```
# -------------------------------------------------------------------
# Purpose: Practical for Naive Bayes Predictive Models in R
# Author : Liew How Hui (2022)
# Reference & Data:
#  1. https://www.statlearning.com/resources-second-edition
#  2. http://www.dbenson.co.uk/Rparts/subpages/spamR/
#  3. http://www.learnbymarketing.com/tutorials/naive-bayes-in-r/
# License: BSD-3
# Software: R 4.x & R 3.6
# Duration: 1 hour
# Remark: Make sure you do the programming instead of running
#         the R script only.  Programming questions do come out in
#         the final exam.
# -------------------------------------------------------------------

# -------------------------------------------------------------------
# Performance Measurements for Binary Classification Problem
# You can replace the following with caret::confusionMatrix
# -------------------------------------------------------------------

performance = function(xtab, desc=""){
    cat(desc,"\n")
    ACR = sum(diag(xtab))/sum(xtab)
    TPR = xtab[1,1]/sum(xtab[,1]); TNR = xtab[2,2]/sum(xtab[,2])
    PPV = xtab[1,1]/sum(xtab[1,]); NPV = xtab[2,2]/sum(xtab[2,])
    FPR = 1 - TNR                ; FNR = 1 - TPR
    # https://standardwisdom.com/softwarejournal/2011/12/confusion-matrix-another-single-va
lue-metric-kappa-statistic/
    RandomAccuracy = (sum(xtab[,2])*sum(xtab[2,]) +
      sum(xtab[,1])*sum(xtab[1,]))/(sum(xtab)^2)
    Kappa = (ACR - RandomAccuracy)/(1 - RandomAccuracy)
    print(xtab)
    cat("\n      Accuracy :", ACR, "\n\n         Kappa :", Kappa, "\n")
    cat("\n   Sensitivity :", TPR,   "\n   Specificity :", TNR, "\n")
    cat("Pos Pred Value :", PPV,     "\nNeg Pred Value :", NPV, "\n")
    cat("           FPR :", FPR,     "\n           FNR :", FNR, "\n")
}

# -------------------------------------------------------------------
#  Analysis of the ‘Fraud’ Dataset with
#  the mixed Categorical & Gaussian Naive Bayes Model
# -------------------------------------------------------------------

#https://liaohaohui.github.io/MEME19903/fraud.csv
fraud = read.csv("fraud.csv")  # categorical data are encoded as integers
# change data type from integer to categorical (mentioned in Practical 3)
col_fac = c("gender", "status", "employment", "account_link", "supplement", "tag")
fraud[col_fac] = lapply(fraud[col_fac], factor)
fraud$id_person = NULL   # Removing the id_person column

# -------------------------------------------------------------------
#  Stratified Sampling Holdout Method
# -------------------------------------------------------------------

### Option 1: Manual Stratified Sampling
set.seed(123)
fraud_tag0 = fraud[fraud$tag=="0", ]
fraud_tag1 = fraud[fraud$tag=="1", ]
tag0_idx = sample(1:nrow(fraud_tag0), size=0.7*nrow(fraud_tag0))
tag1_idx = sample(1:nrow(fraud_tag1), size=0.7*nrow(fraud_tag1))
fraud.train = rbind(fraud_tag0[tag0_idx,],fraud_tag1[tag1_idx,])
fraud.test = rbind(fraud_tag0[-tag0_idx,],fraud_tag1[-tag1_idx,])

#
# Choices for Naive Bayes:
# (1) naivebayes library (used by the main reference book)
# (2) e1071 library
# (3) klaR library?
# (4) etc.
#

library(naivebayes)
#
# Naive Bayes without Laplace Smoothing
#
model.nb = naive_bayes(tag˜., data = fraud.train)
#library(e1071)  # naiveBayes
#model.e1071 = naiveBayes(tag˜., data=fraud.train, laplace=0)
p = ncol(fraud.train)-1
pred.nb = predict(model.nb, newdata = fraud.test[,1:p])  # columns 1:p for inputs
cfmat = table(pred.nb, actual.fraud=fraud.test$tag)
performance(cfmat, "Performance of Naive Bayes without Laplace Smoothing")

#
# Naive Bayes WITH Laplace Smoothing
#
model.nb.lp = naive_bayes(tag˜., data=fraud.train, laplace=1)
pred.nb.lp = predict(model.nb.lp, fraud.test[,1:p])
cfmat = table(pred.nb.lp, actual.fraud=fraud.test$tag)
performance(cfmat, "Performance of Naive Bayes with Laplace Smoothing")


# ------------------------------------------------------------------
#  Analysis of the performance of a Simulated Data using
#  the mixed Categorical & Gaussian Naive Bayes Model
# ------------------------------------------------------------------

no_resp = 500
resp = 100
set.seed(1)
response = factor(c(rep(0,no_resp),rep(1,resp)))
purchased_previously = factor(c(sample(0:1,no_resp,prob=c(0.6,0.4),replace=T),
                           sample(0:1,resp,prob=c(0.2,0.8),replace=T)))
opened_previously = factor(sample(0:1,(no_resp+resp),prob=c(0.8,0.2),replace=T))
sales_12mo = c(rnorm(n=no_resp,mean = 50, sd = 10),
               rnorm(n=resp,mean = 60, sd = 5))
none_open_buy = factor(c(sample(0:1, no_resp,prob=c(0.8,0.2),replace=T),
                           rep(1,resp)))
test_var = sample(LETTERS[1:2],(resp+no_resp),replace=T)

naive_data = data.frame(purchased_previously = purchased_previously,
                        opened_previously = opened_previously,
                        sales_12mo = sales_12mo,
                        none_open_buy = none_open_buy,
                        test_var = test_var,
                        response = response)

#
# Linear Sampling
#
# Shuffle all the rows
naive_data = naive_data[sample(1:nrow(naive_data),nrow(naive_data)),]
# Take first 70% for training and the remainder for testing
train = naive_data[1:(nrow(naive_data)*.7),]
test  = naive_data[(nrow(naive_data)*.7+1):nrow(naive_data),]

# Without Laplace Smoothing
#nb_default = naiveBayes(response˜., data=train[,-4], laplace=0)
nb_default = naive_bayes(response˜., data=train[,-4])  # laplace defaults to 0
default_pred = predict(nb_default, test, type="class")
# To extract information from Naive Bayes Network Model
#default_raw_pred <- predict(nb_default, test, type="raw")
table(default_pred, test$response,dnn=c("Prediction","Actual"))

# With Laplace Smoothing
#nb_laplace1 = naiveBayes(response˜., data=train, laplace=1)
```

```
nb_laplace1 = naive_bayes(response˜., data=train, laplace=1)
laplace1_pred = predict(nb_laplace1, test, type="class")
table(laplace1_pred, test$response,dnn=c("Prediction","Actual"))


# ---------------------------------------------------------------------
#  Spam Filtering with
#  Multinomial Naive Bayes Model and
#  Bernoulli Naive Bayes Model
# ---------------------------------------------------------------------

d.f = read.csv(text=’
ham,1,"Hi sir, just want to ask you if the formula xxx is OK?"
spam,1,"Maxis great deal is here"
ham,2,"If I solve the problem the following way ... is it OK?"
ham,3,"You solution is correct.  Great job"
spam,2,"Discount 20% from Maxis when dinning at ..."
ham,4,"The maximum value for ... is the coefficient for the model ..."
spam,3,"Win a phone when subscribing to Maxis new plan ..."
spam,4,"Upgrade to Digi new plan ..."
spam,5,"Subscribe to ASTRO  ... with only RM250 per month"
ham,5,"Why can\’t I get the right result?"
’,header=F,col.names=c("Y","id", "content"))

library(tm)  # Text Mining package
corpus = VCorpus(VectorSource(d.f$content))
# The DocumentTermMatrix can be slow for large data
# and the stemming is too primitive and brutal!
dtm = DocumentTermMatrix(corpus, control = list(
  tolower = TRUE,
  removeNumbers = TRUE,
  removePunctuation = TRUE,
  stemming = TRUE      # This is bad, need to work on it
))   # Statistical model
### The features are encoded in
# dtm$dimnames$...

#
# For ‘text’ classification with Naive Bayes, we may want to
# turn on the Laplace smoothing!
#

library(naivebayes)

### naivebayes::multinomial_naive_bayes

idx.train = 1:6
train = as.matrix(dtm[idx.train,])
Y.train = d.f$Y[idx.train]
idx.test = 7:10
test  = as.matrix(dtm[idx.test,])
Y.test  = d.f$Y[idx.test]

classifier = multinomial_naive_bayes(train, Y.train, laplace=1)
summary(classifier)
coef(classifier)
#
# Let’s check the word ’you’:
# p = length(dtm$dimnames$Terms) = 45
# P(word=’you’|Y=’ham’) = (2+1)/(28+45)
# number of times the word ’you’ occured in training data of class ’ham’ = 2
# number of words in training data of class ’ham’ = 28
# P(word=’you’|Y=’spam’) = (0+1)/(9+45)
# number of times the word ’you’ occured in training data of class ’ham’ = 2
# number of times the word ’you’ occured in training data of class ’spam’ = 0
# number of words in training data of class ’spam’ = 9
#
yhat = predict(classifier, test)
```

```r
cfmat = table(yhat, Y.test)
print(cfmat)

### https://www.kaggle.com/code/abeperez/building-a-spam-filter-using-fastnaivebayes/notebo
ok
library(fastNaiveBayes)
mnnb = fnb.multinomial(x=train, y=Y.train, laplace=1)
# The fastNaiveBayes provides a nice summary of word counts with
# the list item ’present’:
mnnb$present
yhat = predict(mnnb, test)
cfmat = table(yhat, Y.test)
print(cfmat)

### naivebayes::bernoulli_naive_bayes
convert2bin = function(x){ifelse(x>0,1,0)}
library(Matrix)
train = Matrix(apply(dtm[idx.train,],2,convert2bin),sparse=T)
Y.train = d.f$Y[idx.train]
test = Matrix(apply(dtm[idx.test,],2,convert2bin),sparse=T)
Y.test  = d.f$Y[idx.test]

classifier = bernoulli_naive_bayes(train, Y.train, laplace=1)
yhat = predict(classifier, test)
cfmat = table(yhat, Y.test)
print(cfmat)

#
# Binary Categorical NB == Bernoulli NB ???
#
# naive_bayes from ‘naivebayes’ package has issue with the *PREDICTION*
#classifier = naive_bayes(train, trainLabels, laplace=1)

convert = function(x){ifelse(x>0,"Yes","No")}
train = as.data.frame(apply(dtm[idx.train,],2,convert))
train = as.data.frame(lapply(train, function(c){factor(c,levels=c("No","Yes"))}))
Y.train = factor(d.f$Y[idx.train],levels=c("ham","spam"))
test = as.data.frame(apply(dtm[idx.test,],2,convert))
test = as.data.frame(lapply(test, function(c){factor(c,levels=c("No","Yes"))}))
Y.test  = factor(d.f$Y[idx.test],levels=c("ham","spam"))

library(e1071)
classifier = naiveBayes(train, Y.train, laplace=1)
#classifier$tables$call   # Probability table of seeing the world ‘call’
yhat = predict(classifier, test)

cfmat = table(yhat, Y.test)
performance(cfmat, "e1071 Naive Bayes with Laplace Smoothing")
```