Data Science: Capstone

Robert E Lee Lewis

June 9, 2019

MovieLens Project Introduction

The purpose of this project (final test) is to create an algorithm for predicting movie ratings and calculating the RMSE using the provided data. The dataset comes from Grouplen dataset, Movielens and consists of two files, movie.dat containing 32043 different movie titles and ratings.dat with approximately 10 million user movie ratings.

My first thought about predicting movie rating, is to utilize a collaborative filtering methods i.e: User based and Item based collaborative filtering and POPLAR. I'll be cleaning up the data by removing items that may skew my results and increase performance, create a few different varioution of the models and compare them to find what I believe will return the best results. Finally using a recommender system to predicting the movies for user that will have the predicted ratings with RMSE <= .087750.

This project began with a few challenges, first is to overcome beginning with acually loading the large set of data for analysis and then memory issues. I had was extracting the ratings data from the ml-10m100K/ratings.dat file which I could not complete on my laptop (Alienware I7 6700hq cpu @2.60Ghz and 16GB with Windows 10 Pro for Workstations, of which I still have no idea why it will not import). After days of attempting then purchasing a faster desktop computer then I was able to sucessfully import. During my initial struggles I also found that using fread function compared to read.table function to be faster considerable faster for reading in the data therefore I altered the initial download process from what was given.

Two tables of data called movies and ratings are provided. The datasets will be joined by movieIds and userIDs to make our MovieLens Dataset. The MovieLens data will then be split with 10 percent as Validation dataset and the remainder as EDX dataset. Make sure userId and movieId in validation set are also in edx set then add the Validation set back into EDX.

Initial Analysis of EDX Data:

```
##      userId         movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08  
##  Median :35743   Median : 1834   Median :4.000   Median :1.035e+09  
##  Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##     title              genres         
##  Length:9000061     Length:9000061    
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
##                                       
##                                       
##                                       
```

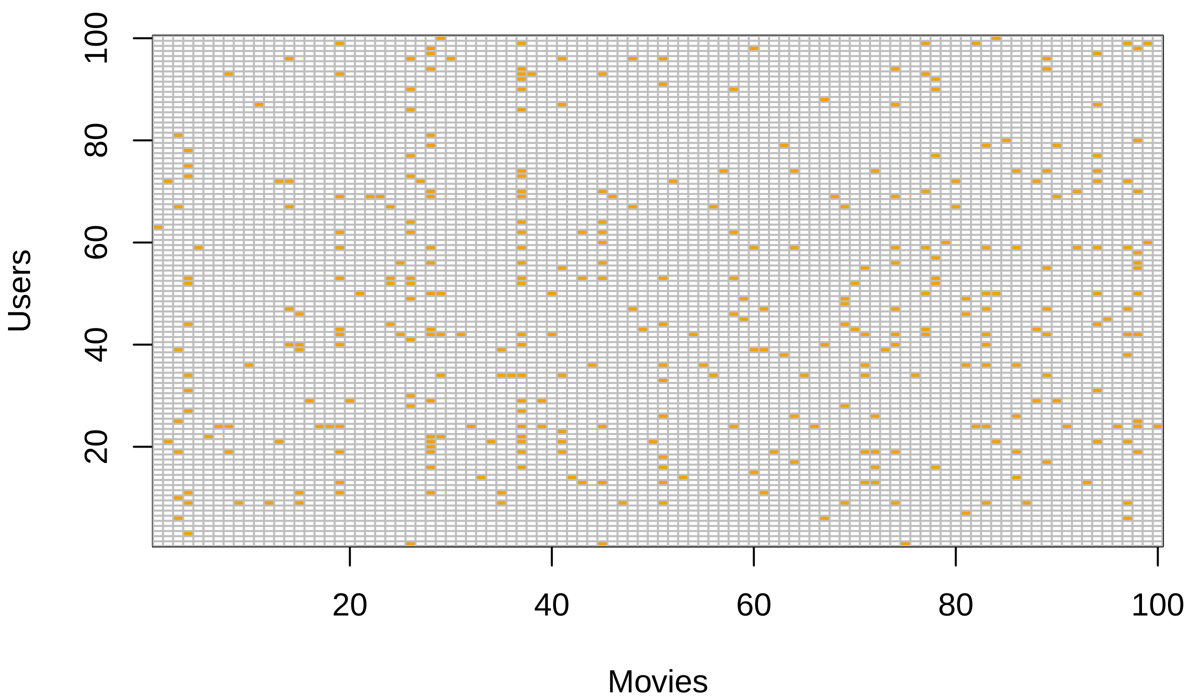
From the summary of edx dataset we know there are 9000061 row with 6 variables UserId, MovieId, rating, timestamps, title and multiple combinations of genres. It also appears there is no missing data.

```
## Observations: 9,000,061
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, 37...
```

```
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumbe...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy",...
```

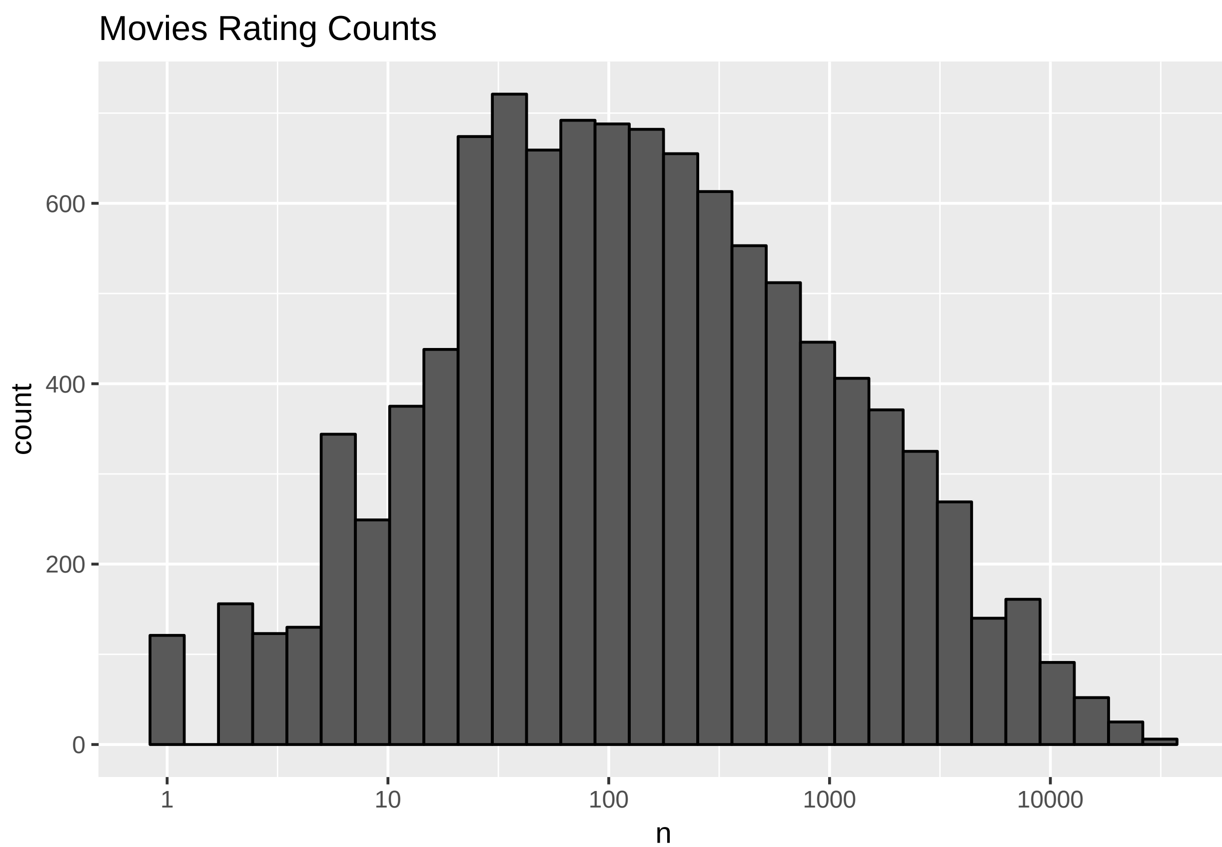
A glimpse of the data I notice timestamp needs to be converted if I am to do any timeseries related predictions which my current plan does not.

Image of 100 users and 100 movies.



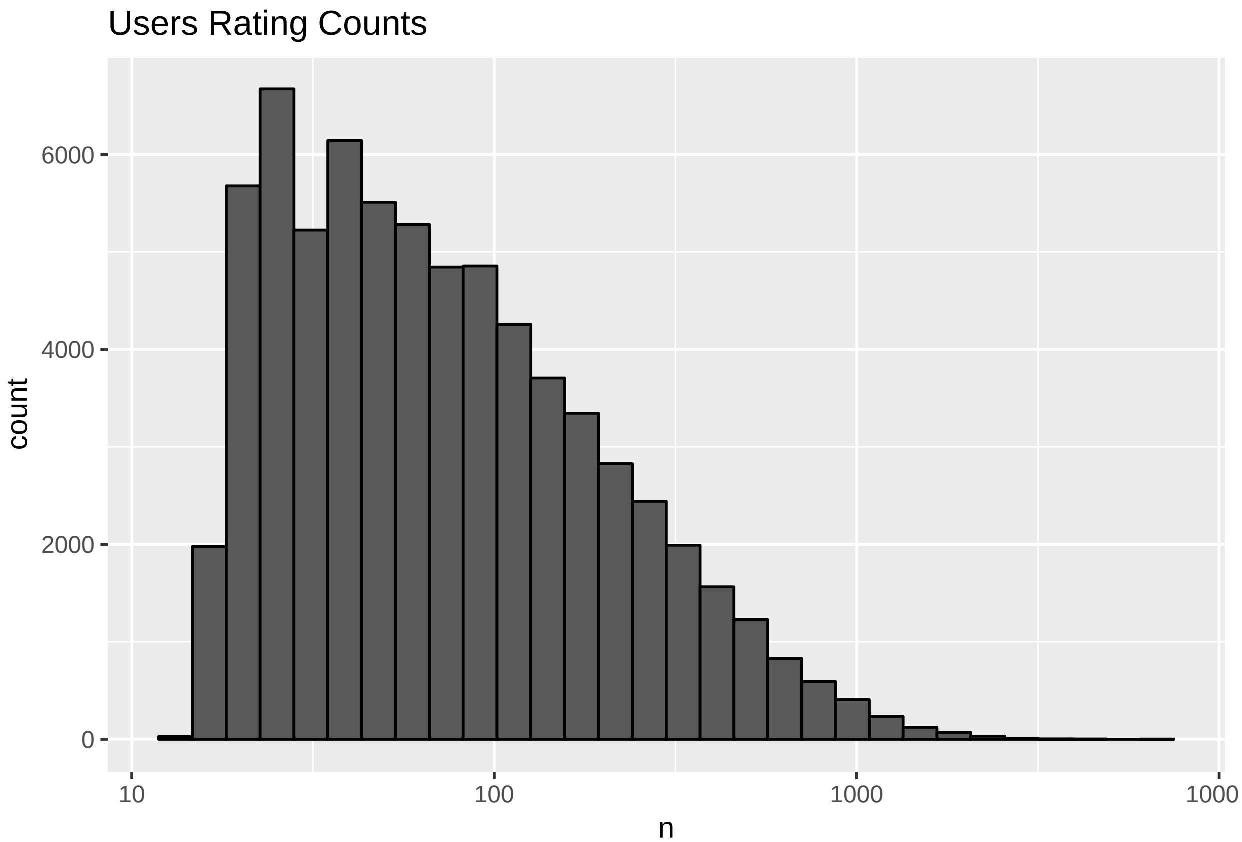
In the image above, note each row represents a user and the columns represent movies they rated. Notice that not every user has rated a movie (row 2) in the sample set to some rating numerous movies. Since I plan to use content based filtering for my predictions, I need my good dataset with many users that have rated at many movies.

Plot edx data for the count of movies rated:



Looking at the Movies Rating Count plot, shows that the majority of movies have been rated over 200 times.

Plot the count of ratings given by users:



Looking at the Users Rating Counts, shows that most users have rated at least 20 movies

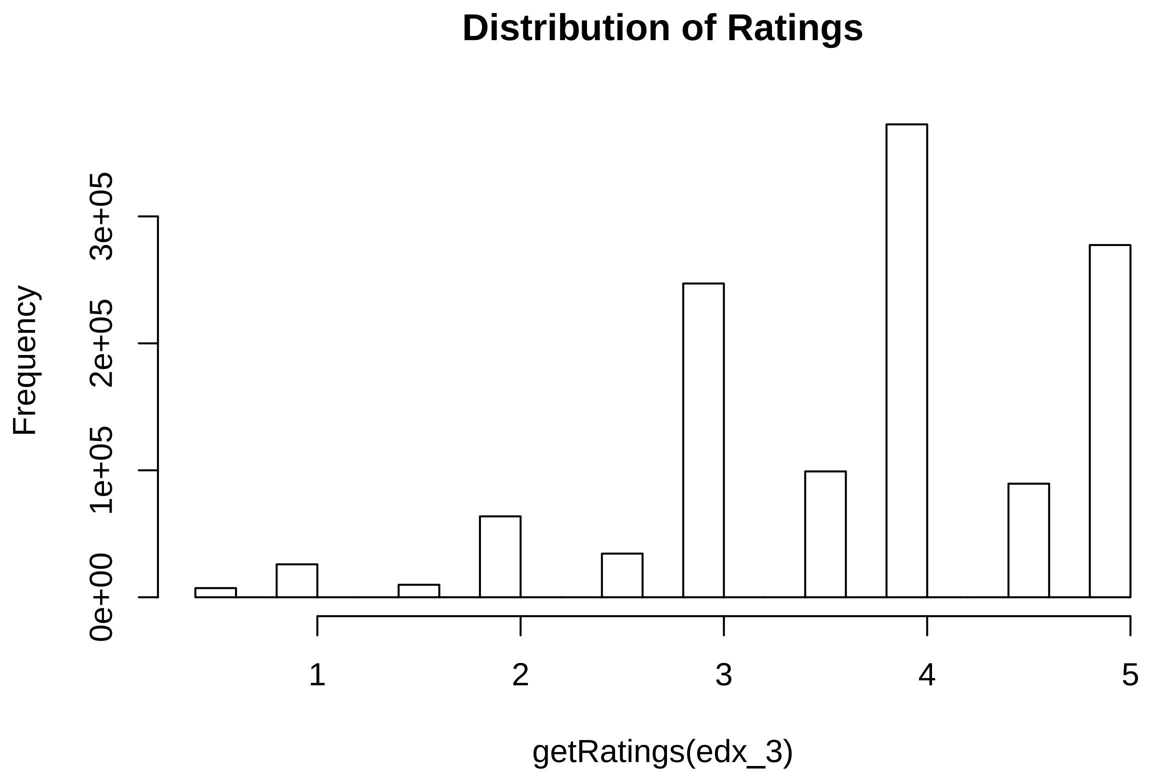
Preparing Data:

Based on the discovered information, I identify users who have rated 20 movies or more, leaving **32298** users.

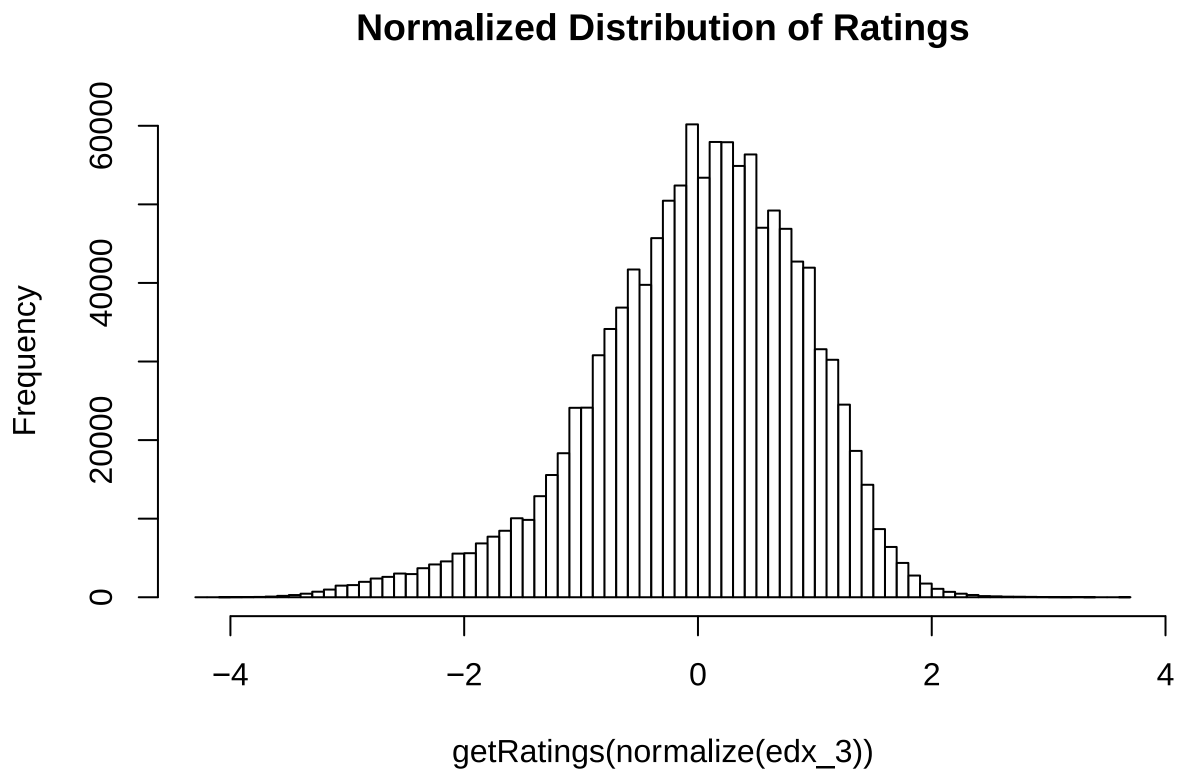
Due to performance issues I use only the top 100 rated movies, giving a dataset with **100** movies and **32298** with **1226733, 6** ratings and columns.

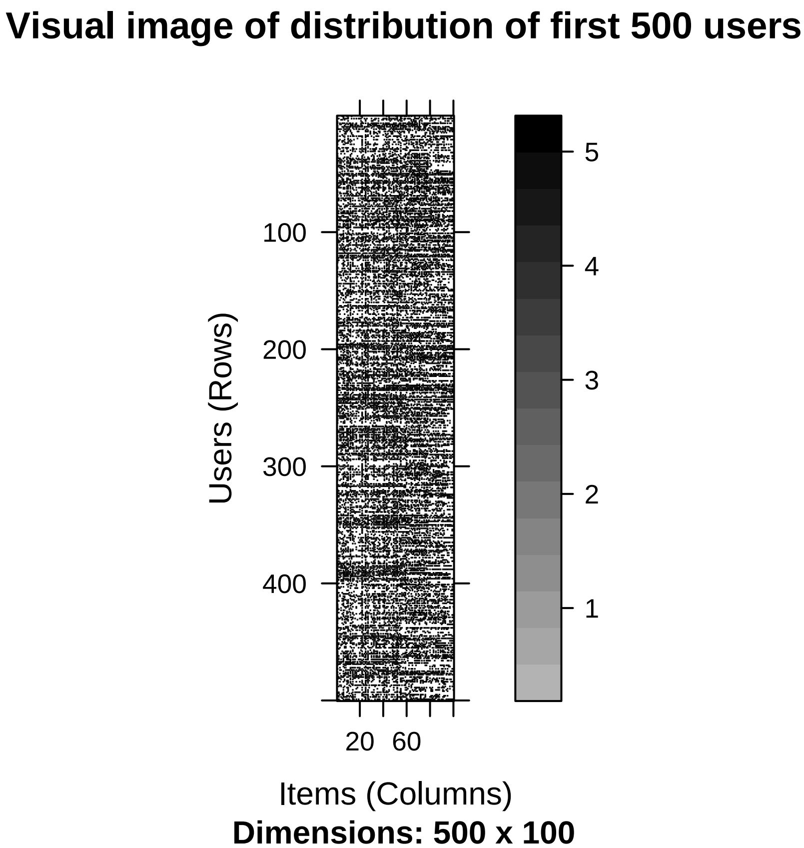
Do to the size of the datasets and reading several articules, I have choosen to use sparse matrix because "Sparse matracies also have significant advandatages in terms of computational efficiency. Unlike opertations with fill matrices, operatios with sparce matricies do not perform unnessessary low-level arithmetics" Priyam (2016). The decision to use sparse matrices has lead my to use recommenderLab, Hahsler (2019) which will involve converting my edx data.frame data to matrix to finally realRatingMatrix class.

Review the prepared dataset



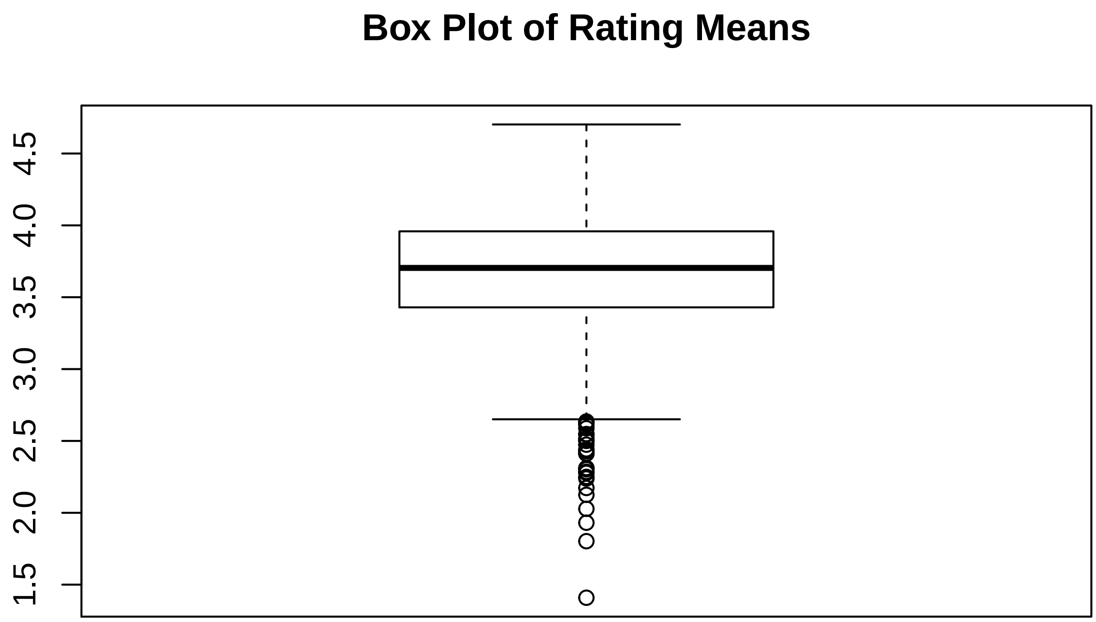
The Distribution of Ratings shows we have 10 different possible ratings from 0.5 to 5 in incriments of 0.5.


Normalized Distribution of Ratings
Frequency
0
20000
40000
60000
−4
−2
0
2
4
getRatings(normalize(edx_3))



Due to performance/memory issues, I remove users with less than 70 movie ratings for my model dataset

```
## 2211 x 100 rating matrix of class 'realRatingMatrix' with 173208 ratings.
```



I can further cleeanup data by removing outlirers, When looking at the previous boxplot of the dataset we have a few outliers with row means below 2.7 and above 4.6 so I will remove them.

Outiers with rowMeans below 2.7 to remove

```
## [1]  39 100
```

Outiers wit rowMeans above 4.6 to remove

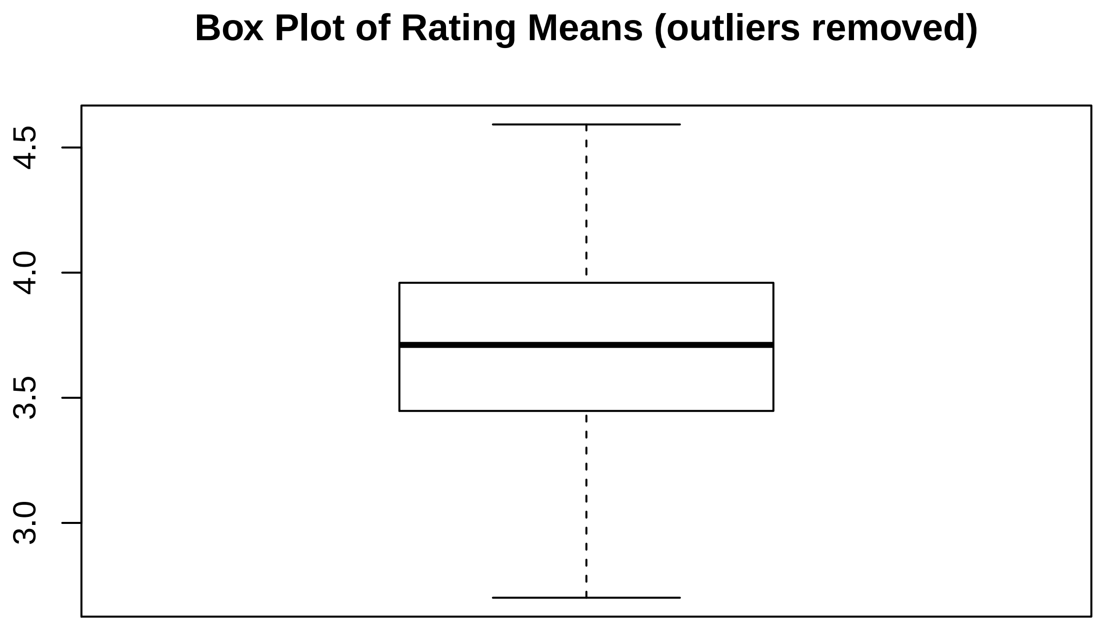
```
## [1]   7 100
```

Rows remaining after removing outliers

```
## [1] 2165  100
```

After Outliers removed. . .

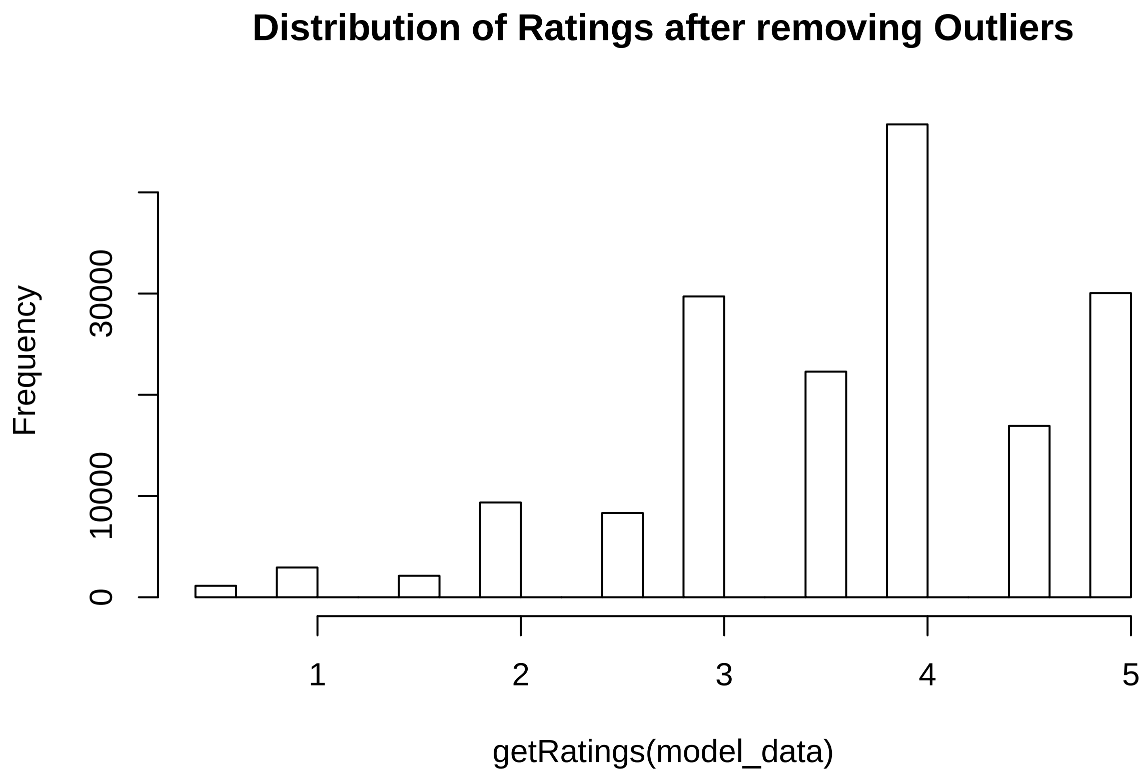
Boxplot of Model Data after outliers removed, data is symetric.



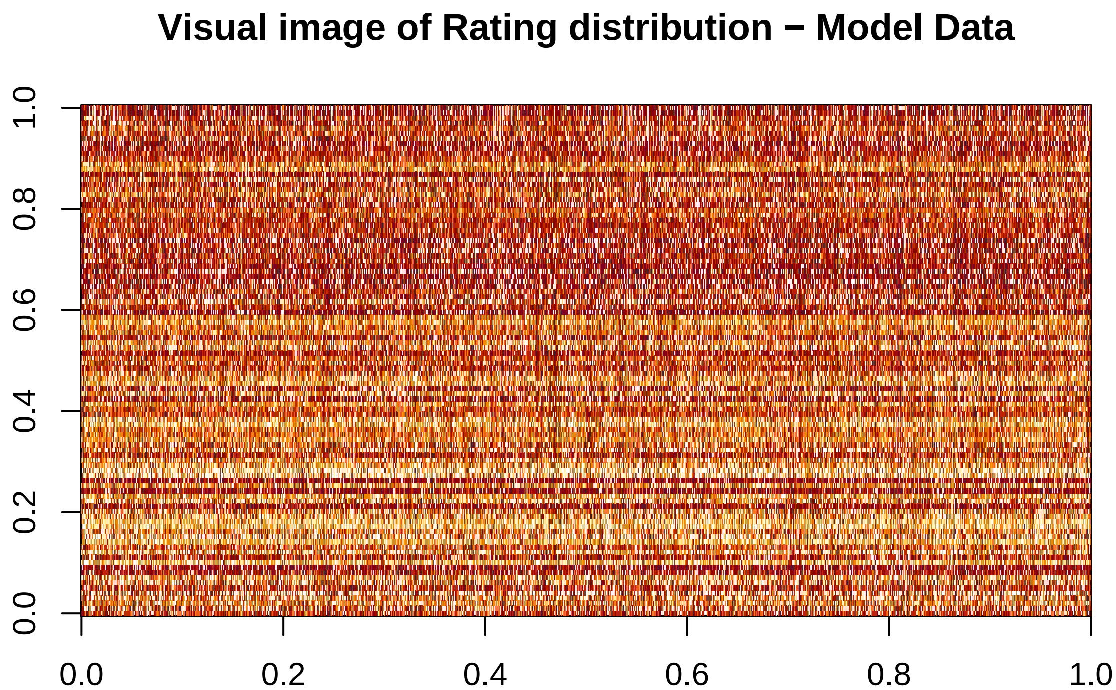
Number of Rating remaining in Model Data

```
## [1] 169579
```

Model dataset Distribution of Ratings



Note rating of 4 is the most popular.



Note Prepared Model data appears to be well ditributed.

Analyze number of movie ratings per user:

```
##
##  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88
## 185 168 143 164 168 130 142 127 122 109 118  82  85  81  87  53  41  33
##  89  90  91  92  93  94  95  96  97
##  29  35  26  22   8   3   2   1   1
```

Note that only one user has rated 96,97 of the top 100 rated movies, no one has rated all movies in our model dataset.

Create Modeling datasets

Divide the prepared model data set into train and test sets, 80/20 respectively.

Train set diminsions: 1727, 100
Test set diminsions: 438, 100

Build Models Options

I will use various models then compare prediction accuracies to determine the best algorithim. The available models I will first use a user-based collaborative filtering algorithm (UBCF), then item-based collaborative filtering (IBCF) and item popularity algorithms (POPULAR).

User Based Collborative Filtering (UBCF) Model

Collaborative filtering uses algorithims to filter users ratings to make personalized recommendations from similiar users (definition from whatis.techtarget.com/definition/collaborative-filtering).

```
## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 1727 users.
```

```
## 1727 x 100 rating matrix of class 'realRatingMatrix' with 135263 ratings.
## Normalized using center on rows.
```

Using the UBCF recommendations

List of recommendation movies for test set users 7 thru 10:

```
## $User1408
##  [1] "Movie1136" "Movie527"  "Movie260"  "Movie541"  "Movie3996"
##  [6] "Movie1097" "Movie1036" "Movie1073" "Movie34"   "Movie141"
##
## $User1860
##  [1] "Movie1136" "Movie1193" "Movie595"  "Movie588"  "Movie457"
##  [6] "Movie339"  "Movie141"  "Movie10"   "Movie2683" "Movie597"
##
## $User2218
##  [1] "Movie260"  "Movie5952" "Movie50"   "Movie1240" "Movie1193"
##  [6] "Movie1221" "Movie527"  "Movie2028" "Movie1213" "Movie858"
##
## $User2596
##  [1] "Movie1221" "Movie296"  "Movie5952" "Movie4993" "Movie1704"
##  [6] "Movie1291" "Movie1097" "Movie150"  "Movie592"  "Movie34"
```

Total number of recommendations by users in test set

```
## number_of_items
##   7   8   9  10
##   1   5   5 427
```

Note that approx 427 users from the test set received 10 recommendations.

Create an evaluators scheme:

Create evaluation datasets using cross-validation method, keeping **30** items and **5** folds with rating threshold of **4** using the recommenderLab evaluationScheme function.

```
## Sizes of Evaluation Sets:     1732 1732 1732 1732 1732
```

```
## 1732 x 100 rating matrix of class 'realRatingMatrix' with 135628 ratings.
```

3 sets will be used: train = training set
known = test set used to build recommendations
unknown = test set to test the recommendations

Create UBCF Recommender

```
## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 1732 users.
```

Calculate the UBCF predictions for known test set

```
## 433 x 100 rating matrix of class 'realRatingMatrix' with 30310 ratings.
```

Calculate the prediction accuracy for each user in unknown test set :

Calculate the overall avgerages in unknown test set:

```
##      RMSE       MSE       MAE
## 0.7526803 0.6119072 0.5949279
```

Calculate the overall accuracy given in unknown test set:

```
##      RMSE       MSE       MAE
## 0.7803144 0.6088906 0.5930912
```

Note the overall RMSE and the accuracy are good.

Using a precicion recall plot to predict accuracy with confusion matrix for known test set

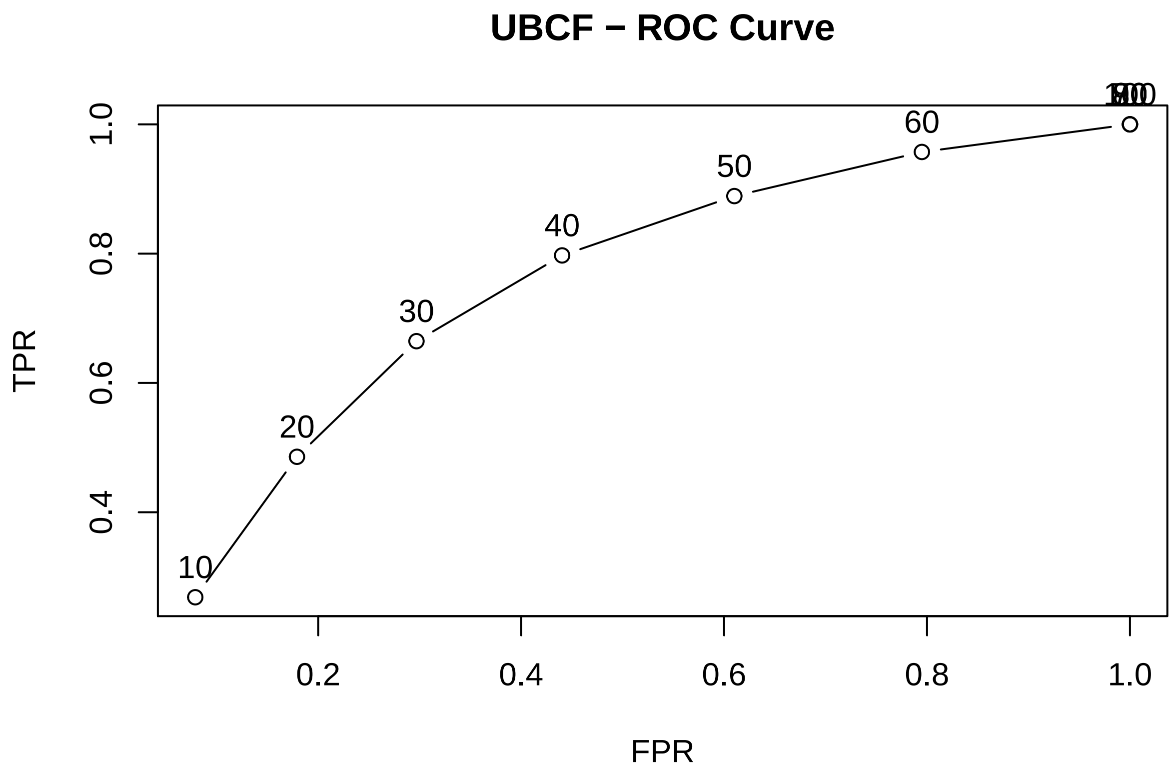
Evaluate the result with confusion matrix

```
##           TP        FP        FN        TN precision    recall       TPR
## 10  6.464203  3.535797 19.734411 40.265589 0.6464203 0.2740248 0.2740248
## 20 11.905312  8.094688 14.293303 35.706697 0.5952656 0.4883041 0.4883041
## 30 16.547344 13.452656  9.651270 30.348730 0.5515781 0.6649784 0.6649784
## 40 20.145497 19.854503  6.053118 23.946882 0.5036374 0.7935690 0.7935690
## 50 22.780600 27.219400  3.418014 16.581986 0.4556120 0.8857795 0.8857795
## 60 24.868360 35.131640  1.330254  8.669746 0.4144727 0.9552275 0.9552275
##           FPR
## 10 0.08017267
## 20 0.18159625
## 30 0.30095226
## 40 0.44483543
## 50 0.61345682
## 60 0.79560514
```

Sum up the UBCF TP, FP, FN, TN indexes and plot:

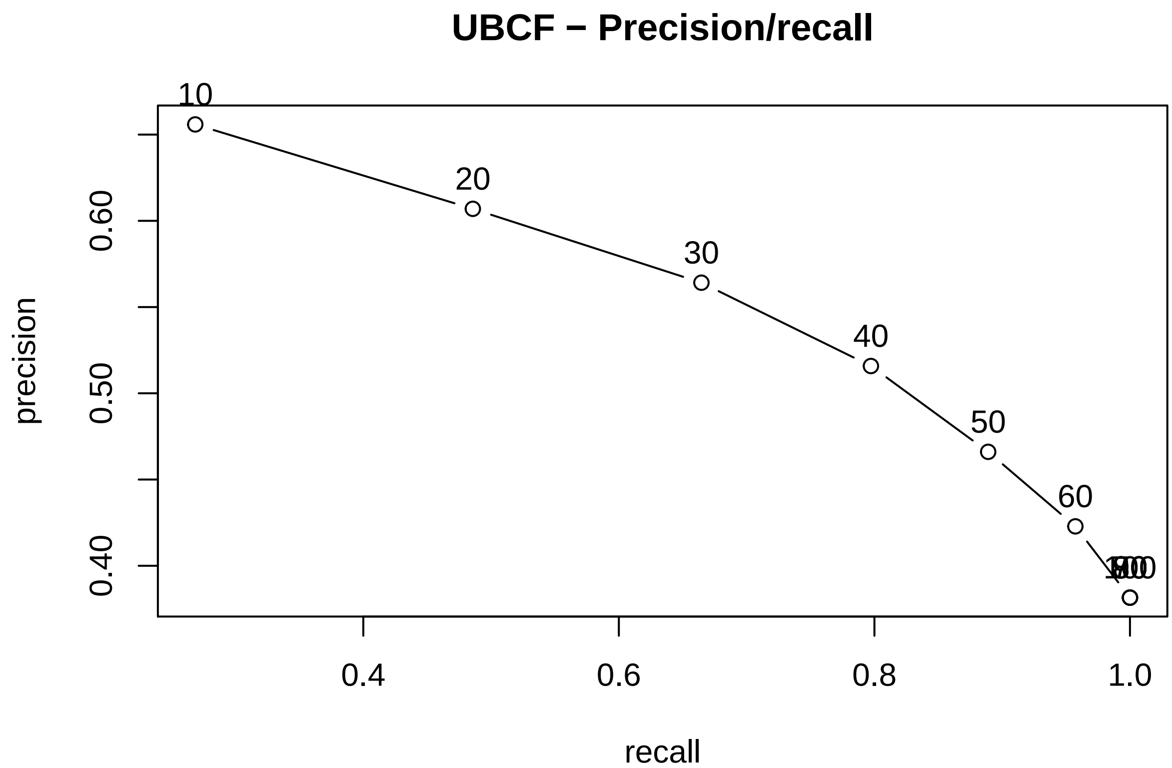
Note: it is difficult to visulize the data provided unless the results are plotted.

Create UBCF Receiver operating characteristic (ROC) plot



Note plot shows the relation ship between TPR and FPR
At 30 the TPR is close to 0.7 and the FPR is less than 0.4 is good
At 40 the TPR is close to 0.7 but the FPR is greater than 0.4 is not as good

Plot UBCF Precision/recall to verify accuracy



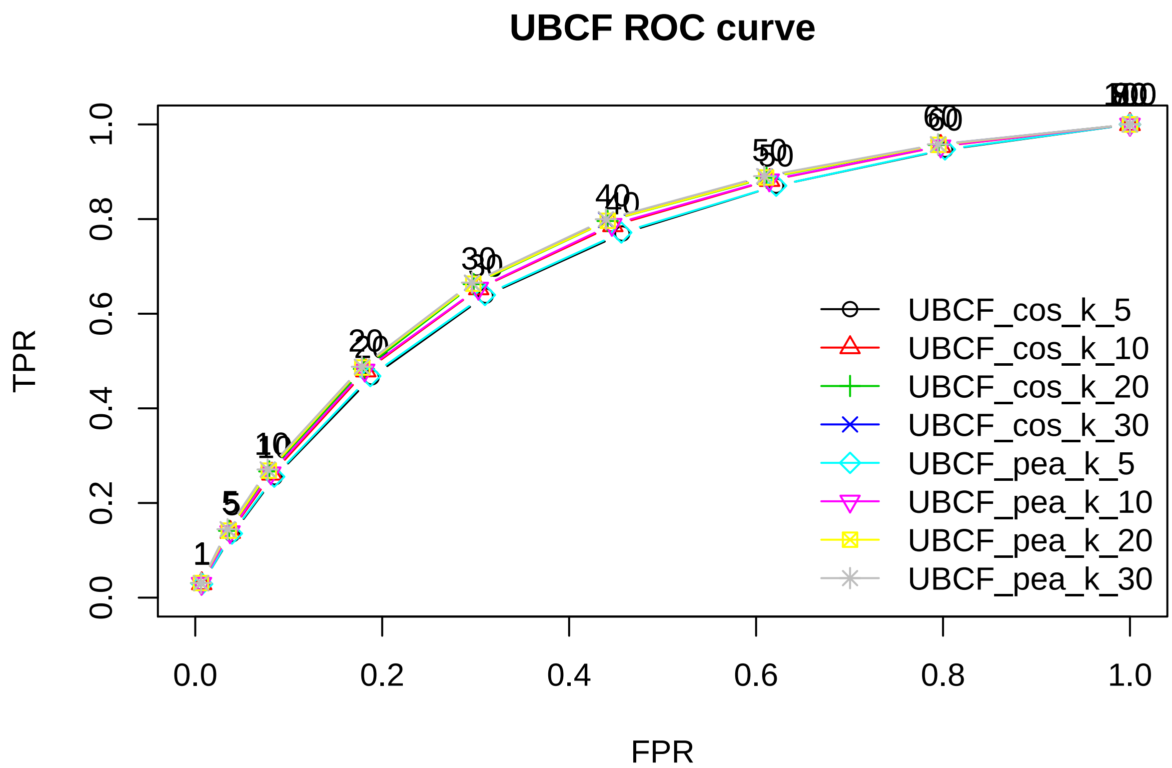
Note the precision/recall at #30 is not the best at 0.58/0.66

Fine Tuning of the Models to get best results

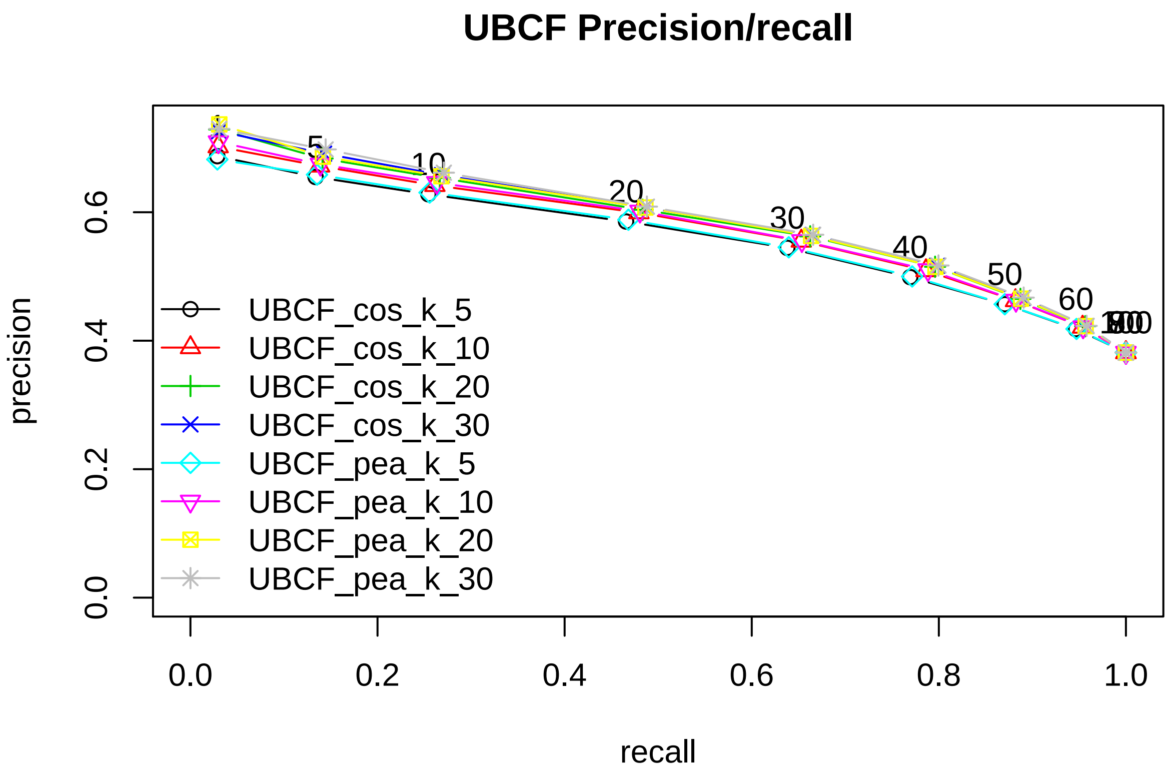
Lets try different factors to see if we can get a better Precision Recall result. Create UBCF Models with varing vector_nn and different methods i.e.: cosine and pearson.

Determine the best UBCF results based on number of recommendations

Plot UBCF Models with varing vector_nn and different methods results



Note: UBCF_pea_k_30 appears to be the best UBCF model with TPR closes to 0.7 and FPR less than 0.4



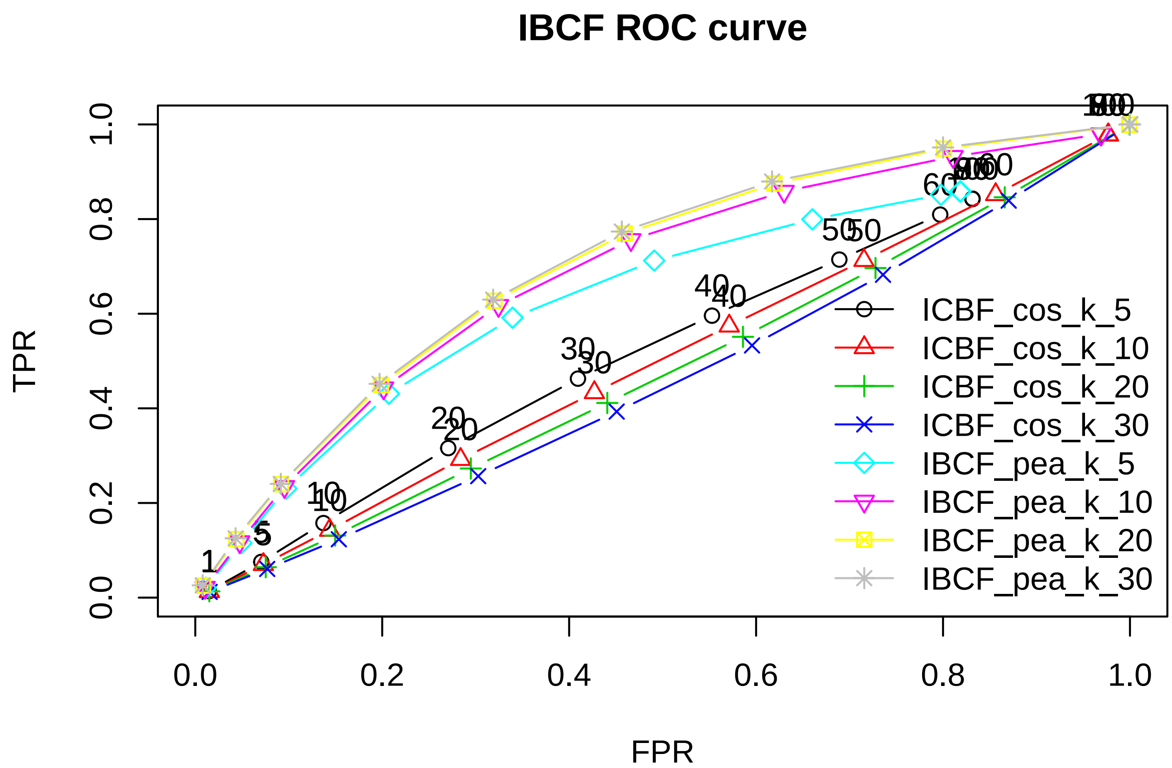
Note: The precision/recall support UBCF_pea_k_30 appears to be the best UBCF model with high persision

Create IBCF Model

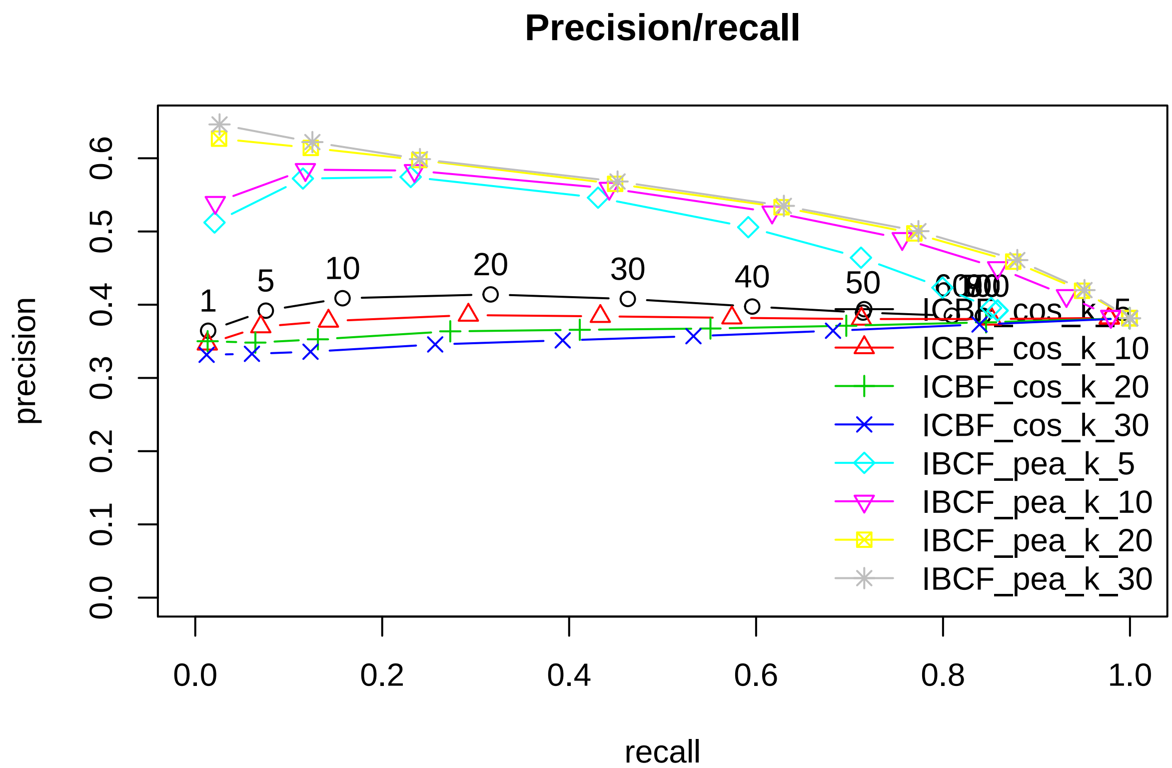
Create IBCF Models with varing vector_kn and different methods i.e.: cosine and pearson

Get IBCF model results

Plot IBCF with varing vector_kn and different methods results



Note ICBF_pea_k30 appears the best

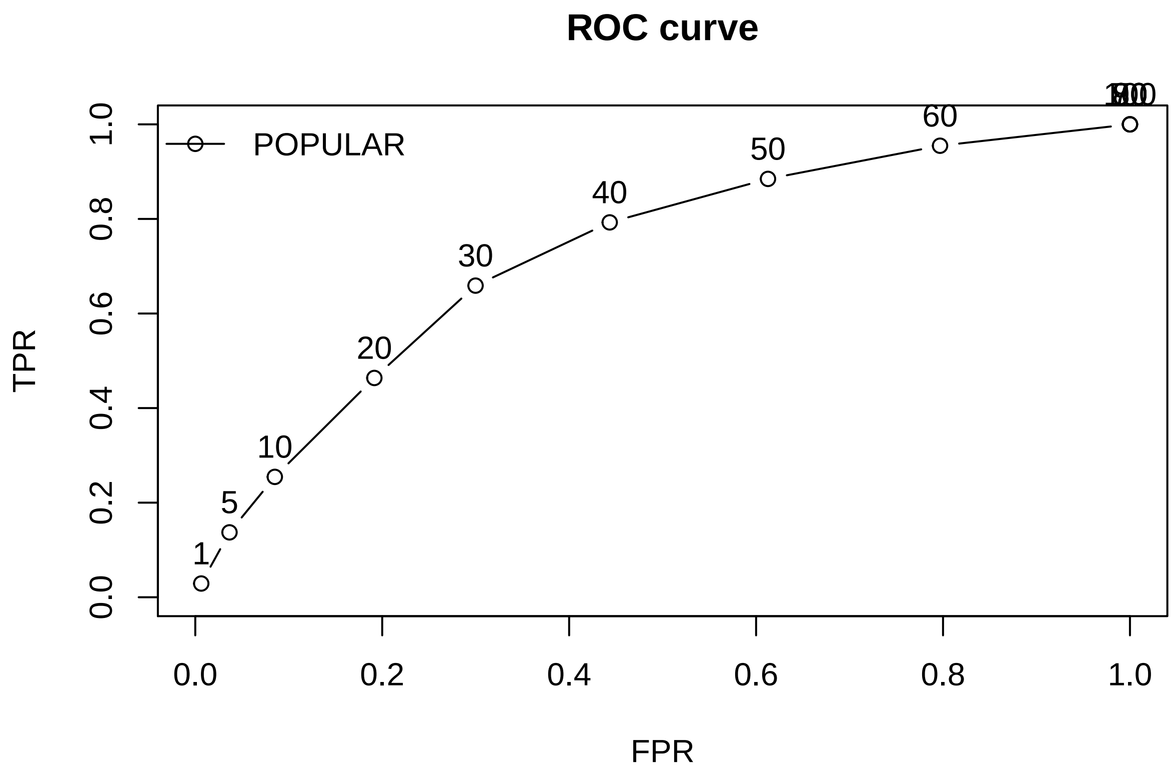


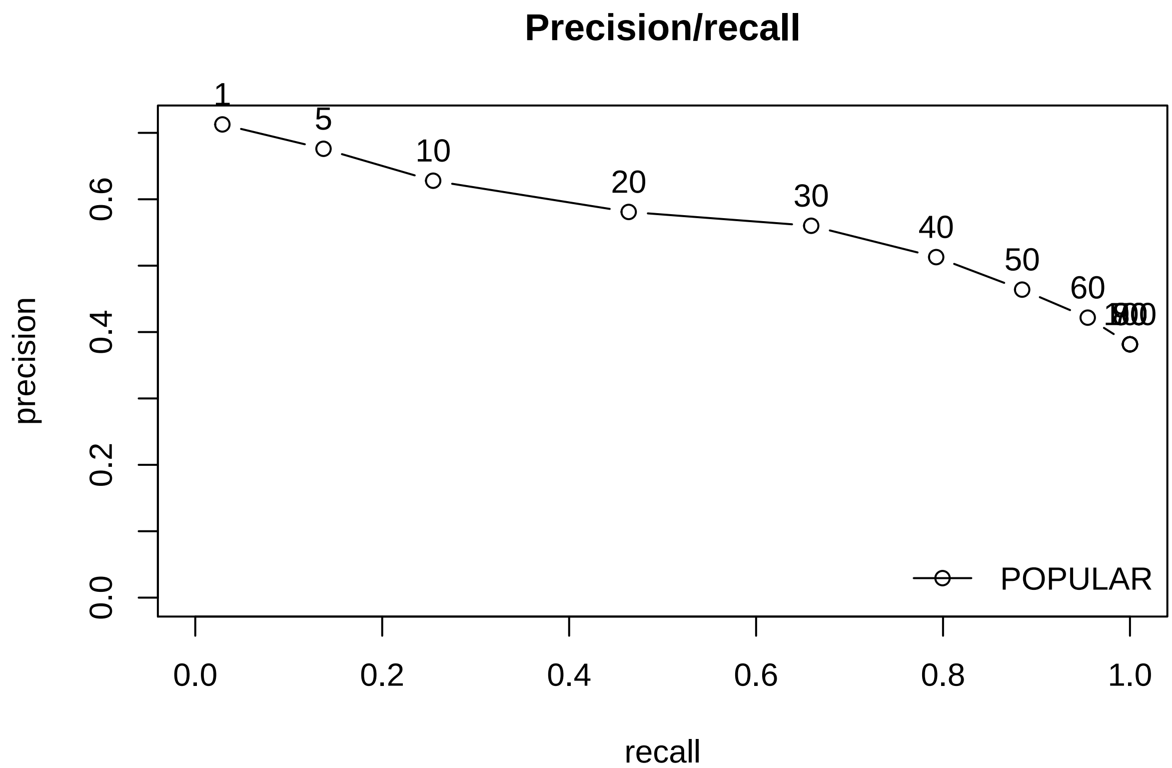
*Note IBCF Pearson with higher k values had better precision than the cosine algorithm.

Create a POPULAR model

The POPULAR model is simple based on items popularity.

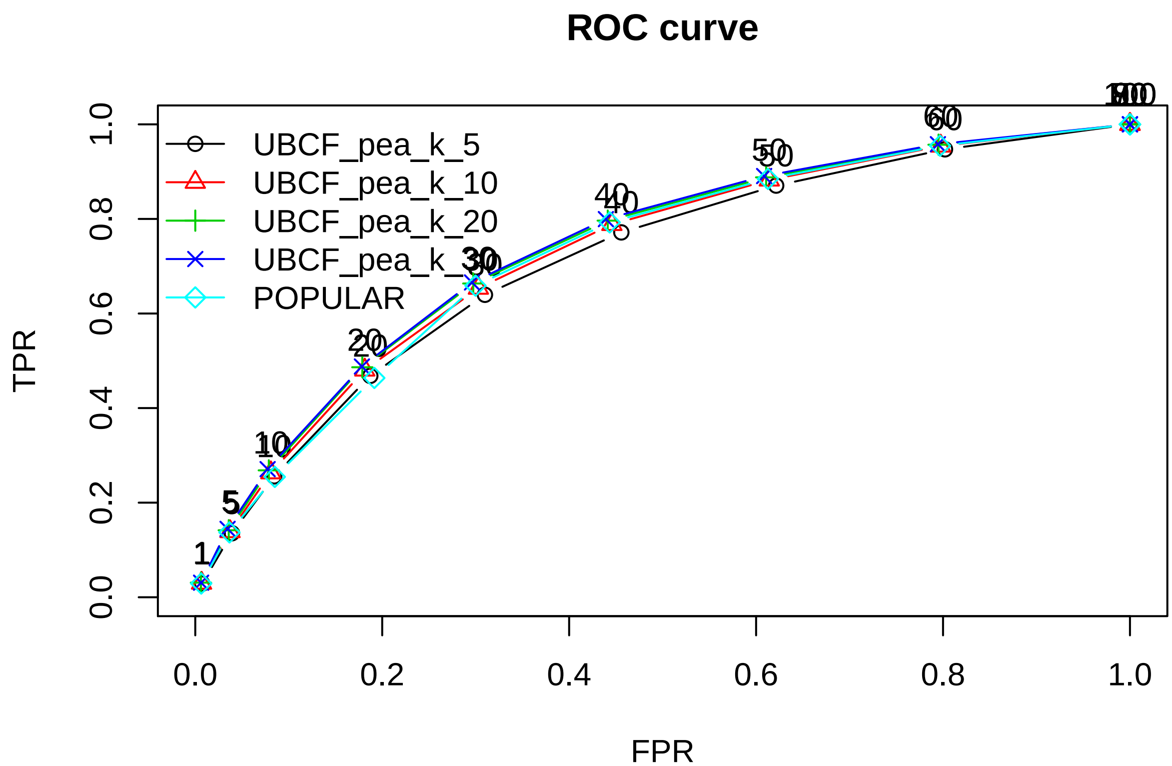
Plot POPULAR Results

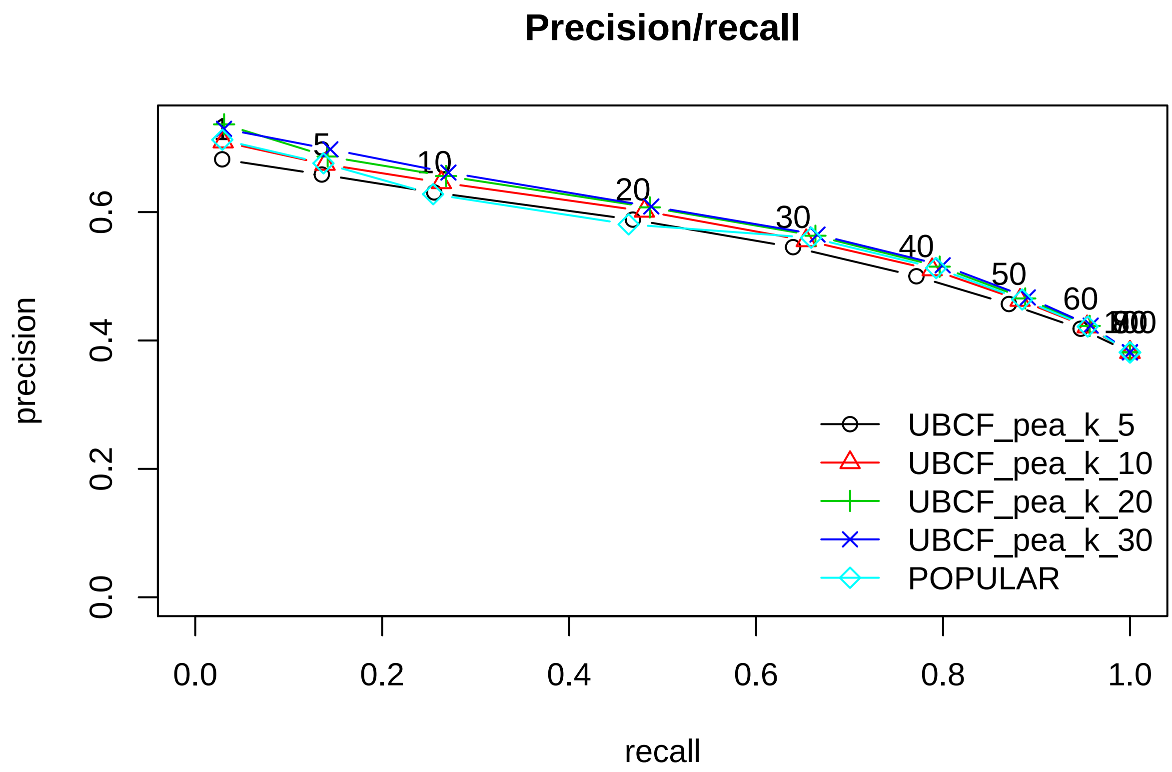

ROC curve
POPULAR
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
1
5
10
20
30
40
50
60



Combine best results from UBCF, IBCF and POPULAR models to determine my Final Model

Plot Best Results


ROC curve
UBCF_pea_k_5
UBCF_pea_k_10
UBCF_pea_k_20
UBCF_pea_k_30
POPULAR
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0



It appears that POPULAR model is best visualy with UBCF_per_k_30 being extremly close.

Final Result with Validation dataset

Based on the previous models I have reviewed all being very close, I will Evaluate the validation set using the same evaluation process and report the results.

```
## Total number of ratings in validation dataset:    13506
```

```
##       Total Users to Movies in validation set:    168 1003
```

```
items_to_keep <-30
rating_threshold <- 4
n_fold <- 5
n_recommendations <- c(1, 5, seq(10,100,10))
val_sets <- evaluationScheme(data = val_data, method = "cross-validation", train
                                = percentage_training, given = items_to_keep, goodRating = rating_thres
```

Final IBCF Evaluation results:

Final POPULAR Evaluation results:

```
POP_eval <- evaluate(x = val_sets, method = "POPULAR", n = n_recommendations, type = "ratings")
```

```
## POPULAR run fold/sample [model time/prediction time]
##   1  [0sec/0.01sec]
##   2  [0sec/0sec]
##   3  [0sec/0sec]
##   4  [0sec/0sec]
##   5  [0.02sec/0sec]
```

```r
head(getConfusionMatrix(POP_eval)[[1]])
```

```
##          RMSE      MSE       MAE
## res 0.8606893 0.740786 0.6763836
```

Conclusion

In conlusion the **POPULAR** algorithm model from the recommenderLab library, keeping **30** items with a predicted movie rating of **4** reports an RMSE of **0.860** which is lower RMSE than required. Although I though the content based filtering approach would have been the the best methods. Another huge take away fron this project is getting the data in the right format can vastly increase performance.

```
## The Prepared dataset was reduced by 1.651159 to 1 byte when converted from matrix to realRatingsMatri
```

As a legacy programmer from the days of assemble language in the 70's, this Data Science Class offered by Harvardx have given me new tools, toys and a different way to approach the future, Thank Rafael Irizarry

References

Michael Hahsler (2019). recommenderlab: Lab for Developing and Testing Recommender Algorithms. R package version 0.2-4. https://github.com/mhahsler/recommenderlab