Project4_Data-612- Accuracy & Beyond

Samriti Malhotra, Vishal Arora

July 02, 2019

Contents

Objective :-

The goal of this assignment is give you practice working with Matrix Factorization techniques. The task is implement a matrix factorization method-such as singular value decomposition (SVD) or Alternating Least Squares (ALS)-in the context of a recommender system.

Solution:-

We took this dataset ml-latest-small.zip from Movie Lens site which describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018.

> Citation :- F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1-19:19. https://doi.org/10.1145/2827872

Libraries used

recommenderlab
dplyr
reshape2

Data loading , preperation of relevant dataset

Data is loaded from the github, and then selecting the columns to create a matrix which is a class of realRatingMatrix. As our matrix doesn't have any NA that means every user has seen every movie and provided ratings but all of them may not be relevant.

```
ratings <- read.csv("https://raw.githubusercontent.com/samriti0202/DATA612-RecommenderSystems/master/Pr
titles <- read.csv("https://raw.githubusercontent.com/samriti0202/DATA612-RecommenderSystems/master/Pro

ratings <- ratings %>% select(userId, movieId, rating)

#converting the ratings data frame into userId-movieId matrix
ratingDT <- acast(ratings, userId~movieId, value.var="rating")

#convert matrix into realRatingMatrix using recommenderLab package
ratingDT <- as(as.matrix(ratingDT), "realRatingMatrix")
dim(ratingDT)
```

```
## [1]  610 9724
```

Data Preperation

1) Select the relevant data

2) Normalize the data

As rule of thumb for beginning user who rating more than 100 movies and movies which have been watched more than 100 time. those are the ones we going to take initially.

```
ratings_movies <- ratingDT[rowCounts(ratingDT)>100, colCounts(ratingDT)>100]

dim(ratings_movies)
```

```
## [1] 245 134
```

Now the dataset has reduced but still it is a large dataset may be we might have to take a smaller dataset for SVD evluation.Lets first do the evaulation using IBCF & UBCF algorithms and compare it with the SVD to see which one has the least RMSE.

Building the Item-based Collaborative Filtering Model (IBCF) and RMSE for IBCF model.

Taking a subset of the relevant dataset ,as the memory imprint was too high and iyt was taking time to build the recommender model.

```
rating_movies <- as(ratings_movies, "realRatingMatrix")
rm()
set.seed(88)
eval_sets <- evaluationScheme(data = rating_movies, method = "split", train = 0.8, given = -1, goodRati

#IBCF
eval_recommender_ibcf <- Recommender(data = getData(eval_sets, "train"), method = "IBCF", parameter = NU
eval_prediction_ibcf <- predict(object = eval_recommender_ibcf, newdata = getData(eval_sets, "known"), r
calcPredictionAccuracy(x = eval_prediction_ibcf, data = getData(eval_sets, "unknown"), byUser = FALSE)
```

```
##      RMSE       MSE       MAE
## 0.8860376 0.7850626 0.6901797
```

Building the User-based Collaborative Filtering Model (UBCF) and then evluate the RMSE for UBCF model

```
#IBCF
eval_recommender_ubcf <- Recommender(data = getData(eval_sets, "train"), method = "UBCF", parameter = NU
eval_prediction_ubcf <- predict(object = eval_recommender_ubcf, newdata = getData(eval_sets, "known"), r
calcPredictionAccuracy(x = eval_prediction_ubcf, data = getData(eval_sets, "unknown"), byUser = FALSE)
```

```
##      RMSE       MSE       MAE
## 0.8047560 0.6476322 0.6497898
```

Building SVD model

```
svdModel <- Recommender(getData(eval_sets, "train"), method = "SVD", parameter = list(k = 50))
svdPredModel <- predict(svdModel, newdata = getData(eval_sets, "known"), type = "ratings")


calcPredictionAccuracy(x=svdPredModel, getData(eval_sets, "unknown"), byUser = FALSE)
```

```
##      RMSE       MSE       MAE
## 0.7996445 0.6394314 0.6375262
```

Summary

From the above RMSE and other values for various models algorithms we can clearly that SVD is slightly better than UBCF and which in turn is better than IBCF. We can evaluate the svd model by manually calculating SVD(using Base R package) and also SVD can be performed step-by-step with R by calculating ATA and AAT then finding the eigenvalues and eigenvectors of the matrices. However, results can be slightly different than the output of the svd()/recommenderLab . There is a nice article on this (SVD Article Aaron)[https://rpubs.com/aaronsc32/singular-value-decomposition-r].

Project 4 Objective a):- is to evaluate various recommendation system and compare Accuracy of 2 systems.

###Evluating Recommender System(s).
In order to evaluate different models, we will define a list of various recommender systems and then evaluate them to plot ROC cure and see which has the largest AUC. For tyhis the starting point will be k-fold evaluation framework, we will use k-fold to create a new data set.

```
n_fold <- 4
items_to_keep <- min(rowCounts(rating_movies))
items_to_keep <- items_to_keep -2
rating_threshold <- 3
eval_sets <- evaluationScheme(data=rating_movies,method="cross-validation",k=n_fold,given=items_to_keep

models_to_evaluate <- list(
  IBCF_cos = list(name="IBCF",param=list(method="cosine")),
  IBCF_cor = list(name="IBCF",param=list(method="pearson")),
  IBCF_norctr = list(name = "IBCF", param = list(normalize = "center",method = "cosine")),
  IBCF_norzscore_cos = list(name = "IBCF", param = list(normalize = "Z-score",method = "cosine")),
  IBCF_norctr_cor = list(name = "IBCF", param = list(normalize = "center",method = "pearson")),
  IBCF_norzscore_cor = list(name = "IBCF", param = list(normalize = "Z-score",method = "pearson")),
  UBCF_cos = list(name="UBCF",param=list(method="cosine")),
  UBCF_cor = list(name="UBCF",param=list(method="pearson")),
  UBCF_norctr_cos = list(name = "UBCF", param = list(normalize = "center",method = "cosine")),
  UBCF_norzscore_cos = list(name = "UBCF", param = list(normalize = "Z-score",method = "cosine")),
  UBCF_norctr_cor = list(name = "UBCF", param = list(normalize = "center",method = "pearson")),
  UBCF_norzscore_cor = list(name = "UBCF", param = list(normalize = "Z-score",method = "pearson")),
  random = list(name="RANDOM",param=NULL)
  )
```

In order to evaluate trhe models properly, we need to test them varying the number of items and use evaluate function to evaluate the list ot recommender system,.

```
n_recommendations <- c(1,5,seq(10,100,10))


list_results <- evaluate(x=eval_sets,method=models_to_evaluate,n=n_recommendations)
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.22sec/0.02sec]
##   2  [0.03sec/0.05sec]
##   3  [0.02sec/0.01sec]
##   4  [0.03sec/0.02sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.04sec/0.02sec]
##   2  [0.04sec/0.02sec]
##   3  [0.04sec/0.02sec]
##   4  [0.03sec/0.02sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.02sec/0.03sec]
##   2  [0.01sec/0.02sec]
##   3  [0.03sec/0sec]
##   4  [0.03sec/0.02sec]
```

```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.05sec/0.01sec]
##   2  [0.04sec/0.04sec]
##   3  [0.05sec/0.01sec]
##   4  [0.03sec/0.02sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.05sec/0.02sec]
##   2  [0.04sec/0.04sec]
##   3  [0.08sec/0.01sec]
##   4  [0.07sec/0.03sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.09sec/0.03sec]
##   2  [0.06sec/0.03sec]
##   3  [0.08sec/0.03sec]
##   4  [0.06sec/0.02sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/0.06sec]
##   2  [0sec/0.05sec]
##   3  [0.02sec/0.1sec]
##   4  [0sec/0.07sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/0.09sec]
##   2  [0sec/0.08sec]
##   3  [0sec/0.08sec]
##   4  [0sec/0.07sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0sec/0.05sec]
##   2  [0sec/0.03sec]
##   3  [0sec/0.04sec]
##   4  [0sec/0.03sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.01sec/0.05sec]
##   2  [0.01sec/0.05sec]
##   3  [0sec/0.04sec]
##   4  [0.02sec/0.03sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.02sec/0.03sec]
##   2  [0sec/0.04sec]
##   3  [0sec/0.03sec]
##   4  [0sec/0.03sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.01sec/0.05sec]
##   2  [0sec/0.05sec]
##   3  [0.01sec/0.03sec]
##   4  [0.02sec/0.05sec]
## RANDOM run fold/sample [model time/prediction time]
##   1  [0sec/0.01sec]
##   2  [0sec/0.02sec]
##   3  [0sec/0.01sec]
##   4  [0sec/0.01sec]
```

The list_results objects is of evaluationResultList object containing individual evaluationResults object, we will verify the same and then take an average confusion matrices using avg function.

```
sapply(list_results,class)=="evaluationResults"
```

```
##           IBCF_cos           IBCF_cor        IBCF_norctr
##               TRUE               TRUE               TRUE
## IBCF_norzscore_cos    IBCF_norctr_cor IBCF_norzscore_cor
##               TRUE               TRUE               TRUE
##           UBCF_cos           UBCF_cor    UBCF_norctr_cos
##               TRUE               TRUE               TRUE
## UBCF_norzscore_cos    UBCF_norctr_cor UBCF_norzscore_cor
##               TRUE               TRUE               TRUE
##             random
##               TRUE
```

```
#taking a average of confusion matrices
avg_matrices <- lapply(list_results,avg)

knitr::kable(head(avg_matrices[[1]][,5:8]), format = "html") %>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

precision

recall

TPR

FPR

1

0.3392846

0.0074802

0.0074802

0.0077116

5

0.3331464

0.0353770

0.0353770

0.0406671

10

0.3373370

0.0728450

0.0728450

0.0802643

20

0.3423613

0.1464633

0.1464633

0.1579789

30

0.3543894

0.2233081

0.2233081

0.2290107

40

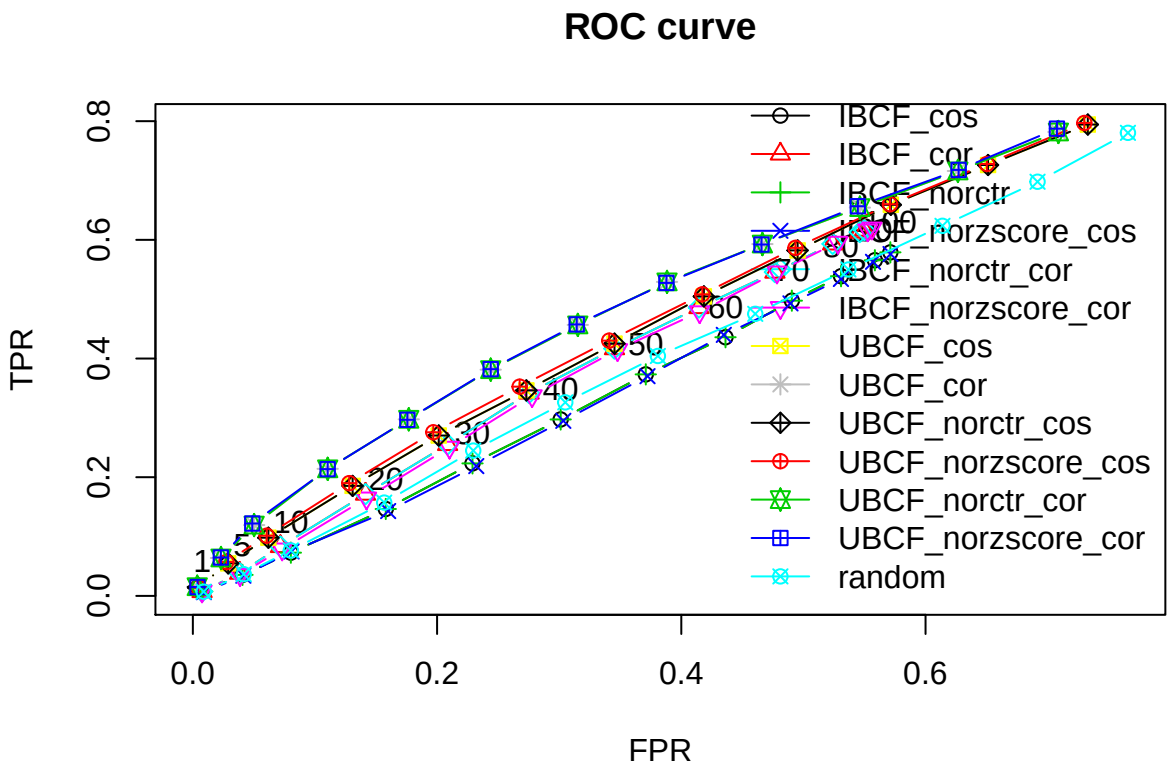
0.3569637

0.2973527

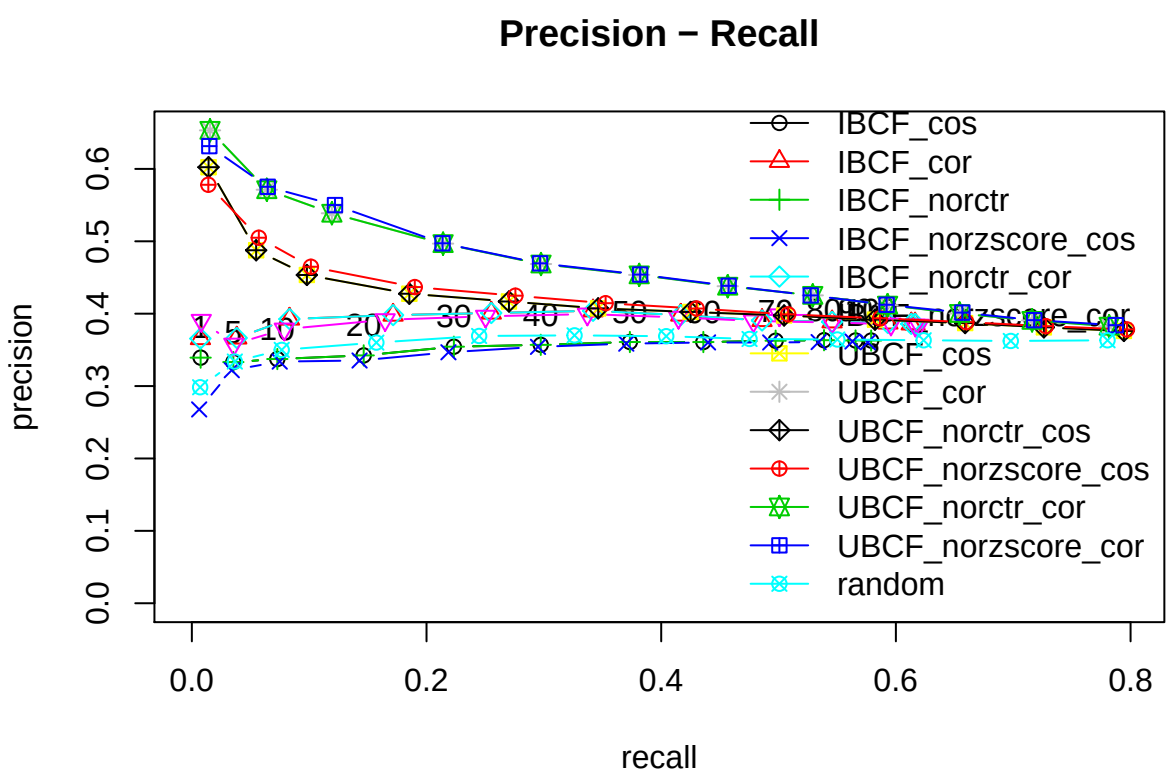
0.2973527

0.3011984

```
#plotting for ROC curve
plot(list_results, annotate=1,legend="bottomright")
title("ROC curve")
```




```
plot(list_results,"prec/rec",annotate=1,legend="bottomright")
title("Precision - Recall")
```



A good performance index is the AUC(Area under the Curve), this is the area under the ROC curve. We can clearly see that in this case UBCF with nornmalize =(center & z-score)with pearson distance is the best performance technique.

Showing Accuracy for 2 recommnder system (i.e. UBCF Normalized z-score with Pearson distance & UBCF normalized with center with pearson distance)

Accuracy = (TP+TN)/(TP+FP+FN+TN)

```
ubcf_zscore_cor <- avg_matrices$UBCF_norzscore_cos[,1:8]
ubcf_cen_cor <- avg_matrices$UBCF_norctr_cos[,1:8]

ubcf_zscore_cor = as.data.frame(getConfusionMatrix(list_results[["UBCF_norzscore_cor"]])[[1]][,1:8])
ubcf_cen_cor = as.data.frame(getConfusionMatrix(list_results[["UBCF_norctr_cor"]])[[1]][,1:8])



ubcf_zscore_cor$Accuracy = (ubcf_zscore_cor$TP + ubcf_zscore_cor$TN)/(ubcf_zscore_cor$TP+ubcf_zscore_co

ubcf_cen_cor$Accuracy = (ubcf_cen_cor$TP + ubcf_cen_cor$TN)/(ubcf_cen_cor$TP+ubcf_cen_cor$FP+ubcf_cen_c



knitr::kable(head(ubcf_zscore_cor$Accuracy))%>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

x
0.6729529
0.6746898
0.6763027
0.6624069
0.6470223
0.6266749

```
knitr::kable(head(ubcf_cen_cor$Accuracy))%>%
  kableExtra::kable_styling(bootstrap_options = c("striped", "hover"))
```

x
0.6739454
0.6746898
0.6748139
0.6621588
0.6445409
0.6261787

We can clearly see that the accuracy for both UBCF models using normalized (center & z-score) using pearson distance have nearly same accuracy and hence as shown through the ROC plot it is clear that both models are best models with best accuracy .

Project 4 Objective b):- Increasing Diversity.

To increase user experience, expand user preferences and also to gather more information about a user, we construct a hybrid model i.e. a mixture of 2 or more models (with high accuracy and low accuracy) so that the end result is mixture of both and we ddon't always end up recommending user always a highly rated product as it can create a bias. Thus diveristy is introduced while building hybrid models, in our case we will have(0.99 vs. 0.01 weight between UBCF and Random models) and construct a hybrid model.

```
model_method <- "UBCF"
model_parameters <- NULL
# Training
modelUBCF <- Recommender(data=getData(eval_sets,"train"), method = model_method,parameter=model_paramet

predUBCF <- predict(modelUBCF,getData(eval_sets,"known"),type="ratings")


model_method <- "RANDOM"
randomModel <- Recommender(data=getData(eval_sets,"train"), method = model_method,parameter=model_param
predRandom <- predict(randomModel,getData(eval_sets,"known"),type="ratings")


hybridModel <- HybridRecommender(
    modelUBCF,
    randomModel,
    weights = c(0.99, 0.01))
predHybrid <- predict(hybridModel, newdata = getData(eval_sets,"known"), type = "ratings")
```

```
#Error in UBCF
( accUBCF <- calcPredictionAccuracy(predUBCF, getData(eval_sets,"unknown")) )
```

```
##      RMSE       MSE       MAE
## 0.9277180 0.8606606 0.7110595
```

```
# Error in Random
( accRandom <- calcPredictionAccuracy(predRandom, getData(eval_sets,"unknown")) )
```

```
##     RMSE      MSE      MAE
## 1.251103 1.565259 0.958571
```

```
#Error in hybrid approach
( accHybrid <- calcPredictionAccuracy(predHybrid, getData(eval_sets,"unknown")) )
```

```
##     RMSE      MSE      MAE
## 1.171745 1.372986 0.822340
```

We can clearly see that model accuracy in hybrid is not too high like in UBCF neither too low like in Random but is somewhat in middle more titled towards UBCF.But it introduces more slippage in accuracy meaning more random things will be recommened to user.

Project Objective c):- online evaluation

Uptil now we have been using offline data and all simulatons and algorithm used were to evaluate offline accuracy.

Offline evaluations :-

Offline evalutaion typically measure the accuracy of a recommender system based on a ground-truth. To measure accuracy, precision at position n (P@n) is often used to express how many items of the ground-truth are recommended within the top n recommendations. Other common evaluation metrics include recall, F-measure, mean reciprocal rank (MRR), normalized discounted cumulative gain (nDCG), mean absolute error, and root mean square error. Offline evaluations are also sometimes used to evaluate aspects such as novelty or serendipity of recommendations.

Online Evaluations:-

Online evaluations measure the acceptance rates of recommendations in real-world recommender systems. Acceptance rates are typically measured by click-through rate (CTR), i.e. the ratio of clicked recommendations to displayed recommendations. For instance, if a recommender system displays 10,000 recommendations and 120 are clicked, CTR is 1.2%. Other metrics include the ratio of downloaded or bought items. Acceptance rate is typically interpreted as an implicit measure for user satisfaction. The assumption is that when a user clicks, downloads, or buys a recommended item, the user liked the recommendation.

For online evaluation we can use a database (SPARK or NoSQL) to store the online transactional data and create some kind of computational system to run and update the algorithms in real time, and design a GUI to present the choices and recommendations to the end-user.