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')

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

```
37 # common functions
38 #-------------------------------------------------------------------------------
39 ReadDataFile <- function(name){
40   # Reads a data file in csv format.
41   #
42   # Args:
43   #   name: file name to be read.
44   #
45   # Returns:
46   #   the data frame with rows and columns from file.
47   #
48   result <- read.csv(file = paste('./data/',name,sep=''), header = FALSE, sep = ' ')
49   return(result)
50 }
51
52 InputByMean <- function(data){
53   # Performs the input missing data by mean.
54   #
55   # Args:
56   #   data: data frame to be processed.
57   #
58   # Returns:
59   #   the data frame processed.
60   #
61   for(i in 1:ncol(data)){
62     data[is.na(data[,i]), i] <- mean(data[,i], na.rm = TRUE)
63   }
64   return(data)
65 }
66
67 NormalizeData <- function(data){
68   # Standardizes the columns for average 0 and standard deviation 1
69   #
70   # Args:
71   #   data: data frame to be normalized.
72   #
73   # Returns:
74   #   the data frame normalized.
75
76   # first, remove columns with zero variance.
77   result <- data[, sapply(data, function(n) { var(n, na.rm=TRUE) != 0 })]
78
79   # second, scales the average to 0 and standard deviation to 1.
80   result <- scale(result, center = TRUE, scale = TRUE)
81
82   return(result)
83 }
84 #-------------------------------------------------------------------------------
85
86
87 CalculateAccuracyForKnn <- function(data, classes){
88   # Calculates the kNN accuracy.
89   #
90   # Args:
91   #   data
92   #   classes
93   #
94   # Returns:
95   #   the accuracy.
96   #
97
98   # applies pca for dimensionality redution
99   pca.out    <- prcomp(data, scale.=T)
100  pca.cumsum <- cumsum(pca.out$sdev^2)/sum(pca.out$sdev^2)
101  pca.min    <- which(pca.cumsum >= 0.80)[1]
102
```

```
103   data.transformed <- as.data.frame(pca.out$x[, 1:pca.min] %*% t(pca.out$rotation[, 1:pca.min]))
104   data.transformed$Class <- classes
105
106   set.seed(300)
107
108   # separates train data from test data
109   external.idx.train  <- createDataPartition(y=data.transformed$Class, p=0.80, list=FALSE)
110   external.train.data <- data.transformed[external.idx.train, ]
111   external.test.data  <- data.transformed[-external.idx.train, ]
112
113
114   # setups knn to perform internal 3-fold with search grid
115   knn.control <- trainControl(method = "cv", number = 3, search = "grid")
116   knn.grid    <- expand.grid(k=c(1,5,11,15,21,25))
117
118   internal.max.accuracy <- 0
119
120   # performs an external 5-fold
121   external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
122   for(e in external.folds) {
123     train.data <- external.train.data[e, ]
124     test.data  <- external.train.data[-e, ]
125
126     set.seed(400)
127     knn.fit <- train(Class ~ ., data = train.data,
128                      method="knn",
129                      trControl=knn.control,
130                      tuneGrid=knn.grid)
131
132     knn.predict <- predict(knn.fit, newdata=test.data)
133     knn.cm <- confusionMatrix(knn.predict, test.data$Class)
134     internal.accuracy <- knn.cm$overall['Accuracy']
135     if ( internal.accuracy > internal.max.accuracy ){
136       internal.max.accuracy <- internal.accuracy
137       internal.best.k       <- knn.fit$bestTune$k
138       cat(sprintf("[Knn] k = %d, accuracy = %.5f\n", knn.fit$bestTune$k, internal.accuracy))
139     }
140   }
141
142   # calculates the accuracy after external 5-fold to select hyperparameters
143   set.seed(400)
144   external.knn.fit <- train(Class ~ ., data = external.train.data,
145                     method="knn",
146                     trControl=knn.control,
147                     tuneGrid=expand.grid(k=c(internal.best.k)))
148
149   external.knn.predict <- predict(external.knn.fit, newdata=external.test.data)
150   external.knn.cm <- confusionMatrix(external.knn.predict, external.test.data$Class)
151   external.accuracy <- external.knn.cm$overall['Accuracy']
152
153   return (external.accuracy)
154 }
155
156 CalculateAccuracyForSvm <- function(data, classes){
157   # Calculates the Svm accuracy.
158   #
159   # Args:
160   #   data
161   #   classes
162   #
163   # Returns:
164   #   the accuracy.
165   #
166   set.seed(300)
167   data$Class <- classes
168
```

```
169   # separates train data from test data.
170   external.idx.train  <- createDataPartition(y=data$Class, p=0.80, list=FALSE)
171   external.train.data <- data[external.idx.train, ]
172   external.test.data  <- data[-external.idx.train, ]
173   external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
174
175   # setups svm to perform internal 3-fold with search grid.
176   svm.control <- trainControl(method = "cv", number = 3, search = "grid")
177   svm.grid <- expand.grid(C=c(2**(-5), 2**(0), 2**(5), 2**(10)),
178                           sigma=c(2**(-15), 2**(-10), 2**(-5), 2**(0), 2**(5)))
179
180   internal.max.accuracy <- 0
181
182   # performs an external 5-fold.
183   external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
184   for(e in external.folds) {
185     train.data <- external.train.data[e, ]
186     test.data  <- external.train.data[-e, ]
187
188     set.seed(400)
189     svm.fit <- train(Class ~ ., data = train.data,
190                      method="svmRadial",
191                      trControl=svm.control,
192                      tuneGrid=svm.grid)
193
194     svm.predict <- predict(svm.fit, newdata=test.data)
195     svm.cm <- confusionMatrix(svm.predict, test.data$Class)
196
197     internal.accuracy <- svm.cm$overall['Accuracy']
198     if ( internal.accuracy >  internal.max.accuracy ){
199       internal.max.accuracy <- internal.accuracy
200       internal.best.C       <- svm.fit$bestTune$C
201       internal.best.sigma   <- svm.fit$bestTune$sigma
202       cat(sprintf("[Svm] C= %.5f, sigma= %.5f, accuracy = %.5f\n",
203                   svm.fit$bestTune$C,
204                   svm.fit$bestTune$sigma,
205                   svm.cm$overall['Accuracy']))
206     }
207   }
208
209   # calculates the accuracy after external 5-fold to select hyperparameters.
210   set.seed(400)
211   external.svm.fit <- train(Class ~ ., data = external.train.data,
212                             method="svmRadial",
213                             trControl=svm.control,
214                             tuneGrid=expand.grid(C=internal.best.C, sigma=internal.best.sigma))
215
216   external.svm.predict <- predict(external.svm.fit, newdata=external.test.data)
217   external.svm.cm <- confusionMatrix(external.svm.predict, external.test.data$Class)
218
219   return (external.svm.cm$overall['Accuracy'])
220 }
221
222 CalculateAccuracyForNnet <- function(data, classes){
223   # Calculates the Neural Network accuracy.
224   #
225   # Args:
226   #   data
227   #   classes
228   #
229   # Returns:
230   #   the accuracy.
231   #
232
233   set.seed(300)
234   data$Class <- classes
```

```
235
236   # separates train data from test data.
237   external.idx.train  <- createDataPartition(y=data$Class, p=0.80, list=FALSE)
238   external.train.data <- data[external.idx.train, ]
239   external.test.data  <- data[-external.idx.train, ]
240
241   # setups nnet to perform internal 3-fold with search grid.
242   nnet.control <- trainControl(method = "cv", number = 3, search = "grid")
243   nnet.grid <- expand.grid(size=c(10, 20, 30, 40), decay=(0.5))
244
245   internal.max.accuracy <- 0
246
247   # performs an external 5-fold.
248   external.folds       <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
249   for(e in external.folds) {
250     train.data <- external.train.data[e, ]
251     test.data  <- external.train.data[-e, ]
252
253     set.seed(400)
254     nnet.fit <- train(Class ~ .,
255                       data = train.data,
256                       method="nnet",
257                       trControl=nnet.control,
258                       tuneGrid=nnet.grid,
259                       MaxNWts=5000,
260                       verbose=FALSE)
261
262     nnet.predict <- predict(nnet.fit, newdata=test.data)
263     nnet.cm <- confusionMatrix(nnet.predict, test.data$Class)
264
265     internal.accuracy <- nnet.cm$overall['Accuracy']
266     if ( internal.accuracy >  internal.max.accuracy ){
267       internal.max.accuracy <- internal.accuracy
268       internal.best.size    <- nnet.fit$bestTune$size
269       internal.best.decay   <- nnet.fit$bestTune$decay
270       cat(sprintf("[Nnet] size= %d, decay= %.2f, accuracy = %.5f\n",
271                   nnet.fit$bestTune$size,
272                   nnet.fit$bestTune$decay,
273                   nnet.cm$overall['Accuracy']))
274     }
275
276   }
277
278   # calculates the accuracy after external 5-fold to select hyperparameters.
279   set.seed(400)
280   external.nnet.fit <- train(Class ~ .,
281                              data = external.train.data,
282                              method="nnet",
283                              trControl=nnet.control,
284                              tuneGrid=expand.grid(size=c(internal.best.size),
285                                                   decay=c(internal.best.decay)),
286                              MaxNWts=5000,
287                              verbose=FALSE)
288
289   external.nnet.predict <- predict(external.nnet.fit, newdata=external.test.data)
290   external.nnet.cm <- confusionMatrix(external.nnet.predict, external.test.data$Class)
291
292   return (external.nnet.cm$overall['Accuracy'])
293 }
294
295 CalculateAccuracyForRf <- function(data, classes){
296   # Calculates the Random Forest accuracy
297   #
298   # Args:
299   #   data
300   #   classes
```

```
301   #
302   # Returns:
303   #   the accuracy.
304   #
305
306   set.seed(300)
307   data$Class <- classes
308
309   # separates train data from test data.
310   external.idx.train  <- createDataPartition(y=data$Class, p=0.80, list=FALSE)
311   external.train.data <- data[external.idx.train, ]
312   external.test.data  <- data[-external.idx.train, ]
313
314
315   # setups rf to perform internal 3-fold with search grid.
316   rf.control <- trainControl(method = "cv", number = 3, search = "grid")
317   rf.grid    <- expand.grid(mtry=c(10, 15, 20, 25))
318
319   internal.max.accuracy <- 0
320
321   # performs an external 5-fold.
322   external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
323   for(e in external.folds) {
324     train.data <- external.train.data[e, ]
325     test.data  <- external.train.data[-e, ]
326
327     for(n in c(100, 200, 300, 400)){
328       cat(sprintf("ntree = %d\n", n))
329       rf.fit <- train(Class ~ .,
330                       data = train.data,
331                       method="rf",
332                       trControl=rf.control,
333                       tuneGrid=rf.grid,
334                       ntree=n)
335
336       rf.predict <- predict(rf.fit, newdata=test.data)
337       rf.cm <- confusionMatrix(rf.predict, test.data$Class)
338
339       internal.accuracy <- rf.cm$overall['Accuracy']
340       if ( internal.accuracy >  internal.max.accuracy ){
341         internal.max.accuracy <- internal.accuracy
342         internal.best.mtry    <- rf.fit$bestTune$mtry
343         internal.best.ntree   <- n
344         cat(sprintf("[Rf] mtry= %d, ntree= %d, accuracy = %.5f\n",
345                     rf.fit$bestTune$mtry,
346                     n,
347                     rf.cm$overall['Accuracy']))
348       }
349     }
350   }
351
352   # calculates the accuracy after external 5-fold to select hyperparameters.
353   set.seed(400)
354   external.rf.fit <- train(Class ~ ., data = external.train.data,
355                            method="rf",
356                            trControl=rf.control,
357                            tuneGrid=expand.grid(mtry=c(internal.best.mtry)),
358                            ntree=internal.best.ntree)
359
360   external.rf.predict <- predict(external.rf.fit, newdata=external.test.data)
361   external.rf.cm <- confusionMatrix(external.rf.predict, external.test.data$Class)
362
363   return (external.rf.cm$overall['Accuracy'])
364 }
365
366 CalculateAccuracyForGbm<- function(data, classes){
```

```
367  # Calculates the Gradient Boosting Machine
368  #
369  # Args:
370  #   data
371  #   classes
372  #
373  # Returns:
374  #   the accuracy.
375  #
376
377  set.seed(300)
378  data$Class <- classes
379
380  # separates train data from test data.
381  external.idx.train  <- createDataPartition(y=data$Class, p=0.80, list=FALSE)
382  external.train.data <- data[external.idx.train, ]
383  external.test.data  <- data[-external.idx.train, ]
384
385  # setups knn to perform internal 3-fold with search grid.
386  gbm.control <- trainControl(method = "cv", number = 3, search = "grid")
387  gbm.grid    <- expand.grid(interaction.depth=5, n.trees=c(30, 70, 100)
388                             , shrinkage=c(0.1,0.5), n.minobsinnode=10)
389  internal.max.accuracy <- 0
390
391  # performs an external 5-fold.
392  external.folds      <- createFolds(external.train.data, k = 5, returnTrain = TRUE)
393  for(e in external.folds) {
394    train.data <- external.train.data[e, ]
395    test.data  <- external.train.data[-e, ]
396
397    set.seed(400)
398    gbm.fit <- train(Class ~ ., data = train.data,
399                     method="gbm",
400                     metric="Accuracy",
401                     trControl=gbm.control,
402                     tuneGrid=gbm.grid)
403
404    gbm.predict <- predict(gbm.fit, newdata=test.data)
405    gbm.cm <- confusionMatrix(gbm.predict, test.data$Class)
406
407    internal.accuracy <- gbm.cm$overall['Accuracy']
408
409    if ( internal.accuracy >  internal.max.accuracy ){
410      internal.max.accuracy <- internal.accuracy
411      internal.best.interaction.depth    <- gbm.fit$bestTune$interaction.depth
412      internal.best.n.trees   <- gbm.fit$bestTune$n.trees
413      internal.best.shrinkage   <- gbm.fit$bestTune$shrinkage
414      internal.best.n.minobsinnode   <- gbm.fit$bestTune$n.minobsinnode
415
416      cat(sprintf("[Rf] interaction.depth= %d, n.trees= %d, shrinkage= %.2f,
417                  n.minobsinnode= %d, accuracy = %.5f\n",
418                  internal.best.interaction.depth,
419                  internal.best.n.trees,
420                  internal.best.shrinkage,
421                  internal.best.n.minobsinnode,
422                  internal.max.accuracy))
423    }
424
425  }
426
427  # calculates the accuracy after external 5-fold to select hyperparameters.
428  set.seed(400)
429  external.gbm.fit <- train(Class ~ ., data = external.train.data,
430                            method="gbm",
431                            metric="Accuracy",
432                            trControl=gbm.control,
```

```
433                                 tuneGrid=expand.grid(interaction.depth=internal.best.interaction.
                                        depth,
434                                                      n.trees=c(internal.best.n.trees),
435                                                      shrinkage=c(internal.best.shrinkage),
436                                                      n.minobsinnode=internal.best.n.minobsinnode))
437
438   external.gbm.predict <- predict(external.gbm.fit, newdata=external.test.data)
439   external.gbm.cm <- confusionMatrix(external.gbm.predict, external.test.data$Class)
440
441   return (external.gbm.cm$overall['Accuracy'])
442 }
443
444 #------------------------------------------------------------------------------
445 # main function
446 #------------------------------------------------------------------------------
447 main <- function() {
448   # reads raw data from files.
449   secom.data <- ReadDataFile('secom.data')
450   secom.data.labels <- ReadDataFile('secom_labels.data')
451
452   # makes inputation and normalization.
453   secom.data <- InputByMean(secom.data)
454   secom.data <- as.data.frame(NormalizeData(secom.data))
455
456   # transform class data in R factor.
457   secom.data.classes <- as.factor(make.names(secom.data.labels$V1))
458
459   output <- data.frame(classifier=character(), accuracy=numeric())
460
461   knn.accuracy <- CalculateAccuracyForKnn(secom.data, secom.data.classes)
462   output <- rbind(output, data.frame(classifier="knn",accuracy=knn.accuracy))
463   cat(sprintf("Accuracy for knn was %.5f\n", knn.accuracy))
464
465   svm.accuracy <- CalculateAccuracyForSvm(secom.data, secom.data.classes)
466   output <- rbind(output, data.frame(classifier="svm",accuracy=svm.accuracy))
467   cat(sprintf("Accuracy for svm was %.5f\n", svm.accuracy))
468
469   nnet.accuracy <- CalculateAccuracyForNnet(secom.data, secom.data.classes)
470   output <- rbind(output, data.frame(classifier="nnet",accuracy=nnet.accuracy))
471   cat(sprintf("Accuracy for nnet was %.5f\n", nnet.accuracy))
472
473   rf.accuracy <- CalculateAccuracyForRf(secom.data, secom.data.classes)
474   output <- rbind(output, data.frame(classifier="rf",accuracy=rf.accuracy))
475   cat(sprintf("Accuracy for rf was %.5f\n", rf.accuracy))
476
477   gbm.accuracy <-
478     CalculateAccuracyForGbm(secom.data, secom.data.classes)
479   output <- rbind(output, data.frame(classifier="gbm",accuracy=gbm.accuracy))
480   cat(sprintf("Accuracy for gbm was %.5f\n", gbm.accuracy))
481
482   write.csv(output, file='./data/result.csv', row.names = FALSE, quote = FALSE)
483 }
484
485 main()
```