Homework 8

**Chinki**

**May 22, 2017**

**The SVM analysis on the OCR analysis letter data**

**Step 1: Collecting the data**

source of the dataset the UCI Machine Learning Data Repository. The dataset contains 20,000 examples of 26 English alphabet capital letters as printed using 20 different randomly reshaped and distorted black and white fonts.the letters are challenging for a computer to identify, yet are easily recognized by a human being

**Step 2: Exploring and preparing the data**

#Reading data into R  
letters <- read.csv("http://www.sci.csueastbay.edu/~esuess/classes/Statistics\_6620/Presentations/ml11/letterdata.csv")

#Getting structure of 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 ...

All the variables are interger type except letter. In the SVM, all features should be numeric. On the other hand, some of the ranges for these integer variables appear fairly wide. This indicates that we need to normalize or standardize the data. But it is good in the SVM because R performe rescaling automatically in SVM. Lets move to divide data into training and test.

# divide into training and test data  
letters\_train <- letters[1:16000, ]  
letters\_test <- letters[16001:20000, ]

**Step 3: Training a model on the data**

I will use package kernlab for SVM. By default, the ksvm() function uses the Gaussian RBF kernel, but a number of other options are provided.

# begin by training a simple linear SVM  
library(kernlab)  
letter\_classifier <- ksvm(letter ~ ., data = letters\_train,  
 kernel = "vanilladot")

## Setting default kernel parameters

vanilladot kernel is used for linera seperable boundry.

# 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   
##   
## Objective Function Value : -14.1746 -20.0072 -23.5628 -6.2009 -7.5524 -32.7694 -49.9786 -18.1824 -62.1111 -32.7284 -16.2209 -32.2837 -28.9777 -51.2195 -13.276 -35.6217 -30.8612 -16.5256 -14.6811 -32.7475 -30.3219 -7.7956 -11.8138 -32.3463 -13.1262 -9.2692 -153.1654 -52.9678 -76.7744 -119.2067 -165.4437 -54.6237 -41.9809 -67.2688 -25.1959 -27.6371 -26.4102 -35.5583 -41.2597 -122.164 -187.9178 -222.0856 -21.4765 -10.3752 -56.3684 -12.2277 -49.4899 -9.3372 -19.2092 -11.1776 -100.2186 -29.1397 -238.0516 -77.1985 -8.3339 -4.5308 -139.8534 -80.8854 -20.3642 -13.0245 -82.5151 -14.5032 -26.7509 -18.5713 -23.9511 -27.3034 -53.2731 -11.4773 -5.12 -13.9504 -4.4982 -3.5755 -8.4914 -40.9716 -49.8182 -190.0269 -43.8594 -44.8667 -45.2596 -13.5561 -17.7664 -87.4105 -107.1056 -37.0245 -30.7133 -112.3218 -32.9619 -27.2971 -35.5836 -17.8586 -5.1391 -43.4094 -7.7843 -16.6785 -58.5103 -159.9936 -49.0782 -37.8426 -32.8002 -74.5249 -133.3423 -11.1638 -5.3575 -12.438 -30.9907 -141.6924 -54.2953 -179.0114 -99.8896 -10.288 -15.1553 -3.7815 -67.6123 -7.696 -88.9304 -47.6448 -94.3718 -70.2733 -71.5057 -21.7854 -12.7657 -7.4383 -23.502 -13.1055 -239.9708 -30.4193 -25.2113 -136.2795 -140.9565 -9.8122 -34.4584 -6.3039 -60.8421 -66.5793 -27.2816 -214.3225 -34.7796 -16.7631 -135.7821 -160.6279 -45.2949 -25.1023 -144.9059 -82.2352 -327.7154 -142.0613 -158.8821 -32.2181 -32.8887 -52.9641 -25.4937 -47.9936 -6.8991 -9.7293 -36.436 -70.3907 -187.7611 -46.9371 -89.8103 -143.4213 -624.3645 -119.2204 -145.4435 -327.7748 -33.3255 -64.0607 -145.4831 -116.5903 -36.2977 -66.3762 -44.8248 -7.5088 -217.9246 -12.9699 -30.504 -2.0369 -6.126 -14.4448 -21.6337 -57.3084 -20.6915 -184.3625 -20.1052 -4.1484 -4.5344 -0.828 -121.4411 -7.9486 -58.5604 -21.4878 -13.5476 -5.646 -15.629 -28.9576 -20.5959 -76.7111 -27.0119 -94.7101 -15.1713 -10.0222 -7.6394 -1.5784 -87.6952 -6.2239 -99.3711 -101.0906 -45.6639 -24.0725 -61.7702 -24.1583 -52.2368 -234.3264 -39.9749 -48.8556 -34.1464 -20.9664 -11.4525 -123.0277 -6.4903 -5.1865 -8.8016 -9.4618 -21.7742 -24.2361 -123.3984 -31.4404 -88.3901 -30.0924 -13.8198 -9.2701 -3.0823 -87.9624 -6.3845 -13.968 -65.0702 -105.523 -13.7403 -13.7625 -50.4223 -2.933 -8.4289 -80.3381 -36.4147 -112.7485 -4.1711 -7.8989 -1.2676 -90.8037 -21.4919 -7.2235 -47.9557 -3.383 -20.433 -64.6138 -45.5781 -56.1309 -6.1345 -18.6307 -2.374 -72.2553 -111.1885 -106.7664 -23.1323 -19.3765 -54.9819 -34.2953 -64.4756 -20.4115 -6.689 -4.378 -59.141 -34.2468 -58.1509 -33.8665 -10.6902 -53.1387 -13.7478 -20.1987 -55.0923 -3.8058 -60.0382 -235.4841 -12.6837 -11.7407 -17.3058 -9.7167 -65.8498 -17.1051 -42.8131 -53.1054 -25.0437 -15.302 -44.0749 -16.9582 -62.9773 -5.204 -5.2963 -86.1704 -3.7209 -6.3445 -1.1264 -122.5771 -23.9041 -355.0145 -31.1013 -32.619 -4.9664 -84.1048 -134.5957 -72.8371 -23.9002 -35.3077 -11.7119 -22.2889 -1.8598 -59.2174 -8.8994 -150.742 -1.8533 -1.9711 -9.9676 -0.5207 -26.9229 -30.429 -5.6289   
## Training error : 0.130062

This information tells us very little about how well the model will perform in the real world. We'll need to examine its performance on the testing dataset to know whether it generalizes well to unseen data. training error is 13%.

**Step 4: Evaluating model performance**

The predict() function allows us to use the letter classification model to make predictions on the testing dataset.

# 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

#Getting table of prediction and test  
table(letter\_predictions, letters\_test$letter)

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

The diagonal values of 144, 121, 120, 156, and 127 indicate the total number of records where the predicted letter matches the true value.It is not possible to read this type of representation.

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

## agreement  
## FALSE TRUE   
## 643 3357

#Getting propotion of true & false values   
prop.table(table(agreement))

## agreement  
## FALSE TRUE   
## 0.16075 0.83925

The recognition accuracy of about 83 percent which is not to good. I can try to improve model.

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

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

It can be challenging, however, to choose from the many different kernel functions. A popular convention is to begin with the Gaussian RBF kernel.

prop.table(table(agreement\_rbf))

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

The recognition accuracy of about 93 percent which is not to good. Lets try another kernel I am going to use polynomial kernel.

set.seed(12345)  
letter\_classifier\_rbf <- ksvm(letter ~ ., data = letters\_train, kernel = "polydot")

## Setting default kernel parameters

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

## agreement\_rbf  
## FALSE TRUE   
## 643 3357

prop.table(table(agreement\_rbf))

## agreement\_rbf  
## FALSE TRUE   
## 0.16075 0.83925

83% accuracy.

set.seed(12345)  
letter\_classifier\_rbf <- ksvm(letter ~ ., data = letters\_train, kernel = "tanhdot")

## Setting default kernel parameters

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

## agreement\_rbf  
## FALSE TRUE   
## 3662 338

Not performing good.

**The Cluster analysis on the sns data**

**Step 1 - collecting data**

we will use a dataset representing a random sample of 30,000 U.S. high school students who had profiles on a well-known SNS in 2006. To protect the users' anonymity, the SNS will remain unnamed.The full dataset is available at the Packt Publishing website with the filename snsdata.csv.The data was sampled evenly across four high school graduation years (2006 through 2009) representing the senior, junior, sophomore, and freshman classes at the time of data collection.

**Step 2 - exploring and preparing the data**

#Reading the data  
teens <- read.csv("http://www.sci.csueastbay.edu/~esuess/classes/Statistics\_6620/Presentations/ml12/snsdata.csv")  
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 ...

30000 teenagers record with 40 variables. In which four variables indicating personal characteristics and 36 words indicating interests. There are missing values in the gender row.

# Table of genders  
table(teens$gender)

##   
## F M   
## 22054 5222

#look at missing data for female variable  
table(teens$gender, useNA = "ifany")

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

2724 records have missing gender data.Females are 4 times more than the male. Age also has missing values.

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

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 3.086 16.310 17.290 17.990 18.260 106.900 5086

5086 missing values in the age colun. Minimum and maximum value is also not making any sence.A 3 year old or a 106 year old is attending high school.

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

#Summary of the age  
summary(teens$age)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's   
## 13.03 16.30 17.26 17.25 18.22 20.00 5523

Now minimum and maximum is making sence to us. 5523 tenagers with the no information about age. Data preparation-Dummy coding for the missing values An easy solution for handling the missing values is to exclude any record with a missing value.

# 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)

Create dummy variables for female and unknown gender.

# check our recoding work  
table(teens$gender, useNA = "ifany")

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

# check our recoding work  
table(teens$female, useNA = "ifany")

##   
## 0 1   
## 7946 22054

# check our recoding work  
table(teens$no\_gender, useNA = "ifany")

##   
## 0 1   
## 27276 2724

The number of 1 values for teensno\_gender matches the number of F and NA values, respectively, so we should be able to trust our work.

Data preparation - imputing the missing values let's eliminate the 5,523 missing age values.

# finding the mean age by cohort  
mean(teens$age, na.rm = TRUE) # works

## [1] 17.25243

This reveals that the average student in our data is about 17 years old.

# 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**

#Z-scoring of the numeric variables   
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**

The success or failure of the model hinges on whether the clusters are useful for their intended purpose.

# look at the size of the clusters  
teen\_clusters$size

## [1] 871 600 5981 1034 21514

Here, we see the five clusters we requested. The smallest cluster has 600 teenagers (2 percent) while the largest cluster has 21,514 (72 percent).

# 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

For example, the third row has the highest value in the basketball column, which means that cluster 3 has the highest average interest in basketball among all the clusters. Hilighted points has highest average. Cluster 3 is substantially above the mean interest level on all the sports.

**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

Using the aggregate() function, we can also look at the demographic characteristics of the clusters.

**The Association analysis on the groceries analysis letter data**

**Step 1 - collecting data**

Our market basket analysis will utilize the purchase data collected from one month of operation at a real-world grocery store. The data contains 9,835 transactions or about 327 transactions per day (roughly 30 transactions per hour in a 12-hour business day), suggesting that the retailer is not particularly large, nor is it particularly small. The typical grocery store offers a huge variety of items. There might be five brands of milk, a dozen different types of laundry detergent, and three brands of coffee. Given the moderate size of the retailer, we will assume that they are not terribly concerned with finding rules that apply only to a specific brand of milk or detergent.

**Step 2 - exploring and preparing the data**

R created four columns to store the items in the transactional data: V1, V2, V3, and V4, which is in the first row. R chose to create four variables because the first line had exactly four comma-separated values. However, we know that grocery purchases can contain more than four items.So we will use sparse matrix. The solution to this problem utilizes a data structure called a sparse matrix. Since there are 169 different items in our grocery store data, our sparse matrix will contain 169 columns.

# load the grocery data into a sparse matrix  
library(arules)

## 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

groceries <- read.transactions("http://www.sci.csueastbay.edu/~esuess/classes/Statistics\_6620/Presentations/ml13/groceries.csv", sep = ",")  
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 abrasive cleaner  
## 2 artif. sweetener  
## 3 baby cosmetics

Dataset has 9835 rows and 169 column.Most frequent item is whole milk.A total of 2,159 transactions contained only a single item, while one transaction had 32 items. The first quartile and median purchase sizes are two and three items, respectively, implying that 25 percent of the transactions contained two or fewer items and the transactions were split in half between those with less than three items and those with more.

# 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}

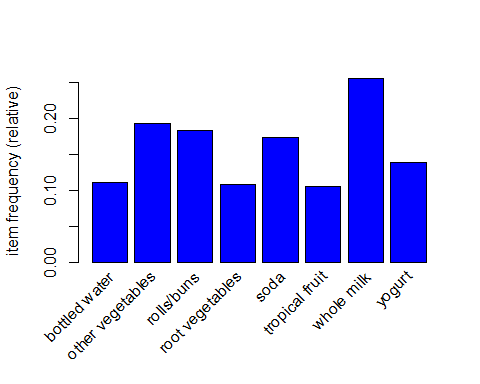
The first five transactions are given.

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

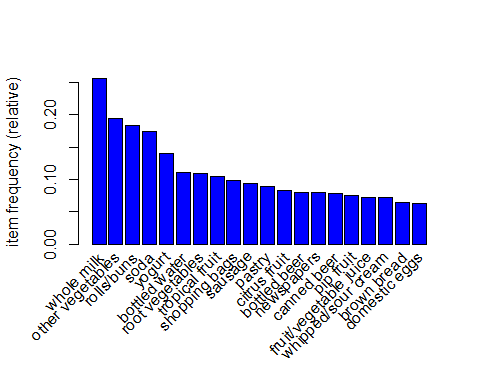
## abrasive cleaner artif. sweetener baby cosmetics   
## 0.0035587189 0.0032536858 0.0006100661

The itemFrequency() function allows us to see the proportion of transactions that contain the item.

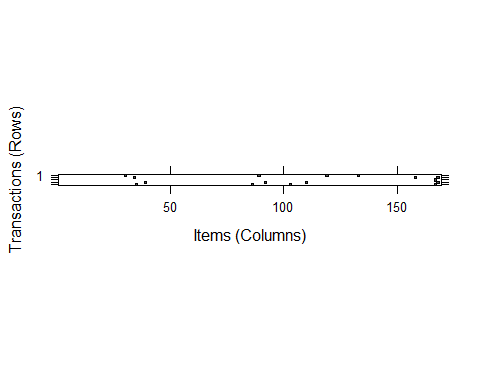
# plot the frequency of items  
itemFrequencyPlot(groceries, support = 0.1,col="blue")

 As shown in the following plot, this results in a histogram showing the eight items in the groceries data with at least 10 percent support. Means the iteams, which accour sell at least 10%.

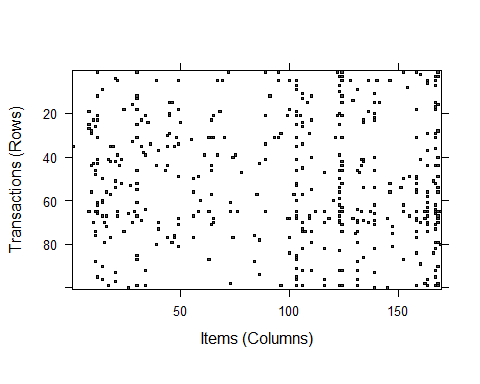
itemFrequencyPlot(groceries, topN = 20,col="blue")

 If you would rather limit the plot to a specific number of items, the topN parameter can be used with itemFrequencyPlot.The histogram is then sorted by decreasing support, as shown in the following diagram of the top 20 items in the groceries data.

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

 The resulting diagram depicts a matrix with 5 rows and 169 columns, indicating the 5 transactions and 169 possible items we requested.

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

 This creates a matrix diagram with 100 rows and 169 columns. The command to create random selection of 100 transactions.A few columns seem fairly heavily populated, indicating some very popular items at the store.Overall distribution iss fairly random.

**Step 3 - training a model on the data**

I am an implementation of the Apriori algorithm in the arules package.

## Step 3: Training a model on the data ----  
library(arules)  
  
# 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

# 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 [0.01s].  
## 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 [0.00s].

groceryrules

## set of 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   
## Min. :0.006101 Min. :0.2500 Min. :0.9932   
## 1st Qu.:0.007117 1st Qu.:0.2971 1st Qu.:1.6229   
## Median :0.008744 Median :0.3554 Median :1.9332   
## Mean :0.011539 Mean :0.3786 Mean :2.0351   
## 3rd Qu.:0.012303 3rd Qu.:0.4495 3rd Qu.:2.3565   
## Max. :0.074835 Max. :0.6600 Max. :3.9565   
##   
## 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

This means if customer buying potted plants they will also buy whole milk with the support 0.0069 and confidence 0.400.

**Step 5 - improving model performance**

# sorting grocery rules by lift  
inspect(sort(groceryrules, by = "lift")[1:5])

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

The first rule, with a lift of about 3.96, implies that people who buy herbs are nearly four times more likely to buy root vegetables than the typical customer.

# 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

# 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 4 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 ...