ECS189G Term Project

Rohail Asad, Arhan Khalid, Raffi Samurkashian

March 2020

1 Introduction

There are many different methods recommender systems can employ in order to construct prediction models and then use said models to make predictions on new data. The program implemented here gives the user the option to select between four of these widely used methods, logistic regression, Nonnegative Matrix Factorization, K Nearest Neighbor, and Classification and Regression Trees, in order to build a prediction model for his or her data. In addition, the program is formatted in such a way that allows for its general use, such as with variety of different datasets.

2 Input Arguemnts

The program commences by calling its main driver function, referred to as ratingProbsFit(). In addition to the five input arguments specified by the prompt, an additional input argument specifying a test dataset was assigned to the function arguemnts list. The five arguments are as follows:

1. dataIn, the data the prediction model will be constructed with
2. maxrating, which specifies the maximum rating that appears in dataIn
3. predMethod, which specifies the prediction model to be built
4. embedMeans, which specifies whether or not dimension reduction by mean embedding will take place
5. specialArgs, which specifies any user defined special parameters
6. newData, which specifies the data the user wishes to apply to the model

There are a few things to consider when calling this function. First, the prediction method must be put in quotation marks so that R does not interpret it as a variable name from the workspace. Second, we have decided to limit the special Arguments to numeric values describing rank values only, as other specifying arguments were considered as having minimal effect on the function of the prediction methods. Finally, when specifying that dimension reduction via mean embedding is to be used, it is important that the user provide a data set for the newData argument that has been similarly mean embedded, otherwise the input data will fall outside the scope of the prediction method and inaccurate probabilities will be returned. In order to perform logistic regression on a dataset named user.data with ratings ranging from 1 to 5, using mean embedding, and with a mean embedded testing dataset named test.data, the following function call is required:

ratingProbsFit(user.data, 5, ”logit”, TRUE, 0, test.data)

3 Generalization of Data Sets

As generality of the program is of paramount importance here, the program must be able to process input datasets in a universal manor. Therefore, when a dataset is supplied as an input argument, the names describing the columns within the dataset are replaced with ”V1,” ”V2,” and ”V3” to denote the first, second, and third columns, respectively. This way, any three column dataset, such that the first two columns contain user and item IDs and the third column contains ratings, can be processed by the program’s four prediction methods. The two datasets that were used to analyze the performance of the prediction methods are the Insteval dataset and the Song List dataset. Note: due to size, the Song List dataset was reduced to one ten thousandth of its original size using random sampling.

4 Prediction Method 1: Logistic Regression

The glm() function in R is used to apply a generalized linear model embedded within the logistic function, which compresses data to fit between an interval of 0 to 1 using an inverse exponential operation. This function differs from the traditional lm() function in that it only accepts input values greater than 0 and less than 1. In order to correctly format the input data, five separate logical vectors were created: one containing a 1 for all instances in which a rating of 1 is given and 0 otherwise, one containing a 1 for all instances in which a rating of 2 is given and 0 otherwise, and so on for all possible ratings. Glm() is called on all of the numerical vectors separately, creating a list of coefficients specific to each rating. The fitted values for from each of the glm() outputs are then used to create the list of rating probabilities for the given input data. Misclassification error for this method was calculated by assigning a correct classification if the highest probability for a given rating was associated with that specified rating in the original dataset, and assigning an incorrect classification otherwise.

5 Prediction Method 2: Nonnegative Matrix Factorization (NMF)

Here, Nonnegative Matrix Factorization was used to find the probabilities of certain ratings. First, dummy variables were created for each of the possible ratings by creating a series of data frames, one for each rating, and then using a for loop to iterate through the dataset. The data frames were populated with ones and zeroes based on the rating given by the user to that specific item. For example, if the rating given by the user to a specific movie was three, a value of one was assigned to the specified index in the third data frame and zeros were assigned to the index in the remaining data frames. At the end, the data frames, containing the logical data, were combined with the user ID and item ID data.
Next, the W and H matrices were calculated. In order to do so, five different training sets were created, each one consisting of the user ID, item ID and the corresponding column/rating for which we the probability was being predicted. For example, when calculating the probability of a specific user giving a rating of three to a specific item, we would place the userID, itemID and the sixth column which the column for rating three into a data memory. The recosystem package was then used to perform NMF on the data. Feeding the data as data memory in the Recotrain function of recosystem package, the W and H matrices were formed. After running Recotrain for every possible rating within the dataset, a series of W and H matrices were created that would later be used to calculate the specific probabilities of giving a specified rating.
The final step was to assign probabilities to the ratings based on these W and H matrices. A unique predict function was written that multiplied the W and H matrices. A for loop to multiply the rows of matrix W with the columns of the transposed matrix H, according to the user and item data that was provided. After multiplying W and H matrices, the program returns a list of probabilities for each of the possible ratings given the list of user IDs and item IDs. Probabilities were normalized to 1 in the event that the sum of all the probabilities was slightly less than or greater than 1.
Several different ranks were tested in order to obtain the most accurate method of prediction using both of the previously specified datasets. Figure 1 shows the mean absoulte prediction error (MAPE) associated with each of the ranked NMF iterations. Using a rank of 700 seems to produce the lowest MAPE with both datasets and is used as a default value, unless the user specifies a different rank in the main function call.

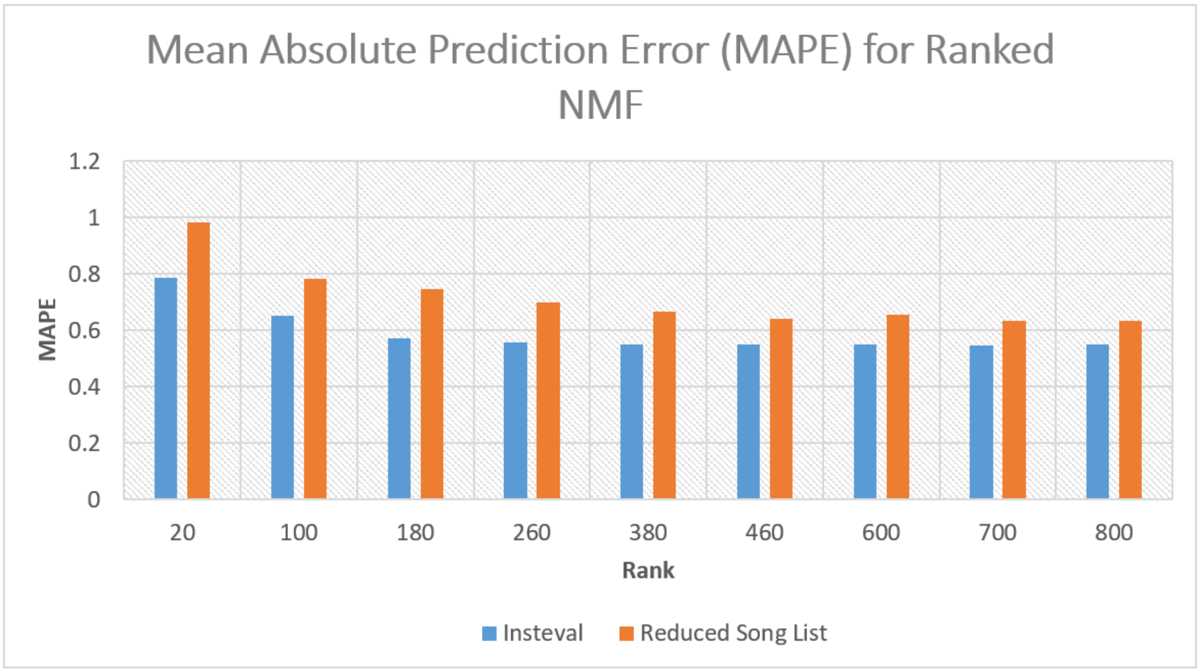


Figure 1: MAPE for NMF

6 Prediction Method 3: K Nearest Neighbor (KNN)

K Nearest Neighbor (KNN) is a classical statistical, non-parametric machine learning technique in which all cases are stored and new cases are classified based on the similarity measure using a distance function. Prediction of new rating for a new or old user in a dataset is done based on the majority vote of a given index's neighbors, with the prediction being assigned to the class most common amongst its K nearest neighbor, measured by a distance function (cosDist(), in this case).In order to make a prediction, the predict() function from the rectools package is used. The predict function is a generic function in R which, when using KNN, dispatches the call to the method predict.usrData() function after the class of the input argument is characterized; the k closest users in the supplied dataset that have given a rating for the specified item are found, and a prediction is made by taking the mean of those ratings.
The input data for the predict.usrData() is run through the formUserData() function which organizes the data into an R list, such that one element per user and each such element denotes the ratings made by that user. Predict.usrData() takes origData (of class usrData), newData, newItem, k (the number of nearest neighbors) as arguments. It outputs an R list, of class usrData that has one element per userID, that element, of class usrDatum has userID, itms, and ratings.

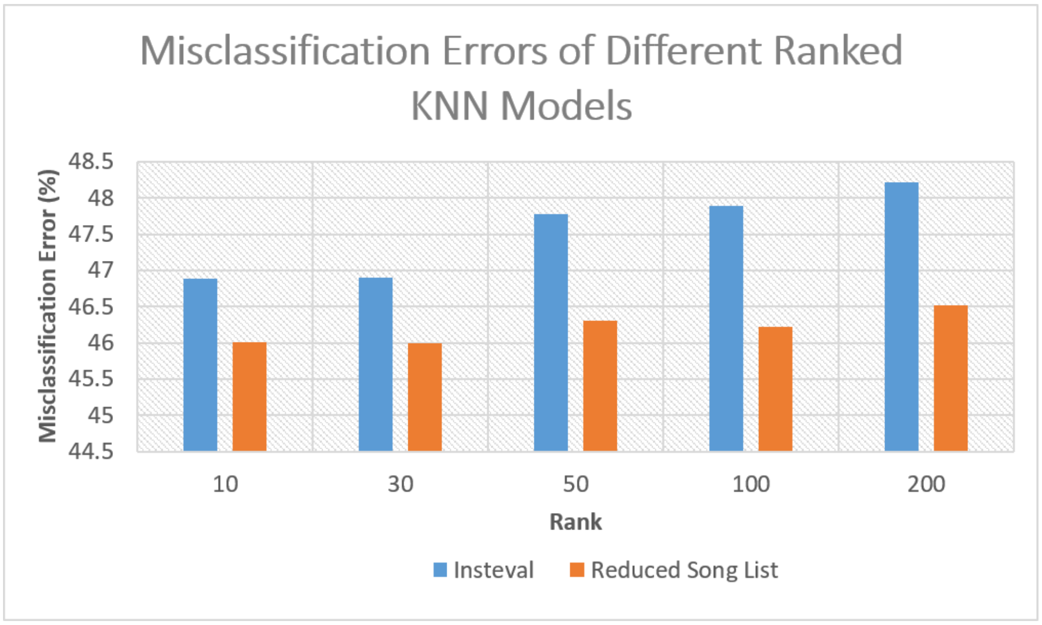


Figure 2: Misclassification Error for KNN

After running the input dataset into formUserData, ratings from each user is put into the required number dummy columns by using factorToDummies() (5 dummy columns for 1-5 ratings in the cases of the Insteval and the reduced song list datasets). This is then passed as origData, which is narrowed down to the users who have rated a given newItem after matching the item of interest in the original dataset. Then, a matrix is created with only users who have rated that item and users that have not rated the given item are removed to simplify the data. In order to find distance from newData to one user within origData, the cosDist() function is used, which calculates the cross product of vectors leading from newData to its specified neighbors. After finding the nearest neighbors, average the predicting weight, then average the specified number of vectors will result in the probabilities that a user will give a specified rating.
Several different ranks were tested in order to obtain the most accurate method of prediction using both of the previously specified datasets. Figure 2, shown above, shows the misclassification rate associated with each of the ranked KNN iterations. Using a rank of 10 seems to produce the lowest misclassification rate with both datasets and is used as a default value, unless the user specifies a different rank in the main function call.
Misclassification rate for this method was calculated by assigning a correct classification if the highest probability for a given rating was associated with that specified rating in the original dataset, and assigning an incorrect classification otherwise.

7 Prediction Method 4: Classification and Regression Trees (CART)

Classification and Regression Trees (CART) make predictions by assembling a system of nodes and vertices into what can be visually described as an 'upside-down tree.' For any given input, its values are fed into the comparative statements governing each node and the resulting logic directs the input to one of nodes below it. This propagation repeats until a path is created that reaches one of the bottom most nodes of the tree, referred to as a leaf. Depending on which leaf is reached, the corresponding prediction is assigned to the input. For the the purposes of this project, a tree was assembled using user ID's and item ID's from the data set. In order to increase the accuracy of the tree, there are multiple different hyperparameters that can be changed in order to alter the structure of the tree. For example, the maxdepth control option (accessed using control = ctreecontrol(maxdepth = x)) allows the user to specify the maximum number of levels the tree is allowed to have, and therefore limiting the number of leaves, referred to as "pruning" the tree. The minsplit control option (accessed using control = ctreecontrol(minsplit = x)) allows the user to specify the threshold for further splitting existing nodes within the tree. Using these hyperparameters, six different tree structures were tested using the generalized, mean embedded datasets.

1. Default Tree (no hyperparameters changed)
2. Maxdepth set to 3
3. Maxdepth set to 5
4. Minsplit set to 10
5. Minsplit set to 20
6. Minsplit set to 5

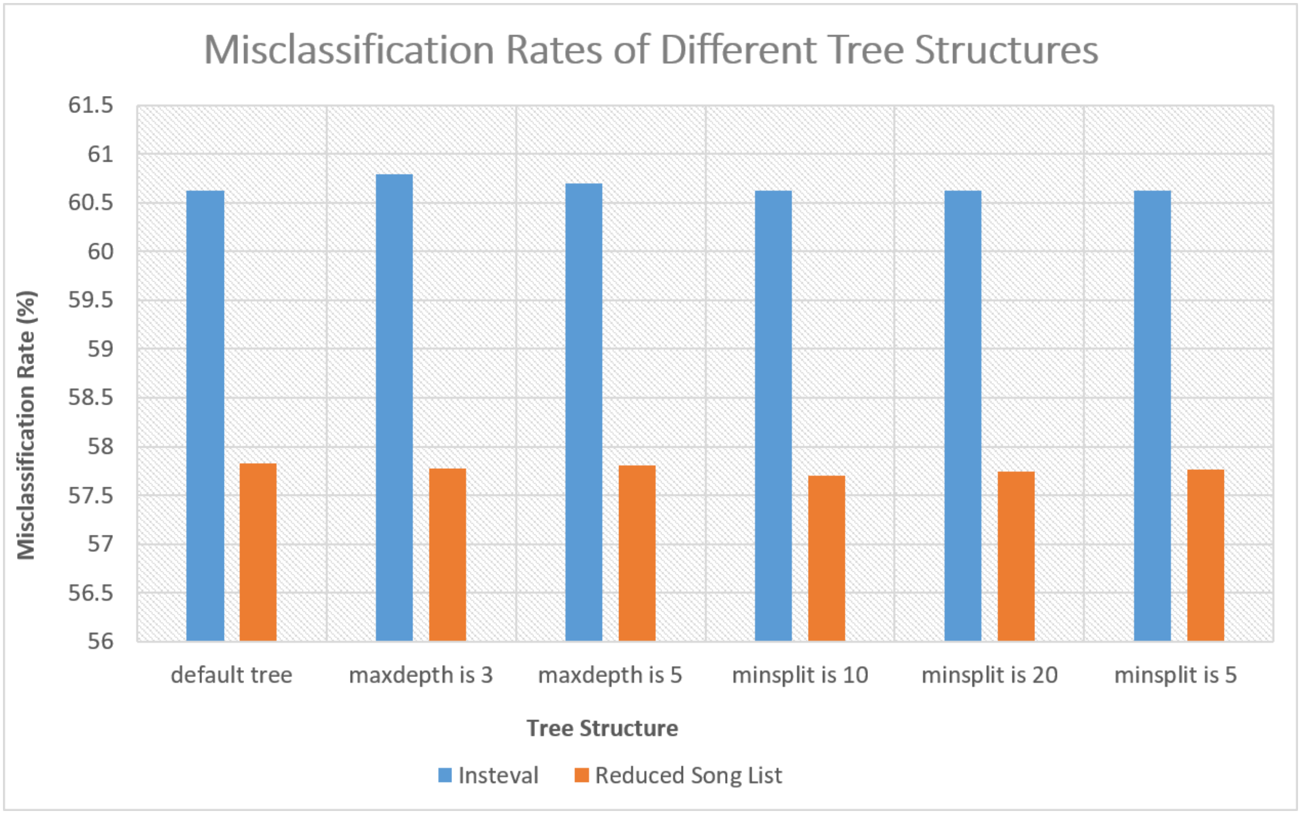


Figure 3: Misclassification Error for CART

Figure 3 shows the misclassification rate, or the proportion of incorrect predictions, of each tree when using the Insteval and reduced song list data sets. While all of the tree structures produce misclassification rates of around 60 to 61 percent for using the Insteval dataset and 57 to 58 percent using the reduced song list dataset, the tree in which minsplit was set to 10 seems to perform the best and was chosen as the tree structure for the final version of the program. While the misclassification rates for the trees are relatively high, this is to be expected, as covariates were not used as comparative parameters for the nodes and accurately predicting rating using just user ID and item ID is difficult. Nevertheless, the structure of the implemented tree's node layout is shown in figure 4.

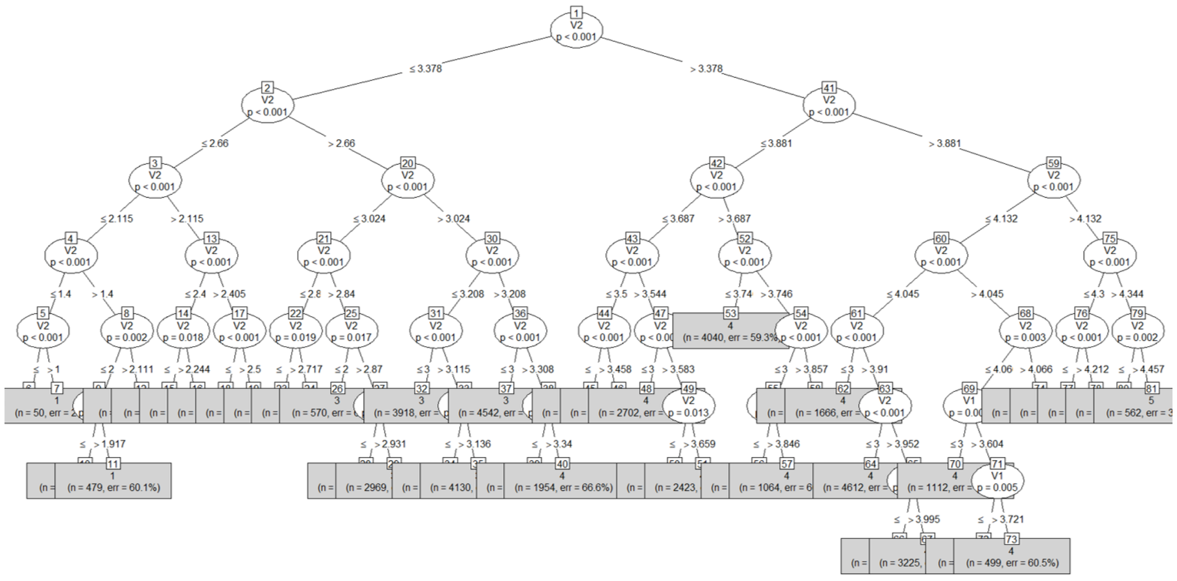
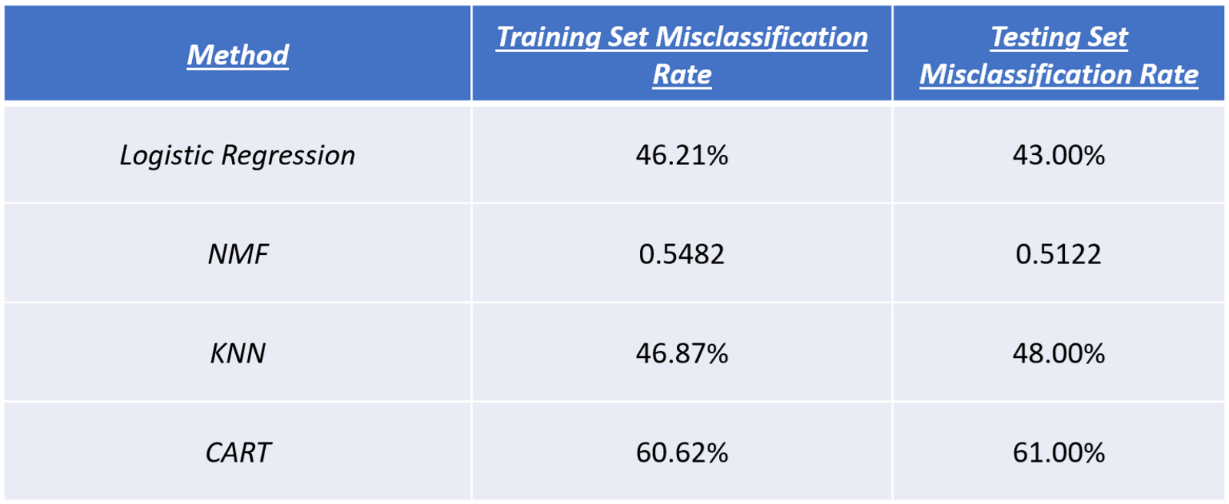


Figure 4: Optimal CART Structure

8 Cross Validation

Cross validation was performed on each of the prediction methods in order to confirm the results seen when training using the original datasets. In order to generate test datasets such that all users and items have already appeared in the training datasets, a small portion of the users and items columns of the original dataset (a size 100 datapoints was used from the Insteval dataset) were copied, and the contents of the items column was randomized, resulting in both individual users and items that have already been seen in both models, however in pairings that have previously not been seen. Figure 5, shown on the following page, shows the results of the cross-validation. Note: cross validation data for NMF is represented as Mean Absolute Prediction Error (MAPE), while the cross-validation data for the other 3 prediction methods is represented as misclassification rate. It is also worth mentioning that a mean embedded testing set was created using the same methodology described above in order from the mean embedded original dataset to cross-validate CART. Figure 5 shows the results of the cross validation.



Method	***Training Set Misclassification Rate***	***Testing Set Misclassification Rate***
Logistic Regression	46.21%	43.00%
NMF	0.5482	0.5122
KNN	46.87%	48.00%
CART	60.62%	61.00%

Figure 5: Results of Cross-validation

9 Generalized recProbs Class

Since a theme of this program is generallity, it is important that the predict function that is called to perform the predictions is done universally across all prediction methods. In order to accomplish this, a recProbs S3 class is created following the creation of each prediction model and loaded with all the necessary data for the recProbs associated predict() function to make its predictions, regardless of the method. Figure 6, shown on the following page, shows the contents of the recProbs class depending on which prediction method is being used. Each object that is created contains a numeric variable, method, which informs the recProbs associated predict() function which prediction method the class decends from and therefore how to make the prediction. For example, if 4 was assigned to method, then the recProbs associated predict() function knows that the data within the recProbs class refers to the node structure of a decision tree and can call the ctree associated predict function. The first six lines of output (usig the bult in head() function) are shown, on the next page in figure 7, when this call is made. Each row represents a user in the newX data rating a specific item and each element within that row represents the probability that the row specified user will give the column specified rating. For example, the first user has a 1.9 percent chance of rating the item as 1, a 5.3 percent chance of rating the item as 2, and so on.

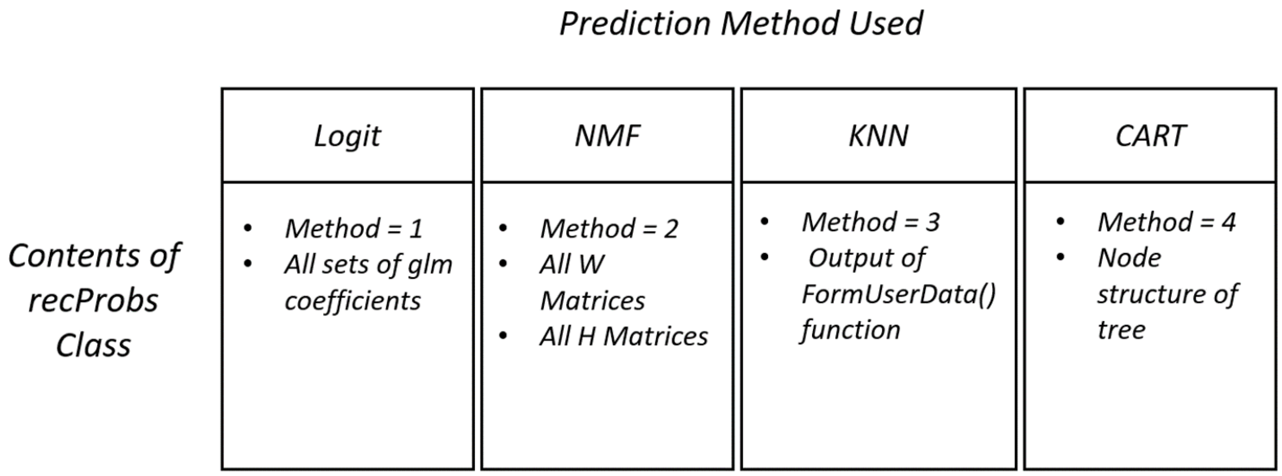
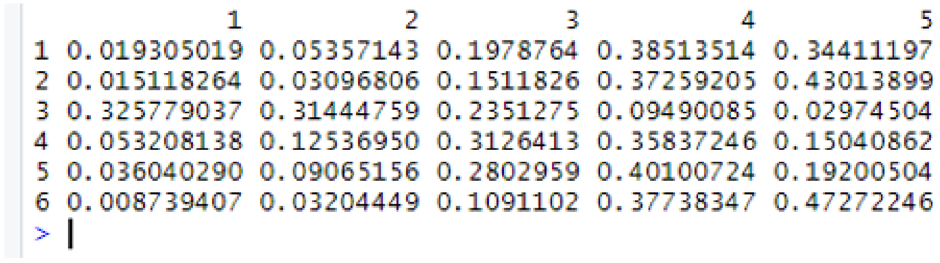


Figure 6: RecProbs Class Contents



```
            1          2         3          4          5
1 0.019305019 0.05357143 0.1978764 0.38513514 0.34411197
2 0.015118264 0.03096806 0.1511826 0.37259205 0.43013899
3 0.325779037 0.31444759 0.2351275 0.09490085 0.02974504
4 0.053208138 0.12536950 0.3126413 0.35837246 0.15040862
5 0.036040290 0.09065156 0.2802959 0.40100724 0.19200504
6 0.008739407 0.03204449 0.1091102 0.37738347 0.47272246
> 
```

Figure 7: Program Output

10 Contributions

Raffi Samurkashian was responsible for writing the code that performed the generalized linear regression model and CART, along with their associated sections in the report, the recProbs class and its associated predict function, and putting together the figures and supporting document. Rohail Asad was responsible for writing the code that performed the NMF model as well as its associated section in the report. Arhan Khalid were responsible for writing the code that performed the KNN model and its associated section in the report.

11. Appendix

```
ratingProbsFit <- function(dataIn,maxRating,predMethods,embedMeans,specialArgs,newData){
  max <- as.numeric(maxRating)
  method <- as.character(predMethods)
  x <- dataIn
  names(x)[1] <- "V1"
  names(x)[2] <- "V2"
  names(x)[3] <- "V3"
  if(method == "logit")
  {
    if(embedMeans == TRUE)
    {
      col_1_means <- tapply(x$V3,x$V1,mean)
      col_2_means <- tapply(x$V3,x$V2,mean)
      x_emb <- x
      x_emb$V1 <- col_1_means[x$V2]
      x_emb$V2 <- col_2_means[x$V2]
      x_emb$V1 <- as.vector(x_emb$V1)
      x_emb$V2 <- col_2_means[x$V2]
      x_emb$V2 <- as.vector(x_emb$V2)
      x_emb$V3 <- as.factor(x_emb$V3)
      range_1 <- as.integer(x_emb$V3 == 1)
      range_2 <- as.integer(x_emb$V3 == 2)
      range_3 <- as.integer(x_emb$V3 == 3)
      range_4 <- as.integer(x_emb$V3 == 4)
      range_5 <- as.integer(x_emb$V3 == 5)
      glmout_1 <- glm(range_1 ~ V1 + V2, data = x_emb, family = "binomial")
      glmout_2 <- glm(range_2 ~ V1 + V2, data = x_emb, family = "binomial")
      glmout_3 <- glm(range_3 ~ V1 + V2, data = x_emb, family = "binomial")
      glmout_4 <- glm(range_4 ~ V1 + V2, data = x_emb, family = "binomial")
      glmout_5 <- glm(range_5 ~ V1 + V2, data = x_emb, family = "binomial")
      probsFitOut <- list(method = 1, data_1 = glmout_1, data_2 = glmout_2, data_3 = glmout_3,
data_4 = glmout_4, data_5 = glmout_5)
      class(probsFitOut) <- "recProbs"
      predict(probsFitOut, newData)
    } else {
    range_1 <- as.integer(x$V3 == 1)
    range_2 <- as.integer(x$V3 == 2)
    range_3 <- as.integer(x$V3 == 3)
    range_4 <- as.integer(x$V3 == 4)
    range_5 <- as.integer(x$V3 == 5)
    glmout_1 <- glm(range_1 ~ V1 + V2, data = x, family = "binomial")
    glmout_2 <- glm(range_2 ~ V1 + V2, data = x, family = "binomial")
    glmout_3 <- glm(range_3 ~ V1 + V2, data = x, family = "binomial")
    glmout_4 <- glm(range_4 ~ V1 + V2, data = x, family = "binomial")
    glmout_5 <- glm(range_5 ~ V1 + V2, data = x, family = "binomial")
    probsFitOut <- list(method = 1, data_1 = glmout_1, data_2 = glmout_2, data_3 = glmout_3,
data_4 = glmout_4, data_5 = glmout_5)
    class(probsFitOut) <- "recProbs"
    predict(probsFitOut, newData)
    }
  }

  if(method == "NMF")
  {
    # songsDataset <- read.csv("songsDataset.csv")
    # x<- songsDataset
    rank <- as.numeric(specialArgs)
    r <- Reco()
    #x <- cbind(InstEval[,c(1:2,7)])
    x<-dataIn
    col1 <- data.frame()
    col2 <- data.frame()
    col3 <- data.frame()
    col4 <- data.frame()
    col5 <- data.frame()
    final <- data.frame()
    number <- nrow(x)
    #number <- 10000
    row <- 1 : number
    #print("working1")
    for (i in row)
    {
      if (x[i,3] ==1)
      {
        col1 <- rbind(col1, c(1))
        col2 <- rbind(col2, c(0))
        col3 <- rbind(col3, c(0))
        col4 <- rbind(col4, c(0))
        col5 <- rbind(col5, c(0))
      }
      if (x[i,3] ==2)
      {
        col1 <- rbind(col1, c(0))
        col2 <- rbind(col2, c(1))
        col3 <- rbind(col3, c(0))
        col4 <- rbind(col4, c(0))
        col5 <- rbind(col5, c(0))
      }
      if (x[i,3] ==3)
      {
        col1 <- rbind(col1, c(0))
        col2 <- rbind(col2, c(0))
        col3 <- rbind(col3, c(1))
        col4 <- rbind(col4, c(0))
        col5 <- rbind(col5, c(0))
      }
      if (x[i,3] ==4)
      {
        col1 <- rbind(col1, c(0))
        col2 <- rbind(col2, c(0))
        col3 <- rbind(col3, c(0))
        col4 <- rbind(col4, c(1))
        col5 <- rbind(col5, c(0))
      }
      if (x[i,3] ==5)
      {
        col1 <- rbind(col1, c(0))
        col2 <- rbind(col2, c(0))
        col3 <- rbind(col3, c(0))
        col4 <- rbind(col4, c(0))
        col5 <- rbind(col5, c(1))
      }
    }
    #print("working2")
    first <- x[1:number,1]
```

```
second <- x[1:number, 2]
data <- data.frame(first, second, col1, col2, col3, col4, col5)
head(data)
#--------------------------------------------------------------------------------
#Running NMF

ie3.trn <- data_memory(data[,1], data[,2], data[,3], index1= TRUE)
r$train(ie3.trn, opts= list(dim=rank, nmf=TRUE))
result1 <- r$output(out_memory(), out_memory())
w1 <- result1$P
h1 <- t(result1$Q)
#-------------
ie4.trn <- data_memory(data[,1], data[,2], data[,4], index1= TRUE)
r$train(ie4.trn, opts= list(dim=rank, nmf=TRUE))
result2 <- r$output(out_memory(), out_memory())
w2 <- result2$P
h2 <- t(result2$Q)
#-------------
ie5.trn <- data_memory(data[,1], data[,2], data[,5], index1= TRUE)
r$train(ie5.trn, opts= list(dim=rank, nmf=TRUE))
result3 <- r$output(out_memory(), out_memory())
w3 <- result3$P
h3 <- t(result3$Q)
#--------------
ie6.trn <- data_memory(data[,1], data[,2], data[,6], index1= TRUE)
r$train(ie6.trn, opts= list(dim=rank, nmf=TRUE))
result4 <- r$output(out_memory(), out_memory())
w4 <- result4$P
h4 <- t(result4$Q)
#---------------
ie7.trn <- data_memory(data[,1], data[,2], data[,7], index1= TRUE)
r$train(ie7.trn, opts= list(dim=rank, nmf=TRUE))
result5 <- r$output(out_memory(), out_memory())
w5 <- result5$P
h5 <- t(result5$Q)
#print("working3")
```

```
   probsFitOut <- list(method = 2,  data_1=w1, data_2=h1, data_3=w2, data_4=h2, data_5 = w3,
data_6=h3, data_7=w4, data_8=h4, data_9=w5, data_10=h5) # <---- Everything for NMF goes
here
   class(probsFitOut) <- "recProbs"
   predict(probsFitOut,newData )

  }
  if(method == "kNN")
  {

   #origData <- data.frame(InstEval[,c(1:2,7)])
   origData <- dataIn
   names(origData) <- c("userID", "itemID", "rating")
   origData$rating <- as.factor(origData$rating)
   ratToDummies <- factorToDummies(origData$rating, "rating")
   origData <- origData[, -3]
   origData <- cbind(origData, ratToDummies)
   inpData<- list(rep(NA, maxRating), type =any)
   for(i in 1:(maxRating-1)){
     inpData[[i]] <- formUserData(origData[, c(1:2, i+2)])
   }
   probsFitOut <- list(method = 3, data_1 = maxRating, data_2= inpData, data_3=specialArgs) #
<---- Everything for KNN goes here
   class(probsFitOut) <- "recProbs"
   # print("working")
   predict(probsFitOut, newData)


  }
  if(method == "CART")
  {
   col_1_means <- tapply(x$V3,x$V1,mean)
   col_2_means <- tapply(x$V3,x$V2,mean)
   x_emb <- x
   x_emb$V1 <- col_1_means[x$V2]
   x_emb$V2 <- col_2_means[x$V2]
   x_emb$V1 <- as.vector(x_emb$V1)
   x_emb$V2 <- col_2_means[x$V2]
   x_emb$V2 <- as.vector(x_emb$V2)
```

```
    x_emb$V3 <- as.factor(x_emb$V3)
    #ctout <- ctree(V3 ~ V1+V2,data=x_emb)
    ctout <- ctree(V3 ~ V1+V2,data=x_emb, control = ctree_control(minsplit = 10))
    #ctout <- ctree(V3 ~ V1+V2,data=x_emb, control = ctree_control(minsplit = 20))
    #ctout <- ctree(V3 ~ V1+V2,data=x_emb, control = ctree_control(minsplit = 5))
    #ctout <- ctree(V3 ~ V1+V2,data=x_emb, control = ctree_control(maxdepth = 3))
    #ctout <- ctree(V3 ~ V1+V2,data=x_emb, control = ctree_control(maxdepth = 5))
    #plot(ctout, type = "simple")
    probsFitOut <- list(data = ctout, method = 4)
    class(probsFitOut) <- "recProbs"
    predict(probsFitOut, newData)
    #head(output)
    #error_table <- table(actual = x$V3, fitted = output)
    #print(error_table)
    #misclassification_error <- (1 - (sum(diag(error_table))/sum(error_table)))
    #print(misclassification_error)

  }

}

predict.recProbs <- function(probsFitOut, newXs){
  if(probsFitOut$method == 1)
  {
    predicted_ratings <- data.frame(prob_1 = exp(predict(probsFitOut$data_1, newXs)), prob_2 =
exp(predict(probsFitOut$data_2, newXs)), prob_3 = exp(predict(probsFitOut$data_3, newXs)),
prob_4 = exp(predict(probsFitOut$data_4, newXs)), prob_5 = exp(predict(probsFitOut$data_5,
newXs)))
    rows <- 1:nrow(predicted_ratings)
    for (i in rows)
    {
      sum <- predicted_ratings[i,1] + predicted_ratings[i,2] + predicted_ratings[i,3] +
predicted_ratings[i,4] + predicted_ratings[i,5]
      scale <- 1/sum
      predicted_ratings[i,1] <- predicted_ratings[i,1]*scale
      predicted_ratings[i,2] <- predicted_ratings[i,2]*scale
      predicted_ratings[i,3] <- predicted_ratings[i,3]*scale
      predicted_ratings[i,4] <- predicted_ratings[i,4]*scale
      predicted_ratings[i,5] <- predicted_ratings[i,5]*scale
    }
    print(predicted_ratings)
  }
  if(probsFitOut$method == 2)
  {
    print("we are predicting NMF")
    w1 <- probsFitOut$data_1
    h1 <- probsFitOut$data_2
    w2 <- probsFitOut$data_3
    h2 <- probsFitOut$data_4
    w3 <- probsFitOut$data_5
    h3 <- probsFitOut$data_6
    w4 <- probsFitOut$data_7
    h4 <- probsFitOut$data_8
    w5 <- probsFitOut$data_9
    h5 <- probsFitOut$data_10

    #------------------------------
    size <- nrow(newXs)
    rat1 <- vector(length=size)
    rat2 <- vector(length=size)
    rat3 <- vector(length=size)
    rat4 <- vector(length=size)
    rat5 <- vector(length=size)
    for (i in 1:size)
    {
      j<- newXs[i,1]
      k<- newXs[i,2]
      if(is.na(j) || is.na(k))
      {
        rat1[i]<- NA
        rat2[i]<- NA
        rat3[i]<- NA
        rat4[i]<- NA
        rat5[i]<- NA
      }
      else
      {
        rat1[i] <- w1[j,] %*%  h1[,k]
        rat2[i] <- w2[j,] %*% h2[,k]
        rat3[i] <- w3[j,] %*% h3[,k]
        rat4[i] <- w4[j,] %*% h4[,k]
        rat5[i] <- w5[j,] %*% h5[,k]
      }
    }
    prediction <-   data.frame(rat1, rat2, rat3, rat4, rat5)
    #print(prediction)
    rows <- nrow(prediction)
    num <- 1:rows
    for(k in num)
    {
      sum <- prediction[k,1] + prediction[k,2]+ prediction[k,3]+ prediction[k,4]+ prediction[k,5]
      scale <- 1 / sum
      prediction[k,1] <-  prediction[k,1] * scale
      prediction[k,2] <-  prediction[k,2] * scale
      prediction[k,3] <-  prediction[k,3] * scale
      prediction[k,4] <-  prediction[k,4] * scale
      prediction[k,5] <-  prediction[k,5] * scale
    }
    print(prediction)
  }
  if(probsFitOut$method == 3)
  {
    print("we are predicting KNN")
    maxRating <- probsFitOut$data_1
    data <- probsFitOut$data_2
    specialArgs <- probsFitOut$data_3

    pred <- matrix(nrow = dim(newXs)[1], ncol= maxRating)
    for(i in 1:(maxRating-1)){
      for(j in 1: dim(newXs)[1]){
        for(k in 1: length(data[[i]])){
          if(data[[i]][[k]]$userID == newXs$userID[j]){
            pred[j,i] <- predict(data[[i]], list(data[[i]][[k]]), newXs$itemID[j],specialArgs)
          }
        }
      }
    }
    pred[, maxRating] <- 1-rowSums(pred[,1:maxRating-1])
    pred <- as.matrix(pred)
    print(pred)
  }
  if(probsFitOut$method == 4)
  {
    output <- predict(probsFitOut$data, newXs, type = "prob")
    print(output)
  }

}
```