# Package ‘penaltyLearning’

October 14, 2022

**Maintainer** Toby Dylan Hocking <toby.hocking@r-project.org>

**Author** Toby Dylan Hocking

**Version** 2020.5.13

**License** GPL-3

**Title** Penalty Learning

**Description** Implementations of algorithms from
Learning Sparse Penalties for Change-point Detection
using Max Margin Interval Regression, by
Hocking, Rigaill, Vert, Bach
<http://proceedings.mlr.press/v28/hocking13.html>
published in proceedings of ICML2013.

**Suggests** neuroblastoma, jointseg, testthat, future, future.apply,
directlabels (>= 2017.03.31)

**Depends** R (>= 2.10)

**URL** https://github.com/tdhock/penaltyLearning

**BugReports** https://github.com/tdhock/penaltyLearning/issues

**Imports** data.table (>= 1.9.8), ggplot2

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2020-05-14 16:20:02 UTC

## R topics documented:

---

`change.colors` *change colors*

---

## Description

character vector of change-point label colors, to be used with ggplot2::scale_*_manual

## Usage

```
"change.colors"
```

`change.labels` *change labels*

## Description

data.table of meta-data for label types.

## Usage

```
"change.labels"
```

`changeLabel` *changeLabel*

## Description

Describe an annotated region label for supervised change-point detection.

## Usage

```
changeLabel(annotation,
    min.changes, max.changes,
    color)
```

## Arguments

`annotation`

`min.changes`

`max.changes`

`color`

## Author(s)

Toby Dylan Hocking

## check_features_targets    *check features targets*

### Description

stop with an informative error if there is a problem with the feature or target matrix.

### Usage

```
check_features_targets(feature.mat,
    target.mat)
```

### Arguments

| | |
|---|---|
| `feature.mat` | n x p numeric input feature matrix. |
| `target.mat` | n x 2 matrix of target interval limits. |

### Value

number of observations/rows.

### Author(s)

Toby Dylan Hocking

## check_target_pred    *check target pred*

### Description

stop with an informative error if there are problems with the target matrix or predicted values.

### Usage

```
check_target_pred(target.mat,
    pred)
```

### Arguments

`target.mat`

`pred`

### Value

number of observations.

### Author(s)

Toby Dylan Hocking

---

`coef.IntervalRegression` *coef IntervalRegression*

---

### Description

Get the learned coefficients of an IntervalRegression model.

### Usage

```
## S3 method for class 'IntervalRegression'
coef(object,
    ...)
```

### Arguments

`object`

`...`

### Value

numeric matrix [features x regularizations] of learned weights (on the original feature scale), can be used for prediction via cbind(1,features) %*% weights.

### Author(s)

Toby Dylan Hocking

---

`demo8` *PeakSegFPOP demo data set*

---

### Description

PeakSegFPOP demo data set with 8 observations

### Usage

```
data("demo8")
```

### Format

A list of two objects: feature.mat is an 8 x 36 input feature matrix, and target.mat is a 8 x 2 output limit matrix.

`featureMatrix` *featureMatrix*

## Description

Compute a feature matrix (segmentation problems x features).

## Usage

```
featureMatrix(data.sequences,
    problem.vars, data.var)
```

## Arguments

| | |
|---|---|
| `data.sequences` | data.frame of sorted sequences of data to segment. |
| `problem.vars` | character vector of columns of `data.sequences` to treat as segmentation problem IDs. |
| `data.var` | character vector of length 1 (column of `data.sequences` to treat as data to segment). |

## Value

Numeric feature matrix. Some entries may be missing or infinite; these columns should be removed before model training.

## Author(s)

Toby Dylan Hocking

## Examples

```
test.df <- data.frame(
  id=rep(1:2, each=10),
  x=rnorm(20))
penaltyLearning::featureMatrix(test.df, "id", "x")
if(requireNamespace("neuroblastoma")){
  data(neuroblastoma, package="neuroblastoma", envir=environment())
  one <- subset(neuroblastoma$profiles, profile.id %in% c(1,2))
  f.mat <- penaltyLearning::featureMatrix(
    one, c("profile.id", "chromosome"), "logratio")
}
```

## featureVector *featureVector*

### Description

Compute a feature vector of constant length which can be used as an input for supervised penalty learning. The output is a target interval of log(penalty) values that achieve minimum incorrect labels (see `targetIntervals`).

### Usage

```
featureVector(data.vec)
```

### Arguments

| | |
|---|---|
| `data.vec` | numeric vector of ordered data. |

### Value

Numeric vector of features.

### Author(s)

Toby Dylan Hocking

### Examples

```
x <- rnorm(10)
penaltyLearning::featureVector(x)
if(requireNamespace("neuroblastoma")){
  data(neuroblastoma, package="neuroblastoma", envir=environment())
  one <- subset(neuroblastoma$profiles, profile.id=="1" & chromosome=="1")
  (f.vec <- penaltyLearning::featureVector(one$logratio))
}
```

## GeomTallRect *GeomTallRect*

### Description

ggproto object for `geom_tallrect`

### Usage

```
"GeomTallRect"
```

## geom_tallrect *geom tallrect*

### Description

ggplot2 geom with xmin and xmax aesthetics that covers the entire y range, useful for clickSelects background elements.

### Usage

```
geom_tallrect(mapping = NULL,
    data = NULL, stat = "identity",
    position = "identity",
    ..., na.rm = FALSE,
    show.legend = NA,
    inherit.aes = TRUE)
```

### Arguments

`mapping`

`data`

`stat`

`position`

`...`

`na.rm`

`show.legend`

`inherit.aes`

### Author(s)

Toby Dylan Hocking

## IntervalRegressionCV *IntervalRegressionCV*

### Description

Use cross-validation to fit an L1-regularized linear interval regression model by optimizing margin and/or regularization parameters. This function repeatedly calls `IntervalRegressionRegularized`, and by default assumes that margin=1. To optimize the margin, specify the `margin.vec` parameter manually, or use `IntervalRegressionCVmargin` (which takes more computation time but yields more accurate models). If the future package is available, two levels of future_lapply are used to parallelize on validation.fold and margin.

### Usage

```
IntervalRegressionCV(feature.mat,
    target.mat, n.folds = ifelse(nrow(feature.mat) <
        10, 3L, 5L),
    fold.vec = sample(rep(1:n.folds,
        l = nrow(feature.mat))),
    verbose = 0, min.observations = 10,
    reg.type = "min",
    incorrect.labels.db = NULL,
    initial.regularization = 0.001,
    margin.vec = 1, LAPPLY = NULL,
    check.unlogged = TRUE,
    ...)
```

### Arguments

`feature.mat` Numeric feature matrix, n observations x p features.

`target.mat` Numeric target matrix, n observations x 2 limits. These should be real-valued (possibly negative). If your data are interval censored positive-valued survival times, you need to log them to obtain `target.mat`.

`n.folds` Number of cross-validation folds.

`fold.vec` Integer vector of fold id numbers.

`verbose` numeric: 0 for silent, bigger numbers (1 or 2) for more output.

`min.observations`
stop with an error if there are fewer than this many observations.

`reg.type` Either "1sd" or "min" which specifies how the regularization parameter is chosen during the internal cross-validation loop. min: first take the mean of the K-CV error functions, then minimize it (this is the default since it tends to yield the least test error). 1sd: take the most regularized model with the same margin which is within one standard deviation of that minimum (this model is typically a bit less accurate, but much less complex, so better if you want to interpret the coefficients).

`incorrect.labels.db`
either NULL or a data.table, which specifies the error function to compute for selecting the regularization parameter on the validation set. NULL means to minimize the squared hinge loss, which measures how far the predicted log(penalty) values are from the target intervals. If a data.table is specified, its first key should correspond to the rownames of `feature.mat`, and columns min.log.lambda, max.log.lambda, fp, fn, possible.fp, possible.fn; these will be used with `ROChange` to compute the AUC for each regularization parameter, and the maximimum will be selected (in the plot this is negative.auc, which is minimized). This data.table can be computed via labelError(modelSelection(. . .),...)$model.errors – see example(`ROChange`). In practice this makes the computation longer, and it should only result in more accurate models if there are many labels per data sequence.

`initial.regularization`
Passed to `IntervalRegressionRegularized`.

| | |
|---|---|
| margin.vec | numeric vector of margin size hyper-parameters. The computation time is linear in the number of elements of margin.vec – more values takes more computation time, but yields slightly more accurate models (if there is enough data). |
| LAPPLY | Function to use for parallelization, by default future_lapply if it is available, otherwise lapply. For debugging with verbose>0 it is useful to specify LAPPLY=lapply in order to interactively see messages, before all parallel processes end. |
| check.unlogged | If TRUE, stop with an error if target matrix is non-negative and has any big difference in successive quantiles (this is an indicator that the user probably forgot to log their outputs). |
| ... | passed to IntervalRegressionRegularized. |

## Value

List representing regularized linear model.

## Author(s)

Toby Dylan Hocking

## Examples

```
if(interactive()){
  library(penaltyLearning)
  data("neuroblastomaProcessed", package="penaltyLearning", envir=environment())
  if(require(future)){
    plan(multiprocess)
  }
  set.seed(1)
  i.train <- 1:100
  fit <- with(neuroblastomaProcessed, IntervalRegressionCV(
    feature.mat[i.train,], target.mat[i.train,],
    verbose=0))
  ## When only features and target matrices are specified for
  ## training, the squared hinge loss is used as the metric to
  ## minimize on the validation set.
  plot(fit)
  ## Create an incorrect labels data.table (first key is same as
  ## rownames of feature.mat and target.mat).
  library(data.table)
  errors.per.model <- data.table(neuroblastomaProcessed$errors)
  errors.per.model[, pid.chr := paste0(profile.id, ".", chromosome)]
  setkey(errors.per.model, pid.chr)
  set.seed(1)
  fit <- with(neuroblastomaProcessed, IntervalRegressionCV(
    feature.mat[i.train,], target.mat[i.train,],
    ## The incorrect.labels.db argument is optional, but can be used if
    ## you want to use AUC as the CV model selection criterion.
    incorrect.labels.db=errors.per.model))
  plot(fit)
```

```
}
```

---

`IntervalRegressionCVmargin`
*IntervalRegressionCVmargin*

---

### Description

Use cross-validation to fit an L1-regularized linear interval regression model by optimizing both margin and regularization parameters. This function just calls `IntervalRegressionCV` with a margin.vec parameter that is computed based on the finite target interval limits. If default parameters are used, this function should be about 10 times slower than `IntervalRegressionCV` (since this function computes n.margin=10 models per regularization parameter whereas `IntervalRegressionCV` only computes one). On large (N > 1000 rows) data sets, this function should yield a model which is a little more accurate than `IntervalRegressionCV` (since the margin parameter is optimized).

### Usage

```
IntervalRegressionCVmargin(feature.mat,
    target.mat, log10.diff = 2,
    n.margin = 10L, ...)
```

### Arguments

| | |
|---|---|
| `feature.mat` | Numeric feature matrix, n observations x p features. |
| `target.mat` | Numeric target matrix, n observations x 2 limits. |
| `log10.diff` | Numeric scalar: factors of 10 below the largest finite limit difference to use as a minimum margin value (difference on the log10 scale which is used to generate margin parameters). Bigger values mean a grid of margin parameters with a larger range. For example if the largest finite limit in `target.mat` is 26 and the smallest finite limit is -4 then the largest limit difference is 30, which will be used as the maximum margin parameter. If `log10.diff` is the default of 2 then that means the smallest margin parameter will be 0.3 (two factors of 10 smaller than 30). |
| `n.margin` | Integer scalar: number of margin parameters, by default 10. |
| `...` | Passed to `IntervalRegressionCV`. |

### Value

Model fit list from `IntervalRegressionCV`.

### Author(s)

Toby Dylan Hocking

### Examples

```
if(interactive()){
  library(penaltyLearning)
  data(
    "neuroblastomaProcessed",
    package="penaltyLearning",
    envir=environment())
  if(require(future)){
    plan(multiprocess)
  }
  set.seed(1)
  fit <- with(neuroblastomaProcessed, IntervalRegressionCVmargin(
    feature.mat, target.mat, verbose=1))
  plot(fit)
  print(fit$plot.heatmap)
}
```

---

## IntervalRegressionInternal
*IntervalRegressionInternal*

---

### Description

Solve the squared hinge loss interval regression problem for one `regularization` parameter: w* = argmin_w L(w) + `regularization` * ||w||_1 where L(w) is the average squared hinge loss with respect to the `targets`, and ||w||_1 is the L1-norm of the weight vector (excluding the first element, which is the un-regularized intercept or bias term). This function performs no scaling of input `features`, and is meant for internal use only! To learn a regression model, try `IntervalRegressionCV` or `IntervalRegressionUnregularized`.

### Usage

```
IntervalRegressionInternal(features,
    targets, initial.param.vec,
    regularization, threshold = 0.001,
    max.iterations = 1000,
    weight.vec = NULL,
    Lipschitz = NULL,
    verbose = 2, margin = 1,
    biggest.crit = 100)
```

### Arguments

| | |
|---|---|
| `features` | Scaled numeric feature matrix (problems x `features`). The first column/feature should be all ones and will not be regularized. |
| `targets` | Numeric target matrix (problems x 2). |
| `initial.param.vec` | initial guess for weight vector (`features`). |

| | |
|---|---|
| `regularization` | Degree of L1-regularization. |
| `threshold` | When the stopping criterion gets below this `threshold`, the algorithm stops and declares the solution as optimal. |
| `max.iterations` | If the algorithm has not found an optimal solution after this many iterations, increase `Lipschitz` constant and `max.iterations`. |
| `weight.vec` | A numeric vector of weights for each training example. |
| `Lipschitz` | A numeric scalar or NULL, which means to compute `Lipschitz` as the mean of the squared L2-norms of the rows of the feature matrix. |
| `verbose` | Cat messages: for restarts and at the end if >= 1, and for every iteration if >= 2. |
| `margin` | Margin size hyper-parameter, default 1. |
| `biggest.crit` | Restart FISTA with a bigger `Lipschitz` (smaller step size) if crit gets larger than this. |

## Value

Numeric vector of scaled weights w of the affine function f_w(X) = X %*% w for a scaled feature matrix X with the first row entirely ones.

## Author(s)

Toby Dylan Hocking

---

`IntervalRegressionRegularized`

*IntervalRegressionRegularized*

---

## Description

Repeatedly use `IntervalRegressionInternal` to solve interval regression problems for a path of regularization parameters. This function does not perform automatic selection of the regularization parameter; instead, it returns regression models for a range of regularization parameters, and it is up to you to select which one to use. For automatic regularization parameter selection, use `IntervalRegressionCV`.

## Usage

```
IntervalRegressionRegularized(feature.mat,
    target.mat, initial.regularization = 0.001,
    factor.regularization = 1.2,
    verbose = 0, margin = 1,
    ...)
```

### Arguments

- `feature.mat` Numeric feature matrix.
- `target.mat` Numeric target matrix.
- `initial.regularization` Initial regularization parameter.
- `factor.regularization` Increase regularization by this factor after finding an optimal solution. Or NULL to compute just one model (`initial.regularization`).
- `verbose` Print messages if >= 1.
- `margin` Non-negative `margin` size parameter, default 1.
- `...` Other parameters to pass to `IntervalRegressionInternal`.

### Value

List representing fit model. You can do fit$predict(feature.matrix) to get a matrix of predicted log penalty values. The param.mat is the n.features * n.regularization numeric matrix of optimal coefficients (on the original scale).

### Author(s)

Toby Dylan Hocking

### Examples

```
if(interactive()){
  library(penaltyLearning)
  data("neuroblastomaProcessed", package="penaltyLearning", envir=environment())
  i.train <- 1:500
  fit <- with(neuroblastomaProcessed, IntervalRegressionRegularized(
    feature.mat[i.train,], target.mat[i.train,]))
  plot(fit)
}
```

---

`IntervalRegressionUnregularized`

## *IntervalRegressionUnregularized*

---

### Description

Use `IntervalRegressionRegularized` with initial.regularization=0 and factor.regularization=NULL, meaning fit one un-regularized interval regression model.

### Usage

```
IntervalRegressionUnregularized(...)
```

### Arguments

| | |
|---|---|
| ... | passed to `IntervalRegressionRegularized`. |

### Value

List representing fit model, see help(`IntervalRegressionRegularized`) for details.

### Author(s)

Toby Dylan Hocking

---

## labelError *Compute incorrect labels*

---

### Description

Compute incorrect `labels` for several change-point detection problems and `models`. Use this function after having computed changepoints, loss values, and model selection functions (see `modelSelection`). The next step after labelError is typically computing target intervals of log(penalty) values that predict changepoints with minimum incorrect `labels` for each problem (see `targetIntervals`).

### Usage

```
labelError(models, labels,
    changes, change.var = "chromStart",
    label.vars = c("min",
        "max"), model.vars = "n.segments",
    problem.vars = character(0),
    annotations = change.labels)
```

### Arguments

| | |
|---|---|
| models | data.frame with one row per (problem,model) combination, typically the output of modelSelection(...). There is a row for each changepoint model that could be selected for a particular segmentation problem. There should be columns `problem.vars` (for problem ID) and `model.vars` (for model complexity). |
| labels | data.frame with one row per (problem,region). Each label defines a region in a particular segmentation problem, and a range of predicted changepoints which are consistent in that region. There should be a column "annotation" with takes one of the corresponding values in the annotation column of `change.labels` (used to determine the range of predicted changepoints which are consistent). There should also be a columns `problem.vars` (for problem ID) and `label.vars` (for region start/end). |
| changes | data.frame with one row per (problem,model,change), for each predicted changepoint (in each model and segmentation problem). Should have columns `problem.vars` (for problem ID), `model.vars` (for model complexity), and `change.var` (for changepoint position). |

- `change.var` character(length=1): column name of predicted change-point position in `labels`. The default "chromStart" is useful for genomic data with segment start/end positions stored in columns named chromStart/chromEnd. A predicted changepoint at position X is interpreted to mean a changepoint between X and X+1.
- `label.vars` character(length=2): column names of start and end positions of `labels`, in same units as change-point positions. The default is c("min", "max"). Labeled regions are (start,end] – open on the left and closed on the right, so for example a 0changes annotation between start=10 and end=20 means that any predicted changepoint at 11, ..., 20 is a false positive.
- `model.vars` character: column names used to identify model complexity. The default "n.segments" is for change-point `models` such as in the jointseg and changepoint packages.
- `problem.vars` character: column names used to identify data set / segmentation problem, should be present in all three data tables (`models`, `labels`, `changes`).
- `annotations` data.table with columns annotation, min.changes, max.changes, possible.fn, possible.fp which is joined to `labels` in order to determine how to compute false positives and false negatives for each annotation.

## Value

list of two data.tables: label.errors has one row for every combination of `models` and `labels`, with status column that indicates whether or not that model commits an error in that particular label; model.errors has one row per model, with columns for computing target intervals and ROC curves (see `targetIntervals` and `ROChange`).

## Author(s)

Toby Dylan Hocking

## Examples

```
label <- function(annotation, min, max){
  data.frame(profile.id=4, chrom="chr14", min, max, annotation)
}
label.df <- rbind(
  label("1change", 70e6, 80e6),
  label("0changes", 20e6, 60e6))
model.df <- data.frame(chrom="chr14", n.segments=1:3)
change.df <- data.frame(chrom="chr14", rbind(
  data.frame(n.segments=2, changepoint=75e6),
  data.frame(n.segments=3, changepoint=c(75e6, 50e6))))
penaltyLearning::labelError(
  model.df, label.df, change.df,
  problem.vars="chrom", # for all three data sets.
  model.vars="n.segments", # for changes and selection.
  change.var="changepoint", # column of changes with breakpoint position.
  label.vars=c("min", "max")) # limit of labels in ann.
```

## largestContinuousMinimumC *largestContinuousMinimumC*

### Description

Find the run of minimum `cost` with the largest `size`. This function use a linear time C implementation, and is meant for internal use. Use `targetIntervals` for real data.

### Usage

```
largestContinuousMinimumC(cost,
    size)
```

### Arguments

| | |
|---|---|
| `cost` | numeric vector of `cost` values. |
| `size` | numeric vector of interval `size` values. |

### Value

Integer vector length 2 (start and end of target interval relative to `cost` and `size`).

### Author(s)

Toby Dylan Hocking

### Examples

```
library(penaltyLearning)
data(neuroblastomaProcessed, envir=environment())
one.problem.error <-
  neuroblastomaProcessed$errors[profile.id=="4" & chromosome=="1"]
indices <- one.problem.error[, largestContinuousMinimumC(
  errors, max.log.lambda-min.log.lambda)]
one.problem.error[indices[["start"]]:indices[["end"]],]
```

`largestContinuousMinimumR` *largestContinuousMinimumR*

## Description

Find the run of minimum `cost` with the largest `size`. This function uses a two pass R implementation, and is meant for internal use. Use `targetIntervals` for real data.

## Usage

```
largestContinuousMinimumR(cost,
    size)
```

## Arguments

| | |
|---|---|
| `cost` | numeric vector of `cost` values. |
| `size` | numeric vector of interval `size` values. |

## Value

Integer vector length 2 (start and end of target interval relative to `cost` and `size`).

## Author(s)

Toby Dylan Hocking

## Examples

```
library(penaltyLearning)
data(neuroblastomaProcessed, envir=environment())
one.problem.error <-
  neuroblastomaProcessed$errors[profile.id=="4" & chromosome=="1"]
indices <- one.problem.error[, largestContinuousMinimumR(
  errors, max.log.lambda-min.log.lambda)]
one.problem.error[indices[["start"]]:indices[["end"]],]
```

---

`modelSelection` *Compute exact model selection function*

---

## Description

Given loss.vec L_i, model.complexity K_i, the model selection function i*(lambda) = argmin_i L_i + lambda*K_i, compute all of the solutions (i, min.lambda, max.lambda) with i being the solution for every lambda in (min.lambda, max.lambda). Use this function after having computed changepoints and `loss` values for each model, and before using `labelError`. This function uses the linear time algorithm implemented in C code (`modelSelectionC`).

## Usage

```
modelSelection(models,
    loss = "loss", complexity = "complexity")
```

## Arguments

| | |
|---|---|
| `models` | data.frame with one row per model. There must be at least two columns models[[loss]] and models[[complexity]], but there can also be other meta-data columns. |
| `loss` | character: column name of `models` to interpret as `loss` L_i. |
| `complexity` | character: column name of `models` to interpret as `complexity` K_i. |

## Value

data.frame with a row for each model that can be selected for at least one lambda value, and the following columns. (min.lambda, max.lambda) and (min.log.lambda, max.log.lambda) are intervals of optimal penalty constants, on the original and log scale; the other columns (and rownames) are taken from `models`. This should be used as the `models` argument of `labelError`.

## Author(s)

Toby Dylan Hocking

---

`modelSelectionC` *Exact model selection function*

---

## Description

Given `loss.vec` L_i, `model.complexity` K_i, the model selection function i*(lambda) = argmin_i L_i + lambda*K_i, compute all of the solutions (i, min.lambda, max.lambda) with i being the solution for every lambda in (min.lambda, max.lambda). This function uses the linear time algorithm implemented in C code. This function is mostly meant for internal use – it is instead recommended to use `modelSelection`.

### Usage

```
modelSelectionC(loss.vec,
    model.complexity,
    model.id)
```

### Arguments

| | |
|---|---|
| `loss.vec` | numeric vector: loss L_i |
| `model.complexity` | numeric vector: model complexity K_i |
| `model.id` | vector: indices i |

### Value

data.frame with a row for each model that can be selected for at least one lambda value, and the following columns. (min.lambda, max.lambda) and (min.log.lambda, max.log.lambda) are intervals of optimal penalty constants, on the original and log scale; `model.complexity` are the K_i values; `model.id` are the model identifiers (also used for row names); and model.loss are the C_i values.

### Author(s)

Toby Dylan Hocking

### Examples

```
loss.vec <- c(
  -9.9, -12.8, -19.2, -22.1, -24.5, -26.1, -28.5, -30.1, -32.2,
  -33.7, -35.2, -36.8, -38.2, -39.5, -40.7, -41.8, -42.8, -43.9,
  -44.9, -45.8)
seg.vec <- seq_along(loss.vec)
exact.df <- penaltyLearning::modelSelectionC(loss.vec, seg.vec, seg.vec)
## Solve the optimization using grid search.
L.grid <- with(exact.df,{
  seq(min(max.log.lambda)-1,
      max(min.log.lambda)+1,
      l=100)
})
lambda.grid <- exp(L.grid)
kstar.grid <- sapply(lambda.grid, function(lambda){
  crit <- with(exact.df, model.complexity * lambda + model.loss)
  picked <- which.min(crit)
  exact.df$model.id[picked]
})
grid.df <- data.frame(log.lambda=L.grid, segments=kstar.grid)
library(ggplot2)
## Compare the results.
ggplot()+
  ggtitle("grid search (red) agrees with exact path computation (black)")+
  geom_segment(aes(min.log.lambda, model.id,
                   xend=max.log.lambda, yend=model.id),
```

```
              data=exact.df)+
  geom_point(aes(log.lambda, segments),
             data=grid.df, color="red", pch=1)+
  ylab("optimal model complexity (segments)")+
  xlab("log(lambda)")
```

## modelSelectionR *Exact model selection function*

### Description

Given `loss.vec` L_i, `model.complexity` K_i, the model selection function i*(lambda) = argmin_i L_i + lambda*K_i, compute all of the solutions (i, min.lambda, max.lambda) with i being the solution for every lambda in (min.lambda, max.lambda). This function uses the quadratic time algorithm implemented in R code. This function is mostly meant for internal use and comparison – it is instead recommended to use `modelSelection`.

### Usage

```
modelSelectionR(loss.vec,
    model.complexity,
    model.id)
```

### Arguments

| | |
|---|---|
| `loss.vec` | numeric vector: loss L_i |
| `model.complexity` | numeric vector: model complexity K_i |
| `model.id` | vector: indices i |

### Value

data.frame with a row for each model that can be selected for at least one lambda value, and the following columns. (min.lambda, max.lambda) and (min.log.lambda, max.log.lambda) are intervals of optimal penalty constants, on the original and log scale; `model.complexity` are the K_i values; `model.id` are the model identifiers (also used for row names); and model.loss are the C_i values.

### Author(s)

Toby Dylan Hocking

### Examples

```
loss.vec <- c(
  -9.9, -12.8, -19.2, -22.1, -24.5, -26.1, -28.5, -30.1, -32.2,
  -33.7, -35.2, -36.8, -38.2, -39.5, -40.7, -41.8, -42.8, -43.9,
  -44.9, -45.8)
seg.vec <- seq_along(loss.vec)
penaltyLearning::modelSelectionR(loss.vec, seg.vec, seg.vec)
```

---

## neuroblastomaProcessed *Processed neuroblastoma data set with features and targets*

---

### Description

Features are inputs and targets are outputs for penalty learning functions like penaltyLearning::IntervalRegressionCV. data(neuroblastoma, package="neuroblastoma") was processed by computing optimal Gaussian segmentation models from 1 to 20 segments (cghseg:::segmeanCO or Segmentor3IsBack::Segmentor), then label error was computed using neuroblastoma$annotations (penaltyLearning::labelError), then target intervals were computed (penaltyLearning::targetInterval). Features were also computed based on neuroblastoma$profiles.

### Usage

```
data("neuroblastomaProcessed")
```

### Format

List of two matrices: feature.mat is n.observations x n.features, and target.mat is n.observations x 2, where n.observations=3418 and n.features=117.

---

## notConverging *Interval regression problem that was not converging*

---

### Description

A small data set which was diverging using a previous implementation of IntervalRegressionCV.

### Usage

```
data("notConverging")
```

### Format

A list with names: X.mat are numeric inputs, y.mat are numeric outputs, fold.vec is an integer vector of fold ID numbers.

### Source

github.com/tdhock/neuroblastoma-data, data/H3K4me3_TDH_other/cv/equal_labels/testFolds/3/sampleSelectionGP_erf/5/c

---

`oneSkip` *oneSkip*

---

### Description

A loss and model complexity function which never selects one of the models, using a linear penalty.

### Usage

```
data("oneSkip")
```

### Format

A list of two data.frames (input and output).

### Source

example(exactModelSelection) in PeakSegDP package.

---

`plot.IntervalRegression` *plot IntervalRegression*

---

### Description

Plot an IntervalRegression model.

### Usage

```
## S3 method for class 'IntervalRegression'
plot(x,
    ...)
```

### Arguments

`x`

`...`

### Value

a ggplot.

### Author(s)

Toby Dylan Hocking

## predict.IntervalRegression

*predict IntervalRegression*

### Description

Compute model predictions.

### Usage

```
## S3 method for class 'IntervalRegression'
predict(object,
    X, ...)
```

### Arguments

`object`

`X`

`...`

### Value

numeric matrix of predicted log(penalty) values.

### Author(s)

Toby Dylan Hocking

## print.IntervalRegression

*print IntervalRegression*

### Description

print learned model parameters.

### Usage

```
## S3 method for class 'IntervalRegression'
print(x,
    ...)
```

### Arguments

`x`

`...`

## Author(s)

Toby Dylan Hocking

---

`ROChange` *ROC curve for changepoints*

---

## Description

Compute a Receiver Operating Characteristic curve for a penalty function.

## Usage

```
ROChange(models, predictions,
    problem.vars = character())
```

## Arguments

| | |
|---|---|
| `models` | data.frame describing the number of incorrect labels as a function of log(lambda), with columns min.log.lambda, max.log.lambda, fp, fn, possible.fp, possible.fn, etc. This can be computed via labelError(modelSelection(...), ...)$model.errors – see examples. |
| `predictions` | data.frame with a column named pred.log.lambda, the predicted log(penalty) value for each segmentation problem. |
| `problem.vars` | character: column names used to identify data set / segmentation problem. |

## Value

named list of results:

| | |
|---|---|
| `roc` | a data.table with one row for each point on the ROC curve |
| `thresholds` | two rows of roc which correspond to the predicted and minimal error thresholds |
| `auc.polygon` | a data.table with one row for each vertex of the polygon used to compute AUC |
| `auc` | numeric Area Under the ROC curve |
| `aum` | numeric Area Under Min(FP,FN) |
| `aum.grad` | data.table with one row for each prediction, and columns hi/lo bound for the aum generalized gradient. |

## Author(s)

Toby Dylan Hocking

## Examples

```
library(penaltyLearning)
library(data.table)

data(neuroblastomaProcessed, envir=environment())
## Get incorrect labels data for one profile.
pid <- 11
pro.errors <- neuroblastomaProcessed$errors[
  profile.id==pid][order(chromosome, min.log.lambda)]
dcast(pro.errors, n.segments ~ chromosome, value.var="errors")
## Get the feature that corresponds to the BIC penalty = log(n),
## meaning log(penalty) = log(log(n)).
chr.vec <- paste(c(1:4, 11, 17))
pid.names <- paste0(pid, ".", chr.vec)
BIC.feature <- neuroblastomaProcessed$feature.mat[pid.names, "log2.n"]
pred <- data.table(pred.log.lambda=BIC.feature, chromosome=chr.vec)
## edit one prediction so that it ends up having the same threshold
## as another one, to illustrate an aum sub-differential with
## un-equal lo/hi bounds.
err.changes <- pro.errors[, {
  .SD[c(NA, diff(errors) != 0), .(min.log.lambda)]
}, by=chromosome]
(ch.vec <- err.changes[, structure(min.log.lambda, names=chromosome)])
other <- "11"
(diff.other <- ch.vec[[other]]-pred[other, pred.log.lambda, on=.(chromosome)])
pred["1", pred.log.lambda := ch.vec[["1"]]-diff.other, on=.(chromosome)]
pred["4", pred.log.lambda := 2, on=.(chromosome)]
ch.vec[["1"]]-pred["1", pred.log.lambda, on=.(chromosome)]
result <- ROChange(pro.errors, pred, "chromosome")
library(ggplot2)
## Plot the ROC curves.
ggplot()+
  geom_path(aes(FPR, TPR), data=result$roc)+
  geom_point(aes(FPR, TPR, color=threshold), data=result$thresholds, shape=1)

## Plot the number of incorrect labels as a function of threshold.
ggplot()+
  geom_segment(aes(
    min.thresh, errors,
    xend=max.thresh, yend=errors),
    data=result$roc)+
  geom_point(aes((min.thresh+max.thresh)/2, errors, color=threshold),
             data=result$thresholds,
             shape=1)+
  xlab("log(penalty) constant added to BIC penalty")

## Plot area under Min(FP,FN).
err.colors <- c(
  "fp"="red",
  "fn"="deepskyblue",
  "min.fp.fn"="black")
```

```
err.sizes <- c(
  "fp"=3,
  "fn"=2,
  "min.fp.fn"=1)
roc.tall <- melt(result$roc, measure.vars=names(err.colors))
area.rects <- data.table(
  chromosome="total",
  result$roc[0<min.fp.fn])
(gg.total <- ggplot()+
   geom_vline(
     xintercept=0,
     color="grey")+
   geom_rect(aes(
     xmin=min.thresh, xmax=max.thresh,
     ymin=0, ymax=min.fp.fn),
     data=area.rects,
     alpha=0.5)+
   geom_text(aes(
     min.thresh, min.fp.fn/2,
     label=sprintf(
       "Area Under Min(FP,FN)=%.3f ",
       result$aum)),
     data=area.rects[1],
     hjust=1,
     color="grey50")+
   geom_segment(aes(
     min.thresh, value,
     xend=max.thresh, yend=value,
     color=variable, size=variable),
     data=data.table(chromosome="total", roc.tall))+
   scale_size_manual(values=err.sizes)+
   scale_color_manual(values=err.colors)+
   theme_bw()+
   theme(panel.grid.minor=element_blank())+
   scale_x_continuous(
     "Prediction threshold")+
   scale_y_continuous(
     "Incorrectly predicted labels",
     breaks=0:10))

## Add individual error curves.
tall.errors <- melt(
  pro.errors[pred, on=.(chromosome)],
  measure.vars=c("fp", "fn"))
gg.total+
  geom_segment(aes(
    min.log.lambda-pred.log.lambda, value,
    xend=max.log.lambda-pred.log.lambda, yend=value,
    size=variable, color=variable),
    data=tall.errors)+
  facet_grid(chromosome ~ ., scales="free", space="free")+
  theme(panel.spacing=grid::unit(0, "lines"))+
  geom_blank(aes(
```

```
      0, errors),
      data=data.table(errors=c(1.5, -0.5)))

  print(result$aum.grad)
  if(interactive()){#this can be too long for CRAN.
    ## Plot how Area Under Min(FP,FN) changes with each predicted value.
    aum.dt <- pred[, {
      data.table(log.pen=seq(0, 4, by=0.5))[, {
        chr <- paste(chromosome)
        new.pred.dt <- data.table(pred)
        new.pred.dt[chr, pred.log.lambda := log.pen, on=.(chromosome)]
        with(
          ROChange(pro.errors, new.pred.dt, "chromosome"),
          data.table(aum))
      }, by=log.pen]
    }, by=chromosome]
    bounds.dt <- melt(
      result$aum.grad,
      measure.vars=c("lo", "hi"),
      variable.name="bound",
      value.name="slope")[pred, on=.(chromosome)]
    bounds.dt[, intercept := result$aum-slope*pred.log.lambda]
    ggplot()+
      geom_abline(aes(
        slope=slope, intercept=intercept),
        size=1,
        data=bounds.dt)+
      geom_text(aes(
        2, 2, label=sprintf("directional derivatives = [%d, %d]", lo, hi)),
        data=result$aum.grad)+
      scale_color_manual(
        values=c(
          predicted="red",
          new="black"))+
      geom_point(aes(
        log.pen, aum, color=type),
        data=data.table(type="new", aum.dt))+
      geom_point(aes(
        pred.log.lambda, result$aum, color=type),
        shape=1,
        data=data.table(type="predicted", pred))+
      theme_bw()+
      theme(panel.spacing=grid::unit(0, "lines"))+
      facet_wrap("chromosome", labeller=label_both)+
      coord_equal()+
      xlab("New log(penalty) value for chromosome")+
      ylab("Area Under Min(FP,FN)
using new log(penalty) for this chromosome
and predicted log(penalty) for others")
  }
```

## `squared.hinge` *squared hinge*

### Description

The squared hinge loss.

### Usage

```
squared.hinge(x, e = 1)
```

### Arguments

`x`

`e`

### Author(s)

Toby Dylan Hocking

## `targetIntervalResidual` *targetIntervalResidual*

### Description

Compute residual of predicted penalties with respect to target intervals. This function is useful for visualizing the errors in a plot of log(penalty) versus a feature.

### Usage

```
targetIntervalResidual(target.mat,
    pred)
```

### Arguments

| | |
|---|---|
| `target.mat` | n x 2 numeric matrix: target intervals of log(penalty) values that yield minimal incorrect labels. |
| `pred` | numeric vector: predicted log(penalty) values. |

### Value

numeric vector of n residuals. Predictions that are too high (above target.mat[,2]) get positive residuals (too few changepoints), and predictions that are too low (below target.mat[,1]) get negative residuals.

### Author(s)

Toby Dylan Hocking

### Examples

```
library(penaltyLearning)
library(data.table)
data(neuroblastomaProcessed, envir=environment())
## The BIC model selection criterion is lambda = log(n), where n is
## the number of data points to segment. This implies log(lambda) =
## log(log(n)), which is the log2.n feature.
row.name.vec <- grep(
  "^(4|520)[.]",
  rownames(neuroblastomaProcessed$feature.mat),
  value=TRUE)
feature.mat <- neuroblastomaProcessed$feature.mat[row.name.vec, ]
target.mat <- neuroblastomaProcessed$target.mat[row.name.vec, ]
pred.dt <- data.table(
  row.name=row.name.vec,
  target.mat,
  feature.mat[, "log2.n", drop=FALSE])
pred.dt[, pred.log.lambda := log2.n ]
pred.dt[, residual := targetIntervalResidual(
  cbind(min.L, max.L),
  pred.log.lambda)]
library(ggplot2)
limits.dt <- pred.dt[, data.table(
  log2.n,
  log.penalty=c(min.L, max.L),
  limit=rep(c("min", "max"), each=.N))][is.finite(log.penalty)]
ggplot()+
  geom_abline(slope=1, intercept=0)+
  geom_point(aes(
    log2.n,
    log.penalty,
    fill=limit),
    data=limits.dt,
    shape=21)+
  geom_segment(aes(
    log2.n, pred.log.lambda,
    xend=log2.n, yend=pred.log.lambda-residual),
    data=pred.dt,
    color="red")+
  scale_fill_manual(values=c(min="white", max="black"))
```

---

`targetIntervalROC` *targetIntervalROC*

---

### Description

Compute a ROC curve using a target interval matrix. A prediction less than the lower limit is considered a false positive (penalty too small, too many changes), and a prediction greater than the upper limit is a false negative (penalty too large, too few changes). WARNING: this ROC curve is less detailed than the one you get from `ROChange`! Use `ROChange` if possible.

### Usage

```
targetIntervalROC(target.mat,
    pred)
```

### Arguments

| | |
|---|---|
| `target.mat` | n x 2 numeric matrix: target intervals of log(penalty) values that yield minimal incorrect labels. |
| `pred` | numeric vector: predicted log(penalty) values. |

### Value

list describing ROC curves, same as `ROChange`.

### Author(s)

Toby Dylan Hocking

### Examples

```
library(penaltyLearning)
library(data.table)
data(neuroblastomaProcessed, envir=environment())

pid.vec <- c("1", "4")
chr <- 2
incorrect.labels <-
  neuroblastomaProcessed$errors[profile.id%in%pid.vec & chromosome==chr]
pid.chr <- paste0(pid.vec, ".", chr)
target.mat <- neuroblastomaProcessed$target.mat[pid.chr, , drop=FALSE]
pred.dt <- data.table(profile.id=pid.vec, pred.log.lambda=1.5)
roc.list <- list(
  labels=ROChange(incorrect.labels, pred.dt, "profile.id"),
  targets=targetIntervalROC(target.mat, pred.dt$pred.log.lambda))

err <- data.table(incorrect=names(roc.list))[, {
  roc.list[[incorrect]]$roc
}, by=incorrect]
library(ggplot2)
ggplot()+
  ggtitle("incorrect targets is an approximation of incorrect labels")+
  scale_size_manual(values=c(labels=2, targets=1))+
  geom_segment(aes(
```

```
        min.thresh, errors,
        color=incorrect,
        size=incorrect,
        xend=max.thresh, yend=errors),
                 data=err)
```

---

`targetIntervals` *Compute target intervals*

---

## Description

Compute target intervals of log(penalty) values that result in predicted changepoint `models` with minimum incorrect labels. Use this function after `labelError`, and before IntervalRegression*.

## Usage

```
targetIntervals(models,
    problem.vars)
```

## Arguments

| | |
|---|---|
| `models` | data.table with columns errors, min.log.lambda, max.log.lambda, typically labelError()$model.errors. |
| `problem.vars` | character: column names used to identify data set / segmentation problem. |

## Value

data.table with columns `problem.vars`, one row for each segmentation problem. The "min.log.lambda", and "max.log.lambda" columns give the largest interval of log(penalty) values which results in the minimum incorrect labels for that problem. This can be used to create the target.mat parameter of the IntervalRegression* functions.

## Author(s)

Toby Dylan Hocking

## Examples

```
library(penaltyLearning)
data(neuroblastomaProcessed, envir=environment())
targets.dt <- targetIntervals(
  neuroblastomaProcessed$errors,
  problem.vars=c("profile.id", "chromosome"))
```

`theme_no_space` *theme no space*

## Description

ggplot2 theme element for no space between panels.

## Usage

```
theme_no_space(...)
```

## Arguments

`...`

## Author(s)

Toby Dylan Hocking

# Index