Homework Set 2 in R

DATA 624-01 Group 3

Z. Palmore, K. Popkin, K. Potter, C. Nan, J. Ramalingam

7/8/2021

Table of Contents

## KJ 6.3

A chemical manufacturing process for a pharmaceutical product was discussed in Sect. 1.4. In this problem, the objective is to understand the relationship between biological measurements of the raw materials (predictors),measurements of the manufacturing process (predictors), and the response of product yield. Biological predictors cannot be changed but can be used to assess the quality of the raw material before processing. On the other hand, manufacturing process predictors can be changed in the manufacturing process. Improving product yield by 1% will boost revenue by approximately one hundred thousand dollars per batch:

### Part A

#### Question

Start R and use these commands to load the data: > library(AppliedPredictiveModeling) > data(chemicalManufacturingProcess)

#### Code

library(AppliedPredictiveModeling)  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
head(df)

#### Response

The matrix processPredictors contains the 57 predictors (12 describing the input biological material and 45 describing the process predictors) for the 176 manufacturing runs. yield contains the percent yield for each run.

### Part B

#### Question

A small percentage of cells in the predictor set contain missing values. Use an imputation function to fill in these missing values (e.g., see Sect. 3.8).

#### Code

#Fill in missing values with the median of each feature  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}

#### Response

### Part C

#### Question

Split the data into a training and a test set, pre-process the data, and tune a model of your choice from this chapter.

#### Code

library(reshape2)  
library(dplyr)  
cormat <- round(cor(df),2)  
melted\_cormat = melt(cormat)  
melted\_cormat\_df = data.frame(melted\_cormat)  
#Filter to only Yield and sort in descending order  
yield\_corr <- melted\_cormat\_df %>%   
 filter(Var2 == "Yield")  
yield\_corr$absvalue = abs(yield\_corr$value)  
yield\_corr2 = yield\_corr[order(-yield\_corr[,4]),]  
yield\_corr2

library(AppliedPredictiveModeling)  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
  
  
#Create the regression model  
lmyield = lm(Yield~ManufacturingProcess32 +   
 ManufacturingProcess36 +   
 ManufacturingProcess09 +   
 ManufacturingProcess13 +   
 BiologicalMaterial02 +   
 BiologicalMaterial06 +   
 BiologicalMaterial03,   
 data=train)  
summary(lmyield)

#### Response

After splitting the data into a 75% training, 25% test datasets, I chose to build a linear regression model using the top 7 features based on the correlations.

Here are the correlations…

Here is the model. The R2 of 61% isn’t very good and the only good predictor based on p-values and 0.05 cutoff is ManufacturingProcess32.

### Part D

#### Question

Predict the response for the test set.What is the value of the performance metric and how does this compare with the resampled performance metric on the training set?

#### Code

#predict with the test data  
predict(lmyield, newdata = test, interval ='prediction')

library(Metrics)  
#Predict with the test data  
lmyield\_test = predict(lmyield, newdata = test, interval ='prediction')  
lmyield\_test\_df = data.frame(lmyield\_test)  
lmyield\_test\_df$actual = test$Yield  
lmyield\_test\_df  
lmyield\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
cat('RMSE of the test data for this model is', lmyield\_test\_rmse)

####Response

Using RMSE as the performance metric, the value if 1.25 as shown below… The optimal value of the performance metric for the Test data is 1.25 as shown below…

### Part E

#### Question

Which predictors are most important in the model you have trained? Do either the biological or process predictors dominate the list?

#### Response

As shown in the above correlations list the first 7 most correlated features are 4 process features, followed by 3 biological features.  
In reviewing the regression model’s summary, the Process32 and Process36 are the most relevant p-values and the BiologicalMaterial02 feature is the third most relevant. Note that only the Process32 feature has a p-value < 0.05.

### Part F

#### Question

Explore the relationships between each of the top predictors and the response. How could this information be helpful in improving yield in future runs of the manufacturing process?

#### Response

Per the previous exploration via correlations and p-values, the process features are topping the list of most relevant features. This bodes well as these features can be changed (biological ones cannot change), so simulations could be done using the regression model and modified values.

Note that prior to experimenting with process feature value changes, more experimentation is needed on what features to use. Having an Adjusted R2 on the training data of only 60% and having only one feature in the model summary with a p-value of 0.05 indicates there is room for improvement. A good next step might be to build the model via forward stepwise or backward stepwise regression. Or, training on some other model such as Random Forest, could also be used to identify which features are most relevant.

## KJ 7.2

Friedman (1991) introduced several benchmark data sets created by simulation. One of these simulations used the following nonlinear equation to create data: y = 10 sin(πx1x2) + 20(x3 − 0.5)2 + 10x4 + 5x5 + N(0, σ2) where the x values are random variables uniformly distributed between [0, 1] (there are also 5 other non-informative variables also created in the simulation).The package mlbench contains a function called mlbench.friedman1 that simulates these data: > library(mlbench) > set.seed(200) > trainingData <- mlbench.friedman1(200, sd = 1) > ## We convert the ‘x’ data from a matrix to a data frame > ## One reason is that this will give the columns names. > trainingDatax) > ## Look at the data using > featurePlot(trainingDatay) > ## or other methods. > > ## This creates a list with a vector ‘y’ and a matrix > ## of predictors ‘x’. Also simulate a large test set to > ## estimate the true error rate with good precision: > testData <- mlbench.friedman1(5000, sd = 1) > testDatax)

### Part A

#### Question

Tune several models on these data. An example is shown in the code. Which models appear to give the best performance? Does MARS select the informative predictors (those named X1–X5)?

registerDoParallel()  
registerDoParallel(cores=2)  
getDoParWorkers()

Code

library(mlbench)  
library(caret)  
library(AppliedPredictiveModeling)  
library(kernlab)  
library(doParallel) # Used for computation  
library(earth) # Package necessary for marsModel  
registerDoParallel(cores=2)  
getDoParWorkers()  
set.seed(200)  
trainingData <- mlbench.friedman1(200, sd = 1)  
## We convert the ' x ' data from a matrix to a data frame  
## One reason is that this will give the columns names.  
trainingData$x <- data.frame(trainingData$x)  
## Look at the data using  
#featurePlot(trainingData$x, trainingData$y)  
## or other methods.  
## This creates a list with a vector ' y ' and a matrix  
## of predictors ' x ' . Also simulate a large test set to  
## estimate the true error rate with good precision:  
testData <- mlbench.friedman1(5000, sd = 1)  
testData$x <- data.frame(testData$x)  
knnModel <- train(x = trainingData$x, y = trainingData$y, method = "knn", preProc = c("center", "scale"), tuneLength = 10)  
knnPred <- predict(knnModel, newdata = testData$x)  
## The function 'postResample' can be used to get the test set   
## performance values  
postResample(pred = knnPred, obs = testData$y)  
svmModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "svmRadial",  
 tuneLength=10,  
 preProc = c("center", "scale"))  
svmPred <- predict(svmModel, newdata = testData$x)  
postResample(pred = svmPred, obs = testData$y)  
nnetGrid <- expand.grid(.decay=c(0, 0.01, 0.1, 0.5, 0.9),  
 .size=c(1, 10, 15, 20),  
 .bag=FALSE)  
nnetModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "avNNet",  
 tuneGrid = nnetGrid,  
 preProc = c("center", "scale"),  
 trace=FALSE,  
 linout=TRUE,  
 maxit=500)  
# Neural net may take several minutes  
nnetPred <- predict(nnetModel, newdata = testData$x)  
postResample(pred = nnetPred, obs = testData$y)  
marsGrid <- expand.grid(.degree=1:2,  
 .nprune=2:20)  
marsModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "earth",  
 tuneGrid = marsGrid,  
 preProc = c("center", "scale"))  
marsPred <- predict(marsModel, newdata = testData$x)  
postResample(pred = marsPred, obs = testData$y)  
knnModel  
svmModel  
nnetModel  
marsModel  
varImp(marsModel)

knnPred <- predict(knnModel, newdata = testData$x)  
## The function 'postResample' can be used to get the test set   
## performance values   
postResample(pred = knnPred, obs = testData$y)

svmModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "svmRadial",  
 tuneLength=10,  
 preProc = c("center", "scale"))  
svmModel  
svmPred <- predict(svmModel, newdata = testData$x)  
postResample(pred = svmPred, obs = testData$y)

nnetGrid <- expand.grid(.decay=c(0, 0.01, 0.1, 0.5, 0.9),  
 .size=c(1, 10, 15, 20),  
 .bag=FALSE)  
  
nnetModel <- train(x = trainingData$x,  
 y = trainingData$y,  
 method = "avNNet",  
 tuneGrid = nnetGrid,  
 preProc = c("center", "scale"),  
 trace=FALSE,  
 linout=TRUE,  
 maxit=500)  
nnetModel  
nnetPred <- predict(nnetModel, newdata = testData$x)  
postResample(pred = nnetPred, obs = testData$y)

postResample(pred = nnetPred, obs = testData$y)  
postResample(pred = svmPred, obs = testData$y)  
postResample(pred = marsPred, obs = testData$y)  
varImp(marsModel)

Response SVM, Neural Networks and MARS models were used for testing performances. Below is R Code for the models. Comparing RMSE values of three models , It is clear that the MARS model performs better. The RMSE value is 1.32. Yes , It selects the informative predictors (X1-X5). R Code and results are below.

KJ 7.5 Exercise 6.3 describes data for a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several nonlinear regression models.

Part A Question Which nonlinear regression model gives the optimal resampling and test set performance?

Code

library(AppliedPredictiveModeling)  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
#Create the train x and train y datasets for HW 7.5  
trainx = train[,2:ncol(train)]  
trainy = train$Yield  
testx = test[,2:ncol(test)]  
testy = test$Yield  
# Recreate RMSE from 6.3

library(caret)  
#Using KNN for HW 7.5  
#Train KNN  
knnModel <- train(x=trainx, y=trainy)  
knnModel  
  
#Example from the book on how to predict  
knnPred <- predict(knnModel, newdata = testx)  
  
## The function 'postResample' can be used to get the test set  
## performance values  
postResample(pred = knnPred, obs = testy)  
#Using MARS for HW7.5  
#Train using MARS  
marsFit <- earth(trainx, trainy)  
summary(marsFit)  
  
#MARS tune  
# Define the candidate models to test  
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:38)  
  
# Fix the seed so that the results can be reproduced  
set.seed(100)  
marsTuned <- train(trainx, trainy,method = "earth",tuneGrid = marsGrid, trControl = trainControl(method = "cv"))  
  
marsTuned  
  
#Predict with MARS  
head(predict(marsTuned, testx))  
  
#Relevant features with MARS  
varImp(marsTuned)

Response MARS and KNN models were trained and tested with the chemical data. Here is the R code. The best performance, but only by a narrow margin, appears to come from MARS with an RMSE of 1.15 versus KNN’s RMSE of 1.21. Performance from MARS summary is that the final values used were nprune = 4 and degree = 2. This produced an RMSE of 1.15 as shown below. Performance from KNN summary is that the best mtry = 29. This produced an RMSE of 1.21 as shown below.

Part B Question Which predictors are most important in the optimal nonlinear regression model? Do either the biological or process variables dominate the list? How do the top ten important predictors compare to the top ten predictors from the optimal linear model?

Code

[missing code to show below - fill in]

Response The Process32, Process09, and Biological12 appear to be the most important features for the model. It appears that Process variables are more important, which bodes well for future improvements, since these variables can be modified. For the linear model, the Process32, Process36, and BiologicalMaterial02 were the top three features based on p-values. For the non-linear model the Process32, Process09, and Biological12 appear to be the most important features. Only the Process 32 appears in both models.

Part C Question Explore the relationships between the top predictors and the response for the predictors that are unique to the optimal nonlinear regression model. Do these plots reveal intuition about the biological or process predictors and their relationship with yield?

Code Response

KJ 8.1 8.1. Recreate the simulated data from Exercise 7.2: >library(mlbench) >set.seed(200) >simulated <- mlbench.friedman1(200, sd = 1) >simulated <- cbind(simulatedy) >simulated <- as.data.frame(simulated) >colnames(simulated)[ncol(simulated)] <- “y”

library(mlbench)  
library(randomForest)  
library(caret)  
library(partykit)  
library(dplyr)  
library(gbm)  
library(Cubist)  
set.seed(200)  
simulated <- mlbench.friedman1(200, sd = 1)  
simulated <- cbind(simulated$x, simulated$y)  
simulated <- as.data.frame(simulated)  
colnames(simulated)[ncol(simulated)] <- "y"  
model1 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp1 <- varImp(model1, scale = FALSE)  
rfImp1  
simulated$duplicate1 <- simulated$V1 + rnorm(200) \* .1  
cor(simulated$duplicate1, simulated$V1)  
model2 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp2 <- varImp(model2, scale = FALSE)  
rfImp2  
cforest\_model <- cforest(y ~ ., data=simulated)  
# Unconditional importance measure  
varimp(cforest\_model) %>% sort(decreasing = T)  
varimp(cforest\_model, conditional=T) %>% sort(decreasing = T)  
gbm\_Model <- gbm(y ~ ., data=simulated, distribution='gaussian')  
summary(gbm\_Model)  
cubistModel <- cubist(x=simulated[,-(ncol(simulated)-1)], y=simulated$y, committees=100)  
varImp(cubistModel)

Part A Question Fit a random forest model to all of the predictors, then estimate the variable importance scores. Did the random forest model significantly use the uninformative predictors (V6 – V10)?

Code

library(randomForest)  
library(caret)  
model1 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp1 <- varImp(model1, scale = FALSE)  
rfImp1

Response The importance given to the predictors (V6-V10) is very less and hence the usage is insignificant.

Part B Question Now add an additional predictor that is highly correlated with one of the informative predictors. Fit another random forest model to these data. Did the importance score for V1 change? What happens when you add another predictor that is also highly correlated with V1?

Code

model2 <- randomForest(y ~ .,   
 data = simulated,  
 importance = TRUE,  
 ntree = 1000)  
rfImp2 <- varImp(model2, scale = FALSE)  
rfImp2

Response The Importance is split between predictors, the importance of V1 predictor is split across the correlated variable.

Part C Question Use the cforest function in the party package to fit a random forest model using conditional inference trees. The party package function varimp can calculate predictor importance. The conditional argument of that function toggles between the traditional importance measure and the modified version described in Strobl et al. (2007). Do these importances show the same pattern as the traditional random forest model?

Code

library(partykit)  
library(dplyr)  
cforest\_model <- cforest(y ~ ., data=simulated)  
  
  
# Unconditional importance measure  
varimp(cforest\_model) %>% sort(decreasing = T)  
  
varimp(cforest\_model, conditional=T) %>% sort(decreasing = T)

Response Uninformative predictors V6 - V10 are rated low Importance of duplicate1 correlated with V1 is reduced Other importance of Predictors are also reduced

Part D Question Repeat this process with different tree models, such as boosted trees and Cubist. Does the same pattern occur? Code

library(gbm)  
gbm\_Model <- gbm(y ~ ., data=simulated, distribution='gaussian')  
summary(gbm\_Model)  
  
library(Cubist)  
cubistModel <- cubist(x=simulated[,-(ncol(simulated)-1)], y=simulated$y, committees=100)  
varImp(cubistModel)

Response Top rated predictor is V4 in GBM and V1 in Cubist V6 - V10 scores are very low Duplicate 1 score is 0 in Cubist but higher in GBM.

Hence based on the observations above, the pattern is different across the three models.

## KJ 8.2

Use a simulation to show tree bias with different granularities.

### Part A

#### Question

Fit a random forest model to all of the predictors, then estimate the variable importance scores:

Code

library(caret)  
library(rpart)  
set.seed(755)  
X1 <- rep(1:2, each=100)  
Y <- X1 + rnorm(600, mean=2, sd=4)  
X2 <- rnorm(600, mean=2, sd=4)  
simData <- data.frame(Y=Y, X1=X1, X2=X2)  
fit <- rpart(Y ~ ., data = simData)  
varImp(fit)

Response The simulation is created with two variables  
X1 -> Lesser variance (Sequence of values) X2 -> Higher Variance (Gaussian Random value) Target Variable Y -> X1 + X2 Regression tree is fitted using rpart . Below is the R Code. The difference between two variables is significant and it shows the tree bias between two variables.

## KJ 8.3

In stochastic gradient boosting the bagging fraction and learning rate will govern the construction of the trees as they are guided by the gradient. Although the optimal values of these parameters should be obtained through the tuning process, it is helpful to understand how the magnitudes of these parameters affect magnitudes of variable importance. Figure 8.24 provides the variable importance plots for boosting using two extreme values for the bagging fraction (0.1 and 0.9) and the learning rate (0.1 and 0.9) for the solubility data. The left-hand plot has both parameters set to 0.1, and the right-hand plot has both set to 0.9:

[insert figure 8.24]

Part A Question Why does the model on the right focus its importance on just the first few of predictors, whereas the model on the left spreads importance across more predictors?

Response There could be two reasons for the model on the right spreads importance on the first few predictors and the model on the left adds importance to few more predictors. Bragging Fraction : It represents the data usage in each iteration of trees. The left hand plot has a bragging fraction of 0.1 which is low and only 10% of data is used for random sampling. Whereas , the right hand plot has a bragging fraction of 0.9 which is large and 90% of data is used on each iteration which is almost the full dataset. Since the full data set is approximately used by the model on the right plot, only few predictors got importance but the left plot used partial dataset to get importance over few more predictors. Learning Rate : It means a higher number of predictions are added to the model output. Since the right hand plot has more predictions the correlation is more and hence only the first few predictors were considered significant.

Part B Question Which model do you think would be more predictive of other samples? Response Because of the above explanations the two parameters are crucial in selecting the more predictive model. In my opinion the model with lesser fraction and learning rate would be more predictive than the one with lower values. Hence the left hand plot is more predictive. Part C Question How would increasing interaction depth affect the slope of predictor importance for either model in Fig. 8.24?

Code

library(gbm)  
library(AppliedPredictiveModeling)  
data(solubility)  
grid1 <- expand.grid(n.trees=100, interaction.depth=1, shrinkage=0.1, n.minobsinnode=10)  
gbm1 <- train(x = solTrainXtrans, y = solTrainY, method = 'gbm', tuneGrid = grid1, verbose = FALSE)  
grid2 <- expand.grid(n.trees=100, interaction.depth=10, shrinkage=0.1, n.minobsinnode=10)  
gbm2 <- train(x = solTrainXtrans, y = solTrainY, method = 'gbm', tuneGrid = grid2, verbose = FALSE)  
varImp(gbm1)  
varImp(gbm2)

varImp(gbm1)

varImp(gbm2)

Response Increasing the interaction depth will improve the predictor importance significantly. The below R code will create two GBM models. Looking at two sets of Importance values It is evident that with the interaction depth 10 the importance is spread across several predictors.

KJ 8.7 Refer to Exercises 6.3 and 7.5 which describe a chemical manufacturing process. Use the same data imputation, data splitting, and pre-processing steps as before and train several tree-based models:

Part A

Question Which tree-based regression model gives the optimal resampling and test set performance?

Code

library(AppliedPredictiveModeling)  
#Load the data  
data(ChemicalManufacturingProcess)  
df = data.frame(ChemicalManufacturingProcess)  
for(i in 1:ncol(df)){  
 df[is.na(df[,i]), i] <- median(df[,i], na.rm = TRUE)}  
#Create Train and Test data sets  
bound <- floor((nrow(df)/4)\*3) #define % of training and test set  
df <- df[sample(nrow(df)), ]   
train <- df[1:bound, ]   
test <- df[(bound+1):nrow(df), ]   
sum(is.na(train))  
sum(is.na(test))  
# Create the Random Forest Tree  
rf\_model = randomForest(Yield~ManufacturingProcess32 + ManufacturingProcess36 + ManufacturingProcess09 + ManufacturingProcess13 + BiologicalMaterial02 + BiologicalMaterial06 + BiologicalMaterial03, data=train)

#Predict with the test data  
rf\_test = predict(rf\_model, newdata = test)  
rf\_test\_df = data.frame(rf\_test)  
rf\_test\_df$actual = test$Yield  
rf\_test\_df  
  
rf\_test\_rmse = rmse(rf\_test\_df$actual, rf\_test\_df$rf\_test)  
cat('RMSE of the test data for this model is', rf\_test\_rmse)

#Create the ctree   
ctree\_model = cforest(Yield~ManufacturingProcess32 + ManufacturingProcess36 + ManufacturingProcess09 + ManufacturingProcess13 + BiologicalMaterial02 + BiologicalMaterial06 + BiologicalMaterial03, data=train)

library(Metrics)  
#Predict with the test data  
ctree\_test = predict(ctree\_model, newdata = test)  
ctree\_test\_df = data.frame(ctree\_test)  
ctree\_test\_df$actual = test$Yield  
ctree\_test\_df  
# USes lmyield from 6.3   
ctree\_test\_rmse = rmse(lmyield\_test\_df$actual, lmyield\_test\_df$fit)  
cat('RMSE of the test data for this model is', ctree\_test\_rmse)

Response I used a randomforest and ctree algorithm to fit and predict two models. The RMSE’s were close at 1.14 for Random Forest and 1.28 for cTree, so RandomForest edged out cTree.

Part B Question Which predictors are most important in the optimal tree-based regression model? Do either the biological or process variables dominate the list? How do the top 10 important predictors compare to the top 10 predictors from the optimal linear and nonlinear models? Code

[missing code - fill in to demonstrate response]

Response As shown in the table below, Process32, 36, 09, and 13 are the most important predictors. Clearly the Process variables dominate the list. For the linear model, the Process32, Process36, and BiologicalMaterial02 were the top three features based on p-values. For the non-linear model the Process32, Process09, and Biological12 appear to be the most important features. For the tree based regression models, the Process 32, Process36, and Process 09 top the list. This is interesting because linear model ranked the Process32 and 36 at the top and the non-linear placed Process09 as second highest. The consistently high ranking of Process32, 36, and 09’s feature importance across the linear, non-linear, and tree based models all substantiate that these Process variables are the most important, and that overall Process variables have a greater influence on Yield than Manufacturing variables. But, the final model that actually get used in Production, will most likely include several of the Manufacturing variables.

Part C Question Plot the optimal single tree with the distribution of yield in the terminal nodes. Does this view of the data provide additional knowledge about the biological or process predictors and their relationship with yield? Code

[missing a plot chunk for visual interpretation]

Response

## Market Basket Analysis

I am assigning one simple problem on market basket analysis / recommender systems. Imagine 10000 receipts sitting on your table. Each receipt represents a transaction with items that were purchased. The receipt is a representation of stuff that went into a customer’s basket – and therefore ‘Market Basket Analysis’. That is exactly what the Groceries Data Set contains: a collection of receipts with each line representing 1 receipt and the items purchased. Each line is called a transaction and each column in a row represents an item. Here is the dataset = GroceryDataSet.csv (comma separated file) Your assignment is to use R to mine the data for association rules. You should report support, confidence and lift and your top 10 rules by lift.

### Part A

#### Question

Fit a random forest model to all of the predictors, then estimate the variable importance scores:

library(arules)  
library(arulesViz)  
library(RColorBrewer)  
library(visNetwork)  
library(igraph)  
df <- read.csv(  
 "https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/GroceryDataSet.csv")  
df\_sparse <- read.transactions(  
 "https://raw.githubusercontent.com/palmorezm/msds/main/624/Data/hw\_2\_data/GroceryDataSet.csv",  
 format="basket",sep=",")  
summary(df\_sparse)  
itemFrequencyPlot(df\_sparse,topN=20,type="absolute",col=brewer.pal(8,'Pastel2'), main="Frequently Purchased Products")  
association.rules <- apriori(df\_sparse, parameter = list(supp=0.004, conf=0.3))  
length(association.rules)  
inspect(sort(association.rules, by = 'lift')[1:10])  
subset.rules <- which(colSums(is.subset(association.rules, association.rules)) > 1) # get subset rules in vector  
length(subset.rules)   
subset.association.rules. <- association.rules[-subset.rules] # remove subset rules.  
inspect(sort(subset.association.rules., by = 'lift')[1:10])  
plot(subset.association.rules.,method="two-key plot")  
plot(association.rules)  
top10subRules <- head(subset.association.rules., n = 10, by = "lift")  
plot(top10subRules, method = "graph", engine = "htmlwidget")  
subRules2<-head(subset.association.rules., n=10, by="lift")  
plot(subRules2, method="paracoord")

Code

df <- read.csv("GroceryDataSet.csv")  
df\_sparse <- read.transactions("GroceryDataset.csv",format="basket",sep=",")  
summary(df\_sparse)

itemFrequencyPlot(df\_sparse,topN=20,type="absolute",col=brewer.pal(8,'Pastel2'), main="Frequently Purchased Products")

association.rules <- apriori(df\_sparse, parameter = list(supp=0.004, conf=0.3,maxlen=5))  
length(association.rules)  
inspect(sort(association.rules, by = 'lift')[1:10])

subset.rules <- which(colSums(is.subset(association.rules, association.rules)) > 1) # get subset rules in vector  
length(subset.rules)   
  
subset.association.rules. <- association.rules[-subset.rules] # remove subset rules.  
summary(subset.association.rules.)  
inspect(sort(subset.association.rules., by = 'lift')[1:10])

plot(subset.association.rules., method="two-key plot")

top10subRules <- head(subset.association.rules., n = 10, by = "lift")  
plot(top10subRules, method = "graph", engine = "htmlwidget")

subRules2<-head(subset.association.rules., n=10, by="lift")  
plot(subRules2, method="paracoord")

Response Association Rules mining in R is achieved by usage of apriori function available under arules package. Association rules mining is a two step process. Frequent Itemset Generation Rules Generation Once the rules are generated the top 10 rules can be classified based on the output. The below steps illustrate the step-by-step approach for mining association rules. Load and Summarize the dataset. Below R code loads the R Sparse matrix and displays summary. Find Frequently purchased products. Below R code plots the top 20 purchased products.

Association Rules generation: The association rules are generated using apriori function by passing the sparse matrix generated in the previous step.

Choice of Minimum Support Supp = (Products purchased at least 5 times a day)/total number of transactions Supp = (6\*7)/nrow(df) = 0.004

Choice of Confidence : After running trial and error on the different combinations the confidence of 0.3 was generating a decent number of rules as well as the combination.

There are a total of 735 rules generated and the top 10 rules are displayed above. From the results it appears that the highest association is created for liquor and bottled beer with the lift value of 5.24

Redundant Rules Cleanup: Lets cleanup the redundant rules by using the subset function as per below R code.

The highest association did not change after removing redundancy, but the other rules have changed. Liquor and bottled beer are the most dependent purchases with the lift value of 5.24.

Visualization of Association Rules

Scatter Plot Below R Code creates a scatter plot for the final set of rules.

Graph Based Visual: Below R Code creates a graph plot for the final set of rules