Project 2

Alejandro D. Osborne

June 18, 2019

```r
library(dplyr)
library(tidyr)
library(ggplot2)
library(recommenderlab)
library(knitr)
```

MovieLens is a data set that contains ratings from the MovieLense website (http://movielens.org). This data set is broken into a number of different sizes for research purposes. We will be using the small data set containing 100,000 ratings applied to 9,000 movies by 700 users

Using the MovieLense data set we will construct a user-based collaborative filter and an item-based collaborative filter recommender systems. We will be using the *recommenderlab* package to assist the building of these systems.

The MovieLense data is broken into two datasets that we are interested in. The first is the movies data that contains the movie ID, the title, and genres. The second is the user ratings data that contains the user ID, the movie ID, and the rating, and a time-stamp that we will mot be using.

```r
movies = read.csv("https://raw.githubusercontent.com/AlejandroOsborne/DATA612/master/movies.csv",
                  header = TRUE, sep = ",", stringsAsFactors = FALSE)
ratings = read.csv("https://raw.githubusercontent.com/AlejandroOsborne/DATA612/master/ratings.csv",
                   header = TRUE, sep =",", stringsAsFactors = FALSE)
ratings <- ratings[,c(1,2,3)]
kable(head(movies))
```

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

```r
kable(head(ratings))
```

userId	movieId	rating
1	1	4
1	3	4
1	6	4
1	47	5
1	50	5
1	70	3
Now we bu	ild the us	er item matrix. Both datasets were reduced in order to help R better process the data.

```r
x = head(data.frame(table(ratings$userId) %>% sort(decreasing =T)), 10000)

colnames(x) = c("userId", "count")

ratex = merge(ratings , x, by="userId")

submovie = subset(movies, select = c("movieId", "title"))

subrating = subset(ratex, select = c("userId","movieId", "rating"))

prematrix = subset(merge(subrating, submovie, by="movieId"), select = c("userId","rating","title"))

y = head(data.frame(table(prematrix$title) %>% sort(decreasing =T)), 500)

colnames(y) = c("title", "count2")

prematrix2 = subset(merge(prematrix, y, by="title"), select = c("title","userId","rating"))

kable(head(prematrix2))
```

title	userId	rating
10 Things I Hate About You (1999)	298	2.5
10 Things I Hate About You (1999)	111	4.0
10 Things I Hate About You (1999)	12	5.0
10 Things I Hate About You (1999)	153	1.0
10 Things I Hate About You (1999)	260	4.5
10 Things I Hate About You (1999)	275	4.0

```r
# Creating a sparse matrix of users as rows and movieId as columns.
user_movie <- ratings %>%
  spread(key = movieId, value = rating) %>%
  as.matrix()
user_movie = user_movie[,-1] #remove userID col. Rows are userIds, cols are movieIds
user_movie <- as(user_movie, "realRatingMatrix")
```

We Reduce the set further to movies that have been reviewed more than 10 times, this leaves us with around 80K ratings to go of off.

```r
dim(user_movie[,colCounts(user_movie) > 10])
```

```
## [1]  610 2121
```

```r
user_movie <- user_movie[,colCounts(user_movie) > 10]
user_movie
```

```
## 610 x 2121 rating matrix of class 'realRatingMatrix' with 79636 ratings.
```

With this we split into the training and test sets so that we can better compare the different models with the training set getting 80% of the data:

```r
set.seed(42)
movie <- evaluationScheme(user_movie, method = "split", train = .8, given = 5, goodRating = 3)
movie
```

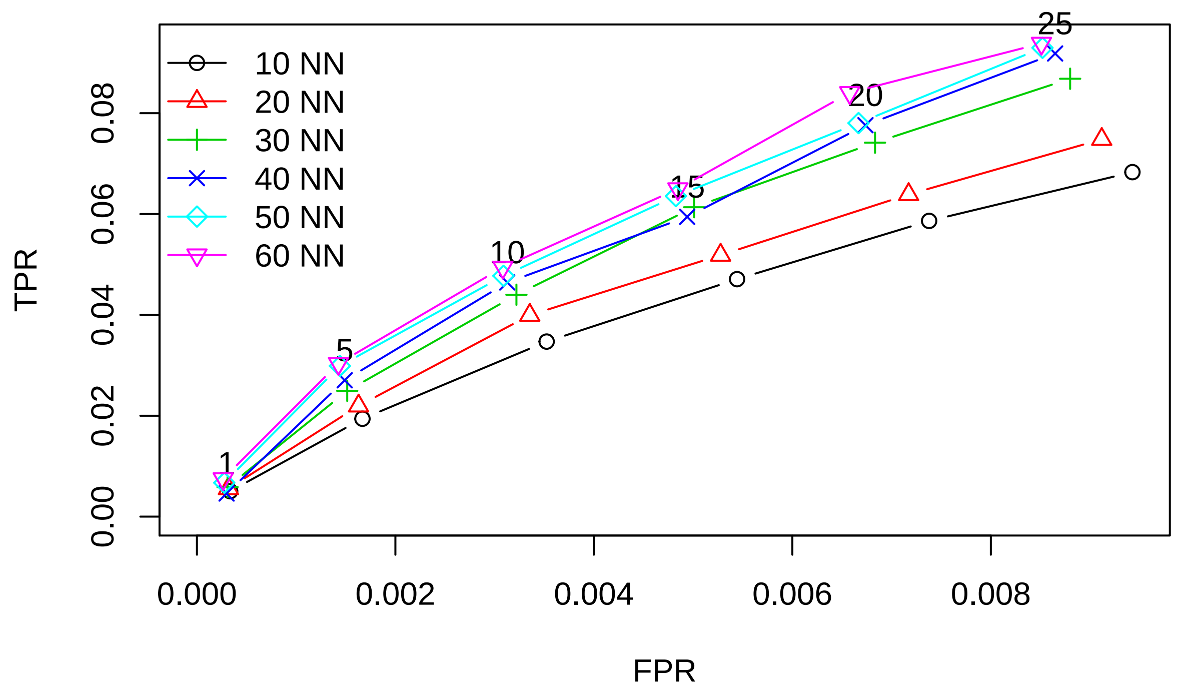
```
## Evaluation scheme with 5 items given
## Method: 'split' with 1 run(s).
## Training set proportion: 0.800
## Good ratings: >=3.000000
## Data set: 610 x 2121 rating matrix of class 'realRatingMatrix' with 79636 ratings.
```

Now we can start by looking at 6 neighborhood sizes for the User-Based Colaborative Filtering recommender:

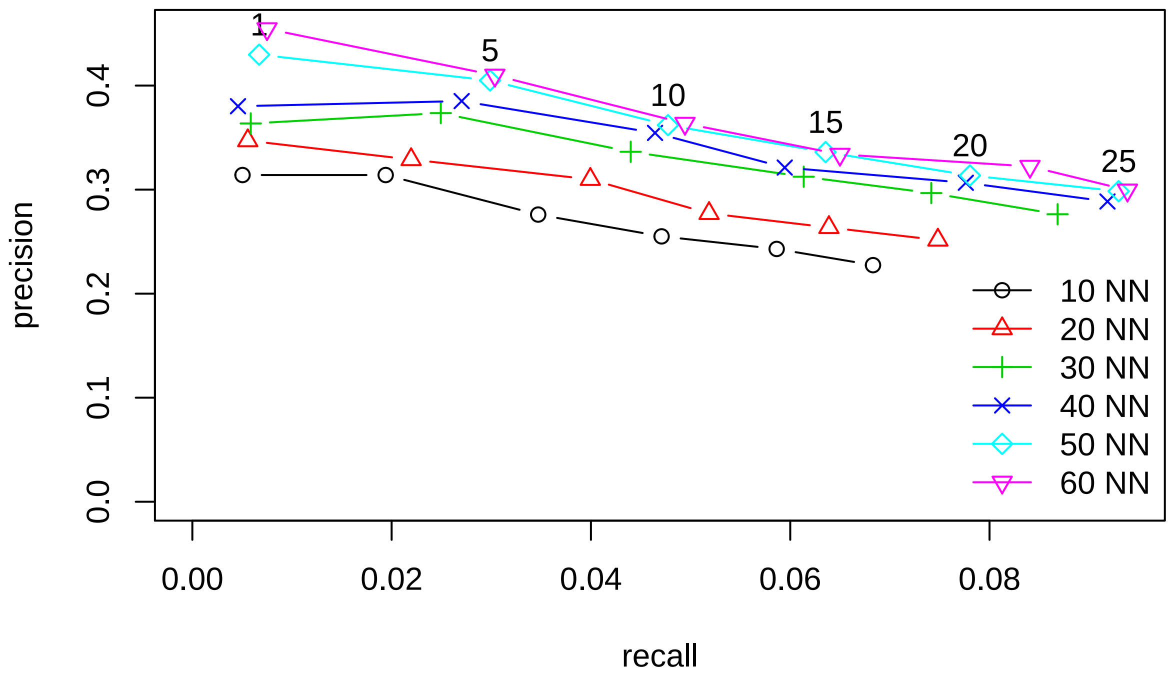
```r
#recommenderlab can run multiple models at once so we will

user_nn <- list(
  "10 NN" = list(name="UBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        nn=10)),
  "20 NN" = list(name="UBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        nn=20)),
  "30 NN" = list(name="UBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        nn=30)),
  "40 NN" = list(name="UBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        nn=40)),
  "50 NN" = list(name="UBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        nn=50)),
  "60 NN" = list(name="UBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        nn=60))
)
# Predict next n movies for comparison purposes
recs <- c(1,5, 10, 15, 20, 25)
user_nn_results <- evaluate(movie, user_nn, n = recs, progress = FALSE)
```

```r
# Draw the ROC curve
plot(x = user_nn_results, y = "ROC", annotate = 4, legend="topleft")
```



```r
# Draw the precision / recall curve
plot(x = user_nn_results, y = "prec/rec", annotate = 5)
```



The above graphs show little difference between 40,50, and 60 nearest neighbors in the ROC and precision vs recall curves for all numbers of recommendations. We use 50 for our best candidate and use this value to check other parameters. We see that using these parameters gives us a RMSE of 1.008187.

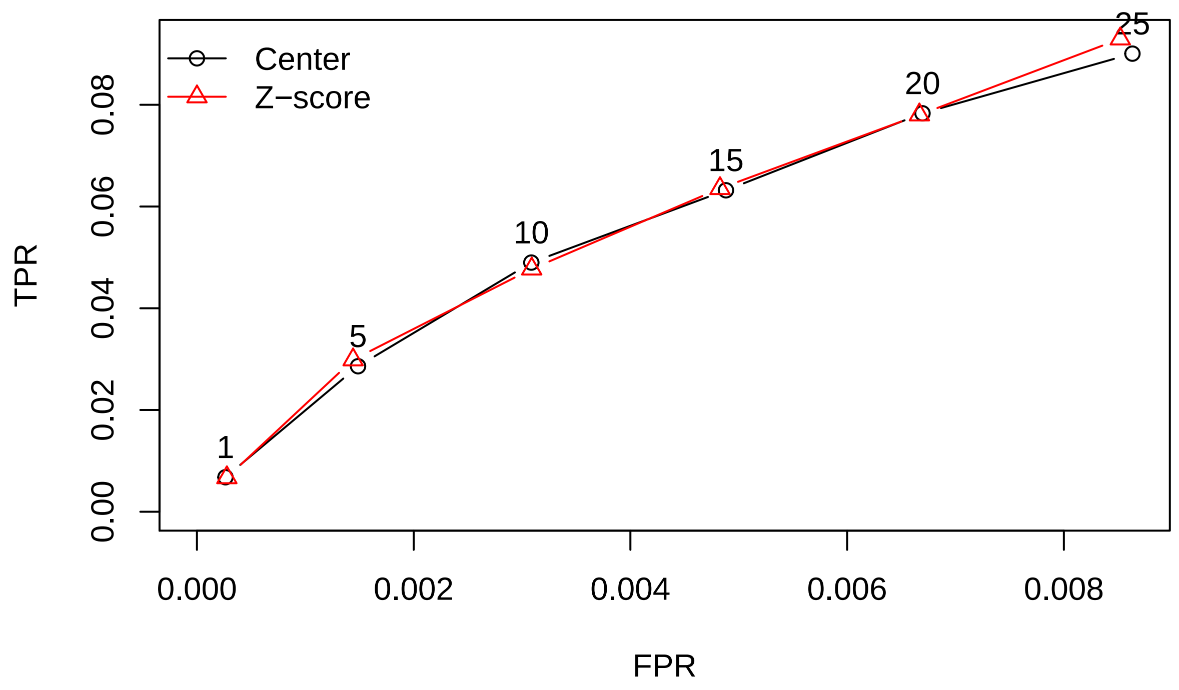
```
model <- Recommender(getData(movie, "train"), method = "UBCF",
                     param=list(normalize = "Z-Score", method="Cosine", nn=50))
prediction <- predict(model, getData(movie, "known"), type="ratings")
rmse_ubcf <- calcPredictionAccuracy(prediction, getData(movie, "unknown"))[1]
rmse_ubcf
```

```
##     RMSE
## 1.008187
```

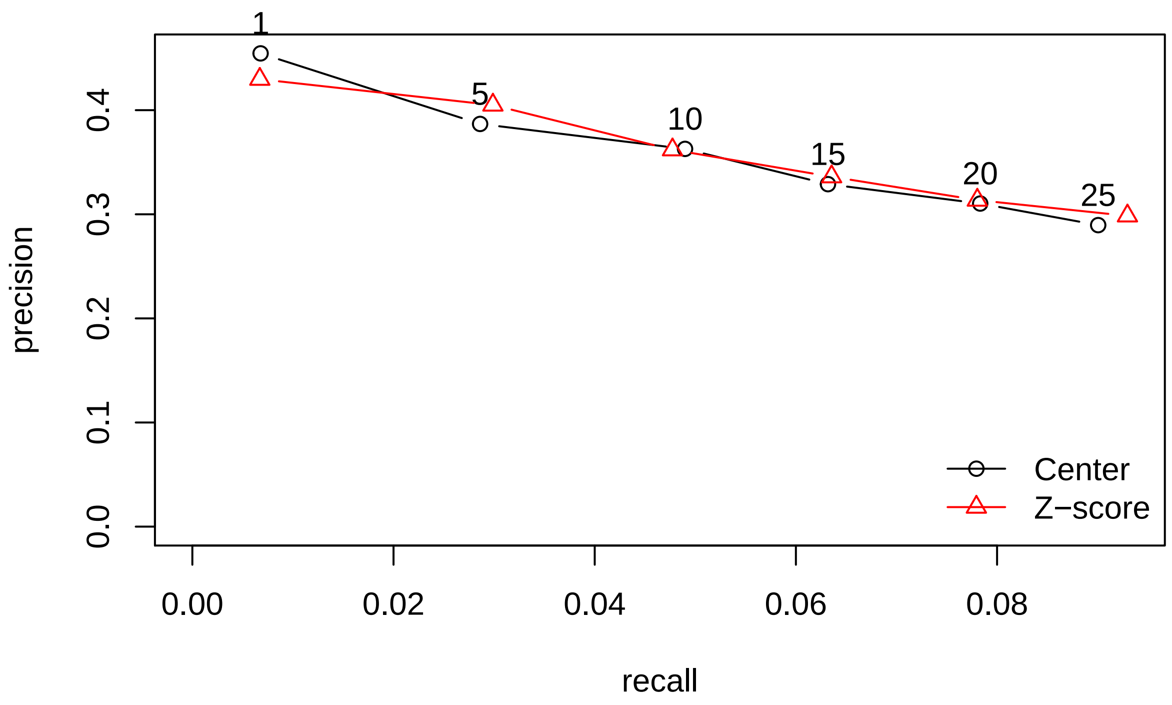
Recommenderlab supports center and z-score as methods of normalization:

```
user_norm <- list(
  "Center" = list(name="UBCF", param=list(normalize = "center",
                                          method="Cosine",
                                          nn=50)),
  "Z-score" = list(name="UBCF", param=list(normalize = "Z-score",
                                          method="Cosine",
                                          nn=50))
)
user_norm_results <- evaluate(movie, user_norm, n = recs, progress = FALSE)
```

```
# Draw the ROC curve
plot(x = user_norm_results, y = "ROC", annotate = 1, legend="topleft")
```



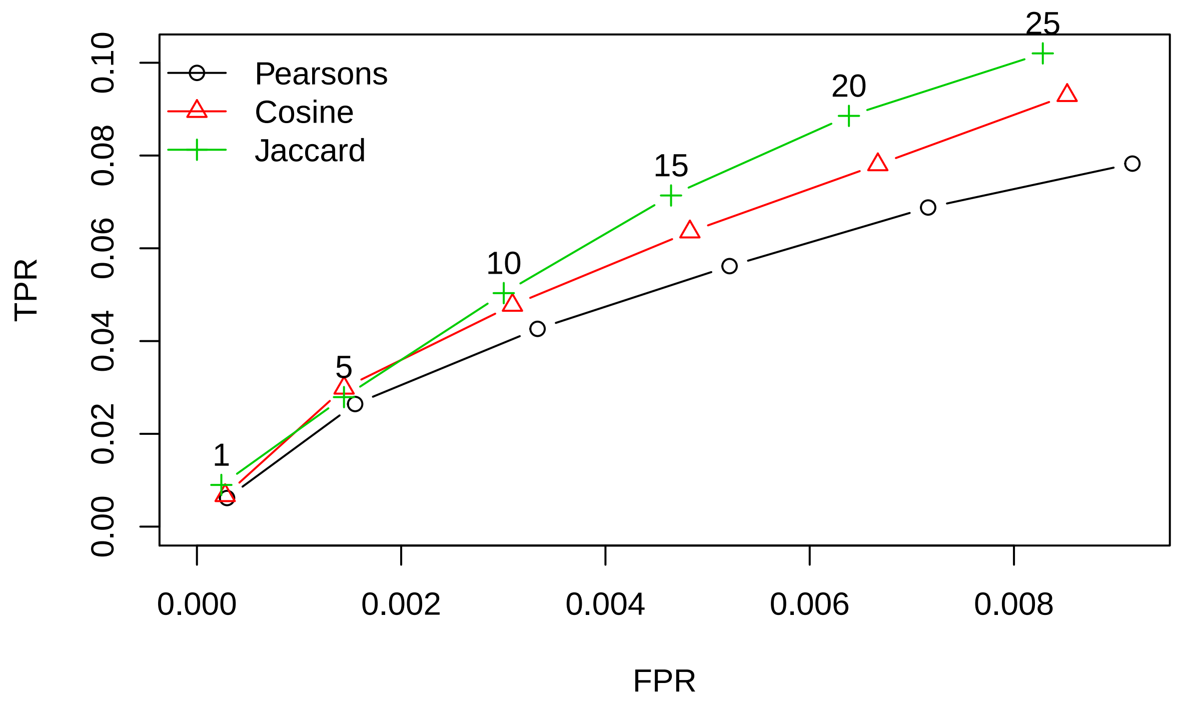
```r
# Draw the precision / recall curve
plot(x = user_norm_results, y = "prec/rec", annotate = 1)
```



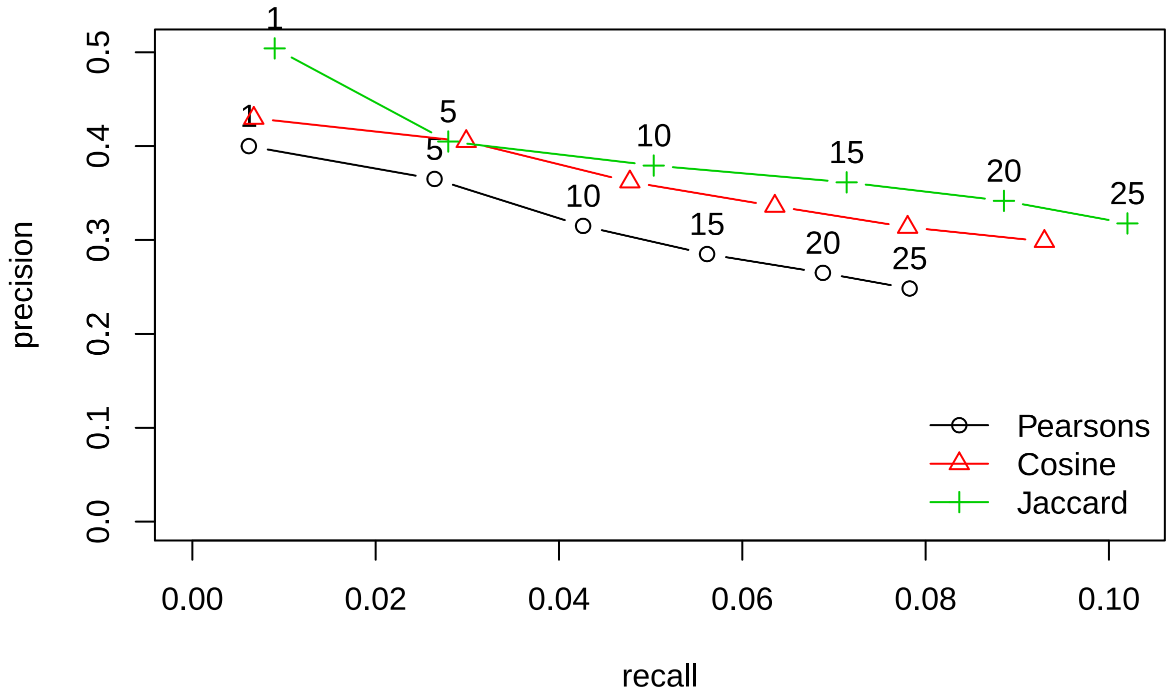
The Z-Score does a slightly better job at many of the recommendation levels, so we'll use this for our UBCF.

```
user_dist <- list(
  "Pearsons" = list(name="UBCF", param=list(normalize = "z-score",
                                          method="pearson",
                                          nn=50)),
  "Cosine" = list(name="UBCF", param=list(normalize = "Z-score",
                                          method="Cosine",
                                          nn=50)),
  "Jaccard" = list(name="UBCF", param=list(normalize = "Z-score",
                                          method="jaccard",
                                          nn=50))
)
user_dist_results <- evaluate(movie, user_dist, n = recs, progress = FALSE)
```

```
# ROC curve
plot(x = user_dist_results, y = "ROC", annotate = 3, legend="topleft")
```



```
# precision / recall curve
plot(x = user_dist_results, y = "prec/rec", annotate = c(1,3))
```



Jaccard Distance RMSE

```
model <- Recommender(getData(movie, "train"), method = "UBCF",
                     param=list(normalize = "Z-Score", method="jaccard", nn=50))
prediction <- predict(model, getData(movie, "known"), type="ratings")
rmse_dist <- calcPredictionAccuracy(prediction, getData(movie, "unknown"))[1]
rmse_dist
```

```
##      RMSE
## 0.9998611
```

Pearson's Distance RMSE

```
model <- Recommender(getData(movie, "train"), method = "UBCF",
                     param=list(normalize = "Z-Score", method="pearson", nn=50))
prediction <- predict(model, getData(movie, "known"), type="ratings")
rmse_dist <- calcPredictionAccuracy(prediction, getData(movie, "unknown"))[1]
rmse_dist
```

```
##      RMSE
## 0.9905006
```

The Jaccard distance slightly outperforms the other methods but the Pearson's distance has very strong performance for smaller amounts of recommendations.

Item Based Collaborative Filtering

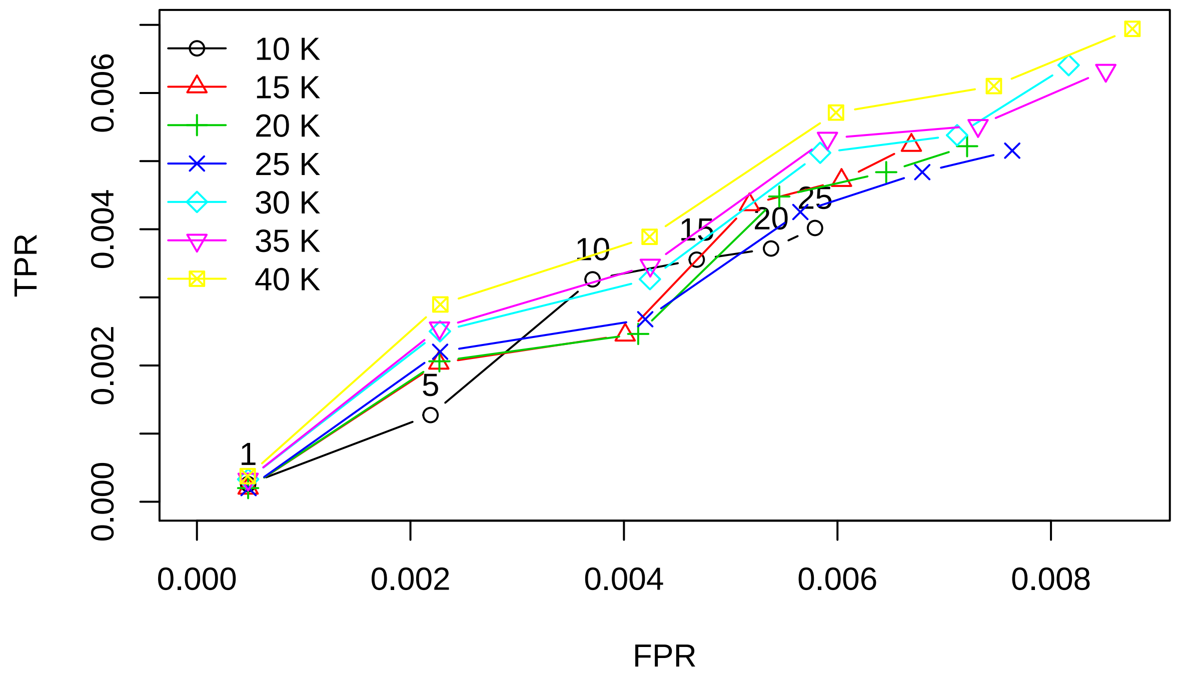
Now we will work on developing an item based collaborative filter. We will follow the same methodology as a above to find our best item based model and compare these two models to each other.

Neighborhood Size

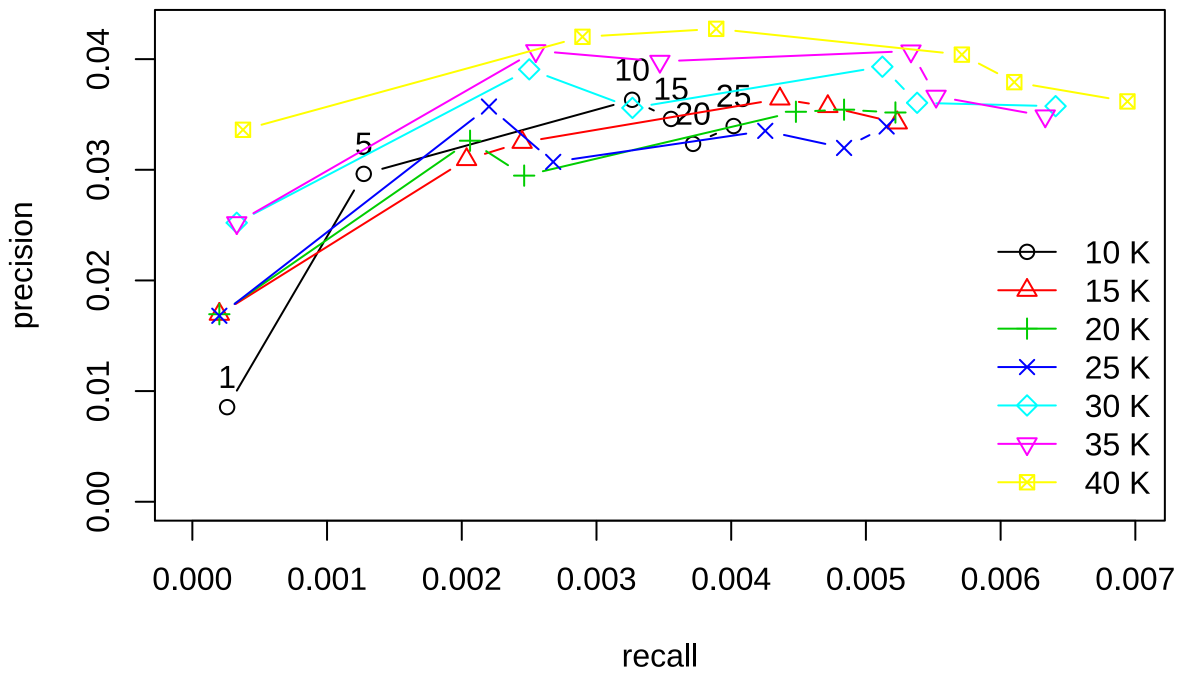
Let's try different neighborhood sizes for the item-based CF.

```r
item_nn <- list(
  "10 K" = list(name="IBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        k=10)),
  "15 K" = list(name="IBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        k=15)),
  "20 K" = list(name="IBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        k=20)),
  "25 K" = list(name="IBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        k=25)),
  "30 K" = list(name="IBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        k=30)),
  "35 K" = list(name="IBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        k=35)),
  "40 K" = list(name="IBCF", param=list(normalize = "Z-score",
                                        method="Cosine",
                                        k=40))
)
# Predict the next n movies for comparisons
item_nn_results <- evaluate(movie, item_nn, n = recs, progress = FALSE)
```

```r
# Draw the ROC curve
plot(x = item_nn_results, y = "ROC", annotate = 1, legend="topleft")
```



```
# Draw the precision / recall curve
plot(x = item_nn_results, y = "prec/rec", annotate = 1)
```



Calculating the RMSE

From the above graphs we see that the best performance comes from including 10 items in the neighborhood in the ROC and precision vs recall curves for all numbers of recommendations. We see that using these parameters gives us a RMSE of 1.405346.

```
item_model <- Recommender(getData(movie, "train"), method = "IBCF",
                          param=list(normalize = "Z-Score", method="Cosine", k=40))
item_prediction <- predict(item_model, getData(movie, "known"), type="ratings")
rmse_ibcf <- calcPredictionAccuracy(item_prediction, getData(movie, "unknown"))[1]
rmse_ibcf
```

```
##     RMSE
## 1.405346
```

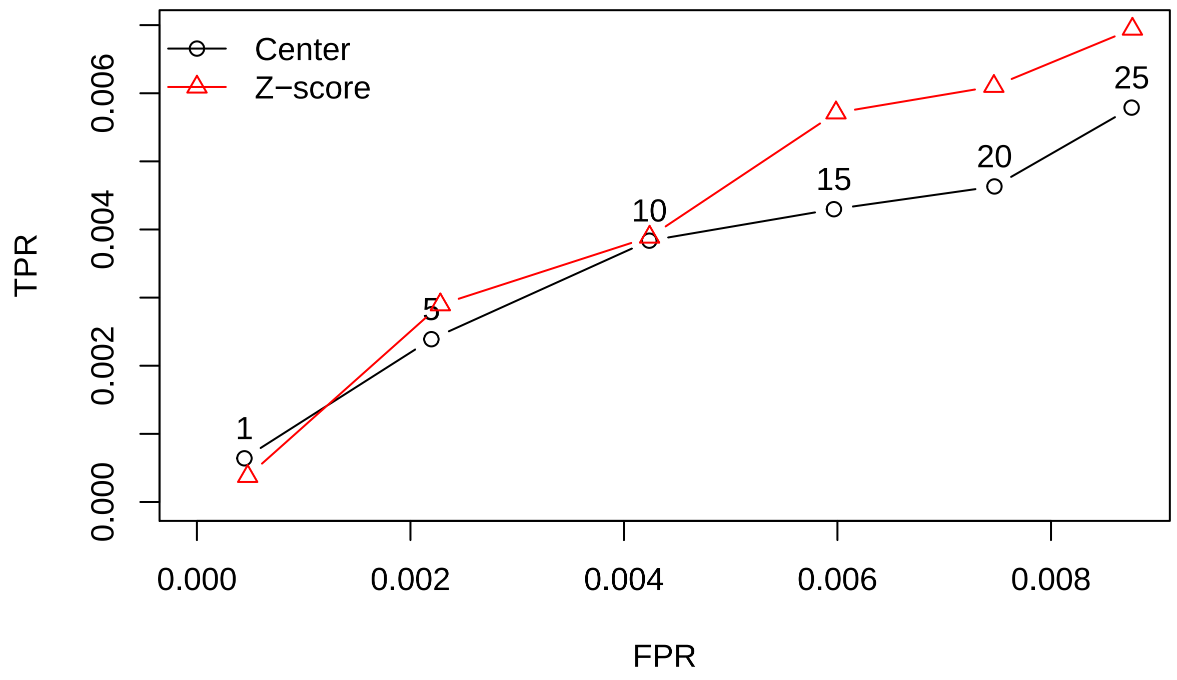
Normalization

Just like with UBCF we need to check out the Z-score and Centering

```
item_norm <- list(
  "Center" = list(name="IBCF", param=list(normalize = "center",
                                          method="Cosine",
                                          k=40)),
  "Z-score" = list(name="IBCF", param=list(normalize = "Z-score",
                                          method="Cosine",
                                          k=40))
)
item_norm_results <- evaluate(movie, item_norm, n = recs, progress = FALSE)
```

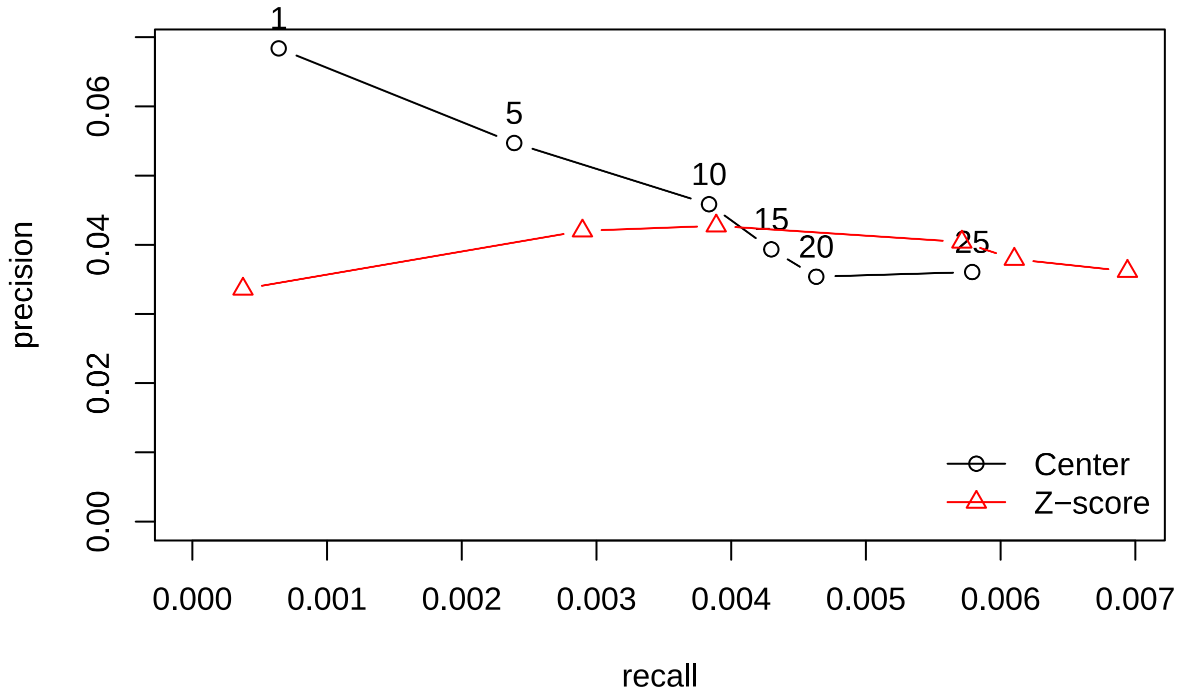
Drawing the ROC Curve

```r
# Draw the ROC curve
plot(x = item_norm_results, y = "ROC", annotate = 1, legend="topleft")
```



Dawing the Precision / Recall Curve

```r
# Draw the precision / recall curve
plot(x = item_norm_results, y = "prec/rec", annotate = 1)
```



Z-Score centering Still does a better job.

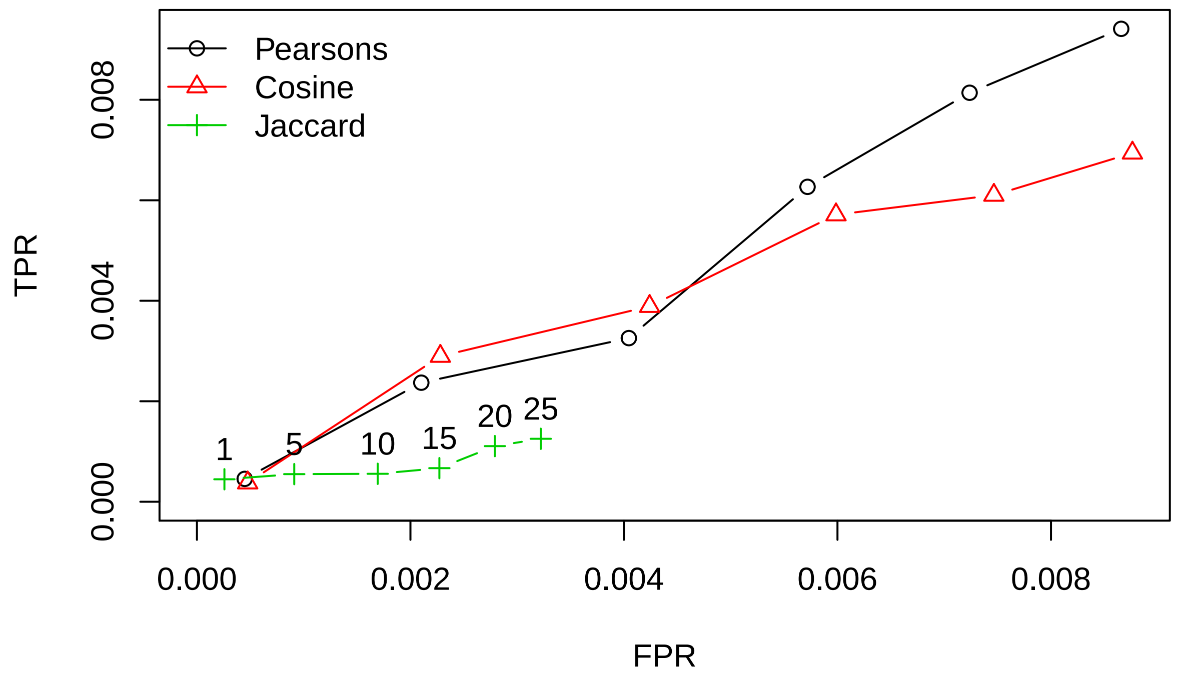
3.2.3 Distance Methods

The final parameter that we will be tweaking is the measurement of the distance or similarity of an item and their nearest neighbors, similarly to the UBCF model.

```r
item_dist <- list(
  "Pearsons" = list(name="IBCF", param=list(normalize = "z-score",
                                            method="pearson",
                                            k=40)),
  "Cosine" = list(name="IBCF", param=list(normalize = "Z-score",
                                            method="Cosine",
                                            k=40)),
  "Jaccard" = list(name="IBCF", param=list(normalize = "Z-score",
                                            method="jaccard",
                                            k=40))
)
item_dist_results <- evaluate(movie, item_dist, n = recs, progress = FALSE)
```

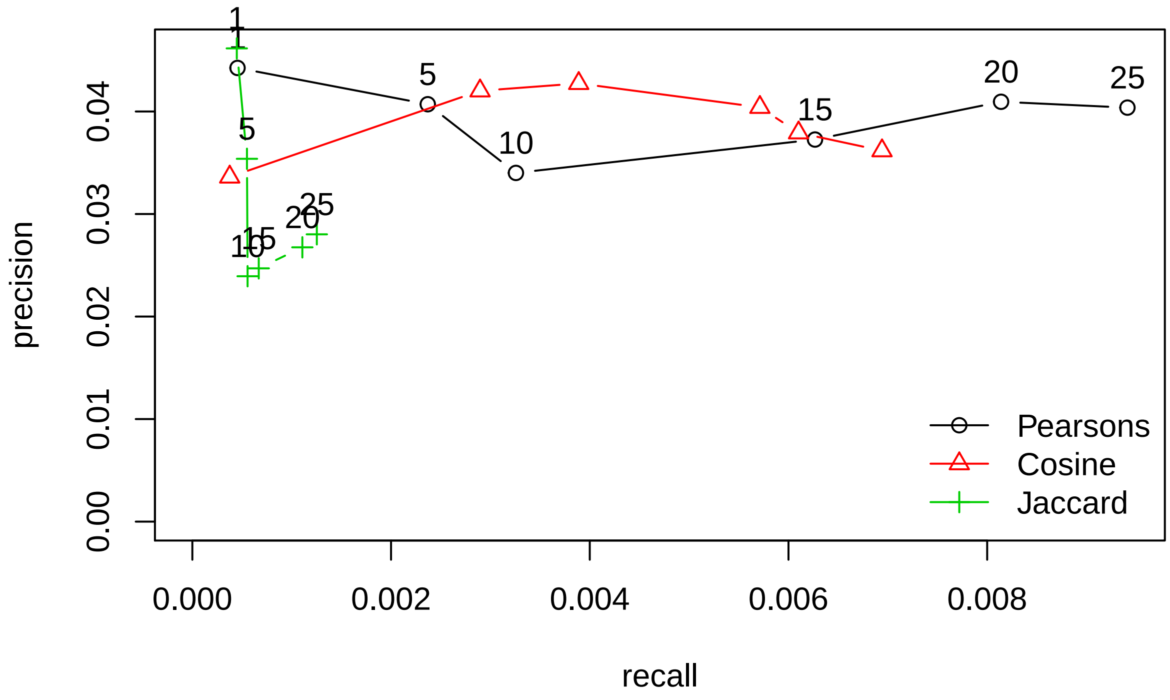
Drawing the ROC Curve

```r
# Draw the ROC curve
plot(x = item_dist_results, y = "ROC", annotate = 3, legend="topleft")
```



Dawing the Precision / Recall Curve

```
# Draw the precision / recall curve
 plot(x = item_dist_results, y = "prec/rec", annotate = c(1,3))
```



Calculating the RMSE We can see from the above graphs that the Pearson's distance seems to outperform the other distance methods.

```
model <- Recommender(getData(movie, "train"), method = "IBCF",
                     param=list(normalize = "Z-Score", method="pearson", k=40))
prediction <- predict(model, getData(movie, "known"), type="ratings")
rmse_item <- calcPredictionAccuracy(prediction, getData(movie, "unknown"))[1]
rmse_item
```

```
##    RMSE
## 1.38961
```

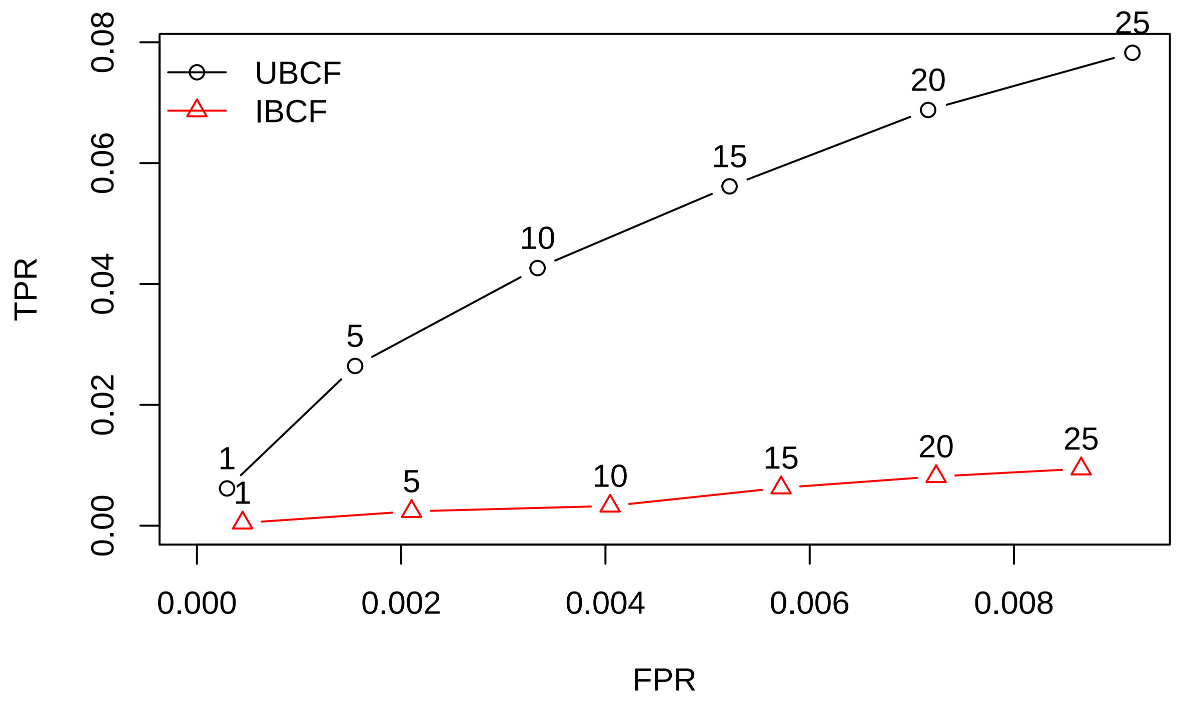
Final Assessment

Now that we have these two models we can form our comparison and deduce with our best parameter set, the UBCF model performs better on our test data set then the IBCF. The UBCF show better results in both the ROC curve and the precision recall curves. When we look at the RMSE, a measure of the accuracy of our predictions, we see that the UBCF has a RMSE of 1.008 while the IBCF has a RMSE of 1.389. Finally we do get some interesting results when we look at times to compile and predict in the model. The UBCF took almost no time to compile while the IBCF took almost a full minute. It appears that the UBCF give us the best accuracy at the cost of slower recommendations while the IBCF gives us slightly worse but faster performance.

```
final_models <- list(
  "UBCF" = list(name="UBCF", param=list(normalize = "z-score",
                                        method="pearson",
                                        nn=50)),
  "IBCF" = list(name="IBCF", param=list(normalize = "Z-score",
                                        method="pearson",
                                        k=40))
)
final_results <- evaluate(movie, final_models, n = recs, progress = FALSE)
```

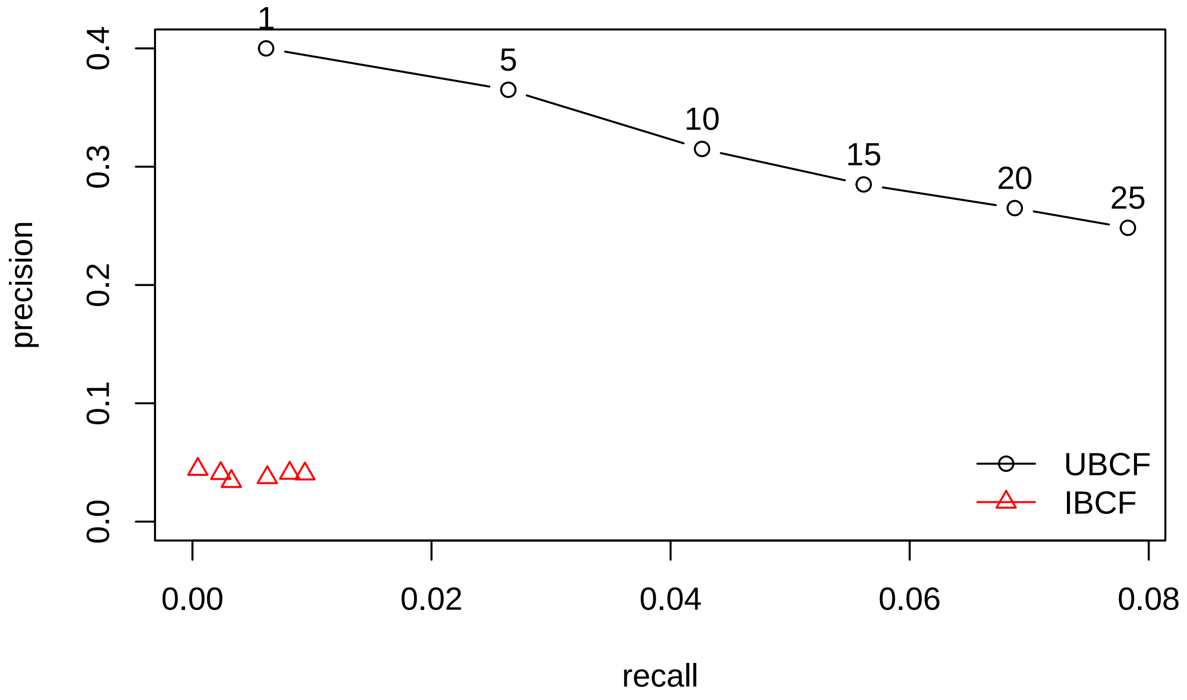
Drawing the ROC Curve

```
# Draw the ROC curve
plot(x = final_results, y = "ROC", annotate = c(1,2), legend="topleft")
```



Dawing the Precision / Recall Curve

```
# Draw the precision / recall curve
 plot(x = final_results, y = "prec/rec", annotate = c(1,3))
```



1
5
10
15
20
25
UBCF
IBCF
precision
recall
0.0
0.1
0.2
0.3
0.4
0.00
0.02
0.04
0.06
0.08