Package 'MTLR'

May 7, 2026

Type Package

Title Survival Prediction with Multi-Task Logistic Regression

Version 0.2.1

Author Humza Haider

Maintainer Humza Haider <hshaider@ualberta.ca>

URL https://github.com/haiderstats/MTLR

BugReports https://github.com/haiderstats/MTLR/issues

Description An implementation of Multi-Task Logistic Regression (MTLR) for R.
This package is based on the method proposed by Yu et al. (2011) which utilized MTLR for generating individual survival curves
by learning feature weights which vary across time. This model was further extended to account for left and interval censored data.

License GPL-2 | file LICENSE

Encoding UTF-8

LazyData true

Imports Rcpp (>= 0.12.18), survival (>= 2.4.0)

Suggests ggplot2 (>= 3.0.0), reshape2 (>= 1.4.3), testthat, vdiffr (>= 0.3.0), covr, knitr, rmarkdown

Depends R (>= 3.4.0)

RoxygenNote 6.1.1

LinkingTo Rcpp, RcppArmadillo

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2019-06-03 21:30:03 UTC

Contents

`create_folds` *Create folds for cross-validation.*

Description

Create the test folds for k-fold cross validation. These cross-validation types differ from typical stratified cross-validation as this function also considers the range of event times in the data.

Usage

```
create_folds(time, delta, nfolds, foldtype = c("fullstrat",
  "censorstrat", "random"))
```

Arguments

`time`	a vector of event times.
`delta`	a vector of indicators for uncensored/censored data. The type of censoring here is not considered so it is suggested this function not be used for data with mixed censoring types. The specific indicator value does not matter as long as censored and uncensored observations have different values for their indicator.
`nfolds`	The number of folds to create.
`foldtype`	type of cross validation folds. Full stratification, "fullstrat", sorts observations by their event time and their event indicators and numbers them off into folds. This effectively give each fold approximately the same number of uncensored observations as well as keeps the range of time points as equivalent as possible across folds. This type of cross-validation is completely deterministic. Censored stratification, "censorstrat", will put approximately the same number of uncensored observations in each fold but not pay any attention to event time. This is partially stochastic. The totally random cross-validation, "random", randomly assigns observations to folds without considering event time nor event status.

Value

a list of size nfolds where each list component contains the indices of the test data for each fold.

See Also

mtlr_cv

---

mtlr *Train a Multi-Task Logistic Regression (MTLR) Model*

Description

Trains a MTLR model for survival prediction. Right, left, and interval censored data are all supported.

Usage

```
mtlr(formula, data, time_points = NULL, nintervals = NULL,
  normalize = T, C1 = 1, train_biases = T, train_uncensored = T,
  seed_weights = NULL, threshold = 1e-05, maxit = 5000,
  lower = -15, upper = 15)
```

Arguments

formula	a formula object with the response to the left of the "~" operator. The response must be a survival object returned by the Surv function.
data	a data.frame containing the features for survival prediction. These must be variables corresponding to the formula object.
time_points	the time points for MTLR to create weights. If left as NULL, the time_points chosen will be based on equally spaced quantiles of the survival times. In the case of interval censored data note that only the start time is considered and not the end time for selecting time points. It is strongly recommended to specify time points if your data is heavily interval censored. If time_points is not NULL then nintervals is ignored.
nintervals	Number of time intervals to use for MTLR. Note the number of time points will be nintervals + 1. If left as NULL a default of sqrt(N) is used where N is the number of observations in the supplied dataset. This parameter is ignored if time_points is specified.
normalize	if TRUE, variables will be normalized (mean 0, standard deviation of 1). This is STRONGLY suggested. If normalization does not occur it is much more likely that MTLR will fail to converge. Additionally, if FALSE consider adjusting "lower" and "upper" used for L-BFGS-B optimization.
C1	The L2 regularization parameter for MTLR. C1 can also be selected via mtlr_cv. See "Learning Patient-Specific Cancer Survival Distributions as a Sequence of Dependent Regressors" by Yu et al. (2011) for details.
train_biases	if TRUE, biases will be trained before feature weights (and again trained while training feature weights). This has shown to speed up total training time.

`train_uncensored`
: if TRUE, one round of training will occur assuming all event times are uncensored. This is done due to the non-convexity issue that arises in the presence of censored data. However if ALL data is censored we recommend setting this option to FALSE as it has shown to give poor results in this case.

`seed_weights`
: the initialization weights for the biases and the features. If left as NULL all weights are initialized to zero. If seed_weights are specified then either nintervals or time_points must also be specified. The length of seed_weights should correspond to (number of features + 1)*(length of time_points) = (number of features + 1)*(nintervals + 1).

`threshold`
: The threshold for the convergence tolerance (in the objective function) when training the feature weights. This threshold will be passed to optim.

`maxit`
: The maximum iterations to run for MTLR. This parameter will be passed to optim.

`lower`
: The lower bound for L-BFGS-B optimization. This parameter will be passed to optim.

`upper`
: The upper bound for L-BFGS-B optimization. This parameter will be passed to optim.

Details

This function allows one to train an MTLR model given a dataset containing survival data. mtlr uses the Limited-Memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS-B) approximation method to train feature weights. This training is outsourced to the internal optim function in R. Currently only a few parameters (namely threshold, maxit,lower, upper) of optim are supported, more will likely become available in the future.

Weights are initialized to 0 prior to training. Under default settings, the bias weights will be trained before considering feature weights. As Yu et al. (2011) specified, the introduction of censored observations creates a non-convex loss function. To address this, weights are first trained assuming all event times are *uncensored*. Once these starting weights have been trained another round of training is performed using the true values of the event indicator (censored/uncensored). However, in the event of all censored data this has shown to negatively effect the results. If all data is censored (either left, right, or interval2) we suggest setting train_uncensored = FALSE.

Yu et al. (2011) actually suggested two regularization parameters, C1 to control the size of the feature weights and C2 to control the smoothness. In Ping Jin's masters thesis (Using Survival Prediction Techniques to Learn Consumer-Specific Reservation Price Distributions) he showed that C2 is not required for smoothness and C1 will suffice (Appendix A.2) so we do not support the C2 parameter in this implementation.

If an error occurs from optim it is likely the weights are getting too large. Including fewer time points (or specifying better time points) in addition to changing the lower/upper bounds of L-BFGS-B may resolve these issues. The most common failure has been that the objective value sees infinite values due to extremely large feature weights.

Censored data: Right, left, and interval censored data are all supported both separately and mixed. The convention to input these types of data follows the Surv object format. Per the Surv documentation, "The [interval2] approach is to think of each observation as a time interval with (-infinity, t) for left censored, (t, infinity) for right censored, (t,t) for exact and (t1, t2) for an interval. This is the

approach used for type = interval2. Infinite values can be represented either by actual infinity (Inf) or NA." See the examples below for an example of inputting this type of data.

Value

An mtlr object returns the following:

- weight_matrix: The matrix of feature weights determined by MTLR.
- x: The dataframe of features (response removed). Note observations with missing values will have been removed (this is the dataset on which MTLR was trained).
- y: The matrix of response values MTLR uses for training. Each column corresponds to an observation and rows as time points. A value of 1 indicates a observation was either censored or had their event occur by that time.
- response: The response as a Surv object (specified by formula).
- time_points: The timepoints selected and used to train MTLR.
- C1: The regularization parameter used.
- Call: The original call to mtlr.
- Terms: The x-value terms used in mtlr. These are later used in `predict.mtlr`
- scale: The means and standard deviations of features when normalize = TRUE. These are used in `predict.mtlr`. Will be NULL if normalize = FALSE.
- xlevels: The levels of the features used. This is used again by `predict.mtlr`.

See Also

`predict.mtlr` `mtlr_cv` `plot.mtlr` `plotcurves`

Examples

```
#Access the Surv function and the leukemia/lung dataset.
library(survival)
simple_mod <- mtlr(Surv(time,status)~., data = leukemia)
simple_mod

bigger_mod <- mtlr(Surv(time,status)~., data = lung)
bigger_mod

#Note that observations with missing data were removed:
nrow(lung)
nrow(bigger_mod$x)


# Mixed censoring types
time1 = c(NA, 4, 7, 12, 10, 6, NA, 3) #NA for right censored
time2 = c(14, 4, 10, 12, NA, 9, 5, NA) #NA for left censored
#time1 == time2 indicates an exact death time. time2> time1 indicates interval censored.
set.seed(42)
dat = cbind.data.frame(time1, time2, importantfeature = rnorm(8))
formula = Surv(time1,time2,type = "interval2")~.
mixedmod = mtlr(formula, dat)
```

mtlr_cv *MTLR Internal Cross-Validation for Selecting C1.*

Description

MTLR Internal Cross-Validation for Selecting C1.

Usage

```
mtlr_cv(formula, data, time_points = NULL, nintervals = NULL,
  normalize = T, C1_vec = c(0.001, 0.01, 0.1, 1, 10, 100, 1000),
  train_biases = T, train_uncensored = T, seed_weights = NULL,
  previous_weights = T, loss = c("ll", "concordance"), nfolds = 5,
  foldtype = c("fullstrat", "censorstrat", "random"), verbose = FALSE,
  threshold = 1e-05, maxit = 5000, lower = -15, upper = 15)
```

Arguments

formula	a formula object with the response to the left of the "~" operator. The response must be a survival object returned by the Surv function.
data	a data.frame containing the features for survival prediction. These must be variables corresponding to the formula object.
time_points	the time points for MTLR to create weights. If left as NULL, the time_points chosen will be based on equally spaced quantiles of the survival times. In the case of interval censored data note that only the start time is considered and not the end time for selecting time points. It is strongly recommended to specify time points if your data is heavily interval censored. If time_points is not NULL then nintervals is ignored.
nintervals	Number of time intervals to use for MTLR. Note the number of time points will be nintervals + 1. If left as NULL a default of sqrt(N) is used where N is the number of observations in the supplied dataset. This parameter is ignored if time_points is specified.
normalize	if TRUE, variables will be normalized (mean 0, standard deviation of 1). This is STRONGLY suggested. If normalization does not occur it is much more likely that MTLR will fail to converge. Additionally, if FALSE consider adjusting "lower" and "upper" used for L-BFGS-B optimization.
C1_vec	a vector of regularization parameters to test. All values must be non-negative. For large datasets you may want to reduce the number of value tried to increase efficiency. Similarly for nfolds.
train_biases	if TRUE, biases will be trained before feature weights (and again trained while training feature weights). This has shown to speed up total training time.
train_uncensored	if TRUE, one round of training will occur assuming all event times are uncensored. This is done due to the non-convexity issue that arises in the presence of censored data. However if ALL data is censored we recommend setting this option to FALSE as it has shown to give poor results in this case.

`seed_weights` the initialization weights for the biases and the features. If left as NULL all weights are initialized to zero. If seed_weights are specified then either nintervals or time_points must also be specified. The length of seed_weights should correspond to (number of features + 1)*(length of time_points) = (number of features + 1)*(nintervals + 1).

`previous_weights` a boolean specifying if sequential folds should use the previous fold's parameters as seed_weights. Doing this will likely speed up the computation time for cross-validation as we are providing weights which are (likely) close to the optimal weights. Note that this is done separately for each value of C1 so there is no parameter sharing between different values of C1, and instead only across the same value of C1.

`loss` a string indicating the loss to optimize for which to choose the regularization parameter. Currently one can optimize for the log-likelihood ("ll") or concordance ("concordance"). See details regarding these losses.

`nfolds` the number of internal cross validation folds, default is 5.

`foldtype` type of cross validation folds. Full stratification, "fullstrat", sorts observations by their event time and their event indicators and numbers them off into folds. This effectively give each fold approximately the same number of uncensored observations as well as keeps the range of time points as equivalent as possible across folds. This type of cross-validation is completely deterministic. Censored stratification, "censorstrat", will put approximately the same number of uncensored observations in each fold but not pay any attention to event time. This is partially stochastic. The totally random cross-validation, "random", randomly assigns observations to folds without considering event time nor event status.

`verbose` if TRUE the progress will be printed for every completed value of C1.

`threshold` The threshold for the convergence tolerance (in the objective function) when training the feature weights. This threshold will be passed to optim.

`maxit` The maximum iterations to run for MTLR. This parameter will be passed to optim.

`lower` The lower bound for L-BFGS-B optimization. This parameter will be passed to optim.

`upper` The upper bound for L-BFGS-B optimization. This parameter will be passed to optim.

Details

The log-likelihood loss and concordance are supported for optimizing C1. Here the log-likelihood loss considers censored and uncensored observations differently. For uncensored observations, we assign a loss of the negative log probability assigned to the interval in which the observation had their event, *e.g.* if an observation had a 20 is -log(0.2). We want these probabilities to be large so we would normally want to maximize this value (since logs of probabilities are negative) but we take the negative and instead minimize the value, thus we want the lowest loss. For censored observations we take the log of the probability of survival at the time of censoring, *e.g.* if an observation is censored at time = 42 we take the negative log of the survival probability assigned to time 42 as the loss.

For the concordance loss, C1 is chosen to maximize the overall concordance when using the negative median as the "risk" score. This is completed using survConcordance in the survival package.

Value

Performing mtlr_cv will return the following:

- best_C1: The value of C1 which achieved the best (lowest) loss.
- avg_loss: The averaged value of loss across the five folds for each value of C1 tested.

See Also

mtlr

Examples

```
library(survival)
cv_mod <- mtlr_cv(Surv(time,status)~., data = lung)
#Note the best C1 also corresponds to the lost average loss:
cv_mod
```

---

plot.mtlr *Graphical Representation of Feature Weights*

Description

Plot the weights of an mtlr object. If packages ggplot2 and reshape2 are not installed, a bargraph of feature *influence* is given where influence is defined as the sum of absolute values of the feature weights across time. If ggplot2 and reshape2 are installed then a plot of feature weight across time is given.

Usage

```
## S3 method for class 'mtlr'
plot(x, numfeatures = 5, featurenames = c(), digits,
  ...)
```

Arguments

x	an object of class mtlr (result from calling mtlr).
numfeatures	the number of weight to plot. Default is 5. The most influential features are chosen first.
featurenames	the names of the specific weight to plot. These should correspond to the names in x$weight_matrix. If featurenames are supplied, then numfeatures is ignored.
digits	the number of digits to round to for the value of the time points.
...	for future methods

Examples

```
#These examples are geared towards users who have installed ggplot2 and reshape2.
library(survival)
mod <- mtlr(Surv(time,status)~., data = lung)
#Basic plot with 5 most influential features
plot(mod)
#Plot all 8 features
plot(mod, numfeatures = 8)
#Suppose we want to see specifically the "meal.cal" and "ph.karno" features:
plot(mod, featurenames = c("meal.cal", "ph.karno"))
```

`plotcurves` *Graphically Visualize MTLR Survival Curves*

Description

Plot the survival curves returned from predict.mtlr. Users must have packages ggplot2 and reshape2 installed in order to use this function. Survival curves for MTLR are smoothed using a monotonic cubic spline using a Hyman filtering between time points. For details regarding this smoothing function see `splinefun`.

Usage

```
plotcurves(curves, index = 1, color = c(), xlim = c(),
  remove_legend = TRUE)
```

Arguments

`curves`	survival curves formatted the same as those from predict.mtlr. Time points must be in the first column of the matrix followed by columns representing survival probabilities for each observation.
`index`	the index of the observation to plot. Here an index of 1 will refer to the second column of the curves object. If over 15 indices are given the legend will be removed as to not take up plotting space. To avoid this behavior set remove_legend = FALSE.
`color`	the color of the plotted survival curve. The length of color must match the length of index.
`xlim`	the limits of the x-axis (must be a 2 length vector).
`remove_legend`	if TRUE the legend will be removed if over 15 indices are supplied. If FALSE the legend will remain, however be aware that the legend may take up lots of space.

See Also

`mtlr` `predict.mtlr`

Examples

```
#Set up the example:
library(survival)
mod <- mtlr(Surv(time,status)~., data = lung)
curves <- predict(mod, type = "survivalcurve")

plotcurves(curves, 1:10)
plotcurves(curves, 1:3, color = c("red","blue","purple"))
plotcurves(curves, 1:10, xlim = c(0,42))

#Note the legend is now gone:
plotcurves(curves, 1:20)

#and it is back again:
plotcurves(curves, 1:20, remove_legend = FALSE)
```

`predict.mtlr` *Predictions for MTLR*

Description

Compute survival curves and other fitted values for a model generated by `mtlr`.

Usage

```
## S3 method for class 'mtlr'
predict(object, newdata, type = c("survivalcurve",
  "prob_times", "prob_event", "mean_time", "median_time"), add_zero = T,
  times = c(), ...)
```

Arguments

`object`	an object of class mtlr, generated by the mtlr.
`newdata`	an optional new dataframe for which to perform predictions using MTLR. If left empty, predictions will be performed using the dataset used to generate the original mtlr object – note that any error calculation on these predictions will be optimistic since this will only be the resubstitution error and not be representative of error on a new test set.
`type`	the type of prediction desired. Options are the survival curve for the time points selected by mtlr ("survivalcurve"), the survival curve for given times ("prob_times"), the probability of survival at the observations event time ("prob_event"), the mean survival time ("mean_time"), and the median survival time ("median_time"). For "survivalcurve" and "prob_times", the first column of the matrix returned will correspond to the time points and all other columns will be the observations survival probability at those associated time points. The index of a (row)

observation in newdata will correspond to the *ith + 1* column of the returned matrix.

If "prob_event" is chosen the response (event time) is required. For both "prob_event" and "prob_times", if the event time is larger than all of the time points used to build the mtlr model then the last (lowest) probability is used. For example, if the event time is 100 but the largest time point estimated by the mtlr model was 80 then the survival probability at 100 is equal to the survival probability at 80, *i.e. S(100) = S(80)*.

For "mean_time", if survival curves do not extend to zero survival probability a linear extension is added (a linear line from (time = 0,probability = 1) to (time = ?, probability =0)). This is the same for "median_time" except the line need only extend to survival probability = 0.5. A mean/median survival time of Inf is returned for survival curves with all survival probabilities of 1.

`add_zero` if TRUE, a time point of "0" and a survival probability of "1" will be added to all survival curves. Additionally, if add_zero is TRUE, type = "mean_time" will represent the average survival time overall but if FALSE, then "mean_time" will be reduced by roughly the value of the first time point. However, "median_time" and "prob_event" will be unchanged.

`times` For prediction method "prob_times" you may specify the times at which to predict the survival probability for each row in newdata. This values defaults to all unique event times (both censored and uncensored) in the data on which the model was trained.

`...` for future methods.

Value

The desired prediction type (a matrix or vector of predictions).

Note

The predictions generated by type = "survivalcurve" can be plotted using `plotcurves` – packages ggplot2 and reshape2 must be installed to use this function.

See Also

`mtlr` `plotcurves`

Examples

```
library(survival)
mod <- mtlr(Surv(time,status)~., data = lung)

#Here our predictions are on the data from which we trained so our results will be optimistic
# since they are produced from resubstitution as opposed to some new test set.
predict(mod, type = "survivalcurve")
predict(mod, type = "prob_event")
predict(mod, type = "median_time")
predict(mod, type = "mean_time")
```

```
#Notice the difference of about 59:
predict(mod, type = "mean_time", add_zero = FALSE)
```

`print.mtlr` *Printing an MTLR object.*

Description

Print an object created by `mtlr`.

Usage

```
## S3 method for class 'mtlr'
print(x, digits = max(options()$digits - 4, 3), ...)
```

Arguments

`x`	an object of class mtlr (result from calling `mtlr`).
`digits`	The number of digits to print mtlr weights.
`...`	for future methods.

Value

Call, the original call to the mtlr function. Time points, the time points selected by the mtlr model. Weights, the weights of each feature across time – rows represent each time point and each column corresponds to a feature.

See Also

`mtlr`

Index