Package ‘CVST’

August 29, 2016

**Type** Package

**Title** Fast Cross-Validation via Sequential Testing

**Version** 0.2-1

**Date** 2013-12-10

**Depends** kernlab, Matrix

**Author** Tammo Krüger, Mikio Braun

**Maintainer** Tammo Krüger <tammokruger@googlemail.com>

**Description** This package implements the fast cross-validation via sequential testing (CVST) procedure. CVST is an improved cross-validation procedure which uses non-parametric testing coupled with sequential analysis to determine the best parameter set on linearly increasing subsets of the data. By eliminating underperforming candidates quickly and keeping promising candidates as long as possible, the method speeds up the computation while preserving the capability of a full cross-validation. Additionally to the CVST the package contains an implementation of the ordinary k-fold cross-validation with a flexible and powerful set of helper objects and methods to handle the overall model selection process. The implementations of the Cochran’s Q test with permutations and the sequential testing framework of Wald are generic and can therefore also be used in other contexts.

**License** GPL (>= 2.0)

**NeedsCompilation** no

**Repository** CRAN

**Date/Publication** 2013-12-10 14:50:04

**R topics documented:**

<table>
<thead>
<tr>
<th>R topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVST-package</td>
<td>2</td>
</tr>
<tr>
<td>cochransq.test</td>
<td>3</td>
</tr>
<tr>
<td>constructCVSTModel</td>
<td>4</td>
</tr>
<tr>
<td>constructData</td>
<td>5</td>
</tr>
<tr>
<td>constructLearner</td>
<td>6</td>
</tr>
<tr>
<td>constructParams</td>
<td>8</td>
</tr>
<tr>
<td>constructSequentialTest</td>
<td>8</td>
</tr>
<tr>
<td>CV</td>
<td>10</td>
</tr>
</tbody>
</table>
Description

This package implements the fast cross-validation via sequential testing (CVST) procedure. CVST is an improved cross-validation procedure which uses non-parametric testing coupled with sequential analysis to determine the best parameter set on linearly increasing subsets of the data. By eliminating underperforming candidates quickly and keeping promising candidates as long as possible, the method speeds up the computation while preserving the capability of a full cross-validation. Additionally to the CVST the package contains an implementation of the ordinary k-fold cross-validation with a flexible and powerful set of helper objects and methods to handle the overall model selection process. The implementations of the Cochran’s Q test with permutations and the sequential testing framework of Wald are generic and can therefore also be used in other contexts.

Details

Package: CVST
Type: Package
Version: 0.2
Date: 2013-03-25
License: GPL (>=2.0)

Author(s)

Tammo Krueger, Mikio Braun
Maintainer: Tammo Krueger <tammokrueger@googlemail.com>

References


cochranq.test


Examples

```r
ns = noisySine(100)
svm = constructSVMLearner()
params = constructParams(kernel="rbfdot", sigma=10^(-3:3), nu=c(0.05, 0.1, 0.2, 0.3))
opt = fastCV(ns, svm, params, constructCVSTModel())
```

---

**cochranq.test  Cochran’s Q Test with Permutation**

**Description**

Performs the Cochran’s Q test on the data. If the data matrix contains too few elements, the chisquare distribution of the test statistic is replaced by a permutation variant.

**Usage**

```r
cochranq.test(mat)
```

**Arguments**

- `mat`  
  The data matrix with the individuals in the rows and treatments in the columns.

**Value**

Returns a `htest` object with the usual entries.

**Author(s)**

Tammo Krueger <tammokrueger@googlemail.com>

**References**


**Examples**

```r
mat = matrix(c(rep(0, 10), 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1,
              0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0,
              1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1), ncol=4)
cochranq.test(mat)
mat = matrix(c(rep(0, 7), 1, rep(0, 12), 1, 1, 0, 1,
              rep(0, 5), 1, 0, 1, 0, 0, 0, 1, 0, 1), nrow=8)
cochranq.test(mat)
```

**constructCVSTModel**

Setup for a CVST Run.

**Description**

This is an helper object of type CVST.setup containing all necessary parameters for a CVST run.

**Usage**

```r
constructCVSTModel(steps = 10, beta = 0.1, alpha = 0.01,
similaritySignificance = 0.05, earlyStoppingSignificance = 0.05,
earlyStoppingWindow = 3, regressionSimilarityViaOutliers = FALSE)
```

**Arguments**

- **steps**: Number of steps CVST should run
- **beta**: Significance level for H0.
- **alpha**: Significance level for H1.
- **similaritySignificance**: Significance level of the similarity test.
- **earlyStoppingSignificance**: Significance level of the early stopping test.
- **earlyStoppingWindow**: Size of the early stopping window.
- **regressionSimilarityViaOutliers**: Should the less strict outlier-based similarity measure for regression tasks be used.

**Value**

A CVST.setup object suitable for fastCV.

**Author(s)**

Tammo Krueger <tammokrueger@googlemail.com>
**constructData**

**References**


**See Also**

fastCV

**constructData**

*Construction and Handling of CVST.data Objects*

**Description**

The CVST methods need a structured interface to both regression and classification data sets. These helper methods allow the construction and consistence handling of these types of data sets.

**Usage**

```
constructData(x, y)
getN(data)
getSubset(data, subset)
getX(data, subset = NULL)
shuffleData(data)
isClassification(data)
isRegression(data)
```

**Arguments**

- `x` The feature data as vector or matrix.
- `y` The observed values (regressands/labels) as list, vector or factor.
- `data` A `CVST.data` object generated via `constructData`.
- `subset` A index set.

**Value**

`constructData` returns a `CVST.data` object. `getN` returns the number of data points in the data set. `getSubset` returns a subset of the data as a `CVST.data` object, while `getX` just return the feature data. `shuffleData` returns a randomly shuffled instance of the data.

**Author(s)**

Tammo Krueger <tammokrueger@googlemail.com>
**Examples**

```r
nsine = noisySine(10)  
isClassification(nsine)  
isRegression(nsine)  
getN(nsine)  
getX(nsine)  
nsineShuffeled = shuffleData(nsine)  
getX(nsineShuffeled)  
getSubset(nsineShuffeled, 1:3)
```

---

**constructLearner**  
*Construction of Specific Learners for CVST*

**Description**

These methods construct a `CVST.learner` object suitable for the CVST method. These objects provide the common interface needed for the `CV` and `fastCV` methods. We provide kernel logistic regression, kernel ridge regression, support vector machines and support vector regression as fully functional implementation templates.

**Usage**

```r
constructLearner(learn, predict)  
constructKLogRegLearner()  
constructKRRLearner()  
constructSVM Learner()  
constructSVRLearner()
```

**Arguments**

- **learn**  
The learning methods which takes a `CVST.data` and list of parameters and return a model.

- **predict**  
The prediction method which takes a model and `CVST.data` and returns the corresponding predictions.

**Details**

The nu-SVM and nu-SVR are build on top the corresponding implementations of the `kernlab` package (see reference). In the list of parameters these implementations expect an entry named `kernel`, which gives the name of the kernel that should be used, an entry named `nu` specifying the nu parameter, and an entry named `C` giving the `C` parameter for the nu-SVR.

The KRR and KLR also expect `kernel` and necessary other parameters to construct the kernel. Both methods expect a lambda parameter and KLR additonally a `tol` and `maxiter` parameter in the parameter list.

Note that the lambda of KRR/KLR and the `C` parameter of SVR are scaled by the data set size to allow for comparable results in the fast CV loop.
**Value**

Returns a learner of type `CVST.learner` suitable for `CV` and `fastCV`.

**Author(s)**

Tammo Krueger <tammokrueger@googlemail.com>

**References**


**See Also**

`CV` `fastCV`

**Examples**

```r
# SVM
ns = noisySine(100)
svm = constructSVMLearner()
p = list(kernel="rbfdot", sigma=100, nu=.1)
m = svm$learn(ns, p)
nsTest = noisySine(1000)
pred = svm$predict(m, nsTest)
sum(pred != nsTest$y) / getN(nsTest)
# Kernel logistic regression
klr = constructKLogRegLearner()
p = list(kernel="rbfdot", sigma=100, lambda=1/getN(ns), tol=1e-6, maxiter=100)
m = klr$learn(ns, p)
pred = klr$predict(m, nsTest)
sum(pred != nsTest$y) / getN(nsTest)
# SVR
ns = noisySine(100)
svr = constructSVRLearner()
p = list(kernel="rbfdot", sigma=100, nu=.1, C=1*getN(ns))
m = svr$learn(ns, p)
nstest = noisySine(1000)
pred = svr$predict(m, nstest)
sum((pred - nstest$y)^2) / getN(nstest)
# Kernel ridge regression
krr = constructKRRLearner()
p = list(kernel="rbfdot", sigma=100, lambda=1/getN(ns))
m = krr$learn(ns, p)
pred = krr$predict(m, nstest)
sum((pred - nstest$y)^2) / getN(nstest)
```
constructParams  

Construct a Grid of Parameters

Description

This is a helper function which, given a named list of parameter choices, expand the complete grid and returns a CVST.params object suitable for CV and fastCV.

Usage

constructParams(...)

Arguments

...  
The parameters that should be expanded.

Value

Returns a CVST.params which is basically a named list of possible parameter values.

Author(s)

Tammo Krueger <tammokrueger@googlemail.com>

See Also

fastCV

Examples

params = constructParams(kernel="rbfdot", sigma=10^(-1:5), nu=c(0.1, 0.2))  
# the expanded grid contains 14 parameter lists:  
length(params)

constructSequentialTest  

Construct and Handle Sequential Tests.

Description

These functions handle the construction and calculation with sequential tests as introduced by Wald (1947). getCVSTTest constructs a special sequential test as introduced in Krueger (2011). testSequence test a sequence of 0/1 whether it is distributed according to H0 or H1.
Usage

constructSequentialTest(piH0 = 0.5, piH1 = 0.9, beta, alpha)
getCVSTest(steps, beta = 0.1, alpha = 0.01)
testSequence(st, s)
plotSequence(st, s)

Arguments

piH0 Probability of the binomial distribution for H0.
piH1 Probability of the binomial distribution for H1.
beta Significance level for H0.
alpha Significance level for H1.
steps Number of steps the CVST procedure should be executed.
st A sequential test of type CVST. sequentialTest.
s A sequence of 0/1 values.

Value

constructSequentialTest and getCVSTest return a CVST. sequentialTest with the specified properties. testSequence returns 1, if H1 can be expected, -1 if H0 can be accepted, and 0 if the test needs more data for a decision. plotSequence gives a graphical impression of the this testing procedure.

Author(s)

Tammo Krueger <tammokrueger@googlemail.com>

References


See Also

fastCV

Examples

st = getCVSTest(10)
s = rbinom(10, 1, 0.5)
plotSequence(st, s)
testSequence(st, s)
Perform a k-fold Cross-validation

Description
Performs the usual k-fold cross-validation procedure on a given data set, parameter grid and learner.

Usage
CV(data, learner, params, fold = 5, verbose = TRUE)

Arguments
- data: The data set as CVST.data object.
- learner: The learner as CVST.learner object.
- params: the parameter grid as CVST.params object.
- fold: The number of folds that should be generated for each set of parameters.
- verbose: Should the procedure report the performance for each model?

Value
Returns the optimal parameter settings as determined by k-fold cross-validation.

Author(s)
Tammo Krueger <tammokrueger@googlemail.com>

References

See Also
- fastCV
- constructData
- constructLearner
- constructParams

Examples
ns = noisySine(100)
svm = constructSVMLearner()
params = constructParams(kernel="rbfdot", sigma=10^(-3:3), nu=c(0.05, 0.1, 0.2, 0.3))
opt = CV(ns, svm, params)
The Fast Cross-Validation via Sequential Testing (CVST) Procedure

Description

CVST is an improved cross-validation procedure which uses non-parametric testing coupled with sequential analysis to determine the best parameter set on linearly increasing subsets of the data. By eliminating underperforming candidates quickly and keeping promising candidates as long as possible, the method speeds up the computation while preserving the capability of a full cross-validation.

Usage

fastCV(train, learner, params, setup, test = NULL, verbose = TRUE)

Arguments

- **train**: The data set as CVST.data object.
- **learner**: The learner as CVST.learner object.
- **params**: The parameter grid as CVST.params object.
- **setup**: A CVST.setup object containing the necessary parameter for the CVST procedure.
- **test**: An independent test set that should be used at each step. If NULL then the remaining data after learning a model at each step is used instead.
- **verbose**: Should the procedure report the performance after each step?

Value

Returns the optimal parameter settings as determined by fast cross-validation via sequential testing.

Author(s)

Tammo Krueger <tammokrueger@googlemail.com>

References


See Also

CV constructCVSTModel constructData constructLearner constructParams
Examples

```r
ns = noisySine(100)
svm = constructSVM Learner()
params = constructParams(kernel="rbfdot", sigma=10^(-3:3), nu=c(0.05, 0.1, 0.2, 0.3))
opt = fastCV(ns, svm, params, constructCVSTModel())
```

noisyDonoho  Generate Donoho's Toy Data Sets

Description

This function allows to generate noisy variants of the toy signals introduced by Donoho (see reference section). The scaling is chosen to reflect the setting as discussed in the original paper.

Usage

```r
noisyDonoho(n, fun = doppler, sigma = 1)
blocks(x, scale = 3.656993)
bumps(x, scale = 10.52884)
doppler(x, scale = 24.22172)
heavisine(x, scale = 2.356934)
```

Arguments

- `n`: Number of data points that should be generated.
- `fun`: Function to use to generate the data.
- `sigma`: Standard deviation of the noise component.
- `x`: Number of data points that should be generated.
- `scale`: Scaling parameter.

Value

Returns a data set of type CVST.data

Author(s)

Tammo Krueger <tammokrueger@googlemail.com>

References


See Also

`constructData`
noisySine

Examples

```r
bumpsSet = noisyDonoho(1000, fun=bumps)
plot(bumpsSet)
dopplerSet = noisyDonoho(1000, fun=doppler)
plot(dopplerSet)
```

---

noisySine  
*Regression and Classification Toy Data Set*

Description

Regression and Classification Toy Data Set based on the sine and sinc function.

Usage

```r
noisySine(n, dim = 5, sigma = 0.25)
noisySinc(n, dim = 2, sigma = 0.1)
```

Arguments

- `n` Number of data points that should be generated.
- `dim` Intrinsic dimensionality of the data set (see references for details).
- `sigma` Standard deviation of the noise component.

Value

Returns a data set of type CVST.data

Author(s)

Tammo Krueger <tammokrueger@googlemail.com>

References


See Also

`constructData`
Examples

nsine = noisySine(1000)
plot(nsine, col=nsine$y)
nsinc = noisySinc(1000)
plot(nsinc)
Index

*Topic **datasets**
  - noisyDonoho, 12
  - noisySine, 13

*Topic **package**
  - CVST-package, 2

blocks (noisyDonoho), 12
bumps (noisyDonoho), 12

cochranq.test, 3
constructCVSTModel, 4, 11
constructData, 5, 10–13
constructKlogRegLearner
  (constructLearner), 6
constructKRR Learner (constructLearner),
  6
constructLearner, 6, 10, 11
constructParams, 8, 10, 11
constructSequentialTest, 8
constructSVM Learner (constructLearner),
  6
constructSVR Learner (constructLearner),
  6
CV, 6–8, 10, 11
CVST (CVST-package), 2
CVST-package, 2
doppler (noisyDonoho), 12

fastCV, 4–10, 11

getCVSTTest (constructSequentialTest), 8
getN (constructData), 5
getSubset (constructData), 5
getX (constructData), 5

heavisine (noisyDonoho), 12

isClassification (constructData), 5
isRegression (constructData), 5

noisyDonoho, 12
noisySinc (noisySine), 13
noisySine, 13
plotSequence (constructSequentialTest), 8
shuffleData (constructData), 5
testSequence (constructSequentialTest), 8