```
1 #Support Vector Machine (SVM) example using iris dataset
2 data(iris)
3 head(iris) # inspecting the first six rows of the dataset
4 str(iris) # structure of the dataset
5 library(ggplot2)
6 library(e1071)
7
```

A data.frame: 6 × 5

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
	<dbl>	<dbl>	<dbl>	<dbl>	<fct>
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

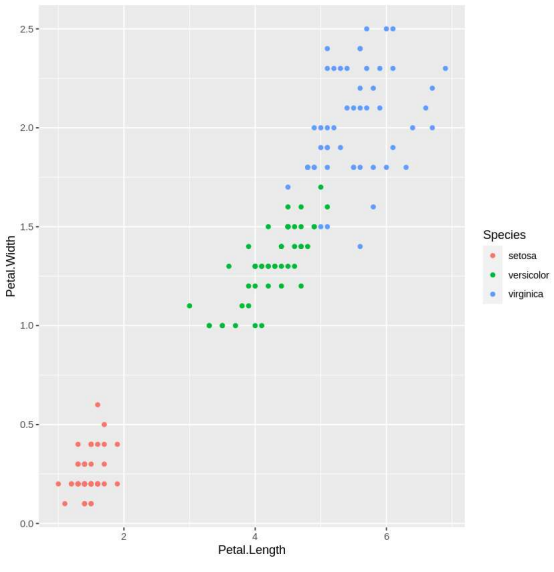
```
'data.frame':   150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
1 install.packages("e1071")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

also installing the dependency ‘proxy’
```

```
1 # we will seperate the Species based on the color and plot with Petal.Legth Vs Petal. Width
2 # We can clearly see the seperation of Setosa however, there is an overlappings in Versicolor and Virginica.
3 # Now we plot using the aplot() fuction, X= Petal. Length, Y= Petal. Width using the color seperation respect to # Species.
4 qplot(Petal.Length, Petal.Width, data=iris, color = Species)
5
```

```
Warning message:
“`qplot()` was deprecated in ggplot2 3.4.0.”
```




```
1 # Now we can use the built in svm() function that comes in the e1071 library # here we will name our first svm model as #svm_model1
2 # read the svm() documentation on RStudio by using the help(svm) function help ("svm")
3 svm_model1 <- svm(Species~., data = iris )
4
```

```
1 summary(svm_model1)
```

```
Call:
svm(formula = Species ~ ., data = iris)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  51

 ( 8 22 21 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

```
1 # Prediction using the model (svm_model1) we created on the iris dataset.
2 pred1 <- predict(svm_model1, iris)
3 # creating a table using the predicted one and the actual iris dataset
4 table1 <- table(Predicted = pred1, Actual = iris$Species)
5 table1
6
```

```
             Actual
Predicted    setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         48         2
  virginica       0          2        48
```

```
1 #We can calculate the modell accuracy
2 Model1_accuracyRate =sum(diag(table1) )/sum(table1)
3 Model1_accuracyRate
4 # We can calcuate the missclassification rate
5 Model1_MissClassificationRate = 1 - Model1_accuracyRate
6 Model1_MissClassificationRate
7
```

0.973333333333333

```
# Now we can use the built in svm() function that comes in the e1071 library # here we will name our first svm model as #svm_model1
# read the svm() documentation on RStudio by using the help(svm) function help ("svm")
svm_model1 <- svm(Species~., data = iris,kernel = "polynomial" )

```

```
summary(svm_model1)
```

```
Call:
svm(formula = Species ~ ., data = iris, kernel = "polynomial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  1
     degree:  3
     coef.0:  0

Number of Support Vectors:  54

 ( 6 26 22 )


Number of Classes:  3

Levels:
 setosa versicolor virginica
```

```
# Prediction using the model (svm_model1) we created on the iris dataset.
pred1 <- predict(svm_model1, iris)
# creating a table using the predicted one and the actual iris dataset
table1 <- table(Predicted = pred1, Actual = iris$Species)
table1
```

```
             Actual
Predicted     setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         50         7
  virginica       0          0        43
```

```
#We can calculate the modell accuracy
Model1_accuracyRate =sum(diag(table1) )/sum(table1)
Model1_accuracyRate
# We can calcuate the missclassification rate
Model1_MissClassificationRate = 1 - Model1_accuracyRate
Model1_MissClassificationRate

```

0.953333333333333
0.0466666666666666

```
#######I noticed decrease in accuracy after the kernel was set to polynomial.
```

```
install.packages("tree")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```

```
# Fitting a Regression Trees
library(MASS)
library(tree)
set.seed (1)
head (Boston)
train =sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston = tree (medv~.,Boston, subset = train )
summary (tree.boston)
# Note that the output summary O indicates that only three of the variables have been
# used to constructing the tree. In the context of a regression tree,
# the deviance is simly the sum of squared errors for the tree.
# Regression Tree
tree ( formula = medv ~., data = Boston, subset = train )

```

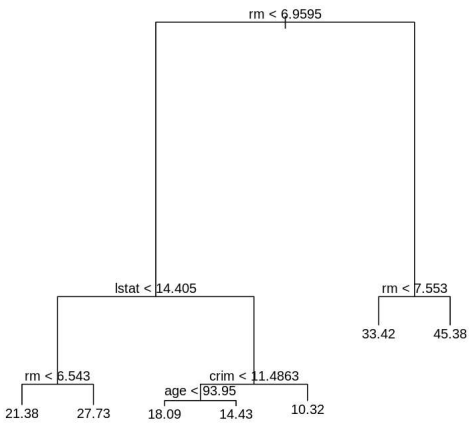
A data.frame: 6 × 14

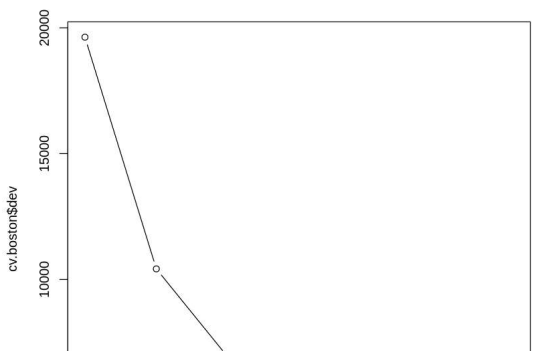
	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
	<dbl>	<dbl>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
2	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
3	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
4	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
5	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
6	0.02985	0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7

```
Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
Variables actually used in tree construction:
[1] "rm"    "lstat" "crim"  "age"
Number of terminal nodes:  7
Residual mean deviance:  10.38 = 2555 / 246
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
node), split, n, deviance, yval
      * denotes terminal node

 1) root 253 19450.0 21.79
   2) rm < 6.9595 222  6794.0 19.35
     4) lstat < 14.405 135  1816.0 22.51
       8) rm < 6.543 111   763.1 21.38 *
       9) rm > 6.543 24   256.5 27.73 *
     5) lstat > 14.405 87  1554.0 14.46
      10) crim < 11.4863 61   613.8 16.23
        20) age < 93.95 30   245.7 18.09 *
        21) age > 93.95 31   164.1 14.43 *
      11) crim > 11.4863 26   302.7 10.32 *
   3) rm > 6.9595 31  1929.0 39.21
     6) rm < 7.553 16   505.5 33.42 *
     7) rm > 7.553 15   317.0 45.38 *
```

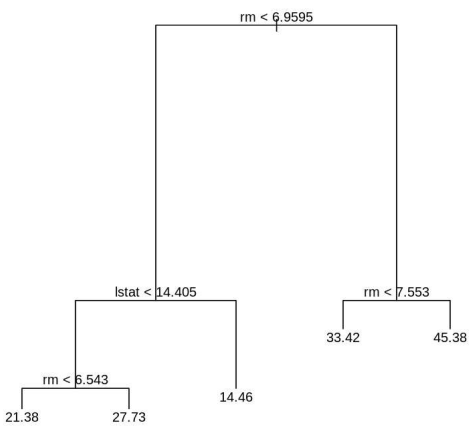
```
 1 # Regression Tree
 2 tree (formula = medv ~., , data = Boston, subset = train)
 3 # We now plot the tree
 4 plot(tree.boston)
 5 text(tree.boston, pretty =0 )
 6 # The variable "Istat" measure the percentage of the individuals with lower socioeconimics status
 7 # The tree indicates that the lower values of lstat corresponds to more expensive houese.
 8 # Now we use the cv.tree() function to see whether pruning the tree will
 9 # improve performance.
10 cv.boston=cv.tree(tree.boston)
11 plot (cv.boston$size, cv.boston$dev, typ ='b')
12
```

```
    node), split, n, deviance, yval
          * denotes terminal node

     1) root 253 19450.0 21.79
       2) rm < 6.9595 222  6794.0 19.35
         4) lstat < 14.405 135  1816.0 22.51
           8) rm < 6.543 111   763.1 21.38 *
           9) rm > 6.543 24   256.5 27.73 *
         5) lstat > 14.405 87  1554.0 14.46
          10) crim < 11.4863 61   613.8 16.23
            20) age < 93.95 30   245.7 18.09 *
            21) age > 93.95 31   164.1 14.43 *
          11) crim > 11.4863 26   302.7 10.32 *
       3) rm > 6.9595 31  1929.0 39.21
         6) rm < 7.553 16   505.5 33.42 *
         7) rm > 7.553 15   317.0 45.38 *
```



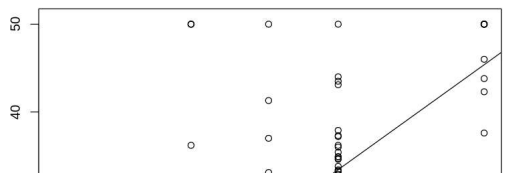



```
1 #In this case, the most complex tree is selected by cross-validation. # How- ever, if we wish to prune the tree, we could do so as follows, #using the prune.tree() function
2 prune.boston=prune.tree (tree.boston , best =5 )
3 plot (prune.boston)
4 text (prune.boston , pretty=0 )
5
```




```
 1
 2 yhat=predict(tree.boston , newdata=Boston [-train , ])
 3 boston.test=Boston [-train , "medv"]
 4 plot (yhat, boston.test)
 5
 6 abline (0,1)
 7 mean ((yhat-boston.test)^2)
 8     #In other words, the test set MSE associated with the regression tree is 25.05 .
 9 # The square root of the MSE is therefore around 5.005,
10 # indicating that this model leads to test predictions that
11 # are within around $5,005 of the true median home value for the suburb.
12
```

35.2868818594623




```
1 install.packages("randomForest")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```

```
|                                                                      |
```

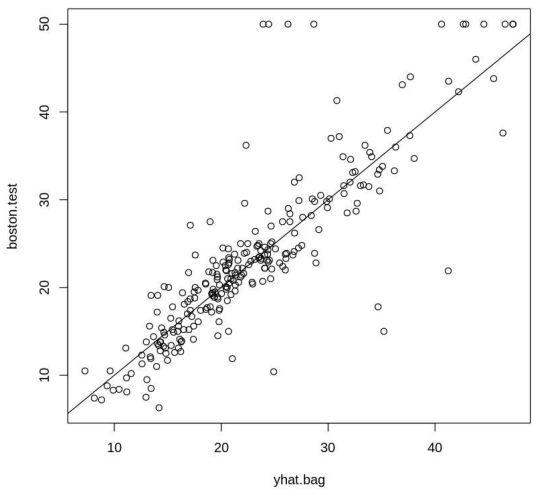
```
1 library(randomForest)
```

```
randomForest 4.7-1.1

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


Attaching package: ‘randomForest’


The following object is masked from ‘package:ggplot2’:

    margin
```

```
1 # Bagging and Random Forest Example
2
3 set.seed(1)
4 bag.boston = randomForest(medv ~., data=Boston, subset = train, mtry=13, importance=TRUE)
5 bag.boston
6 # The argument mtry=13 indicates that all 13 predictors should be considered
7 # for each split of the tree-in other words, that bagging should be done.
8 # How well does this bagged model perform on the test set?
9 yhat.bag = predict (bag.boston , newdata=Boston[-train , ])
10 plot(yhat.bag, boston.test)
11 # adding the abline()
12 abline (0,1)
13 mean ((yhat.bag-boston.test)^2)
14
```

```
Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13, importance = TRUE,      subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 11.39601
                    % Var explained: 85.17
```

23.5927297079061




```
1
2 bag.boston=randomForest (medv~. , data = Boston, subset=train, mtry=13, ntree =25 )
3 yhat.bag = predict (bag.boston , newdata = Boston [-train ,])
4 mean ((yhat.bag-boston.test)^2)
5
```

23.6671575152042

```
1 set.seed(1)
2 rf.boston=randomForest (medv~. , data=Boston, subset=train,
3 mtry =6, importance = TRUE)
4 yhat.rf= predict (rf.boston , newdata=Boston[-train , ])
5 mean ((yhat.rf-boston.test) ^2)
6
```

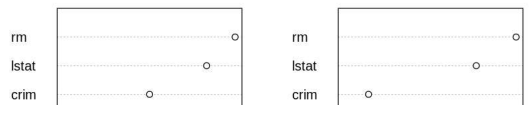
19.6202073910648

```
1 importance(rf.boston)
2 # Two measures of variable importance are reported.
3 #The former is based upon the mean decrease of accuracy in predictions on # the out of bag samples when a given variable is excluded from the model.
4 #The latter is a measure of the total decrease in node impurity that results " from splits over that variable, averaged over all trees.
5 # In the case of regression trees, the node impurity is measured by the training RSS, # and for classification trees by the deviance.
6 # Plots of these importance measures can be produced using the varImpPlot () function.
7 varImpPlot (rf.boston)
8
```

A matrix: 13 × 2 of type dbl

	%IncMSE	IncNodePurity
crim	16.697017	1076.08786
zn	3.625784	88.35342
indus	4.968621	609.53356
chas	1.061432	52.21793
nox	13.518179	709.87339
rm	32.343305	7857.65451
age	13.272498	612.21424
dis	9.032477	714.94674
rad	2.878434	95.80598
tax	9.118801	364.92479
ptratio	8.467062	823.93341
black	7.579482	275.62272
lstat	27.129817	6027.63740

rf.boston




```
1 install.packages("cvTools")
```

```
    Installing package into ‘/usr/local/lib/R/site-library’
    (as ‘lib’ is unspecified)

    also installing the dependencies ‘DEoptimR’, ‘robustbase’
```

```
1 install.packages("robustbase")
```

```
    Installing package into ‘/usr/local/lib/R/site-library’
    (as ‘lib’ is unspecified)
```

```
 1 library(cvTools)
 2 library(robustbase)
 3 data(coleman)
 4 call <- call("lmrob", formula = Y ~ .)
 5 # set up folds for cross-validation
 6 folds <- cvFolds(nrow(coleman), K = 5, R = 10)
 7 # perform cross-validation
 8 cvTool(call, data = coleman, y = coleman$Y, cost = rtmspe, folds = folds, costArgs = list(trim = 0.1))
 9 #vary K and R
10 #look at cvfits, use densityplot,
11 tuning <- list(tuning.psi=seq(2., 6., 20))
12 cvFitsLmrob <- cvTuning(call, data = coleman, y = coleman$Y, tuning = tuning, cost = rtmspe, folds = folds, costArgs = list(trim = 0.1))
13 # look at output
14 cvFitsLmrob
15 # summarize
16 aggregate(cvFitsLmrob, summary)
17
```

```
1.2061900
1.0399689
1.0958950
1.0251480
1.3006212
1.2480431
1.0827943
1.1132592
1.0614847
0.8597776
```

```
Warning message in lmrob.fit(x, y, control, init = init):
“M-step did NOT converge. Returning unconverged SM-estimate”
Warning message in lmrob.S(x, y, control = control):
```

```
install.packages("robustbase")
install.packages("cvTools")
library("robustbase")
require(cvTools)
data("coleman")
set.seed(1234) # set seed for reproducibility
## set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)
## compare raw and reweighted LTS estimators for
## 50% and 75% subsets
# 50% subsets
fitLts50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cvFitLts50 <- cvLts(fitLts50, cost = rtmspe, folds = folds,
fit = "both", trim = 0.1)
# 75% subsets
fitLts75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cvFitLts75 <- cvLts(fitLts75, cost = rtmspe, folds = folds,
fit = "both", trim = 0.1)
# combine results into one object
cvFitsLts <- cvSelect("0.5" = cvFitLts50, "0.75" = cvFitLts75)
cvFitsLts
# "cv" object
ncv(cvFitLts50)
nfits(cvFitLts50)
cvNames(cvFitLts50)
cvNames(cvFitLts50) <- c("improved", "initial")
fits(cvFitLts50)
cvFitLts50
# "cvSelect" object
ncv(cvFitsLts)
nfits(cvFitsLts)
cvNames(cvFitsLts)
cvNames(cvFitsLts) <- c("improved", "initial")
fits(cvFitsLts)
fits(cvFitsLts) <- 1:2
cvFitsLts

```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)


5-fold CV results:
   Fit reweighted      raw
1  0.5   1.165749 1.528544
2 0.75   1.011054 1.167683

Best model:
reweighted        raw
    "0.75"     "0.75"
2
NULL
'reweighted' · 'raw'
NULL
5-fold CV results:
improved  initial
1.165749 1.528544
2
2
'reweighted' · 'raw'
0.5 · 0.75
▶ Levels:

5-fold CV results:
  Fit improved  initial
1   1 1.165749 1.528544
2   2 1.011054 1.167683

Best model:
improved  initial
       2        2
```

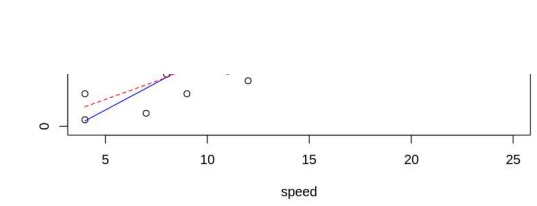
```
# set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)
## compare LS, MM and LTS regression
# perform cross-validation for an LS regression model
fitLm <- lm(Y ~ ., data = coleman)
cvFitLm <- cvLm(fitLm, cost = rtmspe,
folds = folds, trim = 0.1)
# perform cross-validation for an MM regression model
fitLmrob <- lmrob(Y ~ ., data = coleman)
cvFitLmrob <- cvLmrob(fitLmrob, cost = rtmspe,
folds = folds, trim = 0.1)
# perform cross-validation for an LTS regression model
fitLts <- ltsReg(Y ~ ., data = coleman)
cvFitLts <- cvLts(fitLts, cost = rtmspe,
folds = folds, trim = 0.1)
# compare cross-validation results
cvSelect(LS = cvFitLm, MM = cvFitLmrob, LTS = cvFitLts)
## compare raw and reweighted LTS estimators for
## 50% and 75% subsets
# 50% subsets
fitLts50 <- ltsReg(Y ~ ., data = coleman, alpha = 0.5)
cvFitLts50 <- cvLts(fitLts50, cost = rtmspe, folds = folds,
fit = "both", trim = 0.1)
# 75% subsets
fitLts75 <- ltsReg(Y ~ ., data = coleman, alpha = 0.75)
cvFitLts75 <- cvLts(fitLts75, cost = rtmspe, folds = folds,
fit = "both", trim = 0.1)
# combine and plot results
cvSelect("0.5" = cvFitLts50, "0.75" = cvFitLts75)

```

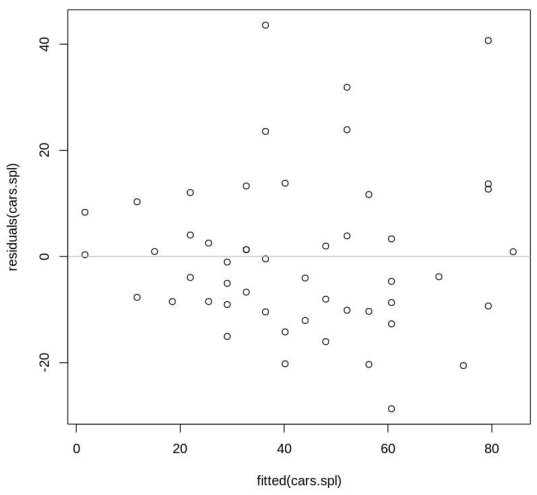
```
Warning message in lf.cov(init, x = x):
“.vcov.avar1: negative diag(<vcov>) fixed up; consider 'cov=".vcov.w."' instead”

5-fold CV results:
  Fit       CV
1  LS 1.742408
2  MM 1.201736
3 LTS 1.219383

Best model:
  CV
"MM"

5-fold CV results:
   Fit reweighted      raw
1  0.5   1.219383 1.607021
2 0.75   1.038942 1.222661

Best model:
reweighted        raw
    "0.75"     "0.75"
```

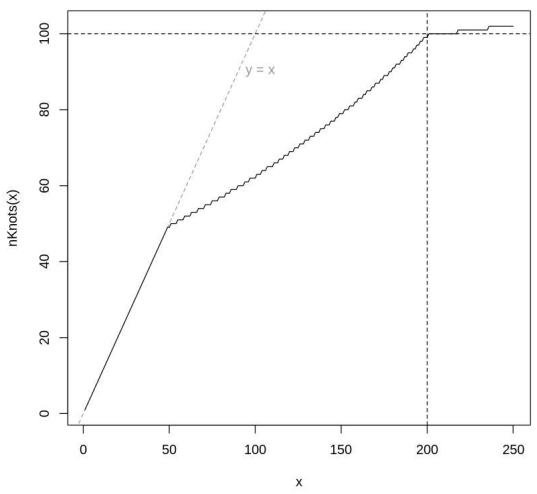
```
set.seed(1234) # set seed for reproducibility
# set up folds for cross-validation
folds <- cvFolds(nrow(coleman), K = 5, R = 10)
# perform cross-validation for an LS regression model
fitLm <- lm(Y ~ ., data = coleman)
repCV(fitLm, cost = rtmspe, folds = folds, trim = 0.1)
# perform cross-validation for an MM regression model
fitLmrob <- lmrob(Y ~ ., data = coleman)
repCV(fitLmrob, cost = rtmspe, folds = folds, trim = 0.1)
# perform cross-validation for an LTS regression model
fitLts <- ltsReg(Y ~ ., data = coleman)
repCV(fitLts, cost = rtmspe, folds = folds, trim = 0.1)
repCV(fitLts, cost = rtmspe, folds = folds,
fit = "both", trim = 0.1)


```

```
5-fold CV results:
      CV
1.775811
Warning message in lmrob.S(x, y, control = control):
“find_scale() did not converge in 'maxit.scale' (= 200) iterations with tol=1e-10, last rel.diff=0”
Warning message in lmrob.S(x, y, control = control):
“find_scale() did not converge in 'maxit.scale' (= 200) iterations with tol=1e-10, last rel.diff=0”
Warning message in lmrob.S(x, y, control = control):
“find_scale() did not converge in 'maxit.scale' (= 200) iterations with tol=1e-10, last rel.diff=0”
5-fold CV results:
      CV
1.023272
5-fold CV results:
      CV
1.165749
5-fold CV results:
reweighted        raw
  1.165749   1.528544
```

```
require(graphics)
data(cars)

attach(cars)
plot(speed, dist, main = "data(cars)  &  smoothing splines")
cars.spl <- smooth.spline(speed, dist)
(cars.spl)
## This example has duplicate points, so avoid cv = TRUE

lines(cars.spl, col = "blue")
lines(smooth.spline(speed, dist, df = 10), lty = 2, col = "red")
legend(5,120,c(paste("default [C.V.] => df =",round(cars.spl$df,1)),
               "s( * , df = 10)"), col = c("blue","red"), lty = 1:2,
       bg = 'bisque')
detach()


## Residual (Tukey Anscombe) plot:
plot(residuals(cars.spl) ~ fitted(cars.spl))
abline(h = 0, col = "gray")

## consistency check:
stopifnot(all.equal(cars$dist,
                    fitted(cars.spl) + residuals(cars.spl)))

## Visualize the behavior of  .nknots.smspl()
nKnots <- Vectorize(.nknots.smspl) ; c.. <- adjustcolor("gray20",.5)
curve(nKnots, 1, 250, n=250)
abline(0,1, lty=2, col=c..); text(90,90,"y = x", col=c.., adj=-.25)
abline(h=100,lty=2); abline(v=200, lty=2)

n <- c(1:799, seq(800, 3490, by=10), seq(3500, 10000, by = 50))
plot(n, nKnots(n), type="l", main = "Vectorize(.nknots.smspl) (n)")
abline(0,1, lty=2, col=c..); text(180,180,"y = x", col=c..)
n0 <- c(50, 200, 800, 3200); c0 <- adjustcolor("blue3", .5)
lines(n0, nKnots(n0), type="h", col=c0)
axis(1, at=n0, line=-2, col.ticks=c0, col=NA, col.axis=c0)
axis(4, at=.nknots.smspl(10000), line=-.5, col=c..,col.axis=c.., las=1)

##-- artificial example
y18 <- c(1:3, 5, 4, 7:3, 2*(2:5), rep(10, 4))
xx  <- seq(1, length(y18), len = 201)
(s2  <- smooth.spline(y18)) # GCV
(s02  <- smooth.spline(y18, spar = 0.2))
(s02. <- smooth.spline(y18, spar = 0.2, cv = NA))
plot(y18, main = deparse(s2$call), col.main = 2)
lines(s2, col = "gray"); lines(predict(s2, xx), col = 2)
lines(predict(s02, xx), col = 3); mtext(deparse(s02$call), col = 3)

## The following shows the problematic behavior of 'spar' searching:
(s2  <- smooth.spline(y18, control =
                      list(trace = TRUE, tol = 1e-6, low = -1.5)))
(s2m <- smooth.spline(y18, cv = TRUE, control =
                      list(trace = TRUE, tol = 1e-6, low = -1.5)))
## both above do quite similarly (Df = 8.5 +- 0.2)

```








```
Call:
smooth.spline(x = y18)
```

```
require(graphics)

op <- par(mfrow = c(2,1), mgp = c(2,.8,0), mar = 0.1+c(3,3,3,1))
n <- 9
x <- 1:n
y <- rnorm(n)
plot(x, y, main = paste("spline[fun](.) through", n, "points"))
lines(spline(x, y))
lines(spline(x, y, n = 201), col = 2)

y <- (x-6)^2
plot(x, y, main = "spline(.) -- 3 methods")
lines(spline(x, y, n = 201), col = 2)
lines(spline(x, y, n = 201, method = "natural"), col = 3)
lines(spline(x, y, n = 201, method = "periodic"), col = 4)
legend(6, 25, c("fmm","natural","periodic"), col = 2:4, lty = 1)

y <- sin((x-0.5)*pi)
f <- splinefun(x, y)
ls(envir = environment(f))
splinecoef <- get("z", envir = environment(f))
curve(f(x), 1, 10, col = "green", lwd = 1.5)
points(splinecoef, col = "purple", cex = 2)
curve(f(x, deriv = 1), 1, 10, col = 2, lwd = 1.5)
curve(f(x, deriv = 2), 1, 10, col = 2, lwd = 1.5, n = 401)
curve(f(x, deriv = 3), 1, 10, col = 2, lwd = 1.5, n = 401)
par(op)

## Manual spline evaluation --- demo the coefficients :
.x <- splinecoef$x
u <- seq(3, 6, by = 0.25)
(ii <- findInterval(u, .x))
dx <- u - .x[ii]
f.u <- with(splinecoef,
            y[ii] + dx*(b[ii] + dx*(c[ii] + dx* d[ii])))
stopifnot(all.equal(f(u), f.u))

## An example with ties (non-unique  x values):
set.seed(1); x <- round(rnorm(30), 1); y <- sin(pi * x) + rnorm(30)/10
plot(x, y, main = "spline(x,y)  when x has ties")
lines(spline(x, y, n = 201), col = 2)
## visualizes the non-unique ones:
tx <- table(x); mx <- as.numeric(names(tx[tx > 1]))
ry <- matrix(unlist(tapply(y, match(x, mx), range, simplify = FALSE)),
             ncol = 2, byrow = TRUE)
segments(mx, ry[, 1], mx, ry[, 2], col = "blue", lwd = 2)

## An example of monotone interpolation
n <- 20
set.seed(11)
x. <- sort(runif(n)) ; y. <- cumsum(abs(rnorm(n)))
plot(x., y.)
curve(splinefun(x., y.)(x), add = TRUE, col = 2, n = 1001)
curve(splinefun(x., y., method = "monoH.FC")(x), add = TRUE, col = 3, n = 1001)
curve(splinefun(x., y., method = "hyman")   (x), add = TRUE, col = 4, n = 1001)
legend("topleft",
       paste0("splinefun( \"", c("fmm", "monoH.FC", "hyman"), "\" )"),
       col = 2:4, lty = 1, bty = "n")

## and one from Fritsch and Carlson (1980), Dougherty et al (1989)
x. <- c(7.09, 8.09, 8.19, 8.7, 9.2, 10, 12, 15, 20)
f <- c(0, 2.76429e-5, 4.37498e-2, 0.169183, 0.469428, 0.943740,
       0.998636, 0.999919, 0.999994)
s0 <- splinefun(x., f)
s1 <- splinefun(x., f, method = "monoH.FC")
s2 <- splinefun(x., f, method = "hyman")
plot(x., f, ylim = c(-0.2, 1.2))
curve(s0(x), add = TRUE, col = 2, n = 1001) -> m0
curve(s1(x), add = TRUE, col = 3, n = 1001)
curve(s2(x), add = TRUE, col = 4, n = 1001)
legend("right",
       paste0("splinefun( \"", c("fmm", "monoH.FC", "hyman"), "\" )"),
       col = 2:4, lty = 1, bty = "n")

## they seem identical, but are not quite:
xx <- m0$x
plot(xx, s1(xx) - s2(xx), type = "l",  col = 2, lwd = 2,
     main = "Difference   monoH.FC - hyman"); abline(h = 0, lty = 3)

x <- xx[xx < 10.2] ## full range: x <- xx .. does not show enough
ccol <- adjustcolor(2:4, 0.8)
matplot(x, cbind(s0(x, deriv = 2), s1(x, deriv = 2), s2(x, deriv = 2))^2,
        lwd = 2, col = ccol, type = "l", ylab = quote({{f*second}(x)}^2),
        main = expression({{f*second}(x)}^2 ~" for the three 'splines'"))
legend("topright",
       paste0("splinefun( \"", c("fmm", "monoH.FC", "hyman"), "\" )"),
       lwd = 2, col  =  ccol, lty = 1:3, bty = "n")

```