Project1 (IS643 - Recommender Systems)

Tulasi Ramarao

6/6/2017

Description

This system recommends restaurants to customers. This is a simple recommender system constructed to recommend restaurants using custom algorithms (based on why user experience matters)

DataSet

The dataset for restaurant customer ratings called 'rating_final.csv' is downloaded from the website - https://archive.ics.uci.edu/ml/datasets/Restaurant+%26+consumer+data.

This dataset has customer ratings for several restaurants. There are 1161 rows of ratings rated by 138 customers (userID) for 130 restaurants. The ratings are on the scale of 0-2.

```r
# Set the working directory
setwd("/Users/tulasiramarao/Documents/Tulasi/CUNYProjects/DATA643/RPrograms")

# Read the restaurant data from the csv downloaded from the website mentioned above
origDF <- read.csv("RCData/rating_final.csv", header = TRUE, sep = ",",stringsAsFactors = FALSE, encodi
head(origDF,2)
```

```
##   userID placeID rating food_rating service_rating
## 1  U1077  135085      2           2              2
## 2  U1077  135038      2           2              1
```

```r
# Take only the first 3 relevant columns
origDF <- subset(origDF,select=c(userID,placeID,rating))
```

The dataset has several missing values(NA) and so, a dense 'user-item' dataframe as a subset is chosen so that the calculations can be verified by hand. This subset it chosen based on a selection of 5 random users and a few restaurants they rated. Restaurant IDs chosen: 135032,135038,135046,135081,135085
userIDs chosen:1108, 1088,1081,1053, 1077

```r
# select a set of users(userIDs) and get their data
filteredDF <- origDF[origDF$userID =='U1077' |origDF$userID == "U1108"|origDF$userID == "U1053"| origDF$

head(filteredDF)
```

```
##    userID placeID rating
## 1   U1077  135085      2
## 2   U1077  135038      2
## 3   U1077  132825      2
## 4   U1077  135060      1
## 65  U1108  135075      2
## 66  U1108  132572      1
```

```r
# select a set of restaurants(placeID) and get that data
rRatingsOrig <- filteredDF[filteredDF$placeID =='135085' |filteredDF$placeID =='135038'|filteredDF$plac
head(rRatingsOrig)
```

```
##     userID placeID rating
## 1    U1077  135085      2
## 2    U1077  135038      2
## 69   U1108  135085      1
## 70   U1108  135032      2
## 73   U1108  135046      1
## 185  U1053  135032      2
```

```
# Now we have a small subset. Lets look at the dimensions
dim(rRatingsOrig)
```

```
## [1] 12  3
```

```
# Make a backup copy
rRatings <- rRatingsOrig

# Rename the column names for ease of use
colnames(rRatings)<-c("CustomerID","RestaurantID","Ratings")
head(rRatings)
```

```
##     CustomerID RestaurantID Ratings
## 1        U1077       135085       2
## 2        U1077       135038       2
## 69       U1108       135085       1
## 70       U1108       135032       2
## 73       U1108       135046       1
## 185      U1053       135032       2
```

```
# Add extra data for a better understanding of this project
extrarows<-data.frame(c("U1081","U1081"),c(135046,135081),c(1,1))
colnames(extrarows)<-c("CustomerID","RestaurantID","Ratings")
extrarows
```

```
##   CustomerID RestaurantID Ratings
## 1      U1081       135046       1
## 2      U1081       135081       1
```

```
# Append these extra rows to the orginal data
rRatings <- rbind(rRatings,extrarows);

# Reshape the dataframe so that the customerIDs are in the rows and restaurantIDs are in the column
library(reshape2)
rRatings.reshaped=dcast(rRatings, CustomerID ~ RestaurantID,  value.var = "Ratings")
head(rRatings.reshaped)
```

```
##   CustomerID 135032 135038 135046 135081 135085
## 1      U1053      2      2     NA      2     NA
## 2      U1077     NA      2     NA     NA      2
## 3      U1081      0     NA      1      1      1
## 4      U1088      2     NA     NA      1     NA
## 5      U1108      2     NA      1     NA      1
```

```
# Make a backup copy of the dataframe
mydf <-rRatings.reshaped
head(mydf)
```

```
##   CustomerID 135032 135038 135046 135081 135085
## 1      U1053      2      2     NA      2     NA
## 2      U1077     NA      2     NA     NA      2
## 3      U1081      0     NA      1      1      1
## 4      U1088      2     NA     NA      1     NA
## 5      U1108      2     NA      1     NA      1
```

```
#  Create train and test sets manually
# Splitting the training and test sets automatically eliminated some users and restaurants.
# We need both train and set to have all users and all restaurants to better understand this project.
target.df <- mydf[,c(2:6)]
n =nrow(target.df)
m = ncol(target.df)
dim(target.df)
```

```
## [1] 5 5
```

```
# create an empty dataframe
mytest <- data.frame(matrix(NA,nrow=5,ncol=5))

#populate with select values for training
for(n in 1:n){
  for(m in 1:m){
    if(m == 1 & n == 1){
       mytest[n,m] <- target.df[n,m]
    }else if(m == 2 & n == 2){
       mytest[n,m] <- target.df[n,m]
    }else if(m == 5 & n == 3){
       mytest[n,m] <- target.df[n,m]
   }else if(m == 4 & n == 4){
       mytest[n,m] <- target.df[n,m]
   }else if(m == 3 & n == 5){
       mytest[n,m] <- target.df[n,m]
   }
  }
}

# copy the original df and NA the test data spots
mytrain <- target.df
#populate NAs for the indexes where test data exists
mytrain[1,1] <- NA
mytrain[2,2] <- NA
mytrain[5,3] <- NA
mytrain[4,4] <- NA
mytrain[3,5] <- NA
```

```
# Now assign mytrain and mytest dataframes to train.in and test.in
train.in <- mytrain
test.in <- mytest

# how many missing ratings(NAs) for each restaurant
apply(is.na(train.in),2,sum)
```

```
## 135032 135038 135046 135081 135085
##      2      4      4      3      3
```

```
# Now calculate the raw average (mean) rating for every user-item combination
avgdf <- train.in
```

```
avgdf$avg = apply(train.in,1,mean,na.rm = T)
# round to 2 digits
rawAvgTrain <- round(mean(avgdf$avg,na.rm = TRUE),2)
rawAvgTrain
```

```
## [1] 1.63
```

Now Root Mean Squared Error(RMSE) for raw average for both training and test datasets are calculated.

```
# RMSE for Train dataset
rmseDFTrain <- sqrt( mean( (train.in-rawAvgTrain)^2 , na.rm = TRUE ) )
rmseDFTrain
```

```
## [1] 0.7096243
```

```
# Similarly RMSE for the test dataset
rmseDFTest <- sqrt( mean( (test.in-rawAvgTrain)^2 , na.rm = TRUE ) )
rmseDFTest
```

```
## [1] 0.5412024
```

Using the training data, the bias for each customer and each restaurant is calculated. Test data is not used because that's the data on which RMSE will be performed.

(solving optimization problem is one way, but a simpler approach is used here). So, comparing each value to the mean value shows if the value is higher or lower than the mean.

```
# Make two copies of the original dataframe - one for restaurant bias and one for the customer bias
eachBiasUser <- train.in
eachBiasRestaurant <- train.in

# Calculate user bias first
# Calculate the mean for each row
vec1 = apply(eachBiasUser,1,mean,na.rm = T)
# Subtract each mean value with the raw average of the dataset calculated earlier - 1.5
eachUserBias <- vec1 - rawAvgTrain

# Calculate restaurant bias next
# Calculate the mean for each column
vec2 = apply(eachBiasRestaurant,2,mean,na.rm = T)

# Subtract each mean value with the raw average of the dataset calculated earlier - 1.5
eachRestaurantBias <- vec2 - rawAvgTrain
```

Now that we have the bias values and also the raw average, the baseline predictors for every user-item combination is calculated next.

```
library(xtable)
# convert bias vectors to dataframes for ease of use
userbias.df<- data.frame(eachUserBias)
restaurantbias.df <- data.frame(eachRestaurantBias)

print(xtable(userbias.df, digits =2, type = "html",caption="User bias"),comment=FALSE)
```

```
print(xtable(restaurantbias.df, digits =2, type = "html",caption="Restaurant bias"),comment=FALSE)
```

```
# Baseline predictor
baseline.pred <- mydf
```

	eachUserBias
1	0.37
2	0.37
3	-0.96
4	0.37
5	-0.13

Table 1: User bias

	eachRestaurantBias
135032	-0.30
135038	0.37
135046	-0.63
135081	-0.13
135085	-0.13

Table 2: Restaurant bias

```r
i=nrow(userbias.df)
j=nrow(restaurantbias.df)


mydf.baselinePred <- mydf[2:6]

#str(mydf.baselinePred)
# Fill the missing values with raw average
mydf.baselinePred[is.na(mydf.baselinePred)] <- rawAvgTrain
# Replace NaNs
mydf.baselinePred <- replace(mydf.baselinePred, is.na(mydf.baselinePred), rawAvgTrain)

# Now calculate baseline for all the values in the dataframe for user items.
for(i in 1:i){
  for (j in 1:j){
    mydf.baselinePred[i,j] = rawAvgTrain +  (userbias.df[i,] + restaurantbias.df[j,])
  }
}

#test
#x <- userbias.df[3,]  + rawAvgTrain + restaurantbias.df[3,]
#x

print(xtable(mydf.baselinePred, digits =2, type = "html",caption="Baseline Predictor"),comment=FALSE)
```

	135032	135038	135046	135081	135085
1	1.70	2.37	1.37	1.87	1.87
2	1.70	2.37	1.37	1.87	1.87
3	0.37	1.04	0.04	0.54	0.54
4	1.70	2.37	1.37	1.87	1.87
5	1.20	1.87	0.87	1.37	1.37

Table 3: Baseline Predictor

Now calculate the RMSE for baseline predictors for both training and test data.

```r
# Split the baseline predictor df into training and test datasets
# For that, create an empty dataframe
mytestbaseline <- data.frame(matrix(NA,nrow=5,ncol=5))

#populate with select values for training
for(n in 1:n){
  for(m in 1:m){
    if(m == 1 & n == 1){
       mytestbaseline[n,m] <- mydf.baselinePred[n,m]
    }else if(m == 2 & n == 2){
       mytestbaseline[n,m] <- mydf.baselinePred[n,m]
    }else if(m == 5 & n == 3){
       mytestbaseline[n,m] <- mydf.baselinePred[n,m]
   }else if(m == 4 & n == 4){
       mytestbaseline[n,m] <- mydf.baselinePred[n,m]
   }else if(m == 3 & n == 5){
      mytestbaseline[n,m] <- mydf.baselinePred[n,m]
   }
  }
}

# RMSE for the test set

rmseDFTestbaseline <- sqrt( mean( (test.in-mytestbaseline)^2 , na.rm = TRUE ) )
rmseDFTestbaseline
```

```
## [1] 0.4926234
```

```r
## Get the train data ready
mytrainbaseline <- mydf.baselinePred
#populate NAs for the indexes where test data exists
mytrainbaseline[1,1] <- NA
mytrainbaseline[2,2] <- NA
mytrainbaseline[5,3] <- NA
mytrainbaseline[4,4] <- NA
mytrainbaseline[3,5] <- NA

# Similarly do RMSE on training set
rmseDFTrainbaseline <- sqrt( mean( (train.in-mytrainbaseline)^2 , na.rm = TRUE ) )
rmseDFTrainbaseline
```

```
## [1] 0.5066094
```

Display the RMSE values beore baseline and after baseline prediction.

```r
rmseCompareDF<-data.frame(c(rmseDFTrain,rmseDFTest),c(rmseDFTrainbaseline,rmseDFTestbaseline))
names(rmseCompareDF) <- c("rmse_Original","rmse_Baseline")
rownames(rmseCompareDF) <- c("Training_Set","Test_Set")


print(xtable(rmseCompareDF, digits =2, type = "html",caption="RMSE Values"),comment=FALSE)
```

Now,to find the percentage improvement, divide the RMSE from the baseline predictor from the RMSE of the original set. Lower the value of the RMSE baseline predictor, the better it is for the performance.

	rmse_Original	rmse_Baseline
Training_Set	0.71	0.51
Test_Set	0.54	0.49

Table 4: RMSE Values

```
TrainImprovement <- (1-rmseDFTrainbaseline/rmseDFTrain) * 100
TrainImprovement
```

```
## [1] 28.60879
```

```
TestImprovement <- (1-rmseDFTestbaseline/rmseDFTest)
TestImprovement
```

```
## [1] 0.08976125
```

Summary

The percentage improvement for the test dataset is 0.089 percent and 28% for the training set. So we have improved almost 30 percent for the training set, but not that much for the test set. The goal was to bring the RMSE value for the test dataset down. It should be noted that performance improvement from this approach will be different if a different example is used. The improvement can even be negative indicating that the performance had gotten worse and unknown patterns in data could be the reason. However by and large, we can expect that the approach used in this project will make the performance better, even if its a slight improvement.

In the customer-restaurant interactions, the customers rated differently. Customer #U1081 is the harshest reviewer and the restaurant #135046 is perceived less favorably by customers, so there is a bias. This bias formed the basis for the baseline predictor above.

References:

Coursera/Stanford Networks Illustrated course

Challenges:

The restaurant dataset was too big to understand this project, so a small subset had to be chosen. Choosing a few users resulted in too many NA values, so two extra rows were appending so to get a fuller dataset. The data could not be split into training and test set using the 'sample' call because this split resulted in some users not in both training and the test data. For this project, having all the users in each set was needed to properly illustrate the user-restaurant bias.