CS498 Applied Machine Learning Homework 1

Huamin Zhang

Monday, Jan 30, 2017

Name: Huamin Zhang

NetID: huaminz2

UIN: 677905090

3.1

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. This can be found at http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes. 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.

(a)

Build a simple naive Bayes classifier to classify this data set. You should hold out 20% of the data for evaluation, and use the other 80% for training. 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), but you may find the function createDataPartition in the R package caret helpful to get the random partition.

```
setwd("C:/Users/98302/Desktop")
set.seed(3)
library(klaR)
```

```
## Loading required package: MASS
```

```
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
#load data file
read.csv("pima-indians-diabetes.data",header=F)->data
head(data)
```

```
##   V1  V2 V3 V4  V5   V6    V7 V8 V9
## 1  6 148 72 35   0 33.6 0.627 50  1
## 2  1  85 66 29   0 26.6 0.351 31  0
## 3  8 183 64  0   0 23.3 0.672 32  1
## 4  1  89 66 23  94 28.1 0.167 21  0
## 5  0 137 40 35 168 43.1 2.288 33  1
## 6  5 116 74  0   0 25.6 0.201 30  0
```

```
input <- data[,-9]#extract the input data
output <- data[,9]#extract the output data
accuracy_train<-array(dim=10)
accuracy_test<-array(dim=10)
sensitivity<-array(dim=10)
specificity<-array(dim=10)
#run 10 times
for(i in 1:10){
  #get the random partition tag
  createDataPartition(output, p=.8, list=FALSE) ->tag
  #80% is training data
  #Set up train and test splits
  input[tag,] -> train_input
  input[-tag,] -> test_input
  output[tag] -> train_output
  output[-tag] -> test_output
  #Set up positive and negative training splits
  train_output == 0 -> negative_tag
  train_output == 1 -> positive_tag
  train_input[negative_tag,] -> negative_train_input
  train_input[positive_tag,] -> positive_train_input
  #calculate log(P(+)) and log(P(-))
  log_p_positive <- log(sum(positive_tag)/length(train_output))
  log_p_negative <- log(sum(negative_tag)/length(train_output))
  #compute the raw model params of +/- examples
  #According to the training data, we compute the model params, positive_mean,
  #positive_sd, negative_mean, negative_sd, which is the mean and standard deviation
  #of each feature given +/- examples. These params will be used to compute the
  #posterior probability
  positive_mean <- sapply(positive_train_input,mean,na.rm=T)
  positive_sd <- sapply(positive_train_input,sd,na.rm=T)
  negative_mean <- sapply(negative_train_input,mean,na.rm=T)
  negative_sd <- sapply(negative_train_input,sd,na.rm=T)
  #The classifier created from the training set using a Gaussian distribution assumption.
  #Assume all attributes in training data are normal distribution
  #(Actually we know some of them may not be)
  #Transform to Standard normal distribution so that we can calculate
  #the probability easily.
  positive_tr_input <- t((t(train_input)-positive_mean) / positive_sd)
  negative_tr_input <- t((t(train_input)-negative_mean ) / negative_sd)
  #calculate the log(P(training example|+)) and log(P(training example|-))
  positive_tr_logP <- rowSums(apply(positive_tr_input,1:2,function(x)
                      log(dnorm(x))),na.rm = T) + log_p_positive
  negative_tr_logP <- rowSums(apply(negative_tr_input,1:2,function(x)
                      log(dnorm(x))),na.rm = T) + log_p_negative
  #compare log(P(training example|+)) and log(P(training example|-))
  positive_tr_logP > negative_tr_logP -> tr_p_tag
  #calculate the accuracy of training data
  accuracy_train[i] <- sum(tr_p_tag == train_output) / length(train_output)
  #predict the label of test data
  positive_te_input <- t((t(test_input)-positive_mean) / positive_sd)
  negative_te_input <- t((t(test_input)-negative_mean ) / negative_sd)
  positive_te_logP <- rowSums(apply(positive_te_input,1:2,function(x)
```

```
                      log(dnorm(x))),na.rm = T) + log_p_positive
  negative_te_logP <- rowSums(apply(negative_te_input,1:2,function(x)
                      log(dnorm(x))),na.rm = T) + log_p_negative
  positive_te_logP > negative_te_logP -> te_p_tag
  #calculate the accuracy,sensitivity and specificity of test data
  result<-confusionMatrix(data=as.numeric(te_p_tag),test_output)
  accuracy_test[i] <- result$overall["Accuracy"]
  sensitivity[i] <- result$byClass["Sensitivity"]
  specificity[i] <- result$byClass["Specificity"]
}
```

We run 10 times and the following is the accuracy, sensitivity, specificity of the naive Bayes classifier

```
accuracy_train
```

```
##  [1] 0.7252033 0.7170732 0.7479675 0.7284553 0.7138211 0.7219512 0.7317073
##  [8] 0.7219512 0.7365854 0.7186992
```

```
accuracy_test
```

```
##  [1] 0.7647059 0.6993464 0.7581699 0.7254902 0.7712418 0.7843137 0.8104575
##  [8] 0.7450980 0.6993464 0.7647059
```

```
sensitivity
```

```
##  [1] 0.7741935 0.6565657 0.8039216 0.7058824 0.7755102 0.7766990 0.8315789
##  [8] 0.7474747 0.6875000 0.7570093
```

```
specificity
```

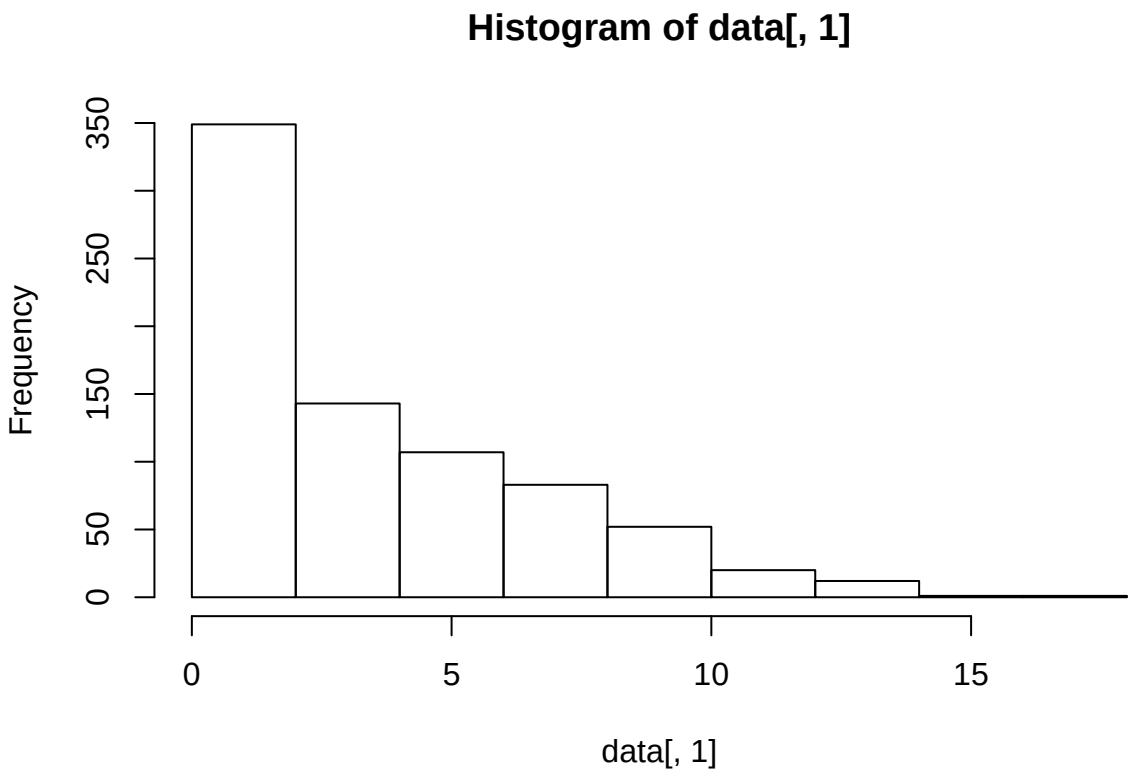
```
##  [1] 0.7500000 0.7777778 0.6666667 0.7647059 0.7636364 0.8000000 0.7758621
##  [8] 0.7407407 0.7192982 0.7826087
```

```
mean(accuracy_test)
```

```
## [1] 0.7522876
```

The mean accuracy of the test data is `0.7522876` The sensitivity is about 65-80% and Specificity is about 65-80%. ###Comment Do histgram of each feature in data, we will find that some of them are not normal distributionsuch as the first feature

```
hist(data[,1])
```



So if we can figure out a better distrubution, I think the result of predict will be better.

(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. Does this affect the accuracy of your classifier?

```
rm(list=ls())
setwd("C:/Users/98302/Desktop")
set.seed(3)
library(klaR)
library(caret)
#load data file
read.csv("pima-indians-diabetes.data",header=F)->data
input <- data[,-9]#extract the input data
output <- data[,9]#extract the output data
#change value=0 to NA
for (j in c(3,5,6,8)){
  input[,j] == 0 -> flag
  input[flag,j] <- NA
}
accuracy_train<-array(dim=10)
accuracy_test<-array(dim=10)
```

```
sensitivity<-array(dim=10)
specificity<-array(dim=10)
#run 10 times
for(i in 1:10){
  #get the random partition tag
  #80% is training data
  createDataPartition(output, p=.8, list=FALSE) ->tag
  #Set up train and test splits
  input[tag,] -> train_input
  input[-tag,] -> test_input
  output[tag] -> train_output
  output[-tag] -> test_output
  #Set up positive and negative training splits
  train_output == 0 -> negative_tag
  train_output == 1 -> positive_tag
  train_input[negative_tag,] -> negative_train_input
  train_input[positive_tag,] -> positive_train_input
  #calculate log(P(+)) and log(P(-))
  log_p_positive <- log(sum(positive_tag)/length(train_output))
  log_p_negative <- log(sum(negative_tag)/length(train_output))
  #compute the raw model params of +/- examples
  #According to the training data, we compute the model params, positive_mean,
  #positive_sd, negative_mean, negative_sd, which is the mean and standard deviation
  #of each feature given +/- examples. These params will be used to compute the
  #posterior probability
  positive_mean <- sapply(positive_train_input,mean,na.rm=T)
  positive_sd <- sapply(positive_train_input,sd,na.rm=T)
  negative_mean <- sapply(negative_train_input,mean,na.rm=T)
  negative_sd <- sapply(negative_train_input,sd,na.rm=T)
  #The classifier created from the training set using a Gaussian distribution assumption.
  #Assume all attributes in training data are normal distribution
  #(Actually we know some of them may not be)
  #Transform to Standard normal distribution so that we can calculate
  #the probability easily.
  positive_tr_input <- t((t(train_input)-positive_mean) / positive_sd)
  negative_tr_input <- t((t(train_input)-negative_mean ) / negative_sd)
  #calculate the log(P(training example|+)) and log(P(training example|-))
  positive_tr_logP <- rowSums(apply(positive_tr_input,1:2,function(x)                    log
  negative_tr_logP <- rowSums(apply(negative_tr_input,1:2,function(x)                    log
  #compare log(P(training example|+)) and log(P(training example|-))
  positive_tr_logP > negative_tr_logP -> tr_p_tag
  #calculate the accuracy of training data
  accuracy_train[i] <- sum(tr_p_tag == train_output) / length(train_output)
  #predict the label of test data
  positive_te_input <- t((t(test_input)-positive_mean) / positive_sd)
  negative_te_input <- t((t(test_input)-negative_mean ) / negative_sd)
  positive_te_logP <- rowSums(apply(positive_te_input,1:2,function(x)                    log
  negative_te_logP <- rowSums(apply(negative_te_input,1:2,function(x)                    log
  positive_te_logP > negative_te_logP -> te_p_tag
  #calculate the accuracy,sensitivity and specificity of test data
  result<-confusionMatrix(data=as.numeric(te_p_tag),test_output)
  accuracy_test[i] <- result$overall["Accuracy"]
  sensitivity[i] <- result$byClass["Sensitivity"]
```

```
  specificity[i] <- result$byClass["Specificity"]
}
```

Compared with (a), we regard a missing value as NA. We run 10 times and the following is the accuracy, sensitivity, specificity of the naive Bayes classifier

```
accuracy_train
```

```
##  [1] 0.7479675 0.7414634 0.7398374 0.7463415 0.7284553 0.7333333 0.7414634
##  [8] 0.7398374 0.7284553 0.7317073
```

```
accuracy_test
```

```
##  [1] 0.7320261 0.7124183 0.6993464 0.7189542 0.7516340 0.7320261 0.7124183
##  [8] 0.6993464 0.7124183 0.7058824
```

```
sensitivity
```

```
##  [1] 0.7684211 0.7472527 0.6981132 0.7383178 0.7717391 0.7500000 0.6960784
##  [8] 0.7608696 0.7040816 0.6862745
```

```
specificity
```

```
##  [1] 0.6724138 0.6612903 0.7021277 0.6739130 0.7213115 0.6981132 0.7450980
##  [8] 0.6065574 0.7272727 0.7450980
```

```
mean(accuracy_test)
```

```
## [1] 0.7176471
```

The mean accuracy of the test data is `0.7176471` which is a little less the result in (a) The sensitivity is about 70-80% and Specificity is about 60-75%.

(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. The klaR package can estimate class-conditional densities using a density estimation procedure that I will describe much later in the course. Use the cross-validation mechanisms in caret to estimate the accuracy of your classifier. 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).

```
rm(list=ls())
setwd("C:/Users/98302/Desktop")
set.seed(3)
library(klaR)
library(caret)
#load data file
```

```r
read.csv("pima-indians-diabetes.data",header=F)->data
input <- data[,-9]#extract the input data
output <- as.factor(data[,9])#extract the output data
#Set up train and test splits
#80% is training data
createDataPartition(output, p=.8, list=FALSE) ->tag
input[tag,] -> train_input
output[tag] -> train_output
#train the data with a naive bayes classifier
#use 10-fold cross-validation
#non-preProcessing
model<-train(train_input, train_output, 'nb', trControl=trainControl(method='cv', number=10))
model
```

```
## Naive Bayes
##
## 615 samples
##   8 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 553, 554, 553, 554, 553, 553, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.7561079  0.4467558
##    TRUE      0.7383131  0.4075398
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE
##  and adjust = 1.
```

```r
#predict the label of test data
predict_result<-predict(model,newdata=input[-tag,])
#calculate the accuracy, sensitivity and specificity of test data
confusionMatrix(data=predict_result, output[-tag])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 87 21
##          1 13 32
##
##                Accuracy : 0.7778
##                  95% CI : (0.7036, 0.8409)
##     No Information Rate : 0.6536
##     P-Value [Acc > NIR] : 0.000586
##
```

```
##                   Kappa : 0.4912
##  Mcnemar's Test P-Value : 0.229949
##
##             Sensitivity : 0.8700
##             Specificity : 0.6038
##          Pos Pred Value : 0.8056
##          Neg Pred Value : 0.7111
##              Prevalence : 0.6536
##          Detection Rate : 0.5686
##    Detection Prevalence : 0.7059
##       Balanced Accuracy : 0.7369
##
##        'Positive' Class : 0
##
```

Using the 10-fold cross-validation mechanisms, the accuracy of the naive bayes classifier was estimated as 77.8%. The sensitivity and Specificity is 0.8700 and 0.6038.

(d)

Now install SVMLight, which you can find at 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. You don't need to understand much about SVM's to do this — we'll do that in following exercises. You should hold out 20% of the data for evaluation, and use the other 80% for training. You should NOT substitute NA values for zeros for attributes 3, 4, 6, and 8.

```
rm(list=ls())
setwd("C:/Users/98302/Desktop")
set.seed(3)
library(klaR)
library(caret)
#load data file
read.csv("pima-indians-diabetes.data",header=F)->data
input <- data[,-9]#extract the input data
output <- as.factor(data[,9])#extract the output data
#Set up train and test splits
#80% is training data
createDataPartition(output, p=.8, list=FALSE) ->tag
#train the data with svm
svm_model <- svmlight(input[tag,], output[tag], pathsvm='C:/Users/98302/Desktop/svm_light_windows64')
#predict the label
result <- predict(svm_model, input[-tag,])
predict_label <- result$class
#calculate the accuracy
accuracy <- sum(predict_label == output[-tag])/ length(output[-tag])
accuracy
```

```
## [1] 0.7843137
```

```
#calculate sensitivity and specificity
test_output <- output[-tag]
result<-confusionMatrix(data=as.numeric(predict_label),as.numeric(test_output))
```

```
sensitivity <- result$byClass["Sensitivity"]
specificity <- result$byClass["Specificity"]
sensitivity
```

```
## Sensitivity
##        0.91
```

```
specificity
```

```
## Specificity
##   0.5471698
```

We hold out 20% of the data for testing, and use the other 80% for training. Using the SVM model, the accuracy of the naive bayes classifier was estimated as 78.4%. The sensitivity and specificity is `0.91` and `0.5471698`.

3.3

The UC Irvine machine learning data repository hosts a collection of data on heart disease. The data was collected and supplied by Andras Janosi, M.D., of the Hungarian Institute of Cardiology, Budapest; William Steinbrunn, M.D., of the University Hospital, Zurich, Switzerland; Matthias Pfisterer, M.D., of the University Hospital, Basel, Switzerland; and Robert Detrano, M.D., Ph.D., of the V.A. Medical Center, Long Beach and Cleveland Clinic Foundation. You can find this data at https://archive.ics.uci.edu/ml/datasets/Heart+Disease. Use the processed Cleveland dataset, where there are a total of 303 instances with 14 attributes each. The irrelevant attributes described in the text have been removed in these. The 14'th attribute is the disease diagnosis. There are records with missing attributes, and you should drop these.

(a)

Take the disease attribute, and quantize this into two classes, num = 0 and num > 0. Build and evaluate a naive bayes classifier that predicts the class from all other attributes Estimate accuracy by cross-validation. You should use at least 10 folds, excluding 15% of the data at random to serve as test data, and average the accuracy over those folds. Report the mean and standard deviation of the accuracy over the folds.

```
rm(list=ls())
setwd("C:/Users/98302/Desktop")
set.seed(3)
library(klaR)
library(caret)
accuracy_test<-array(dim=10)
sensitivity<-array(dim=10)
specificity<-array(dim=10)
#load data file
read.csv("processed.cleveland.data",header=F)->data
#remove records with missing attributes
rowSums(data == "?") == 0 -> remove
data <- data[remove,]
#Quantize the disease attribute into two classes,num = 0 and num > 0
data[,14] > 0 -> flag
data[flag,14] <- 1
```

```
input <- data[,-14]#extract the input data
output <- as.factor(data[,14])#extract the output data
#Set up train and test splits
#85% is training data
for(i in 1:10){
  createDataPartition(output, p=.85, list=FALSE) ->tag
  input[tag,] -> train_input
  output[tag] -> train_output
  #train the data with a naive bayes classifier
  #use 10-fold cross-validation
  #non-preProcessing
  model<-train(train_input, train_output, 'nb', trControl=trainControl(method='cv', number=10))
  model
  #predict the label of test data
  predict_result<-predict(model,newdata=input[-tag,])
  #calculate the accuracy, sensitivity and specificity of test data
  result <- confusionMatrix(data=predict_result, output[-tag])
  accuracy_test[i] <- result$overall["Accuracy"]
  sensitivity[i] <- result$byClass["Sensitivity"]
  specificity[i] <- result$byClass["Specificity"]
}
accuracy_test
```

```
##  [1] 0.8181818 0.7727273 0.8409091 0.9318182 0.8181818 0.8636364 0.7045455
##  [8] 0.7727273 0.8409091 0.8863636
```

```
sensitivity
```

```
##  [1] 0.8750000 0.7500000 0.9166667 0.9583333 0.8750000 0.9166667 0.7083333
##  [8] 0.7916667 0.8750000 0.9166667
```

```
specificity
```

```
##  [1] 0.75 0.80 0.75 0.90 0.75 0.80 0.70 0.75 0.80 0.85
```

```
mean(accuracy_test)
```

```
## [1] 0.825
```

```
sd(accuracy_test)
```

```
## [1] 0.06432706
```

Using the 10-fold cross-validation mechanisms, the mean of accuracy of the naive bayes classifier was estimated as 82.5%. The standard deviation of accuracy was 0.0643. The sensitivity is about 70-90% and Specificity is about 70-90%.

(b).

Now revise your classifier to predict each of the possible values of the disease attribute (0-4 as I recall). Estimate accuracy by cross-validation. You should use at least 10 folds, excluding 15% of the data at random to serve as test data, and average the accuracy over those folds. Report the mean and standard deviation of the accuracy over the folds.

```
rm(list=ls())
setwd("C:/Users/98302/Desktop")
set.seed(3)
library(klaR)
library(caret)
accuracy_test<-array(dim=10)
#load data file
read.csv("processed.cleveland.data",header=F)->data
#remove records with missing attributes
rowSums(data == "?") == 0 -> remove
data <- data[remove,]
input <- data[,-14]#extract the input data
output <- as.factor(data[,14])#extract the output data
#Set up train and test splits
#85% is training data
for(i in 1:10){
  createDataPartition(output, p=.85, list=FALSE) ->tag
  input[tag,] -> train_input
  output[tag] -> train_output
  #train the data with a naive bayes classifier
  #use 10-fold cross-validation
  #non-preProcessing
  model<-train(train_input, train_output, 'nb', trControl=trainControl(method='cv', number=10))
  model
  #predict the label of test data
  predict_result<-predict(model,newdata=input[-tag,])
  #calculate the accuracy of test data
  result <- confusionMatrix(data=predict_result, output[-tag])
  accuracy_test[i] <- result$overall["Accuracy"]
}
accuracy_test
```

```
##  [1] 0.6046512 0.5813953 0.5581395 0.5813953 0.5813953 0.6046512 0.5348837
##  [8] 0.6511628 0.6279070 0.5813953
```

```
mean(accuracy_test)
```

```
## [1] 0.5906977
```

```
sd(accuracy_test)
```

```
## [1] 0.03325211
```

Using the 10-fold cross-validation mechanisms, the mean of accuracy of the naive bayes classifier was estimated as 59.1%. The standard deviation of accuracy was 0.0333.