AprendizajeSupervisadoVoluntaria\_EGT

Elena Garcia Torres

8 de enero de 2017

# INDICE

* 1. Enunciado
  2. Filtrar el número de variables de entrada
  3. Clasificador K-NN (ten fold cross-validation)
  4. Clasificador C-SVM con kernel linear (ten fold cross-validation)
  5. Clasificador C-SVM con kernel no lineal RBF (ten fold cross-validation)
  6. Random Forest (ten fold cross-validation)
  7. Proyectar Datos (PCA)
  8. Comparar los diferentes modelos

## 1. Enunciado

Para los datos de Spam proporcionados en el moodle realizar las siguientes tareas: \* Filtrar el número de variables de entrada. Identificar las palabras y símbolos que tienen mayor relevancia para la clasificación de Spam. \* Para el clasificador k-NN, obtener el porcentaje de error y el índice kappa utilizando ten fold cross-validation. Determinar el valor de k óptimo. ¿ Qué ventajas e inconvenientes tiene este clasificador ? \* Para el clasificador C-SVM con kernel lineal, obtener el porcentaje de error y el índice kappa utilizando ten fold cross-validation. Determinar el valor óptimo para el parámetro C. ¿ Qué podría comentar sobre la eficiencia computacional ? ¿ Y sobre la capacidad de generalización ? \* Repetir el proceso anterior pero utilizando kernel no lineal RBF. Obtener los parámetros óptimos. ¿ Qué podría comentar ? \* Para el clasificador Random Forest, obtener el porcentaje de error y el índice kappa utilizando ten fold cross-validation. Comentar el resultado y compararlo con los anteriores. \* Proyectar los datos sobre un subespacio de dimensión menor utilizando PCA. ¿Cuántas componentes principales se deben calcular ? Comparar los resultados con los obtenidos en puntos anteriores. \* Comparar los diferentes modelos y determinar cuáles serían los mejores en esta aplicación atendiendo a índice de error y eficiencia computacional.

## 2. Filtrar el número de variables de entrada

# Se comprueba si las librerias están isntaladas y si no lo están, se instalan

currentDir <- getwd()  
  
libs <- c("tm", "NLP", "kernlab", "caret", "ggplot2", "randomForest", "lattice")  
 for (i in libs){  
 print(i)  
 if(!require(i, character.only = TRUE)) { install.packages(i); library(i) }  
 }

## [1] "tm"

## Loading required package: tm

## Loading required package: NLP

## [1] "NLP"  
## [1] "kernlab"

## Loading required package: kernlab

## [1] "caret"

## Loading required package: caret

## Loading required package: lattice

## Loading required package: ggplot2

##   
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':  
##   
## alpha

## The following object is masked from 'package:NLP':  
##   
## annotate

## [1] "ggplot2"  
## [1] "randomForest"

## Loading required package: randomForest

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

## [1] "lattice"

data(spam)  
  
# se asigna seed para no tener valores aleatorios  
cte.seed <- 1234  
  
set.seed(cte.seed)  
spam.traintest = rbinom(4601, size = 1, prob = 0.5)  
table(spam.traintest)

## spam.traintest  
## 0 1   
## 2296 2305

train.Spam = spam[spam.traintest == 1, ]  
test.Spam = spam[spam.traintest == 0, ]  
  
table(train.Spam$type)

##   
## nonspam spam   
## 1405 900

table(test.Spam$type)

##   
## nonspam spam   
## 1383 913

# Se comprueba si las dimensiones son congruentes  
dim(train.Spam)

## [1] 2305 58

dim(test.Spam)

## [1] 2296 58

# Las proporciones se mantienen  
prop.table( table(spam$type))

##   
## nonspam spam   
## 0.6059552 0.3940448

prop.table( table(train.Spam$type))

##   
## nonspam spam   
## 0.6095445 0.3904555

prop.table( table(test.Spam$type))

##   
## nonspam spam   
## 0.6023519 0.3976481

#Identificar las palabras y símbolos que tienen mayor relevancia para la clasificación de Spam

## 3. Clasificador K-NN (ten fold cross-validation)

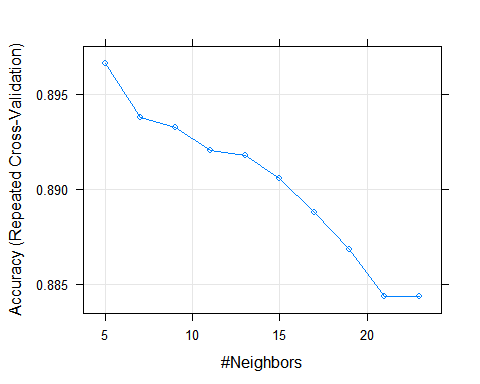
set.seed (cte.seed)  
   
 # se usa la funcion "train" para construir el modelo K-NN  
 Knn.spam <- train(type ~ ., data = train.Spam, method = "knn", tuneLength = 10, preProc = c("center", "scale"))  
   
 Knn.spam

## k-Nearest Neighbors   
##   
## 2305 samples  
## 57 predictor  
## 2 classes: 'nonspam', 'spam'   
##   
## Pre-processing: centered (57), scaled (57)   
## Resampling: Bootstrapped (25 reps)   
## Summary of sample sizes: 2305, 2305, 2305, 2305, 2305, 2305, ...   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.8756117 0.7365372  
## 7 0.8761215 0.7367925  
## 9 0.8779876 0.7403640  
## 11 0.8797522 0.7434887  
## 13 0.8796356 0.7427898  
## 15 0.8802308 0.7435507  
## 17 0.8791350 0.7411664  
## 19 0.8770666 0.7365040  
## 21 0.8769927 0.7358962  
## 23 0.8755585 0.7325637  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 15.

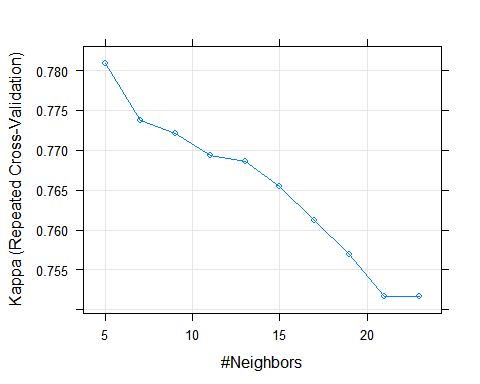
# Variacion cruzada, trainControl  
 ctrl <- trainControl(method = "repeatedcv", repeats = 5)  
 Knn.spam.ctrl <- train(type ~ ., data = train.Spam, method = "knn", tuneLength = 10, trControl = ctrl, preProc = c("center", "scale"))  
 Knn.spam.ctrl

## k-Nearest Neighbors   
##   
## 2305 samples  
## 57 predictor  
## 2 classes: 'nonspam', 'spam'   
##   
## Pre-processing: centered (57), scaled (57)   
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 2074, 2074, 2074, 2074, 2075, 2075, ...   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.8966520 0.7809495  
## 7 0.8937896 0.7737434  
## 9 0.8932776 0.7721745  
## 11 0.8920583 0.7694054  
## 13 0.8917971 0.7686450  
## 15 0.8905831 0.7654454  
## 17 0.8887593 0.7612556  
## 19 0.8868500 0.7569717  
## 21 0.8843369 0.7516359  
## 23 0.8843362 0.7516205  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 5.

plot(Knn.spam.ctrl)



plot( Knn.spam.ctrl, metric="Kappa")



# Predicción  
 Knn.spam.predict <- predict(Knn.spam.ctrl, newdata = test.Spam )  
   
 # La eval del modelo se hace o por la matriz de confsion o tabla de contigencia y por la matrix RO  
 confusionMatrix(Knn.spam.predict, test.Spam$type )

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction nonspam spam  
## nonspam 1279 137  
## spam 104 776  
##   
## Accuracy : 0.895   
## 95% CI : (0.8818, 0.9073)  
## No Information Rate : 0.6024   
## P-Value [Acc > NIR] : < 2e-16   
##   
## Kappa : 0.7795   
## Mcnemar's Test P-Value : 0.03927   
##   
## Sensitivity : 0.9248   
## Specificity : 0.8499   
## Pos Pred Value : 0.9032   
## Neg Pred Value : 0.8818   
## Prevalence : 0.6024   
## Detection Rate : 0.5571   
## Detection Prevalence : 0.6167   
## Balanced Accuracy : 0.8874   
##   
## 'Positive' Class : nonspam   
##

## 4. Clasificador C-SVM con kernel lineal (ten fold cross-validation)

set.seed(cte.seed)  
 svm.spam.ctrl <- trainControl(method="repeatedcv", repeats=5)  
 svm.spam.lineal <- train (type ~ ., data = train.Spam, method = "svmLinear", tuneLength = 10, trControl = svm.spam.ctrl)  
   
 svm.spam.lineal

## Support Vector Machines with Linear Kernel   
##   
## 2305 samples  
## 57 predictor  
## 2 classes: 'nonspam', 'spam'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 2075, 2074, 2075, 2075, 2074, 2074, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.9219108 0.8347474  
##   
## Tuning parameter 'C' was held constant at a value of 1  
##

svm.spam.lineal.predict <- predict(svm.spam.lineal, test.Spam, "raw")  
   
 confusionMatrix(svm.spam.lineal.predict, test.Spam$type)

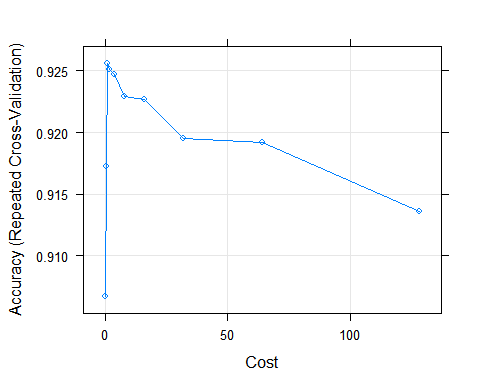
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction nonspam spam  
## nonspam 1318 103  
## spam 65 810  
##   
## Accuracy : 0.9268   
## 95% CI : (0.9154, 0.9371)  
## No Information Rate : 0.6024   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8462   
## Mcnemar's Test P-Value : 0.004309   
##   
## Sensitivity : 0.9530   
## Specificity : 0.8872   
## Pos Pred Value : 0.9275   
## Neg Pred Value : 0.9257   
## Prevalence : 0.6024   
## Detection Rate : 0.5740   
## Detection Prevalence : 0.6189   
## Balanced Accuracy : 0.9201   
##   
## 'Positive' Class : nonspam   
##

## 5. Clasificador C-SVM con kernel no lineal RBF (ten fold cross-validation)

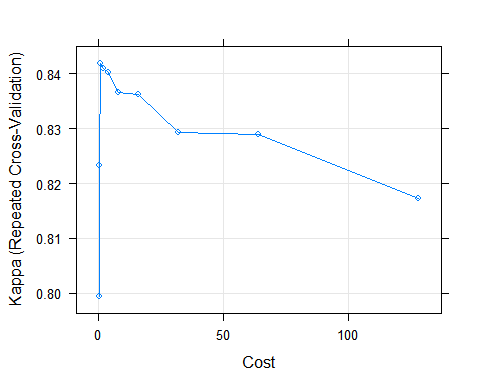
set.seed(cte.seed)  
 svmR.spam.ctrl <- trainControl(method="repeatedcv", repeats=5)  
 svmR.spam.radial <- train (type ~ ., data = train.Spam, method = "svmRadial", tuneLength = 10, trControl = svmR.spam.ctrl)  
 svmR.spam.radial

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 2305 samples  
## 57 predictor  
## 2 classes: 'nonspam', 'spam'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 2075, 2074, 2075, 2075, 2074, 2074, ...   
## Resampling results across tuning parameters:  
##   
## C Accuracy Kappa   
## 0.25 0.9066403 0.7992873  
## 0.50 0.9172291 0.8232745  
## 1.00 0.9256458 0.8419957  
## 2.00 0.9251233 0.8410188  
## 4.00 0.9246896 0.8403207  
## 8.00 0.9229554 0.8366363  
## 16.00 0.9226945 0.8362892  
## 32.00 0.9194850 0.8293680  
## 64.00 0.9192249 0.8288815  
## 128.00 0.9135818 0.8172027  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.03004475  
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were sigma = 0.03004475 and C = 1.

plot (svmR.spam.radial)



plot (svmR.spam.radial, metric="Kappa")



radial.predict <- predict(svmR.spam.radial, test.Spam, "raw")  
 confusionMatrix(radial.predict, test.Spam$type)

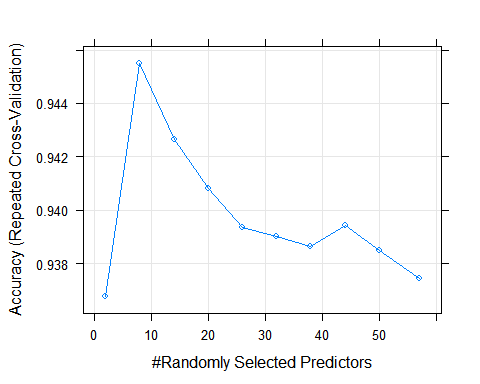
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction nonspam spam  
## nonspam 1316 113  
## spam 67 800  
##   
## Accuracy : 0.9216   
## 95% CI : (0.9098, 0.9323)  
## No Information Rate : 0.6024   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8349   
## Mcnemar's Test P-Value : 0.0007962   
##   
## Sensitivity : 0.9516   
## Specificity : 0.8762   
## Pos Pred Value : 0.9209   
## Neg Pred Value : 0.9227   
## Prevalence : 0.6024   
## Detection Rate : 0.5732   
## Detection Prevalence : 0.6224   
## Balanced Accuracy : 0.9139   
##   
## 'Positive' Class : nonspam   
##

## 6. Random Forest (ten fold cross-validation)

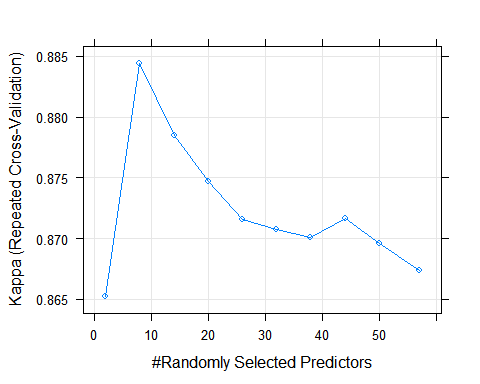
set.seed(cte.seed)  
 rf.spam.ctrl <- trainControl(method="repeatedcv", repeats=5)  
 rf.spam <- train (type ~ ., data = train.Spam, method = "rf", tuneLength = 10, trControl = rf.spam.ctrl)  
 rf.spam

## Random Forest   
##   
## 2305 samples  
## 57 predictor  
## 2 classes: 'nonspam', 'spam'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 2075, 2074, 2075, 2075, 2074, 2074, ...   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.9367514 0.8652074  
## 8 0.9455133 0.8844431  
## 14 0.9426516 0.8784888  
## 20 0.9408304 0.8747158  
## 26 0.9393518 0.8715390  
## 32 0.9390070 0.8707457  
## 38 0.9386588 0.8700694  
## 44 0.9394414 0.8716820  
## 50 0.9384845 0.8696210  
## 57 0.9374444 0.8673816  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 8.

plot (rf.spam)



plot (rf.spam, metric="Kappa")



predictions.randomforest <- predict(rf.spam, test.Spam, "raw")  
confusionMatrix(predictions.randomforest, test.Spam$type)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction nonspam spam  
## nonspam 1338 62  
## spam 45 851  
##   
## Accuracy : 0.9534   
## 95% CI : (0.944, 0.9617)  
## No Information Rate : 0.6024   
## P-Value [Acc > NIR] : <2e-16   
##   
## Kappa : 0.9024   
## Mcnemar's Test P-Value : 0.1219   
##   
## Sensitivity : 0.9675   
## Specificity : 0.9321   
## Pos Pred Value : 0.9557   
## Neg Pred Value : 0.9498   
## Prevalence : 0.6024   
## Detection Rate : 0.5828   
## Detection Prevalence : 0.6098   
## Balanced Accuracy : 0.9498   
##   
## 'Positive' Class : nonspam   
##

## 7. Proyectar Datos (PCA)

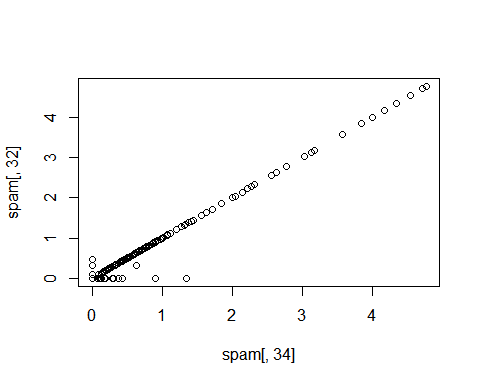
set.seed (cte.seed)  
M <- abs(cor(train.Spam[,-58]))  
  
diag(M) <- 0  
  
#selección alta correlación  
which(M > 0.8, arr.ind=TRUE)

## row col  
## num857 32 31  
## num415 34 31  
## telnet 31 32  
## num415 34 32  
## direct 40 32  
## telnet 31 34  
## num857 32 34  
## direct 40 34  
## num857 32 40  
## num415 34 40

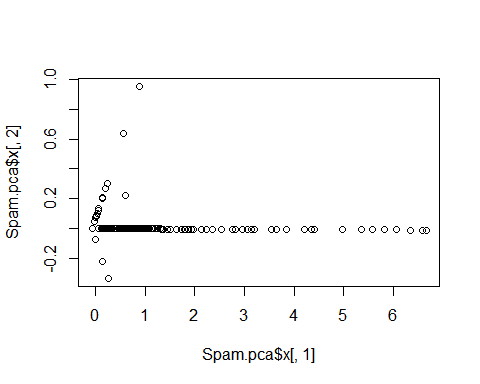
names(spam)[c(34,32)]

## [1] "num415" "num857"

plot(spam[,34], spam[,32])



# PCA  
Spam.Reducido <- spam[,c(34,32)]  
Spam.pca <- prcomp(Spam.Reducido)  
plot(Spam.pca$x[,1], Spam.pca$x[,2])



Spam.pca$rotation

## PC1 PC2  
## num415 0.7080625 0.7061498  
## num857 0.7061498 -0.7080625

## 8. Comparar los diferentes modelos

models <- list(Knn.spam.ctrl, svm.spam.lineal, svmR.spam.radial, rf.spam)  
  
cv.samples <- resamples(models)  
summary( cv.samples)

##   
## Call:  
## summary.resamples(object = cv.samples)  
##   
## Models: Model1, Model2, Model3, Model4   
## Number of resamples: 50   
##   
## Accuracy   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## Model1 0.8478 0.8842 0.9000 0.8967 0.9077 0.9437 0  
## Model2 0.8788 0.9087 0.9221 0.9219 0.9350 0.9524 0  
## Model3 0.8745 0.9174 0.9241 0.9256 0.9351 0.9652 0  
## Model4 0.9174 0.9348 0.9436 0.9455 0.9565 0.9740 0  
##   
## Kappa   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## Model1 0.6786 0.7562 0.7867 0.7809 0.8070 0.8800 0  
## Model2 0.7399 0.8066 0.8352 0.8347 0.8633 0.8998 0  
## Model3 0.7301 0.8239 0.8391 0.8420 0.8639 0.9264 0  
## Model4 0.8246 0.8613 0.8802 0.8844 0.9082 0.9450 0

# Visualización resultados comparación  
  
dotplot( cv.samples)

