Predictive Modeling with the caret Package

for Exercises

Eszter Katalin Bognar

02-06-2020

Exercises

Load the dataset, split into train and test set

```
# use the caret package
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
# load the used segmentationData dataset
data(segmentationData)

# get rid of the cell identifier
?segmentationData$Cell <- NULL
```

```
## starting httpd help server ...
```

```
##  done
```

```
#split the dataset into training and test set using the value of the Case column (Train or Test)
training <- subset(segmentationData, Case == "Train")
testing <- subset(segmentationData, Case == "Test")

# get rid of the Case column
training$Case <- NULL
testing$Case <- NULL

# check the first 6 columns in the training dataset
str(training[,1:6])
```

```
## 'data.frame':    1009 obs. of  6 variables:
##  $ Cell       : int  207932307 207932463 207932470 207932484 207932459 207827779 207827784 207827645
##  $ Class      : Factor w/ 2 levels "PS","WS": 1 2 1 2 1 1 1 2 2 2 ...
##  $ AngleCh1   : num  133.8 106.6 69.2 109.4 104.3 ...
##  $ AreaCh1    : int  819 431 298 256 258 358 158 315 246 223 ...
##  $ AvgIntenCh1: num  31.9 28 19.5 18.8 17.6 ...
##  $ AvgIntenCh2: num  207 116 102 127 125 ...
```

Preprocess the dataset

```
# Since channel 1 is the cell body, AreaCh1 measures the size of the cell.
# First, estimate the standardization parameters:
# take the dataset without the Class column as trainX
trainX <- training[, names(training) != "Class"]
# Methods are "BoxCox", "YeoJohnson", center", "scale",
# "range", "knnImpute", "bagImpute", "pca", "ica" and
# "spatialSign"
# preprocess the training set with standardization (centring and scaling)
preProcValues <- preProcess(trainX, method = c("center", "scale"))
preProcValues
```

```
## Created from 1009 samples and 59 variables
##
## Pre-processing:
##   - centered (59)
##   - ignored (0)
##   - scaled (59)
```

```
# Apply them to the data sets:
scaledTrain <- predict(preProcValues, trainX)
```

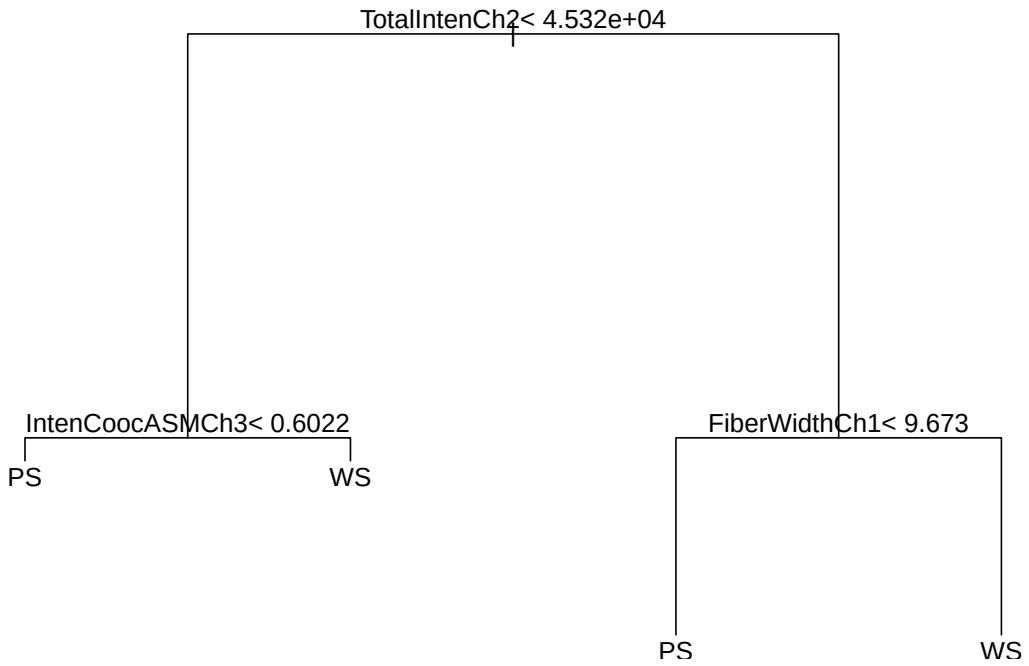
CART: Classification and regression trees

create and visualize the model

```
library(rpart)
# applying an rpart model on the training set with maximum 2 depth
rpart1 <- rpart(Class ~ ., data = training,
control = rpart.control(maxdepth = 2))
rpart1
```

```
## n= 1009
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 1009 373 PS (0.63032706 0.36967294)
##   2) TotalIntenCh2< 45324.5 454  34 PS (0.92511013 0.07488987)
##     4) IntenCoocASMCh3< 0.6021832 447  27 PS (0.93959732 0.06040268) *
##     5) IntenCoocASMCh3>=0.6021832 7   0 WS (0.00000000 1.00000000) *
##   3) TotalIntenCh2>=45324.5 555 216 WS (0.38918919 0.61081081)
##     6) FiberWidthCh1< 9.673245 154  47 PS (0.69480519 0.30519481) *
##     7) FiberWidthCh1>=9.673245 401 109 WS (0.27182045 0.72817955) *
```

```
# plot rpart the model
plot(rpart1)
# display the model in text format
text(rpart1)
```



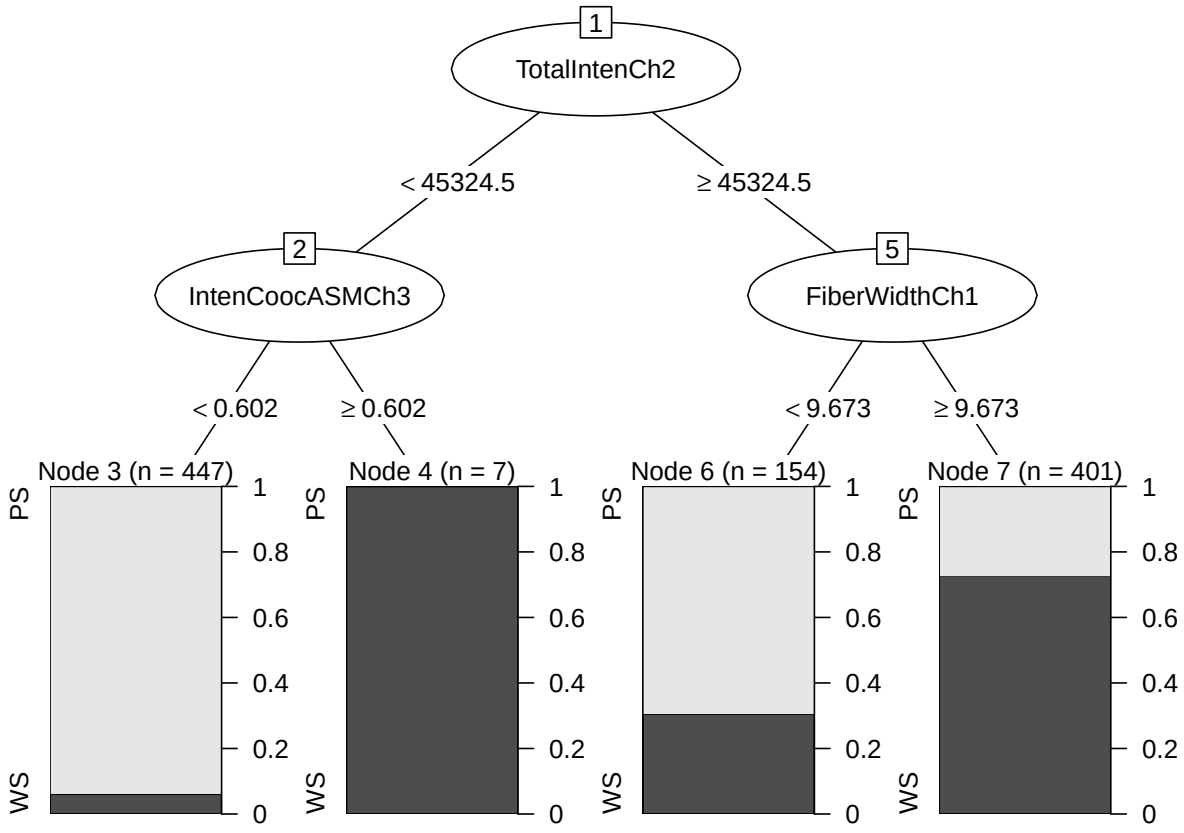

```
# load the partykis library
library(partykit)
```

```
## Loading required package: grid
```

```
## Loading required package: libcoin
```

```
## Loading required package: mvtnorm
```

```
# use the partykit library to visualize the tree structured regression or classification model
rpart1a <- as.party(rpart1)
# plot the model - we can see the partykit package gives us a more detailed model visualization than rp
plot(rpart1a)
```

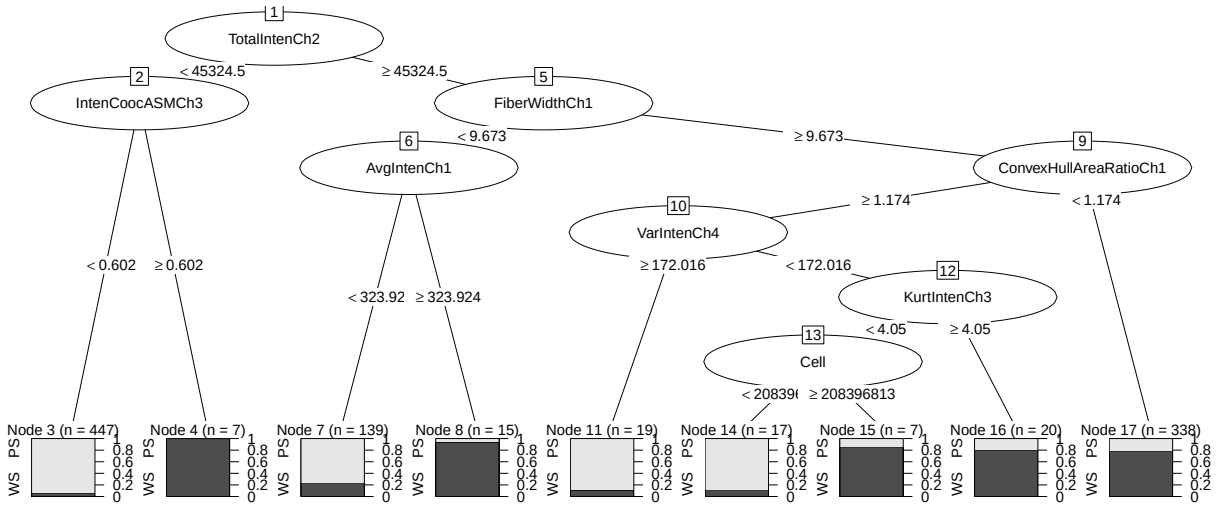



```
# Full tree without any control parameter:
rpartFull <- rpart(Class ~ ., data = training)
# display the model in text format
rpartFull
```

```
## n= 1009
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1009 373 PS (0.63032706 0.36967294)
##     2) TotalIntenCh2< 45324.5 454  34 PS (0.92511013 0.07488987)
##       4) IntenCoocASMCh3< 0.6021832 447  27 PS (0.93959732 0.06040268) *
##       5) IntenCoocASMCh3>=0.6021832 7   0 WS (0.00000000 1.00000000) *
##     3) TotalIntenCh2>=45324.5 555 216 WS (0.38918919 0.61081081)
##       6) FiberWidthCh1< 9.673245 154  47 PS (0.69480519 0.30519481)
##        12) AvgIntenCh1< 323.9243 139  33 PS (0.76258993 0.23741007) *
##        13) AvgIntenCh1>=323.9243 15   1 WS (0.06666667 0.93333333) *
##       7) FiberWidthCh1>=9.673245 401 109 WS (0.27182045 0.72817955)
##        14) ConvexHullAreaRatioCh1>=1.173618 63  26 PS (0.58730159 0.41269841)
##          28) VarIntenCh4>=172.0165 19   2 PS (0.89473684 0.10526316) *
##          29) VarIntenCh4< 172.0165 44  20 WS (0.45454545 0.54545455)
##            58) KurtIntenCh3< 4.05017 24   8 PS (0.66666667 0.33333333)
```

```
##             116) Cell< 2.083968e+08 17   2 PS (0.88235294 0.11764706) *
##             117) Cell>=2.083968e+08 7   1 WS (0.14285714 0.85714286) *
##           59) KurtIntenCh3>=4.05017 20   4 WS (0.20000000 0.80000000) *
##       15) ConvexHullAreaRatioCh1< 1.173618 338  72 WS (0.21301775 0.78698225) *
```

```
# Plot the Full tree
plot(as.party(rpartFull))
```



Apply the model and evaluate model performance

```
# make prediction on the testing set
rpartPred <- predict(rpartFull, testing, type = "class")
# display the confusion matrix to evaluate model performance
confusionMatrix(rpartPred, testing$Class) # requires 2 factor vectors
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  PS  WS
##         PS 532  82
##         WS 132 264
##
##                Accuracy : 0.7881
##                  95% CI : (0.7616, 0.8129)
##     No Information Rate : 0.6574
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5453
##
##  Mcnemar's Test P-Value : 0.0008094
##
##             Sensitivity : 0.8012
##             Specificity : 0.7630
```

```
##          Pos Pred Value : 0.8664
##          Neg Pred Value : 0.6667
##              Prevalence : 0.6574
##          Detection Rate : 0.5267
##    Detection Prevalence : 0.6079
##       Balanced Accuracy : 0.7821
##
##        'Positive' Class : PS
##
```

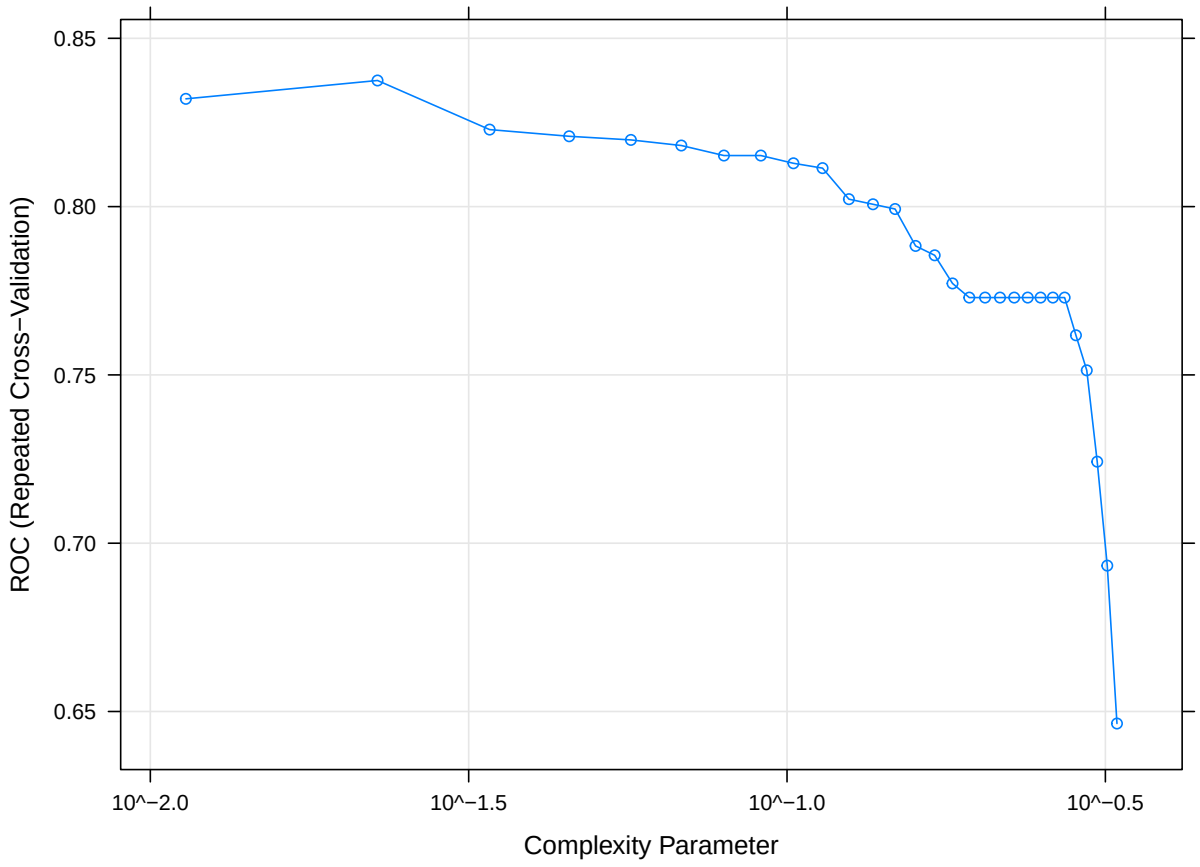
Tuning the model

```
# apply 10 fold cross-validation with 3 repeats
# for the rpart model we tune the C(p) complexity parameter
cvCtrl <- trainControl(method = "repeatedcv", repeats = 3,
summaryFunction = twoClassSummary,
classProbs = TRUE)
set.seed(1)
# tune the rpart model with 30 grid points using the ROC metric for evaluation criterion for optimal mo
rpartTune <- train(Class ~ ., data = training, method = "rpart",
tuneLength = 30, metric = "ROC",
trControl = cvCtrl)
rpartTune
```

```
## CART
##
## 1009 samples
##   59 predictor
##    2 classes: 'PS', 'WS'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 908, 908, 908, 908, 908, 909, ...
## Resampling results across tuning parameters:
##
##   cp          ROC        Sens       Spec
##   0.00000000  0.8419085  0.8237930  0.6838786
##   0.01137099  0.8320198  0.8375083  0.7089142
##   0.02274198  0.8374713  0.8348049  0.7329777
##   0.03411297  0.8228830  0.8122851  0.7411332
##   0.04548396  0.8208966  0.8222718  0.7125415
##   0.05685495  0.8198157  0.8342510  0.6857041
##   0.06822594  0.8181538  0.8394841  0.6588193
##   0.07959693  0.8151735  0.8253142  0.6724988
##   0.09096792  0.8151735  0.8253142  0.6724988
##   0.10233891  0.8128709  0.8111690  0.6991465
##   0.11370990  0.8114351  0.8033565  0.7135609
##   0.12508089  0.8022170  0.7506118  0.8025605
##   0.13645188  0.8006967  0.7443618  0.8079659
##   0.14782287  0.7992828  0.7354332  0.8223803
##   0.15919386  0.7883189  0.6987103  0.8553106
##   0.17056485  0.7855351  0.6866733  0.8714557
```

```
##   0.18193584  0.7771761  0.6624421  0.8865813
##   0.19330683  0.7729770  0.6503638  0.8955903
##   0.20467782  0.7729770  0.6503638  0.8955903
##   0.21604881  0.7729770  0.6503638  0.8955903
##   0.22741980  0.7729770  0.6503638  0.8955903
##   0.23879079  0.7729770  0.6503638  0.8955903
##   0.25016178  0.7729770  0.6503638  0.8955903
##   0.26153277  0.7729770  0.6503638  0.8955903
##   0.27290376  0.7729770  0.6503638  0.8955903
##   0.28427475  0.7617791  0.6613013  0.8622570
##   0.29564574  0.7513625  0.6738013  0.8289237
##   0.30701673  0.7242247  0.7141204  0.7343291
##   0.31838772  0.6933399  0.7461392  0.6405405
##   0.32975871  0.6464300  0.8029597  0.4899004
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
```

```r
# plot the ROC curve with the defined scales
plot(rpartTune, scales = list(x = list(log = 10)))
```



Apply the model and evaluate model performance aftertuning

```
# predict new data
rpartPred2 <- predict(rpartTune, testing)
# display the confusion matrix
confusionMatrix(rpartPred2, testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  PS  WS
##         PS 543 100
##         WS 121 246
##
##                Accuracy : 0.7812
##                  95% CI : (0.7544, 0.8063)
##     No Information Rate : 0.6574
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5212
##
##  Mcnemar's Test P-Value : 0.1785
##
##             Sensitivity : 0.8178
##             Specificity : 0.7110
##          Pos Pred Value : 0.8445
##          Neg Pred Value : 0.6703
##              Prevalence : 0.6574
##          Detection Rate : 0.5376
##    Detection Prevalence : 0.6366
##       Balanced Accuracy : 0.7644
##
##        'Positive' Class : PS
##
```

```
# Predict class probabilities
rpartProbs <- predict(rpartTune, testing, type = "prob")
head(rpartProbs)
```

```
##           PS         WS
## 1 0.97681159 0.02318841
## 5 0.97681159 0.02318841
## 6 0.04716981 0.95283019
## 7 0.04716981 0.95283019
## 8 0.97681159 0.02318841
## 9 0.97681159 0.02318841
```

```
# load the pROC package
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

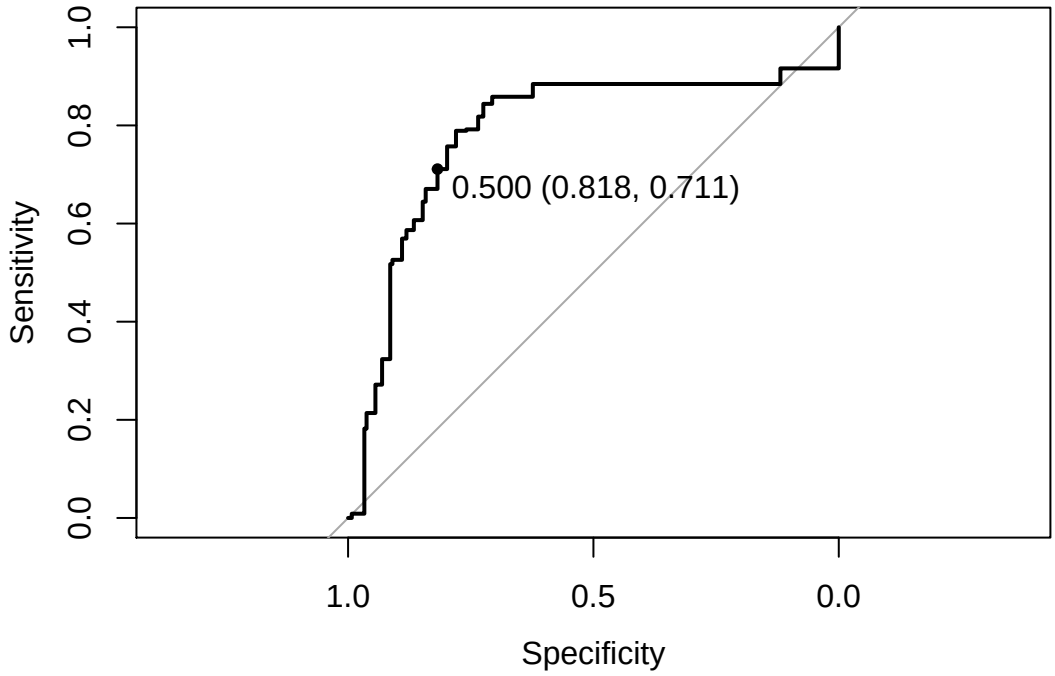
```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# creating the ROC curve
rpartROC <- roc(testing$Class, rpartProbs[, "PS"], levels = levels(testing$Class))
```

```
## Setting direction: controls > cases
```

```
# plot the ROC curve
# Setting direction: controls > cases
plot(rpartROC, type = "S", print.thres = .5)
```




```
# examine the created ROC object
rpartROC
```

```
##
## Call:
## roc.default(response = testing$Class, predictor = rpartProbs[,     "PS"], levels = levels(testing$Cla
##
## Data: rpartProbs[, "PS"] in 664 controls (testing$Class PS) > 346 cases (testing$Class WS).
## Area under the curve: 0.8068
```

SVM

split data, preprocess it and tune the svm model

```
set.seed(1)
# The default grid of cost parameters go from 2^-2,
# 0.5 to 1,
# Well fit 9 values in that sequence via the tuneLength
# argument.
svmTune <- train(x = trainX,
y = training$Class,
method = "svmRadial",
tuneLength = 9,
# add options from preProcess here too
preProc = c("center", "scale"),
metric = "ROC",
trControl = cvCtrl)
# display the tuning process
svmTune
```

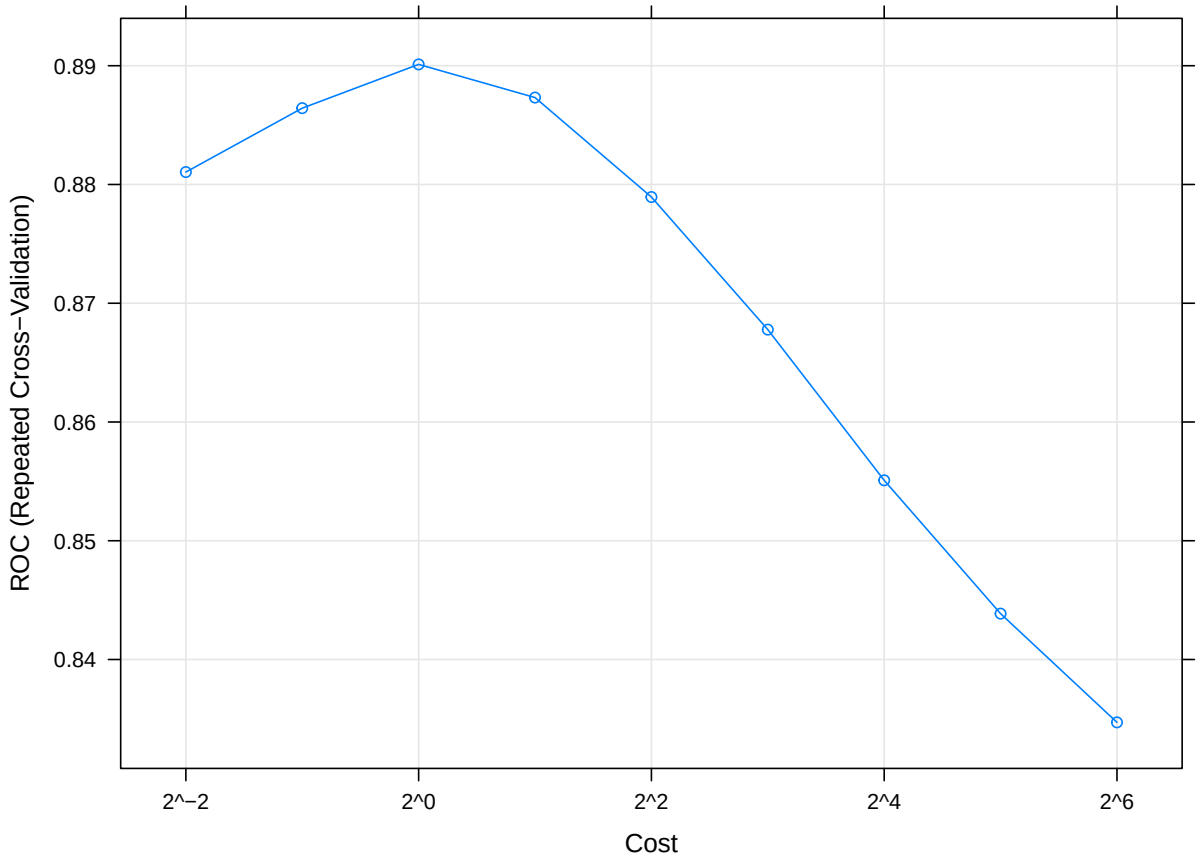
```
## Support Vector Machines with Radial Basis Function Kernel
##
## 1009 samples
##   59 predictor
##    2 classes: 'PS', 'WS'
##
## Pre-processing: centered (59), scaled (59)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 908, 908, 908, 908, 908, 909, ...
## Resampling results across tuning parameters:
##
##   C      ROC        Sens       Spec
##    0.25  0.8810451  0.8620701  0.7247511
##    0.50  0.8864256  0.8710400  0.7246799
##    1.00  0.8901121  0.8741815  0.7309388
##    2.00  0.8873190  0.8673694  0.7175202
##    4.00  0.8789408  0.8636574  0.7112376
##    8.00  0.8677775  0.8599785  0.6923898
##   16.00  0.8550875  0.8516121  0.6674490
##   32.00  0.8438621  0.8526538  0.6468943
##   64.00  0.8347070  0.8500248  0.6336178
##
## Tuning parameter 'sigma' was held constant at a value of 0.0144895
## ROC was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.0144895 and C = 1.
```

```
# display the final model
svmTune$finalModel
```

```
## Support Vector Machine object of class "ksvm"
##
## SV type: C-svc  (classification)
```

```
##  parameter : cost C = 1
##
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.0144895025565308
##
## Number of Support Vectors : 544
##
## Objective Function Value : -393.567
## Training error : 0.124876
## Probability model included.
```

```r
# plot the model
plot(svmTune, metric = "ROC", scales = list(x = list(log=2)))
```



Evaluate model performance

```r
# Make predictions on the test set:
svmPred <- predict(svmTune, testing[, names(testing) != "Class"])
# display the conusion matrix
confusionMatrix(svmPred, testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  PS  WS
##         PS 573  96
##         WS  91 250
##
##                Accuracy : 0.8149
##                  95% CI : (0.7895, 0.8384)
##     No Information Rate : 0.6574
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5875
##
##  Mcnemar's Test P-Value : 0.7699
##
##             Sensitivity : 0.8630
##             Specificity : 0.7225
##          Pos Pred Value : 0.8565
##          Neg Pred Value : 0.7331
##              Prevalence : 0.6574
##          Detection Rate : 0.5673
##    Detection Prevalence : 0.6624
##       Balanced Accuracy : 0.7927
##
##        'Positive' Class : PS
##
```

Random Forests

preprocessing and model tuning

```
set.seed(1)
# set tuning and preprocessing
rfTune <- train(x = trainX,
y = training$Class,
method = "rf",
preProc = c("center", "scale"),
metric = "ROC",
trControl = cvCtrl)
# display the tuning process
rfTune
```

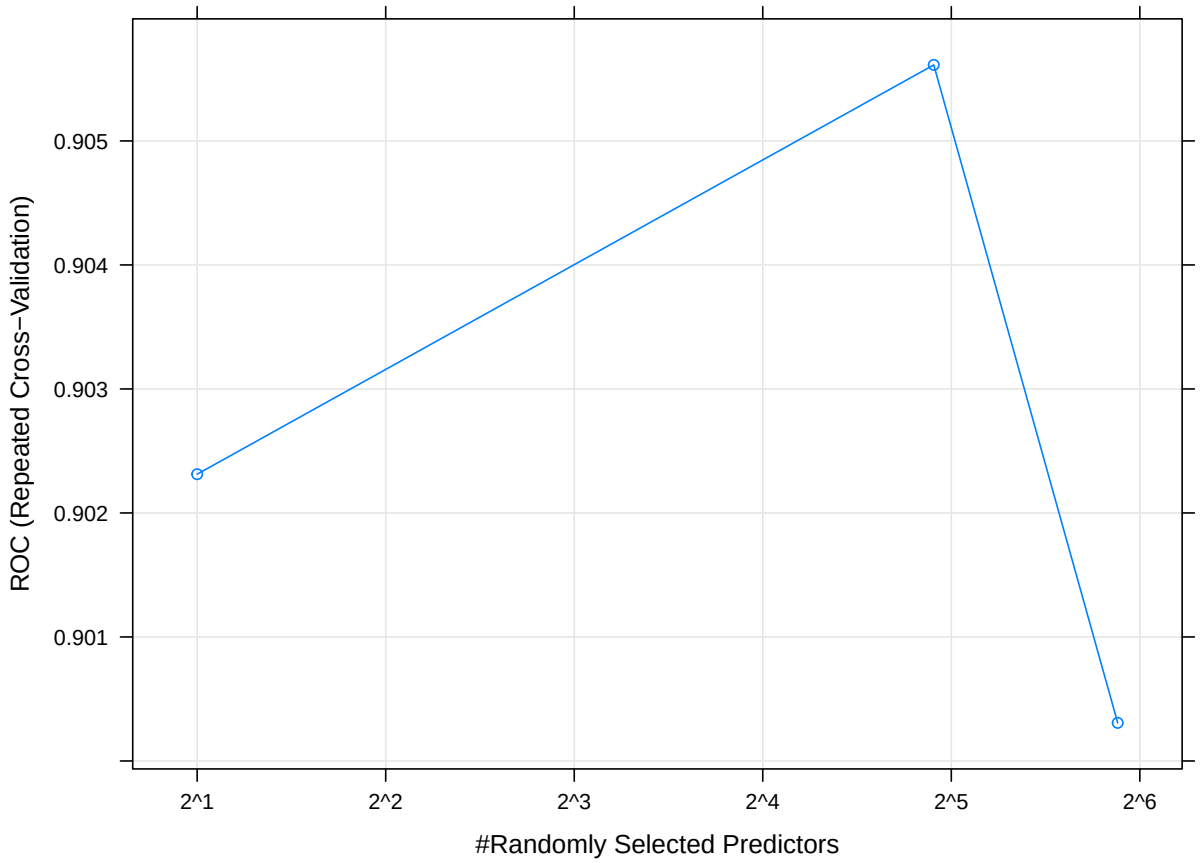
```
## Random Forest
##
## 1009 samples
##   59 predictor
##    2 classes: 'PS', 'WS'
##
## Pre-processing: centered (59), scaled (59)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
```

```
## Summary of sample sizes: 908, 908, 908, 908, 908, 909, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.9023126  0.8788525  0.7344239
##   30    0.9056132  0.8699735  0.7782835
##   59    0.9003073  0.8605655  0.7854433
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 30.
```

```
# display the final model
rfTune$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 30
##
##         OOB estimate of  error rate: 15.86%
## Confusion matrix:
##     PS  WS class.error
## PS 555  81   0.1273585
## WS  79 294   0.2117962
```

```
# plot the model
plot(rfTune, metric = "ROC", scales = list(x = list(log=2)))
```



Evaluate model performance

```
# make preditions on the test set
rfPred <- predict(rfTune, testing[, names(testing) != "Class"])
# display the confusion matrix
confusionMatrix(rfPred, testing$Class)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  PS  WS
##         PS 564  81
##         WS 100 265
##
##                Accuracy : 0.8208
##                  95% CI : (0.7957, 0.844)
##     No Information Rate : 0.6574
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6073
##
```

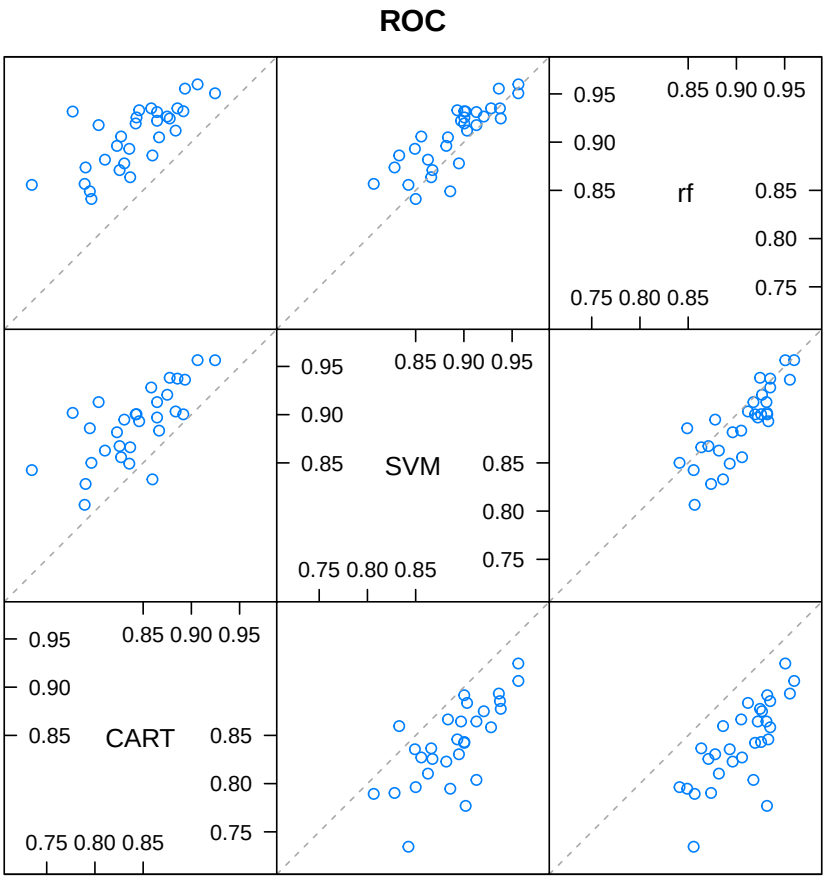
```
##  Mcnemar's Test P-Value : 0.1809
##
##             Sensitivity : 0.8494
##             Specificity : 0.7659
##          Pos Pred Value : 0.8744
##          Neg Pred Value : 0.7260
##              Prevalence : 0.6574
##          Detection Rate : 0.5584
##    Detection Prevalence : 0.6386
##       Balanced Accuracy : 0.8076
##
##        'Positive' Class : PS
##
```

Collecting results with resamples

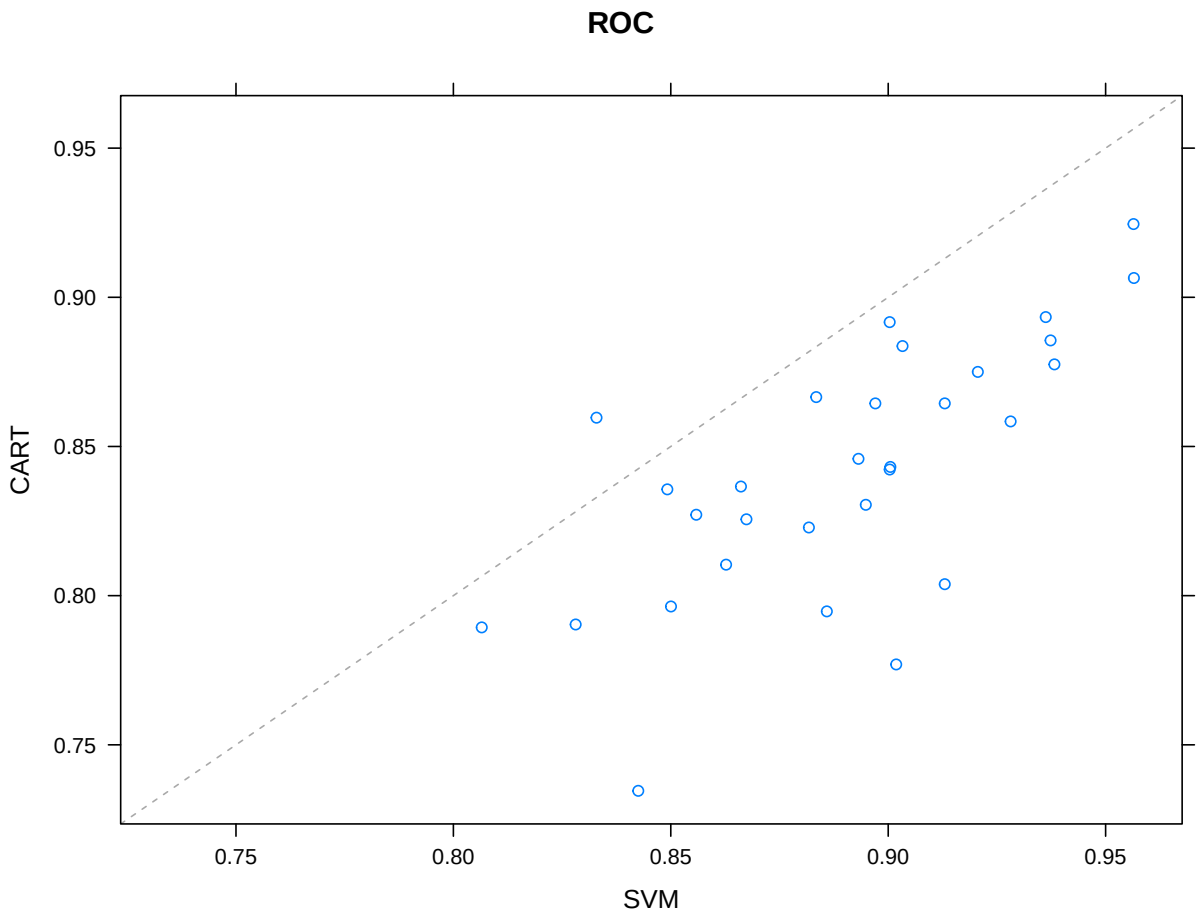
```
# collect resamples from the tuned rpart, svm and rf models
cvValues <- resamples(list(CART = rpartTune, SVM = svmTune, rf = rfTune))
# display the summary statistics of the values
summary(cvValues)
```

```
##
## Call:
## summary.resamples(object = cvValues)
##
## Models: CART, SVM, rf
## Number of resamples: 30
##
## ROC
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## CART 0.7345806 0.8134979 0.8427026 0.8419085 0.8728885 0.9245477    0
## SVM  0.8065208 0.8635728 0.8959439 0.8901121 0.9130068 0.9565034    0
## rf   0.8410610 0.8789197 0.9148015 0.9056132 0.9315696 0.9598818    0
##
## Sens
##           Min.   1st Qu.    Median      Mean   3rd Qu.     Max. NA's
## CART 0.7301587 0.7968750 0.8267609 0.8237930 0.8437500 0.906250    0
## SVM  0.7936508 0.8412698 0.8740079 0.8741815 0.9012277 0.968254    0
## rf   0.7777778 0.8437500 0.8740079 0.8699735 0.9047619 0.937500    0
##
## Spec
##           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## CART 0.5526316 0.6315789 0.6756757 0.6838786 0.7297297 0.8648649    0
## SVM  0.5945946 0.6623400 0.7332859 0.7309388 0.8040541 0.8918919    0
## rf   0.6486486 0.7364865 0.7631579 0.7782835 0.8323257 0.9189189    0
```

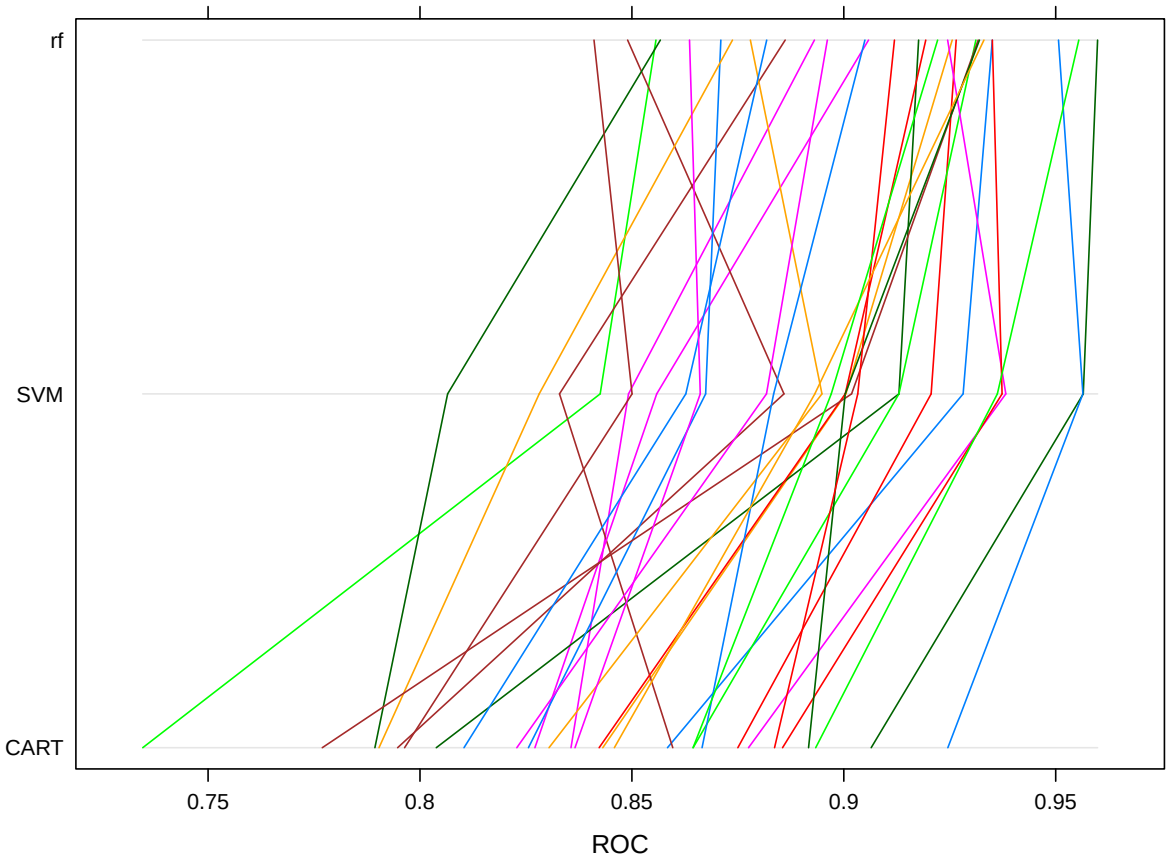
```
# Visualize resamples with splom
splom(cvValues, metric = "ROC")
```



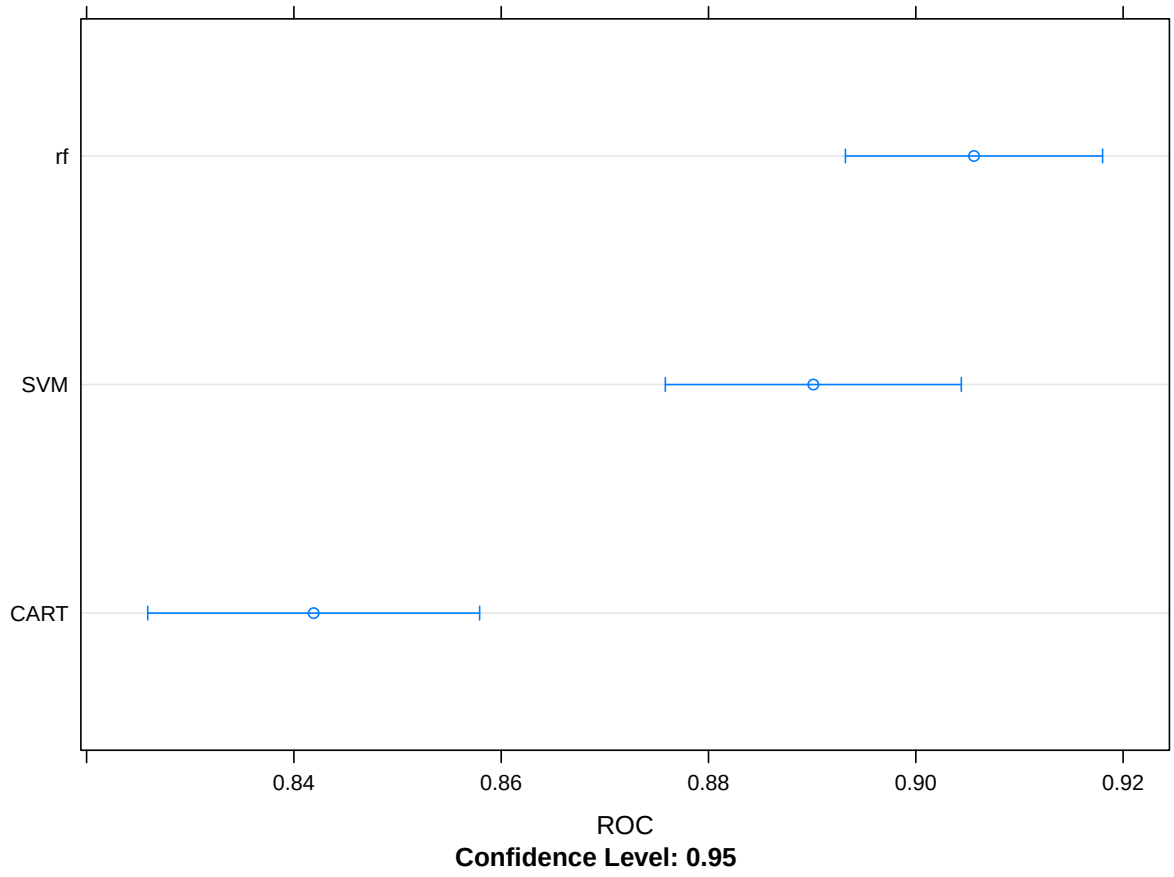

```
# Visualize resamples with xyplot
xyplot(cvValues, metric = "ROC")
```




```
# Visualize resamples with paralellplot
parallelplot(cvValues, metric = "ROC")
```




```
# Visualize resamples with dotplot
dotplot(cvValues, metric = "ROC")
```



Comparing models

```
# Comparing models
rocDiffs <- diff(cvValues, metric = "ROC")
# display comparison
summary(rocDiffs)
```

```
##
## Call:
## summary.diff.resamples(object = rocDiffs)
##
## p-value adjustment: bonferroni
## Upper diagonal: estimates of the difference
## Lower diagonal: p-value for H0: difference = 0
##
## ROC
##      CART      SVM      rf
## CART           -0.0482  -0.0637
## SVM  9.369e-09          -0.0155
## rf   2.793e-12 0.001624
```

```r
# Visualizing differences
dotplot(rocDiffs, metric = "ROC")
```