AML_HomeWork1_Part1_Naive_Bayes

Himanshu Shah hs8

January 27, 2018

Homework 1 Problem 1

This is an explanation of Homework 1 Problem 1. Please do your coding in the R language for this assignment. You will have a place to upload your code with the submission.

The UC Irvine machine learning data repository hosts a famous collection of data on whether a patient has diabetes, the Pima Indians dataset, originally owned by the National Institute of Diabetes and Digestive and Kidney Diseases and donated by Vincent Sigillito. The site is located here: [http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes (http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes)]

You should look over the site and check the description of the data. In the “Data Folder” directory, the primary file you need is named “pima-indians-diabetes.data”. This data has a set of attributes of patients, and a categorical variable telling whether the patient is diabetic or not. For several attributes in this data set, a value of 0 may indicate a missing value of the variable.

Part A

Build a simple naive Bayes classifier to classify this data set. We will use 20% of the data for evaluation and the other 80% for training. There are a total of 768 data-points.

You should use a normal distribution to model each of the class-conditional distributions. You should write this classifier yourself (it’s quite straight-forward).

Report the accuracy of the classifier on the 20% evaluation data, where accuracy is the number of correct predictions as a fraction of total predictions.

Train/Test Scores calculation

```
wdat <- read.csv("pima-indians-diabetes.data", header=FALSE)
bigx <- wdat[, -c(9)] # matrix of features
bigy <- wdat[, 9]     # labels; class value 1 means "tested positive for diabetes"

trscorea <- array(dim = 10)
tescorea <- array(dim = 10)

for (wi in 1:10) { #Make 10 loops and mean to get average
#  set.seed(2018)
  wtd <- createDataPartition(y = bigy, p = 0.8, list = FALSE) # Split 80% of the data into training, 20% into test
  nbx <- bigx                               # matrix of features
  ntrbx <- nbx[wtd, ]                       # training features
  ntrby <- bigy[wtd]                        # training labels
  ntebx <- nbx[-wtd, ]                      # test rows - features
  nteby <- bigy[-wtd]                       # test rows - labels

  trposflag <- ntrby > 0                    # training labels for diabetes positive, (index of observation which has posit
ive result)
  ptregs <- ntrbx[trposflag, ]              # training rows features with diabetes positive (Extract raws which has positi
ve)
  ntregs <- ntrbx[!trposflag, ]             # training rows features with diabetes negative (Extract raws which has negati
ve)
  ptrLength = sum(ntrby == TRUE)               #Postiive Training dataset length
  ntrLength = sum(ntrby == FALSE)              #Negative Training dataset length
  pteLength = sum(nteby == TRUE)               #Postiive Training dataset length
  nteLength = sum(nteby == FALSE)              #Negative Training dataset length


# For Gaussian, You need to find mean and standard deviaiton first for each classifier (Classifier is diabetes postiive or n
egative)
  ptrmean <- sapply(ptregs, mean, na.rm = T)  # vector of means for training, diabetes positive
  ntrmean <- sapply(ntregs, mean, na.rm = T)  # vector of means for training, diabetes negative
  ptrsd   <- sapply(ptregs, sd, na.rm = T)    # vector of sd for training, diabetes positive
  ntrsd   <- sapply(ntregs, sd, na.rm = T)    # vector of sd for training, diabetes negative

#It is always good idea to noramlize data, to avoid one variable/feature overshadowing other variable because of very large
 value
  ptroffsets <- t(t(ntrbx) - ptrmean)         # first step normalize training diabetes pos, subtract mean
  ptrscales  <- t(t(ptroffsets) / ptrsd)      # second step normalize training diabetes pos, divide by sd
#Use log likelihood (Cond probability) to avoid multiplying large probabilities.
```

```
ptrlogs    <- -(1/2) * rowSums(apply(ptrscales, c(1,2),
              function(x) x^2), na.rm = T) - sum(log(ptrsd))  # Log likelihoods based on
                                                                    # normal distr. for diabetes positive
ntroffsets <- t(t(ntrbx) - ntrmean)
ntrscales  <- t(t(ntroffsets) / ntrsd)
ntrlogs    <- -(1/2) * rowSums(apply(ntrscales, c(1,2)
                                  , function(x) x^2), na.rm = T) - sum(log(ntrsd))
                                                        # Log likelihoods based on
                                                                    # normal distr for diabetes negative
                                                        # (It is done separately on each class)
#Now that We have calculated likelihood(Cond probability), next calculate positive prior
ptrPrior <- ptrLength / (ptrLength + ntrLength) #Positive  Prior
ptrPriorlog <- log(ptrPrior)                #Positive  log Prior
ntrPrior <- ntrLength / (ptrLength + ntrLength) #Negative  Prior
ntrPriorlog <- log(ntrPrior)                #Negative  log Prior

# Rows classified as diabetes positive by classifier
lvwtr      <- (ptrlogs + ptrPriorlog) > (ntrlogs +  ntrPriorlog)
#lvwtr      <- ptrlogs > ntrlogs              # Rows classified as diabetes positive by classifier
gotrighttr <- lvwtr == ntrby                 # compare with true labels
trscorea[wi] <- sum(gotrighttr)/(sum(gotrighttr)+sum(!gotrighttr)) # Accuracy with training set

##Now Repeat above for test observations
pteoffsets <- t(t(ntebx)-ptrmean)            # Normalize test dataset with parameters from training
ptescales  <- t(t(pteoffsets)/ptrsd)
ptelogs    <- -(1/2)*rowSums(apply(ptescales,c(1, 2)
                                , function(x)x^2), na.rm=TRUE)-sum(log(ptrsd))

nteoffsets <- t(t(ntebx)-ntrmean)            # Normalize again for diabetes negative class
ntescales  <- t(t(nteoffsets)/ntrsd)
ntelogs    <- -(1/2)*rowSums(apply(ntescales,c(1, 2)
                                , function(x)x^2), na.rm=TRUE)-sum(log(ntrsd))

#Now that We have calculated likelihood(Cond probability), next calculate positive prior
ptePrior <- pteLength / (pteLength + nteLength) #Positive  Prior
ptePriorlog <- log(ptePrior)                #Positive  log Prior
ntePrior <- nteLength / (pteLength + nteLength) #Negative  Prior
ntePriorlog <- log(ntePrior)                #Negative  log Prior

lvwte<-(ptelogs + ptrPriorlog) > (ntelogs + ntrPriorlog)
#lvwte<-ptelogs>ntelogs
```

```
  gotright<-lvwte==nteby
  tescorea[wi] <- sum(gotright)/(sum(gotright)+sum(!gotright))  # Accuracy on the test set
}
 #mean(trscorea)
 #mean(tescorea)
```

Accuracy for part A model for 20% test data is 0.7457516 compared to train data is 0.7650407.

Part B

Now adjust your code so that, for attribute 3 (Diastolic blood pressure), attribute 4 (Triceps skin fold thickness), attribute 6 (Body mass index), and attribute 8 (Age), it regards a value of 0 as a missing value when estimating the class-conditional distributions, and the posterior. R uses a special number NA to flag a missing value. Most functions handle this number in special, but sensible, ways; but you'll need to do a bit of looking at manuals to check.

Report the accuracy of the classifier on the 20% that was held out for evaluation.

Train/Test Scores calculation with na removed

```
wdat <- read.csv("pima-indians-diabetes.data", header=FALSE)

bigx <- wdat[, -c(9)] # matrix of features
bigy <- wdat[, 9]     # labels; class value 1 means "tested positive for diabetes"

#Replacing row's 3,4,6 and 8 value of 0 with NA
bigx[,3][bigx[,3] == 0] <- NA
bigx[,4][bigx[,4] == 0] <- NA
bigx[,6][bigx[,6] == 0] <- NA
bigx[,8][bigx[,8] == 0] <- NA

trscoreb <- array(dim = 10)
tescoreb <- array(dim = 10)

for (wi in 1:10) { #Make 10 loops and mean to get average
#  set.seed(2018)
  wtd <- createDataPartition(y = bigy, p = 0.8, list = FALSE) # Split 80% of the data into training, 20% into test
  nbx <- bigx                                # matrix of features
  ntrbx <- nbx[wtd, ]                        # training features
  ntrby <- bigy[wtd]                         # training labels

  trposflag <- ntrby > 0                     # training labels for diabetes positive, (index of observation which has posit
ive result)
  ptregs <- ntrbx[trposflag, ]               # training rows features with diabetes positive (Extract raws which has positi
ve)
  ntregs <- ntrbx[!trposflag, ]              # training rows features with diabetes negative (Extract raws which has negati
ve)
  ptrLength = sum(ntrby == TRUE, na.rm = T)               #Postiive Training dataset length
  ntrLength = sum(ntrby == FALSE, na.rm = T)              #Negative Training dataset length

  ntebx <- nbx[-wtd, ]                       # test rows - features
  nteby <- bigy[-wtd]                        # test rows - labels

# For Gaussian, You need to find mean and standard deviaiton first for each classifier (Classifier is diabetes postiive or n
egative)
  ptrmean <- sapply(ptregs, mean, na.rm = T)  # vector of means for training, diabetes positive
  ntrmean <- sapply(ntregs, mean, na.rm = T)  # vector of means for training, diabetes negative
  ptrsd   <- sapply(ptregs, sd, na.rm = T)    # vector of sd for training, diabetes positive
  ntrsd   <- sapply(ntregs, sd, na.rm = T)    # vector of sd for training, diabetes negative
```

```
#It is always good idea to noramlize data, to avoid one variable/feature overshadowing other variable because of very large
 value
  ptroffsets <- t(t(ntrbx) - ptrmean)         # first step normalize training diabetes pos, subtract mean
  ptrscales  <- t(t(ptroffsets) / ptrsd)      # second step normalize training diabetes pos, divide by sd
#Use log likelihood (Cond probability) to avoid multiplying large probabilities.
  ptrlogs    <- -(1/2) * rowSums(apply(ptrscales, c(1,2),
                function(x) x^2), na.rm = T) - sum(log(ptrsd))  # Log likelihoods based on

  #Repeat for negative observations                                                              # normal distr.
 for diabetes positive
  ntroffsets <- t(t(ntrbx) - ntrmean)
  ntrscales  <- t(t(ntroffsets) / ntrsd)
  ntrlogs    <- -(1/2) * rowSums(apply(ntrscales, c(1,2)
                                   , function(x) x^2), na.rm = T) - sum(log(ntrsd))
                                                  # Log likelihoods based on
                                                                # normal distr for diabetes negative
                                                  # (It is done separately on each class)
  #Now that We have calculated likelihood(Cond probability), next calculate positive prior
  ptrPrior <- ptrLength / (ptrLength + ntrLength) #Positive  Prior
  ptrPriorlog <- log(ptrPrior)                #Positive  log Prior
  ntrPrior <- ntrLength / (ptrLength + ntrLength) #Negative  Prior
  ntrPriorlog <- log(ntrPrior)                #Negative  log Prior

  # Rows classified as diabetes positive by classifier
  lvwtr      <- (ptrlogs + ptrPriorlog) > (ntrlogs +  ntrPriorlog)
  #lvwtr      <- ptrlogs > ntrlogs              # Rows classified as diabetes positive by classifier
  gotrighttr <- lvwtr == ntrby                 # compare with true labels
  trscoreb[wi] <- sum(gotrighttr)/(sum(gotrighttr)+sum(!gotrighttr)) # Accuracy with training set

  ##Now Repeat above for test observations
  pteoffsets <- t(t(ntebx)-ptrmean)            # Normalize test dataset with parameters from training
  ptescales  <- t(t(pteoffsets)/ptrsd)
  ptelogs    <- -(1/2)*rowSums(apply(ptescales,c(1, 2)
                                  , function(x)x^2), na.rm=TRUE)-sum(log(ptrsd))

  nteoffsets <- t(t(ntebx)-ntrmean)            # Normalize again for diabetes negative class
  ntescales  <- t(t(nteoffsets)/ntrsd)
  ntelogs    <- -(1/2)*rowSums(apply(ntescales,c(1, 2)
                                  , function(x)x^2), na.rm=TRUE)-sum(log(ntrsd))

  lvwte<-(ptelogs + ptrPriorlog) > (ntelogs + ntrPriorlog)
```

```
  #lvwte<-ptelogs>ntelogs
  gotright<-lvwte==nteby
  tescoreb[wi] <- sum(gotright)/(sum(gotright)+sum(!gotright))  # Accuracy on the test set
}
 #mean(trscorea)
 #mean(tescorea)
 #mean(trscoreb)
 #mean(tescoreb)
```

Accuracy for part B model for 20% test data is 0.7509804 compared to train data is 0.7523577.

Part C

Now use the caret and klaR packages to build a naive bayes classifier for this data, assuming that no attribute has a missing value. The caret package does cross-validation (look at train) and can be used to hold out data. You should do 10-fold cross-validation:

The klaR package can estimate class-conditional densities using a density estimation procedure that I will describe much later in the course. I have not been able to persuade the combination of caret and klaR to handle missing values the way I'd like them to, but that may be ignorance (look at the na.action argument).

Report the accuracy of the classifier on the held out 20%.

```
wdat <- read.csv("pima-indians-diabetes.data", header=FALSE)

bigx<-wdat[,-c(9)] # matrix of features
bigy<-as.factor(wdat[,9]) # labels

wtd<-createDataPartition(y=bigy, p=.8, list=FALSE)
trax<-bigx[wtd,]
tray<-bigy[wtd]

modelcaret<-train(trax, tray, 'nb', trControl=trainControl(method='cv', number=10))
teclassescaret<-predict(modelcaret,newdata=bigx[-wtd,])
#confusionMatrix(data=teclassescaret, bigy[-wtd])

tescorec = sum(teclassescaret==bigy[-wtd])/(sum(teclassescaret==bigy[-wtd])+sum(!(teclassescaret==bigy[-wtd])))
#tescorec
```

Accuracy for part C model for 20% test data is 0.7581699.

Part D

Now install SVMLight, which you can find at http://svmlight.joachims.org (http://svmlight.joachims.org), via the interface in klaR (look for svmlight in the manual) to train and evaluate an SVM to classify this data. For training the model, use:

You don’t need to understand much about SVMs to do this - we’ll do that in following exercises. You should NOT substitute NA values for zeros for attributes 3, 4, 6, and 8.

Using the predict function in R, report the accuracy of the classifier on the held out 20%.

Hint: If you are having trouble invoking svmlight from within R Studio, make sure your svmlight executable directory is added to your system path. Here are some instructions about editing your system path on various operating systems: https://www.java.com/en/download/help/path.xml (https://www.java.com/en/download/help/path.xml) You would need to restart R Studio (or possibly restart your computer) afterwards for the change to take effect.

```
wdat <- read.csv("pima-indians-diabetes.data", header=FALSE)

bigx <- wdat[,-c(9)]
bigy<-as.factor(wdat[,9])

wtd<-createDataPartition(y=bigy, p=.8, list=FALSE)
trax<-bigx[wtd,]
tray<-bigy[wtd]

svm<-svmlight(trax, tray, pathsvm='D:/Downloads/AML/HomeWork1')
labels<-predict(svm, bigx[-wtd,])
#confusionMatrix(data=labels$class, bigy[-wtd])

foo<-labels$class
tescored = sum(foo==bigy[-wtd])/(sum(foo==bigy[-wtd])+sum(!(foo==bigy[-wtd])))
#tescored
```

Accuracy for part D model for 20% test data is 0.7647059.

```
#library(xtable)
library(knitr)
values = c(mean(tescorea), mean(tescoreb), tescorec, tescored)
summaryModels <- as.matrix(values)
colnames(summaryModels)  <- c( "Test Accuracy")
rownames(summaryModels)  <-c("A: Naive Bayes wo NA","B: Naive Bayes with NA", "C: Naive Bayes using algo", "D: Naive Bayes u
sing svm")
#X
```

Summary of all 4 models for 20% test data is.

```
summaryModels
```

```
##                          Test Accuracy
## A: Naive Bayes wo NA          0.7457516
## B: Naive Bayes with NA        0.7509804
## C: Naive Bayes using algo     0.7581699
## D: Naive Bayes using svm      0.7647059
```