Iris dataset

Vinh Dang

10/26/2016

Introduction

Iris dataset probably is one of the most famous dataset. The dataset can be found here. However, the CSV format can be found here.

Getting started

Loading the dataset

```r
iris = read.csv("iris.csv")
```

Take a quick look

```r
str(iris)
```

```
## 'data.frame':    150 obs. of  6 variables:
##  $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ 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 ...
```

There are 150 lines, i.e. 150 instances in the dataset, 4 columns (4 features), and 3 classes.

We might want to divide train/test dataset (or using cross-validation as later).

```r
require(caTools)
```

```
## Loading required package: caTools
```

```r
# 75% for train and 25% for test
sample = sample.split(iris$Species, SplitRatio = 0.75)
train = subset(iris, sample == TRUE)
test = subset(iris, sample == FALSE)
```

Test the division

```r
nrow(train)
```

```
## [1] 114
```

```r
nrow(test)
```

```
## [1] 36
```

Check for NA values

```r
sum(is.na(df))
```

```
## Warning in is.na(df): is.na() applied to non-(list or vector) of type
## 'closure'
```

```
## [1] 0
```

Classification

Multinomial Logistic Regression

```r
require(caret)
```

```
## Loading required package: caret
```

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

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

```r
# train
train.multinom = train(Species ~ Sepal.Length + Sepal.Width +
    Petal.Length + Petal.Width, data = train, method = "multinom")
```

```
## Loading required package: nnet
```

```r
# predict with test data
predict.multinom = predict(train.multinom, newdata = test)

# check the result
table.multinom = table(predict.multinom, test$Species)

# full result
table.multinom
```

```
##                 
## predict.multinom setosa versicolor virginica
##       setosa         12          0         0
##       versicolor      0         12         1
##       virginica       0          0        11
```

```
# accuracy
sum(diag(table.multinom))/sum(table.multinom)
```

```
## [1] 0.9722222
```

Well, quite cool. The accuracy is 97%. How about other algorithms?

SVM

```
library(e1071)
library(caret)
train.svm = train(Species ~ Sepal.Length + Sepal.Width + Petal.Length +
    Petal.Width, data = train, method = "svmLinear")
```

```
# predict with test data
predict.svm = predict(train.svm, newdata = test)

# check the result
table.svm = table(predict.svm, test$Species)

# full result
table.svm
```

```
##
## predict.svm  setosa versicolor virginica
##   setosa         12          0         0
##   versicolor      0         12         0
##   virginica       0          0        12
```

```
# accuracy
sum(diag(table.svm))/sum(table.svm)
```

```
## [1] 1
```

Again, the accuracy score is 97.2%.

Decision tree

```
library(e1071)
library(caret)
train.rpart = train(Species ~ Sepal.Length + Sepal.Width + Petal.Length +
    Petal.Width, data = train, method = "rpart")
```

```
# predict with test data
predict.rpart = predict(train.rpart, newdata = test)

# check the result
```

```
table.rpart = table(predict.rpart, test$Species)

# full result
table.rpart
```

```
##
## predict.rpart setosa versicolor virginica
##    setosa         12          0         0
##    versicolor      0         11         2
##    virginica       0          1        10
```

```
# accuracy
sum(diag(table.rpart))/sum(table.rpart)
```

```
## [1] 0.9166667
```

Not very good this time. Only 91%.

Random forest

Usually random forest provide very good classification results. How about this case?

```
# we will use h2o for speeding up not really need because the
# data is small
library(h2o)
```

```
## Loading required package: statmod
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit http://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

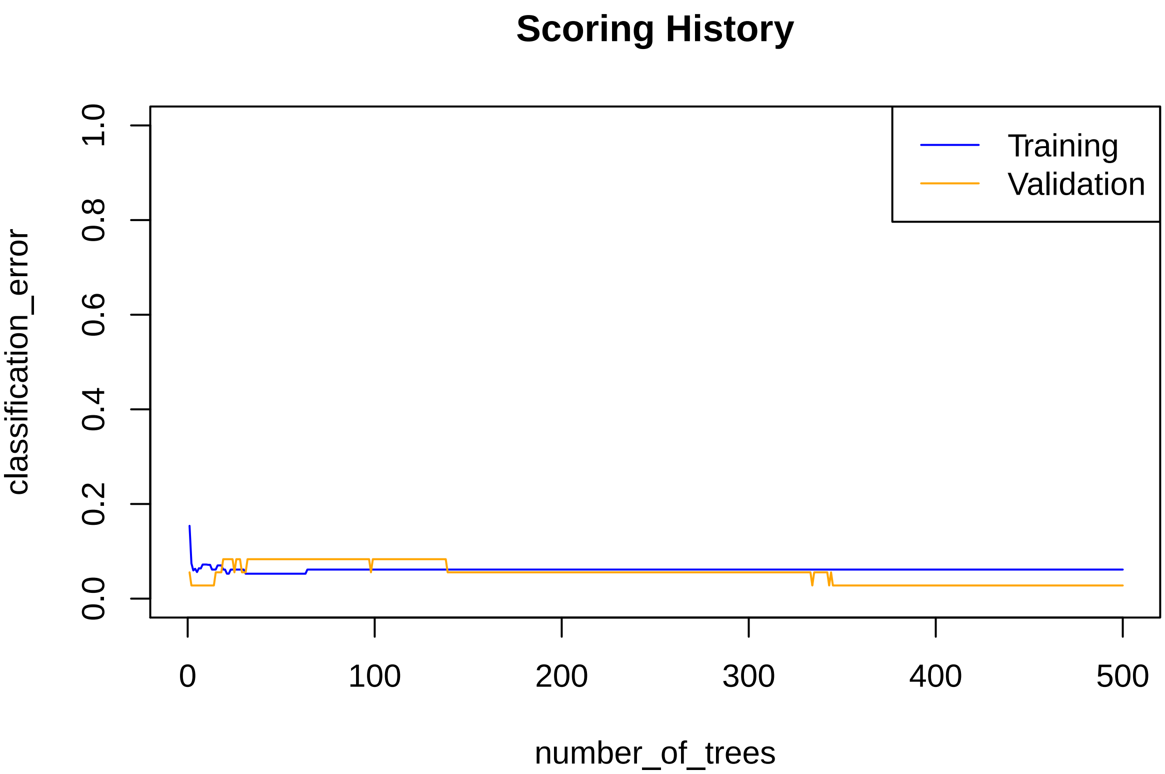
```
h2o.init()

train.rf = h2o.randomForest(y = 6, x = 2:5, training_frame = as.h2o(train),
    validation_frame = as.h2o(test), ntrees = 500)
```

```
print(train.rf)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: drf
## Model ID:  DRF_model_R_1477488686931_1
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1             500                     1500              192663         1
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1         9    3.24800          2         15     5.18400
##
##
## H2OMultinomialMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("train")`
## MSE: (Extract with `h2o.mse`) 0.03315535
## RMSE: (Extract with `h2o.rmse`) 0.1820861
## Logloss: (Extract with `h2o.logloss`) 0.1009518
## Mean Per-Class Error: 0.06140351
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: vertical: actual; across: predicted
##            setosa versicolor virginica  Error      Rate
## setosa         38          0         0 0.0000 =  0 / 38
## versicolor      0         35         3 0.0789 =  3 / 38
## virginica       0          4        34 0.1053 =  4 / 38
## Totals         38         39        37 0.0614 = 7 / 114
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-3 Hit Ratios:
##   k hit_ratio
## 1 1  0.938596
## 2 2  1.000000
## 3 3  1.000000
```

```
##
##
## H2OMultinomialMetrics: drf
## ** Reported on validation data. **
##
## Validation Set Metrics:
## =====================
##
## Extract validation frame with `h2o.getFrame("test")`
## MSE: (Extract with `h2o.mse`) 0.04260067
## RMSE: (Extract with `h2o.rmse`) 0.2063993
## Logloss: (Extract with `h2o.logloss`) 0.1526042
## Mean Per-Class Error: 0.02777778
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,valid = TRUE)`)
## =========================================================================
## Confusion Matrix: vertical: actual; across: predicted
##            setosa versicolor virginica  Error     Rate
## setosa         12          0         0 0.0000 = 0 / 12
## versicolor      0         12         0 0.0000 = 0 / 12
## virginica       0          1        11 0.0833 = 1 / 12
## Totals         12         13        11 0.0278 = 1 / 36
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,valid = TRUE)`
## =======================================================================
## Top-3 Hit Ratios:
##   k hit_ratio
## 1 1  0.972222
## 2 2  1.000000
## 3 3  1.000000
```

```r
plot(train.rf)
```



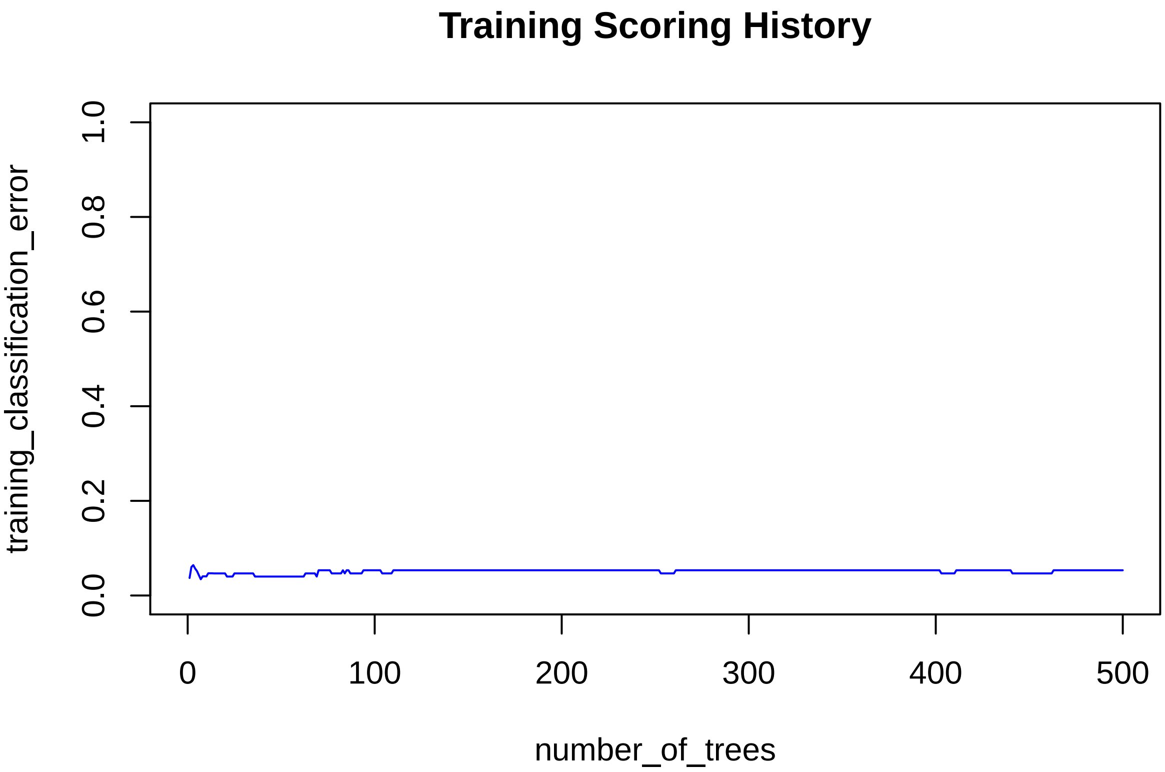
Random Forest gave us the accuracy score of 100%. So impressive. How about if we use cross-validation?

```
train.rf = h2o.randomForest(y = 6, x = 2:5, training_frame = as.h2o(iris),
    ntrees = 500, nfolds = 5)
```

```
print(train.rf)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: drf
## Model ID:  DRF_model_R_1477488686931_2
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1             500                     1500              211853         1
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1        11    3.71067          2         17     6.19600
##
##
## H2OMultinomialMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("iris")`
## MSE: (Extract with `h2o.mse`) 0.03552398
## RMSE: (Extract with `h2o.rmse`) 0.1884781
## Logloss: (Extract with `h2o.logloss`) 0.1151747
## Mean Per-Class Error: 0.05333333
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: vertical: actual; across: predicted
##            setosa versicolor virginica  Error      Rate
## setosa         50          0         0 0.0000 =  0 / 50
## versicolor      0         47         3 0.0600 =  3 / 50
## virginica       0          5        45 0.1000 =  5 / 50
## Totals         50         52        48 0.0533 = 8 / 150
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-3 Hit Ratios:
##   k hit_ratio
## 1 1  0.946667
## 2 2  1.000000
## 3 3  1.000000
##
##
##
## H2OMultinomialMetrics: drf
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **
##
## Cross-Validation Set Metrics:
## =====================
##
## Extract cross-validation frame with `h2o.getFrame("iris")`
## MSE: (Extract with `h2o.mse`) 0.033464
## RMSE: (Extract with `h2o.rmse`) 0.1829317
## Logloss: (Extract with `h2o.logloss`) 0.109004
## Mean Per-Class Error: 0.05333333
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,xval = TRUE)`
## =======================================================================
## Top-3 Hit Ratios:
##   k hit_ratio
## 1 1  0.946667
## 2 2  1.000000
## 3 3  1.000000
##
##
## Cross-Validation Metrics Summary:
##                                 mean          sd  cv_1_valid   cv_2_valid
## accuracy                   0.9496956 0.018866044   0.9444444          1.0
## err                      0.050304387 0.018866044 0.055555556          0.0
## err_count                        1.6  0.56568545         2.0          0.0
## logloss                   0.10382761 0.039743733  0.14378677 0.0094938725
## max_per_class_error      0.096007325  0.03864481 0.083333336          0.0
## mean_per_class_accuracy   0.95371187 0.017162396   0.9484127          1.0
## mean_per_class_error     0.046288155 0.017162396 0.051587302          0.0
## mse                      0.031515434 0.012744342 0.045896105  3.162059E-4
## r2                         0.9513115 0.018444968    0.924516   0.99960476
## rmse                      0.16078062 0.053221356  0.21423376   0.01778218
##                          cv_3_valid  cv_4_valid  cv_5_valid
## accuracy                       0.92  0.94285715   0.9411765
## err                            0.08 0.057142857  0.05882353
## err_count                       2.0         2.0         2.0
## logloss                  0.17367926   0.1083575  0.08382064
## max_per_class_error      0.15384616         0.1  0.14285715
## mean_per_class_accuracy  0.94871795  0.94285715   0.9285714
## mean_per_class_error    0.051282052 0.057142857 0.071428575
## mse                     0.051181864 0.035247233  0.02493576
## r2                       0.93602264  0.94117457  0.95523953
## rmse                      0.2262341  0.18774246  0.15791062
```

```
plot(train.rf)
```



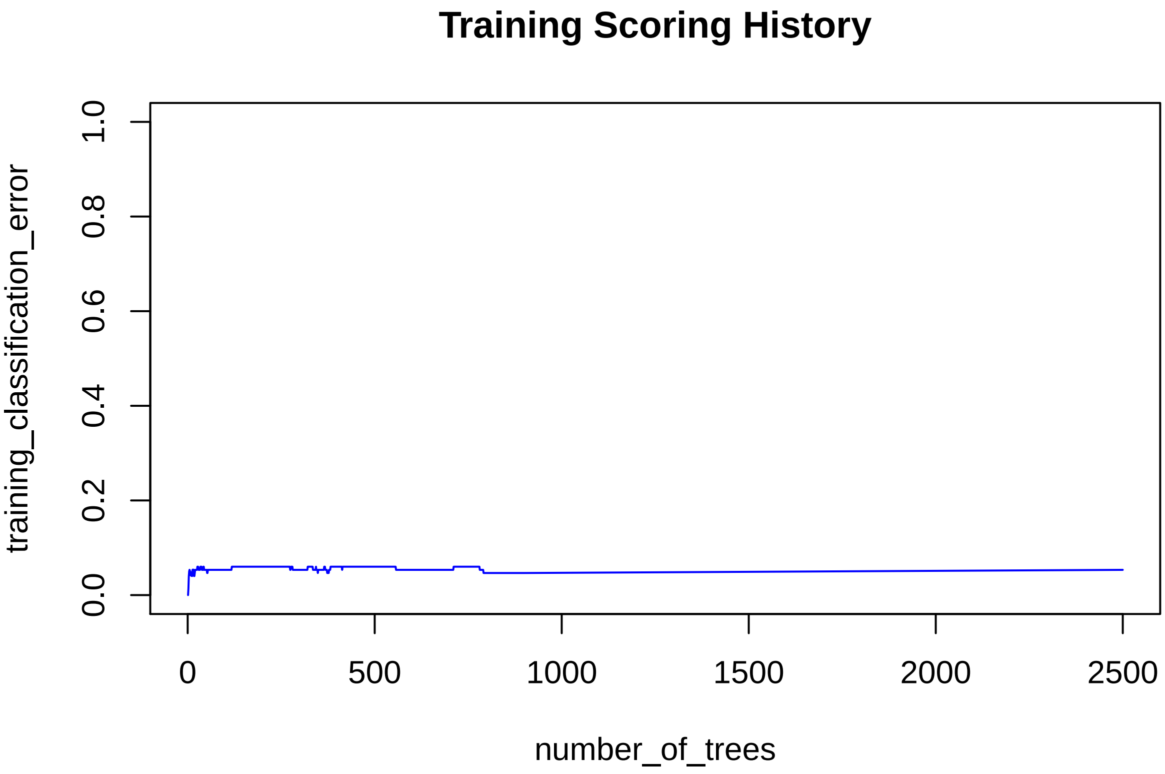
How about if we increas number of tree?

```
train.rf = h2o.randomForest(y = 6, x = 2:5, training_frame = as.h2o(iris),
    ntrees = 2500, nfolds = 5)
```

```
print(train.rf)
```

```
## Model Details:
## ==============
##
## H2OMultinomialModel: drf
## Model ID:  DRF_model_R_1477488686931_3
## Model Summary:
##   number_of_trees number_of_internal_trees model_size_in_bytes min_depth
## 1            2500                     7500             1066171         1
##   max_depth mean_depth min_leaves max_leaves mean_leaves
## 1        10    3.70453          2         18     6.20680
##
##
## H2OMultinomialMetrics: drf
## ** Reported on training data. **
## ** Metrics reported on Out-Of-Bag training samples **
##
## Training Set Metrics:
## =====================
##
## Extract training frame with `h2o.getFrame("iris")`
## MSE: (Extract with `h2o.mse`) 0.03549926
## RMSE: (Extract with `h2o.rmse`) 0.1884125
## Logloss: (Extract with `h2o.logloss`) 0.1141144
## Mean Per-Class Error: 0.05333333
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>,train = TRUE)`)
## =========================================================================
## Confusion Matrix: vertical: actual; across: predicted
##            setosa versicolor virginica  Error      Rate
## setosa         50          0         0 0.0000 =  0 / 50
## versicolor      0         47         3 0.0600 =  3 / 50
## virginica       0          5        45 0.1000 =  5 / 50
## Totals         50         52        48 0.0533 = 8 / 150
##
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,train = TRUE)`
## =======================================================================
## Top-3 Hit Ratios:
##   k hit_ratio
## 1 1  0.946667
## 2 2  1.000000
## 3 3  1.000000
##
##
##
## H2OMultinomialMetrics: drf
## ** Reported on cross-validation data. **
## ** 5-fold cross-validation on training data (Metrics computed for combined holdout predictions) **
##
## Cross-Validation Set Metrics:
## =====================
##
## Extract cross-validation frame with `h2o.getFrame("iris")`
## MSE: (Extract with `h2o.mse`) 0.03793098
## RMSE: (Extract with `h2o.rmse`) 0.1947588
## Logloss: (Extract with `h2o.logloss`) 0.1208446
## Mean Per-Class Error: 0.06
## Hit Ratio Table: Extract with `h2o.hit_ratio_table(<model>,xval = TRUE)`
## =======================================================================
## Top-3 Hit Ratios:
##   k hit_ratio
## 1 1  0.940000
## 2 2  1.000000
## 3 3  1.000000
##
##
## Cross-Validation Metrics Summary:
##                                mean          sd  cv_1_valid  cv_2_valid
## accuracy                 0.94055974 0.027000353   0.8888889  0.92105263
## err                     0.059440266 0.027000353  0.11111111 0.078947365
## err_count                       1.8  0.82462114         3.0         3.0
## logloss                  0.11602521 0.047408137  0.15816341  0.20216043
## max_per_class_error      0.13600732   0.0696644         0.3  0.15384616
## mean_per_class_accuracy  0.94860363 0.025470305         0.9   0.9184149
## mean_per_class_error     0.05139638 0.025470305         0.1  0.08158508
## mse                     0.036668606 0.016380891 0.055433407 0.060977507
## r2                        0.9415697 0.026248796   0.9286026  0.90642774
## rmse                     0.17473753 0.055386845    0.235443  0.24693625
##                          cv_3_valid   cv_4_valid  cv_5_valid
## accuracy                 0.96428573          1.0   0.9285714
## err                     0.035714287          0.0 0.071428575
## err_count                       1.0          0.0         2.0
## logloss                  0.05825078   0.01971234   0.1418391
## max_per_class_error     0.083333336          0.0  0.14285715
## mean_per_class_accuracy   0.9722222          1.0  0.95238096
## mean_per_class_error    0.027777778          0.0  0.04761905
## mse                     0.017049648 0.0016997926 0.048182685
## r2                       0.97143817    0.9977452   0.9036346
## rmse                      0.1305743   0.04122854  0.21950555
```

```r
plot(train.rf)
```



Shut down h2o

```r
h2o.shutdown(FALSE)
```