Recommender System

Aayush Agrawal

March 16, 2017

```
#Installing Packages
suppressWarnings({ ## Supressing Warning messages
  suppressPackageStartupMessages({ ## Supressing startup messages
    library(dplyr)
    library(data.table)
    library(ggplot2)
    library(recommenderlab)
    library(reshape2)
    library(Matrix)
  })
})
```

1) Reading Dataset

```
#Importing datasets
beer_data <- read.csv('./beer_data.csv',na.strings = "")
print(dim(beer_data))
```

```
## [1] 475984      3
```

```
#Checking for duplicates
beer_data <- unique(beer_data)
print(dim(beer_data))
```

```
## [1] 475404      3
```

```
#Print summary
summary(beer_data)
```

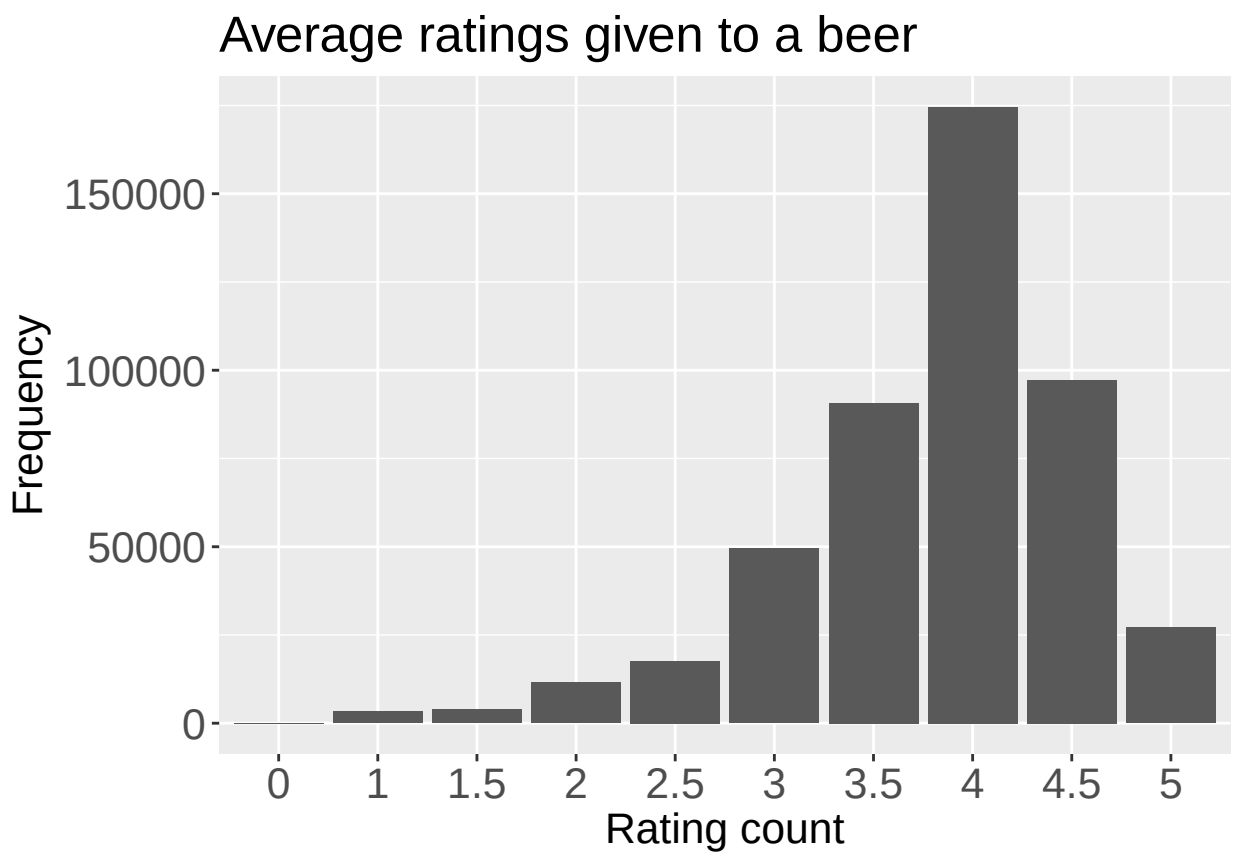
```
##   beer_beerid         review_profilename review_overall
##  Min.   :    3   northyorksammy:  1844   Min.   :0.000
##  1st Qu.: 1716   mikesgroove    :  1377   1st Qu.:3.500
##  Median :13896   BuckeyeNation  :  1337   Median :4.000
##  Mean   :21663   Thorpe429      :  1072   Mean   :3.814
##  3rd Qu.:39397   ChainGangGuy   :  1046   3rd Qu.:4.500
##  Max.   :77317   (Other)        :468628   Max.   :5.000
##                  NA's           :   100
```

2) Exploration-

How are rating distributed?

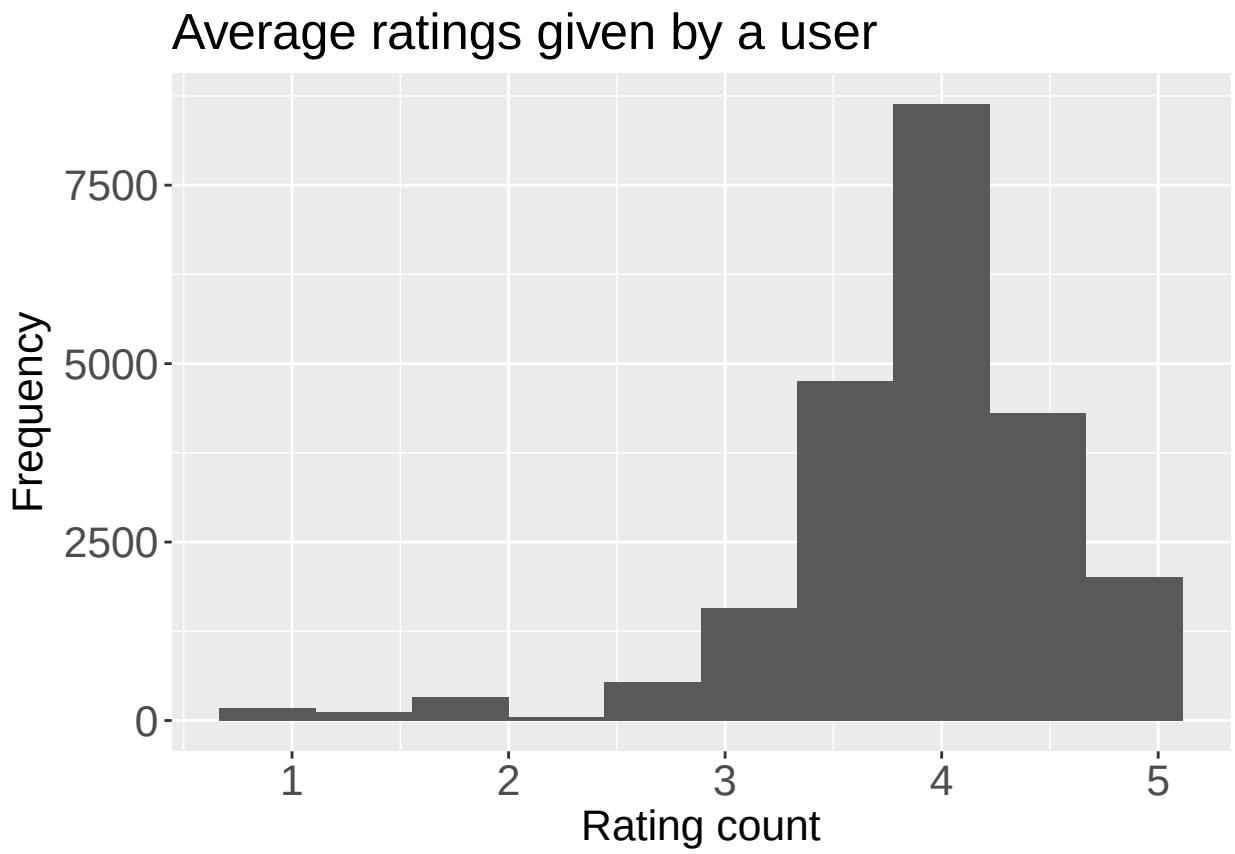
```
ggplot(data = beer_data, aes(x = factor(review_overall))) +
  geom_bar() +
```

```
  labs(title="Average ratings given to a beer", x="Rating count", y="Frequency") +
  theme(text = element_text(size=16),axis.text = element_text(size=16))
```



Distribution of Average rating by user?

```
x <- beer_data %>%
  group_by(review_profilename) %>%
  summarise(ratings_mean = mean(review_overall))
##How are rating distributed
ggplot(data = x, aes(x = ratings_mean)) +
  geom_histogram(bins = 10) +
  labs(title="Average ratings given by a user", x="Rating count", y="Frequency") +
  theme(text = element_text(size=16),axis.text = element_text(size=16))
```



Most people Rate movies highly.

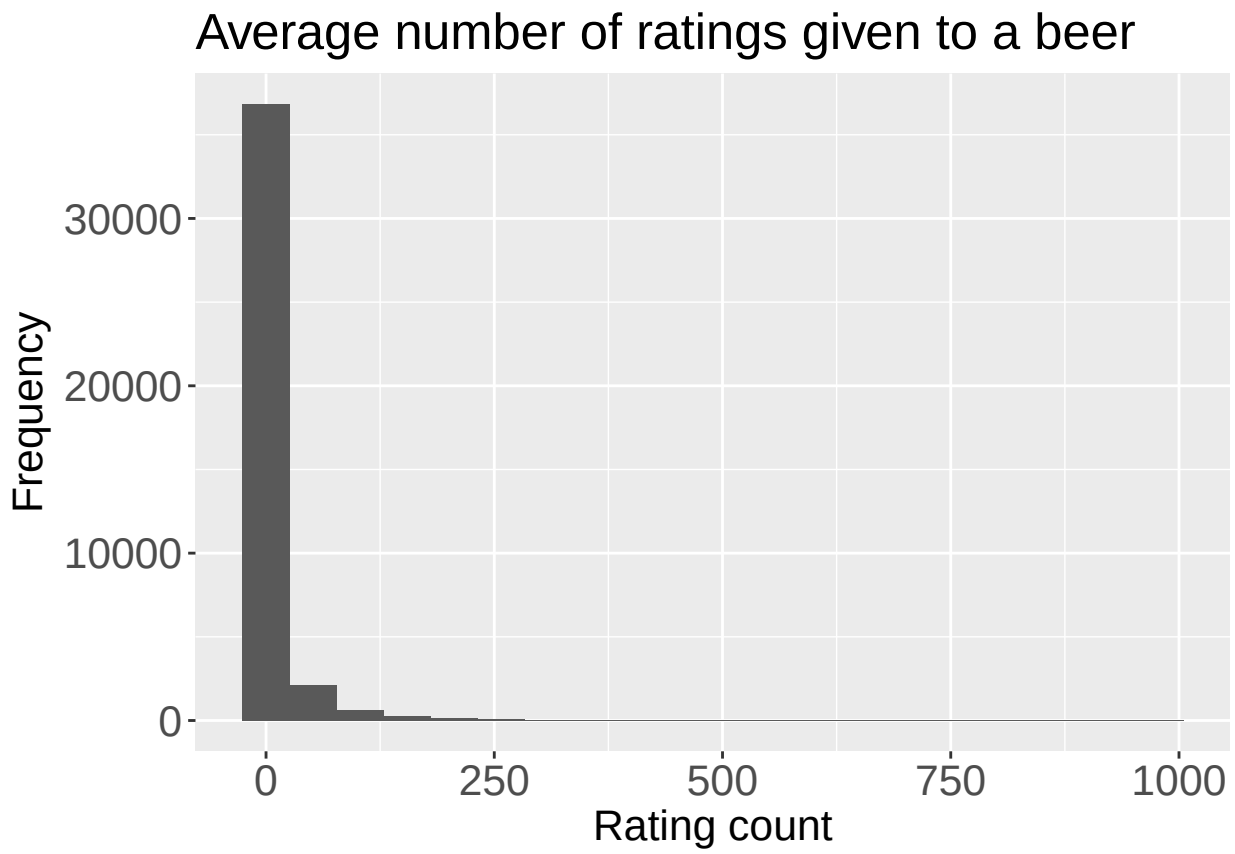
Distribution nummber of rating given to a beer?

```
beer_wise_count <- beer_data %>%
  group_by(beer_beerid) %>%
  summarise(rating_count = length(review_overall))

summary(beer_wise_count)
```

```
##   beer_beerid     rating_count
##  Min.   :    3   Min.   :  1.00
##  1st Qu.:16879   1st Qu.:  1.00
##  Median :37362   Median :  2.00
##  Mean   :36972   Mean   : 11.79
##  3rd Qu.:56232   3rd Qu.:  5.00
##  Max.   :77317   Max.   :980.00
```

```
ggplot(data= beer_wise_count,aes(x=rating_count)) +
  geom_histogram(bins = 20) +
  labs(title="Average number of ratings given to a beer", x="Rating count", y="Frequency") +
  theme(text = element_text(size=16),axis.text = element_text(size=16))
```



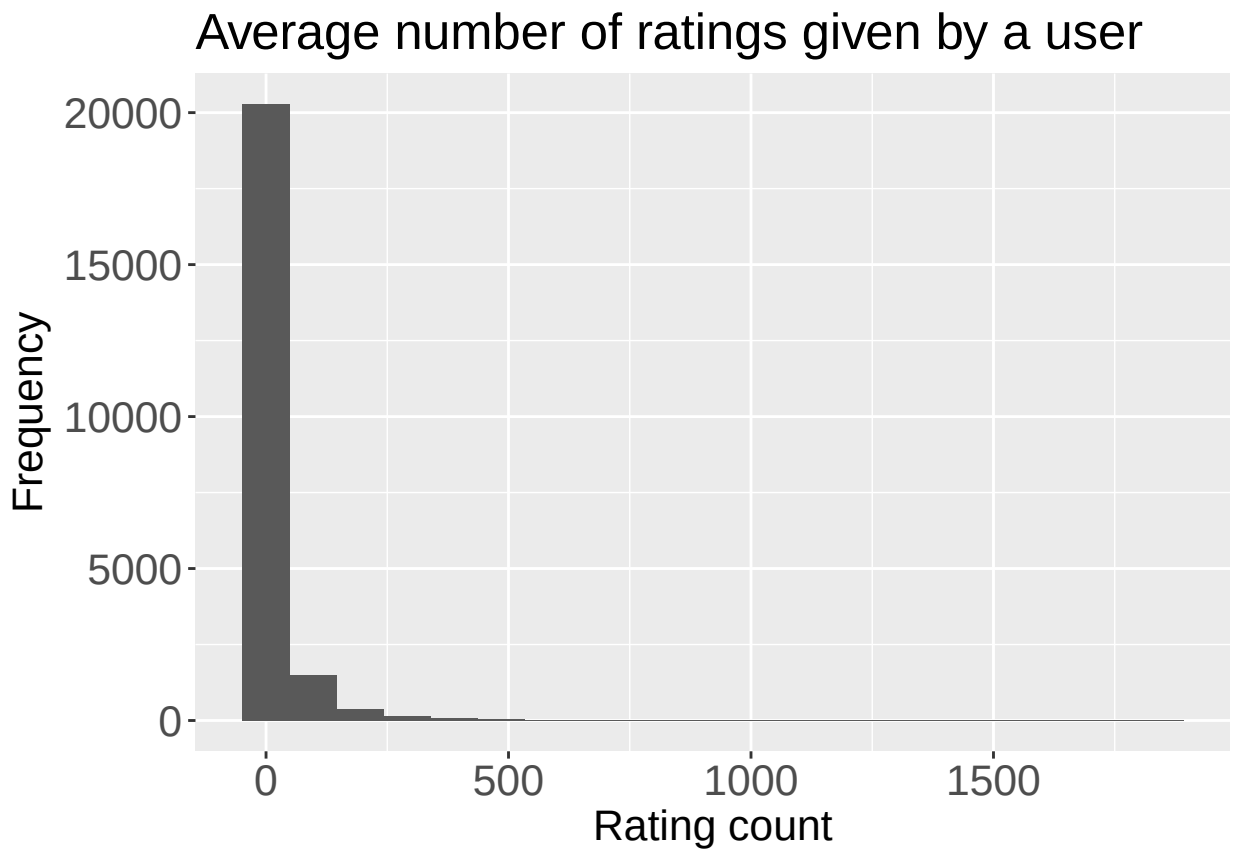
Distribution of average number rating given by a user?

```
review_profilename_count <- beer_data %>%
  group_by(review_profilename) %>%
  summarise(rating_count = length(review_overall))

summary(review_profilename_count)
```

```
##    review_profilename  rating_count
##  0110x011   :    1    Min.   :   1.00
##  01Ryan10   :    1    1st Qu.:   1.00
##  03SVTCobra :    1    Median :   3.00
##  04101Brewer:    1    Mean   :  21.13
##  05Harley   :    1    3rd Qu.:  11.00
##  (Other)    :22492    Max.   :1844.00
##  NA's       :    1
```

```
ggplot(data= review_profilename_count,aes(x=rating_count)) +
  geom_histogram(bins = 20) +
  labs(title="Average number of ratings given by a user", x="Rating count", y="Frequency") +
  theme(text = element_text(size=16),axis.text = element_text(size=16))
```



As we can see that only some beers have been rated many times while 75% of beer have less than 5 ratings. We will only use beers which atleast have more than 100 rating count. Let's check how much beers we would be convering.

```
sprintf('Coverage : %s%%',round(sum(beer_wise_count$rating_count >= 100)*100 / nrow(beer_wise_count),2))
```

```
## [1] "Coverage : 2.53%"
```

There are 2.53% of beers which have more than 100 ratings and we would be only interested in them. Recommender systems suffer with the problem of cold start and 100 is a good number to give appropriate recommendation. Let's filter our data for only these beers.

```
beer_of_interest <- beer_wise_count[beer_wise_count$rating_count >= 100,'beer_beerid']

beer_data_filter <- merge(beer_data, beer_of_interest, by = 'beer_beerid')

sprintf('Drop in observations : %s%%',(round(100 - (nrow(beer_data_filter)*100/nrow(beer_data)),2)))
```

```
## [1] "Drop in observations : 53.1%"
```

As we can see by only taking 2.53% of beers recommendation we only loose 53.1% of rating data. It's similar to Pareto principle as 20% things contribute to 80% of result. Also filter users who have rated more than 10 ratings

```
review_profilename_count <- beer_data_filter %>%
  group_by(review_profilename) %>%
  summarise(rating_count = length(review_overall))

user_of_interest <- review_profilename_count[review_profilename_count$rating_count > 10,'review_profile

beer_data_filter <- merge(beer_data_filter, user_of_interest, by = 'review_profilename')

sprintf('Drop in observations : %s%%',(round(100 - (nrow(beer_data_filter)*100/nrow(beer_data)),2)))
```

```
## [1] "Drop in observations : 60.8%"
```

OVerall drop in data is 60.8%

```
#Removing observations with no username
beer_data_filter <- beer_data_filter[!is.na(beer_data_filter$review_profilename),]

#This step will take a lot of time and memory
beer_data_filter1 <- acast(beer_data_filter, review_profilename ~ beer_beerid,value.var = 'review_overal

#Converting NaN to zero
beer_data_filter1[is.na(beer_data_filter1) ] = 0

## Making a sparse matrix
rating_matrix <- Matrix(beer_data_filter1,sparse = TRUE)
print(dim(rating_matrix))
```

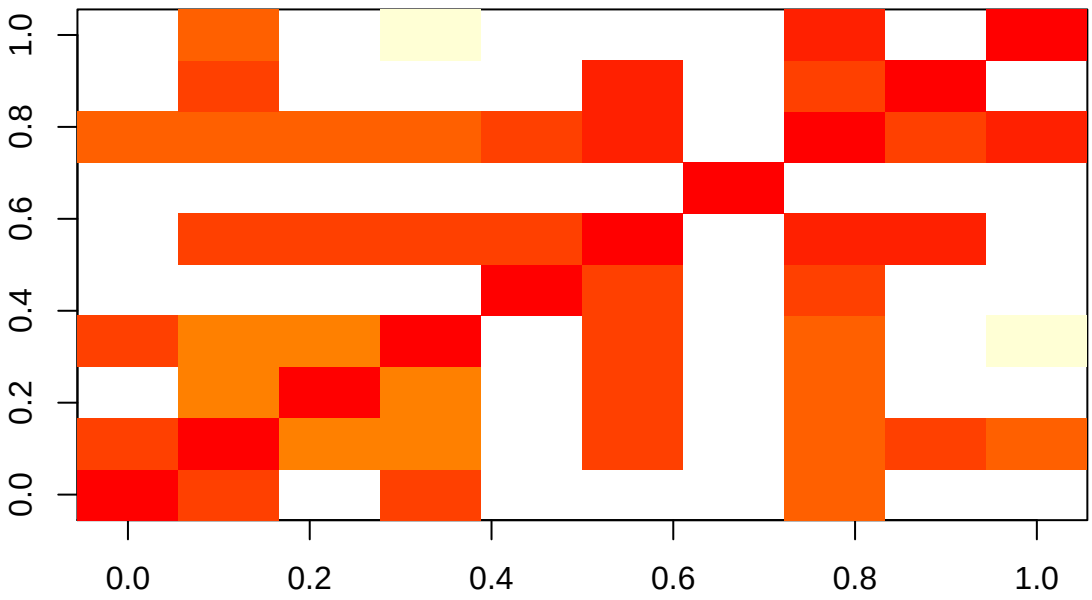
```
## [1] 4336 1020
```

That means 4336 users and 1020 beers. Now to convert the sparse matrix to a real rating matrix.

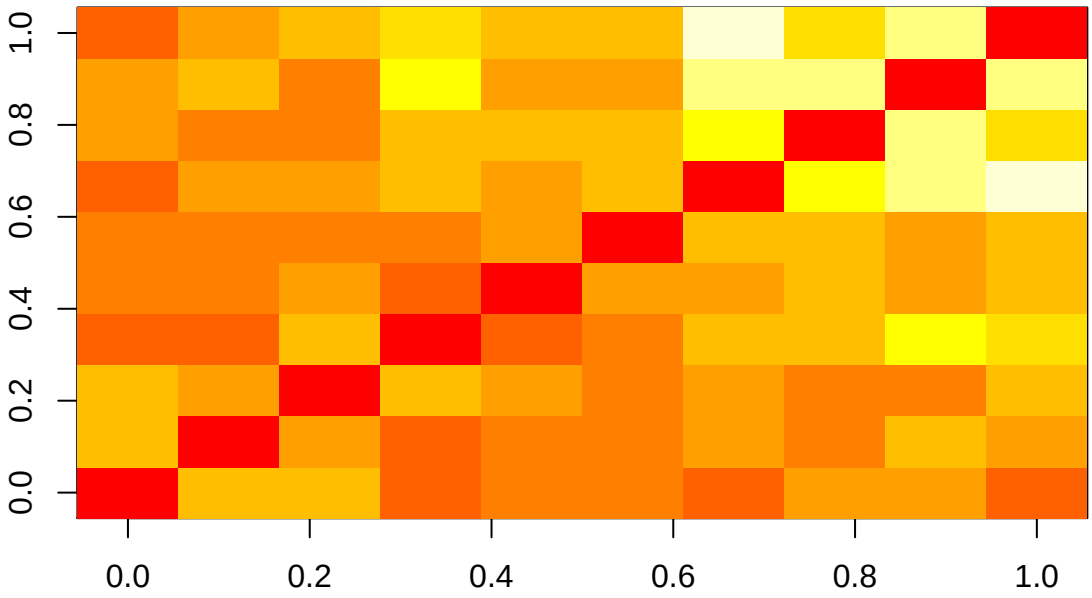
```
real_matrix <- new("realRatingMatrix", data = rating_matrix)
```

Let's visualize similarity between users and items

```
#First 10 users
image(as.matrix(similarity(real_matrix[1:10,],method = 'cosine',which = 'users')))
```




```
#First 10 items
image(as.matrix(similarity(real_matrix[,1:10],method = 'cosine',which = 'items')))
```



Let's start by doing a split validation of 80% and 20% and build our UBCF & IBCF models. Given is setted as 10 as we filtered that each beer will have atleast 10 ratings and goodRating 3 which is mid of 1-5.

Split Validation

```
# Making evaluation data set with 80 - 20 splot
eval_sets <- evaluationScheme(real_matrix, method = "split",train = 0.8, given = 10,goodRating = 3)

#Defining models to be evaluated
models <- list(
IBCF_cos = list(name = "IBCF",parameter = NULL),
UBCF_cos = list(name = "UBCF",parameter = NULL)
)

#evaluation
eval_results <- evaluate(x = eval_sets,method = models,n=seq(1,19,2))
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [15.35sec/0.44sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.02sec/18.15sec]
```

```
#Printing Model performance
print(eval_results$IBCF_cos@results[[1]])
```

```
## An object of class "confusionMatrix"
```

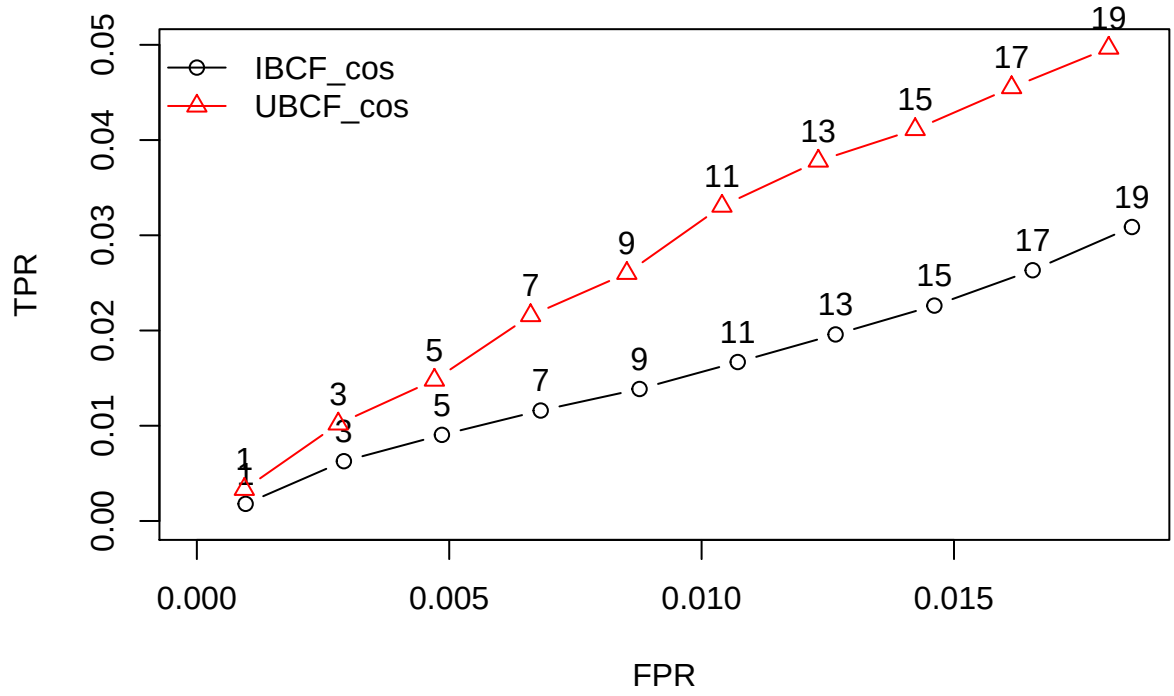
```
## Slot "cm":
##            TP         FP       FN       TN  precision      recall
## 1  0.05069124  0.9470046 30.71429 978.2880 0.05080831 0.001799046
## 3  0.13940092  2.8536866 30.62558 976.3813 0.04657429 0.006278469
## 5  0.23156682  4.7569124 30.53341 974.4781 0.04642032 0.009043235
## 7  0.31105991  6.6728111 30.45392 972.5622 0.04453976 0.011597887
## 9  0.38824885  8.5910138 30.37673 970.6440 0.04323839 0.013860945
## 11 0.47811060 10.4965438 30.28687 968.7385 0.04356498 0.016689937
## 13 0.57488479 12.3951613 30.19009 966.8399 0.04432404 0.019589084
## 15 0.65322581 14.3122120 30.11175 964.9228 0.04364896 0.022617441
## 17 0.74078341 16.2200461 30.02419 963.0150 0.04367613 0.026328629
## 19 0.80990783 18.1463134 29.95507 961.0887 0.04272517 0.030868014
##            TPR          FPR
## 1  0.001799046 0.0009664119
## 3  0.006278469 0.0029131228
## 5  0.009043235 0.0048552252
## 7  0.011597887 0.0068118412
## 9  0.013860945 0.0087703182
## 11 0.016689937 0.0107156847
## 13 0.019589084 0.0126539583
## 15 0.022617441 0.0146115516
## 17 0.026328629 0.0165585762
## 19 0.030868014 0.0185256546
##
## Slot "model":
## NULL
```

```
print(eval_results$UBCF_cos@results[[1]])
```

```
## An object of class "confusionMatrix"
## Slot "cm":
##            TP         FP       FN       TN  precision     recall
## 1  0.07488479  0.9228111 30.69009 978.3122 0.07505774 0.00332522
## 3  0.24654378  2.7465438 30.51843 976.4885 0.08237105 0.01019090
## 5  0.37327189  4.6152074 30.39171 974.6198 0.07482679 0.01479602
## 7  0.50000000  6.4838710 30.26498 972.7512 0.07159353 0.02157198
## 9  0.62788018  8.3513825 30.13710 970.8836 0.06992558 0.02600603
## 11 0.77534562 10.1993088 29.98963 969.0357 0.07064875 0.03306309
## 13 0.89976959 12.0702765 29.86521 967.1647 0.06937289 0.03779784
## 15 1.01382488 13.9516129 29.75115 965.2834 0.06774442 0.04111110
## 17 1.13709677 15.8237327 29.62788 963.4113 0.06704252 0.04550932
## 19 1.24769585 17.7085253 29.51728 961.5265 0.06581986 0.04965682
##           TPR          FPR
## 1  0.00332522 0.0009409987
## 3  0.01019090 0.0028011938
## 5  0.01479602 0.0047067307
## 7  0.02157198 0.0066126090
## 9  0.02600603 0.0085184728
## 11 0.03306309 0.0104038184
## 13 0.03779784 0.0123120902
## 15 0.04111110 0.0142310598
## 17 0.04550932 0.0161417802
## 19 0.04965682 0.0180648846
##
## Slot "model":
```

```
## NULL
```

```
#Making ROC curve
plot(eval_results, annotate = 1:2, legend="topleft")
```



As we can see UBCF model do better than IBCF models.

Cross validation - 5 Fold

```
#Making evaluation data set with 5 fold
eval_cv <- evaluationScheme(real_matrix, method="cross-validation",k=5,  given = 10, goodRating = 3)

#Evaluation
eval_results1 <- evaluate(x = eval_cv, method = models,n=seq(1,19,2))
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [15.09sec/0.39sec]
##   2  [16.28sec/0.5sec]
##   3  [15.42sec/0.38sec]
##   4  [15.2sec/0.39sec]
##   5  [15.19sec/0.49sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.02sec/17.53sec]
##   2  [0sec/17.42sec]
##   3  [0.02sec/16.11sec]
##   4  [0.02sec/17.37sec]
```

```
##   5  [0.02sec/16.59sec]
```

```
#Printing model performance
print(eval_results1$IBCF_cos@results[[1]])
```

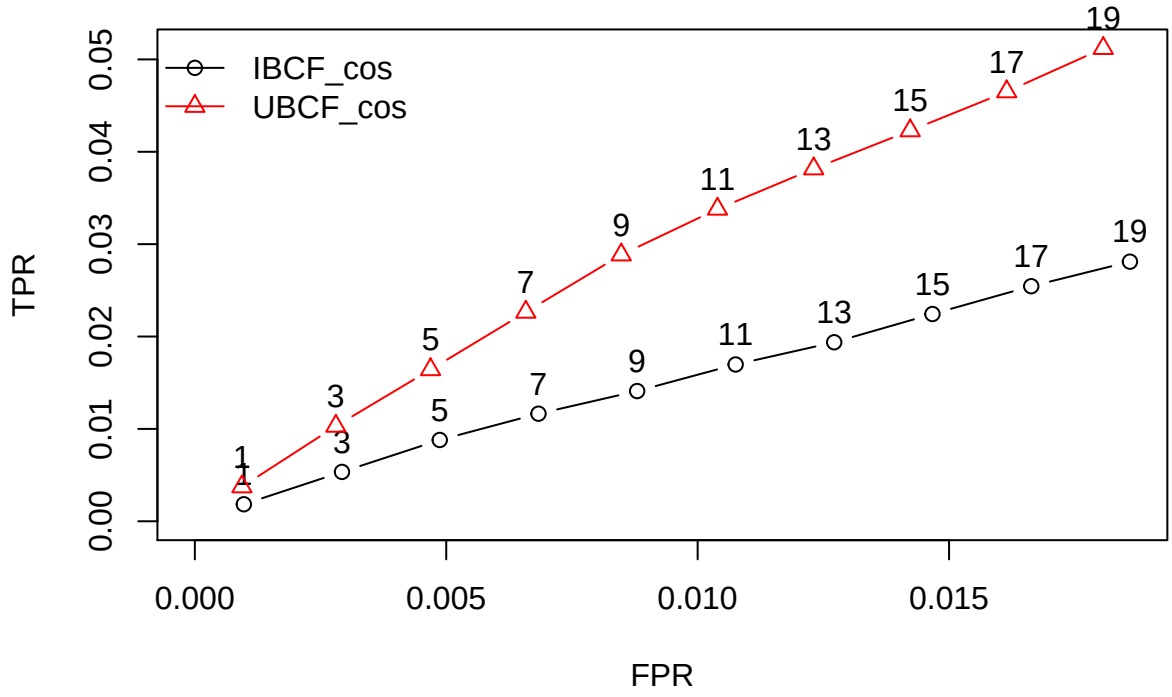
```
## An object of class "confusionMatrix"
## Slot "cm":
##            TP         FP       FN       TN  precision      recall
## 1  0.04608295  0.9539171 31.12097 977.8790 0.04608295 0.003385360
## 3  0.12672811  2.8732719 31.04032 975.9597 0.04224270 0.008438473
## 5  0.21774194  4.7822581 30.94931 974.0507 0.04354839 0.011814092
## 7  0.28341014  6.7165899 30.88364 972.1164 0.04048716 0.013640908
## 9  0.35829493  8.6417051 30.80876 970.1912 0.03981055 0.015381492
## 11 0.42741935 10.5725806 30.73963 968.2604 0.03885630 0.017728286
## 13 0.50921659 12.4907834 30.65783 966.3422 0.03917051 0.020651468
## 15 0.58525346 14.4147465 30.58180 964.4182 0.03901690 0.023705262
## 17 0.65552995 16.3444700 30.51152 962.4885 0.03856059 0.026393407
## 19 0.72235023 18.2776498 30.44470 960.5553 0.03801843 0.028849597
##            TPR          FPR
## 1  0.003385360 0.0009750832
## 3  0.008438473 0.0029362853
## 5  0.011814092 0.0048857316
## 7  0.013640908 0.0068616756
## 9  0.015381492 0.0088272582
## 11 0.017728286 0.0107999630
## 13 0.020651468 0.0127582928
## 15 0.023705262 0.0147239766
## 17 0.026393407 0.0166952216
## 19 0.028849597 0.0186703571
##
## Slot "model":
## NULL
```

```
print(eval_results1$UBCF_cos@results[[1]])
```

```
## An object of class "confusionMatrix"
## Slot "cm":
##            TP         FP       FN       TN  precision     recall
## 1  0.08064516  0.9193548 31.08641 977.9136 0.08064516 0.00353482
## 3  0.25000000  2.7500000 30.91705 976.0829 0.08333333 0.01067643
## 5  0.40322581  4.5967742 30.76382 974.2362 0.08064516 0.01697380
## 7  0.53456221  6.4654378 30.63249 972.3675 0.07636603 0.02216652
## 9  0.66589862  8.3341014 30.50115 970.4988 0.07398874 0.02670007
## 11 0.79377880 10.2062212 30.37327 968.6267 0.07216171 0.03262083
## 13 0.92050691 12.0794931 30.24654 966.7535 0.07080822 0.03621809
## 15 1.04147465 13.9585253 30.12558 964.8744 0.06943164 0.04083219
## 17 1.16013825 15.8398618 30.00691 962.9931 0.06824343 0.04427815
## 19 1.26958525 17.7304147 29.89747 961.1025 0.06682028 0.04770660
##           TPR          FPR
## 1  0.00353482 0.0009387554
## 3  0.01067643 0.0028058416
## 5  0.01697380 0.0046900401
## 7  0.02216652 0.0065958916
## 9  0.02670007 0.0085032444
## 11 0.03262083 0.0104151621
```

```
## 13 0.03621809 0.0123262912
## 15 0.04083219 0.0142440183
## 17 0.04427815 0.0161640617
## 19 0.04770660 0.0180936658
##
## Slot "model":
## NULL
```

```
#Making ROC curve
plot(eval_results1, annotate = 1:2, legend="topleft")
```



As we can see UBCF model do better than IBCF models even in cross validation approach.

Making predictions using IBCF

```
#Building UBCF model
Rec.model=Recommender(real_matrix,method="UBCF")

#Making Predictions
recom <- predict(Rec.model,1:dim(real_matrix)[1],data = real_matrix,n=5)

# For cokes
as(recom, "list")$cokes
```

```
## [1] "1010"  "99"    "353"   "29619" "224"
```

```r
# For genog
as(recom, "list")$genog
```

```
## [1] "1093"  "1708"  "19960" "11757" "412"
```

```r
# For giblet
as(recom, "list")$giblet
```

```
## [1] "56973" "88"    "1093"  "1118"  "19960"
```