Homework 3

Group 5 Charu Aggarwal

1. Mei-Chun Hung
2. Sukanya Aswini Dutta
3. Vaibhavi Gaekwad
4. Wasinee Sriapha
5. Yufei Wang

206-880-9298 (Tel of Student 1)
206-941-3762 (Tel of Student 2)
425-624-5609 (Tel of Student 3)
206-380-6328 (Tel of Student 4)
206-319-8422(Tel of Student 5)

Percentage of Effort Contributed by Student 1:_____20%_______
Percentage of Effort Contributed by Student 2:_____20%_______
Percentage of Effort Contributed by Student 3:_____20%_______
Percentage of Effort Contributed by Student 4:_____20%_______
Percentage of Effort Contributed by Student 5:_____20%_______

Signature of Student 1:____MH_______________________________
Signature of Student 2:____SAD______________________________
Signature of Student 3:____VG_______________________________
Signature of Student 4:____WOS______________________________
Signature of Student 5:____YW_______________________________

Submission Date:_________04/01/20________________

IE7275 HW3 Group 5

Group 5

4/01/2020

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com (http://rmarkdown.rstudio.com).

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```r
#load packages
library(readr)
library(readxl)
library(forecast)
library(tidyverse)
library(caret)
library(rpart)
library(caret)
library(e1071)
library(data.table)
library(leaps)
library(MASS)
library(readr)
library(corrplot)
library(gridExtra)
library(formattable)
library(FNN)
```

Problem 7.1

```r
#Problem 7.1
#read the dataset
UniversalBank <- read_csv("UniversalBank.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_double(),
##   Age = col_double(),
##   Experience = col_double(),
##   Income = col_double(),
##   `ZIP Code` = col_double(),
##   Family = col_double(),
##   CCAvg = col_double(),
##   Education = col_double(),
##   Mortgage = col_double(),
##   `Personal Loan` = col_double(),
##   `Securities Account` = col_double(),
##   `CD Account` = col_double(),
##   Online = col_double(),
##   CreditCard = col_double()
## )
```

```
str(UniversalBank)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 5000 obs. of
14 variables:
##  $ ID                : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age               : num  25 45 39 35 35 37 53 50 35 34 ...
##  $ Experience        : num  1 19 15 9 8 13 27 24 10 9 ...
##  $ Income            : num  49 34 11 100 45 29 72 22 81 180 ...
##  $ ZIP Code          : num  91107 90089 94720 94112 91330 ...
##  $ Family            : num  4 3 1 1 4 4 2 1 3 1 ...
##  $ CCAvg             : num  1.6 1.5 1 2.7 1 0.4 1.5 0.3 0.6 8.9 ...
##  $ Education         : num  1 1 1 2 2 2 2 3 2 3 ...
##  $ Mortgage          : num  0 0 0 0 0 155 0 0 104 0 ...
##  $ Personal Loan     : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ Securities Account: num  1 1 0 0 0 0 0 0 0 0 ...
##  $ CD Account        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Online            : num  0 0 0 0 0 1 1 0 1 0 ...
##  $ CreditCard        : num  0 0 0 0 1 0 0 1 0 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   ID = col_double(),
##   ..   Age = col_double(),
##   ..   Experience = col_double(),
##   ..   Income = col_double(),
##   ..   `ZIP Code` = col_double(),
##   ..   Family = col_double(),
##   ..   CCAvg = col_double(),
##   ..   Education = col_double(),
##   ..   Mortgage = col_double(),
##   ..   `Personal Loan` = col_double(),
##   ..   `Securities Account` = col_double(),
##   ..   `CD Account` = col_double(),
##   ..   Online = col_double(),
##   ..   CreditCard = col_double()
##   .. )
```

```r
#check the missing values
sapply(UniversalBank, function(x) sum(is.na(x)))
```

```
##                  ID                 Age          Experience             In
come
##                   0                   0                   0
0
##            ZIP Code              Family               CCAvg          Educa
tion
##                   0                   0                   0
0
##            Mortgage       Personal Loan Securities Account           CD Acc
ount
##                   0                   0                   0
0
##              Online          CreditCard
##                   0                   0
```

```
# drop the irrelevant columns
UniversalBank$ID <- NULL
UniversalBank$`ZIP Code` <- NULL

## Remove spaces from column names
university01<- setnames(x = UniversalBank, old = names(UniversalBank),
         new = gsub(" ", "", names(UniversalBank)))

# get the chategorical response variable
university01$PersonalLoan <- factor(university01$PersonalLoan)

#reorder the columns
university02 <- university01[,c(8,1,2,3,4,5,6,7,9,10,11,12)]
```

Experience: # of years of professional experience Income: annual income (*$000) Family: family size CCAvg: Average monthly credit card spending ($000) Education Education level: 1: undergrad; 2, Graduate; 3; Advance/Professional Mortgage: Value of house mortgage ($000) (response variable) Personal loan :Did this customer accept the personal loan offered in he last campaign? 1, yes; 0, no

1: yes and 0: no Securities Acct: Does the customer have a securities account with the bank? CD Account: Does the customer have a certifcate of deposit (CD) account with the bank? Online: Does the customer use internet bank facilities? CreditCard: Does the customer use a credit card issued by the Bank?

```r
#part a
#load and partition the dataset: training (60%) and validation (40%) sets
set.seed(105)
indexknn<- sample(1:nrow(university02),size=nrow(university02)*0.6,replace
        = FALSE)
train_knn<- university02[indexknn,] # 60% training data
test_knn<- university02[-indexknn,]

#create the separate dataframe
train_knn_pl<- university02[indexknn,1]

# initialize normalized training, validation data, complete data frames to
        originals
train.norm.df <- train_knn
valid.norm.df <- test_knn
bank.norm.df <- university02

# use preProcess() from the caret package to normalize features
norm.values <- preProcess(train_knn[, -1], method=c("center", "scale"))

train.norm.df[, -1] <- predict(norm.values, train_knn[, -1])
valid.norm.df[, -1] <- predict(norm.values, test_knn[, -1])
bank.norm.df[, -1] <- predict(norm.values, university02[, -1])
```

```r
# for the new customer; create the data frame
library(dplyr)
set.seed(105)
new.df1<-data.frame(40,10,84,2,2,1,0,0,0,1,1) %>%
        setNames(names(university02[, -1]))

new.df2<-data.frame(40,10,84,2,2,0,0,0,0,1,1) %>%
        setNames(names(university02[, -1]))

#normalize
new.norm.df1 <- predict(norm.values, new.df1)

#build the prediction model
cl <- train_knn_pl[,1,drop = TRUE]

knn.pred_new1<-knn(train = train.norm.df[,-1],
                         test = new.norm.df1,
                         cl, k = 1)
row.names(train_knn)[attr(knn.pred_new1,"nn.index")]
```

```
## [1] "176"
```

a.**How would this customer be classified?** when k=1, the closest customer in the train dataset locates at the number 176 row, and the personloan status is 0. Therefore, for this new customer, he will be classified as the class that he will not accept the personal loan offered in his last campaign.

b.**What is a choice of k that balances between overfitting and ignoring the predictor information?** Generally speasking, if k is too low, we may be fitting the noise in the data, and if k is too high, we may miss out on the method`s ability to capture the local structure in the data. If we want to balance between overfitting to the prodictor information and ignore the predictor information, we would consider the extreme condition that k is the same number of records in the training dataset. we simply assign all records to the majority class in the training data, oversmoothing the absense of useful information in the predict about the class

```
#part c
#compute knn for different k on validation to find the best k
# initialize a data frame with two columns: k, and accuracy
set.seed(105)
i=1
k.optm=1
for (i in 1:20){
    knn.mod <-  knn(train=train.norm.df[,-1], test=valid.norm.df[, -1], c
        l, k=i)
    k.optm[i] <- 100 * sum(knn.mod == test_knn$PersonalLoan)/NROW(test_knn
        $PersonalLoan)
    k=i
    cat(k,'=',k.optm[i],'\n')
}
```

```
## 1 = 95.9
## 2 = 95.55
## 3 = 96.35
## 4 = 95.85
## 5 = 96.55
## 6 = 95.65
## 7 = 96.35
## 8 = 95.5
## 9 = 95.8
## 10 = 95.4
## 11 = 95.55
## 12 = 95.15
## 13 = 95.2
## 14 = 94.85
## 15 = 95.15
## 16 = 94.65
## 17 = 94.95
## 18 = 94.6
## 19 = 94.7
## 20 = 94.6
```

```r
#part c
#the best k=5, with the highest accuracy
knn.5 <- knn(train=train.norm.df[,-1], test=valid.norm.df[, -1], cl, k=5)

#show the confusion matrix for the validation data
library(caret)
confusionMatrix(knn.5,valid.norm.df$PersonalLoan)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1808   65
##          1    4  123
##
##                Accuracy : 0.9655
##                  95% CI : (0.9565, 0.9731)
##     No Information Rate : 0.906
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.763
##
##  Mcnemar's Test P-Value : 5.08e-13
##
##             Sensitivity : 0.9978
##             Specificity : 0.6543
##          Pos Pred Value : 0.9653
##          Neg Pred Value : 0.9685
##              Prevalence : 0.9060
##          Detection Rate : 0.9040
##    Detection Prevalence : 0.9365
##       Balanced Accuracy : 0.8260
##
##        'Positive' Class : 0
##
```

```r
#part d
set.seed(1060)
knn.pred_new1<-knn(train = train.norm.df[,-1],
                         test = new.norm.df1,
                         cl, k = 5)
knn.pred_new1
```

```
## [1] 0
## attr(,"nn.index")
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  176 1731 2613 2921 2713
## attr(,"nn.dist")
##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.2996233 0.4527126 0.5767518 0.6262866 0.9698353
## Levels: 0
```

```
row.names(train_knn)[attr(knn.pred_new1,"nn.index")]
```

```
## [1] "176"  "1731" "2613" "2921" "2713"
```

d. **Classify the customer using the best k.** when k=5, the closest customer in the train dataset locates at the number 176,1731,2613,2921,and 2713 rows, and the personloan status is 0. Therefore, for this new customer, he will be classified as the class that he will not accept the personal loan offered in his last campaign.

```
#part e
#Repartition the data, this time into training, validation, and test sets
         (50% : 30% : 20%).
set.seed(1050)
splitSample <- sample(1:3, size=nrow(university02), prob=c(0.5,0.3,0.2), r
        eplace = TRUE)
train.hex <- university02[splitSample==1,]
valid.hex <- university02[splitSample==2,]
test.hex <- university02[splitSample==3,]

#create the separate dataframe for personloan
train.hex_pl <- university02[splitSample==1,1]
```

```
# initialize normalized training, validation data, complete data frames to
        originals
train.norm.df <- train.hex
valid.norm.df <- valid.hex
test.norm.df <- test.hex

# use preProcess() from the caret package to normalize features
set.seed(1050)
norm.values <- preProcess(train.hex[, -1], method=c("center", "scale"))
train.norm.df[, -1] <- predict(norm.values, train.hex[, -1])
valid.norm.df[, -1] <- predict(norm.values, valid.hex[, -1])
test.norm.df[, -1] <- predict(norm.values, test.hex[, -1])
```

```r
#apply the k-NN method with the k=5
set.seed(500)
cl <- train.hex_pl[,1,drop = TRUE]

model1 <- knn(train=train.norm.df[,-1], test=valid.norm.df[, -1], cl, k=5)
model2 <- knn(train=train.norm.df[,-1], test=test.norm.df[, -1], cl, k=5)

#show the confusion matrix for the validation data
library(caret)
confusionMatrix(model1,valid.norm.df$PersonalLoan)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1359   79
##          1    6   75
##
##                Accuracy : 0.944
##                  95% CI : (0.9313, 0.9551)
##     No Information Rate : 0.8986
##     P-Value [Acc > NIR] : 1.559e-10
##
##                   Kappa : 0.6111
##
##  Mcnemar's Test P-Value : 5.742e-15
##
##             Sensitivity : 0.9956
##             Specificity : 0.4870
##          Pos Pred Value : 0.9451
##          Neg Pred Value : 0.9259
##              Prevalence : 0.8986
##          Detection Rate : 0.8947
##    Detection Prevalence : 0.9467
##       Balanced Accuracy : 0.7413
##
##        'Positive' Class : 0
##
```

```r
#show the confusion matrix for the test data
library(caret)
confusionMatrix(model2,test.norm.df$PersonalLoan)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 916  48
##          1   3  60
##
##                Accuracy : 0.9503
##                  95% CI : (0.9352, 0.9628)
##     No Information Rate : 0.8948
##     P-Value [Acc > NIR] : 1.314e-10
##
##                   Kappa : 0.6767
##
##  Mcnemar's Test P-Value : 7.218e-10
##
##             Sensitivity : 0.9967
##             Specificity : 0.5556
##          Pos Pred Value : 0.9502
##          Neg Pred Value : 0.9524
##              Prevalence : 0.8948
##          Detection Rate : 0.8919
##    Detection Prevalence : 0.9387
##       Balanced Accuracy : 0.7761
##
##        'Positive' Class : 0
##
```

e.**Compare the confusion matrix of the test set with that of the training and validation sets. Comment on the differences and their reason.** Train the knn model with training dataset and k=5. When we used the validation set to test the model, the model accuracy is 0.944. When we used the test set to test the model, the model accuracy is 0.95, which is higher than the previous accuracy. KNN is a lazy learner. For every record to be predicted, we compute its distance from the entire set of trainging records. Because the validation data size is larger than the testing data size, the error for validation the model should be higher if we choose the same training dataset.

Problem 7.2

```
#problem7.2
#load the dataset
library(readr)
housing <- read_csv("BostonHousing.csv")
```

```
## Parsed with column specification:
## cols(
##   CRIM = col_double(),
##   ZN = col_double(),
##   INDUS = col_double(),
##   CHAS = col_double(),
##   NOX = col_double(),
##   RM = col_double(),
##   AGE = col_double(),
##   DIS = col_double(),
##   RAD = col_double(),
##   TAX = col_double(),
##   PTRATIO = col_double(),
##   LSTAT = col_double(),
##   MEDV = col_double(),
##   `CAT. MEDV` = col_double()
## )
```

```
str(housing)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 506 obs. of  1
4 variables:
##  $ CRIM     : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ ZN       : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ INDUS    : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87
...
##  $ CHAS     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ NOX      : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.52
4 0.524 ...
##  $ RM       : num  6.58 6.42 7.18 7 7.15 ...
##  $ AGE      : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ DIS      : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ RAD      : num  1 2 2 3 3 3 5 5 5 5 ...
##  $ TAX      : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ PTRATIO  : num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2
...
##  $ LSTAT    : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ MEDV     : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
##  $ CAT. MEDV: num  0 0 1 1 1 0 0 0 0 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   CRIM = col_double(),
##   ..   ZN = col_double(),
##   ..   INDUS = col_double(),
##   ..   CHAS = col_double(),
##   ..   NOX = col_double(),
##   ..   RM = col_double(),
##   ..   AGE = col_double(),
##   ..   DIS = col_double(),
##   ..   RAD = col_double(),
##   ..   TAX = col_double(),
##   ..   PTRATIO = col_double(),
##   ..   LSTAT = col_double(),
##   ..   MEDV = col_double(),
##   ..   `CAT. MEDV` = col_double()
##   .. )
```

```
#check the missing values
sapply(housing, function(x) sum(is.na(x)))
```

```
##      CRIM        ZN     INDUS      CHAS       NOX        RM       AGE
DIS
##         0         0         0         0         0         0         0
0
##       RAD       TAX   PTRATIO     LSTAT      MEDV CAT. MEDV
##         0         0         0         0         0         0
```

```
#ignore the irrelevant column
housing$`CAT. MEDV` <- NULL
```

```
#part a
#load and partition the dataset: training (60%) and validation (40%) sets
set.seed(500)
indexknn<- sample(1:nrow(housing),size=nrow(housing)*0.6,replace = FALSE)
train_knn <- housing[indexknn,] # 60% training data
test_knn<- housing[-indexknn,]

#create the separate dataframe for MEDV feature
train_knn_MEDV<- housing[indexknn,13]
test_knn_MEDV<- housing[-indexknn,13]

# initialize normalized training, validation data, complete data frames to
        originals
train.norm.df <- train_knn
valid.norm.df <- test_knn
housing.norm.df <-housing

# use preProcess() from the caret package to normalize features
norm.values <- preProcess(train_knn[, -13], method=c("center", "scale"))

train.norm.df[, -13] <- predict(norm.values, train_knn[, -13])
valid.norm.df[, -13] <- predict(norm.values, test_knn[, -13])
housing.norm.df[, -13] <- predict(norm.values, housing[, -13])
```

```
library(class)
```

```
##
## Attaching package: 'class'
```

```
## The following objects are masked from 'package:FNN':
##
##     knn, knn.cv
```

```r
library(caret)
library(FNN)
#initialize a data frame with two columns: k, and accuracy
#trying values of k from 1 to 5
set.seed(105)
accuracy.df <- data.frame(k = seq(1, 5, 1), RMSE = rep(0, 5))

# compute knn for different k on validation. Column 13 is MEDV
#train-matrix or data frame of training set cases.
#test-matrix or data frame of test set cases. A vector will be interpreted
        as a row vector for a single case.
#cl-actor of true classifications of training set

cl <- train_knn_MEDV[,1,drop = TRUE]
for(i in 1:5){
  knn.pred<-class::knn(train = train.norm.df[,-13],
                       test = valid.norm.df[,-13],
                       cl, k = i)
  accuracy.df[i,2]<-RMSE(as.numeric(as.character(knn.pred)),test_knn$MEDV)
}

accuracy.df
```

k <dbl>	RMSE <dbl>
1	5.659658
2	5.637087
3	6.178817
4	6.743349
5	6.625667

5 rows

a.**What is the best k? What does it mean?** In general, the RMSE corresponds to the square root of the average difference between the observed known outcome values and the predicted values. The lower the RMSE, the better the model performance. Therefore, for a KNN for a numerical outcome, the best k is the number that provides smmoothing that reduces the risk of overfitting due to noise in the training data, with the lowest RMSE value. In this case, k=2 is the best k with the lowest RMSE (5.63)

```
#part b
#predict the MEDV for a tract with the following information, using the be
        st k:
library(dplyr)
set.seed(1015)
#New data
new.df<-data.frame(0.2,0,7,0,0.538,6,62,4.7,4,307,21,10) %>%
        setNames(names(housing[, -13]))
#normalize
new.norm.df <- predict(norm.values, new.df)

#build the prediction model
knn.pred_new<-class::knn(train = train.norm.df[,-13],
                         test = new.norm.df,
                         cl, k = 2)
accuracy.df_new <- data.frame(MEDV = knn.pred_new, RMSE = RMSE(as.numeric(
        as.character(knn.pred_new)),test_knn$MEDV))
accuracy.df_new
```

MEDV <fctr>	RMSE <dbl>
18.5	9.540396

1 row

b.**Predict the MEDV for a tract with the following information, using the best k:** The predicted MEDV is 18.5 and the corresponding RMSE is 9.54

```
#part c
#build the prediction model
set.seed(100)
cl <- train_knn_MEDV[,1,drop = TRUE]
knn.pred<-class::knn(train = train.norm.df[,-13],
                     test = train.norm.df[,-13],
                     cl, k = 2)
#get the error of the training set
a <-RMSE(as.numeric(as.character(knn.pred)),train_knn$MEDV)
a
```

```
## [1] 3.268572
```

c.**If we used the above k-NN algorithm to score the training data, what would be the error of the training set?** Training error here is the error you'll have when you input your training set to your KNN as test set. As you can see, when I input the training set as the test set, the error of the training set is 3.26, which is overly optimistic compared to the error rate when applying this k-NN predictor to the test set (5.63).

d.**Why is the validation data error overly optimistic compared to the error rate when applying this k-NN predictor to new data?** Since your test sample is in the training dataset, it'll choose itself as the closest and never make mistake. For this reason, the training error will be zero when K = 1, irrespective of the dataset. In our case, k=2, which means that the model will choose itself and one closest to itself for the validation, therefore, If we used the k-NN algorithm to score the training data itself, we will get optimistic errors.

e.**If the purpose is to predict MEDV for several thousands of new tracts, what would be the disadvantage of using k-NN prediction? List the operations that the algorithm goes through in order to produce each prediction.**

As KNN is a lazy-learning algorithm, it's slow as it stores the training data and calculates based on the current data set instead of coming up with an algorithm based on historical data. If we were to work with MEDV for several thousands of new tracts, the prediction stage might be very slow. In other words, the algorithm must compute the distance and sort all the training data at each prediction, which can be even slower when there is a large number of training data. The operation steps are as following:

1: The algorithm finds distance by computing the distances between the new data and all the training data. Mostly used metrics for calculating distance are Euclidean, Manhattan and Minkowski.

2: Then it finds a minimum distance, sort in order, and determine k nearest neighbors based on minimum distance values and cross validated RMSE values

3: The category of the nearest neighbors are then analyzed and assigned to the test data based on highest majority/weight

4: The predicted class label is returned.

Problem 8.1

```
#problem 8.1
#read the dataset
UniversalBank <- read_csv("UniversalBank.csv")
```

```
## Parsed with column specification:
## cols(
##   ID = col_double(),
##   Age = col_double(),
##   Experience = col_double(),
##   Income = col_double(),
##   `ZIP Code` = col_double(),
##   Family = col_double(),
##   CCAvg = col_double(),
##   Education = col_double(),
##   Mortgage = col_double(),
##   `Personal Loan` = col_double(),
##   `Securities Account` = col_double(),
##   `CD Account` = col_double(),
##   Online = col_double(),
##   CreditCard = col_double()
## )
```

```
#check the missing values
sapply(UniversalBank, function(x) sum(is.na(x)))
```

```
##                 ID                Age         Experience                 In
come
##                  0                  0                  0                   
0
##           ZIP Code             Family              CCAvg              Educa
tion
##                  0                  0                  0                   
0
##           Mortgage      Personal Loan Securities Account             CD Acc
ount
##                  0                  0                  0                   
0
##             Online         CreditCard
##                  0                  0
```

```r
# drop the irrelevant columns
UniversalBank$ID <- NULL
UniversalBank$`ZIP Code` <- NULL

## Remove spaces from column names
university01<- setnames(x = UniversalBank, old = names(UniversalBank),
         new = gsub(" ", "", names(UniversalBank)))

#reorder the columns
university02 <- university01[,c(8,1,2,3,4,5,6,7,9,10,11,12)]

#convert variables to categorical type
for (i in c(1:dim(university02)[2])){
  university02[,i] = data.frame(apply(university02[i], 2, as.factor))
}
str(university02)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    5000 obs. of  12 variable
s:
##  $ PersonalLoan     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2
...
##  $ Age              : Factor w/ 45 levels "23","24","25",..: 3 23 17 13
13 15 31 28 13 12 ...
##  $ Experience       : Factor w/ 47 levels "-1","-2","-3",..: 5 15 11 47
46 9 24 21 6 47 ...
##  $ Income           : Factor w/ 162 levels "10","100","101",..: 120 109
10 2 118 104 139 98 147 72 ...
##  $ Family           : Factor w/ 4 levels "1","2","3","4": 4 3 1 1 4 4 2
1 3 1 ...
##  $ CCAvg            : Factor w/ 108 levels "0","0.1","0.2",..: 20 19 13
36 13 5 19 4 7 106 ...
##  $ Education        : Factor w/ 3 levels "1","2","3": 1 1 1 2 2 2 2 3 2
3 ...
##  $ Mortgage         : Factor w/ 347 levels "0","100","101",..: 1 1 1 1
1 57 1 1 6 1 ...
##  $ SecuritiesAccount: Factor w/ 2 levels "0","1": 2 2 1 1 1 1 1 1 1 1
...
##  $ CDAccount        : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1
...
##  $ Online           : Factor w/ 2 levels "0","1": 1 1 1 1 1 2 2 1 2 1
...
##  $ CreditCard       : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 2 1 1
...
```

```
#get the two predictor variables and response variable
university03 <- university02[,c("PersonalLoan","Online","CreditCard")]

#partition the data into training (60%) and validation (40%) sets
set.seed(1005)
index<- sample(1:nrow(university03),size=nrow(university03)*0.6,replace =
        FALSE)
train_data<- university03[index,] # 60% training data
test_data<- university03[-index,]
```

```
#a create the pivot table
set.seed(500)
#1 use the reshape
library(reshape)
```

```
##
## Attaching package: 'reshape'
```

```
## The following object is masked from 'package:class':
##
##     condense
```

```
## The following object is masked from 'package:data.table':
##
##     melt
```

```
## The following object is masked from 'package:dplyr':
##
##     rename
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, smiths
```

```
mdata <- melt(train_data, id=c("Online","CreditCard"))
cast(mdata, Online + CreditCard ~ variable)
```

```
## Aggregation requires fun.aggregate: length used as default
```

	Online	CreditCard	PersonalLoan
	<fctr>	<fctr>	<int>
1	0	0	840

	Online <fctr>	CreditCard <fctr>	PersonalLoan <int>
2	0	1	383
3	1	0	1209
4	1	1	568

4 rows

```
#2 use the pivot table
library(rpivotTable)
rpivotTable::rpivotTable(train_data,rows = c("CreditCard", "Online"), cols
        = "PersonalLoan",
            width = "100%", height="400%")
```

Table

Count ↕ ↔

PersonalLoan

CreditCard

Online

		PersonalLoan			
CreditCard	Online		0	1	Totals
0	0		762	78	**840**
	1		1,094	115	**1,209**
1	0		349	34	**383**
	1		516	52	**568**
		Totals	**2,721**	**279**	**3,000**

```
#b
p <- round(52/568,5)
p
```

```
## [1] 0.09155
```

```
paste("The exact bayes conditional probability that this customer will acc
        ept the loan offer:", p*100, "%")
```

```
## [1] "The exact bayes conditional probability that this customer will ac
cept the loan offer: 9.155 %"
```

```
#c
#pivot table 1:  Loan (rows) as a function of Online (columns)
library(rpivotTable)
rpivotTable::rpivotTable(train_data, rows = "PersonalLoan", cols = "Onlin
        e",
            width = "100%", height="400%")
```

Table

Count ↕ ↔

CreditCard

Online

PersonalLoan

	Online	0	1	Totals
PersonalLoan				
0		1,111	1,610	2,721
1		112	167	279
	Totals	1,223	1,777	3,000

```
#c
#pivot table 2:Loan (rows) as a function of CC
library(rpivotTable)
rpivotTable(train_data, rows = "PersonalLoan", cols = "CreditCard",
            width = "100%", height="400%")
```

Table

Count ↕ ↔

Online

CreditCard

PersonalLoan

	CreditCard	0	1	Totals
PersonalLoan				
0		1,856	865	2,721
1		193	86	279
	Totals	2,049	951	3,000

```
#d
p1 <- round(86/279,4)
paste("P(CC  =  1 |Loan  =  1) =", p1*100,"%")
```

```
## [1] "P(CC  =  1 |Loan  =  1) = 30.82 %"
```

```
p2<- round(167/279,4)
paste("P(Online = 1 |Loan = 1)=", p2*100,"%")
```

```
## [1] "P(Online = 1 |Loan = 1)= 59.86 %"
```

```
p3<- round(279/3000,4)
paste("P(Loan = 1)=", p3*100,"%")
```

```
## [1] "P(Loan = 1)= 9.3 %"
```

```
p4 <- round(865/2721,4)
paste("P(CC= 1 |Loan = 0)=", p4*100,"%")
```

```
## [1] "P(CC= 1 |Loan = 0)= 31.79 %"
```

```
p5 <- round(1610/2721,4)
paste("P(Online = 1 |Loan = 0)=", p5*100,"%")
```

```
## [1] "P(Online = 1 |Loan = 0)= 59.17 %"
```

```
p6 <- round(2721/3000,4)
paste("P(Loan = 0)=", p6*100,"%")
```

```
## [1] "P(Loan = 0)= 90.7 %"
```

e.P(Loan = 1 |CC = 1, Online = 1) = P(CC =1 | L=1)P(Online=1 | L=1)P(L=1) /(P(CC =1 | L=1) P(Online=1 | L=1) P(L=1)+P(CC =1 | L=0) P(Online=1 | L=0) P(L=0))

```
#e
#compute the naive bayes probability

p <- round((p1*p2*p3)/((p1*p2*p3)+(p4*p5*p6)),5)
paste("P(Loan = 1 |CC = 1, Online = 1) =", p*100,"%")
```

```
## [1] "P(Loan = 1 |CC = 1, Online = 1) = 9.138 %"
```

f.**Compare this value with the one obtained from the pivot table in (b). Which is a more accurate estimate?** From (b), the exact bayes conditional probability that this customer will accept the loan offer is 9.155%". From (e), the naive bayes conditional probability that this customer will accept the loan offer is 9.138%. The redsult from (e) is more accurate than the result from (b) because we used the entire training dataset to calculate the condistional probabilities in part (e), and we no longer restrict the probablity calculation to those records that match the resord to be classified.

```
#g
#run the NB model on training data
nb_university <-naiveBayes(PersonalLoan ~ CreditCard + Online, train_data)
nb_university
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     0     1
## 0.907 0.093
##
## Conditional probabilities:
##    CreditCard
## Y           0         1
##   0 0.6821022 0.3178978
##   1 0.6917563 0.3082437
##
##    Online
## Y           0         1
##   0 0.4083058 0.5916942
##   1 0.4014337 0.5985663
```

```
pred_prob <- predict(nb_university, newdata = train_data, type = "raw")


#find the entry that corresponds to P(Loan = 1 | CC = 1, Online = 1)
df <- data.frame(CC=train_data$CreditCard,
                 Online=train_data$Online,actual=train_data$PersonalLoan,p
        red_prob)
df1 <- df %>% filter(CC=="1",Online=="1",actual=="1")
df1
```

CC <fctr>	Online <fctr>	actual <fctr>	X0 <dbl>	X1 <dbl>
1	1	1	0.9086146	0.09138543
1	1	1	0.9086146	0.09138543
1	1	1	0.9086146	0.09138543
1	1	1	0.9086146	0.09138543
1	1	1	0.9086146	0.09138543

CC <fctr>	Online <fctr>	actual <fctr>	X0 <dbl>	X1 <dbl>
1	1	1	0.9086146	0.09138543
1	1	1	0.9086146	0.09138543
1	1	1	0.9086146	0.09138543
1	1	1	0.9086146	0.09138543
1	1	1	0.9086146	0.09138543

1-10 of 52 rows Previous **1** 2 3 4 5 6 Next

g.**Examine the model output on training data, and find the entry that corresponds to P(Loan = 1 | CC = 1, Online = 1).Compare this to the number you obtained in (e)** As you can see the X1 from table df1, which represents the probability(P(Loan = 1 | CC = 1, Online = 1)). The probability is around 9.138%. Compare this to the result from part (e) (probability is 9.138%), we can make the conclusion that naive bayes conditional probability is very accurate.

Problem 8.2

```
#problem 8.2
#NB is only suitful for categorical predictor variables
library(readr)
accident <- read_csv("Accidents.csv")
```

```
## Parsed with column specification:
## cols(
##   RushHour = col_double(),
##   WRK_ZONE = col_double(),
##   WKDY = col_double(),
##   INT_HWY = col_double(),
##   LGTCON_day = col_double(),
##   LEVEL = col_double(),
##   SPD_LIM = col_double(),
##   SUR_COND_dry = col_double(),
##   TRAF_two_way = col_double(),
##   WEATHER_adverse = col_double(),
##   MAX_SEV = col_character()
## )
```

```
#insert a column INJURY
accident1 <- mutate(accident, INJURY = ifelse(accident$MAX_SEV == "no-inju
        ry", 0, 1))

#convert variables to categorical type
for (i in c(1:dim(accident1)[2])){
  accident1[,i] = data.frame(apply(accident1[i], 2, as.factor))
}
str(accident1)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 600 obs. of  1
2 variables:
##  $ RushHour       : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 1 2 ...
##  $ WRK_ZONE       : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ WKDY           : Factor w/ 2 levels "0","1": 2 2 1 2 2 2 2 2 2 2 ...
##  $ INT_HWY        : Factor w/ 3 levels "0","1","9": 2 1 1 1 1 1 1 1 2 2
...
##  $ LGTCON_day     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ LEVEL          : Factor w/ 2 levels "0","1": 2 1 1 2 1 1 1 2 1 1 ...
##  $ SPD_LIM        : Factor w/ 14 levels "10","15","20",..: 13 10 6 6 4
6 11 8 10 13 ...
##  $ SUR_COND_dry   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 2 2 2 ...
##  $ TRAF_two_way   : Factor w/ 2 levels "0","1": 1 2 1 1 1 1 1 2 1 1 ...
##  $ WEATHER_adverse: Factor w/ 2 levels "0","1": 2 1 2 2 2 2 1 1 1 1 ...
##  $ MAX_SEV        : Factor w/ 3 levels "fatal","no-injury",..: 2 3 2 2
3 3 2 3 2 3 ...
##  $ INJURY         : Factor w/ 2 levels "0","1": 1 2 1 1 2 2 1 2 1 2 ...
```

```
#part a
library(dplyr)
paste("Total number of Non-injury:" ,
      length(which(accident$MAX_SEV == "no-injury")))
```

```
## [1] "Total number of Non-injury: 292"
```

```
paste("Total number of Injury:",
      length(which(accident$MAX_SEV == "non-fatal" | accident$MAX_SEV ==
         "fatal")))
```

```
## [1] "Total number of Injury: 308"
```

```
#calculate the probability of injury
a <- 308/(308+292)*100
a
```

```
## [1] 51.33333
```

a.**Using the information in this dataset, if an accident has just been reported and no further information is available, what should the prediction be? (INJURY = Yes or No?) Why?** Since ~51% of the accidents in our data set resulted in an accident, we should predict that an accident will result in injury because it is slightly more likley.

```
#part b (i)
#create a new subset with only the required records
accident2 <- accident1[1:12, c("INJURY", "WEATHER_adverse", "TRAF_two_way"
        )]

#install.packages("rpivotTable")
library(rpivotTable)
rpivotTable::rpivotTable(accident2,rows = "INJURY", cols = c("WEATHER_adve
        rse","TRAF_two_way"),
            width = "100%", height="400%")
```

Table

Count ↕ ↔ WEATHER_adverse TRAF_two_way

INJURY

	WEATHER_adverse	0		1	Totals
	TRAF_two_way	0	1	0	
INJURY					
0		3		3	6
1		2	2	2	6
	Totals	5	2	5	12

```
#part b (ii)
#Exact Bayes Conditional Probabilities of an injury

#1 P(Injury=1|WEATHER_R = 1, TRAF_CON_R =0):
p1 <- 2/5
p1
```

```
## [1] 0.4
```

```
#2 P(Injury=1|WEATHER_R = 1, TRAF_CON_R =1):
p2 <- 0
p2
```

```
## [1] 0
```

```
#3 P(Injury=1|WEATHER_R = 0, TRAF_CON_R =0):
p3 <- 2/5
p3
```

```
## [1] 0.4
```

```
#4 P(Injury=1|WEATHER_R = 0, TRAF_CON_R =1):
p4 <- 1
p4
```

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

```
#part b (iii)
#Classify the 12 accidents using these probabilities and a cutoff of 0.5
#Insert the probability according to the result above
accident2$Prob_INJURY <- c(0.6,1,0.6,0.6,0.4,0.4,0.6,1,0.6,0.4,0.6,0.4)
#Insert the prediction according to the pro_INJURY with a cutoff of 0.5
accident2 <- mutate(accident2, Predict_class = ifelse(Prob_INJURY>0.5, 1,
        0))
accident2$Predict_class <- as.factor(accident2$Predict_class)
str(accident2)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    12 obs. of  5 variables:
##  $ INJURY         : Factor w/ 2 levels "0","1": 1 2 1 1 2 2 1 2 1 2 ...
##  $ WEATHER_adverse: Factor w/ 2 levels "0","1": 2 1 2 2 2 2 1 1 1 1 ...
##  $ TRAF_two_way   : Factor w/ 2 levels "0","1": 1 2 1 1 1 1 1 2 1 1 ...
##  $ Prob_INJURY    : num  0.6 1 0.6 0.6 0.4 0.4 0.6 1 0.6 0.4 ...
##  $ Predict_class  : Factor w/ 2 levels "0","1": 2 2 2 2 1 1 2 2 2 1 ...
```

```
#part b (vi)
#Compute manually the naive Bayes conditional probability of an injury giv
        en WEATHER_R = 1 and TRAF_CON_R = 1.
p <- (1/3)*(1/3)*(1/2)/((1/3)*(1/3)*(1/2)+(1/2)*0*(1/2))
p
```

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

```
#part b (v)
#Run a naive Bayes classifier on the 12 records
set.seed(105)
library(e1071)
accident2_nb <- naiveBayes(INJURY ~ WEATHER_adverse + TRAF_two_way, accide
        nt2)

#Check the model output to obtain probabilities and classifications for al
        l 12 records
pred_prob <- predict(accident2_nb, newdata = accident2, type = "raw")
pred_class <- predict(accident2_nb, newdata = accident2)

#Compare this to the exact Bayes classification
library(caret)
confusionMatrix(pred_class,accident2$Predict_class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##          0 4 6
##          1 0 2
##
##                Accuracy : 0.5
##                  95% CI : (0.2109, 0.7891)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : 0.93355
##
##                   Kappa : 0.1818
##
##  Mcnemar's Test P-Value : 0.04123
##
##             Sensitivity : 1.0000
##             Specificity : 0.2500
##          Pos Pred Value : 0.4000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3333
##          Detection Rate : 0.3333
##    Detection Prevalence : 0.8333
##       Balanced Accuracy : 0.6250
##
##        'Positive' Class : 0
##
```

```
#compare this with the true classification
confusionMatrix(pred_class,accident2$INJURY)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction 0 1
##          0 6 4
##          1 0 2
##
##                Accuracy : 0.6667
##                  95% CI : (0.3489, 0.9008)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.1938
##
##                   Kappa : 0.3333
##
##  Mcnemar's Test P-Value : 0.1336
##
##             Sensitivity : 1.0000
##             Specificity : 0.3333
##          Pos Pred Value : 0.6000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5000
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.8333
##       Balanced Accuracy : 0.6667
##
##        'Positive' Class : 0
##
```

b.**Compare this to the exact Bayes classification. Are the resulting classifications equivalent? Is the ranking (= ordering) of observations equivalent?** As you can see from the results of the confusion matrix, the accuracy is 50% comapring the NB model result with exact bayes classifiction, and the accuracy is 66.7% comparing the NB model result with true response classification. The resulting classifications are not equivalent and the ranking of observations are not equivalent. Since we trained on the same data we are testing, it is expected that the trained data performs better than our manual calculations.

```
#part c (i)
#Choose the preditors
predictor <- c("INJURY","RushHour", "WRK_ZONE", "WKDY", "INT_HWY", "LGTCON
        _day", "LEVEL", "SPD_LIM", "SUR_COND_dry", "TRAF_two_way", "WEATH
        ER_adverse")
accident3 <- accident1[,predictor]
```

```r
#part c (ii)
#split the dataset
set.seed(105)
train.index <- sample(c(1:dim(accident3)[1]), dim(accident3)[1]*0.6)
train.df <- accident3[train.index,]
valid.df <- accident3[-train.index,]

nb_train <- naiveBayes(INJURY ~., data=train.df)
nb_train
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##         0         1
## 0.5027778 0.4972222
##
## Conditional probabilities:
##    RushHour
## Y           0         1
##   0 0.5635359 0.4364641
##   1 0.5865922 0.4134078
##
##    WRK_ZONE
## Y            0          1
##   0 0.98342541 0.01657459
##   1 0.98324022 0.01675978
##
##    WKDY
## Y           0         1
##   0 0.2154696 0.7845304
##   1 0.2458101 0.7541899
##
##    INT_HWY
## Y             0           1           9
##   0 0.850828729 0.149171271 0.000000000
##   1 0.865921788 0.128491620 0.005586592
##
##    LGTCON_day
## Y           0         1
##   0 0.2651934 0.7348066
##   1 0.2625698 0.7374302
##
##    LEVEL
## Y           0         1
##   0 0.7734807 0.2265193
##   1 0.7877095 0.2122905
##
##    SPD_LIM
## Y             10          15          20          25          30
35
##   0 0.000000000 0.000000000 0.016574586 0.082872928 0.055248619 0.22651
9337
##   1 0.005586592 0.000000000 0.005586592 0.067039106 0.072625698 0.21229
0503
```

```
##    SPD_LIM
## Y           40          45          50          55          60
65
##   0 0.093922652 0.176795580 0.033149171 0.209944751 0.027624309 0.01104
9724
##   1 0.122905028 0.217877095 0.022346369 0.111731844 0.067039106 0.07262
5698
##    SPD_LIM
## Y           70          75
##   0 0.060773481 0.005524862
##   1 0.016759777 0.005586592
##
##    SUR_COND_dry
## Y          0         1
##   0 0.2486188 0.7513812
##   1 0.1843575 0.8156425
##
##    TRAF_two_way
## Y          0         1
##   0 0.4419890 0.5580110
##   1 0.3910615 0.6089385
##
##    WEATHER_adverse
## Y          0         1
##   0 0.8011050 0.1988950
##   1 0.8715084 0.1284916
```

```
#generate the confusion matrix using the train.df, the prediction and the
         classes
confusionMatrix(predict(nb_train, train.df),train.df$INJURY)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  97  58
##          1  84 121
##
##                Accuracy : 0.6056
##                  95% CI : (0.553, 0.6564)
##     No Information Rate : 0.5028
##     P-Value [Acc > NIR] : 5.638e-05
##
##                   Kappa : 0.2117
##
##  Mcnemar's Test P-Value : 0.03591
##
##             Sensitivity : 0.5359
##             Specificity : 0.6760
##          Pos Pred Value : 0.6258
##          Neg Pred Value : 0.5902
##              Prevalence : 0.5028
##          Detection Rate : 0.2694
##    Detection Prevalence : 0.4306
##       Balanced Accuracy : 0.6059
##
##        'Positive' Class : 0
##
```

```
error_train <- 1-0.6056
paste("train error:", error_train*100, "%")
```

```
## [1] "train error: 39.44 %"
```

```
#part c (iii)
#What is the overall error for the validation set
confusionMatrix(predict(nb_train, valid.df),valid.df$INJURY)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 51 50
##          1 60 79
##
##                Accuracy : 0.5417
##                  95% CI : (0.4764, 0.6059)
##     No Information Rate : 0.5375
##     P-Value [Acc > NIR] : 0.4748
##
##                   Kappa : 0.0723
##
##  Mcnemar's Test P-Value : 0.3908
##
##             Sensitivity : 0.4595
##             Specificity : 0.6124
##          Pos Pred Value : 0.5050
##          Neg Pred Value : 0.5683
##              Prevalence : 0.4625
##          Detection Rate : 0.2125
##    Detection Prevalence : 0.4208
##       Balanced Accuracy : 0.5359
##
##        'Positive' Class : 0
##
```

```
# accuracy = 0.5417
 error_valid <- 1-0.5417
paste("valid error:", error_valid*100, "%")
```

```
## [1] "valid error: 45.83 %"
```

```
#part c (iv) What is the percent improvement relative to the naive rule (u
        sing the validation set))
imp <-  error_valid-error_train
paste("The percent improvement is ",scales::percent(imp,0.01))
```

```
## [1] "The percent improvement is  6.39%"
```

part c. (v) **Examine the conditional probabilities output. Why do we get a probability of zero for P(INJURY = No | SPD_LIM = 5)?** As you can see the result from partc (iii); the output of nb_train, there is conditional probability for SPD_LIM = 5. Therefore, this probability (P(INJURY = No | SPD_LIM = 5)) is zero.