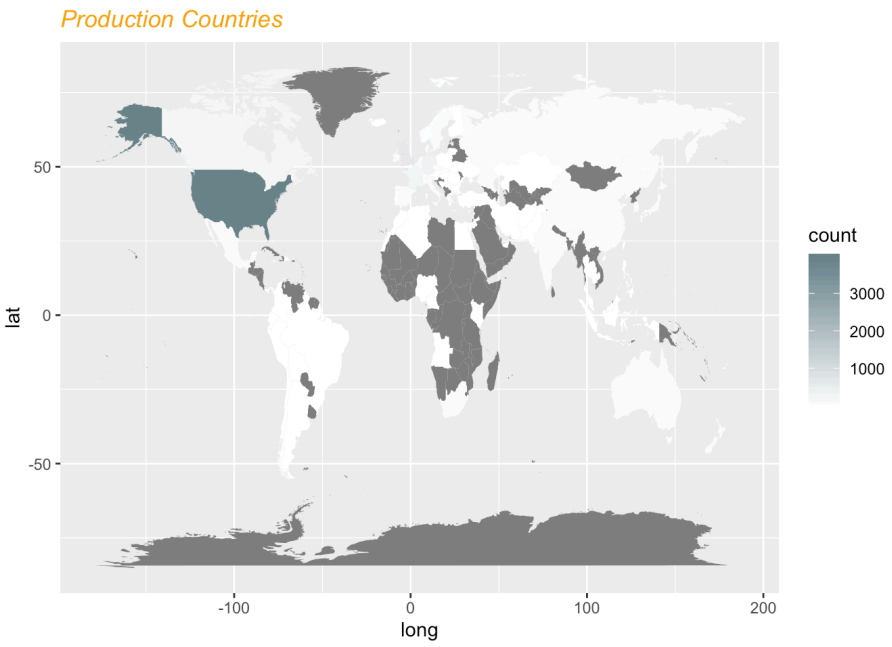


Group the months that the movie gets the highest revenue. Movie revenue are highest in June, May, July, November and December. It is naturally to think that most students are having Summer and Winter breaks at that time, they might tend to go to the theater during these months. Our expectation accords to the histogram, shows the dataset is convincing.

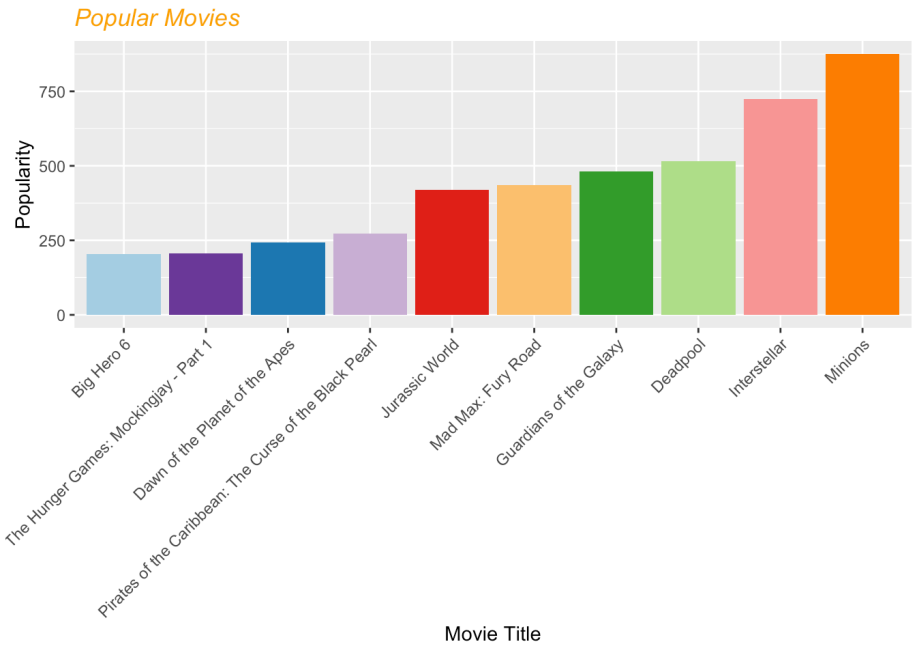


We group all movies by their production countries. We can see USA outperforms UK about 6 times. Then we project this table into a global map. In the global map, USA has the darkest color, then following UK and Europe. The color comparison in the global map is obvious, this is because the USA's movie production number is UK's 6 times.

```
## # A tibble: 10 x 2
##    country   count
##    <fct>     <int>
##  1 USA        3956
##  2 UK          636
##  3 Germany     324
##  4 France      306
##  5 Canada      261
##  6 Australia   110
##  7 Italy        72
##  8 Spain        71
##  9 China        59
## 10 Japan        58
```

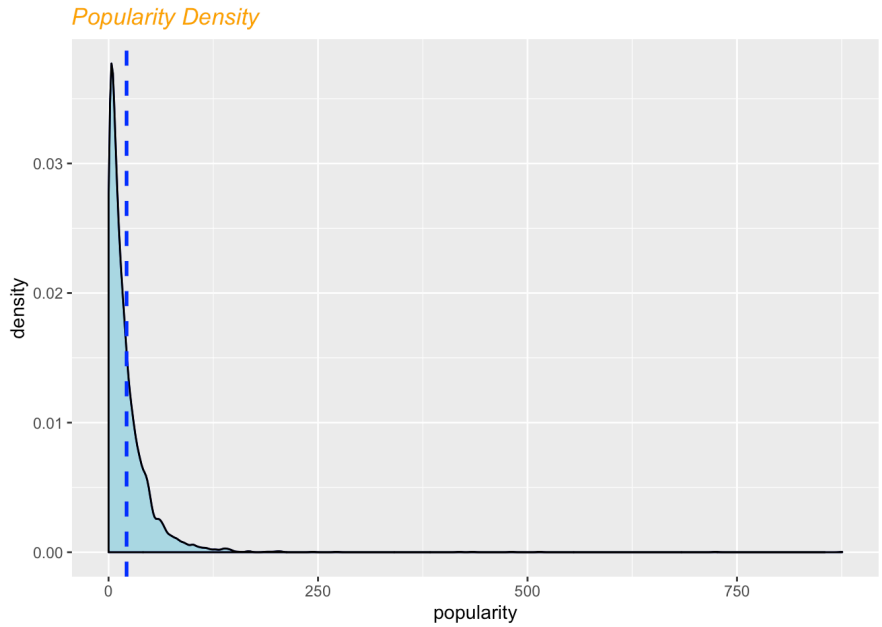


Look at the popularity, make a histogram of popular movies. Minions is the most popular movie, then follows Interstellar, Deadpool, Guardians of the Galaxy and Mad Max: Fury Road.

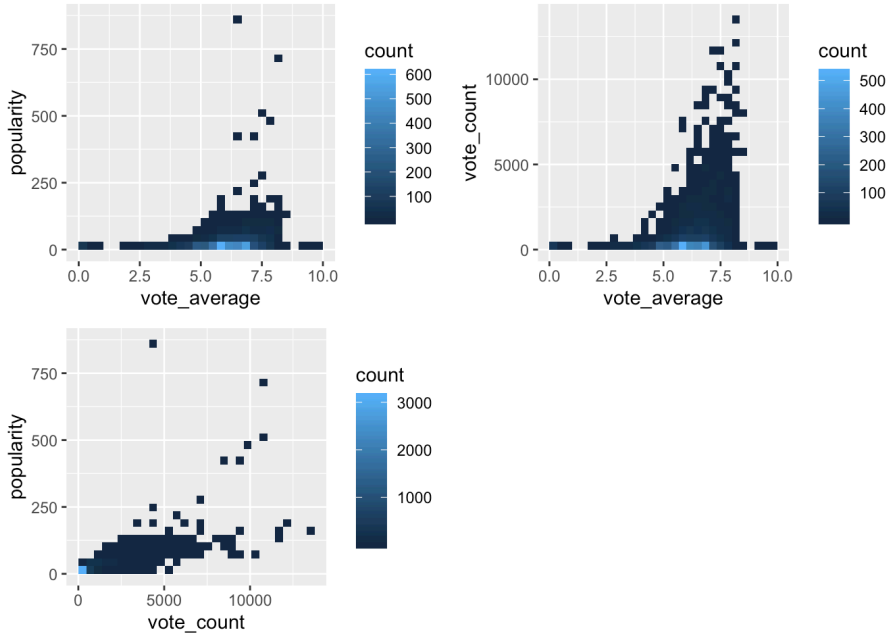


Look at the relationship between vote and popularity. by making the density graph, we find the popularity density is extremely right skewed. It makes sense since the average is about 20 (the blue dot line), the maximum number is about 800.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   4.668  12.922  21.492  28.314 875.581
```



Make graphs of the scatter plot to see the relationship between Vote_Average and Popularity, Vote_Average and Vote_Count, Vote_Count and Popularity.



By looking at the Pearson correlation, alpha equals to 0.05, we can see the vote average and popularity have a very low correlation(0.27), which means there is hardly a relationship between vote average and popularity. Hence we cannot just say the movie is good just because it has a high vote average or high popularity.

```
##
##  Pearson's product-moment correlation
##
## data:  movie$vote_average and movie$popularity
## t = 19.737, df = 4801, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.2475880 0.2999102
## sample estimates:
##       cor
## 0.2739518
```

by looking at the Pearson correlation, alpha equals to 0.05, we can see the vote average and vote count have a very low correlation(0.31), which means the movie has been voted the most doesn't mean it has a high vote average.

```
##
##  Pearson's product-moment correlation
##
## data:  movie$vote_average and movie$vote_count
## t = 22.835, df = 4801, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.2872582 0.3382849
## sample estimates:
##       cor
## 0.3129974
```

Vote count and popularity has a strong relationship about 0.78, which means people tend to rate popular movies, maybe this is because of the popular movies have more audience than other movies'.

```
##
##  Pearson's product-moment correlation
##
## data:  movie$vote_count and movie$popularity
## t = 85.839, df = 4801, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.7667211 0.7890473
## sample estimates:
##       cor
## 0.7781299
```

Simplest Recommendation System

We need a good formula to check the quality of a movie since vote average and popularity hardly have a relationship. I use the IMDB formula to check the quality, the formula is given in the IMDB website. This recommendation is based on the IMDB's weighted rating. The following formula is from IMDB website:

```
weighted rating (WR) = (v ÷ (v+m)) × R + (m ÷ (v+m)) × C
```

Where: R = average for the movie (mean) = (rating) v = number of votes for the movie = (votes) M = minimum votes required to be listed in the Top Rated list C = the mean vote across the whole report.

From the movie dataset, we already have, R(average for the movie) and V(number of votes) To calculate C, we calculate the mean of the vote average column

```
c = mean(movie$vote_average)
```

We set M to be 75% of the total votes. It means the if one movie is in the Top Rated list, it must has higher votes of than other 75% movies. Now we write a function to calculate each movie's IMDB score and ordered them in decreasing order.

In the table below, the first column is the movie IMDB score we just calculated, the second column is movie id, the third is the movie name. The Shawshank Redemption had the highest IMDB score, following The Godfather, Fight club, Pulp fiction and The Dark Knight. This accords to the real world recommender engine works. Because for me, I always seeing the Shawshank Redemption be the first movie for people to recommend.

```
  v = movie$vote_count
  r = movie$vote_average
function_score <-  function(v,m,r,c){
    wr<-(v / (v+m)) * r + (m / (v+m)) * c
    return(wr)
}
score <- function_score(v,m,r,c)[movie$vote_count>=m]
```

```
top_movies<-data_frame(score,id = movie$id[movie$vote_count>=m])
```

```
## # A tibble: 1,203 x 3
##    score     id title
##    <dbl>  <int> <chr>
##  1  8.30    278 The Shawshank Redemption
##  2  8.14    238 The Godfather
##  3  8.14    550 Fight Club
##  4  8.12    680 Pulp Fiction
##  5  8.08    155 The Dark Knight
##  6  8.02     13 Forrest Gump
##  7  8.00  27205 Inception
##  8  7.98    424 Schindler's List
##  9  7.97 244786 Whiplash
## 10  7.97 157336 Interstellar
## # … with 1,193 more rows
```

Most Popular (item average) Method Recommendation

```
$POPULAR_realRatingMatrix
Recommender method: POPULAR for realRatingMatrix
Description: Recommender based on item popularity.
Reference: NA
Parameters:
  normalize    aggregationRatings aggregationPopularity
1  "center" new("standardGeneric" new("standardGeneric"
```

Above is the default parameter and description of Most popular method in Recommenderlab package document. Most popular (item average) approach computes average rating for each item based on available ratings and predicts each unknown rating as average for item. As a result, missed ratings for each item will be the same for each user.

Algorithm:
1. Calculate average rating for each item;
2. Predict missed ratings in *R*(rating matrix) as average for item.

We want to see the frequency of rating. Movies have been rated 4.0 over 25,000 times; movies have been rated as s3.0 over 20,000; then following are 5.0 and 2.0. Fewest movies have been rated as 1.0. From this lightly left skewed pattern, we can tell most users are quite generous and they tend to rate 3.0 to 5.0 when rating movies instead of giving low scores. Since this data has no missing value and all ratings are in the range from 0 to 5, we skip the data cleaning step.

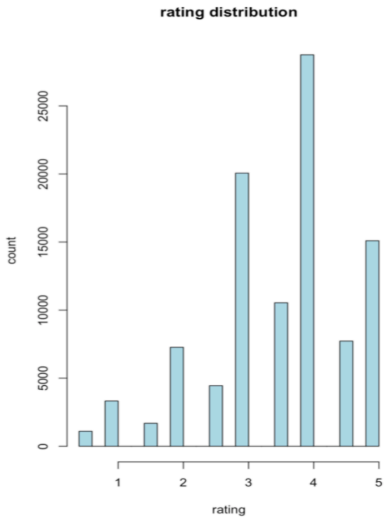


Figure 1: movie score frequency

To work with recommenderlab package, data firstly is needed to be converted to sparse format, I just displayed the user 15-20's rating to movie 1:10:

```
##
## [1,] 2.0 2 . . 4.5 4.0 . . . 3
## [2,] .   . . . .   .   . . . .
## [3,] .   . . . .   4.5 . . . .
## [4,] .   . . . 3.0 4.0 3 . 3 .
## [5,] 3.0 3 3 3 .   3.0 3 . 3 3
## [6,] 3.5 . . . .   .   . . . .
```

Split the data into the training and test set by 0.8 and 0.2. Here we set given equals to 20 means for the test set 20 items will be given to the recommender algorithm and the other items will be held out for computing the error. Testing is performed by withholding items (parameter given). Breese et al. (1998) introduced the four experimental withholding protocols called Given 2, Given 5, Given 10 and All-but-1. During testing, the Given x protocol presents the algorithm with only x randomly chosen items for the test user, and the algorithm is evaluated by how well it is able to predict the withheld items.

Good Rating is the threshold for calculating the confusion matrix. We set it equal to 3.

TABLE II. CONFUSION MATRIX

Actual/ Predicted	Not recommended	Recommended
Not preferred	True-Negative (TN)	False-positive (FP)
Preferred	False-Negative (FN)	True-Positive (TP)

$$\text{Precision} = TP/(TP+FP) \quad (10)$$

$$\text{Recall or TPR} = TP/(TP+FN) \quad (11)$$

$$FPR = 1\text{-specificity} = 1-TN/(TN+FP) = FP/(TN+FP) \quad (12)$$

$$\text{Accuracy} = (TN+TP)/(TN+FP+FN+TP) \quad (13)$$

$$F1 = 2*\text{precision}*\text{recall}/(\text{precision}+\text{recall}) \quad (14)$$

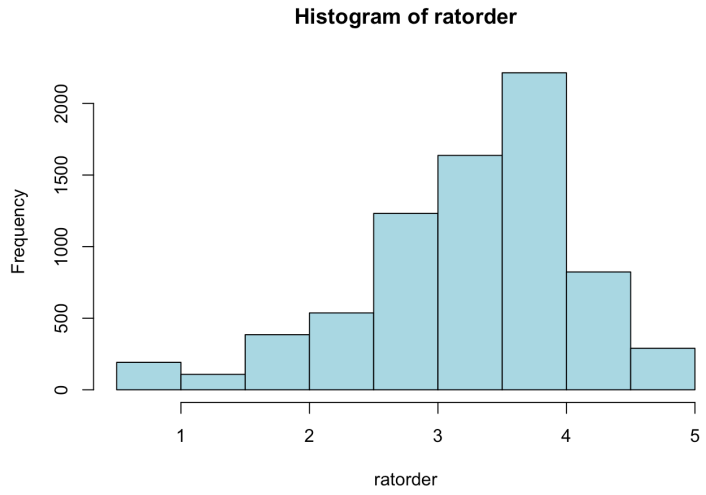
```
# split the data into the training and the test set:
e <- evaluationScheme(ratingmat, method="split", train=0.8, given=20, goodRating=3)
```

We use the Recommender function from Recommenderlab package, it automatically returns the result, RMSE is 0.9933889.

```
rpopular <- Recommender(getData(e, "train"), "POPULAR",param=list(normalize = NULL))
d <- predict(rpopular, getData(e, "known"), type="ratings")
rmse_popular <- calcPredictionAccuracy(d, getData(e, "unknown"))
rmse_popular
```

```
##      RMSE       MSE       MAE
## 0.9933889 0.9868216 0.7708772
```

We make a histogram of the prediction, it is left skewed as the movie rating frequency.



After that, we combined the movie ID and movie name together.

```
rating3<-left_join(rating,metadata %>% select(id, title),by=c("tmdbId"="id"))
head(rating3)
```

```
##   userId movieId rating  timestamp imdbId tmdbId                title
## 1      1      31    2.5 1260759144 112792   9909      Dangerous Minds
## 2      1    1029    3.0 1260759179  33563  11360                Dumbo
## 3      1    1061    3.0 1260759182 117665    819             Sleepers
## 4      1    1129    2.0 1260759185  82340   1103 Escape from New York
## 5      1    1172    4.0 1260759205  95765  11216      Cinema Paradiso
## 6      1    1263    2.0 1260759151  77416  11778      The Deer Hunter
```

We try to predict a specific user, for example, user 12's ratings to movies he/she haven't seen. Also provided the predicted score. Ideally, all user's prediction should be the same.

```
recom <- predict(rpopular,ratingmat[12], n=10) #Obtain top 10 recommendations for 42st user in dataset
recom_list <- as(recom, "list") #convert recommenderlab object to readable list
```

```
##       Movie Name                     Predicted Score
##  [1,] "The Mosquito Coast"           "5"
##  [2,] "Happy Gilmore"                "5"
##  [3,] "Willow"                       "5"
##  [4,] "Star Wars"                    "5"
##  [5,] "Lethal Weapon"                "5"
##  [6,] "The Giver"                    "5"
##  [7,] "The Double Life of Veronique" "5"
##  [8,] "Shadowlands"                  "5"
##  [9,] "Run Lola Run"                 "5"
## [10,] "Dangerous Minds"              "5"
```

User Based Collaborative Filtering (UBCF)

Strengths: User-based Collaborative Filtering gives recommendations that can be complements to the item the user was interacting with. This might be a stronger recommendation than what a content-based recommender can provide as users might not be looking for direct substitutes to a movie they had just viewed or previously watched. Individuals will find potential interests instead of the type of movie they already love.

Weaknesses: User-based Collaborative Filtering is a type of Memory-based Collaborative Filtering that uses all user data in the database to create recommendations. Comparing the pairwise correlation of every user in your dataset is not scalable. If there were millions of users, this computation would be very time consuming. Possible ways to get around this would be to implement some form of dimensionality reduction, such as Principal Component Analysis, or to use a model-based algorithm instead. Also, user-based collaborative filtering relies on past user choices to make future recommendations. The implications of this is that it assumes that a user's taste and preference remains more or less constant over time, which might not be true and makes it difficult to pre-compute user similarities offline (also called cold start problem).

User-based CF (Goldberg et al. 1992; Resnick, Iacovou, Suchak, Bergstrom, and Riedl 1994; Shardanand and Maes 1995) is a memory-based algorithm which tries to mimics word-of-mouth by analyzing rating data from many individuals. The assumption is that users with similar preferences will rate items similarly. Thus missing ratings for a user can be predicted by first finding a neighborhood of similar users and then aggregate the ratings of these users to form a prediction.

The neighborhood is defined in terms of similarity between users, either by taking a given number of most similar users (k nearest neighbors) or all users within a given similarity threshold. Popular similarity measures for CF are the Pearson correlation coefficient and the Cosine similarity and Euclidean Distance. These similarity measures are defined between two users u_x and u_y as

$$\mathrm{sim}_{\mathrm{Pearson}}(\vec{x},\vec{y}) = \frac{\sum_{i\in I}(\vec{x}_i\,\bar{\vec{x}})(\vec{y}_i\,\bar{\vec{y}})}{(|I|-1)\,\mathrm{sd}(\vec{x})\,\mathrm{sd}(\vec{y})}$$

and

$$\mathrm{sim}_{\mathrm{Cosine}}(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{\|\vec{x}\|\|\vec{y}\|},$$

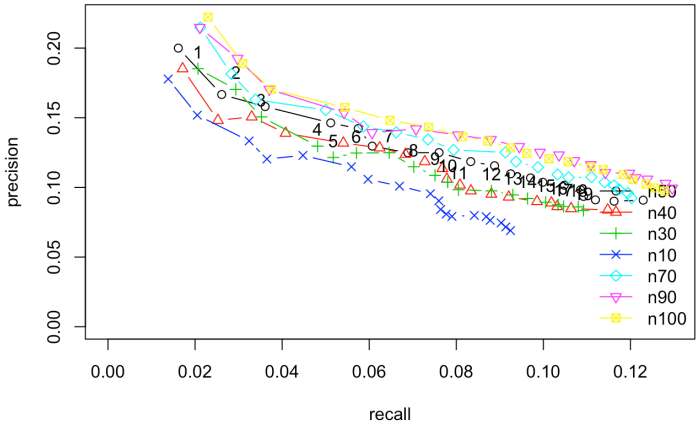
$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}.$$

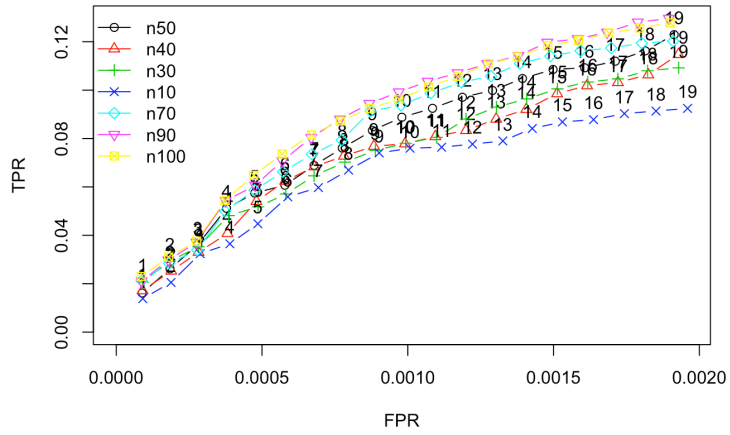
Where $\vec{x} = r_X$ and $\vec{y} = r_Y$ represent the row vectors in R with the two users' profile vectors. sd(·) is the standard deviation and $\| \cdot \|$ is the l^2-norm of a vector. For calculating similarity using rating data only the dimensions (items) are used which were rated by both users.

Now the neighborhood for the active user N(a) ⊂ U can be selected by either a threshold on the similarity or by taking the k nearest neighbors. Once the users in the neighborhood are found, their ratings are aggregated to form the predicted rating for the active user. The easiest form is to just average the ratings in the neighborhood.

$$\hat{r}_{aj} = \frac{1}{|\mathcal{N}(a)|} \sum_{i \in \mathcal{N}(a)} r_{ij}$$

Now we are going to find the best KNN, the best number of Nearest Neighbor changes every time. But most of time, n90 means set KNN equals to 90 has the best ROC curve.





We use Recommender function to do the prediction, UBCF_N_C stands for we use the raw dataset, knn=90, use cosine to calculate the user similarity. UBCF_C_C and UBCF_Z_C means the same except they use different transformed data, C stands for centering, Z stands for Z score. All following models' names are followed this pattern: UBCF means User Based Collaborative Filtering; the second letter means the condition of the dataset, if it is transformed; the last letter means the method we use to calculate the user similarity or so called user distance.

```
#train UBCF cosine similarity models

# non-normalized
UBCF_N_C <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = NULL, method="Cosine",nn=90 ))

# centered
UBCF_C_C <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = "center",method="Cosine",nn=90 ))

# Z-score normalization
UBCF_Z_C <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = "Z-score",method="Cosine",nn=90 ))
```

```
# compute predicted ratings
a1 <- predict(UBCF_N_C, getData(e, "known"), type="ratings")

a2 <- predict(UBCF_C_C, getData(e, "known"), type="ratings")

a3 <- predict(UBCF_Z_C, getData(e, "known"), type="ratings")
```

```
# aggregate the performance statistics
error_UCOS <- rbind(
  UBCF_N_C = calcPredictionAccuracy(a1, getData(e, "unknown")),
  UBCF_C_C = calcPredictionAccuracy(a2, getData(e, "unknown")),
  UBCF_Z_C = calcPredictionAccuracy(a3, getData(e, "unknown"))
)
kable(error_UCOS)
```

Predict function can return either top-N lists (default setting) or predicted ratings. object is the recommender object, newdata is the data for the active users. For top-N lists n is the maximal number of recommended items in each list and predict() will return an objects of class topNList which contains one top-N list for each active user. For "ratings" and "ratingMatrix", n is ignored and an object of realRatingMatrix is returned. Each row contains the predicted ratings for one active user. Here we set type equals to ratings.

	RMSE	MSE	MAE
UBCF_N_C	3.4898302	12.1789148	3.3184552
UBCF_C_C	0.9858655	0.9719308	0.7721794
UBCF_Z_C	0.9856041	0.9714154	0.7720178

Now we use Euclidean Distance to calculate user similarity.

```
#train UBCF Euclidean Distance models

# non-normalized
UBCF_N_E <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = NULL, method="Euclidean",nn=90))

# centered
UBCF_C_E <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = "center",method="Euclidean",nn=90))

# Z-score normalization
UBCF_Z_E <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = "Z-score",method="Euclidean",nn=90))
```

```
# compute predicted ratings
b1 <- predict(UBCF_N_E, getData(e, "known"), type="ratings")

b2 <- predict(UBCF_C_E, getData(e, "known"), type="ratings")

b3 <- predict(UBCF_Z_E, getData(e, "known"), type="ratings")
```

```
# aggregate the performance statistics
error_UEUC <- rbind(
  UBCF_N_E = calcPredictionAccuracy(b1, getData(e, "unknown")),
  UBCF_C_E = calcPredictionAccuracy(b2, getData(e, "unknown")),
  UBCF_Z_E = calcPredictionAccuracy(b3, getData(e, "unknown"))
)
kable(error_UEUC)
```

	RMSE	MSE	MAE
UBCF_N_E	3.4630854	11.9929608	3.2902528
UBCF_C_E	0.9831391	0.9665625	0.7691763
UBCF_Z_E	0.9816203	0.9635784	0.7681161

Same for Pearson Correlation

```
#train UBCF pearson correlation models

# non-normalized
UBCF_N_P <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = NULL, method="pearson",nn=90))

# centered
UBCF_C_P <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = "center",method="pearson",nn=90 ))

# Z-score normalization
UBCF_Z_P <- Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = "Z-score",method="pearson",nn=90 ))
```

```
# compute predicted ratings
c1 <- predict(UBCF_N_P, getData(e, "known"), type="ratings")

c2 <- predict(UBCF_C_P, getData(e, "known"), type="ratings")

c3 <- predict(UBCF_Z_P, getData(e, "known"), type="ratings")
```

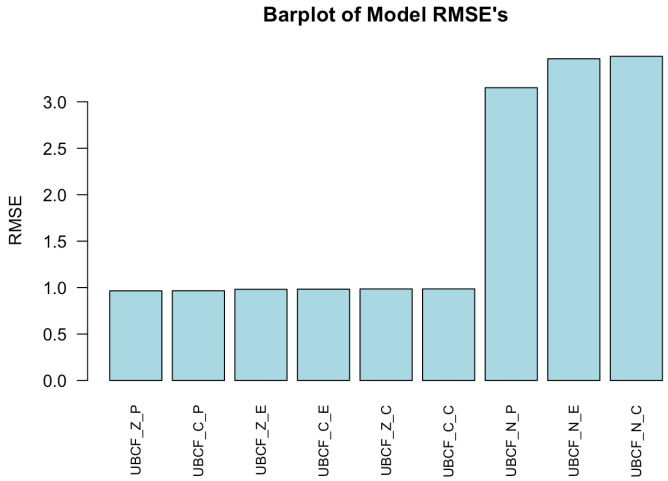
```
# aggregate the performance statistics
error_UPC <- rbind(
  UBCF_N_P = calcPredictionAccuracy(c1, getData(e, "unknown")),
  UBCF_C_P = calcPredictionAccuracy(c2, getData(e, "unknown")),
  UBCF_Z_P = calcPredictionAccuracy(c3, getData(e, "unknown"))
)
kable(error_UPC)
```

	RMSE	MSE	MAE
UBCF_N_P	3.1517304	9.9334043	2.9490972
UBCF_C_P	0.9660958	0.9333411	0.7521455
UBCF_Z_P	0.9650920	0.9314025	0.7513465

We combine all these RMSE tables and order RMSE by decreasing order.

	RMSE	MSE	MAE
UBCF_Z_P	0.9650920	0.9314025	0.7513465
UBCF_C_P	0.9660958	0.9333411	0.7521455
UBCF_Z_E	0.9816203	0.9635784	0.7681161
UBCF_C_E	0.9831391	0.9665625	0.7691763
UBCF_Z_C	0.9856041	0.9714154	0.7720178
UBCF_C_C	0.9858655	0.9719308	0.7721794
UBCF_N_P	3.1517304	9.9334043	2.9490972
UBCF_N_E	3.4630854	11.9929608	3.2902528
UBCF_N_C	3.4898302	12.1789148	3.3184552

Make a histogram. Since the difference of each model are quite small, we can't tell the huge difference. But we can clearly see that UBCF_N_P, UBCF_N_E, UBCF_N_C all have very high RMSE comparing to other models. The second letter represents the data transformation, N stands for Null, which means we use the raw data. Hence it is necessary to normalize the dataset.

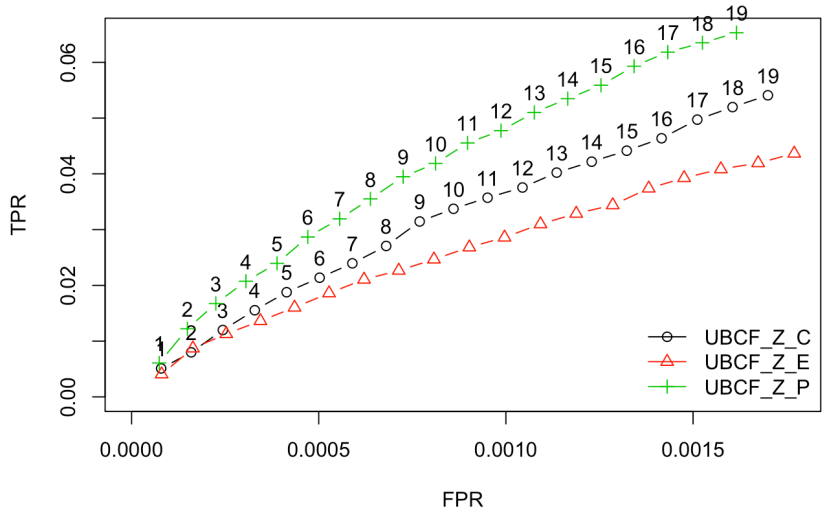


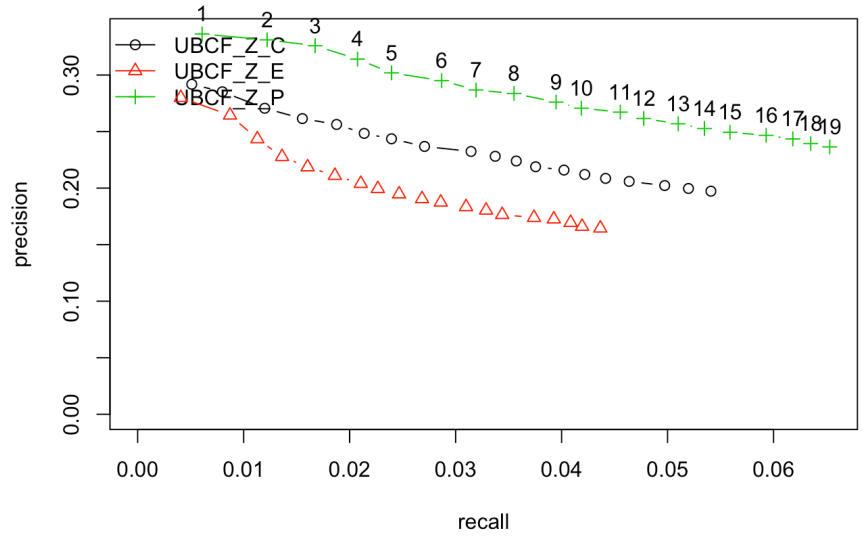
Since all models performs great, the difference of RMSE is small, it is a bit hard to do model selection. We chose the Z score normalized data to find the best method to do calculate the user similarity. k=5 meaning a 5-fold cross validation. given=20 meaning a Given-20 protocol. threshold for determining what rating is a good rating. Used only if x is a topNList and data is a "realRatingMatrix".

```
evaluation_scheme <- evaluationScheme(ratingmat, method="cross-validation", k=5, given=20, goodRating=3)

algorithms <- list(
UBCF_Z_C = list(name = "UBCF", param = list(normalize = "Z-score",method = "Cosine")),
UBCF_Z_E = list(name = "UBCF", param = list(normalize = "Z-score",method = "Euclidean")),
UBCF_Z_P = list(name = "UBCF", param = list(normalize = "Z-score",method = "pearson"))
)



evaluation_results <- evaluate(evaluation_scheme,algorithms, type="topNList",n=1:19)
```

The running time is extremely fast.

```
## UBCF run fold/sample [model time/prediction time]
##   1  [0.04sec/4.156sec]
##   2  [0.026sec/4.843sec]
##   3  [0.026sec/4.492sec]
##   4  [0.031sec/4.944sec]
##   5  [0.027sec/4.393sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.024sec/4.232sec]
##   2  [0.024sec/4.323sec]
##   3  [0.029sec/4.767sec]
##   4  [0.024sec/4.462sec]
##   5  [0.03sec/4.184sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.024sec/2.59sec]
##   2  [0.026sec/2.681sec]
##   3  [0.023sec/2.682sec]
##   4  [0.025sec/2.615sec]
##   5  [0.03sec/2.978sec]
```

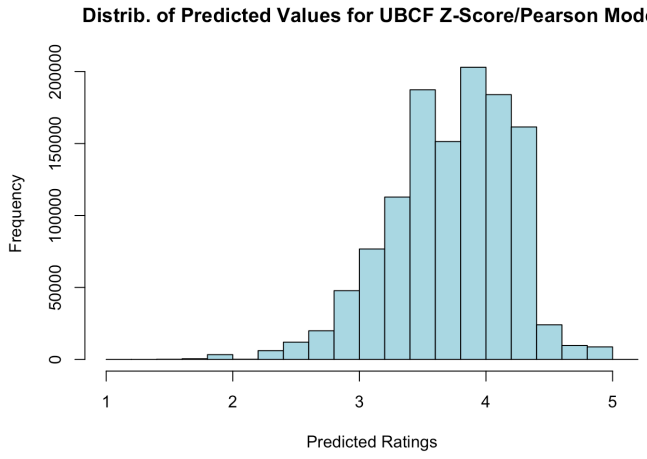


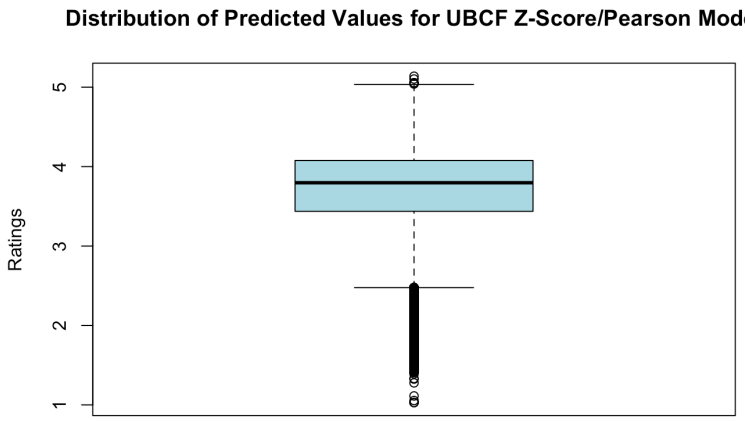


By looking at the ROC curve, it is very clear that UBCF_Z_P performs the best, it outperformed UBCF_Z_C and UBCF_Z_O a lot. Below is the UBCF_Z_P's confusion matrix by using the 5 cross validation. The output is the average of those 5 folds.

```
##           TP         FP       FN       TN precision      recall
## 1  0.3185185  0.6814815 96.35556 8948.644 0.3185185 0.007006394
## 3  0.9851852  2.0148148 95.68889 8947.311 0.3283951 0.018966219
## 5  1.5407407  3.4592593 95.13333 8945.867 0.3081481 0.026766769
## 10 2.7407407  7.2592593 93.93333 8942.067 0.2740741 0.044909489
## 15 3.6222222 11.3777778 93.05185 8937.948 0.2414815 0.055384952
## 20 4.5851852 15.4148148 92.08889 8933.911 0.2292593 0.067279237
##            TPR          FPR
## 1  0.007006394 7.581551e-05
## 3  0.018966219 2.240743e-04
## 5  0.026766769 3.847909e-04
## 10 0.044909489 8.075202e-04
## 15 0.055384952 1.266287e-03
## 20 0.067279237 1.715894e-03
```

Make a histogram & boxplot of prediction using the UBCF_Z_P model, which is the best model.

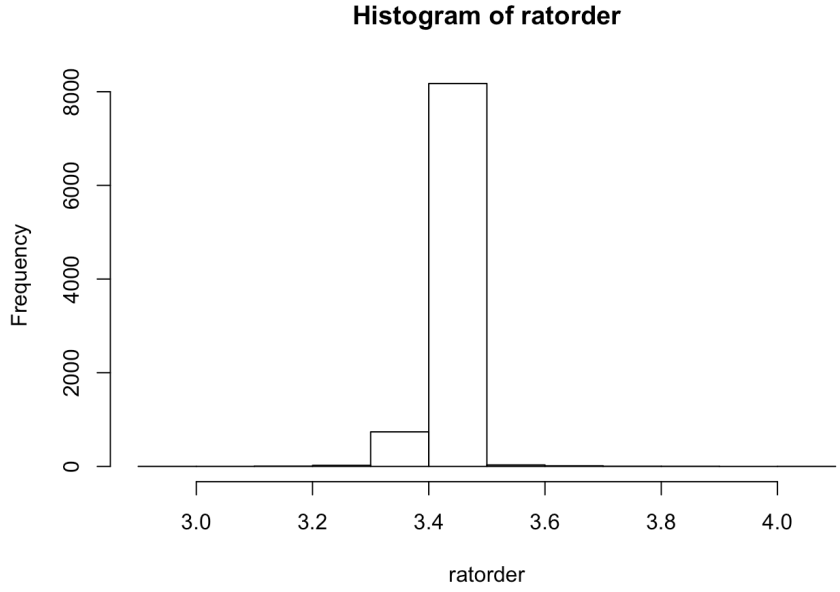




We want to predict user 58's prediction. The max rating he/she will give to a movie he/she haven't seen is 4.074508. This rating is not equal to the maximum number of rating score 5.0. This condition is reasonable since we are using K Nearest Neighbor method here, the user's predicted rating is heavily affected by other users. From the data exploring, we know that the movie rating distribution is left skewed, most users will rate a movie at score 4.0. Hence a specific user's predicted maximum score should be around 4.

```
##      318
## 4.074508
```

Make a histogram of user 58's prediction. The distribution of this specific user is also left skewed, user 48 tends to rate a movie at score 3.4



Ordered by the predicted score we just calculated, the output returns the top10 movie user 48 would like.

```
##       Movie Name                Predicted Score
##  [1,] "The Mosquito Coast"      "4.07450754522954"
##  [2,] "Shadowlands"             "3.89827967909442"
##  [3,] "Happy Gilmore"           "3.89219783902496"
##  [4,] "Batman"                  "3.8430654159295"
##  [5,] "Star Wars"               "3.79987328485231"
##  [6,] "Willow"                  "3.77062957387128"
##  [7,] "To Die For"              "3.73012056746307"
##  [8,] "Buena Vista Social Club" "3.71151190708168"
##  [9,] "Pulp Fiction"            "3.68925443458709"
## [10,] "Leon: The Professional"  "3.68486784608961"
```

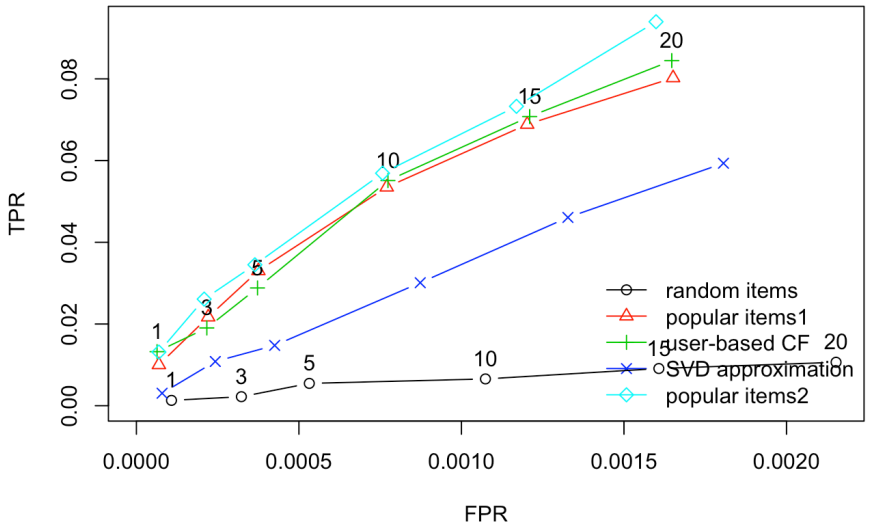
We just skipped the Item Base Collaborative Filtering (IBCF) since our dataset has about 600 users, but we have about 9000 movies. in this circumstance, which is item number is much more than users, UBCF is much better to use since it is easier for us to find the relationship between small amount of users; for IBCF, the matrix is too large for the system to find relationship for each items, it might need a large RAM to do the calculation.

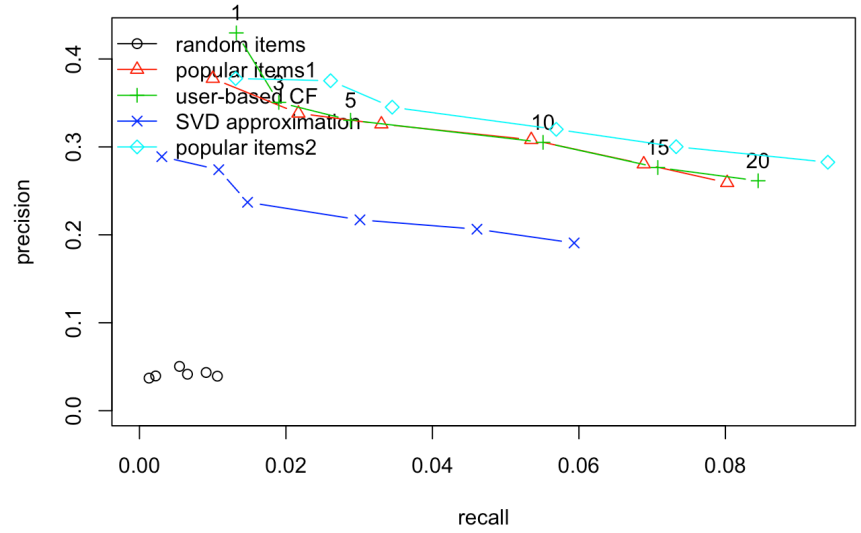
Memory Based Model Comparison

We compared several models here. The popular items2 is just the popular model we talked previous; popular item1 is the popular model that used the centering transformed data; random item is the raw data, we just randomly recommend any movies to users without using any algorithm; SVD Approximation is not a memory based model, it is a model based, but we want to test its performance comparing to memory based model; user-based CF is the UBCF we talked previously, it selects nn=100, uses z-score transformed data and use Pearson Correlation to calculate user distance.

```
algorithms <- list(
"random items" = list(name="RANDOM", param=NULL),
"popular items1" = list(name="POPULAR", param=list(normalize = "center")),
"user-based CF" = list(name="UBCF", param=list(nn=100,normalize="z-score",method="pearson")),
"SVD approximation" = list(name="SVD", param=list(k = 50)),
"popular items2" = list(name="POPULAR", param=list(normalize = NULL))
)

# run algorithms
results <- evaluate(e, algorithms, type = "topNList",
 n=c(1, 3, 5, 10, 15, 20))
```

The poppular2 method perform the best, hence we should not do any transformation to the data when applying the Most Popular(item average) method. Then follows the User-based CF, popular1, SVD Approximation. The random items performs the worst, it is reasonable since it just randomly recommend movies. Hence it is quite important to use our recommender engine to do suggestion.
Another interesting insight is that popular2 method is the best model rather than the User-based CF. It is a little surprising that User-based CF is not the best one since this algorithm is working on specific users and calculate every user's predictions, not like Popular(item based) method which returns the same output for every users. From the following plots, we can draw an interesting conclusion that we are not that different, we all like popular stuffs.





Memory Based (Method-Matrix Factorization)

- Gradient Descent
- Singular Value Decomposition (SVD) Approximation
- Simon Funk SVD
- Alternating Least Square (ALS)
- Hybrid
- Slope One

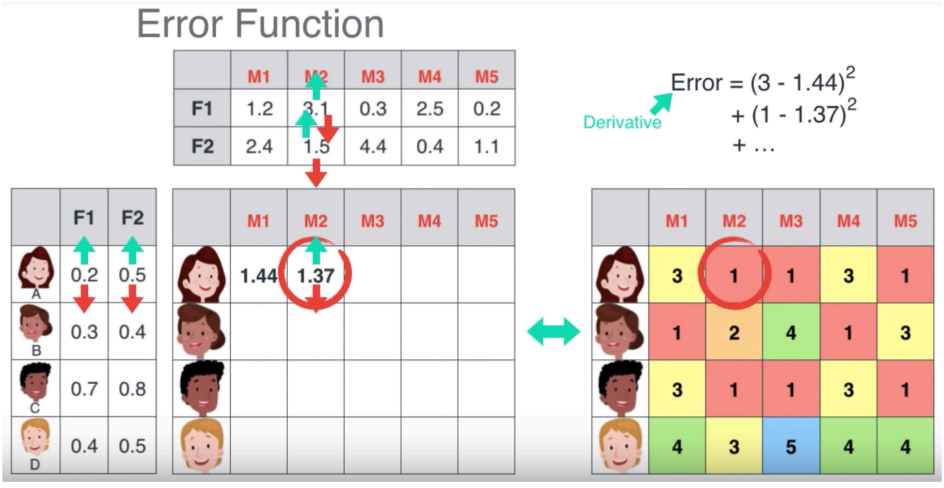
1. Model learns to factorize rating matrix into user and movie representations, which allows model to predict better personalized movie ratings for users
2. With matrix factorization, less-known movies can have rich latent representations as much as popular movies have, which improves recommender's ability to recommend less-known movies

Gradient Descent

A popular technique to solve the recommender system problem is the matrix factorization method. The idea is to approximate the whole rating matrix $R_{m\times n}$ by the product of two matrices of lower dimensions, $P_{k\times m}$ and $Q_{k\times n}$, such that $R \approx P'Q$

Let p_u be the u-th column of P, and q_v be the v-th column of Q, then the rating given by user u on item v would be predicted as $p'_u q_v$.

A question might have come to your mind by now: if we find two matrices P and Q such that $P\times Q$ approximates R, isn't that our predictions of all the unseen ratings will be zeros? In fact, we are not really trying to come up with P and Q such that we can reproduce R exactly. Instead, we will only try to minimize the errors of the observed user-item pairs. In other words, if we let T be a set of tuples, each of which is in the form of (ui,dj,rij), such that T contains all the observed user-item pairs together with the associated ratings, we are only trying to minimize every eij for $(ui,dj,rij) \in T$. (In other words, T is our set of training data.) As for the rest of the unknowns, we will be able to determine their values once the associations between the users, items and features have been learnt.



Using the above update rules, we can then iteratively perform the operation until the error converges to its minimum. We can check the overall error as calculated using the following equation and determine when we should stop the process.

$$E = \sum_{(u_i, d_j, r_{ij}) \in T} e_{ij} = \sum_{(u_i, d_j, r_{ij}) \in T} (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

The above algorithm is a very basic algorithm for factorizing a matrix. There are a lot of methods to make things look more complicated. A common extension to this basic algorithm is to introduce regularization to avoid overfitting.

Increasing the number of latent factors will improve personalization, until the number of factors becomes too high, at which point the model starts to overfit. A common strategy to avoid overfitting is to add regularization terms to the objective function.

This is done by adding a parameter β and modify the squared error as follows:

$$e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (||P||^2 + ||Q||^2)$$

In other words, the new parameter β is used to control the magnitudes of the user-feature and item-feature vectors such that P and Q would give a good approximation of R without having to contain large numbers.

When predicting the ratings of users given to items, it is useful to consider how ratings are generated. In the above discussion, we have assumed that ratings are generated based on matching the users preferences on some latent factors and the items' characteristics on the latent factors.

A typical solution for P and Q is given by the following optimization problem (Chin, Zhuang, et al. 2015a; Chin, Zhuang, et al. 2015b), We will use this formula in our code below):

$$\min_{P,Q} \sum_{(u,v) \in R} \left[f(p_u, q_v; r_{u,v}) + \mu_P ||p_u||_1 + \mu_Q ||q_v||_1 + \frac{\lambda_P}{2} ||p_u||_2^2 + \frac{\lambda_Q}{2} ||q_v||_2^2 \right]$$

where (u,v) are locations of observed entries in R, ru,v is the observed rating, f is the loss function, and μP,μQ,λP,λQ are penalty parameters to avoid overfitting.

The process of solving the matrices P and Q is referred to as model training, and the selection of penalty parameters is called parameter tuning.

We are using the recosystem package here to do Matrix Factorization with Gradient Descent.

Like we talked about it before, Matrix Factorization is a popular technique to solve recommender system problem. The main idea is to approximate the matrix $R(m \times n)$ by the product of two matrixes of lower dimension: P (k x m) and Q (k x n).

Matrix P represents latent factors of users. So, each k-elements column of matrix P represents each user. Each k-elements column of matrix Q represents each item. So, to find rating for item i by user u we simply need to compute two vectors: $P[,u]' \times Q[,i]$.

This package works with data saved on disk in 3 columns with no headers and we separate the whole dataset into train and test set.

```
ratings_train <- rating4[(in_train)]
ratings_test <- rating4[(!in_train)]

ratings_train
```

```
##        userId movieId rating
##     1:      1      31    2.5
##     2:      1    1029    3.0
##     3:      1    1061    3.0
##     4:      1    1129    2.0
##     5:      1    1172    4.0
##    ---
## 99329:    671    6268    2.5
## 99330:    671    6269    4.0
## 99331:    671    6365    4.0
## 99332:    671    6385    2.5
## 99333:    671    6565    3.5
```

```
ratings_test
```

```
##      userId movieId rating
##   1:      2      10      4
##   2:      4    1334      5
##   3:      4    1918      2
##   4:      4    2096      5
##   5:      4    2124      4
##  ---
## 667:    664   68157      4
## 668:    665    1073      5
## 669:    665    2571      5
## 670:    665    4333      2
## 671:    671    3481      2
```

```
#Store train and test as text file in working directory
write.table(ratings_train, file = "trainset.txt", sep = " ", row.names = FALSE, col.names = FALSE)
write.table(ratings_test, file = "testset.txt", sep = " ", row.names = FALSE, col.names = FALSE)


#Build Recommender object
r = Reco()

#index1 = TRUE, since our user and movie index start from 1
train_data <- data_file('trainset.txt', index1 = TRUE)
test_data <- data_file('testset.txt', index1 = TRUE)
```

Following is the list of parameters that we can tune, the list from the Recosystem package document:

Parameters and Options

The `opts` argument should be a list that provides the candidate values of tuning parameters and some other options. For tuning parameters (`dim`, `costp_l1`, `costp_l2`, `costq_l1`, `costq_l2`, and `lrate`), users can provide a numeric vector for each one, so that the model will be evaluated on each combination of the candidate values. For other non-tuning options, users should give a single value. If a parameter or option is not set by the user, the program will use a default one.

See below for the list of available parameters and options:

`dim` Tuning parameter, the number of latent factors. Can be specified as an integer vector, with default value `c(10L, 20L)`.

`costp_l1` Tuning parameter, the L1 regularization cost for user factors. Can be specified as a numeric vector, with default value `c(0, 0.1)`.

`costp_l2` Tuning parameter, the L2 regularization cost for user factors. Can be specified as a numeric vector, with default value `c(0.01, 0.1)`.

`costq_l1` Tuning parameter, the L1 regularization cost for item factors. Can be specified as a numeric vector, with default value `c(0, 0.1)`.

`costq_l2` Tuning parameter, the L2 regularization cost for item factors. Can be specified as a numeric vector, with default value `c(0.01, 0.1)`.

`lrate` Tuning parameter, the learning rate, which can be thought of as the step size in gradient descent. Can be specified as a numeric vector, with default value `c(0.01, 0.1)`.

`loss` Character string, the loss function. Default is "l2", see section **Parameters and Options** in `$train()` for details.

`nfold` Integer, the number of folds in cross validation. Default is 5.

`niter` Integer, the number of iterations. Default is 20.

`nthread` Integer, the number of threads for parallel computing. Default is 1.

`nbin` Integer, the number of bins. Must be greater than `nthread`. Default is 20.

`nmf` Logical, whether to perform non-negative matrix factorization. Default is `FALSE`.

`verbose` Logical, whether to show detailed information. Default is `FALSE`.

`progress` Logical, whether to show a progress bar. Default is `TRUE`.

Now we could tune parameters of Recommender: number of latent factors (dim), gradient descend step rate (lrate), and penalty parameter to avoid overfitting (cost). To make this easier and faster to run, cost could be set to some small value and dim while tuning will adopt to it: here we will set L1 regulation to zero, only use the L2 regulation by the default value; the dimension, number of latent factors is set from 1 to 20, to find the best dim number by looking at which one has the lowest RMSE; the learning rate is 0.05, with a small number, we can get a more precise number instead of missed the smallest point. The output shows we should chose 19 latent factors, the L2 regulation should set to 0.01.

```
## $min
## $min$dim
## [1] 19
##
## $min$costp_l1
## [1] 0
##
## $min$costp_l2
## [1] 0.01
##
## $min$costq_l1
## [1] 0
##
## $min$costq_l2
## [1] 0.1
##
## $min$lrate
## [1] 0.05
##
## $min$loss_fun
## [1] 0.9769595
```

After we tune the parameter, we write the predicted values into file and calculate the RMSE, which is 0.9045648.

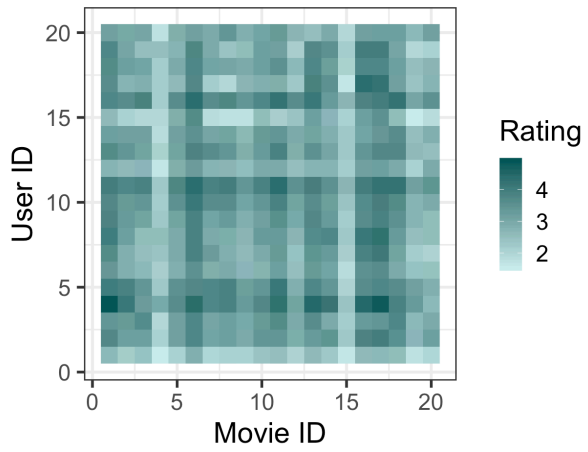
```
# Write predicted values into file
out_pred = out_file(tempfile())
r$predict(test_data, out_pred)
```

```
## prediction output generated at /var/folders/mw/m_16_rzd0r9d_ffj3g34tkp40000gn/T//RtmpuKKlZ5/fileff294291ee95
```

```
scores_real <- read.table('testset.txt', header = FALSE, sep = " ")$V3
scores_pred <- scan(out_pred@dest)

rmse_mf <- sqrt(mean((scores_real-scores_pred) ^ 2))
rmse_mf
```

```
## [1] 0.9045648
```

In the end, we predict the first 20 users' ratings to the first 20 movies.

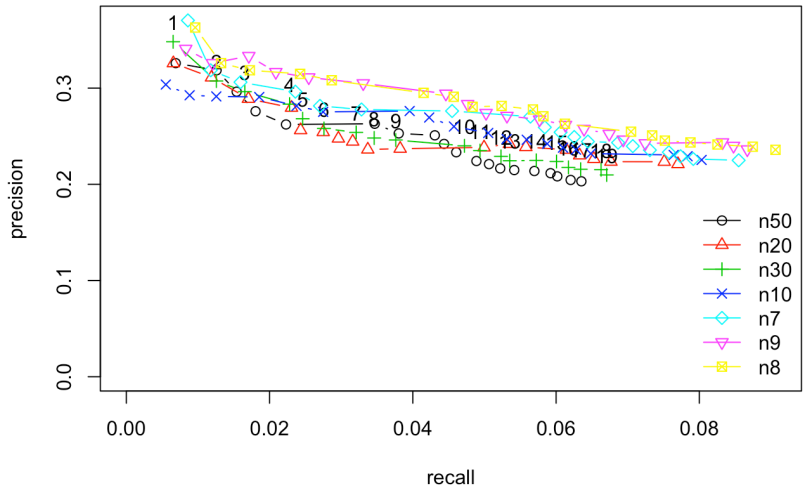


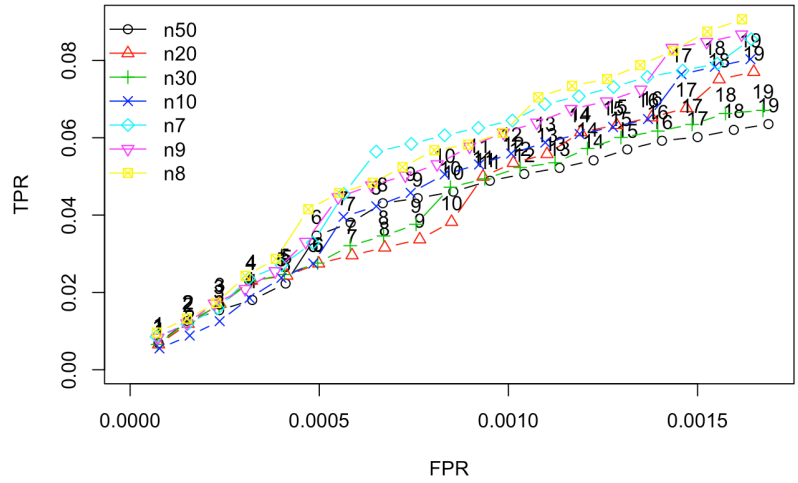
Singular Value Decomposition (SVD) Approximation

We use the recommenderlab package again. Following is the default parameter and description of SVD from the recommenderlab package document. Here K stands for the latent factors.

```
$SVD_realRatingMatrix
Recommender method: SVD for realRatingMatrix
Description: Recommender based on SVD approximation with column-mean imputation.
Reference: NA
Parameters:
   k maxiter normalize
1 10     100  "center"
```

First, we again find the best K by looking at the ROC curve, most of the time, when k=7, which means we will select 7 latent factors, produces the best prediction. The latent number is the same as our Matrix Gradient Descent's latent number since when we tried to run the the gradient descent method for our code several times, sometimes it returns 7 as the best dim. We are in the right track.





Since we only use the column mean to do the imputation of NA, the SVD Approximation method has a high RMSE, 0.9860490.

```
##      RMSE       MSE       MAE
## 0.9860490 0.9722926 0.7630577
```

Simon Funk SVD

Difference between SVD and SGD:

Simon funk is the apparent inventor of a simple & ingenious SVD (singular value decomposition) algorithm during the netflix contest although the algorithm may have predated his discovery. SGD = stochastic gradient descent which can be applied to all kinds of optimization problems, including SVD.

SGD may find a superior solution but it may take longer. so basically both SGD/Funk are two approaches to computing the U,V matrices. By SGD, the different factors in pu (and qi) are all updated at the same time. Funk's original algorithm was a bit different: he actually trained the first factor, then the second, then the third, etc. This gave his algorithm a more SVDesque flavor. A nice discussion about this can be found in Aggarwal's Textbook on recommender systems.

Once all the vectors pu and qi have been computed, we can estimate all the ratings we want using the formula:

$$\hat{r}_{ui} = p_u \cdot q_i.$$

There's a hat on r hat to indicate that it's an estimation, not its real value.

We are using recommenderlab package to do the Simon Funk SVD. The parameter description is write below.

x: a matrix, potentially containing NAs.

k: number of features (i.e, rank of the approximation).

gamma: regularization term.

lambda: learning rate.

min_improvement: required minimum improvement per iteration.

min_epochs: minimum number of iterations per feature.

max_epochs: maximum number of iterations per feature.

verbose: show progress.

we use the default parameters and get a quite small RMSE, 0.9070839

```
##      RMSE       MSE       MAE
## 0.9079839 0.8244347 0.6861910
```

Alternating Least Square (ALS)

Alternating Least Square (ALS) is also a matrix factorization algorithm and it runs itself in a parallel fashion. ALS is implemented in Apache Spark ML and built for a large-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.

Some high-level ideas behind ALS are:

- Its objective function is slightly different than Funk SVD: ALS uses L2 regularization while Funk uses L1 regularization
- Its training routine is different: ALS minimizes two loss functions alternatively; It first holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix

By using the Recommenderlab package, we calculate the RMSE of ALS, 0.9796331

```
##      RMSE       MSE       MAE
## 0.9796331 0.9596809 0.7610202
```

Hybrid

Creates and combines recommendations using several recommender algorithms. We brought together ideas from model based method and memory based method to build an engine that gave movie suggestions to a particular user based on the estimated ratings that it had internally calculated for that user.

we set the weights of recommender algorithm to be SVD, UBCF and Popular to be 0.1, 0.6, 0.3. We put the tuned parameters that have found previously in Collaborative Filtering part into the list function. The combined algorithm's RMSE is lower than all these method runs separately.

```
recom <- HybridRecommender(
      Recommender(getData(e, "train"), "svd", param=list(k = 7)),
      Recommender(getData(e, "train"), "UBCF",
      param=list(normalize = "Z-score",method="pearson",nn=90 )),
      Recommender(getData(e, "train"), "POPULAR",param=list(normalize = NULL)),
      weights = c(.1,.6,.3))
a11<- predict(recom, getData(e, "known"), type="ratings")
a11
```

```
##      RMSE       MSE       MAE
## 0.9519534 0.9062152 0.7312905
```

Slope One

m1	m2	
u1	4	5
—	——	——
u2	4	3
—	——	——
u3	2	3
—	——	——
u4	3	?
—	——	——

The column represents user movie 1 to 3, column represents user 1 to 3. We want to calculate the question mark, the rating user 4 will rate the movie 3. It's quite easy to do the calculation: [(4-5)+(4-3)+(2-3)]/3 = -0.333 3-(-0.333) = 3.333

The method mainly solve two issues: the reduce of accuracy with the increasing sparsity resulted from new users and items entering the system; the worse scalability is due to the surge of nearest neighbor searching space caused by the expansion of system.

Since it needs a lot of RAM, I didn't run the code below. It uses SlopeOne package in R.

```
names(ratings) <- c("user_id", "item_id", "rating") ratings <- data.table(ratings)

ratings[, user_id := as.character(user_id)] ratings[, item_id := as.character(item_id)]

setkey(ratings, user_id, item_id)

set.seed(1)

in_train <- rep(TRUE, nrow(ratings)) in_train[sample(1:nrow(ratings), size = round(0.2 *
length(unique(ratings$user_id)), 0) * 5)] <- FALSE

ratings_train <- ratings[(in_train)] ratings_test <- ratings[(!in_train)]

ratings_train_norm <- normalize_ratings(ratings_train)

model <- build_slopeone(ratings_train_norm$ratings)

predictions <- predict_slopeone(model, ratings_test[ , c(1, 2), with = FALSE], ratings_train_norm$ratings)
unnormalized_predictions <- unnormalize_ratings(normalized = ratings_train_norm, ratings = predictions)

rmse_slopeone <- sqrt(mean(((unnormalized_predictions$predicted_rating) − (ratings_test$rating)) ^ 2))
rmse_slopeone
```

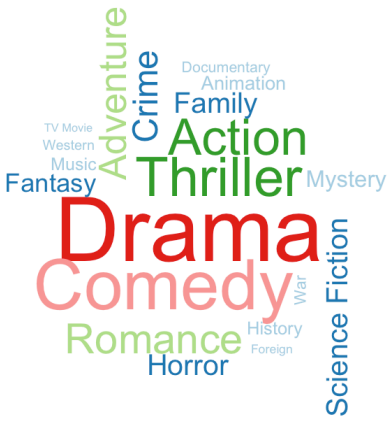
Content Based

Based on genres recommendation

there are 20 unique genres

```
##  [1] Action          Adventure       Fantasy         Science Fiction
##  [5] Crime           Drama           Thriller        Animation
##  [9] Family          Western         Comedy          Romance
## [13] Horror          Mystery         History         War
## [17] Music           Documentary     Foreign         TV Movie
## 20 Levels: Action Adventure Animation Comedy Crime Documentary ... Western
```

World cloud of genres. Drama and comedy is the most frequently appeared genre.



We make a table showing movie id 58, 6479, 19995, 49026's genres.

id	title	g1	g2	g3	g4
58	Pirates of the Caribbean: Dead Man's Chest	Adventure	Fantasy	Action	NA
6479	I Am Legend	Drama	Horror	Action	Thriller
19995	Avatar	Action	Adventure	Fantasy	Science Fiction
49026	The Dark Knight Rises	Action	Crime	Drama	Thriller

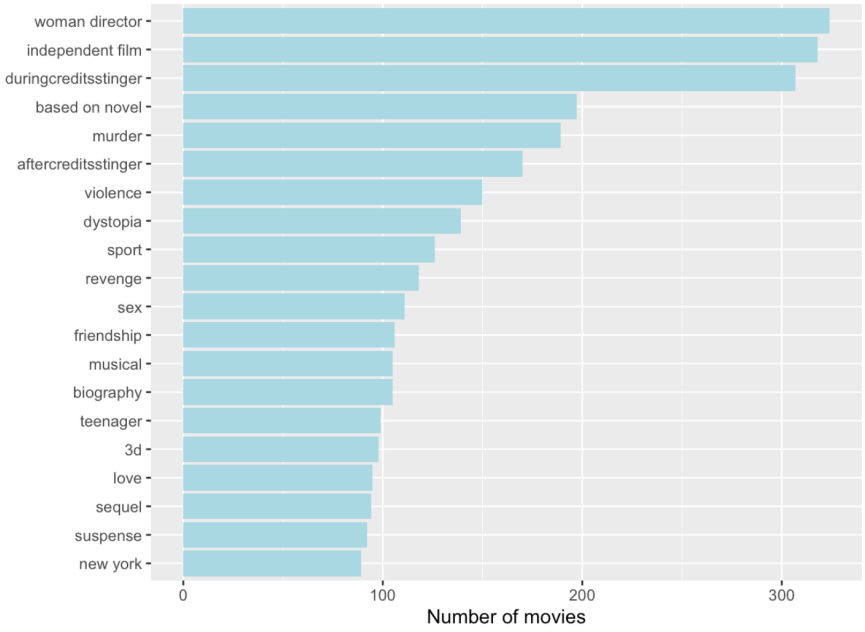
We build a function to recommend users by providing genres they want to see, for example, we want to see romance movies. The engine will return romance movies by the IMDB's score in the decreasing order.

```
recommendGenre <- function(Genre) {
    top_movie_genres %>% filter(genre==Genre)%>% slice(1:5)
}
```

```
kable(recommendGenre(Genre = "Romance"))
```

id	title	genre	score
13	Forrest Gump	Romance	8.020698
152601	Her	Romance	7.624375
38	Eternal Sunshine of the Spotless Mind	Romance	7.596430
266856	The Theory of Everything	Romance	7.489064
194	Amélie	Romance	7.488987

The movie's keywords histogram & word cloud in below. Women director keywords appears the most.



Recommend by plotting/Natural Language Processing

In the following content-based engine, for my laptop's limited storage, I only use first 70 movie overviews to do natural language processing.

tf-idf, short for term frequency–inverse document frequency, is a numeric measure that is use to score the importance of a word in a document based on how often did it appear in that document and a given collection of documents. The intuition for this measure is : If a word appears frequently in a document, then it should be important and we should give that word a high score. But if a word appears in too many other documents, it's probably not a unique identifier, therefore we should assign a lower score to that word. The math formula for this measure :

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

Where t denotes the terms; d denotes each document; D denotes the collection of documents.

The first part of the formula tf(t,d) is simply to calculate the number of times each word appeared in each document. Of course, as with common text mining methods: stop words like "a", "the", punctuation marks will be removed beforehand and words will all be converted to lower cases. There are 1508 different words in the first 70 movies.

```
doc_corpus <- Corpus( VectorSource(overviewdf_70$overview) )
control_list <- list(removePunctuation = TRUE, stopwords = TRUE, tolower = TRUE)
tdm <- TermDocumentMatrix(doc_corpus, control = control_list)

# print
tf <- as.matrix(tdm)
dim(tf)
```

```
## [1] 1508   70
```

IDF Inverse Document Frequency, the complete math formula for IDF.

$$idf(t,D) = log\frac{|D|}{1 + |\{d \in D : t \in d\}|}$$

The numerator: D is inferring to our document space. It can also be seen as D = d1,d2,…,dn where n is the number of documents in your collection. Thus for our example | D |, the size of our document space is 70, since we're only using 70 documents.

The denominator : | {d∈D:t∈d} | implies the total number of times in which term t appeared in all of your document d (the d∈D restricts the document to be in your current document space). Note that this implies it doesn't matter if a term appeard 1 time or 100 times in a document, it will still be counted as 1, since it simply did appear in the document. As for the plus 1, it is there to avoid zero division.

Using our term frequency matrix, the idf weight for can be calculated like below.

```
idf <- log( ncol(tf) / ( 1 + rowSums(tf != 0) ) )
```

Now that we have our matrix with the term frequency and the idf weight, we're ready to calculate the full tf-idf weight. To do this matrix multiplication, we will also have to transform the idf vector into a diagonal matrix. Both calculations are shown below. Since the dim is 1508, after we do the diag function, it becomes a 1508*1508 matrix.

```
 idf <- diag(idf)
dim(idf)
```

```
## [1] 1508 1508
```

There's still one more step to do for this tf-idf matrix. Recall that in the tf (term frequency) section, we're representing each term as the number of times they appeared in the document. The main issue for this representation is that it will create a bias towards long documents, as a given term has more chance to appear in longer document, making them look more important than actually they are.

Thus the approach the resolve this issue is the good old L2 normalization. Math formula :

$$\hat{v} = \frac{\vec{v}}{\|\vec{v}\|}$$

For each vector v, divide it by its norm (length, magnitude). Calculation as below

```
tf_idf <- crossprod(tf, idf)
colnames(tf_idf) <- rownames(tf)

new_tf_idf<-tf_idf / sqrt( rowSums( tf_idf^2 ) )
```

Now that we have this tf-idf matrix, one thing we can do with it is to perform text clustering !!

To performing document clustering using the tf-idf weight matrix, we'll use the cosine similarity to measure how close are two given documents. Math formula :

$$cos(\theta) = \frac{v \cdot w}{\|v\|\|w\|} = \frac{\sum_{i=1}^{n} v_i w_i}{\sqrt{\sum_{i=1}^{n} v_i^2}\sqrt{\sum_{i=1}^{n} w_i^2}}$$

Where v and w are the two vectors that you wish to calculate the distance; vi and wi are components of vector v and w respectively; and n is the number of components you have. After calculating cos(θ), you can also obtain the actual θ (degree) using the acos function in R. Note that the function returns the radian, you have to multiply it by 180 and divide by pi to obtain the actual degrees. As for why we're using this distance measure, remember what we've said in the normalization part, since documents are usually not of equal length, simply computing the difference between two vectors by using Euclidean distance has the disadvantage that documents of similar content but different length are not regarded as similar in the vector space.

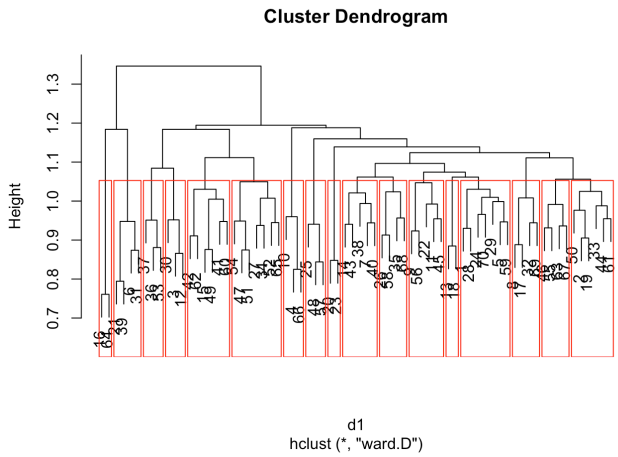
```
Cosine <- function(x, y) {
    similarity <- sum(x * y) / ( sqrt( sum(y ^ 2) ) * sqrt( sum(x ^ 2) ) )

    # given the cosine value, use acos to convert back to degrees
    # acos returns the radian, multiply it by 180 and divide by pi to obtain degrees
    return( acos(similarity) * 180 / pi )
}
```

```
d1 <- dist(new_tf_idf, method = "Cosine")

d1
```

Performs hierarchical clustering and visualize the clustering result with a genogram. Note that we won't be needing to normalize the tf-idf matrix before calculating the cosine distance, cosine distance will do that for us.

```
cluster1 <- hclust(d1, method = "ward.D")
plot(cluster1)
rect.hclust(cluster1, 17)
```

We separate the 70 overviews into17 groups(an arbitrary number) by the cosine distance.



We'll examine a potential cluster that the algorithm provided and print out the original overview to determine whether the result matches our intuition. Here we test the group 3. Given the below output, they all mention about James Bond, hence the separation is quite convincing. However, I don't suggest this method: 1. to choose the optimal group number need write more code, here I just randomly chose a number 17, we can't know which number will return the best output 2. we want to create an engine that can return similar movies to us, we are not interested in clustering, but this clustering is interesting and give us a new point of view.

```
# split into 17 clusters
groups1 <- cutree(cluster1, 17)

# you can look at the distribution size of each cluster
# table(groups1)

overviewdf_70$overview[groups1 == 3 ]
```

```
## [1] "A cryptic message from Bond's past sends him on a trail to uncover a sinister organization. While M battl
es political forces to keep the secret service alive, Bond peels back the layers of deceit to reveal the terrible
truth behind SPECTRE."
## [2] "Quantum of Solace continues the adventures of James Bond after Casino Royale. Betrayed by Vesper, the wom
an he loved, 007 fights the urge to make his latest mission personal. Pursuing his determination to uncover the t
ruth, Bond and M interrogate Mr. White, who reveals that the organization that blackmailed Vesper is far more com
plex and dangerous than anyone had imagined."
## [3] "When Bond's latest assignment goes gravely wrong and agents around the world are exposed, MI6 is attacked
forcing M to relocate the agency. These events cause her authority and position to be challenged by Gareth Mallor
y, the new Chairman of the Intelligence and Security Committee. With MI6 now compromised from both inside and ou
t, M is left with one ally she can trust: Bond. 007 takes to the shadows - aided only by field agent, Eve - follo
wing a trail to the mysterious Silva, whose lethal and hidden motives have yet to reveal themselves."
```

We can do the movie recommendation based on providing a specific movie id and return top 3 movie that has similar overview with the provided movie. We randomly choose movie 66. By looking at the pairwise cosine matrix, we sort the matrix in increasing order and chose the first 3 movies. These movies have the smallest distance with the movie number 66. I still keep movie 66 itself as a comparison, of course, the similarity should be 1. Hence n here is 4, which means movie 66 itself and 3 top similar movies.

d1 needs to be transformed into a matrix first

```
d1[66,]
```

```
##         1         2         3         4         5         6         7
## 1.0000000 1.0000000 1.0000000 0.8249847 1.0000000 1.0000000 0.9835806
##         8         9        10        11        12        13        14
## 0.9793852 1.0000000 0.9138254 1.0000000 1.0000000 1.0000000 1.0000000
##        15        16        17        18        19        20        21
## 1.0000000 0.9823338 0.9799133 0.9900248 1.0000000 1.0000000 1.0000000
##        22        23        24        25        26        27        28
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        29        30        31        32        33        34        35
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        36        37        38        39        40        41        42
## 0.9818817 0.9804174 1.0000000 1.0000000 0.9769083 1.0000000 1.0000000
##        43        44        45        46        47        48        49
## 0.9885003 1.0000000 1.0000000 0.9707452 0.9835652 0.9912537 0.9676539
##        50        51        52        53        54        55        56
## 0.9790179 1.0000000 0.9665932 0.9725547 0.9897971 1.0000000 1.0000000
##        57        58        59        60        61        62        63
## 1.0000000 1.0000000 0.9284108 0.9928251 1.0000000 1.0000000 1.0000000
##        64        65        66        67        68        69        70
## 0.9898512 1.0000000 0.0000000 1.0000000 1.0000000 1.0000000 0.9937040
```

The top3 movies' overview that similar to movie 66 is movie id 4, 10, 59

```
(top3<-head(order(d1[66,],decreasing = FALSE),n=4))
```

```
## [1] 66  4 10 59
```

We can look into these movies' overviews, they do talk about the same thing, Batman.

```
##
overview
## 66                                  Batman raises the stakes in his war on crime. With the help of Lt. Jim Gord
on and District Attorney Harvey Dent, Batman sets out to dismantle the remaining criminal organizations that plag
ue the streets. The partnership proves to be effective, but they soon find themselves prey to a reign of chaos un
leashed by a rising criminal mastermind known to the terrified citizens of Gotham as the Joker.
## 4  Following the death of District Attorney Harvey Dent, Batman assumes responsibility for Dent's crimes to pr
otect the late attorney's reputation and is subsequently hunted by the Gotham City Police Department. Eight years
later, Batman encounters the mysterious Selina Kyle and the villainous Bane, a new terrorist leader who overwhelm
s Gotham's finest. The Dark Knight resurfaces to protect a city that has branded him an enemy.
## 10                                                    Fearing the actions of a god-like Super Hero left unch
ecked, Gotham City's own formidable, forceful vigilante takes on Metropolis's most revered, modern-day savior, wh
ile the world wrestles with what sort of hero it really needs. And with Batman and Superman at war with one anoth
er, a new threat quickly arises, putting mankind in greater danger than it's ever known before.
## 59                                       After an attempted assassination on Ambassador Han, Inspector Lee an
d Detective Carter are back in action as they head to Paris to protect a French woman with knowledge of the Triad
s' secret leaders. Lee also holds secret meetings with a United Nations authority, but his personal struggles wit
h a Chinese criminal mastermind named Kenji, which reveals that it's Lee's long-lost...brother.
##        id                              title
## 66    155                    The Dark Knight
## 4   49026               The Dark Knight Rises
## 10 209112 Batman v Superman: Dawn of Justice
## 59   5174                       Rush Hour 3
```

We can also build a searching engine by using tf-idf. For example, who is James bond? We can improve this engine later, which are not covered in this code, here are some suggestions: 1. delete some common words like movie, like, favorite, etc. 2. replace some words by their synonym to get a better searching result.

After we put the query in the last row, the dimension of the matrix is 71 by 3.

```
(query <- data.frame("who is James bond","",""))
```

```
##   X.who.is.James.bond. X.. X...1
## 1    who is James bond
```

```
names(query)<-c("overview","id","title")
newquery <- rbind(overviewdf_70,query)

dim(newquery)
```

```
## [1] 71  3
```

Every code is the same as we did previously, I skip these steps. After the all the tf-idf and cosine similarity, now we have a similarity matrix of movie 71 to all other movies. Remember, actually movie 71 is our query, it is actually not a real movie overview, we just treat it as an overview to find similar movie overviews, and then do the recommendation.

```
d2[71,]
```

```
##         1         2         3         4         5         6         7
## 1.0000000 1.0000000 0.8950871 1.0000000 1.0000000 1.0000000 1.0000000
##         8         9        10        11        12        13        14
## 1.0000000 1.0000000 1.0000000 1.0000000 0.7129685 1.0000000 1.0000000
##        15        16        17        18        19        20        21
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        22        23        24        25        26        27        28
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        29        30        31        32        33        34        35
## 1.0000000 0.9287077 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        36        37        38        39        40        41        42
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        43        44        45        46        47        48        49
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        50        51        52        53        54        55        56
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        57        58        59        60        61        62        63
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        64        65        66        67        68        69        70
## 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##        71
## 0.0000000
```

The top 3 movie overviews that similar to “who is James Bond” is movie 12, 3 and 30.

```
(top3<-head(order(d2[71,],decreasing = FALSE),n=4))
```

```
## [1] 71 12  3 30
```

We look into these movie’s overviews. The first one is movie 71, which is our query “who is James Bond”, the key words is James Bond, we look the overviews of movie 12, 3, and movie 30. They do all mention James Bond.

```
newquery[c(top3),]
```

```
##
overview
## 71
who is James bond
## 12
Quantum of Solace continues the adventures of James Bond after Casino Royale. Betrayed by Vesper, the woman he lo
ved, 007 fights the urge to make his latest mission personal. Pursuing his determination to uncover the truth, Bo
nd and M interrogate Mr. White, who reveals that the organization that blackmailed Vesper is far more complex and
dangerous than anyone had imagined.
## 3
A cryptic message from Bond’s past sends him on a trail to uncover a sinister organization. While M battles polit
ical forces to keep the secret service alive, Bond peels back the layers of deceit to reveal the terrible truth b
ehind SPECTRE.
## 30 When Bond's latest assignment goes gravely wrong and agents around the world are exposed, MI6 is attacked f
orcing M to relocate the agency. These events cause her authority and position to be challenged by Gareth Mallor
y, the new Chairman of the Intelligence and Security Committee. With MI6 now compromised from both inside and ou
t, M is left with one ally she can trust: Bond. 007 takes to the shadows - aided only by field agent, Eve - follo
wing a trail to the mysterious Silva, whose lethal and hidden motives have yet to reveal themselves.
##        id             title
## 71
## 12  10764 Quantum of Solace
## 3  206647           Spectre
## 30  37724           Skyfall
```

Convert the cast column which is in Json format into the column, we use filter to find movies Leonardo DiCaprio takes part in.

```
kable(cast %>% filter(name=="Leonardo DiCaprio"))
```

movie_idtitle	cast_idcharacter	gender	idname	order
597Titanic	21Jack Dawson	26193	Leonardo DiCaprio	1
64682The Great Gatsby	2Jay Gatsby	26193	Leonardo DiCaprio	0
27205Inception	1Dom Cobb	26193	Leonardo DiCaprio	0
281957The Revenant	1Hugh Glass	26193	Leonardo DiCaprio	0
2567The Aviator	1Howard Hughes	26193	Leonardo DiCaprio	0
68718Django Unchained	3Calvin Candie	26193	Leonardo DiCaprio	2
1372Blood Diamond	27Danny Archer	26193	Leonardo DiCaprio	0
106646The Wolf of Wall Street	8Jordan Belfort	26193	Leonardo DiCaprio	0
3131Gangs of New York	2Amsterdam Vallon	26193	Leonardo DiCaprio	0
1422The Departed	2William “Billy” Costigan, Jr.	26193	Leonardo DiCaprio	0
11324Shutter Island	2Teddy Daniels	26193	Leonardo DiCaprio	0
12113Body of Lies	4Roger Ferris	26193	Leonardo DiCaprio	0
640Catch Me If You Can	23Frank Abagnale Jr.	26193	Leonardo DiCaprio	0
1907The Beach	2Richard	26193	Leonardo DiCaprio	0
4148Revolutionary Road	2Frank Wheeler	26193	Leonardo DiCaprio	0
9313The Man in the Iron Mask	1King Louis XIV / Philippe	26193	Leonardo DiCaprio	0
88794J. Edgar	1J. Edgar Hoover	26193	Leonardo DiCaprio	0
12106The Quick and the Dead	14Fee Herod "The Kid“	26193	Leonardo DiCaprio	3
9819Marvin’s Room	3Hank	26193	Leonardo DiCaprio	2
454Romeo + Juliet	1Romeo	26193	Leonardo DiCaprio	0
9466Celebrity	10Brandon	26193	Leonardo DiCaprio	3
1587What’s Eating Gilbert Grape	2Arnie Grape	26193	Leonardo DiCaprio	1

We also use the same code to convert crew column and use filter function to find movies James Cameron directed.

movie_idtitle	department	gender	idjob	name
19995Avatar	Directing	22710	Director	James Cameron
19995Avatar	Writing	22710	Writer	James Cameron
19995Avatar	Editing	22710	Editor	James Cameron
19995Avatar	Production	22710	Producer	James Cameron
19995Avatar	Writing	22710	Screenplay	James Cameron
597Titanic	Writing	22710	Screenplay	James Cameron
597Titanic	Directing	22710	Director	James Cameron
597Titanic	Editing	22710	Editor	James Cameron
597Titanic	Production	22710	Producer	James Cameron
296Terminator 3: Rise of the Machines	Writing	22710	Characters	James Cameron
87101Terminator Genisys	Writing	22710	Characters	James Cameron
280Terminator 2: Judgment Day	Directing	22710	Director	James Cameron
280Terminator 2: Judgment Day	Production	22710	Producer	James Cameron
280Terminator 2: Judgment Day	Writing	22710	Writer	James Cameron
36955True Lies	Writing	22710	Screenplay	James Cameron
36955True Lies	Directing	22710	Director	James Cameron
36955True Lies	Production	22710	Producer	James Cameron
2756The Abyss	Writing	22710	Screenplay	James Cameron
2756The Abyss	Directing	22710	Director	James Cameron
1369Rambo: First Blood Part II	Writing	22710	Screenplay	James Cameron
94352Cirque du Soleil: Worlds Away	Production	22710	Producer	James Cameron
679Aliens	Writing	22710	Screenplay	James Cameron
679Aliens	Directing	22710	Director	James Cameron
218The Terminator	Directing	22710	Director	James Cameron
218The Terminator	Writing	22710	Writer	James Cameron

What we want to do is find similar movies based on the actors, director, keywords, and genres.

We composed a function that recommends 5 similar movies for a movie. What we am doing is to sort all other movies based on:

1. The number of similarities (same director, same actors, same genres).
2. When the number of similarities is the same for a number of movies, I am sorting them on descending weighted rating.

In the table below, you will see the recommendations for the Titanic. Of course, the similarities for Titanic itself are perfect, but we have left the movie itself in the table as a reference. The recommendations are the other 5 movies.

We build a function to do the sorting. we create a variable called similarity_vars, it contains actor_1, actor_2, actor_3, director, genre_1, genre_2, genre_3 that we defined previously.

Then we do a function to build a recommender system. When we type movie to the function, we can get movies_filter variable that contains actor_1, actor_2, actor_3, director, genre_1, genre_2, genre_3 that accords the movie we type in the function.

Then we build an empty table called rec_df, we assign NA to all columns: same_director, same_a1, same_a2, same_a3, same_g1, same_g2, same_g3

If content of rec_df equals to actor 1 or actor 2, actor 3, then we assign it has a weight 1; if any of these columns are not equal to rec_df, then we assign the weight 0. We defined a column named sim_count which is the sum of all weight we just assigned, it is the sum of columns from 10 to 16, same_director, same_a1, same_a2, same_a3, same_g1, same_g2, same_g3

Then we define a Top5_rec which is ordered by the decreasing sim_count

```
similarity_vars <- c("actor_1", "actor_2", "actor_3", "director", "genre_1", "genre_2", "genre_3")

movies_filter <- movies %>% select(id, title, !!similarity_vars)
movies_filter <- movies_filter %>% mutate_if(is.factor, as.character)

recommend_similar <- function(movie){
        director <- movies_filter$director[movies_filter$id==movie]
actor1 <- movies_filter$actor_1[movies_filter$id==movie]
actor2 <- movies_filter$actor_2[movies_filter$id==movie]
actor3 <- movies_filter$actor_3[movies_filter$id==movie]
genre1 <- movies_filter$genre_1[movies_filter$id==movie]
genre2 <- movies_filter$genre_2[movies_filter$id==movie]
genre3 <- movies_filter$genre_3[movies_filter$id==movie]

rec_df <- movies_filter
```

```
rec_df$same_director <- NA
rec_df$same_a1 <- NA
rec_df$same_a2 <- NA
rec_df$same_a3 <- NA
rec_df$same_g1 <- NA
rec_df$same_g2 <- NA
rec_df$same_g3 <- NA

rec_df$same_director <- ifelse(rec_df$director==director, 1, 0)
rec_df$same_a1 <- ifelse(rec_df$actor_1==actor1|rec_df$actor_2==actor1|rec_df$actor_3==actor1, 1, 0)
rec_df$same_a2 <- ifelse(rec_df$actor_1==actor2|rec_df$actor_2==actor2|rec_df$actor_3==actor2, 1, 0)
rec_df$same_a3 <- ifelse(rec_df$actor_1==actor3|rec_df$actor_2==actor3|rec_df$actor_3==actor3, 1, 0)
rec_df$same_g1 <- ifelse(rec_df$genre_1==genre1|rec_df$genre_2==genre1|rec_df$genre_3==genre1, 1, 0)
rec_df$same_g2 <- ifelse(rec_df$genre_1==genre2|rec_df$genre_2==genre2|rec_df$genre_3==genre2, 1, 0)
rec_df$same_g3 <- ifelse(rec_df$genre_1==genre3|rec_df$genre_2==genre3|rec_df$genre_3==genre3, 1, 0)

rec_df <- rec_df %>% mutate_at(vars(10: 16), funs(replace(., is.na(.), 0)))

rec_df$sim_count <- rowSums(rec_df[,10:16])

rec_df <- left_join(rec_df, movies %>% select(id, weighted_rating), by="id")

Top5_rec <- rec_df %>% arrange(desc(sim_count), desc(weighted_rating)) %>% slice(1:6) %>% select(id, title, sim_c
ount, weighted_rating, everything())

kable(Top5_rec) %>%
        kable_styling(full_width=TRUE)
}

recommend_similar(597)
```

Below is the output of this recommender engine, the sim_count of Titanic to Titanic itself is 7 since every actor, director and genre are the same. 3 actors plus 3 genres plus 1 director equals 7. The movie that similar to Titanic is Revolutionary Road, which Sim_Count is 4, then following The Departed, Shutter Island, Eternal Sunshine of the Spotless Mind and The Great Gatsby. The second column of the table is weighted_rating, which is the IMDB's score we just calculated previously.

id	title	sim_count	weighted_rating	actor_1	actor_2	actor_3	director	genre_1	genre_2	genre_3	same_director
597	Titanic	7	7.374977	Kate Winslet	Leonardo DiCaprio	Frances Fisher	James Cameron	Drama	Romance	Thriller	1
4148	Revolutionary Road	4	6.407782	Leonardo DiCaprio	Kate Winslet	Michael Shannon	Sam Mendes	Drama	Romance	NA	0
1422	The Departed	3	7.637516	Leonardo DiCaprio	Matt Damon	Jack Nicholson	Martin Scorsese	Drama	Thriller	Crime	0
11324	Shutter Island	3	7.622046	Leonardo DiCaprio	Mark Ruffalo	Ben Kingsley	Martin Scorsese	Drama	Thriller	Mystery	0
38	Eternal Sunshine of the Spotless Mind	3	7.596430	Jim Carrey	Kate Winslet	Kirsten Dunst	Michel Gondry	Science Fiction	Drama	Romance	0
64682	The Great Gatsby	3	7.102448	Leonardo DiCaprio	Tobey Maguire	Carey Mulligan	Baz Luhrmann	Drama	Romance	NA	0