week2proj

Charley Ferrari

June 24, 2016

Evaluating Recommendation Algorithms.

I'll be looking at item based and user based collaborative filtering. Below are lists of the parameters I can tweak:

```
recommenderRegistry$get_entries(dataType =  "realRatingMatrix")$IBCF_realRatingMatrix
```

```
## Recommender method: IBCF
## Description: Recommender based on item-based collaborative filtering (real data).
## Parameters:
##    k method normalize normalize_sim_matrix alpha na_as_zero
## 1 30 Cosine    center                FALSE   0.5      FALSE
```

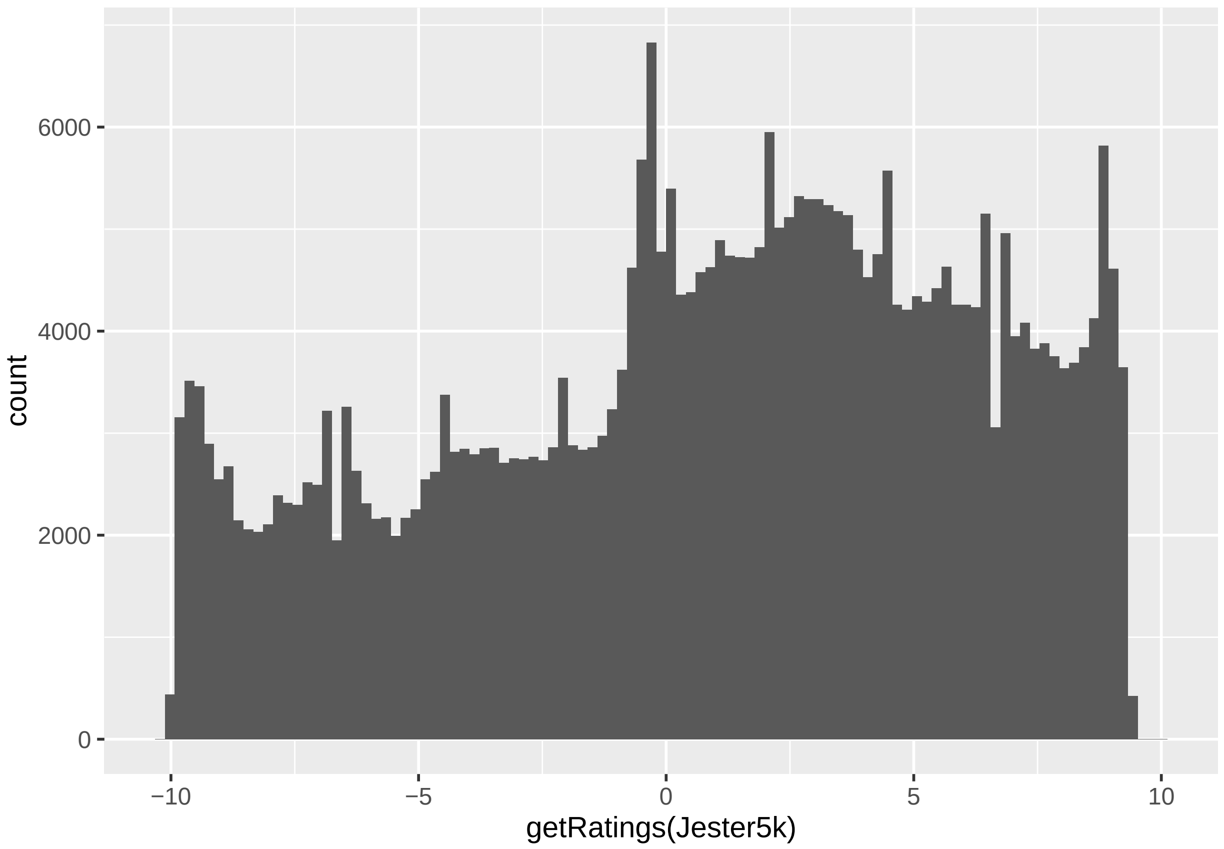
```
recommenderRegistry$get_entries(dataType =  "realRatingMatrix")$UBCF_realRatingMatrix
```

```
## Recommender method: UBCF
## Description: Recommender based on user-based collaborative filtering (real data).
## Parameters:
##   method nn sample normalize
## 1 cosine 25  FALSE    center
```

I'll tweak different variables one at a time, to get a sense for how they will affect my models. For IBCF, I'll tweak k, similarity method (either cosine or pearson), and the normalization method (center or z-score).

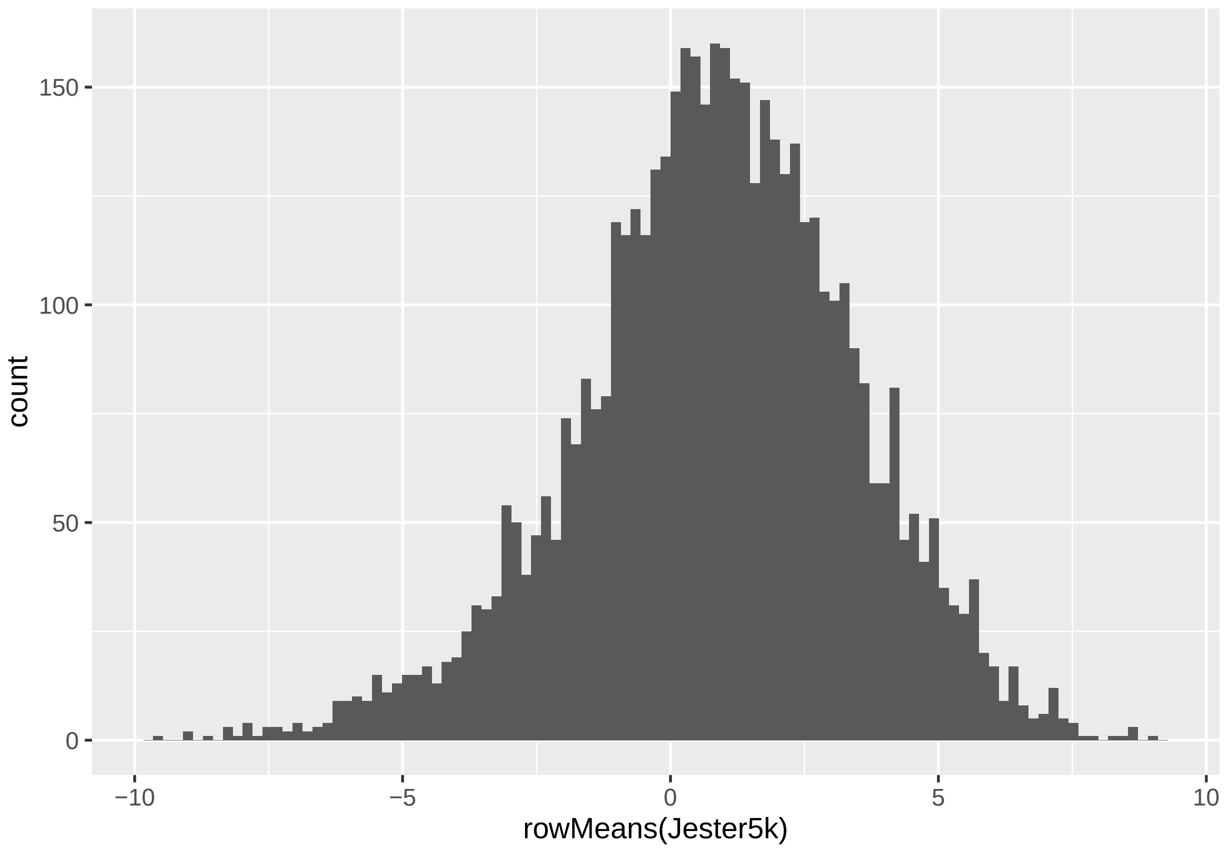
The ratings I'll be using is the built in dataset: Jester5k. Jester5k is a collection of joke ratings for 100 jokes by 5000 users, with 362106 ratings. First, lets look at a Histogram of the ratings:

```
qplot(getRatings(Jester5k), bins=100)
```



It looks like these ratings vary from -10 to 10. There is still an overall mean of 0.85, so centering the ratings will help normalize the ratings. Lets also look at the average ratings by user:

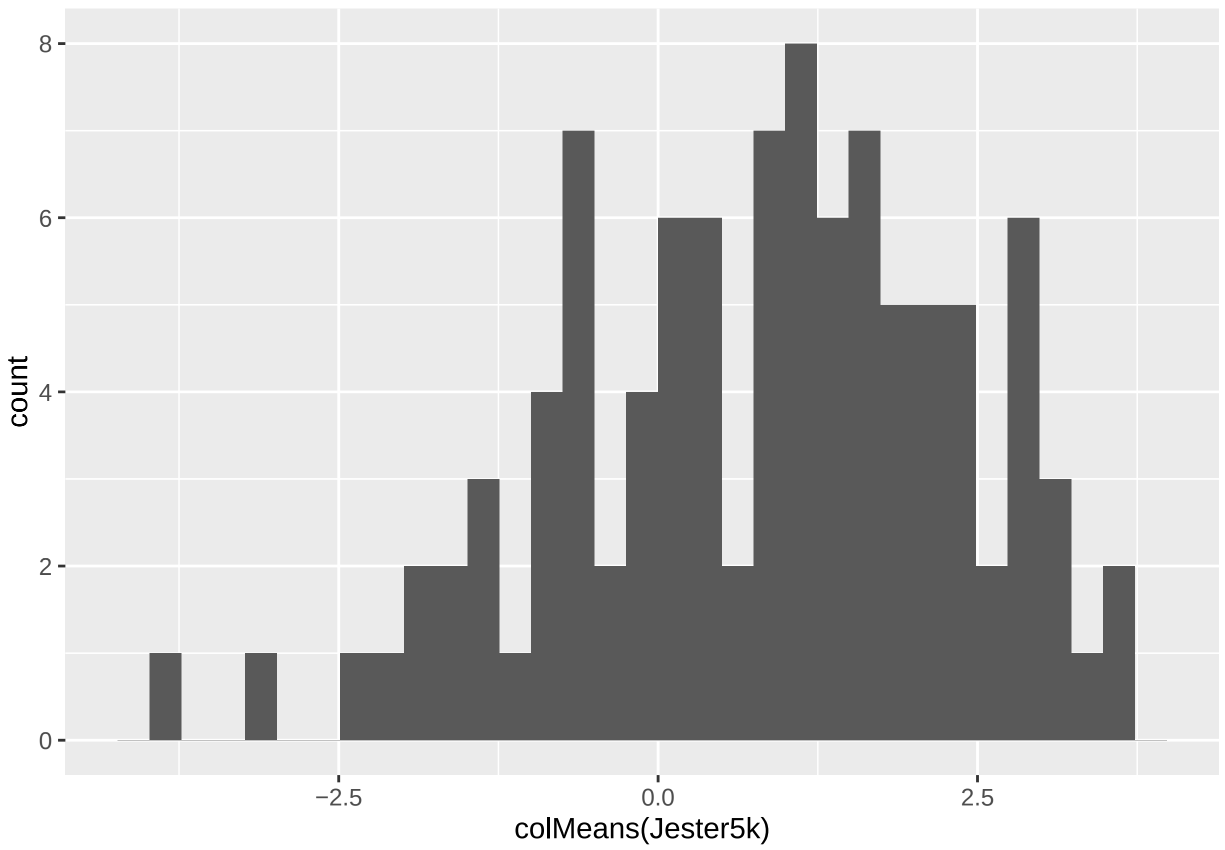
```
qplot(rowMeans(Jester5k), bins=100)
```



and the average rating by joke:

```r
qplot(colMeans(Jester5k))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

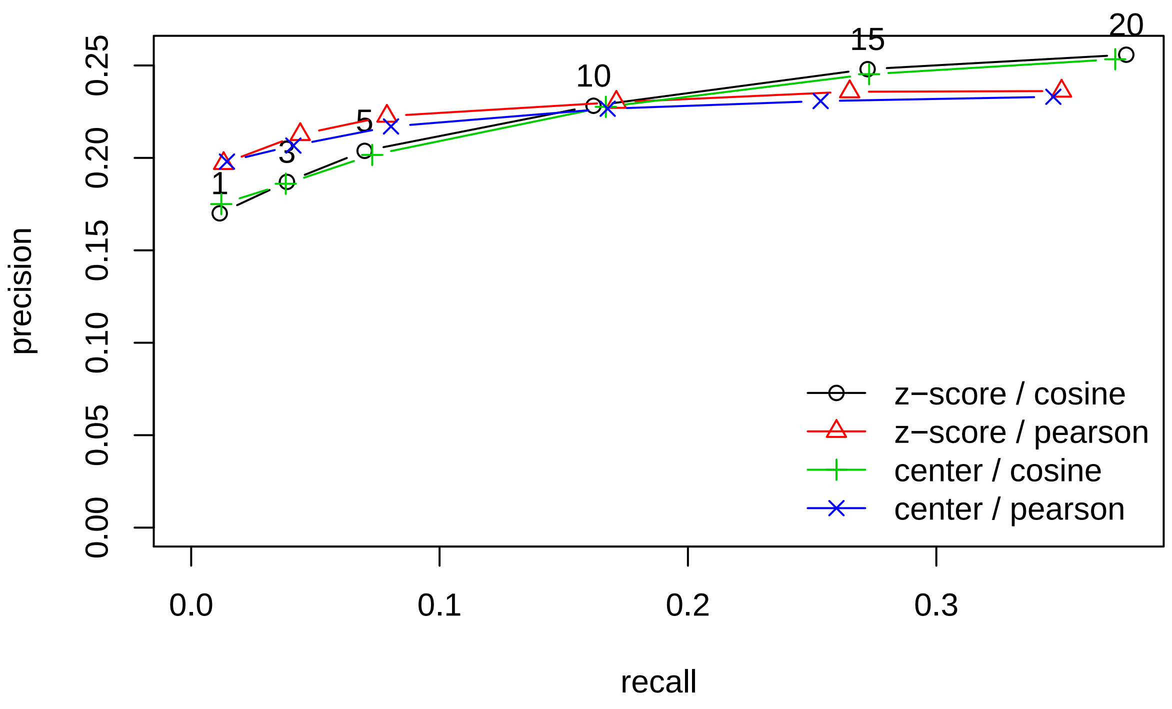


These both appear to be roughly normally distributed. As our first evaluation technique, we'll hold k constant at 30, and tweak our similarity methods and normalization techniques. Both of these have two choices we'll be evaluating: z-score and centered for normalization and Pearson and Cosine for the similarity measure.)

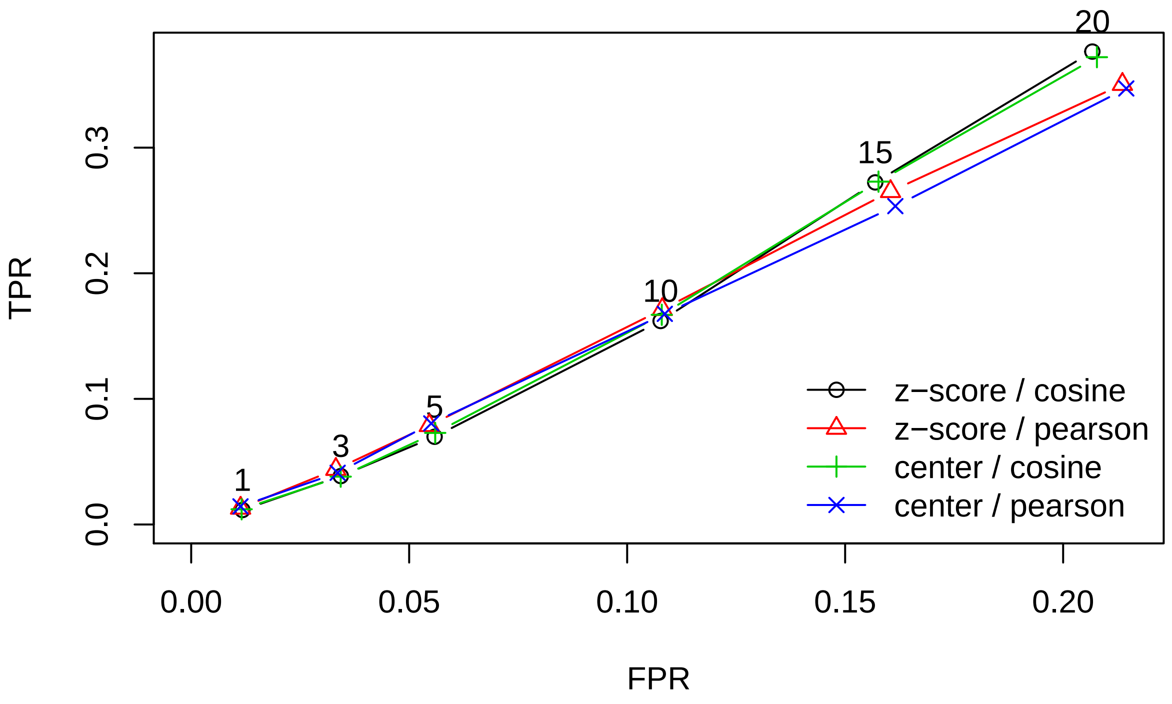
```r
algorithms <- list(
  "z-score / cosine" = list(name="IBCF", param=list(normalize = "Z-score")),
  "z-score / pearson" = list(name="IBCF", param=list(normalize = "Z-score",
                                                     method = "Pearson")),
  "center / cosine" = list(name="IBCF", param=list()),
  "center / pearson" = list(name="IBCF", param=list(method = "Pearson"))

)

scheme <- evaluationScheme(Jester5k, method = "split", train = .9,
                           k = 1, given = 20, goodRating = 4)

scheme <- evaluationScheme(Jester5k, method="split", train=0.8, given=15, goodRating=5)

results <- evaluate(scheme, algorithms, n=c(1, 3, 5, 10, 15, 20))
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.471sec/0.283sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.58sec/0.267sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.33sec/0.236sec]
## IBCF run fold/sample [model time/prediction time]
##    1  [0.472sec/0.237sec]
```

```r
plot(results, "prec/rec", annotate = TRUE, main = "Precision-recall")
```



```r
plot(results, annotate = 1, main = 'ROC Curve')
```



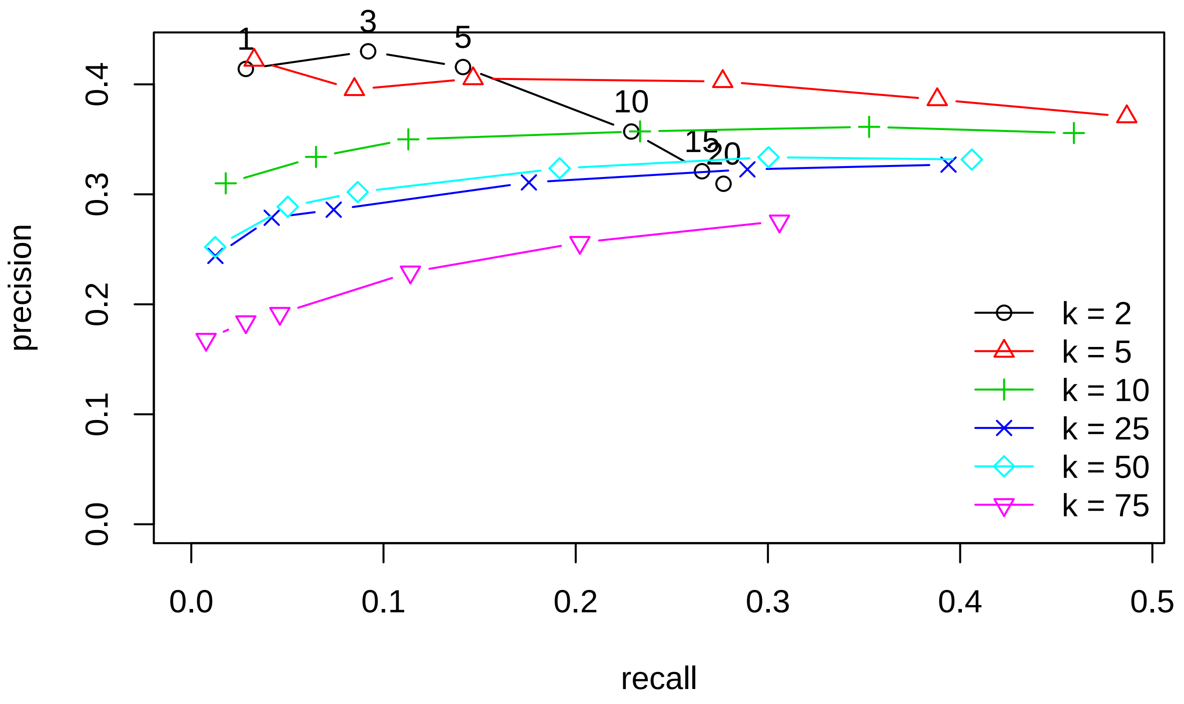
Our results look very similar, with cosine distance performing slightly better than pearson (and there being little effect with z-score versus center.)

Next, we could look at different values of k and see how varying the number of nearest neighbors affects things.

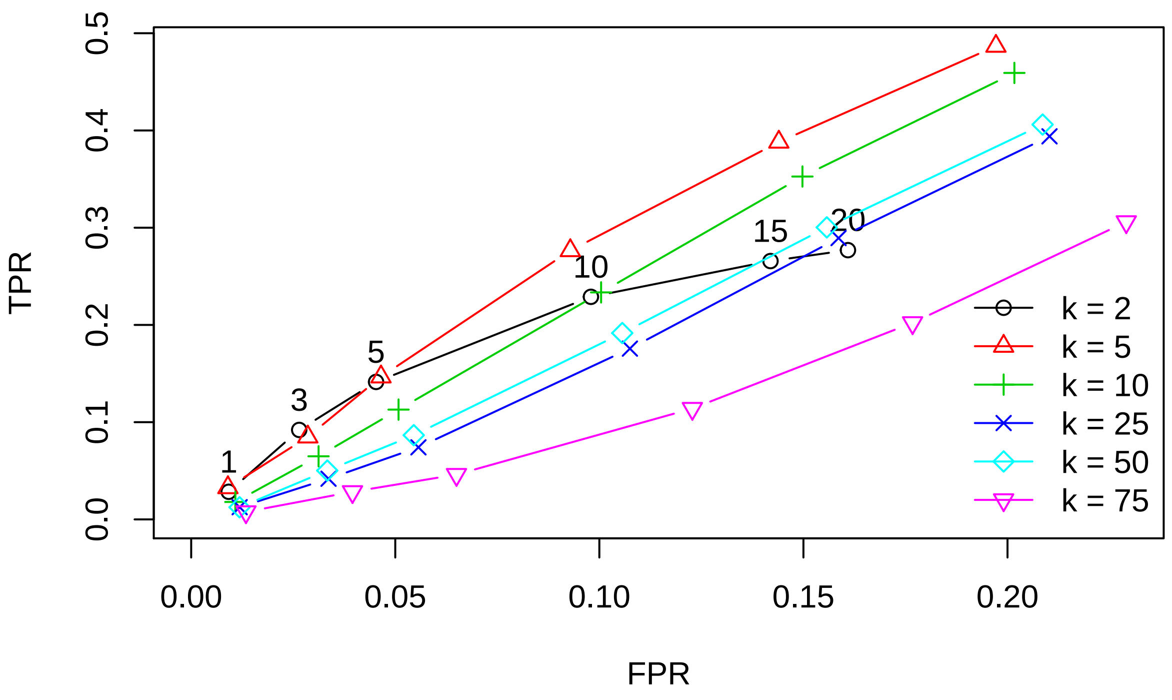
```
algorithms <- list(
  "k = 2" = list(name="IBCF", param=list(k = 1)),
  "k = 5" = list(name="IBCF", param=list(k = 5)),
  "k = 10" = list(name="IBCF", param=list(k = 10)),
  "k = 25" = list(name="IBCF", param=list(k = 25)),
  "k = 50" = list(name="IBCF", param=list(k = 50)),
  "k = 75" = list(name="IBCF", param=list(k=75))

)

scheme <- evaluationScheme(Jester5k, method = "split", train = .9,
                           k = 1, given = 20, goodRating = 4)

results <- evaluate(scheme, algorithms, n=c(1, 3, 5, 10, 15, 20))
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.496sec/0.081sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.378sec/0.105sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.381sec/0.106sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.496sec/0.114sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.374sec/0.113sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.378sec/0.104sec]
```

```
plot(results, "prec/rec", annotate = TRUE, main = "Precision-recall")
```



```r
plot(results, annotate = 1, main = 'ROC Curve')
```



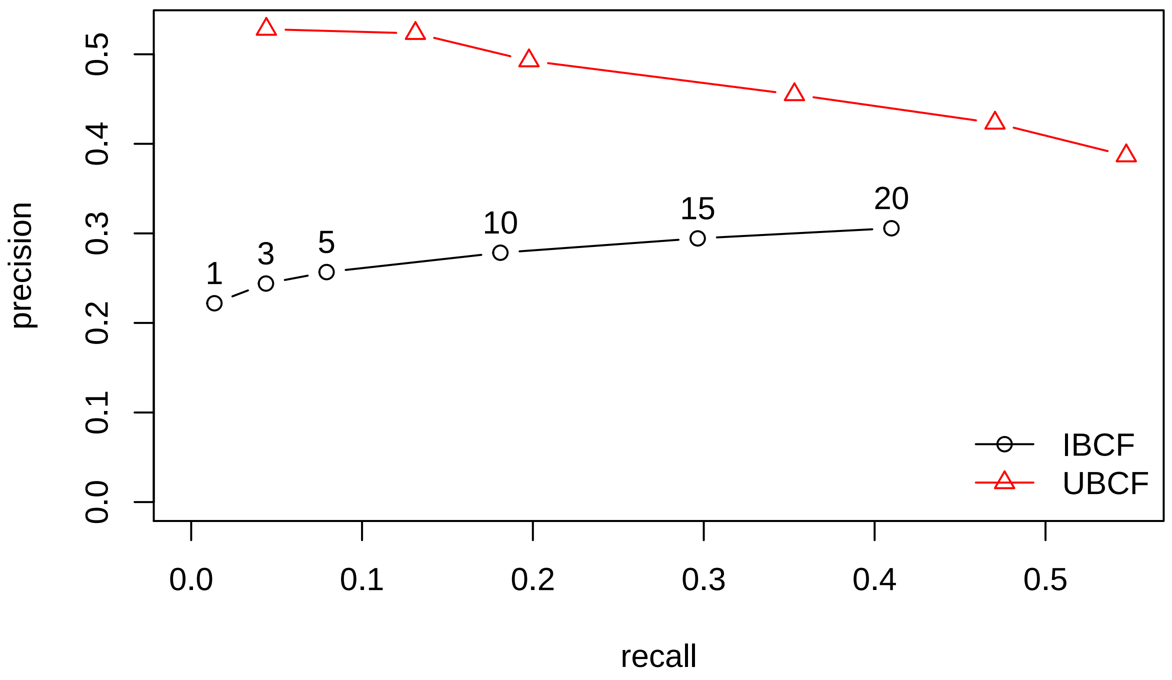
Performance seems to go down as k increases. K = 2 seems to offer worse results as the number of recommended movies goes up, which might indicate higher variability when using a smaller number of neighbors. This makes intuitive sense, as a smaller number of neighbors may result in more chances for error. It's unclear why k = 25 seems to be worse than k = 50…

Lastly, we could compare IBCF and UBCF.

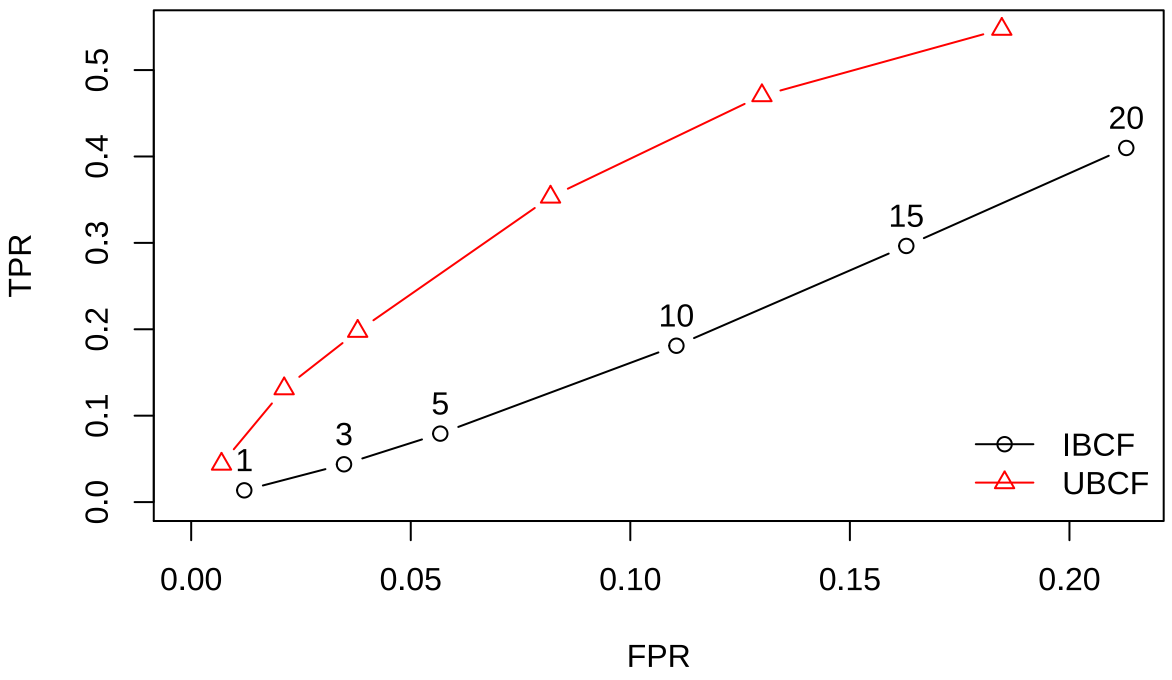
```r
algorithms <- list(
  IBCF = list(name="IBCF", param=list()),
  UBCF = list(name="UBCF", param=list())

)

scheme <- evaluationScheme(Jester5k, method = "split", train = .9,
                           k = 1, given = 20, goodRating = 4)

results <- evaluate(scheme, algorithms, n=c(1, 3, 5, 10, 15, 20))
```

```
## IBCF run fold/sample [model time/prediction time]
##    1  [0.37sec/0.123sec]
## UBCF run fold/sample [model time/prediction time]
##    1  [0.043sec/3.897sec]
```

```r
plot(results, "prec/rec", annotate = TRUE, main = "Precision-recall")
```



```
plot(results, annotate = 1, main = 'ROC Curve')
```



The user based collaborative method seems to perform better than the item based method.

This page which gave me some of the inspiration for the code in this project, explains the trade off in using IBCF versus UBCF. Apparently UBCF is a more computationally intensive method, while IBCF only calculates based on the k closest individuals (and not across the entire matrix.)