Homework 8

**Name: Surabhi Asati**

**Stat 6620**

**Section 1**

##### Chapter 7: Neural Networks and Support Vector Machines ——————-

# Question 1

##### —– Support Vector Machines ——————-

## Example: Optical Character Recognition —-

## Step 1: Collecting the data ----  
# read in data and examine structure  
letters <- read.csv("letterdata.csv")  
  
#Step 2: Exploring and preparing the data ----  
str(letters)

## 'data.frame': 20000 obs. of 17 variables:  
## $ letter: Factor w/ 26 levels "A","B","C","D",..: 20 9 4 14 7 19 2 1 10 13 ...  
## $ xbox : int 2 5 4 7 2 4 4 1 2 11 ...  
## $ ybox : int 8 12 11 11 1 11 2 1 2 15 ...  
## $ width : int 3 3 6 6 3 5 5 3 4 13 ...  
## $ height: int 5 7 8 6 1 8 4 2 4 9 ...  
## $ onpix : int 1 2 6 3 1 3 4 1 2 7 ...  
## $ xbar : int 8 10 10 5 8 8 8 8 10 13 ...  
## $ ybar : int 13 5 6 9 6 8 7 2 6 2 ...  
## $ x2bar : int 0 5 2 4 6 6 6 2 2 6 ...  
## $ y2bar : int 6 4 6 6 6 9 6 2 6 2 ...  
## $ xybar : int 6 13 10 4 6 5 7 8 12 12 ...  
## $ x2ybar: int 10 3 3 4 5 6 6 2 4 1 ...  
## $ xy2bar: int 8 9 7 10 9 6 6 8 8 9 ...  
## $ xedge : int 0 2 3 6 1 0 2 1 1 8 ...  
## $ xedgey: int 8 8 7 10 7 8 8 6 6 1 ...  
## $ yedge : int 0 4 3 2 5 9 7 2 1 1 ...  
## $ yedgex: int 8 10 9 8 10 7 10 7 7 8 ...

# divide into training and test data  
letters\_train <- letters[1:16000, ]  
letters\_test <- letters[16001:20000, ]  
  
## Step 3: Training a model on the data ----  
# begin by training a simple linear SVM  
library(kernlab)  
letter\_classifier <- ksvm(letter ~ ., data = letters\_train,  
 kernel = "vanilladot")

## Setting default kernel parameters

# look at basic information about the model  
letter\_classifier

## Support Vector Machine object of class "ksvm"   
##   
## SV type: C-svc (classification)   
## parameter : cost C = 1   
##   
## Linear (vanilla) kernel function.   
##   
## Number of Support Vectors : 7037   
##   
## Training error : 0.130062

## Step 4: Evaluating model performance ----  
# predictions on testing dataset  
letter\_predictions <- predict(letter\_classifier, letters\_test)  
  
head(letter\_predictions)

## [1] U N V X N H  
## Levels: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

table(letters\_test$letter, letter\_predictions)

## letter\_predictions  
## A B C D E F G H I J K L M N O P Q  
## A 144 0 0 2 0 0 1 0 0 0 1 0 0 0 1 0 0  
## B 0 121 0 2 0 0 1 0 1 1 1 0 0 0 0 0 0  
## C 0 0 120 0 5 0 2 0 0 0 9 0 1 0 2 0 0  
## D 0 5 0 156 0 0 1 1 0 0 0 0 1 0 1 1 0  
## E 0 2 4 0 127 0 9 0 0 0 0 2 0 0 0 0 0  
## F 0 0 0 1 3 138 2 1 1 1 0 0 0 1 0 2 0  
## G 0 1 10 3 1 2 123 0 0 0 2 1 1 0 1 1 8  
## H 0 2 2 10 1 2 2 102 0 2 5 1 1 1 2 0 2  
## I 0 0 2 4 0 6 0 0 141 5 0 0 0 0 0 0 0  
## J 1 0 0 3 0 0 0 2 8 128 0 0 0 0 1 0 0  
## K 0 1 1 4 3 0 1 3 0 0 118 0 0 0 0 0 0  
## L 0 0 3 3 4 0 2 2 0 0 0 133 0 0 0 0 3  
## M 1 1 0 0 0 0 1 3 0 0 0 0 135 0 0 0 0  
## N 2 0 0 5 0 0 0 4 0 0 2 0 4 145 1 0 0  
## O 2 0 2 5 0 0 1 20 0 1 0 0 0 0 99 2 3  
## P 0 2 0 3 0 16 2 0 1 1 1 0 0 0 3 130 1  
## Q 5 2 0 1 2 0 8 2 0 3 0 1 0 0 3 0 124  
## R 0 3 0 4 0 0 2 3 0 0 7 0 0 3 0 0 0  
## S 1 5 0 0 10 3 4 0 3 2 0 5 0 0 0 0 5  
## T 1 0 0 0 0 0 3 3 0 0 1 0 0 0 0 0 0  
## U 1 0 0 0 0 0 0 0 0 0 3 0 3 1 3 0 0  
## V 0 2 0 0 0 1 0 2 0 0 0 0 0 0 0 0 0  
## W 1 0 0 0 0 0 0 0 0 0 0 0 8 2 0 0 0  
## X 0 1 0 3 2 1 1 0 5 1 5 0 0 0 0 0 0  
## Y 0 0 0 3 0 2 0 1 1 0 0 0 0 0 0 1 2  
## Z 1 0 0 1 3 0 0 0 1 6 0 1 0 0 0 0 0  
## letter\_predictions  
## R S T U V W X Y Z  
## A 0 1 0 1 0 0 0 3 2  
## B 7 1 0 0 0 0 1 0 0  
## C 0 0 0 3 0 0 0 0 0  
## D 0 0 0 1 0 0 0 0 0  
## E 1 1 3 0 0 0 2 0 1  
## F 0 0 2 0 1 0 0 0 0  
## G 3 3 0 0 3 1 0 0 0  
## H 8 0 0 2 4 0 1 1 0  
## I 0 1 0 0 0 0 3 0 3  
## J 0 1 0 0 0 0 0 0 4  
## K 13 0 1 0 0 0 1 0 0  
## L 0 1 0 0 0 0 6 0 0  
## M 0 0 0 0 1 2 0 0 0  
## N 1 0 0 0 2 0 0 0 0  
## O 1 0 0 1 1 0 1 0 0  
## P 1 0 0 0 0 0 0 7 0  
## Q 0 14 0 0 3 0 0 0 0  
## R 138 0 0 0 1 0 0 0 0  
## S 0 101 3 0 0 0 1 0 18  
## T 1 3 133 0 0 0 0 3 3  
## U 0 0 1 152 0 4 0 0 0  
## V 1 0 0 0 126 4 0 0 0  
## W 0 0 0 0 1 127 0 0 0  
## X 0 2 0 1 0 0 137 0 0  
## Y 0 0 2 1 4 0 1 127 0  
## Z 0 10 2 0 0 0 1 0 132

# look only at agreement vs. non-agreement  
# construct a vector of TRUE/FALSE indicating correct/incorrect predictions  
agreement <- letter\_predictions == letters\_test$letter  
table(agreement)

## agreement  
## FALSE TRUE   
## 643 3357

prop.table(table(agreement))

## agreement  
## FALSE TRUE   
## 0.16075 0.83925

## Step 5: Improving model performance ----  
set.seed(12345)  
letter\_classifier\_rbf <- ksvm(letter ~ ., data = letters\_train, kernel = "rbfdot")  
letter\_predictions\_rbf <- predict(letter\_classifier\_rbf, letters\_test)  
  
table(letters\_test$letter, letter\_predictions\_rbf)

## letter\_predictions\_rbf  
## A B C D E F G H I J K L M N O P Q  
## A 151 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0  
## B 0 128 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0  
## C 0 0 133 0 2 0 2 0 0 0 1 0 0 0 2 0 0  
## D 0 3 0 161 0 0 0 1 0 0 0 0 0 0 0 0 0  
## E 0 0 3 0 137 0 8 0 0 0 0 0 0 0 0 0 0  
## F 0 1 0 0 2 148 0 0 0 0 0 0 0 2 0 0 0  
## G 0 0 1 2 0 0 154 2 0 0 0 1 1 0 0 0 0  
## H 0 2 0 8 0 0 2 126 0 0 4 0 1 0 0 0 1  
## I 0 0 2 2 0 3 0 0 151 3 0 0 0 0 0 1 0  
## J 0 0 0 3 1 0 0 1 3 136 0 0 0 0 1 0 0  
## K 0 0 0 1 0 0 0 2 0 0 132 0 0 0 0 0 0  
## L 0 1 1 0 4 0 2 1 0 0 0 142 0 0 0 0 0  
## M 0 2 0 0 0 0 2 1 0 0 0 0 138 0 0 0 0  
## N 0 1 0 1 0 0 0 3 0 0 1 0 1 150 5 0 0  
## O 0 0 0 1 0 0 2 0 0 0 0 0 0 0 129 0 3  
## P 0 2 0 3 1 11 1 1 0 0 0 0 0 0 2 141 3  
## Q 3 1 0 1 0 0 0 1 0 0 0 0 0 0 4 0 158  
## R 0 3 0 3 0 0 0 0 0 0 3 0 0 2 0 0 0  
## S 0 3 0 0 2 1 0 0 0 0 0 1 0 0 0 0 0  
## T 1 0 0 2 1 0 2 2 0 0 0 0 0 0 0 0 0  
## U 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0  
## V 0 3 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0  
## W 0 1 0 0 0 0 0 0 0 0 0 0 2 1 0 0 0  
## X 0 1 0 2 0 0 0 0 1 0 2 0 0 0 0 0 0  
## Y 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0  
## Z 0 0 0 0 2 0 0 0 0 3 0 0 0 0 0 0 0  
## letter\_predictions\_rbf  
## R S T U V W X Y Z  
## A 0 0 0 0 0 0 0 4 0  
## B 3 2 0 0 0 0 1 0 0  
## C 1 0 0 1 0 0 0 0 0  
## D 1 0 0 1 0 0 0 0 0  
## E 0 0 0 0 0 0 1 0 3  
## F 0 0 0 0 0 0 0 0 0  
## G 2 0 0 0 0 1 0 0 0  
## H 5 0 0 1 0 0 0 1 0  
## I 0 1 0 0 0 0 0 0 2  
## J 0 2 0 0 0 0 0 0 1  
## K 9 0 0 0 0 0 2 0 0  
## L 1 1 0 0 0 0 4 0 0  
## M 0 0 0 0 0 1 0 0 0  
## N 3 0 0 0 1 0 0 0 0  
## O 2 0 0 0 0 2 0 0 0  
## P 1 0 0 0 0 0 0 2 0  
## Q 0 0 0 0 0 0 0 0 0  
## R 150 0 0 0 0 0 0 0 0  
## S 0 152 0 0 0 0 1 0 1  
## T 1 0 140 0 0 0 1 1 0  
## U 0 0 0 161 2 3 0 0 0  
## V 0 0 0 0 131 0 0 0 0  
## W 0 0 0 0 0 135 0 0 0  
## X 0 0 0 0 0 0 153 0 0  
## Y 0 0 1 1 1 0 1 138 0  
## Z 0 2 0 0 0 0 1 0 150

agreement\_rbf <- letter\_predictions\_rbf == letters\_test$letter  
table(agreement\_rbf)

## agreement\_rbf  
## FALSE TRUE   
## 275 3725

prop.table(table(agreement\_rbf))

## agreement\_rbf  
## FALSE TRUE   
## 0.06875 0.93125

# Question 2

##### Chapter 9: Clustering with k-means ——————-

## Example: Finding Teen Market Segments —-

## Step 1: Collecting the data ----  
teens <- read.csv("http://www.sci.csueastbay.edu/~esuess/classes/Statistics\_6620/Presentations/ml12/snsdata.csv")  
  
  
## Step 2: Exploring and preparing the data ----  
str(teens)

## 'data.frame': 30000 obs. of 40 variables:  
## $ gradyear : int 2006 2006 2006 2006 2006 2006 2006 2006 2006 2006 ...  
## $ gender : Factor w/ 2 levels "F","M": 2 1 2 1 NA 1 1 2 1 1 ...  
## $ age : num 19 18.8 18.3 18.9 19 ...  
## $ friends : int 7 0 69 0 10 142 72 17 52 39 ...  
## $ basketball : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ football : int 0 1 1 0 0 0 0 0 0 0 ...  
## $ soccer : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ softball : int 0 0 0 0 0 0 0 1 0 0 ...  
## $ volleyball : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ swimming : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ cheerleading: int 0 0 0 0 0 0 0 0 0 0 ...  
## $ baseball : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ tennis : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ sports : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ cute : int 0 1 0 1 0 0 0 0 0 1 ...  
## $ sex : int 0 0 0 0 1 1 0 2 0 0 ...  
## $ sexy : int 0 0 0 0 0 0 0 1 0 0 ...  
## $ hot : int 0 0 0 0 0 0 0 0 0 1 ...  
## $ kissed : int 0 0 0 0 5 0 0 0 0 0 ...  
## $ dance : int 1 0 0 0 1 0 0 0 0 0 ...  
## $ band : int 0 0 2 0 1 0 1 0 0 0 ...  
## $ marching : int 0 0 0 0 0 1 1 0 0 0 ...  
## $ music : int 0 2 1 0 3 2 0 1 0 1 ...  
## $ rock : int 0 2 0 1 0 0 0 1 0 1 ...  
## $ god : int 0 1 0 0 1 0 0 0 0 6 ...  
## $ church : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ jesus : int 0 0 0 0 0 0 0 0 0 2 ...  
## $ bible : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ hair : int 0 6 0 0 1 0 0 0 0 1 ...  
## $ dress : int 0 4 0 0 0 1 0 0 0 0 ...  
## $ blonde : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ mall : int 0 1 0 0 0 0 2 0 0 0 ...  
## $ shopping : int 0 0 0 0 2 1 0 0 0 1 ...  
## $ clothes : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ hollister : int 0 0 0 0 0 0 2 0 0 0 ...  
## $ abercrombie : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ die : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ death : int 0 0 1 0 0 0 0 0 0 0 ...  
## $ drunk : int 0 0 0 0 1 1 0 0 0 0 ...  
## $ drugs : int 0 0 0 0 1 0 0 0 0 0 ...

# look at missing data for female variable  
table(teens$gender)

##   
## F M   
## 22054 5222

table(teens$gender, useNA = "ifany")

##   
## F M <NA>   
## 22054 5222 2724

# look at missing data for age variable  
summary(teens$age)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 3.086 16.312 17.287 17.994 18.259 106.927 5086

# eliminate age outliers  
teens$age <- ifelse(teens$age >= 13 & teens$age < 20,  
 teens$age, NA)  
  
summary(teens$age)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 13.03 16.30 17.27 17.25 18.22 20.00 5523

# reassign missing gender values to "unknown"  
teens$female <- ifelse(teens$gender == "F" &  
 !is.na(teens$gender), 1, 0)  
teens$no\_gender <- ifelse(is.na(teens$gender), 1, 0)  
  
# check our recoding work  
table(teens$gender, useNA = "ifany")

##   
## F M <NA>   
## 22054 5222 2724

table(teens$female, useNA = "ifany")

##   
## 0 1   
## 7946 22054

table(teens$no\_gender, useNA = "ifany")

##   
## 0 1   
## 27276 2724

# finding the mean age by cohort  
mean(teens$age) # doesn't work

## [1] NA

mean(teens$age, na.rm = TRUE) # works

## [1] 17.25243

# age by cohort  
aggregate(data = teens, age ~ gradyear, mean, na.rm = TRUE)

## gradyear age  
## 1 2006 18.65586  
## 2 2007 17.70617  
## 3 2008 16.76770  
## 4 2009 15.81957

# create a vector with the average age for each gradyear, repeated by person  
ave\_age <- ave(teens$age, teens$gradyear,  
 FUN = function(x) mean(x, na.rm = TRUE))  
  
  
teens$age <- ifelse(is.na(teens$age), ave\_age, teens$age)  
  
# check the summary results to ensure missing values are eliminated  
summary(teens$age)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 13.03 16.28 17.24 17.24 18.21 20.00

## Step 3: Training a model on the data ----  
interests <- teens[5:40]  
interests\_z <- as.data.frame(lapply(interests, scale))  
  
set.seed(2345)  
teen\_clusters <- kmeans(interests\_z, 5)  
  
## Step 4: Evaluating model performance ----  
# look at the size of the clusters  
teen\_clusters$size

## [1] 871 600 5981 1034 21514

# look at the cluster centers  
teen\_clusters$centers

## basketball football soccer softball volleyball swimming  
## 1 0.16001227 0.2364174 0.10385512 0.07232021 0.18897158 0.23970234  
## 2 -0.09195886 0.0652625 -0.09932124 -0.01739428 -0.06219308 0.03339844  
## 3 0.52755083 0.4873480 0.29778605 0.37178877 0.37986175 0.29628671  
## 4 0.34081039 0.3593965 0.12722250 0.16384661 0.11032200 0.26943332  
## 5 -0.16695523 -0.1641499 -0.09033520 -0.11367669 -0.11682181 -0.10595448  
## cheerleading baseball tennis sports cute  
## 1 0.3931445 0.02993479 0.13532387 0.10257837 0.37884271  
## 2 -0.1101103 -0.11487510 0.04062204 -0.09899231 -0.03265037  
## 3 0.3303485 0.35231971 0.14057808 0.32967130 0.54442929  
## 4 0.1856664 0.27527088 0.10980958 0.79711920 0.47866008  
## 5 -0.1136077 -0.10918483 -0.05097057 -0.13135334 -0.18878627  
## sex sexy hot kissed dance band  
## 1 0.020042068 0.11740551 0.41389104 0.06787768 0.22780899 -0.10257102  
## 2 -0.042486141 -0.04329091 -0.03812345 -0.04554933 0.04573186 4.06726666  
## 3 0.002913623 0.24040196 0.38551819 -0.03356121 0.45662534 -0.02120728  
## 4 2.028471066 0.51266080 0.31708549 2.97973077 0.45535061 0.38053621  
## 5 -0.097928345 -0.09501817 -0.13810894 -0.13535855 -0.15932739 -0.12167214  
## marching music rock god church jesus  
## 1 -0.10942590 0.1378306 0.05905951 0.03651755 -0.00709374 0.01458533  
## 2 5.25757242 0.4981238 0.15963917 0.09283620 0.06414651 0.04801941  
## 3 -0.10880541 0.2844999 0.21436936 0.35014919 0.53739806 0.27843424  
## 4 -0.02014608 1.1367885 1.21013948 0.41679142 0.16627797 0.12988313  
## 5 -0.11098063 -0.1532006 -0.12460034 -0.12144246 -0.15889274 -0.08557822  
## bible hair dress blonde mall shopping  
## 1 -0.03692278 0.43807926 0.14905267 0.06137340 0.60368108 0.79806891  
## 2 0.05863810 -0.04484083 0.07201611 -0.01146396 -0.08724304 -0.03865318  
## 3 0.22990963 0.23612853 0.39407628 0.03471458 0.48318495 0.66327838  
## 4 0.08478769 2.55623737 0.53852195 0.36134138 0.62256686 0.27101815  
## 5 -0.06813159 -0.20498730 -0.14348036 -0.02918252 -0.18625656 -0.22865236  
## clothes hollister abercrombie die death  
## 1 0.5651537331 4.1521844 3.96493810 0.043475966 0.09857501  
## 2 -0.0003526292 -0.1678300 -0.14129577 0.009447317 0.05135888  
## 3 0.3759725120 -0.0553846 -0.07417839 0.037989066 0.11972190  
## 4 1.2306917174 0.1610784 0.26324494 1.712181870 0.93631312  
## 5 -0.1865419798 -0.1557662 -0.14861104 -0.094875180 -0.08370729  
## drunk drugs  
## 1 0.035614771 0.03443294  
## 2 -0.086773220 -0.06878491  
## 3 -0.009688746 -0.05973769  
## 4 1.897388200 2.73326605  
## 5 -0.087520105 -0.11423381

## Step 5: Improving model performance ----  
# apply the cluster IDs to the original data frame  
teens$cluster <- teen\_clusters$cluster  
  
# look at the first five records  
teens[1:5, c("cluster", "gender", "age", "friends")]

## cluster gender age friends  
## 1 5 M 18.982 7  
## 2 3 F 18.801 0  
## 3 5 M 18.335 69  
## 4 5 F 18.875 0  
## 5 4 <NA> 18.995 10

# mean age by cluster  
aggregate(data = teens, age ~ cluster, mean)

## cluster age  
## 1 1 16.86497  
## 2 2 17.39037  
## 3 3 17.07656  
## 4 4 17.11957  
## 5 5 17.29849

# proportion of females by cluster  
aggregate(data = teens, female ~ cluster, mean)

## cluster female  
## 1 1 0.8381171  
## 2 2 0.7250000  
## 3 3 0.8378198  
## 4 4 0.8027079  
## 5 5 0.6994515

# mean number of friends by cluster  
aggregate(data = teens, friends ~ cluster, mean)

## cluster friends  
## 1 1 41.43054  
## 2 2 32.57333  
## 3 3 37.16185  
## 4 4 30.50290  
## 5 5 27.70052

# Question 3

##### Chapter 8: Association Rules ——————-

## Example: Identifying Frequently-Purchased Groceries —-

library(arules)

## Warning: package 'arules' was built under R version 3.4.4

## Loading required package: Matrix

##   
## Attaching package: 'arules'

## The following object is masked from 'package:kernlab':  
##   
## size

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

library(arulesViz)

## Loading required package: grid

library(DT)  
  
## Step 1: Collecting the data ----  
  
# load the grocery data into a sparse matrix  
#fn in arulesviz to read transcation so read fn is diff.........................  
groceries <- read.transactions("groceries.csv", sep = ",")  
  
## Step 2: Exploring and preparing the data ----  
  
summary(groceries)

## transactions as itemMatrix in sparse format with  
## 9835 rows (elements/itemsets/transactions) and  
## 169 columns (items) and a density of 0.02609146   
##   
## most frequent items:  
## whole milk other vegetables rolls/buns soda   
## 2513 1903 1809 1715   
## yogurt (Other)   
## 1372 34055   
##   
## element (itemset/transaction) length distribution:  
## sizes  
## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15   
## 2159 1643 1299 1005 855 645 545 438 350 246 182 117 78 77 55   
## 16 17 18 19 20 21 22 23 24 26 27 28 29 32   
## 46 29 14 14 9 11 4 6 1 1 1 1 3 1   
##   
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1.000 2.000 3.000 4.409 6.000 32.000   
##   
## includes extended item information - examples:  
## labels  
## 1 Instant food products  
## 2 UHT-milk  
## 3 abrasive cleaner

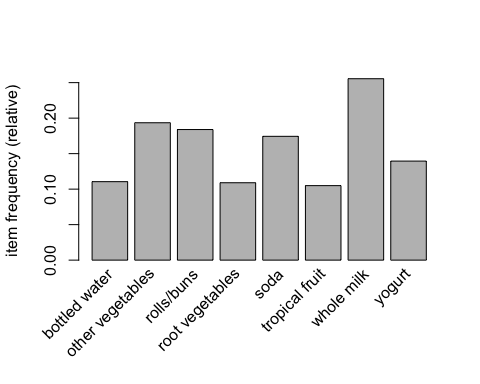
# look at the first five transactions  
inspect(groceries[1:5])

## items   
## [1] {citrus fruit,   
## margarine,   
## ready soups,   
## semi-finished bread}   
## [2] {coffee,   
## tropical fruit,   
## yogurt}   
## [3] {whole milk}   
## [4] {cream cheese,   
## meat spreads,   
## pip fruit,   
## yogurt}   
## [5] {condensed milk,   
## long life bakery product,  
## other vegetables,   
## whole milk}

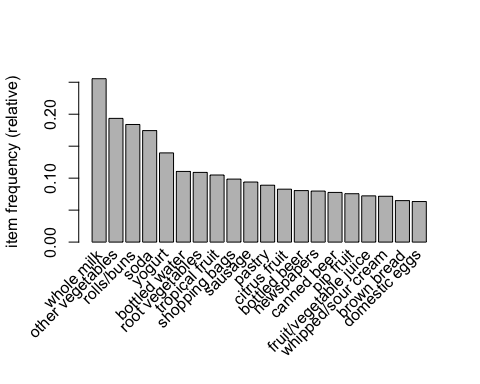
# examine the frequency of items  
itemFrequency(groceries[, 1:3])

## Instant food products UHT-milk abrasive cleaner   
## 0.008032537 0.033451957 0.003558719

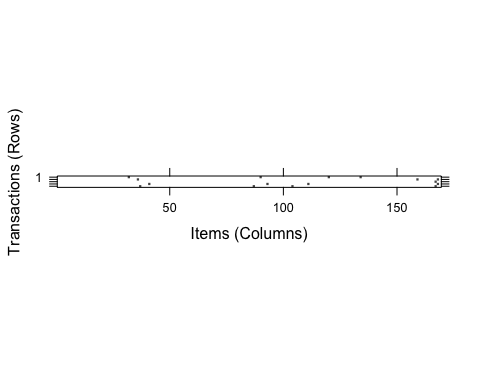
# plot the frequency of items  
itemFrequencyPlot(groceries, support = 0.1)



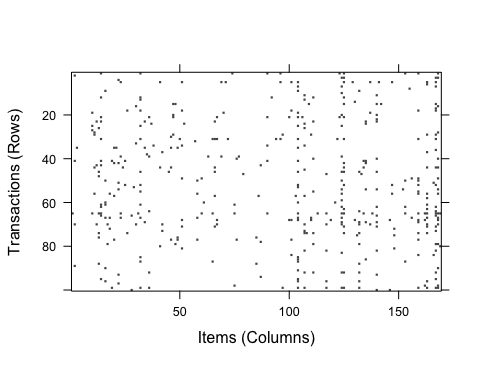
itemFrequencyPlot(groceries, topN = 20)



# a visualization of the sparse matrix for the first five transactions  
image(groceries[1:5])



# visualization of a random sample of 100 transactions  
image(sample(groceries, 100))



## Step 3: Training a model on the data ----  
  
# default settings result in zero rules learned  
apriori(groceries)

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.8 0.1 1 none FALSE TRUE 5 0.1 1  
## maxlen target ext  
## 10 rules FALSE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 983   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].  
## sorting and recoding items ... [8 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 done [0.00s].  
## writing ... [0 rule(s)] done [0.00s].  
## creating S4 object ... done [0.00s].

## set of 0 rules

#default support is too high s=10%---huge, c=80%.............so change it   
# set better support and confidence levels to learn more rules  
groceryrules <- apriori(groceries, parameter = list(support =  
 0.006, confidence = 0.25, minlen = 2))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.25 0.1 1 none FALSE TRUE 5 0.006 2  
## maxlen target ext  
## 10 rules FALSE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 59   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].  
## sorting and recoding items ... [109 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4 done [0.00s].  
## writing ... [463 rule(s)] done [0.00s].  
## creating S4 object ... done [0.01s].

groceryrules

## set of 463 rules

#now there are 463 rules.........  
## Step 4: Evaluating model performance ----  
# summary of grocery association rules  
summary(groceryrules)

## set of 463 rules  
##   
## rule length distribution (lhs + rhs):sizes  
## 2 3 4   
## 150 297 16   
##   
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 2.000 2.000 3.000 2.711 3.000 4.000   
##   
## summary of quality measures:  
## support confidence lift count   
## Min. :0.006101 Min. :0.2500 Min. :0.9932 Min. : 60.0   
## 1st Qu.:0.007117 1st Qu.:0.2971 1st Qu.:1.6229 1st Qu.: 70.0   
## Median :0.008744 Median :0.3554 Median :1.9332 Median : 86.0   
## Mean :0.011539 Mean :0.3786 Mean :2.0351 Mean :113.5   
## 3rd Qu.:0.012303 3rd Qu.:0.4495 3rd Qu.:2.3565 3rd Qu.:121.0   
## Max. :0.074835 Max. :0.6600 Max. :3.9565 Max. :736.0   
##   
## mining info:  
## data ntransactions support confidence  
## groceries 9835 0.006 0.25

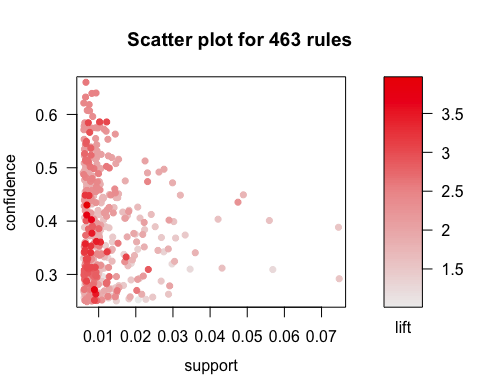
# look at the first three rules  
inspect(groceryrules[1:3])

## lhs rhs support confidence lift   
## [1] {potted plants} => {whole milk} 0.006914082 0.4000000 1.565460  
## [2] {pasta} => {whole milk} 0.006100661 0.4054054 1.586614  
## [3] {herbs} => {root vegetables} 0.007015760 0.4312500 3.956477  
## count  
## [1] 68   
## [2] 60   
## [3] 69

# with a data.table  
  
inspectDT(groceryrules)

# read about the arulesViz package https://cran.r-project.org/web/packages/arulesViz/vignettes/arulesViz.pdf  
  
plot(groceryrules)

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.

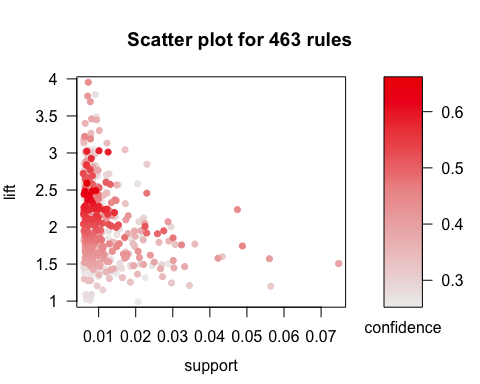


head(quality(groceryrules))

## support confidence lift count  
## 1 0.006914082 0.4000000 1.565460 68  
## 2 0.006100661 0.4054054 1.586614 60  
## 3 0.007015760 0.4312500 3.956477 69  
## 4 0.007727504 0.4750000 2.454874 76  
## 5 0.007727504 0.4750000 1.858983 76  
## 6 0.007015760 0.4233129 1.656698 69

plot(groceryrules, measure = c("support", "lift"), shading = "confidence")

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.



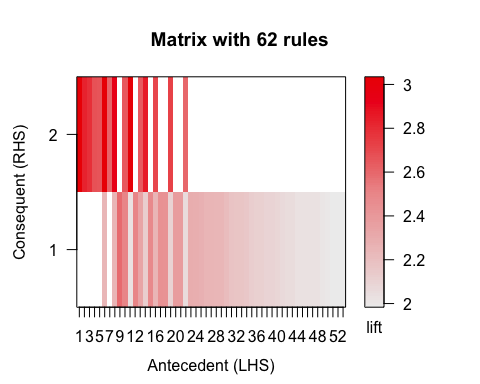
plot(groceryrules, method = "two-key plot")

## To reduce overplotting, jitter is added! Use jitter = 0 to prevent jitter.



subrules <- groceryrules[quality(groceryrules)$confidence > 0.5]  
  
plot(subrules, method = "matrix", measure = "lift")

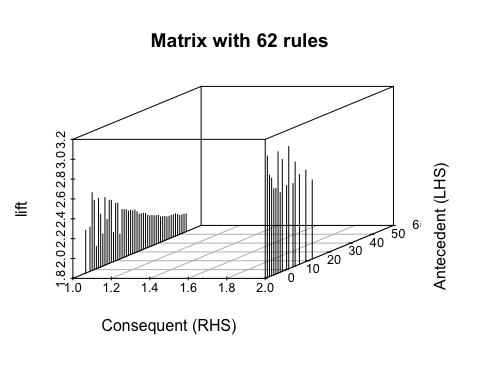
## Itemsets in Antecedent (LHS)  
## [1] "{root vegetables,tropical fruit,whole milk}"   
## [2] "{onions,whole milk}"   
## [3] "{root vegetables,whole milk,yogurt}"   
## [4] "{root vegetables,shopping bags}"   
## [5] "{pork,root vegetables}"   
## [6] "{root vegetables,tropical fruit}"   
## [7] "{tropical fruit,whole milk,yogurt}"   
## [8] "{tropical fruit,whipped/sour cream}"   
## [9] "{butter,whipped/sour cream}"   
## [10] "{butter,root vegetables}"   
## [11] "{citrus fruit,root vegetables}"   
## [12] "{butter,yogurt}"   
## [13] "{domestic eggs,root vegetables}"   
## [14] "{fruit/vegetable juice,root vegetables}"   
## [15] "{curd,tropical fruit}"   
## [16] "{pip fruit,root vegetables}"   
## [17] "{butter,tropical fruit}"   
## [18] "{other vegetables,tropical fruit,yogurt}"   
## [19] "{frozen vegetables,root vegetables}"   
## [20] "{domestic eggs,tropical fruit}"   
## [21] "{other vegetables,root vegetables,yogurt}"   
## [22] "{rolls/buns,root vegetables}"   
## [23] "{other vegetables,sugar}"   
## [24] "{curd,yogurt}"   
## [25] "{citrus fruit,whipped/sour cream}"   
## [26] "{curd,other vegetables}"   
## [27] "{butter,other vegetables}"   
## [28] "{other vegetables,root vegetables,tropical fruit}"  
## [29] "{curd,root vegetables}"   
## [30] "{root vegetables,yogurt}"   
## [31] "{frankfurter,yogurt}"   
## [32] "{root vegetables,whipped/sour cream}"   
## [33] "{domestic eggs,other vegetables}"   
## [34] "{pork,rolls/buns}"   
## [35] "{frozen vegetables,other vegetables}"   
## [36] "{domestic eggs,yogurt}"   
## [37] "{margarine,rolls/buns}"   
## [38] "{rolls/buns,whipped/sour cream}"   
## [39] "{cream cheese,yogurt}"   
## [40] "{pip fruit,yogurt}"   
## [41] "{whipped/sour cream,yogurt}"   
## [42] "{baking powder}"   
## [43] "{beef,yogurt}"   
## [44] "{sausage,tropical fruit}"   
## [45] "{other vegetables,pip fruit}"   
## [46] "{tropical fruit,yogurt}"   
## [47] "{pastry,yogurt}"   
## [48] "{root vegetables,sausage}"   
## [49] "{other vegetables,yogurt}"   
## [50] "{other vegetables,rolls/buns,root vegetables}"   
## [51] "{pastry,tropical fruit}"   
## [52] "{other vegetables,whipped/sour cream}"   
## [53] "{fruit/vegetable juice,yogurt}"   
## Itemsets in Consequent (RHS)  
## [1] "{whole milk}" "{other vegetables}"



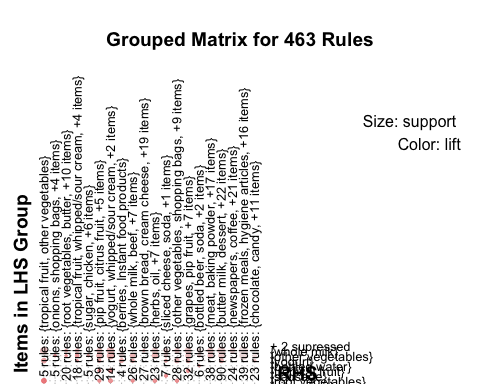
plot(subrules, method = "matrix3D", measure = "lift")

## Warning in plot.rules(subrules, method = "matrix3D", measure = "lift"):  
## method 'matrix3D' is deprecated use method 'matrix' with engine '3d'

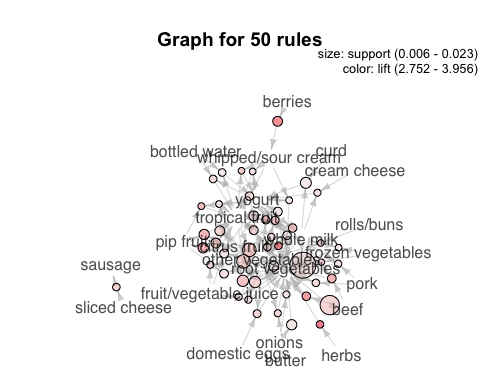
## Itemsets in Antecedent (LHS)  
## Itemsets in Consequent (RHS)  
## [1] "{whole milk}" "{other vegetables}"



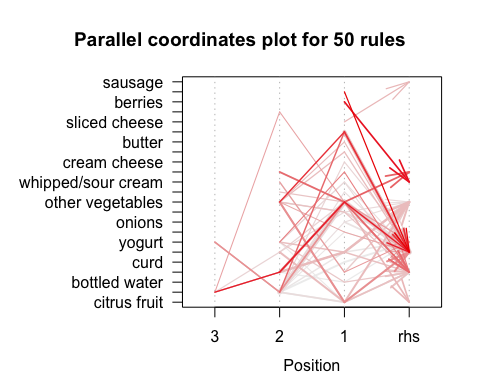
plot(groceryrules, method = "grouped")



subrules2 <- head(groceryrules, n = 50, by = "lift")  
  
plot(subrules2, method = "graph")



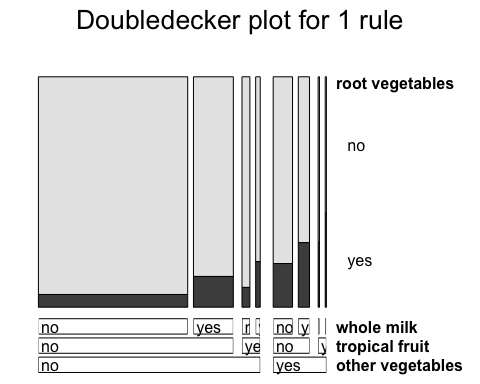
plot(subrules2, method = "paracoord")



oneRule <- sample(groceryrules, 1)  
  
inspect(oneRule)

## lhs rhs support confidence lift count  
## [1] {other vegetables,   
## tropical fruit,   
## whole milk} => {root vegetables} 0.00701576 0.4107143 3.768074 69

plot(oneRule, method = "doubledecker", data = groceries)



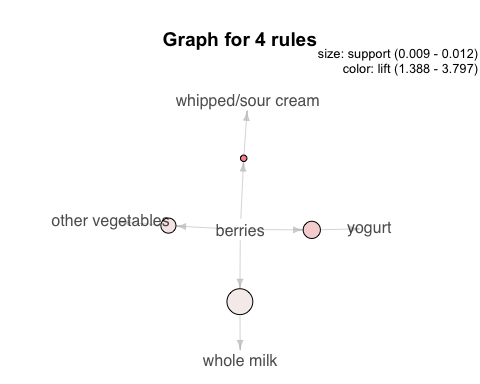
## Step 5: Improving model performance ----  
  
# sorting grocery rules by lift  
inspect(sort(groceryrules, by = "lift")[1:5])

## lhs rhs support confidence lift count  
## [1] {herbs} => {root vegetables} 0.007015760 0.4312500 3.956477 69  
## [2] {berries} => {whipped/sour cream} 0.009049314 0.2721713 3.796886 89  
## [3] {other vegetables,   
## tropical fruit,   
## whole milk} => {root vegetables} 0.007015760 0.4107143 3.768074 69  
## [4] {beef,   
## other vegetables} => {root vegetables} 0.007930859 0.4020619 3.688692 78  
## [5] {other vegetables,   
## tropical fruit} => {pip fruit} 0.009456024 0.2634561 3.482649 93

# finding subsets of rules containing any berry items  
berryrules <- subset(groceryrules, items %in% "berries")  
  
inspect(berryrules)

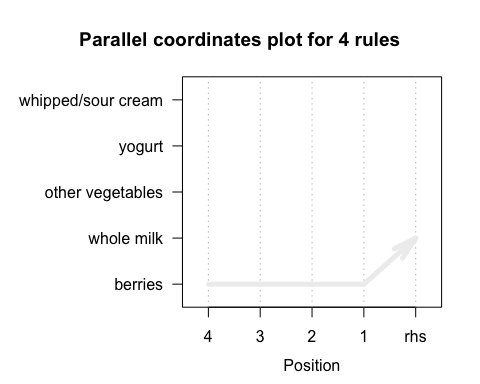
## lhs rhs support confidence lift   
## [1] {berries} => {whipped/sour cream} 0.009049314 0.2721713 3.796886  
## [2] {berries} => {yogurt} 0.010574479 0.3180428 2.279848  
## [3] {berries} => {other vegetables} 0.010269446 0.3088685 1.596280  
## [4] {berries} => {whole milk} 0.011794611 0.3547401 1.388328  
## count  
## [1] 89   
## [2] 104   
## [3] 101   
## [4] 116

plot(berryrules, method = "graph")



plot(berryrules, method = "paracoord")

## Warning in cbind(pl, pr): number of rows of result is not a multiple of  
## vector length (arg 2)



# writing the rules to a CSV file  
write(groceryrules, file = "groceryrules.csv",  
 sep = ",", quote = TRUE, row.names = FALSE)  
  
# converting the rule set to a data frame  
groceryrules\_df <- as(groceryrules, "data.frame")  
  
str(groceryrules\_df)

## 'data.frame': 463 obs. of 5 variables:  
## $ rules : Factor w/ 463 levels "{baking powder} => {other vegetables}",..: 340 302 207 206 208 341 402 21 139 140 ...  
## $ support : num 0.00691 0.0061 0.00702 0.00773 0.00773 ...  
## $ confidence: num 0.4 0.405 0.431 0.475 0.475 ...  
## $ lift : num 1.57 1.59 3.96 2.45 1.86 ...  
## $ count : num 68 60 69 76 76 69 70 67 63 88 ...