Package ‘RcppDynProg’

July 24, 2019

Type Package
Title ‘Rcpp’ Dynamic Programming
Version 0.1.3
Date 2019-07-24

URL https://github.com/WinVector/RcppDynProg/,
              https://winvector.github.io/RcppDynProg/

BugReports https://github.com/WinVector/RcppDynProg/issues

Maintainer John Mount <jmount@win-vector.com>

Description Dynamic Programming implemented in ‘Rcpp’. Includes example partition and out
of sample fitting applications. Also supplies additional custom coders for the ‘vtreat’ package.

License GPL-2 | GPL-3
Depends R (>= 3.4.0)
Imports wrapr (>= 1.8.4), Rcpp (>= 1.0.0), utils, stats
LinkingTo Rcpp, RcppArmadillo
RoxygenNote 6.1.1
Suggests RUnit, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Author John Mount [aut, cre],
       Nina Zumel [aut],
       Win-Vector LLC [cph]

Repository CRAN

Date/Publication 2019-07-24 17:30:02 UTC
**R topics documented:**

- const_costs .......................................................... 2
- const_costs_logistic .............................................. 2
- lin_costs ............................................................ 3
- lin_costs_logistic .................................................. 4
- piecewise_constant .................................................. 4
- piecewise_constant_coder ......................................... 5
- piecewise_linear .................................................... 6
- piecewise_linear_coder ............................................ 6
- RcppDynProg .......................................................... 7
- run_RcppDynProg_tests ............................................ 7
- score_solution ....................................................... 8
- solve_for_partition ................................................ 9
- solve_for_partitionc ............................................... 10
- solve_interval_partition .......................................... 11
- solve_interval_partition_k ...................................... 12
- solve_interval_partition_no_k ................................... 13

---

**const_costs**

**Description**

Built matrix of total out of sample interval square error costs for held-out means. One indexed.

**Usage**

```r
const_costs(y, w, min_seg, indices)
```

**Arguments**

- `y` NumericVector, values to group in order.
- `w` NumericVector, weights.
- `min_seg` positive integer, minimum segment size.
- `indices` IntegerVector, order list of indices to pair.

**Value**

`xcosts` NumericMatrix, for j>=i `xcosts(i,j)` is the cost of partition element [i,...,j] (inclusive).

**Examples**

```r
const_costs(c(1, 1, 2, 2), c(1, 1, 1, 1), 1, 1:4)
```
const_costs_logistic

Description

Built matrix of interval logistic costs for held-out means. One indexed.

Usage

const_costs_logistic(y, w, min_seg, indices)

Arguments

y NumericVector, 0/1 values to group in order (should be in interval [0,1]).
w NumericVector, weights (should be positive).
min_seg positive integer, minimum segment size.
dindices IntegerVector, order list of indices to pair.

Value

xcosts NumericMatix, for j>=i xcosts(i,j) is the cost of partition element [i,...,j] (inclusive).

Examples

const_costs_logistic(c(0.1, 0.1, 0.2, 0.2), c(1, 1, 1, 1), 1, 1:4)

lin_costs

Description

Built matrix of interval costs for held-out linear models. One indexed.

Usage

lin_costs(x, y, w, min_seg, indices)

Arguments

x NumericVector, x-coords of values to group.
y NumericVector, values to group in order.
w NumericVector, weights.
min_seg positive integer, minimum segment size.
dindices IntegerVector, ordered list of indices to pair.
lin_costs_logistic

Value

xcosts NumericMatrix, for j>=i xcosts(i,j) is the cost of partition element [i,...,j] (inclusive).

Examples

lin_costs(c(1, 2, 3, 4), c(1, 2, 2, 1), c(1, 1, 1, 1), 1, 1:4)

lin_costs_logistic lin_costs_logistic deviance costs.

Description

Built matrix of interval deviance costs for held-out logistic models. Fits are evaluated in-sample. One indexed.

Usage

lin_costs_logistic(x, y, w, min_seg, indices)

Arguments

x NumericVector, x-coords of values to group.
y NumericVector, values to group in order (should be in interval [0,1]).
w NumericVector, weights (should be positive).
min_seg positive integer, minimum segment size.
indices IntegerVector, ordered list of indices to pair.

Value

xcosts NumericMatrix, for j>=i xcosts(i,j) is the cost of partition element [i,...,j] (inclusive).

Examples

lin_costs_logistic(c(1, 2, 3, 4, 5, 6, 7), c(0, 0, 1, 0, 1, 1, 0), c(1, 1, 1, 1, 1, 1, 1), 3,
piecewise_constant  

Piecewise constant fit.

Description

treat custom coder based on RcppDynProg::solve_for_partition().

Usage

```
piecewise_constant(varName, x, y, w = NULL)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>varName</td>
<td>character, name of variable to work on.</td>
</tr>
<tr>
<td>x</td>
<td>numeric, input values.</td>
</tr>
<tr>
<td>y</td>
<td>numeric, values to estimate.</td>
</tr>
<tr>
<td>w</td>
<td>numeric, weights.</td>
</tr>
</tbody>
</table>

Examples

```
piecewise_constant("x", 1:8, c(-1, -1, -1, -1, 1, 1, 1, 1))
```

piecewise_constant_coder  

Piecewise constant fit coder factory.

Description

Build a piecewise constant fit coder with some parameters bound in.

Usage

```
piecewise_constant_coder(penalty = 1, min_n_to_chunk = 1000, min_seg = 10, max_k = 1000)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>penalty</td>
<td>per-segment cost penalty.</td>
</tr>
<tr>
<td>min_n_to_chunk</td>
<td>minimum n to subdivied problem.</td>
</tr>
<tr>
<td>min_seg</td>
<td>positive integer, minimum segment size.</td>
</tr>
<tr>
<td>max_k</td>
<td>maximum segments to divide into.</td>
</tr>
</tbody>
</table>
Value

a vtreat coder

Examples

coder <- piecewise_constant_coder(min_seg = 1)
coder("x", 1:8, c(-1, -1, -1, -1, 1, 1, 1, 1))

---

piecewise_linear

Piecewise linear fit.

Description

vtreat custom coder based on RcppDynProg::solve_for_partition().

Usage

piecewise_linear(varName, x, y, w = NULL)

Arguments

varName character, name of variable to work on.
x numeric, input values.
y numeric, values to estimate.
w numeric, weights.

Examples

piecewise_linear("x", 1:8, c(1, 2, 3, 4, 4, 3, 2, 1))

---

piecewise_linear_coder

Piecewise linear fit coder factory.

Description

Build a piecewise linear fit coder with some parameters bound in.

Usage

piecewise_linear_coder(penalty = 1, min_n_to_chunk = 1000,
                        min_seg = 10, max_k = 1000)
Arguments

penalty per-segment cost penalty.
min_n_to_chunk minimum n to subdivided problem.
min_seg positive integer, minimum segment size.
max_k maximum segments to divide into.

Value

a vtreat coder

Examples

```r
coder <- piecewise_linear_coder(min_seg = 1)
coder("x", 1:8, c(1, 2, 3, 4, 4, 3, 2, 1))
```

Description

Rcpp dynamic programming solutions for partitioning and machine learning problems. Includes out of sample fitting applications. Also supplies additional custom coders for the vtreat package. Please see https://github.com/WinVector/RcppDynProg for details.

Author(s)

John Mount

Description

For all files with names of the form "^test_.+\R$" in the package directory unit_tests run all functions with names of the form "^test_.+$" as RUnit tests. Attaches RUnit and pkg, requires RUnit. Stops on error.
Usage

run_RcppDynProg_tests(..., verbose = TRUE,  
    package_test_dirs = "unit_tests", test_dirs = character(0),  
    stop_on_issue = TRUE, stop_if_no_tests = TRUE, 
    require_RUnit_attached = FALSE, require_pkg_attached = TRUE, 
    rngKind = "Mersenne-Twister", rngNormalKind = "Inversion")

Arguments

... not used, force later arguments to bind by name.
verbose logical, if TRUE print more.
package_test_dirs directory names to look for in the installed package.
test_dirs paths to look for tests in.
stop_on_issue logical, if TRUE stop after errors or failures.
stop_if_no_tests logical, if TRUE stop if no tests were found.
require_RUnit_attached logical, if TRUE require RUnit be attached before testing.
require_pkg_attached logical, if TRUE require pkg be attached before testing.
rngKind pseudo-random number generator method name.
rngNormalKind pseudo-random normal generator method name.

Details

Based on https://github.com/RcppCore/Rcpp/blob/master/tests/doRUnit. 
R. This version is GPL-3, works derived from it must be distributed GPL-3.

Value

RUnit test results (invisible).

score_solution compute the price of a partition solution (and check is valid).

Description

compute the price of a partition solution (and check is valid).

Usage

score_solution(x, solution)
**solve_for_partition**

**Arguments**

- **x**  
  Numeric matrix, for \( j \geq i \) \( x(i,j) \) is the cost of partition element \([i,..,j]\) (inclusive).
- **solution**  
  vector of indices

**Value**

- **price**

**Examples**

```r
x <- matrix(c(1,1,5,1,1,0,5,0,1), nrow=3)
s <- c(1, 2, 4)
score_solution(x, s)
```

---

**Description**

Solve for a good set of right-exclusive x-cuts such that the overall graph of \( y \sim x \) is well-approximated by a piecewise linear function. Solution is a ready for use with `base::findInterval()` and `stats::approx()` (demonstrated in the examples).

**Usage**

```r
solve_for_partition(x, y, ..., w = NULL, penalty = 0,
  min_n_to_chunk = 1000, min_seg = 1, max_k = length(x))
```

**Arguments**

- **x**  
  numeric, input variable (no NAs).
- **y**  
  numeric, result variable (no NAs, same length as x).
- **...**  
  not used, force later arguments by name.
- **w**  
  numeric, weights (no NAs, positive, same length as x).
- **penalty**  
  per-segment cost penalty.
- **min_n_to_chunk**  
  minimum n to subdivied problem.
- **min_seg**  
  positive integer, minimum segment size.
- **max_k**  
  maximum segments to divide into.

**Value**

a data frame appropriate for `stats::approx()`.
Examples

```r
# example data
d <- data.frame(
  x = 1:8,
  y = c(1, 2, 3, 4, 4, 3, 2, 1))

# solve for break points
soln <- solve_for_partition(d$x, d$y)
# show solution
print(soln)

# label each point
d$group <- base::findInterval(
  d$x,
  soln$x[which('left')]soln$what == 'left'])

# apply piecewise approximation
d$estimate <- stats::approx(
  soln$x,
  soln$pred,
  xout = d$x,
  method = 'linear',
  rule = 2)$y
# show result
print(d)
```

solve_for_partitionc

Solve for a piecewise constant partition.

Description

Solve for a good set of right-exclusive x-cuts such that the overall graph of y~x is well-approximated by a piecewise linear function. Solution is a ready for use with with `base::findInterval()` and `stats::approx()` (demonstrated in the examples).

Usage

`solve_for_partitionc(x, y, ..., w = NULL, penalty = 0,
  min_n_to_chunk = 1000, min_seg = 1, max_k = length(x))`

Arguments

- `x` numeric, input variable (no NAs).
- `y` numeric, result variable (no NAs, same length as x).
- `...` not used, force later arguments by name.
- `w` numeric, weights (no NAs, positive, same length as x).
solve_interval_partition

penalty per-segment cost penalty.
min_n_to_chunk minimum n to subdived problem.
min_seg positive integer, minimum segment size.
max_k maximum segments to divide into.

Value

a data frame appropriate for stats::approx().

Examples

# example data
d <- data.frame(
  x = 1:8,
  y = c(-1, -1, -1, -1, 1, 1, 1, 1))

# solve for break points
soln <- solve_for_partitionc(d$x, d$y)
# show solution
print(soln)

# label each point
d$group <- base::findInterval(
  d$x,
  soln$x[soln$what=='left'])
# apply piecewise approximation
d$estimate <- stats::approx(
  soln$x,
  soln$pred,
  xout = d$x,
  method = 'constant',
  rule = 2)$y
# show result
print(d)
solve_interval_partition

Usage

solve_interval_partition(x, kmax)

Arguments

x NumericMatrix, for j>=i x(i,j) is the cost of partition element [i,...,j] (inclusive).
kmax int, maximum number of segments in solution.

Value
dynamic program solution.

Examples

costs <- matrix(c(1.5, NA, NA, 1, 0, NA, 5, -1, 1), nrow = 3)
solve_interval_partition(costs, nrow(costs))

solve_interval_partition_k

solve_interval_partition_k interval partition problem with a bound on number of steps.

Description

Solve a for a minimal cost partition of the integers [1,...,nrow(x)] problem where for j>=i x(i,j) is the cost of choosing the partition element [i,...,j]. Returned solution is an ordered vector v of length k<=kmax where: v[1]==1, v[k]==nrow(x)+1, and the partition is of the form [v[i], v[i+1)) (intervals open on the right).

Usage

solve_interval_partition_k(x, kmax)

Arguments

x NumericMatrix, for j>=i x(i,j) is the cost of partition element [i,...,j] (inclusive).
kmax int, maximum number of segments in solution.

Value
dynamic program solution.
Examples

```
costs <- matrix(c(1.5, NA, NA, 1, 0, NA, 5, -1, 1), nrow = 3)
solve_interval_partition(costs, nrow(costs))
```

solve_interval_partition_no_k

interval partition problem, no bound on the number of steps.

Description

Not working yet.

Usage

```
solve_interval_partition_no_k(x)
```

Arguments

- `x`: NumericMatrix, for j\(\geq i\) \(x(i,j)\) is the cost of partition element \([i,\ldots,j]\) (inclusive).

Details

Solve a for a minimal cost partition of the integers \([1,\ldots,nrow(x)]\) problem where for \(j\geq i\) \(x(i,j)\) is the cost of choosing the partition element \([i,\ldots,j]\). Returned solution is an ordered vector \(v\) of length \(k\) where: \(v[1]=1\), \(v[k]=nrow(x)+1\), and the partition is of the form \([v[i], v[i+1]]\) (intervals open on the right).

Value

dynamic program solution.

Examples

```
costs <- matrix(c(1.5, NA, NA, 1, 0, NA, 5, -1, 1), nrow = 3)
solve_interval_partition(costs, nrow(costs))
```