AprendizajeSupervisadoObligatoria\_EGT

Elena Garcia Torres

8 de enero de 2017

# INDICE

* 1. Enunciado
  2. Carga y preparación de datos de sonar
  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. Voluntario: Repetir los ejercicios anteriores pero utilizando como medida de error el área bajo la curva ROC
  8. Voluntario: Comparar los diferentes modelos y hacer una recomendación, si es posible, para esta aplicación.

## 1. Enunciado

1. Obligatoria: Para los datos de Sonar proporcionados por la librería mlbench realizar las siguientes tareas:

* 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.
* 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.
* Repetir el proceso anterior pero utilizando kernel no lineal RBF. Obtener los parámetros óptimos. ¿Qué podrías 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.
* Voluntario: Repetir los ejercicios anteriores pero utilizando como medida de error el área bajo la curva ROC. Comentar los resultados.
* Voluntario: Comparar los diferentes modelos y hacer una recomendación, si es posible, para esta aplicación.

## 2. Carga y preparación de datos de sonar

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

libs <- c("ggplot2","lattice","caret", "dplyr", "mlbench", "ipred","knitr", "kernlab", "randomForest", "rpart")  
   
 for (i in libs){  
 print(i)  
 if(!require(i, character.only = TRUE)) { install.packages(i); library(i) }  
 }

## [1] "ggplot2"

## Loading required package: ggplot2

## [1] "lattice"

## Loading required package: lattice

## [1] "caret"

## Loading required package: caret

## [1] "dplyr"

## Loading required package: dplyr

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

## [1] "mlbench"

## Loading required package: mlbench

## [1] "ipred"

## Loading required package: ipred

## [1] "knitr"

## Loading required package: knitr

## [1] "kernlab"

## Loading required package: kernlab

##   
## Attaching package: 'kernlab'

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

## [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:dplyr':  
##   
## combine

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

## [1] "rpart"

## Loading required package: rpart

```r  
# se asigna seed para no tener valores aleatorios  
cte.seed <- 1234  
  
# se cargan los datos de "Sonar"  
data(Sonar)  
Sonar[1:2, ]  
```  
  
```  
## V1 V2 V3 V4 V5 V6 V7 V8 V9 V10  
## 1 0.0200 0.0371 0.0428 0.0207 0.0954 0.0986 0.1539 0.1601 0.3109 0.2111  
## 2 0.0453 0.0523 0.0843 0.0689 0.1183 0.2583 0.2156 0.3481 0.3337 0.2872  
## V11 V12 V13 V14 V15 V16 V17 V18 V19 V20  
## 1 0.1609 0.1582 0.2238 0.0645 0.0660 0.2273 0.31 0.2999 0.5078 0.4797  
## 2 0.4918 0.6552 0.6919 0.7797 0.7464 0.9444 1.00 0.8874 0.8024 0.7818  
## V21 V22 V23 V24 V25 V26 V27 V28 V29 V30  
## 1 0.5783 0.5071 0.4328 0.5550 0.6711 0.6415 0.7104 0.8080 0.6791 0.3857  
## 2 0.5212 0.4052 0.3957 0.3914 0.3250 0.3200 0.3271 0.2767 0.4423 0.2028  
## V31 V32 V33 V34 V35 V36 V37 V38 V39 V40  
## 1 0.1307 0.2604 0.5121 0.7547 0.8537 0.8507 0.6692 0.6097 0.4943 0.2744  
## 2 0.3788 0.2947 0.1984 0.2341 0.1306 0.4182 0.3835 0.1057 0.1840 0.1970  
## V41 V42 V43 V44 V45 V46 V47 V48 V49 V50  
## 1 0.0510 0.2834 0.2825 0.4256 0.2641 0.1386 0.1051 0.1343 0.0383 0.0324  
## 2 0.1674 0.0583 0.1401 0.1628 0.0621 0.0203 0.0530 0.0742 0.0409 0.0061  
## V51 V52 V53 V54 V55 V56 V57 V58 V59 V60  
## 1 0.0232 0.0027 0.0065 0.0159 0.0072 0.0167 0.018 0.0084 0.0090 0.0032  
## 2 0.0125 0.0084 0.0089 0.0048 0.0094 0.0191 0.014 0.0049 0.0052 0.0044  
## Class  
## 1 R  
## 2 R  
```  
  
```r  
# Se crean las particiones de entrenamiento y test  
index.train.sonar <- createDataPartition(Sonar[,61], p=0.8, list=F)  
  
train.sonar <- Sonar[index.train.sonar,]  
train.sonar [1:2,]  
```  
  
```  
## V1 V2 V3 V4 V5 V6 V7 V8 V9 V10  
## 1 0.0200 0.0371 0.0428 0.0207 0.0954 0.0986 0.1539 0.1601 0.3109 0.2111  
## 2 0.0453 0.0523 0.0843 0.0689 0.1183 0.2583 0.2156 0.3481 0.3337 0.2872  
## V11 V12 V13 V14 V15 V16 V17 V18 V19 V20  
## 1 0.1609 0.1582 0.2238 0.0645 0.0660 0.2273 0.31 0.2999 0.5078 0.4797  
## 2 0.4918 0.6552 0.6919 0.7797 0.7464 0.9444 1.00 0.8874 0.8024 0.7818  
## V21 V22 V23 V24 V25 V26 V27 V28 V29 V30  
## 1 0.5783 0.5071 0.4328 0.5550 0.6711 0.6415 0.7104 0.8080 0.6791 0.3857  
## 2 0.5212 0.4052 0.3957 0.3914 0.3250 0.3200 0.3271 0.2767 0.4423 0.2028  
## V31 V32 V33 V34 V35 V36 V37 V38 V39 V40  
## 1 0.1307 0.2604 0.5121 0.7547 0.8537 0.8507 0.6692 0.6097 0.4943 0.2744  
## 2 0.3788 0.2947 0.1984 0.2341 0.1306 0.4182 0.3835 0.1057 0.1840 0.1970  
## V41 V42 V43 V44 V45 V46 V47 V48 V49 V50  
## 1 0.0510 0.2834 0.2825 0.4256 0.2641 0.1386 0.1051 0.1343 0.0383 0.0324  
## 2 0.1674 0.0583 0.1401 0.1628 0.0621 0.0203 0.0530 0.0742 0.0409 0.0061  
## V51 V52 V53 V54 V55 V56 V57 V58 V59 V60  
## 1 0.0232 0.0027 0.0065 0.0159 0.0072 0.0167 0.018 0.0084 0.0090 0.0032  
## 2 0.0125 0.0084 0.0089 0.0048 0.0094 0.0191 0.014 0.0049 0.0052 0.0044  
## Class  
## 1 R  
## 2 R  
```  
  
```r  
test.sonar <- Sonar[ -index.train.sonar, ]  
test.sonar [1:2,]  
```  
  
```  
## V1 V2 V3 V4 V5 V6 V7 V8 V9 V10  
## 11 0.0039 0.0063 0.0152 0.0336 0.0310 0.0284 0.0396 0.0272 0.0323 0.0452  
## 17 0.0352 0.0116 0.0191 0.0469 0.0737 0.1185 0.1683 0.1541 0.1466 0.2912  
## V11 V12 V13 V14 V15 V16 V17 V18 V19 V20  
## 11 0.0492 0.0996 0.1424 0.1194 0.0628 0.0907 0.1177 0.1429 0.1223 0.1104  
## 17 0.2328 0.2237 0.2470 0.1560 0.3491 0.3308 0.2299 0.2203 0.2493 0.4128  
## V21 V22 V23 V24 V25 V26 V27 V28 V29 V30  
## 11 0.1847 0.3715 0.4382 0.5707 0.6654 0.7476 0.7654 0.8555 0.9720 0.9221  
## 17 0.3158 0.6191 0.5854 0.3395 0.2561 0.5599 0.8145 0.6941 0.6985 0.8660  
## V31 V32 V33 V34 V35 V36 V37 V38 V39 V40  
## 11 0.7502 0.7209 0.7757 0.6055 0.5021 0.4499 0.3947 0.4281 0.4427 0.3749  
## 17 0.5930 0.3664 0.6750 0.8697 0.7837 0.7552 0.5789 0.4713 0.1252 0.6087  
## V41 V42 V43 V44 V45 V46 V47 V48 V49 V50  
## 11 0.1972 0.0511 0.0793 0.1269 0.1533 0.0690 0.0402 0.0534 0.0228 0.0073  
## 17 0.7322 0.5977 0.3431 0.1803 0.2378 0.3424 0.2303 0.0689 0.0216 0.0469  
## V51 V52 V53 V54 V55 V56 V57 V58 V59 V60  
## 11 0.0062 0.0062 0.0120 0.0052 0.0056 0.0093 0.0042 0.0003 0.0053 0.0036  
## 17 0.0426 0.0346 0.0158 0.0154 0.0109 0.0048 0.0095 0.0015 0.0073 0.0067  
## Class  
## 11 R  
## 17 R  
```  
  
```r  
# Se comprueba si las dimensiones son congruentes  
dim(index.train.sonar)  
```  
  
```  
## [1] 167 1  
```  
  
```r  
dim(train.sonar)  
```  
  
```  
## [1] 167 61  
```  
  
```r  
dim(test.sonar)  
```  
  
```  
## [1] 41 61  
```  
  
```r  
# Las proporciones se mantienen  
prop.table( table(Sonar[,61]))  
```  
  
```  
##   
## M R   
## 0.5336538 0.4663462  
```  
  
```r  
prop.table( table(train.sonar[,61]))  
```  
  
```  
##   
## M R   
## 0.5329341 0.4670659  
```  
  
```r  
prop.table( table(test.sonar[,61]))  
```  
  
```  
##   
## M R   
## 0.5365854 0.4634146  
```

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

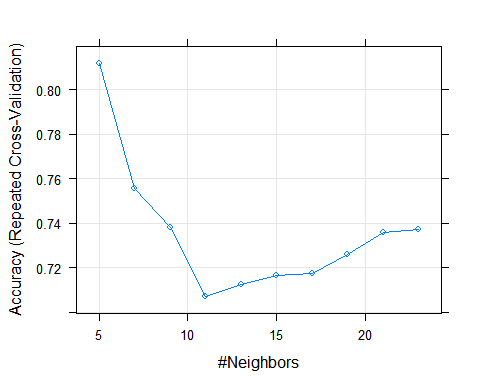
set.seed (cte.seed)  
   
 # se usa la funcion "train" para construir el modelo K-NN  
 Knn.sonar <- train(Class ~ ., data = train.sonar, method = "knn", tuneLength = 10, preProc = c("center", "scale"))  
   
 Knn.sonar

## k-Nearest Neighbors   
##   
## 167 samples  
## 60 predictor  
## 2 classes: 'M', 'R'   
##   
## Pre-processing: centered (60), scaled (60)   
## Resampling: Bootstrapped (25 reps)   
## Summary of sample sizes: 167, 167, 167, 167, 167, 167, ...   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.7594684 0.5104170  
## 7 0.7480431 0.4867395  
## 9 0.7254028 0.4405720  
## 11 0.7223504 0.4332522  
## 13 0.7208706 0.4290620  
## 15 0.7210784 0.4293315  
## 17 0.7114609 0.4125689  
## 19 0.7050547 0.4005126  
## 21 0.7066189 0.4052943  
## 23 0.7024637 0.3983823  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 5.

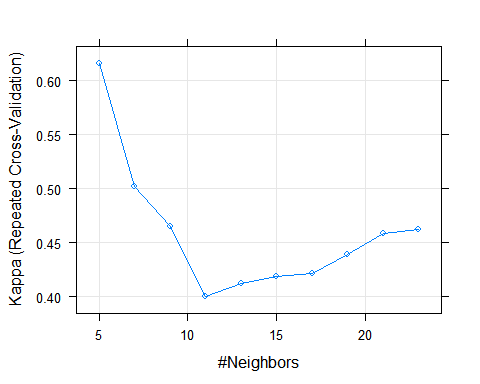
# Variacion cruzada, trainControl  
 ctrl <- trainControl(method = "repeatedcv", repeats = 5)  
 Knn.sonar.ctrl <- train(Class ~ ., data = train.sonar, method = "knn", tuneLength = 10, trControl = ctrl, preProc = c("center", "scale"))  
 Knn.sonar.ctrl

## k-Nearest Neighbors   
##   
## 167 samples  
## 60 predictor  
## 2 classes: 'M', 'R'   
##   
## Pre-processing: centered (60), scaled (60)   
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 150, 151, 150, 150, 151, 150, ...   
## Resampling results across tuning parameters:  
##   
## k Accuracy Kappa   
## 5 0.8119118 0.6165218  
## 7 0.7556618 0.5021452  
## 9 0.7378676 0.4648962  
## 11 0.7069118 0.3997987  
## 13 0.7127206 0.4118274  
## 15 0.7165441 0.4191198  
## 17 0.7175735 0.4218878  
## 19 0.7261765 0.4393398  
## 21 0.7357353 0.4586197  
## 23 0.7371324 0.4622339  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was k = 5.

plot(Knn.sonar.ctrl)



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



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

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction M R  
## M 20 6  
## R 2 13  
##   
## Accuracy : 0.8049   
## 95% CI : (0.6513, 0.9118)  
## No Information Rate : 0.5366   
## P-Value [Acc > NIR] : 0.0003284   
##   
## Kappa : 0.6019   
## Mcnemar's Test P-Value : 0.2888444   
##   
## Sensitivity : 0.9091   
## Specificity : 0.6842   
## Pos Pred Value : 0.7692   
## Neg Pred Value : 0.8667   
## Prevalence : 0.5366   
## Detection Rate : 0.4878   
## Detection Prevalence : 0.6341   
## Balanced Accuracy : 0.7967   
##   
## 'Positive' Class : M   
##

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

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

## Support Vector Machines with Linear Kernel   
##   
## 167 samples  
## 60 predictor  
## 2 classes: 'M', 'R'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 150, 150, 152, 150, 151, 150, ...   
## Resampling results:  
##   
## Accuracy Kappa   
## 0.7281765 0.451493  
##   
## Tuning parameter 'C' was held constant at a value of 1  
##

svm.sonar.lineal.predict <- predict(svm.sonar.lineal, test.sonar, "raw")  
   
 confusionMatrix(svm.sonar.lineal.predict, test.sonar$Class)

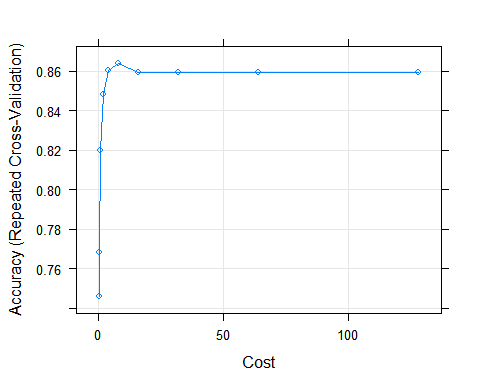
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction M R  
## M 17 4  
## R 5 15  
##   
## Accuracy : 0.7805   
## 95% CI : (0.6239, 0.8944)  
## No Information Rate : 0.5366   
## P-Value [Acc > NIR] : 0.001099   
##   
## Kappa : 0.5602   
## Mcnemar's Test P-Value : 1.000000   
##   
## Sensitivity : 0.7727   
## Specificity : 0.7895   
## Pos Pred Value : 0.8095   
## Neg Pred Value : 0.7500   
## Prevalence : 0.5366   
## Detection Rate : 0.4146   
## Detection Prevalence : 0.5122   
## Balanced Accuracy : 0.7811   
##   
## 'Positive' Class : M   
##

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

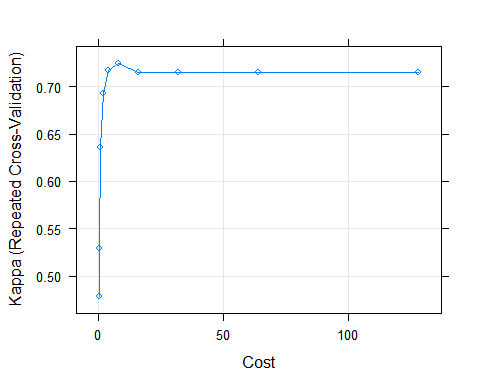
set.seed(cte.seed)  
 svmR.sonar.ctrl <- trainControl(method="repeatedcv", repeats=5)  
 svmR.sonar.radial <- train (Class ~ ., data = train.sonar, method = "svmRadial", tuneLength = 10, trControl = svmR.sonar.ctrl)  
 svmR.sonar.radial

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 167 samples  
## 60 predictor  
## 2 classes: 'M', 'R'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 150, 150, 152, 150, 151, 150, ...   
## Resampling results across tuning parameters:  
##   
## C Accuracy Kappa   
## 0.25 0.7456127 0.4778475  
## 0.50 0.7680490 0.5293159  
## 1.00 0.8196667 0.6356146  
## 2.00 0.8482696 0.6929486  
## 4.00 0.8604755 0.7171849  
## 8.00 0.8642353 0.7253001  
## 16.00 0.8594559 0.7157036  
## 32.00 0.8594559 0.7157036  
## 64.00 0.8594559 0.7157036  
## 128.00 0.8594559 0.7157036  
##   
## Tuning parameter 'sigma' was held constant at a value of 0.01285491  
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were sigma = 0.01285491 and C = 8.

plot (svmR.sonar.radial)



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



radial.predict <- predict(svmR.sonar.radial, test.sonar, "raw")  
confusionMatrix(radial.predict, test.sonar$Class)

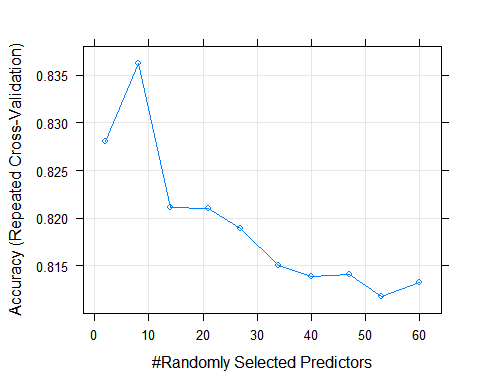
## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction M R  
## M 21 3  
## R 1 16  
##   
## Accuracy : 0.9024   
## 95% CI : (0.7687, 0.9728)  
## No Information Rate : 0.5366   
## P-Value [Acc > NIR] : 5.253e-07   
##   
## Kappa : 0.8024   
## Mcnemar's Test P-Value : 0.6171   
##   
## Sensitivity : 0.9545   
## Specificity : 0.8421   
## Pos Pred Value : 0.8750   
## Neg Pred Value : 0.9412   
## Prevalence : 0.5366   
## Detection Rate : 0.5122   
## Detection Prevalence : 0.5854   
## Balanced Accuracy : 0.8983   
##   
## 'Positive' Class : M   
##

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

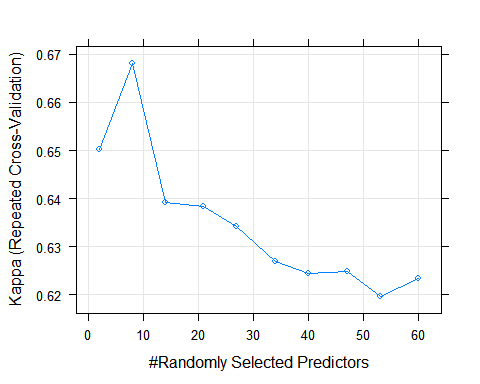
set.seed(cte.seed)  
 rf.sonar.ctrl <- trainControl(method="repeatedcv", repeats=5)  
 rf.sonar <- train (Class ~ ., data = train.sonar, method = "rf", tuneLength = 10, trControl = rf.sonar.ctrl)  
 rf.sonar

## Random Forest   
##   
## 167 samples  
## 60 predictor  
## 2 classes: 'M', 'R'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 150, 150, 152, 150, 151, 150, ...   
## Resampling results across tuning parameters:  
##   
## mtry Accuracy Kappa   
## 2 0.8280490 0.6501788  
## 8 0.8362843 0.6682017  
## 14 0.8211373 0.6391635  
## 21 0.8209902 0.6383283  
## 27 0.8189314 0.6342528  
## 34 0.8150343 0.6268634  
## 40 0.8138578 0.6243908  
## 47 0.8140784 0.6248547  
## 53 0.8117255 0.6195261  
## 60 0.8132794 0.6233609  
##   
## Accuracy was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 8.

plot (rf.sonar)



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



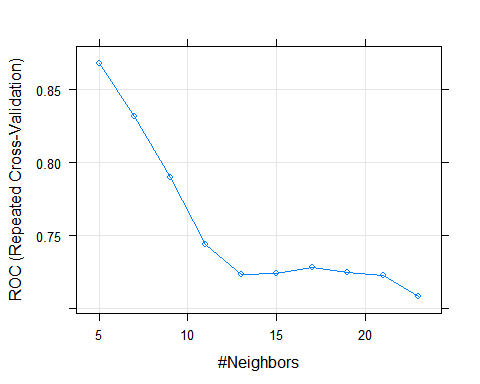
predictions.randomforest <- predict(rf.sonar, test.sonar, "raw")  
confusionMatrix(predictions.randomforest, test.sonar$Class)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction M R  
## M 18 5  
## R 4 14  
##   
## Accuracy : 0.7805   
## 95% CI : (0.6239, 0.8944)  
## No Information Rate : 0.5366   
## P-Value [Acc > NIR] : 0.001099   
##   
## Kappa : 0.557   
## Mcnemar's Test P-Value : 1.000000   
##   
## Sensitivity : 0.8182   
## Specificity : 0.7368   
## Pos Pred Value : 0.7826   
## Neg Pred Value : 0.7778   
## Prevalence : 0.5366   
## Detection Rate : 0.4390   
## Detection Prevalence : 0.5610   
## Balanced Accuracy : 0.7775   
##   
## 'Positive' Class : M   
##

## 7. Voluntario: Repetir los ejercicios anteriores pero utilizando como medida de error el área bajo la curva ROC.

### Si queremos seleccionar el modelo utilizando las curvas ROC ### 7.1. K-NN

set.seed (cte.seed)  
knn.sonar.control.roc <- trainControl(method="repeatedcv", repeats=5, classProbs=T, summaryFunction= twoClassSummary )  
knn.sonar.roc <- train(Class ~ ., data = train.sonar, method = "knn", tuneLength = 10, trControl=knn.sonar.control.roc, metric="ROC")  
plot(knn.sonar.roc)



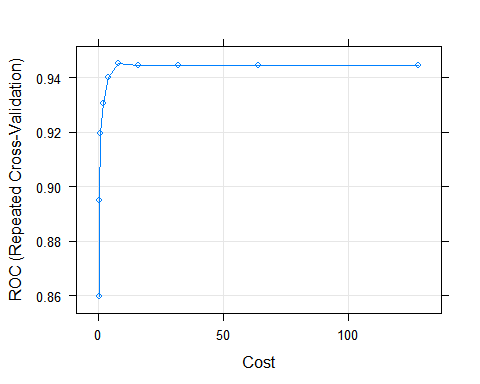
### 7.2. C-SVM Kernel Lineal

set.seed (cte.seed)  
svm.sonar.control.roc <- trainControl(method="repeatedcv", repeats=5, classProbs=T, summaryFunction= twoClassSummary )  
svm.sonar.roc <- train(Class ~ ., data = train.sonar, method = "svmLinear", tuneLength = 10, trControl=svm.sonar.control.roc, metric="ROC")  
svm.sonar.roc

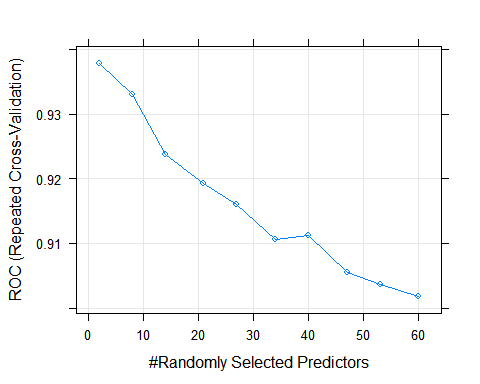
## Support Vector Machines with Linear Kernel   
##   
## 167 samples  
## 60 predictor  
## 2 classes: 'M', 'R'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold, repeated 5 times)   
## Summary of sample sizes: 150, 150, 152, 150, 151, 150, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.8069395 0.7338889 0.7067857  
##   
## Tuning parameter 'C' was held constant at a value of 1  
##

### 7.3. C-SVM Kernel No Lineal

set.seed (cte.seed)  
svmR.sonar.control.roc <- trainControl(method="repeatedcv", repeats=5, classProbs=T, summaryFunction= twoClassSummary )  
svmR.sonar.roc <- train(Class ~ ., data = train.sonar, method = "svmRadial", tuneLength = 10, trControl=svmR.sonar.control.roc, metric="ROC")  
plot (svmR.sonar.roc)

 ### 7.4. Random Forest

set.seed (cte.seed)  
rf.sonar.control.roc <- trainControl(method="repeatedcv", repeats=5, classProbs=T, summaryFunction= twoClassSummary )  
rf.sonar.roc <- train(Class ~ ., data = train.sonar, method = "rf", tuneLength = 10, trControl=rf.sonar.control.roc, metric="ROC")  
plot (rf.sonar.roc)

 ## 8. Voluntario: Comparar los diferentes modelos y hacer una recomendación, si es posible, para esta aplicación.

models <- list(Knn.sonar.ctrl, svm.sonar.lineal, svmR.sonar.radial, rf.sonar)  
  
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 0.5294 0.7059 0.8235 0.8119 0.8824 1.0000 0  
## Model2 0.5000 0.6572 0.7279 0.7282 0.8208 0.9412 0  
## Model3 0.6250 0.8235 0.8787 0.8642 0.9375 1.0000 0  
## Model4 0.6875 0.7647 0.8235 0.8363 0.8824 1.0000 0  
##   
## Kappa   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## Model1 0.02857 0.3993 0.6383 0.6165 0.7606 1.0000 0  
## Model2 0.00000 0.3218 0.4486 0.4515 0.6350 0.8811 0  
## Model3 0.25000 0.6383 0.7553 0.7253 0.8750 1.0000 0  
## Model4 0.33330 0.5294 0.6434 0.6682 0.7639 1.0000 0

# Visualización resultados comparación  
  
dotplot( cv.samples)

