Universidade Estadual de Campinas - Instituto de Computação (IC/Unicamp)
Disciplina Aprendizado de Máquina (MO444)
Professor: Jacques Wainer, PhD. **Aluno: Luiz Alberto Ferreira Gomes**
Atividade 3, 23 de outubro de 2016 **RA:007275**

Questão 1:

Determine a acurácia dos algoritmos kNN, SVM com RBF, redes neurais (Nnet), Random Forest (RF) e Gradient Boosting Machine (GBM).

Solução:

A tabela abaixo reporta as acurácias calculadas pelo script em R(ver anexo I) por meio do um 5-fold externo. De acordo com essa tabela, o algoritmos RF apresentou a acurácia ligeiramente melhor do que os outros durante os testes.

Classificador	**Acurária**
kNN	0.93589
SVM	0.93589
Nnet	0.92307
RF	0.93590
GBM	0.93269

Anexo I: Script fonte em R

```
# --------------------------------------------------------------------------
# Description:
#   solutions for activity 3 (MO444)
#
# Version: 1.0
#
# Author:
#   Luiz Alberto, gomes.luiz@gmail.com
#
# History:
#   Sep 15th,   2016   started
#
# To do:
#  -
# --------------------------------------------------------------------------

 if (!require(caret)) install.packages('caret')
 if (!require(gbm)) install.packages('gbm')
 if (!require(kernlab)) install.packages('kernlab')
 if (!require(nnet)) install.packages('nnet')
 if (!require(randomForest)) install.packages('randomForest')

 library(caret)
 library(gbm)
 library(kernlab)
 library(nnet)
 library(randomForest)


# cleans up execution environment.
rm(list=ls())

# sets up path to data files.
setwd('~/Workspace/doutorado/disciplinas/mo444b/atividades/3')

# ----------------------------------------------------------------------------------
```

```r
# common functions
# ------------------------------------------------------------------------------
ReadDataFile <- function(name){
  # Reads a data file in csv format.
  #
  # Args:
  #   name: file name to be read.
  #
  # Returns:
  #   the data frame with rows and columns from file.
  #
  result <- read.csv(file = paste('./data/',name,sep=''), header = FALSE, sep = ' ')
  return(result)
}

InputByMean <- function(data){
  # Performs the input missing data by mean.
  #
  # Args:
  #   data: data frame to be processed.
  #
  # Returns:
  #   the data frame processed.
  #
  for(i in 1:ncol(data)){
    data[is.na(data[,i]), i] <- mean(data[,i], na.rm = TRUE)
  }
  return(data)
}

NormalizeData <- function(data){
  # Standardizes the columns for average 0 and standard deviation 1
  #
  # Args:
  #   data: data frame to be normalized.
  #
  # Returns:
  #   the data frame normalized.

  # first, remove columns with zero variance.
  result <- data[, sapply(data, function(n) { var(n, na.rm=TRUE) != 0 })]

  # second, scales the average to 0 and standard deviation to 1.
  result <- scale(result, center = TRUE, scale = TRUE)

  return(result)
}
# ------------------------------------------------------------------------------


CalculateAccuracyForKnn <- function(data, classes){
  # Calculates the kNN accuracy.
  #
  # Args:
  #   data
  #   classes
  #
  # Returns:
  #   the accuracy.
  #

  # applies pca for dimensionality redution
  pca.out    <- prcomp(data, scale.=T)
  pca.cumsum <- cumsum(pca.out$sdev^2)/sum(pca.out$sdev^2)
  pca.min    <- which(pca.cumsum >= 0.80)[1]

  data.transformed <- as.data.frame(pca.out$x[, 1:pca.min] %*% t(pca.out$rotation[, 1:pca.min]))
  data.transformed$Class <- classes

  set.seed(300)

  # separates train data from test data
  external.idx.train  <- createDataPartition(y=data.transformed$Class, p=0.80, list=FALSE)
  external.train.data <- data.transformed[external.idx.train, ]
  external.test.data  <- data.transformed[-external.idx.train, ]


  # setups knn to perform internal 3-fold with search grid
  knn.control <- trainControl(method = "cv", number = 3, search = "grid")
  knn.grid    <- expand.grid(k=c(1,5,11,15,21,25))

  internal.max.accuracy <- 0

  # performs an external 5-fold
  external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
  for(e in external.folds) {
    train.data <- external.train.data[e, ]
    test.data  <- external.train.data[-e, ]

    set.seed(400)
    knn.fit <- train(Class ~ ., data = train.data,
                     method="knn",
                     trControl=knn.control,
                     tuneGrid=knn.grid)

    knn.predict <- predict(knn.fit, newdata=test.data)
    knn.cm <- confusionMatrix(knn.predict, test.data$Class)
    internal.accuracy <- knn.cm$overall['Accuracy']
    if ( internal.accuracy >  internal.max.accuracy ){
      internal.max.accuracy <- internal.accuracy
      internal.best.k       <- knn.fit$bestTune$k
      cat(sprintf("[Knn] k = %d, accuracy = %.5f\n", knn.fit$bestTune$k, internal.accuracy))
    }
  }

  # calculates the accuracy after external 5-fold to select hyperparameters
  set.seed(400)
  external.knn.fit <- train(Class ~ ., data = external.train.data,
                   method="knn",
                   trControl=knn.control,
                   tuneGrid=expand.grid(k=c(internal.best.k)))

  external.knn.predict <- predict(external.knn.fit, newdata=external.test.data)
  external.knn.cm <- confusionMatrix(external.knn.predict, external.test.data$Class)
  external.accuracy <- external.knn.cm$overall['Accuracy']

  return (external.accuracy)
}

CalculateAccuracyForSvm <- function(data, classes){
  # Calculates the Svm accuracy.
  #
  # Args:
  #   data
  #   classes
  #
  # Returns:
  #   the accuracy.
  #
  set.seed(300)
  data$Class <- classes

  # separates train data from test data.
  external.idx.train  <- createDataPartition(y=data$Class, p=0.80, list=FALSE)
  external.train.data <- data[external.idx.train, ]
  external.test.data  <- data[-external.idx.train, ]
  external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)

  # setups svm to perform internal 3-fold with search grid.
  svm.control <- trainControl(method = "cv", number = 3, search = "grid")
  svm.grid <- expand.grid(C=c(2**(-5), 2**(0), 2**(5), 2**(10)),
                          sigma=c(2**(-15), 2**(-10), 2**(-5), 2**(0), 2**(5)))

  internal.max.accuracy <- 0

  # performs an external 5-fold.
  external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
  for(e in external.folds) {
    train.data <- external.train.data[e, ]
    test.data  <- external.train.data[-e, ]

    set.seed(400)
    svm.fit <- train(Class ~ ., data = train.data,
                     method="svmRadial",
                     trControl=svm.control,
                     tuneGrid=svm.grid)

    svm.predict <- predict(svm.fit, newdata=test.data)
    svm.cm <- confusionMatrix(svm.predict, test.data$Class)

    internal.accuracy <- svm.cm$overall['Accuracy']
    if ( internal.accuracy >  internal.max.accuracy ){
      internal.max.accuracy <- internal.accuracy
      internal.best.C       <- svm.fit$bestTune$C
      internal.best.sigma   <- svm.fit$bestTune$sigma
      cat(sprintf("[Svm] C= %.5f, sigma= %.5f, accuracy = %.5f\n",
                  svm.fit$bestTune$C,
                  svm.fit$bestTune$sigma,
                  svm.cm$overall['Accuracy']))
    }
  }

  # calculates the accuracy after external 5-fold to select hyperparameters.
  set.seed(400)
  external.svm.fit <- train(Class ~ ., data = external.train.data,
                     method="svmRadial",
                     trControl=svm.control,
                     tuneGrid=expand.grid(C=internal.best.C, sigma=internal.best.sigma))

  external.svm.predict <- predict(external.svm.fit, newdata=external.test.data)
  external.svm.cm <- confusionMatrix(external.svm.predict, external.test.data$Class)

  return (external.svm.cm$overall['Accuracy'])
}

CalculateAccuracyForNnet <- function(data, classes){
  # Calculates the Neural Network accuracy.
  #
  # Args:
  #   data
  #   classes
  #
  # Returns:
  #   the accuracy.
  #

  set.seed(300)
  data$Class <- classes

  # separates train data from test data.
  external.idx.train  <- createDataPartition(y=data$Class, p=0.80, list=FALSE)
  external.train.data <- data[external.idx.train, ]
  external.test.data  <- data[-external.idx.train, ]

  # setups nnet to perform internal 3-fold with search grid.
  nnet.control <- trainControl(method = "cv", number = 3, search = "grid")
  nnet.grid <- expand.grid(size=c(10, 20, 30, 40), decay=(0.5))

  internal.max.accuracy <- 0

  # performs an external 5-fold.
  external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
  for(e in external.folds) {
    train.data <- external.train.data[e, ]
    test.data  <- external.train.data[-e, ]

    set.seed(400)
    nnet.fit <- train(Class ~ .,
                      data = train.data,
                      method="nnet",
                      trControl=nnet.control,
                      tuneGrid=nnet.grid,
                      MaxNWts=5000,
                      verbose=FALSE)

    nnet.predict <- predict(nnet.fit, newdata=test.data)
    nnet.cm <- confusionMatrix(nnet.predict, test.data$Class)

    internal.accuracy <- nnet.cm$overall['Accuracy']
    if ( internal.accuracy >  internal.max.accuracy ){
      internal.max.accuracy <- internal.accuracy
      internal.best.size     <- nnet.fit$bestTune$size
      internal.best.decay    <- nnet.fit$bestTune$decay
      cat(sprintf("[Nnet] size= %d, decay= %.2f, accuracy = %.5f\n",
                  nnet.fit$bestTune$size,
                  nnet.fit$bestTune$decay,
                  nnet.cm$overall['Accuracy']))
    }

  }

  # calculates the accuracy after external 5-fold to select hyperparameters.
  set.seed(400)
  external.nnet.fit <- train(Class ~ .,
                             data = external.train.data,
                             method="nnet",
                             trControl=nnet.control,
                             tuneGrid=expand.grid(size=c(internal.best.size),
                                                  decay=c(internal.best.decay)),
                             MaxNWts=5000,
                             verbose=FALSE)

  external.nnet.predict <- predict(external.nnet.fit, newdata=external.test.data)
  external.nnet.cm <- confusionMatrix(external.nnet.predict, external.test.data$Class)

  return (external.nnet.cm$overall['Accuracy'])
}

CalculateAccuracyForRf <- function(data, classes){
  # Calculates the Random Forest accuracy
  #
  # Args:
  #   data
  #   classes
  #
  # Returns:
  #   the accuracy.
  #

  set.seed(300)
  data$Class <- classes

  # separates train data from test data.
  external.idx.train  <- createDataPartition(y=data$Class, p=0.80, list=FALSE)
  external.train.data <- data[external.idx.train, ]
  external.test.data  <- data[-external.idx.train, ]


  # setups rf to perform internal 3-fold with search grid.
  rf.control <- trainControl(method = "cv", number = 3, search = "grid")
  rf.grid    <- expand.grid(mtry=c(10, 15, 20, 25))

  internal.max.accuracy <- 0

  # performs an external 5-fold.
  external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
  for(e in external.folds) {
    train.data <- external.train.data[e, ]
    test.data  <- external.train.data[-e, ]

    for(n in c(100, 200, 300, 400)){
      cat(sprintf("ntree = %d\n", n))
      rf.fit <- train(Class ~ .,
                      data = train.data,
                      method="rf",
                      trControl=rf.control,
                      tuneGrid=rf.grid,
                      ntree=n)

      rf.predict <- predict(rf.fit, newdata=test.data)
      rf.cm <- confusionMatrix(rf.predict, test.data$Class)

      internal.accuracy <- rf.cm$overall['Accuracy']
      if ( internal.accuracy >  internal.max.accuracy ){
        internal.max.accuracy <- internal.accuracy
        internal.best.mtry    <- rf.fit$bestTune$mtry
        internal.best.ntree   <- n
        cat(sprintf("[Rf] mtry= %d, ntree= %d, accuracy = %.5f\n",
                    rf.fit$bestTune$mtry,
                    n,
                    rf.cm$overall['Accuracy']))
      }
    }
  }

  # calculates the accuracy after external 5-fold to select hyperparameters.
  set.seed(400)
  external.rf.fit <- train(Class ~ ., data = external.train.data,
                           method="rf",
                           trControl=rf.control,
                           tuneGrid=expand.grid(mtry=c(internal.best.mtry)),
                           ntree=internal.best.ntree)

  external.rf.predict <- predict(external.rf.fit, newdata=external.test.data)
  external.rf.cm <- confusionMatrix(external.rf.predict, external.test.data$Class)

  return  (external.rf.cm$overall['Accuracy'])
}

CalculateAccuracyForGbm<- function(data, classes){
```

```
# Calculates the Gradient Boosting Machine
#
# Args:
#   data
#   classes
#
# Returns:
#   the accuracy.
#

set.seed(300)
data$Class <- classes

# separates train data from test data.
external.idx.train  <- createDataPartition(y=data$Class, p=0.80, list=FALSE)
external.train.data <- data[external.idx.train, ]
external.test.data  <- data[-external.idx.train, ]

# setups knn to perform internal 3-fold with search grid.
gbm.control <- trainControl(method = "cv", number = 3, search = "grid")
gbm.grid     <- expand.grid(interaction.depth=5, n.trees=c(30, 70, 100)
                           , shrinkage=c(0.1,0.5), n.minobsinnode=10)
internal.max.accuracy <- 0

# performs an external 5-fold.
external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
for(e in external.folds) {
  train.data <- external.train.data[e, ]
  test.data  <- external.train.data[-e, ]

  set.seed(400)
  gbm.fit <- train(Class ~ ., data = train.data,
                   method="gbm",
                   metric="Accuracy",
                   trControl=gbm.control,
                   tuneGrid=gbm.grid)

  gbm.predict <- predict(gbm.fit, newdata=test.data)
  gbm.cm <- confusionMatrix(gbm.predict, test.data$Class)

  internal.accuracy <- gbm.cm$overall['Accuracy']

  if ( internal.accuracy >  internal.max.accuracy ){
    internal.max.accuracy <- internal.accuracy
    internal.best.interaction.depth    <- gbm.fit$bestTune$interaction.depth
    internal.best.n.trees    <- gbm.fit$bestTune$n.trees
    internal.best.shrinkage    <- gbm.fit$bestTune$shrinkage
    internal.best.n.minobsinnode    <- gbm.fit$bestTune$n.minobsinnode

    cat(sprintf("[Rf] interaction.depth= %d, n.trees= %d, shrinkage= %.2f,
                n.minobsinnode= %d, accuracy = %.5f\n",
                internal.best.interaction.depth,
                internal.best.n.trees,
                internal.best.shrinkage,
                internal.best.n.minobsinnode,
                internal.max.accuracy))
  }

}

# calculates the accuracy after external 5-fold to select hyperparameters.
set.seed(400)
external.gbm.fit <- train(Class ~ ., data = external.train.data,
                          method="gbm",
                          metric="Accuracy",
                          trControl=gbm.control,
                              tuneGrid=expand.grid(interaction.depth=internal.best.interaction.
                                  depth,
                                                    n.trees=c(internal.best.n.trees),
                                                    shrinkage=c(internal.best.shrinkage),
                                                    n.minobsinnode=internal.best.n.minobsinnode))

  external.gbm.predict <- predict(external.gbm.fit, newdata=external.test.data)
  external.gbm.cm <- confusionMatrix(external.gbm.predict, external.test.data$Class)

  return (external.gbm.cm$overall['Accuracy'])
}

# ------------------------------------------------------------------------------
# main function
# ------------------------------------------------------------------------------
main <- function() {
  # reads raw data from files.
  secom.data <- ReadDataFile('secom.data')
  secom.data.labels <- ReadDataFile('secom_labels.data')

  # makes inputation and normalization.
  secom.data <- InputByMean(secom.data)
  secom.data <- as.data.frame(NormalizeData(secom.data))

  # transform class data in R factor.
  secom.data.classes <- as.factor(make.names(secom.data.labels$V1))

  output <- data.frame(classifier=character(), accuracy=numeric())

  knn.accuracy <- CalculateAccuracyForKnn(secom.data, secom.data.classes)
  output <- rbind(output, data.frame(classifier="knn",accuracy=knn.accuracy))
  cat(sprintf("Accuracy for knn was %.5f\n", knn.accuracy))

  svm.accuracy <- CalculateAccuracyForSvm(secom.data, secom.data.classes)
  output <- rbind(output, data.frame(classifier="svm",accuracy=svm.accuracy))
  cat(sprintf("Accuracy for svm was %.5f\n", svm.accuracy))

  nnet.accuracy <- CalculateAccuracyForNnet(secom.data, secom.data.classes)
  output <- rbind(output, data.frame(classifier="nnet",accuracy=nnet.accuracy))
  cat(sprintf("Accuracy for nnet was %.5f\n", nnet.accuracy))

  rf.accuracy <- CalculateAccuracyForRf(secom.data, secom.data.classes)
  output <- rbind(output, data.frame(classifier="rf",accuracy=rf.accuracy))
  cat(sprintf("Accuracy for rf was %.5f\n", rf.accuracy))

  gbm.accuracy <-
    CalculateAccuracyForGbm(secom.data, secom.data.classes)
  output <- rbind(output, data.frame(classifier="gbm",accuracy=gbm.accuracy))
  cat(sprintf("Accuracy for gbm was %.5f\n", gbm.accuracy))

  write.csv(output, file='./data/result.csv', row.names = FALSE, quote = FALSE)
}

main()
```