Package ‘ordinalNet’

January 10, 2020

Type Package

Title Penalized Ordinal Regression

Version 2.7

Description Fits ordinal regression models with elastic net penalty. Supported model families include cumulative probability, stopping ratio, continuation ratio, and adjacent category. These families are a subset of vector glm's which belong to a model class we call the elementwise link multinomial-ordinal (ELMO) class. Each family in this class links a vector of covariates to a vector of class probabilities. Each of these families has a parallel form, which is appropriate for ordinal response data, as well as a nonparallel form that is appropriate for an unordered categorical response, or as a more flexible model for ordinal data. The parallel model has a single set of coefficients, whereas the nonparallel model has a set of coefficients for each response category except the baseline category. It is also possible to fit a model with both parallel and nonparallel terms, which we call the semi-parallel model. The semi-parallel model has the flexibility of the nonparallel model, but the elastic net penalty shrinks it toward the parallel model. For details, refer to Wurm, Hanlon, and Rathouz (2017) <arXiv:1706.05003>.

LazyData TRUE

License MIT + file LICENSE

Imports stats, graphics

Suggests testthat (>= 1.0.2), MASS (>= 7.3-45), glmnet (>= 2.0-5), penalized (>= 0.9-50), glmnetcr (>= 1.0.3), VGAM (>= 1.0-3), rms (>= 5.1-0)

RoxygenNote 7.0.2

NeedsCompilation yes

Author Michael Wurm [aut, cre], Paul Rathouz [aut], Bret Hanlon [aut]

Maintainer Michael Wurm <wurm@uwalumni.com>

Repository CRAN

Date/Publication 2020-01-10 15:00:03 UTC
R topics documented:

- coef.ordinalNet
- ordinalNet
- ordinalNetCV
- ordinalNetTune
- plot.ordinalNetTune
- predict.ordinalNet
- print.ordinalNet
- print.ordinalNetCV
- print.ordinalNetTune
- summary.ordinalNet
- summary.ordinalNetCV
- summary.ordinalNetTune

Index

| coef.ordinalNet | Method to extract fitted coefficients from an "ordinalNet" object. |

Description

Method to extract fitted coefficients from an "ordinalNet" object.

Usage

```r
## S3 method for class 'ordinalNet'
coef(
  object, 
  matrix = FALSE,
  whichLambda = NULL,
  criteria = c("aic", "bic"),
  ... 
)
```

Arguments

- **object**: An "ordinalNet" S3 object.
- **matrix**: Logical. If TRUE, coefficient estimates are returned in matrix form. Otherwise a vector is returned.
- **whichLambda**: Optional index number of the desired lambda within the sequence of lambdaVals. By default, the solution with the best AIC is returned.
- **criteria**: Selects the best lambda value by AIC or BIC. Only used if whichLambda=NULL.
- **...**: Not used. Additional coef arguments.

Value

The object returned depends on matrix.
 ordinalNet

See Also

 ordinalNet

Examples

# See ordinalNet() documentation for examples.

---

**ordinalNet**

**Ordinal regression models with elastic net penalty**

**Description**

Fits ordinal regression models with elastic net penalty by coordinate descent. Supported model families include cumulative probability, stopping ratio, continuation ratio, and adjacent category. These families are a subset of vector glm’s which belong to a model class we call the elementwise link multinomial-ordinal (ELMO) class. Each family in this class links a vector of covariates to a vector of class probabilities. Each of these families has a parallel form, which is appropriate for ordinal response data, as well as a nonparallel form that is appropriate for an unordered categorical response, or as a more flexible model for ordinal data. The parallel model has a single set of coefficients, whereas the nonparallel model has a set of coefficients for each response category except the baseline category. It is also possible to fit a model with both parallel and nonparallel terms, which we call the semi-parallel model. The semi-parallel model has the flexibility of the nonparallel model, but the elastic net penalty shrinks it toward the parallel model.

**Usage**

```r
ordinalNet(
x,
y,
alpha = 1,
standardize = TRUE,
penaltyFactors = NULL,
positiveID = NULL,
family = c("cumulative", "sratio", "cratio", "acat"),
reverse = FALSE,
link = c("logit", "probit", "cloglog", "cauchit"),
customLink = NULL,
parallelTerms = TRUE,
nonparallelTerms = FALSE,
parallelPenaltyFactor = 1,
lambdaVals = NULL,
nLambda = 20,
lambdaMinRatio = 0.01,
includeLambda0 = FALSE,
alphaMin = 0.01,
pMin = 1e-08,
```

stopThresh = 1e-08,
threshOut = 1e-08,
threshIn = 1e-08,
maxiterOut = 100,
maxiterIn = 100,
printIter = FALSE,
printBeta = FALSE,
warn = TRUE,
keepTrainingData = TRUE
)

Arguments

x  Covariate matrix. It is recommended that categorical covariates are converted
to a set of indicator variables with a variable for each category (i.e. no baseline
category); otherwise the choice of baseline category will affect the model fit.

y  Response variable. Can be a factor, ordered factor, or a matrix where each row is
a multinomial vector of counts. A weighted fit can be obtained using the matrix
option, since the row sums are essentially observation weights. Non-integer
matrix entries are allowed.

alpha  The elastic net mixing parameter, with $0 \leq \alpha \leq 1$. $\alpha=1$
corresponds to the lasso penalty, and $\alpha=0$ corresponds to the ridge penalty.

standardize  If standardize=TRUE, the predictor variables are scaled to have unit variance.
Coefficient estimates are returned on the original scale.

penaltyFactors  Optional nonnegative vector of penalty factors with length equal to the number
of columns in x. If this argument is used, then the penalty for each variable is
scaled by its corresponding factor. If NULL, the penalty factor is one for each
coefficient.

positiveID  Logical vector indicating whether each coefficient should be constrained to be
non-negative. If NULL, the default value is FALSE for all coefficients.

family  Specifies the type of model family. Options are "cumulative" for cumulative
probability, "sratio" for stopping ratio, "cratio" for continuation ratio, and "acat"
for adjacent category.

reverse  Logical. If TRUE, then the "backward" form of the model is fit, i.e. the model
is defined with response categories in reverse order. For example, the reverse
cumulative model with $K+1$ response categories applies the link function to the
cumulative probabilities $P(Y \geq 2), \ldots, P(Y \geq K+1)$, rather then $P(Y \leq
1), \ldots, P(Y \leq K)$.

link  Specifies the link function. The options supported are logit, probit, complementary
log-log, and cauchit. Only used if customLink=NULL.

customLink  Optional list containing a vectorized link function $g$, a vectorized inverse link
$h$, and the Jacobian function of the inverse link getQ. The Jacobian should be
defined as $\partial h(\eta) / \partial \eta^T$ (as opposed to the transpose of this matrix).

parallelTerms  Logical. If TRUE, then parallel coefficient terms will be included in the model.
parallelTerms and nonparallelTerms cannot both be FALSE.
nonparallelTerms
Logical. if TRUE, then nonparallel coefficient terms will be included in the
model. parallelTerms and nonparallelTerms cannot both be FALSE.

parallelPenaltyFactor
Nonnegative numeric value equal to one by default. The penalty on all parallel
terms is scaled by this factor (as well as variable-specific penaltyFactors).
Only used if parallelTerms=TRUE.

lambdaVals
An optional user-specified lambda sequence (vector). If NULL, a sequence will be
generated based on nLambda and lambdaMinRatio. In this case, the maximum
lambda is the smallest value that sets all penalized coefficients to zero, and the
minimum lambda is the maximum value multiplied by the factor lambdaMinRatio.

nLambda
Positive integer. The number of lambda values in the solution path. Only used
if lambdaVals=NULL.

lambdaMinRatio
A factor greater than zero and less than one. Only used if lambdaVals=NULL.

includeLambda0
Logical. If TRUE, then zero is added to the end of the sequence of lambdaVals.
This is not done by default because it can significantly increase computational
time. An unpenalized saturated model may have infinite coefficient solutions, in
which case the fitting algorithm will still terminate when the relative change in
log-likelihood becomes small. Only used if lambdaVals=NULL.

alphaMin
max(alpha, alphaMin) is used to calculate the starting lambda value when lambdaVals=NULL.
In this case, the default lambda sequence begins with the smallest lambda value
such that all penalized coefficients are set to zero (i.e. the value where the first
penalized coefficient enters the solution path). The purpose of this argument is
to help select a starting value for the lambda sequence when alpha = 0, because
otherwise it would be infinite. Note that alphaMin is only used to determine the
default lambda sequence and that the model is always fit using alpha to calculate
the penalty.

pMin
Value greater than zero, but much less than one. During the optimization routine,
the Fisher information is calculated using fitted probabilities. For this calculation,
fitted probabilities are capped below by this value to prevent numerical
instability.

stopThresh
In the relative log-likelihood change between successive lambda values falls
below this threshold, then the last model fit is used for all remaining lambda.

threshOut
Convergence threshold for the coordinate descent outer loop. The optimization
routine terminates when the relative change in the penalized log-likelihood be-
tween successive iterations falls below this threshold. It is recommended to set
threshOut equal to threshIn.

threshIn
Convergence threshold for the coordinate descent inner loop. Each iteration con-
ists of a single loop through each coefficient. The inner loop terminates when
the relative change in the penalized approximate log-likelihood between success-
ive iterations falls below this threshold. It is recommended to set threshOut
equal to threshIn.

maxiterOut
Maximum number of outer loop iterations.

maxiterIn
Maximum number of inner loop iterations.

printIter
Logical. If TRUE, the optimization routine progress is printed to the terminal.
The `ordinalNet` function fits regression models for a categorical response variable with $K + 1$ levels. Conditional on the covariate vector $x_i$ (the $i^{th}$ row of $x$), each observation has a vector of $K + 1$ class probabilities $(p_{i1}, ..., p_{i(K+1)})$. These probabilities sum to one, and can therefore be parametrized by $p_i = (p_{i1}, ..., p_{iK})$. The probabilities are mapped to a set of $K$ quantities $\delta_i = (\delta_{i1}, ..., \delta_{iK}) \in (0, 1)^K$, which depends on the choice of model family. The elementwise link function maps $\delta_i$ to a set of $K$ linear predictors. Together, the family and link specify a link function between $p_i$ and $\eta_i$.

**Model families:**

Let $Y$ denote the random response variable for a single observation, conditional on the covariates values of the observation. The random variable $Y$ is discrete with support $\{1, ..., K + 1\}$. The following model families are defined according to these mappings between the class probabilities and the values $\delta_1, ..., \delta_K$:

- **Cumulative probability** $\delta_j = P(Y \leq j)$
- **Reverse cumulative probability** $\delta_j = P(Y \geq j + 1)$
- **Stopping ratio** $\delta_j = P(Y = j | Y \geq j)$
- **Reverse stopping ratio** $\delta_j = P(Y = j + 1 | Y \leq j + 1)$
- **Continuation ratio** $\delta_j = P(Y > j | Y \geq j)$
- **Reverse continuation ratio** $\delta_j = P(Y = j + 1 | Y \leq j)$
- **Adjacent category** $\delta_j = P(Y = j | j \leq Y \leq j + 1)$
- **Reverse adjacent category** $\delta_j = P(Y = j | j + 1 \leq Y \leq j + 1)$

**Parallel, nonparallel, and semi-parallel model forms:**

Models within each of these families can take one of three forms, which have different definitions for the linear predictor $\eta_i$. Suppose each $x_i$ has length $P$. Let $b$ be a length $P$ vector of regression coefficients. Let $B$ be a $P \times K$ matrix of regression coefficient. Let $b_0$ be a vector of $K$ intercept terms. The three model forms are the following:

- **Parallel** $\eta_i = b_0 + b^T x_i$ (parallelTerms=TRUE, nonparallelTerms=FALSE)
- **Nonparallel** $\eta_i = b_0 + B^T x_i$ (parallelTerms=FALSE, nonparallelTerms=TRUE)
- **Semi-parallel** $\eta_i = b_0 + b^T x_i + B^T x_i$ (parallelTerms=TRUE, nonparallelTerms=TRUE)
The parallel form has the defining property of ordinal models, which is that a single linear combination $b^T x_i$ shifts the cumulative class probabilities $P(Y \leq j)$ in favor of either higher or lower categories. The linear predictors are parallel because they only differ by their intercepts ($b_0$). The nonparallel form is a more flexible model, and it does not shift the cumulative probabilities together. The semi-parallel model is equivalent to the nonparallel model, but the elastic net penalty shrinks the semi-parallel coefficients toward a common value (i.e. the parallel model), as well as shrinking all coefficients toward zero. The nonparallel model, on the other hand, simply shrinks all coefficients toward zero. When the response categories are ordinal, any of the three model forms could be applied. When the response categories are unordered, only the nonparallel model is appropriate.

**Elastic net penalty:**

The elastic net penalty is defined for each model form as follows. $\lambda$ and $\alpha$ are the usual elastic net tuning parameters, where $\lambda$ determines the degree to which coefficients are shrunk toward zero, and $\alpha$ specifies the amount of weight given to the L1 norm and squared L2 norm penalties. Each covariate is allowed a unique penalty factor $c_j$, which is specified with the penaltyFactors argument. By default $c_j = 1$ for all $j$. The semi-parallel model has a tuning parameter $\rho$ which determines the degree to which the parallel coefficients are penalized. Small values of $\rho$ will result in a fit closer to the parallel model, and large values of $\rho$ will result in a fit closer to the nonparallel model.

**Parallel**

$$\lambda \sum_{j=1}^{P} c_j \{ \alpha |b_j| + \frac{1}{2} (1 - \alpha) b_j^2 \}$$

**Nonparallel**

$$\lambda \sum_{j=1}^{P} c_j \{ \sum_{k=1}^{K} \alpha |B_{jk}| + \frac{1}{2} (1 - \alpha) B_{jk}^2 \}$$

**Semi-parallel**

$$\lambda \sum_{j=1}^{P} c_j \{ \alpha |b_j| + \frac{1}{2} (1 - \alpha) b_j^2 \} + \sum_{j=1}^{P} c_j \{ \sum_{k=1}^{K} \alpha |B_{jk}| + \frac{1}{2} (1 - \alpha) B_{jk}^2 \}$$

`ordinalNet` minimizes the following objective function. Let $N$ be the number of observations, which is defined as the sum of the $y$ elements when $y$ is a matrix.

$$\text{objective} = -1/N \ast \loglik + \text{penalty}$$

**Value**

An object with S3 class "ordinalNet". Model fit information can be accessed through the `coef`, `predict`, and `summary` methods.

- **coefs** Matrix of coefficient estimates, with each row corresponding to a lambda value. (If covariates were scaled with `standardize=TRUE`, the coefficients are returned on the original scale).
- **lambdaVals** Sequence of lambda values. If user passed a sequence to the `lambdaVals` argument, then it is this sequence. If `lambdaVals` argument was `NULL`, then it is the sequence generated.
- **loglik** Log-likelihood of each model fit.
- **nNonzero** Number of nonzero coefficients of each model fit, including intercepts.
- **aic** AIC, defined as $-2 \ast \loglik + 2 \ast nNonzero$.
- **bic** BIC, defined as $-2 \ast \loglik + \log(N) \ast nNonzero$.
- **devPct** Percentage deviance explained, defined as $1 - \loglik / loglik_0$, where $loglik_0$ is the log-likelihood of the null model.
- **iterOut** Number of coordinate descent outer loop iterations until convergence for each lambda value.
- **iterIn** Number of coordinate descent inner loop iterations on last outer loop for each lambda value.
**dif** Relative improvement in objective function on last outer loop for each lambda value. Can be used to diagnose convergence issues. If `iterOut` reached `maxiterOut` and `dif` is large, then `maxiterOut` should be increased. If `dif` is negative, this means the objective did not improve between successive iterations. This usually only occurs when the model is saturated and/or close to convergence, so a small negative value is not of concern. (When this happens, the algorithm is terminated for the current lambda value, and the coefficient estimates from the previous outer loop iteration are returned.)

**nLev** Number of response categories.

**nVar** Number of covariates in `x`.

**xNames** Covariate names.

**args** List of arguments passed to the `ordinalNet` function.

### Examples

```r
# Simulate x as independent standard normal
# Simulate y|x from a parallel cumulative logit (proportional odds) model
set.seed(1)

n <- 50
intercepts <- c(-1, 1)
beta <- c(1, 1, 0, 0, 0)
ncat <- length(intercepts) + 1  # number of response categories
p <- length(beta)  # number of covariates
x <- matrix(rnorm(n*p), ncol=p)  # n x p covariate matrix
eta <- c(x %*% beta) + matrix(intercepts, nrow=n, ncol=ncat-1, byrow=TRUE)
invlogit <- function(x) 1 / (1+exp(-x))
cumprob <- t(apply(eta, 1, invlogit))
prob <- cbind(cumprob, 1) - cbind(0, cumprob)
yint <- apply(prob, 1, function(p) sample(1:ncat, size=1, prob=p))
y <- as.factor(yint)

# Fit parallel cumulative logit model
fit1 <- ordinalNet(x, y, family="cumulative", link="logit",
                   parallelTerms=TRUE, nonparallelTerms=FALSE)
summary(fit1)
coef(fit1)
coef(fit1, matrix=TRUE)
predict(fit1, type="response")
predict(fit1, type="class")

# Fit nonparallel cumulative logit model
fit2 <- ordinalNet(x, y, family="cumulative", link="logit",
                   parallelTerms=FALSE, nonparallelTerms=TRUE)
fit2
coef(fit2)
coef(fit2, matrix=TRUE)
predict(fit2, type="response")
predict(fit2, type="class")

# Fit semi-parallel cumulative logit model (with both parallel and nonparallel terms)
fit3 <- ordinalNet(x, y, family="cumulative", link="logit",
                   parallelTerms=TRUE, nonparallelTerms=TRUE)
```


fit3
coef(fit3)
coef(fit3, matrix=TRUE)
predict(fit3, type="response")
predict(fit3, type="class")

ordinalNetCV

Uses K-fold cross validation to obtain out-of-sample log-likelihood
and misclassification rates. Lambda is tuned within each cross val-
idation fold.

Description

The data is divided into K folds. ordinalNet is fit K times, each time leaving out one fold as a
test set. For each of the K model fits, lambda can be tuned by AIC or BIC, or cross validation. If
cross validation is used, the user can choose whether to use the best average out-of-sample log-
likelihood, misclassification rate, Brier score, or percentage of deviance explained. The user can
also choose the number of cross validation folds to use for tuning. Once the model is tuned, the out
of sample log-likelihood, misclassification rate, Brier score, and percentage of deviance explained
are calculated on the held out test set.

Usage

ordinalNetCV(
  x,
  y,
  lambdaVals = NULL,
  folds = NULL,
  nFolds = 5,
  nFoldsCV = 5,
  tuneMethod = c("cvLoglik", "cvMisclass", "cvBrier", "cvDevPct", "aic", "bic"),
  printProgress = TRUE,
  warn = TRUE,
  ...
)

Arguments

x
  Covariate matrix.

y
  Response variable. Can be a factor, ordered factor, or a matrix where each row is
  a multinomial vector of counts. A weighted fit can be obtained using the matrix
  option, since the row sums are essentially observation weights. Non-integer
  matrix entries are allowed.

lambdaVals
  An optional user-specified lambda sequence (vector). If NULL, a sequence will
  be generated using the model fit to the full training data. This default se-
  quence is based on nLambda and lambdaMinRatio, which can be passed as
additional arguments (otherwise ordinalNet default values are used). The maximum lambda is the smallest value that sets all penalized coefficients to zero, and the minimum lambda is the maximum value multiplied by the factor lambdaMinRatio.

folds
An optional list, where each element is a vector of row indices corresponding to a different cross validation fold. Indices correspond to rows of the x matrix. Each index number should be used in exactly one fold. If NULL, the data will be randomly divided into equally-sized partitions. It is recommended to call set.seed before calling ordinalNetCV for reproducibility.

nFolds
Number of cross validation folds. Only used if folds=NULL.

nFoldsCV
Number of cross validation folds used to tune lambda for each training set (i.e. within each training fold). Only used if tuneMethod is "cvLoglik", "cvMiscalss", "cvBrier", or "cvDevPct.

tuneMethod
Method used to tune lambda for each training set (i.e. within each training fold). The "cvLoglik", "cvMiscalss", "cvBrier", and "cvDevPct" methods use cross validation with nFoldsCV folds and select the lambda value with the best average out-of-sample performance. The "aic" and "bic" methods are less computationally intensive because they do not require the model to be fit multiple times. Note that for the methods that require cross validation, the fold splits are determined randomly and cannot be specified by the user. The set.seed() function should be called prior to ordinalNetCV for reproducibility.

printProgress
Logical. If TRUE the fitting progress is printed to the terminal.

warn
Logical. If TRUE, the following warning message is displayed when fitting a cumulative probability model with nonparallelTerms=TRUE (i.e. nonparallel or semi-parallel model). "Warning message: For out-of-sample data, the cumulative probability model with nonparallelTerms=TRUE may predict cumulative probabilities that are not monotone increasing." The warning is displayed by default, but the user may wish to disable it.

... Other arguments (besides x, y, lambdaVals, and warn) passed to ordinalNet.

Details

- The fold partition splits can be passed by the user via the folds argument. By default, the data are randomly divided into equally-sized partitions. Note that if lambda is tuned by cross validation, the fold splits are determined randomly and cannot be specified by the user. The set.seed function should be called prior to ordinalNetCV for reproducibility.

- A sequence of lambda values can be passed by the user via the lambdaVals argument. By default, the sequence is generated by first fitting the model to the full data set (this sequence is determined by the nLambda and lambdaMinRatio arguments of ordinalNet).

- The standardize argument of ordinalNet can be modified through the additional arguments (...). If standardize=TRUE, then the data are scaled within each cross validation fold. If standardize=TRUE and lambda is tuned by cross validation, then the data are also scaled within each tuning sub-fold. This is done because scaling is part of the statistical procedure and should be repeated each time the procedure is applied.
Value

An S3 object of class "ordinalNetCV", which contains the following:

**loglik** Vector of out-of-sample log-likelihood values. Each value corresponds to a different fold.

**misclass** Vector of out-of-sample misclassification rates. Each value corresponds to a different fold.

**brier** Vector of out-of-sample Brier scores. Each value corresponds to a different fold.

**devPct** Vector of out-of-sample percentages of deviance explained. Each value corresponds to a different fold.

**bestLambdaIndex** The index of the value within the lambda sequence selected for each fold by the tuning method.

**lambdaVals** The sequence of lambda values used for all cross validation folds.

**folds** A list containing the index numbers of each fold.

**fit** An object of class "ordinalNet", resulting from fitting ordinalNet to the entire dataset.

Examples

```r
## Not run:
# Simulate x as independent standard normal
# Simulate y|x from a parallel cumulative logit (proportional odds) model
set.seed(1)
n <- 50
intercepts <- c(-1, 1)
beta <- c(1, 1, 0, 0, 0)
ncat <- length(intercepts) + 1  # number of response categories
p <- length(beta)  # number of covariates
x <- matrix(rnorm(n*p), ncol=p)  # n x p covariate matrix
eta <- c(x %*% beta) + matrix(intercepts, nrow=n, ncol=ncat-1, byrow=TRUE)
invlogit <- function(x) 1 / (1+exp(-x))
cumprob <- apply(eta, 1, invlogit)
prob <- cbind(cumprob, 1) - cbind(0, cumprob)
yint <- apply(prob, 1, function(p) sample(1:ncat, size=1, prob=p))
y <- as.factor(yint)

# Evaluate out-of-sample performance of the cumulative logit model
# when lambda is tuned by cross validation (best average out-of-sample log-likelihood)
# cv <- ordinalNetCV(x, y, tuneMethod="cvLoglik")
# summary(cv)
```

## End(Not run)

---

**ordinalNetTune**  
Uses K-fold cross validation to obtain out-of-sample log-likelihood and misclassification rates for a sequence of lambda values.
**Description**

The data is divided into K folds. ordinalNet is fit K times (K=nFolds), each time leaving out one fold as a test set. The same sequence of lambda values is used each time. The out-of-sample log-likelihood, misclassification rate, Brier score, and percentage of deviance explained are obtained for each lambda value from the held out test set. It is up to the user to determine how to tune the model using this information.

**Usage**

```r
ordinalNetTune(
  x, y,
  lambdaVals = NULL,
  folds = NULL,
  nFolds = 5,
  printProgress = TRUE,
  warn = TRUE,
  ...
)
```

**Arguments**

- `x` Covariate matrix.
- `y` Response variable. Can be a factor, ordered factor, or a matrix where each row is a multinomial vector of counts. A weighted fit can be obtained using the matrix option, since the row sums are essentially observation weights. Non-integer matrix entries are allowed.
- `lambdaVals` An optional user-specified lambda sequence (vector). If NULL, a sequence will be generated using the model fit to the full training data. This default sequence is based on nLambda and lambdaMinRatio, which can be passed as additional arguments (otherwise ordinalNet default values are used). The maximum lambda is the smallest value that sets all penalized coefficients to zero, and the minimum lambda is the maximum value multiplied by the factor lambdaMinRatio.
- `folds` An optional list, where each element is a vector of row indices corresponding to a different cross validation fold. Indices correspond to rows of the x matrix. Each index number should be used in exactly one fold. If NULL, the data will be randomly divided into equal-sized partitions. It is recommended to use set.seed before calling this function to make results reproducible.
- `nFolds` Number of cross validation folds. Only used if folds=NULL.
- `printProgress` Logical. If TRUE the fitting progress is printed to the terminal.
- `warn` Logical. If TRUE, the following warning message is displayed when fitting a cumulative probability model with nonparallelTerms=TRUE (i.e. nonparallel or semi-parallel model). "Warning message: For out-of-sample data, the cumulative probability model with nonparallelTerms=TRUE may predict cumulative probabilities that are not monotone increasing." The warning is displayed by default, but the user may wish to disable it.
Other arguments (besides x, y, lambdaVals, and warn) passed to ordinalNet.

Details

- The fold partition splits can be passed by the user via the folds argument. By default, the data are randomly divided into equally-sized partitions. The set.seed function should be called prior to ordinalNetCV for reproducibility.
- A sequence of lambda values can be passed by the user via the lambdaVals argument. By default, the sequence is generated by first fitting the model to the full data set (this sequence is determined by the nLambda and lambdaMinRatio arguments of ordinalNet).
- The standardize argument of ordinalNet can be modified through the additional arguments (...). If standardize=TRUE, then the data are scaled within each cross validation fold. This is done because scaling is part of the statistical procedure and should be repeated each time the procedure is applied.

Value

An S3 object of class "ordinalNetTune", which contains the following:

- loglik Matrix of out-of-sample log-likelihood values. Each row corresponds to a lambda value, and each column corresponds to a fold.
- misclass Matrix of out-of-sample misclassification rates. Each row corresponds to a lambda value, and each column corresponds to a fold.
- brier Matrix of out-of-sample Brier scores. Each row corresponds to a lambda value, and each column corresponds to a fold.
- devPct Matrix of out-of-sample percentages of deviance explained. Each row corresponds to a lambda value, and each column corresponds to a fold.
- lambdaVals The sequence of lambda values used for all cross validation folds.
- folds A list containing the index numbers of each fold.
- fit An object of class "ordinalNet", resulting from fitting ordinalNet to the entire dataset.

Examples

```r
# Not run:
set.seed(1)
n <- 50
intercepts <- c(-1, 1)
beta <- c(1, 1, 0, 0, 0)
ncat <- length(intercepts) + 1  # number of response categories
p <- length(beta)  # number of covariates
x <- matrix(rnorm(n*p), ncol=p)  # n x p covariate matrix
eta <- c(x %*% beta) + matrix(intercepts, nrow=n, ncol=ncat-1, byrow=TRUE)
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlogit <- function(x) 1 / (1+exp(-x))
invlo...
y <- as.factor(yint)

# Fit parallel cumulative logit model; select lambda by cross validation
tunefit <- ordinalNetTune(x, y)
summary(tunefit)
plot(tunefit)
bestLambdaIndex <- which.max(rowMeans(tunefit$loglik))
coef(tunefit$fit, whichLambda=bestLambdaIndex, matrix=TRUE)
predict(tunefit$fit, whichLambda=bestLambdaIndex)

## End(Not run)

---

**plot.ordinalNetTune**  
*Plot method for "ordinalNetTune" object.*

**Description**

Plots the average out-of-sample log-likelihood, misclassification rate, Brier score, or percentage of deviance explained for each lambda value in the solution path. The average is taken over all cross validation folds.

**Usage**

```r
## S3 method for class 'ordinalNetTune'
plot(x, type = c("loglik", "misclass", "brier", "devPct"), ...)
```

**Arguments**

- `x`  
  An "ordinalNetTune" S3 object.

- `type`  
  Which performance measure to plot. Either "loglik", "misclass", "brier", or "devPct".

- `...`  
  Additional plot arguments.

**See Also**

- *ordinalNetTune*

**Examples**

```r
# See ordinalNetTune() documentation for examples.
```
predict.ordinalNet  

**Predict method for an "ordinalNet" object**

**Description**

Obtains predicted probabilities, predicted class, or linear predictors.

**Usage**

```r
## S3 method for class 'ordinalNet'
predict(
  object, 
  newx = NULL, 
  whichLambda = NULL, 
  criteria = c("aic", "bic"), 
  type = c("response", "class", "link"), 
  ... 
)
```

**Arguments**

- `object`: An "ordinalNet" S3 object.
- `newx`: Optional covariate matrix. If NULL, fitted values will be obtained for the training data, as long as the model was fit with the argument `keepTrainingData=TRUE`.
- `whichLambda`: Optional index number of the desired lambda value within the solution path sequence.
- `criteria`: Selects the best lambda value by AIC or BIC. Only used if `whichLambda=NULL`.
- `type`: The type of prediction required. Type "response" returns a matrix of fitted probabilities. Type "class" returns a vector containing the class number with the highest fitted probability. Type "link" returns a matrix of linear predictors.
- `...`: Not used. Additional predict arguments.

**Value**

The object returned depends on `type`.

**See Also**

- `ordinalNet`

**Examples**

# See ordinalNet() documentation for examples.
print.ordinalNetCV

Description
Prints the data frame returned by the summary.ordinalNetCV() method.

Usage
## S3 method for class 'ordinalNetCV'
print(x, ...)

Arguments
x An "ordinalNetCV" S3 object
... Not used. Additional print arguments.

See Also
ordinalNetCV

Examples
# See ordinalNet() documentation for examples.

---

print.ordinalNet

Print method for an "ordinalNet" object.

Description
Prints the data frame returned by the summary.ordinalNet() method.

Usage
## S3 method for class 'ordinalNet'
print(x, ...)

Arguments
x An "ordinalNet" S3 object
... Not used. Additional plot arguments.

See Also
ordinalNet

Examples
# See ordinalNet() documentation for examples.

---

print.ordinalNetCV

Print method for an "ordinalNetCV" object.

Description
Prints the data frame returned by the summary.ordinalNetCV() method.

Usage
## S3 method for class 'ordinalNetCV'
print(x, ...)

Arguments
x An "ordinalNetCV" S3 object
... Not used. Additional print arguments.

See Also
ordinalNetCV

Examples
# See ordinalNet() documentation for examples.

---

print.ordinalNet

Print method for an "ordinalNet" object.

Description
Prints the data frame returned by the summary.ordinalNet() method.

Usage
## S3 method for class 'ordinalNet'
print(x, ...)

Arguments
x An "ordinalNet" S3 object
... Not used. Additional plot arguments.

See Also
ordinalNet

Examples
# See ordinalNet() documentation for examples.

---

print.ordinalNetCV

Print method for an "ordinalNetCV" object.

Description
Prints the data frame returned by the summary.ordinalNetCV() method.

Usage
## S3 method for class 'ordinalNetCV'
print(x, ...)

Arguments
x An "ordinalNetCV" S3 object
... Not used. Additional print arguments.

See Also
ordinalNetCV

Examples
# See ordinalNet() documentation for examples.
print.ordinalNetTune

Examples

# See ordinalNetCV() documentation for examples.

print.ordinalNetTune    Print method for an "ordinalNetTune" object.

Description

Prints the data frame returned by the summary.ordinalNetTune() method.

Usage

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

Arguments

x    An "ordinalNetTune" S3 object.
...
    Not used. Additional print arguments.

See Also

ordinalNetTune

Examples

# See ordinalNetTune() documentation for examples.

summary.ordinalNet    Summary method for an "ordinalNet" object.

Description

Provides a data frame which summarizes the model fit at each lambda value in the solution path.model fit summary as a data frame.

Usage

## S3 method for class 'ordinalNet'
summary(object, ...)
Arguments

object       An "ordinalNet" S3 object
...       Not used. Additional summary arguments.

Value

A data frame containing a record for each lambda value in the solution path. Each record contains the following fields: lambda value, degrees of freedom (number of nonzero parameters), log-likelihood, AIC, BIC, and percent deviance explained.

See Also

ordinalNet

Examples

# See ordinalNet() documentation for examples.
Examples

# See ordinalNetCV() documentation for examples.

summary.ordinalNetTune

Summary method for an "ordinalNetTune" object.

Description

Provides a data frame which summarizes the cross validation results and may be useful for selecting an appropriate value for the tuning parameter lambda.

Usage

## S3 method for class 'ordinalNetTune'
summary(object, ...)  

Arguments

object  An "ordinalNetTune" S3 object.
...
Not used. Additional summary arguments.

Value

A data frame containing a record for each lambda value in the solution path. Each record contains the following: lambda value, average log-likelihood, average misclassification rate, average Brier score, and average percentage of deviance explained. Averages are taken across all cross validation folds.

See Also

ordinalNetTune

Examples

# See ordinalNetTune() documentation for examples.
Index

coef.ordinalNet, 2
ordinalNet, 3, 3, 15, 16, 18
ordinalNetCV, 9, 16, 18
ordinalNetTune, 11, 14, 17, 19
plot.ordinalNetTune, 14
predict.ordinalNet, 15
print.ordinalNet, 16
print.ordinalNetCV, 16
print.ordinalNetTune, 17
summary.ordinalNet, 17
summary.ordinalNetCV, 18
summary.ordinalNetTune, 19