In [1]:
```
# installs the packages if necessary and then uses them
install.packages('klaR', repos='http://cran.us.r-project.org', dependenc
ies=TRUE)
install.packages('caret', repos='http://cran.us.r-project.org', dependen
cies=TRUE)
library('klaR')
library('caret')
```

```
Updating HTML index of packages in '.Library'
Making 'packages.html' ... done
Updating HTML index of packages in '.Library'
Making 'packages.html' ... done
Loading required package: MASS
Loading required package: lattice
Loading required package: ggplot2
```

In [2]:

```
print('3.1')
print('PART A:')

# reads in all the data
all_data<-read.csv('pima-indians-diabetes.data', header=FALSE)

feature_vector<-all_data[,-c(9)]
class_vector<-all_data[,9]
train_score<-array(dim=10)
test_score<-array(dim=10)
for (iteration in 1:10)
{
    # splits the data into a partition using 80% of the data
    subset_data<-createDataPartition(y=class_vector, p=.8, list=FALSE)

    # filters subset_data to get the 80% of the rows that were present i
n subset_data
    # the filter is provided by the feature_vectors variable
    subset_feature_vector<-feature_vector[subset_data, ]

    # this essentially gets the 9th column from the subset_data
    subset_class_vector<-class_vector[subset_data]

    # positive training set - I define it as the training set whose clas
sification is equal to 1
    # this is another filtering process
    pos_set_train<-subset_feature_vector[subset_class_vector > 0, ]

    # negative training set - pretty much the opposite as above
    # this is another filtering process, but does not derive from pos_se
t
    neg_set_train<-subset_feature_vector[!(subset_class_vector > 0),]

    # this is the log of the prior that we are going to add in (part of
 Naive Bayes)
    prob_add_log<-log(nrow(pos_set_train)/nrow(subset_feature_vector))
    prob_sub_log<-log(nrow(neg_set_train)/nrow(subset_feature_vector))

    # gets the data that was NOT included by the createDataPartition fun
ction
    filtered_data_feature<-feature_vector[-subset_data, ]

    # gets the classification data that was NOT included by the createDa
taPartition function
    filtered_data_class<-class_vector[-subset_data]





    # Now that the filtering of data is done, we can actually create the
 classifier
```

```
    # computes the mean with all NA values removed
    pos_set_train_mean<-sapply(pos_set_train, mean, na.rm=TRUE)
    neg_set_train_mean<-sapply(neg_set_train, mean, na.rm=TRUE)

    # computes the standard deviation with all the NA values removed
    pos_set_train_sd<-sapply(pos_set_train, sd, na.rm=TRUE)
    neg_set_train_sd<-sapply(neg_set_train, sd, na.rm=TRUE)


    # computes the "left term" in naive bayes formula
    temp_var<-t(t(subset_feature_vector)-pos_set_train_mean)
    left_term<-t(t(temp_var)/pos_set_train_sd)
    # generates the probability for the
    pos_set_train_logs<-(-(1/2)*rowSums(apply(left_term,c(1, 2), functio
n(x)x^2), na.rm=TRUE)-sum(log(pos_set_train_sd))) + prob_add_log

    # same steps as above, except for the negative training set
    temp_var<-t(t(subset_feature_vector)-neg_set_train_mean)
    left_term<-t(t(temp_var)/neg_set_train_sd)
    neg_set_train_logs<-(-(1/2)*rowSums(apply(left_term,c(1, 2), functio
n(x)x^2), na.rm=TRUE)-sum(log(neg_set_train_sd))) + prob_sub_log

    # compare the values and classify them in comparison to the values i
n the 9th column
    compared_vals<-pos_set_train_logs>neg_set_train_logs
    results_train<-compared_vals==subset_class_vector
    train_score[iteration]<-
sum(results_train)/(sum(results_train)+sum(!results_train))



    # repeat the few steps for the testing set
    temp_var<-t(t(filtered_data_feature)-pos_set_train_mean)
    left_term<-t(t(temp_var)/pos_set_train_sd)
    pos_set_test_logs<-(-(1/2)*rowSums(apply(left_term,c(1, 2),
function(x)x^2), na.rm=TRUE)-sum(log(pos_set_train_sd)))

    temp_var<-t(t(filtered_data_feature)-neg_set_train_mean)
    left_term<-t(t(temp_var)/neg_set_train_sd)
    neg_set_test_logs<-(-(1/2)*rowSums(apply(left_term,c(1, 2),
function(x)x^2), na.rm=TRUE)-sum(log(neg_set_train_sd)))

    compared_vals<-pos_set_test_logs>neg_set_test_logs
    results_test<-compared_vals==filtered_data_class
    test_score[iteration]<-sum(results_test)/(sum(results_test)+sum(!res
ults_test))
}

print('Training score:')
mean(train_score)

print('Test score:')
mean(test_score)
```

```
[1] "3.1"
[1] "PART A:"
[1] "Training score:"
```

0.759674796747967

```
[1] "Test score:"
```

0.773856209150327

In [3]:

```r
print('PART B:')

# reads in all the data
all_data<-read.csv('pima-indians-diabetes.data', header=FALSE)

feature_vector<-all_data[,-c(9)]

class_vector<-all_data[,9]
train_score<-array(dim=10)
test_score<-array(dim=10)

# these are the attributes that we will treat as 0
attributes<-c(3, 4, 6, 8)
for(i in attributes) {
    condition<-feature_vector[, i] == 0
    feature_vector[condition, i] = NA
}


for (iteration in 1:10)
{
    # splits the data into a partition using 80% of the data
    subset_data<-createDataPartition(y=class_vector, p=.8, list=FALSE)

    # filters subset_data to get the 80% of the rows that were present i
n subset_data
    # the filter is provided by the feature_vectors variable
    subset_feature_vector<-feature_vector[subset_data, ]

    # this essentially gets the 9th column from the subset_data
    subset_class_vector<-class_vector[subset_data]

    # positive training set - I define it as the training set whose clas
sification is equal to 1
    # this is another filtering process
    pos_set_train<-subset_feature_vector[subset_class_vector > 0, ]

    # negative training set - pretty much the opposite as above
    # this is another filtering process, but does not derive from pos_se
t
    neg_set_train<-subset_feature_vector[!(subset_class_vector > 0),]

    # this is the log of the prior that we are going to add in (part of
 Naive Bayes)
    prob_add_log<-log(nrow(pos_set_train)/nrow(subset_feature_vector))
    prob_sub_log<-log(nrow(neg_set_train)/nrow(subset_feature_vector))

    # gets the data that was NOT included by the createDataPartition fun
ction
    filtered_data_feature<-feature_vector[-subset_data, ]

    # gets the classification data that was NOT included by the createDa
taPartition function
    filtered_data_class<-class_vector[-subset_data]
```

```
    # Now that the filtering of data is done, we can actually create the
 classifier





    # computes the mean with all NA values removed
    pos_set_train_mean<-sapply(pos_set_train, mean, na.rm=TRUE)
    neg_set_train_mean<-sapply(neg_set_train, mean, na.rm=TRUE)

    # computes the standard deviation with all the NA values removed
    pos_set_train_sd<-sapply(pos_set_train, sd, na.rm=TRUE)
    neg_set_train_sd<-sapply(neg_set_train, sd, na.rm=TRUE)



    # computes the "left term" in naive bayes formula
    temp_var<-t(t(subset_feature_vector)-pos_set_train_mean)
    left_term<-t(t(temp_var)/pos_set_train_sd)
    # generates the probability for the
    pos_set_train_logs<-(-(1/2)*rowSums(apply(left_term,c(1, 2), functio
n(x)x^2), na.rm=TRUE)-sum(log(pos_set_train_sd))) + prob_add_log

    # same steps as above, except for the negative training set
    temp_var<-t(t(subset_feature_vector)-neg_set_train_mean)
    left_term<-t(t(temp_var)/neg_set_train_sd)
    neg_set_train_logs<-(-(1/2)*rowSums(apply(left_term,c(1, 2), functio
n(x)x^2), na.rm=TRUE)-sum(log(neg_set_train_sd))) + prob_sub_log

    # compare the values and classify them in comparison to the values i
n the 9th column
    compared_vals<-pos_set_train_logs>neg_set_train_logs
    results_train<-compared_vals==subset_class_vector
    train_score[iteration]<-
sum(results_train)/(sum(results_train)+sum(!results_train))



    # repeat the few steps for the testing set
    temp_var<-t(t(filtered_data_feature)-pos_set_train_mean)
    left_term<-t(t(temp_var)/pos_set_train_sd)
    pos_set_test_logs<-(-(1/2)*rowSums(apply(left_term,c(1, 2),
function(x)x^2), na.rm=TRUE)-sum(log(pos_set_train_sd)))

    temp_var<-t(t(filtered_data_feature)-neg_set_train_mean)
    left_term<-t(t(temp_var)/neg_set_train_sd)
    neg_set_test_logs<-(-(1/2)*rowSums(apply(left_term,c(1, 2),
function(x)x^2), na.rm=TRUE)-sum(log(neg_set_train_sd)))

    compared_vals<-pos_set_test_logs>neg_set_test_logs
    results_test<-compared_vals==filtered_data_class
    test_score[iteration]<-sum(results_test)/(sum(results_test)+sum(!res
ults_test))
```

```
}

print('Training score:')
mean(train_score)

print('Test score:')
mean(test_score)
```

```
[1] "PART B:"
[1] "Training score:"
```

0.749918699186992

```
[1] "Test score:"
```

0.737908496732026

In [4]:

```
print('PART C:')
options(warn=-1)

# reads in all the data
all_data<-read.csv('pima-indians-diabetes.data', header=FALSE)

feature_vector<-all_data[,-c(9)]
class_vector<-as.factor(all_data[,9])
subset_data<-createDataPartition(y=class_vector, p=.8, list=FALSE)

subset_feature_vector<-feature_vector[subset_data,]
subset_class_vector<-class_vector[subset_data]

model<-train(subset_feature_vector, subset_class_vector, 'nb',
trControl=trainControl(method='cv', number=10))
test_class<-predict(model,newdata=feature_vector[-subset_data,])

confusionMatrix(data=test_class, class_vector[-subset_data])
options(warn=0)
```

```
[1] "PART C:"

Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 84 17
         1 16 36

               Accuracy : 0.7843
                 95% CI : (0.7106, 0.8466)
    No Information Rate : 0.6536
    P-Value [Acc > NIR] : 0.0003018

                  Kappa : 0.5216
 Mcnemar's Test P-Value : 1.0000000

            Sensitivity : 0.8400
            Specificity : 0.6792
         Pos Pred Value : 0.8317
         Neg Pred Value : 0.6923
             Prevalence : 0.6536
         Detection Rate : 0.5490
   Detection Prevalence : 0.6601
      Balanced Accuracy : 0.7596

       'Positive' Class : 0
```

In [5]:

```
print('PART D:')

rm(list=ls())
all_data<-read.csv('pima-indians-diabetes.data', header=FALSE)
feature_vector<-all_data[,-c(9)]
class_vector<-as.factor(all_data[,9])
subset_data<-createDataPartition(y=class_vector, p=.8, list=FALSE)

svm<-svmlight(feature_vector[subset_data,], class_vector[subset_data], p
athsvm='~/Desktop/AML/hw1/')

labels<-predict(svm, feature_vector[-subset_data,])
foo<-labels$class
sum(foo==class_vector[-subset_data])/(sum(foo==class_vector[-
subset_data])+sum(!(foo==class_vector[-subset_data])))
```

```
[1] "PART D:"
```

0.718954248366013

In [6]:
```
print('3.3')
print('PART A:')

# suppresses annoying warnings
options(warn=-1)

# reads in all the data
all_data<-read.csv('processed-cleveland.data', header=FALSE)

# array to store our results
ret<-array(dim=10)

# marks the rows that we want to use
indexing_zero<-all_data[, 14] > 0
all_data[indexing_zero, 14] = 1

# rest is similar to previous questions
feature_vector<-all_data[,-c(14)]
class_vector<-as.factor(all_data[,14])

for(i in 1:10) {
    subset_data<-createDataPartition(y=class_vector, p=.85, list=FALSE)

    subset_feature_vector<-feature_vector[subset_data,]
    subset_class_vector<-class_vector[subset_data]

    model<-train(subset_feature_vector, subset_class_vector, 'nb', trCon
trol=trainControl(method='cv', number=10))
    test_class<-predict(model,newdata=feature_vector[-subset_data,])
    cm_matrix<-confusionMatrix(data=test_class, class_vector[-subset_dat
a])
    ret[i]<-cm_matrix$overall['Accuracy']
}
print("Mean:")
mean(ret)
print("Standard Deviation:")
sd(ret)
options(warn=0)
```

```
[1] "3.3"
[1] "PART A:"
[1] "Mean:"
```

0.829545454545455

```
[1] "Standard Deviation:"
```

0.0548915785840509

In [7]:

```
print('PART B:')
options(warn=-1)

# reads in all the data
all_data<-read.csv('processed-cleveland.data', header=FALSE)

# array to store our results
ret<-array(dim=10)

# rest is similar to above code...
feature_vector<-all_data[,-c(14)]
class_vector<-as.factor(all_data[,14])

for(i in 1:10) {
    subset_data<-createDataPartition(y=class_vector, p=.85, list=FALSE)

    subset_feature_vector<-feature_vector[subset_data,]
    subset_class_vector<-class_vector[subset_data]

    model<-train(subset_feature_vector, subset_class_vector, 'nb', trCon
trol=trainControl(method='cv', number=10))
    test_class<-predict(model,newdata=feature_vector[-subset_data,])

    cm_matrix<-confusionMatrix(data=test_class, class_vector[-subset_dat
a])
    ret[i]<-cm_matrix$overall['Accuracy']
}
print("Mean:")
mean(ret)
print("Standard Deviation:")
sd(ret)
options(warn=0)
```

```
[1] "PART B:"
[1] "Mean:"
```

0.586046511627907

```
[1] "Standard Deviation:"
```

0.0392220484982125

In [ ]: