Naive_Bayes_Amitesh_Shukla

##Own Implementation of Naive Bayes. Library Not used.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
#Read and load the dataset
pimdata<-read.csv('pima-indians-diabetes.data', header=FALSE)

ind_feat<-pimdata[,-c(9)] # Remove the target label. Conatins independent features
dep_label<-pimdata[,9] # Target label. Dependent variable for Naive Bayes

#Array to store the scores in each of the 10 runs
training_score<-array(dim=10)
test_score<-array(dim=10)
set.seed(26)
#Create different partitions 10 times and do the prediction
#Measure the score/accuracy in each run
for (wi in 1:10) {
 data_partition<-createDataPartition(y=dep_label, p=.8, list=FALSE)

 temp_data<-ind_feat
 #80% portion of the training features and corresponding predicted labels
 eighty_training<-temp_data[data_partition, ]
 eighty_label<-dep_label[data_partition]

 #Logic to group the training data in class - diabetic vs non diabetic
 is_diabetes <- eighty_label > 0
 positive_training_sample <- eighty_training[is_diabetes, ]
 negative_training_sample <- eighty_training[!is_diabetes,]

 #20% portion of the training features and corresponding predicted labels
 twenty_test <- temp_data[-data_partition, ]
 twenty_label <- dep_label[-data_partition]

 #To implement Naive Bayes we need to calculate the mean and standard deviation of the values in each c
#diabetes/non diabetes class.
 positive_training_mean <-
   sapply(positive_training_sample, mean, na.rm=TRUE)
 negative_training_mean <-
   sapply(negative_training_sample, mean, na.rm=TRUE)

 positive_training_sd <-
   sapply(positive_training_sample, sd, na.rm=TRUE)
 negative_training_sd <-
   sapply(negative_training_sample, sd, na.rm=TRUE)


 #Positive Training Sample
 #ptroffset is the distance from the mean of the postive group
```

```
ptroffsets <- t(t(eighty_training) - positive_training_mean)
ptrscales <- t(t(ptroffsets)/positive_training_sd)

#log probability of the postive prediction in 80% split
logpy <- sum(eighty_label>0)/length(eighty_label)

#Calculate the log probability of positive predictions
# Based on which group a future data is closer to, it will be predicted
ptrlogs <- -(1/2)*rowSums(apply(ptrscales,c(1, 2), function(x)x^2),
                          na.rm=TRUE) - sum(log(positive_training_sd))
                          + logpy

#Negative Training Sample
ntroffsets <- t(t(eighty_training)-negative_training_mean)
ntrscales <- t(t(ntroffsets)/negative_training_sd)


lognpy <- sum(eighty_label==0)/length(eighty_label)

#Calculate the log probability of negative predictions
#Based on which group a future data is closer to, it will be predicted
ntrlogs<- -(1/2)*rowSums(apply(ntrscales,c(1, 2), function(x)x^2),
                          na.rm=TRUE) - sum(log(negative_training_sd))
                          + lognpy

lvwtr <- ptrlogs > ntrlogs
correct_prediction <- lvwtr == eighty_label

training_score[wi] <- sum(correct_prediction)/
  (sum(correct_prediction)+sum(!correct_prediction))

#Test the above model with 20% test data
pteoffsets <- t(t(twenty_test)-positive_training_mean)
ptescales <- t(t(pteoffsets)/positive_training_sd)

logtpy <- sum(twenty_label>0)/length(twenty_label)

ptelogs<- -(1/2)*rowSums(apply(ptescales,c(1, 2), function(x)x^2),
                          na.rm=TRUE) - sum(log(positive_training_sd))
                          + logtpy

nteoffsets<-t(t(twenty_test)-negative_training_mean)
ntescales<-t(t(nteoffsets)/negative_training_sd)

logntpy <- sum(twenty_label==0)/length(twenty_label)

ntelogs<--(1/2)*rowSums(apply(ntescales,c(1, 2), function(x)x^2),
                          na.rm=TRUE) - sum(log(negative_training_sd))
                          + logntpy

lvwte <- ptelogs > ntelogs
gotrighttest <- lvwte == twenty_label
test_score[wi]<-sum(gotrighttest)/
```

```
   (sum(gotrighttest)+sum(!gotrighttest))

}
test_score
```

```
##  [1] 0.7581699 0.7320261 0.6993464 0.7712418 0.7843137 0.7908497 0.7516340
##  [8] 0.7450980 0.8104575 0.8169935
```

```
average_accuracy <- sum(test_score)/10
average_accuracy
```

```
## [1] 0.7660131
```

Adjust so that 0 is treated as a missing value

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

ind_feat<-pimdata[,-c(9)]
dep_label<-pimdata[,9]

set.seed(260)
training_score<-array(dim=10)
test_score<-array(dim=10)
for (wi in 1:10) {
 data_partition<-createDataPartition(y=dep_label, p=.8, list=FALSE)

 temp_data<-ind_feat
 eighty_training<-temp_data[data_partition, ]
  # https://stackoverflow.com/questions/13871614/replacing-values-from-a-column-using-a-condition-in-r
 # Replacing 0 with NA
 eighty_training$V3[eighty_training$V3 == 0] <- NA
 eighty_training$V4[eighty_training$V4 == 0] <- NA
 eighty_training$V6[eighty_training$V6 == 0] <- NA
 eighty_training$V8[eighty_training$V8 == 0] <- NA

 eighty_label<-dep_label[data_partition]

 is_diabetes <- eighty_label > 0
 positive_training_sample <- eighty_training[is_diabetes, ]
 negative_training_sample <- eighty_training[!is_diabetes,]

 twenty_test <- temp_data[-data_partition, ]
 # twenty_test$V3[twenty_test$V3 == 0] <- NA
 # twenty_test$V4[twenty_test$V4 == 0] <- NA
 # twenty_test$V6[twenty_test$V6 == 0] <- NA
 # twenty_test$V8[twenty_test$V8 == 0] <- NA


 twenty_label <- dep_label[-data_partition]

 positive_training_mean <-
   sapply(positive_training_sample, mean, na.rm=TRUE)
 negative_training_mean <-
```

```
  sapply(negative_training_sample, mean, na.rm=TRUE)

positive_training_sd <-
  sapply(positive_training_sample, sd, na.rm=TRUE)
negative_training_sd <-
  sapply(negative_training_sample, sd, na.rm=TRUE)

#why is transpose needed??
#Positive Training Sample
ptroffsets <- t(t(eighty_training) - positive_training_mean)
ptrscales <- t(t(ptroffsets)/positive_training_sd)

logpy <- sum(eighty_label>0)/length(eighty_label)

ptrlogs <- -(1/2)*rowSums(apply(ptrscales,c(1, 2), function(x)x^2),
                          na.rm=TRUE) - sum(log(positive_training_sd)) + logpy

#Negative Training Sample
ntroffsets <- t(t(eighty_training)-negative_training_mean)
ntrscales <- t(t(ntroffsets)/negative_training_sd)

lognpy <- sum(eighty_label==0)/length(eighty_label)

ntrlogs<- -(1/2)*rowSums(apply(ntrscales,c(1, 2), function(x)x^2),
                          na.rm=TRUE) - sum(log(negative_training_sd)) + lognpy

lvwtr <- ptrlogs > ntrlogs
correct_prediction <- lvwtr == eighty_label

training_score[wi] <- sum(correct_prediction)/
  (sum(correct_prediction)+sum(!correct_prediction))

#Test Sample
pteoffsets <- t(t(twenty_test)-positive_training_mean)
ptescales <- t(t(pteoffsets)/positive_training_sd)

logtpy <- sum(twenty_label>0)/length(twenty_label)

ptelogs<- -(1/2)*rowSums(apply(ptescales,c(1, 2), function(x)x^2),
                          na.rm=TRUE) - sum(log(positive_training_sd)) + logtpy

nteoffsets<-t(t(twenty_test)-negative_training_mean)
ntescales<-t(t(nteoffsets)/negative_training_sd)

logntpy <- sum(twenty_label==0)/length(twenty_label)

ntelogs<--(1/2)*rowSums(apply(ntescales,c(1, 2), function(x)x^2),
                          na.rm=TRUE) - sum(log(negative_training_sd)) + logntpy

lvwte <- ptelogs > ntelogs
gotrighttest <- lvwte == twenty_label
test_score[wi]<-sum(gotrighttest)/
  (sum(gotrighttest)+sum(!gotrighttest))
```

```
}
test_score
```

```
##  [1] 0.6993464 0.7908497 0.7254902 0.7581699 0.8039216 0.7450980 0.8104575
##  [8] 0.7254902 0.7189542 0.7450980
```

```
average_accuracy <- sum(test_score)/10
average_accuracy
```

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

Use naive bayes library

```
pimdata<-read.csv('pima-indians-diabetes.data', header=FALSE)
library(klaR, quietly = TRUE)
library(caret, quietly = TRUE)

set.seed(26)
independent_features<-pimdata[,-c(9)]
target_labels<-as.factor(pimdata[,9])
partition_index<-createDataPartition(y=target_labels, p=.8, list=FALSE)
trax<-independent_features[partition_index,]
tray<-target_labels[partition_index]
#Build a Naive Bayes model with 10 fold cross validation
model<-train(trax, tray, 'naive_bayes', trControl=trainControl(method='cv', number=10))
testClass<-predict(model,newdata=independent_features[-partition_index,])
confusionMatrix(data=testClass, target_labels[-partition_index])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 90 22
##          1 10 31
##
##                Accuracy : 0.7908
##                  95% CI : (0.7178, 0.8523)
##     No Information Rate : 0.6536
##     P-Value [Acc > NIR] : 0.0001499
##
##                   Kappa : 0.5122
##
##  Mcnemar's Test P-Value : 0.0518299
##
##             Sensitivity : 0.9000
##             Specificity : 0.5849
##          Pos Pred Value : 0.8036
##          Neg Pred Value : 0.7561
##              Prevalence : 0.6536
##          Detection Rate : 0.5882
##    Detection Prevalence : 0.7320
##       Balanced Accuracy : 0.7425
##
```

```
##        'Positive' Class : 0
##
```

Use svmlight to compare the results

```
library(klaR)
library(caret)
rm(list=ls())
set.seed(120)
pimdata<-read.csv('pima-indians-diabetes.data', header=FALSE)
ind_feat<-pimdata[,-c(9)]
dep_label<-as.factor(pimdata[,9])

partition<-createDataPartition(y=dep_label, p=.8, list=FALSE)
svm<-svmlight(ind_feat[partition,], dep_label[partition], pathsvm='svm_light_osx.8.4_i7')
labels<-predict(svm, ind_feat[-partition,])
data_label<-labels$class
accuracy <- sum(data_label==dep_label[-partition])/(sum(data_label==dep_label[-partition])+sum(!(data_la
accuracy
```

```
## [1] 0.7908497
```

MNIST data with Naive Bayes

##Untouched image - Gaussian

```
load_image_file = function(filename) {
  ret = list()
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n    = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  nrow = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  ncol = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  x = readBin(f, 'integer', n = n * nrow * ncol, size = 1, signed = FALSE)
  close(f)
  data.frame(matrix(x, ncol = nrow * ncol, byrow = TRUE))
}

# load label files
load_label_file = function(filename) {
  f = file(filename, 'rb')
  readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  n = readBin(f, 'integer', n = 1, size = 4, endian = 'big')
  y = readBin(f, 'integer', n = n, size = 1, signed = FALSE)
  close(f)
  y
}

# load images
train = load_image_file("train-images-idx3-ubyte")
test  = load_image_file("t10k-images-idx3-ubyte")

# load labels
```

```
train$y = as.factor(load_label_file("train-labels-idx1-ubyte"))
test$y  = as.factor(load_label_file("t10k-labels-idx1-ubyte"))

# create pixel header
pixel_header = function(x)
{
  out = array()
  for (ix in 1:x)
  {
    out[ix] = sprintf("pixel%i", ix-1)
  }
  out[ix+1] = "class"
  return (out)
}

ph = pixel_header(784) #adds "class" at [785]
names(train) = ph      #sets header of training set data frame
names(test) = ph       #sets header of test set data frame

write.csv(train, file="mnist_train.csv", row.names=FALSE)
write.csv(test, file="mnist_test.csv", row.names=FALSE)
```

```
train = read.csv("mnist_train.csv", header = TRUE)
test = read.csv("mnist_test.csv", header = TRUE)
```

```
library(naivebayes)
```

```
## naivebayes 0.9.6 loaded
```

```
data<-train[,-c(785)]
data_label<-as.factor(train[,785])
test_data_label<-as.factor(test[,785])

set.seed(260)
model<- naive_bayes(x=data,y=data_label)

predictions<-predict(model,newdata=test)
```

```
## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.
```

```
confusionMatrix(data=predictions, test_data_label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0  862    0   94   40   22   69   16    2   13    8
##          1    0 1081   24   33    2   25   13   14   67    7
##          2    1    1  212    4    2    1    2    0    4    1
##          3    3    0   87  310    0   19    0    7    6    5
##          4    3    0    2    1  131    2    1    8    2    0
##          5    4    0    2    4    4   35    4    2    7    0
##          6   31   10  284   59   66   39  890    5   13    2
##          7    1    0    5    7    7    1    0  228    3    3
```

```
##          8   44   36  298  437  188  592   27   49  642   22
##          9   31    7   24  115  560  109    5  713  217  961
##
## Overall Statistics
##
##                Accuracy : 0.5352
##                  95% CI : (0.5254, 0.545)
##     No Information Rate : 0.1135
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4832
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.8796   0.9524   0.2054   0.3069   0.1334  0.03924
## Specificity            0.9707   0.9791   0.9982   0.9859   0.9979  0.99704
## Pos Pred Value         0.7655   0.8539   0.9298   0.7094   0.8733  0.56452
## Neg Pred Value         0.9867   0.9938   0.9161   0.9268   0.9136  0.91377
## Prevalence             0.0980   0.1135   0.1032   0.1010   0.0982  0.08920
## Detection Rate         0.0862   0.1081   0.0212   0.0310   0.0131  0.00350
## Detection Prevalence   0.1126   0.1266   0.0228   0.0437   0.0150  0.00620
## Balanced Accuracy      0.9252   0.9658   0.6018   0.6464   0.5656  0.51814
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9290   0.2218   0.6591   0.9524
## Specificity            0.9437   0.9970   0.8124   0.8019
## Pos Pred Value         0.6362   0.8941   0.2749   0.3505
## Neg Pred Value         0.9921   0.9179   0.9567   0.9934
## Prevalence             0.0958   0.1028   0.0974   0.1009
## Detection Rate         0.0890   0.0228   0.0642   0.0961
## Detection Prevalence   0.1399   0.0255   0.2335   0.2742
## Balanced Accuracy      0.9364   0.6094   0.7358   0.8772
```

#Bounded/stretched image - Gaussian

```
library(imager, quietly)
```

```
## Loading required package: magrittr
##
## Attaching package: 'imager'
## The following object is masked from 'package:magrittr':
##
##     add
## The following objects are masked from 'package:stats':
##
##     convolve, spectrum
## The following object is masked from 'package:graphics':
##
##     frame
## The following object is masked from 'package:base':
##
```

```
##     save.image
```

```
train.read = file("train-images-idx3-ubyte", "rb")

ntrain <- matrix(data=NA,nrow=60000,ncol=400)

readBin(train.read, integer(), n=4, endian="big")
```

```
## [1]  2051 60000    28    28
```

```
for(i in 1:60000) {
  m = matrix(readBin(train.read,integer(), size=1, n=28*28,     endian="big",signed=FALSE),28,28)
  p<-resize(autocrop(as.cimg(m)), size_x = 20, size_y = 20)[, , 1, 1]
  ntrain[i,] = as.vector(p)
}
ntrain = as.data.frame(ntrain)

test.read = file("t10k-images-idx3-ubyte", "rb")

ntest <- matrix(data=0,nrow=10000,ncol=400)

readBin(test.read, integer(), n=4, endian="big")
```

```
## [1]  2051 10000    28    28
```

```
for(i in 1:10000) {
  m = matrix(readBin(test.read,integer(), size=1, n=28*28,     endian="big",signed=FALSE),28,28)
  p<-resize(autocrop(as.cimg(m)), size_x = 20, size_y = 20)[, , 1, 1]
  ntest[i,] = as.vector(p)
}
ntest = as.data.frame(ntest)
```

```
# load labels
ntrain$y = as.factor(load_label_file("train-labels-idx1-ubyte"))
ntest$y  = as.factor(load_label_file("t10k-labels-idx1-ubyte"))

# create pixel header
pixel_header = function(x)
{
  out = array()
  for (ix in 1:x)
  {
    out[ix] = sprintf("pixel%i", ix-1)
  }
  out[ix+1] = "class"
  return (out)
}

ph = pixel_header(400) #adds "class" at [785]
names(ntrain) = ph      #sets header of training set data frame
names(ntest) = ph       #sets header of test set data frame

write.csv(ntrain, file="mnist_train_stretch.csv", row.names=FALSE)
write.csv(ntest, file="mnist_test_stretch.csv", row.names=FALSE)
```

```
newtrain = read.csv("mnist_train_stretch.csv", header = TRUE)
```

```r
newtest = read.csv("mnist_test_stretch.csv", header = TRUE)
```

```r
library(naivebayes)

data<-newtrain[,-c(401)]
data_label<-as.factor(newtrain[,401])
test_data_label<-as.factor(newtest[,401])

set.seed(260)
model<- naive_bayes(x=data,y=data_label)

predictions<-predict(model,newdata=newtest)
```

```
## Warning: predict.naive_bayes(): More features in the newdata are provided
## as there are probability tables in the object. Calculation is performed
## based on features to be found in the tables.
```

```r
confusionMatrix(data=predictions, test_data_label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 931   1  16   9   1  19  14   1  10   4
##          1   0 919  16   9  27  15  11  53  42  15
##          2   3  49 834  25   5   3   7  34  29   7
##          3   3   2  10 860   0  67   1  12  12   9
##          4   6  30   7   3 809  10  45  17  14  47
##          5   7  16  15  29  11 693  18   7  30   7
##          6  12  21   9   0  24  25 845   0   9   0
##          7   0  59  55  20   2   6   0 828  13  18
##          8  15  28  56  38  16  20  14  18 688  47
##          9   3  10  14  17  87  34   3  58 127 855
##
## Overall Statistics
##
##                Accuracy : 0.8262
##                  95% CI : (0.8186, 0.8336)
##     No Information Rate : 0.1135
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8068
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9500   0.8097   0.8081   0.8515   0.8238   0.7769
## Specificity            0.9917   0.9788   0.9819   0.9871   0.9802   0.9846
## Pos Pred Value         0.9254   0.8302   0.8373   0.8811   0.8188   0.8319
## Neg Pred Value         0.9946   0.9757   0.9780   0.9834   0.9808   0.9783
## Prevalence             0.0980   0.1135   0.1032   0.1010   0.0982   0.0892
## Detection Rate         0.0931   0.0919   0.0834   0.0860   0.0809   0.0693
## Detection Prevalence   0.1006   0.1107   0.0996   0.0976   0.0988   0.0833
```

```
## Balanced Accuracy      0.9708   0.8942   0.8950   0.9193   0.9020   0.8808
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.8820   0.8054   0.7064   0.8474
## Specificity            0.9889   0.9807   0.9721   0.9607
## Pos Pred Value         0.8942   0.8272   0.7319   0.7078
## Neg Pred Value         0.9875   0.9778   0.9684   0.9825
## Prevalence             0.0958   0.1028   0.0974   0.1009
## Detection Rate         0.0845   0.0828   0.0688   0.0855
## Detection Prevalence   0.0945   0.1001   0.0940   0.1208
## Balanced Accuracy      0.9355   0.8931   0.8392   0.9041
```

##Untouched image - Bernoulli

```
library(naivebayes)

set.seed(26)
data<-train[,-c(785)]
data[data<=127] <- 0
data[data>127] <- 1

data=as.data.frame(data)
level <- c(0,1)
data[]<-data.frame(lapply(data, factor, levels=level))
data_label<-as.factor(train[,785])

dtest<-test[,-c(785)]
dtest[dtest<=127] <- 0
dtest[dtest>127] <- 1


dtest=as.data.frame(dtest)
dtest[]<-data.frame(lapply(dtest, factor, levels=level))
test_data_label<-as.factor(test[,785])

model<- naive_bayes(x=data,y=data_label,laplace=1)
predictions<-predict(model,newdata=dtest)
confusionMatrix(data=predictions, test_data_label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1    2    3    4    5    6    7    8    9
##          0  882    0   17    4    2   16   17    2   10   12
##          1    0 1086   14   17    7   11   14   33   28   13
##          2    3    6  840   36    4    5   15   17   12    6
##          3    4    5   32  839    0  106    1    3   68    8
##          4    1    0   21    1  799   24   14   17   16   64
##          5   51    9    5   30    2  655   37    0   27    9
##          6   21    4   28    7   16   18  853    0   10    0
##          7    1    0   16   13    2    8    0  864    6   29
##          8   17   25   57   44   12   24    7   25  762   21
##          9    0    0    2   19  138   25    0   67   35  847
##
## Overall Statistics
##
```

```
##                Accuracy : 0.8427
##                  95% CI : (0.8354, 0.8498)
##     No Information Rate : 0.1135
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8251
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9000   0.9568   0.8140   0.8307   0.8136   0.7343
## Specificity            0.9911   0.9845   0.9884   0.9747   0.9825   0.9813
## Pos Pred Value         0.9168   0.8880   0.8898   0.7871   0.8349   0.7939
## Neg Pred Value         0.9892   0.9944   0.9788   0.9809   0.9798   0.9742
## Prevalence             0.0980   0.1135   0.1032   0.1010   0.0982   0.0892
## Detection Rate         0.0882   0.1086   0.0840   0.0839   0.0799   0.0655
## Detection Prevalence   0.0962   0.1223   0.0944   0.1066   0.0957   0.0825
## Balanced Accuracy      0.9456   0.9707   0.9012   0.9027   0.8981   0.8578
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.8904   0.8405   0.7823   0.8394
## Specificity            0.9885   0.9916   0.9743   0.9682
## Pos Pred Value         0.8913   0.9201   0.7666   0.7476
## Neg Pred Value         0.9884   0.9819   0.9765   0.9817
## Prevalence             0.0958   0.1028   0.0974   0.1009
## Detection Rate         0.0853   0.0864   0.0762   0.0847
## Detection Prevalence   0.0957   0.0939   0.0994   0.1133
## Balanced Accuracy      0.9394   0.9161   0.8783   0.9038
```

##Bounded/Stretched - Bernoulli

```
library(naivebayes)

set.seed(26)
data<-newtrain[,-c(401)]

data[data<=127] <- 0
data[data>127] <- 1

data=as.data.frame(data)
level <- c(0,1)
data[]<-data.frame(lapply(data, factor, levels=level))
data_label<-as.factor(newtrain[,401])

dtest<-newtest[,-c(401)]
dtest[dtest<=127] <- 0
dtest[dtest>127] <- 1
dtest=as.data.frame(dtest)
level <- c(0,1)
dtest[]<-data.frame(lapply(dtest, factor, levels=level))
test_data_label<-as.factor(newtest[,401])

model<- naive_bayes(x=data,y=data_label, laplace=1)
predictions<-predict(model,newdata=dtest)
```

```
confusionMatrix(data=predictions, test_data_label)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0 938   1  29   2   2  10  12   1   8   3
##          1   5 873  32  10  39  24  35  91  51  31
##          2   5  33 816  16   6   3   7  14  12   4
##          3   5  11  21 911   0 117   4   9  44   8
##          4   2  39   4   2 839   7  18  22  23  59
##          5  10  23  15  15   6 684  27   6  37   8
##          6   9  17   9   2  17  14 851   0   8   0
##          7   1  33  45  19   0   3   0 818  21  32
##          8   5 103  57  25   8   9   2  20 684  50
##          9   0   2   4   8  65  21   2  47  86 814
##
## Overall Statistics
##
##                Accuracy : 0.8228
##                  95% CI : (0.8152, 0.8302)
##     No Information Rate : 0.1135
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.803
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9571   0.7692   0.7907   0.9020   0.8544   0.7668
## Specificity            0.9925   0.9641   0.9888   0.9756   0.9805   0.9839
## Pos Pred Value         0.9324   0.7330   0.8908   0.8062   0.8266   0.8231
## Neg Pred Value         0.9953   0.9703   0.9762   0.9888   0.9841   0.9773
## Prevalence             0.0980   0.1135   0.1032   0.1010   0.0982   0.0892
## Detection Rate         0.0938   0.0873   0.0816   0.0911   0.0839   0.0684
## Detection Prevalence   0.1006   0.1191   0.0916   0.1130   0.1015   0.0831
## Balanced Accuracy      0.9748   0.8666   0.8898   0.9388   0.9174   0.8753
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.8883   0.7957   0.7023   0.8067
## Specificity            0.9916   0.9828   0.9691   0.9739
## Pos Pred Value         0.9180   0.8416   0.7103   0.7760
## Neg Pred Value         0.9882   0.9767   0.9679   0.9782
## Prevalence             0.0958   0.1028   0.0974   0.1009
## Detection Rate         0.0851   0.0818   0.0684   0.0814
## Detection Prevalence   0.0927   0.0972   0.0963   0.1049
## Balanced Accuracy      0.9400   0.8893   0.8357   0.8903
```