Package 'pemultinom'

July 23, 2025

Type Package

Title L1-Penalized Multinomial Regression with Statistical Inference

Version 0.1.1

Description We aim for fitting a multinomial regression model with Lasso penalty and doing statistical inference (calculating confidence intervals of coefficients and p-values for individual variables). It implements 1) the coordinate descent algorithm to fit an l1-penalized multinomial regression model (parameterized with a reference level); 2) the debiasing approach to obtain the inference results, which is described in ``Tian, Y., Rusinek, H., Masurkar, A. V., & Feng, Y. (2024). L1-Penalized Multinomial Regression: Estimation, Inference, and Prediction, With an Application to Risk Factor Identification for Different Dementia Subtypes. Statistics in Medicine, 43(30), 5711-5747."

Imports foreach, doParallel, stats, Rcpp, nnet, magrittr, utils, lpSolve

LinkingTo Rcpp

License GPL-2

Depends R (>= 3.5.0)

Encoding UTF-8

RoxygenNote 7.3.2

Suggests knitr, rmarkdown

NeedsCompilation yes

Author Ye Tian [aut, cre], Henry Rusinek [aut], Arjun V. Masurkar [aut], Yang Feng [aut]

Maintainer Ye Tian <ye.t@columbia.edu>

Repository CRAN

Date/Publication 2025-02-28 09:40:02 UTC

Contents

`cv.pemultinom` *Fit a multinomial regression model with Lasso penalty.*

Description

Fit a multinomial regression model with Lasso penalty. This function implements the l1-penalized multinomial regression model (parameterized with a reference level). A cross-validation procedure is applied to choose the tuning parameter. See Tian et al. (2024) for details.

Usage

```
cv.pemultinom(
  x,
  y,
  ref = NULL,
  nfolds = 5,
  nlambda = 100,
  max_iter = 200,
  tol = 0.001,
  ncores = 1,
  standardized = TRUE,
  weights = NULL,
  info = TRUE,
  lambda_min_ratio = 0.01
)
```

Arguments

`x`	the design/predictor matrix, each row of which is an observation vector.
`y`	the response variable. Can be of one type from factor/integer/character.
`ref`	the reference level. Default = NULL, which sets the reference level to the last category (sorted by alphabetical order)
`nfolds`	the number of cross-validation folds to use. Default = 5.
`nlambda`	the number of penalty parameter candidates. Default = 100.
`max_iter`	the maximum iteration rounds in each iteration of the coordinate descent algorithm. Default = 200.
`tol`	convergence threshold (tolerance level) for coordinate descent. Default = 1e-3.
`ncores`	the number of cores to use for parallel computing. Default = 1.
`standardized`	logical flag for x variable standardization, prior to fitting the model sequence. Default = TRUE. Note that the fitting results will be translated to the original scale before output.

`weights` observation weights. Should be a vector of non-negative numbers of length n (the number of observations). Default = NULL, which sets equal weights for all observations.

`info` whether to print the information or not. Default = TRUE.

`lambda_min_ratio`
the ratio between lambda.min and lambda.max, where lambda.max is automatically determined by the code and the lambda sequence will be determined by 'exp(seq(log(lambda.max), log(lambda.min), len = nlambda))'.

Value

A list with the following components.

`beta.list` the estimates of coefficients. It is a list of which the k-th component is the contrast coefficient between class k and the reference class corresponding to different lambda values. The j-th column of each list component corresponds to the j-th lambda value.

`beta.1se` the coefficient estimate corresponding to lambda.1se. It is a matrix, and the k-th column is the contrast coefficient between class k and the reference class.

`beta.min` the coefficient estimate corresponding to lambda.min. It is a matrix, and the k-th column is the contrast coefficient between class k and the reference class.

`lambda.1se` the largest value of lambda such that error is within 1 standard error of the minimum. See Chapter 2.3 of Hastie et al. (2015) for more details.

`lambda.min` the value of lambda that gives minimum cvm.

`cvm` the weights in objective function.

`cvsd` the estimated marginal probability for each class.

`lambda` lambda values considered in the cross-validation process.

References

Hastie, T., Tibshirani, R., & Wainwright, M. (2015). Statistical learning with sparsity. Monographs on statistics and applied probability, 143.

Tian, Y., Rusinek, H., Masurkar, A. V., & Feng, Y. (2024). L1-Penalized Multinomial Regression: Estimation, Inference, and Prediction, With an Application to Risk Factor Identification for Different Dementia Subtypes. Statistics in Medicine, 43(30), 5711-5747.

See Also

`debiased_lasso`, `predict_pemultinom`.

Examples

```
# generate data from a logistic regression model with n = 100, p = 50, and K = 3
set.seed(0, kind = "L'Ecuyer-CMRG")
n <- 100
p <- 50
K <- 3
```

```
Sigma <- outer(1:p, 1:p, function(x,y) {
  0.9^(abs(x-y))
})
R <- chol(Sigma)
s <- 3
beta_coef <- matrix(0, nrow = p+1, ncol = K-1)
beta_coef[1+1:s, 1] <- c(3, 3, 3)
beta_coef[1+1:s+s, 2] <- c(3, 3, 3)

x <- matrix(rnorm(n*p), ncol = p) %*% R
y <- sapply(1:n, function(j){
  prob_i <- c(sapply(1:(K-1), function(k){
    exp(sum(x[j, ]*beta_coef[-1, k]))
  }), 1)
  prob_i <- prob_i/sum(prob_i)
  sample(1:K, size = 1, replace = TRUE, prob = prob_i)
})

# fit the l1-penalized multinomial regression model
fit <- cv.pemultinom(x, y, ncores = 2)
beta <- fit$beta.min

# generate test data from the same model
x.test <- matrix(rnorm(n*p), ncol = p) %*% R
y.test <- sapply(1:n, function(j){
  prob_i <- c(sapply(1:(K-1), function(k){
    exp(sum(x.test[j, ]*beta_coef[-1, k]))
  }), 1)
  prob_i <- prob_i/sum(prob_i)
  sample(1:K, size = 1, replace = TRUE, prob = prob_i)
})

# predict labels of test data and calculate the misclassification error rate (using beta.min)
ypred.min <- predict_pemultinom(fit$beta.min, ref = 3, xnew = x.test, type = "class")
mean(ypred.min != y.test)

# predict labels of test data and calculate the misclassification error rate (using beta.1se)
ypred.1se <- predict_pemultinom(fit$beta.1se, ref = 3, xnew = x.test, type = "class")
mean(ypred.1se != y.test)

# predict posterior probabilities of test data
ypred.prob <- predict_pemultinom(fit$beta.min, ref = 3, xnew = x.test, type = "prob")
```

`debiased_lasso` *Doing statistical inference on l1-penalized multinomial regression via debiased Lasso (or desparisified Lasso).*

Description

Doing statistical inference on l1-penalized multinomial regression via debiased Lasso (or desparisified Lasso). This function implements the algorithm described in Tian et al. (2024), which is an extension of the original debiased Lasso (Van de Geer et al. 2014; Zhang and Zhang. 2014; Javanmard, A. and Montanari, A. (2014)) to the multinomial case.

Usage

```
debiased_lasso(
  x,
  y,
  ref = NULL,
  beta = NULL,
  nfolds = 5,
  ncores = 1,
  nlambda = 50,
  max_iter = 200,
  tol = 0.001,
  lambda.choice = "lambda.min",
  alpha = 0.05,
  method = c("nodewise_reg", "LP"),
  info = TRUE,
  LP.parameter = list(eta = NULL, split.ratio = 0.5),
  lambda_min_ratio = 0.01
)
```

Arguments

x	the design/predictor matrix, each row of which is an observation vector.
y	the response variable. Can be of one type from factor/integer/character.
ref	the reference level. Default = NULL, which sets the same reference level as used in obtaining beta. Even when the user inputs ref manually, it should be always the same reference level as used in obtaining beta.
beta	the beta estimate from l1-penalized multinomial regression. Should be in the same format as beta.min or beta.1se in output of function cv.pemultinom. The user is recommended to directly pass beta.min or beta.1se from the output of function cv.pemultinom to parameter beta.
nfolds	the number of cross-validation folds to use. Cross-validation is used to determine the best tuning parameter lambda in the nodewise regression, i.e., Algorithm 2 in Tian et al. (2024). Default = 5.
ncores	the number of cores to use for parallel computing. Default = 1.
nlambda	the number of penalty parameter candidates in the cross-validation procedure. Cross-validation is used to determine the best tuning parameter lambda in the nodewise regression, i.e., Algorithm 2 in Tian et al. (2024). Default = 100.
max_iter	the maximum iteration rounds in each iteration of the coordinate descent algorithm. Default = 200.

`tol` convergence threshold (tolerance level) for coordinate descent. Default = 1e-3.

`lambda.choice` the choice of the tuning parameter lambda used in the nodewise regression. It can only be either "lambda.min" or "lambda.1se". The interpretation is the same as in the ‘cv.pemultinom‘ function. Default = "lambda.min".

`alpha` significance level used in the output confidence interval. Has to be a number between 0 and 1. Default = 0.05.

`method` which method to estimate the Hessian inverse matrix. Can be either "nodewise_reg" or "LP".

- nodewise_reg: the method presented in the main text of Tian et al. (2024).
- LP: the method presented in Section A of the supplements of Tian et al. (2024). Warning: This method is not well implemented as ‘eta‘ parameter has to be set as fixed and currently cannot be automatically tuned.

`info` whether to print the information or not. Default = TRUE.

`LP.parameter` If `method` = ‘LP‘, then this parameter will be used. It has to be a list of two components like ‘list(eta = NULL, split.ratio = 0.5)‘. Here is the interpretation:

- eta: This is the parameter used in the LP method which is the righthand side of the contraints. Default ‘eta‘ = ‘NULL‘, which will be set as $2\sqrt{\log(p)/n}$ where p is the number of features and n is the total sample size.
- split.ratio: The split ratio used in the LP method. The data will be splitted into two parts by class based on ‘split.ratio‘. The first part will be used to fit the Lasso estimator, and the second part will be used to fit the Hessian inverse through the LP method. Default ‘split.ratio‘ = 0.5.

`lambda_min_ratio` the ratio between lambda.min and lambda.max used for the Lasso problem in the nodewise regression (for estimating the Hessian inverse), where lambda.max is automatically determined by the code and the lambda sequence will be determined by ‘exp(seq(log(lambda.max), log(lambda.min), len = nlambda))‘.

Value

A list of data frames, each of which contains the inference results for each class (v.s. the reference class). In the data frame, each row represents a variable. The columns include:

`beta` the debiased point estimate of the coefficient

`p_value` p value of each variable

`CI_lower` lower endpoint of the confidence interval for each coefficient

`CI_upper` upper endpoint of the confidence interval for each coefficient

`std_dev` the estimated standard deviation of each component of beta estimate

References

Tian, Y., Rusinek, H., Masurkar, A. V., & Feng, Y. (2024). L1-Penalized Multinomial Regression: Estimation, Inference, and Prediction, With an Application to Risk Factor Identification for Different Dementia Subtypes. Statistics in Medicine, 43(30), 5711-5747.

Van de Geer, S., Bühlmann, P., Ritov, Y. A., & Dezeure, R. (2014). On asymptotically optimal confidence regions and tests for high-dimensional models. The Annals of Statistics, 42(3), 1166-1202.

Zhang, C. H., & Zhang, S. S. (2014). Confidence intervals for low dimensional parameters in high dimensional linear models. Journal of the Royal Statistical Society: Series B (Statistical Methodology), 76(1), 217-242.

Javanmard, A., & Montanari, A. (2014). Confidence intervals and hypothesis testing for high-dimensional regression. The Journal of Machine Learning Research, 15(1), 2869-2909.

See Also

cv.pemultinom, predict_pemultinom.

Examples

```
# generate data from a logistic regression model with n = 100, p = 50, and K = 3
set.seed(0, kind = "L'Ecuyer-CMRG")
n <- 100
p <- 50
K <- 3

Sigma <- outer(1:p, 1:p, function(x,y) {
  0.9^(abs(x-y))
})
R <- chol(Sigma)
s <- 3
beta_coef <- matrix(0, nrow = p+1, ncol = K-1)
beta_coef[1+1:s, 1] <- c(3, 3, 3)
beta_coef[1+1:s+s, 2] <- c(3, 3, 3)

x <- matrix(rnorm(n*p), ncol = p) %*% R
y <- sapply(1:n, function(j){
  prob_i <- c(sapply(1:(K-1), function(k){
    exp(sum(x[j, ]*beta_coef[-1, k]))
  }), 1)
  prob_i <- prob_i/sum(prob_i)
  sample(1:K, size = 1, replace = TRUE, prob = prob_i)
})

# fit the l1-penalized multinomial regression model
fit <- cv.pemultinom(x, y, ncores = 2)
beta <- fit$beta.min

# run the debiasing approach
fit_debiased <- debiased_lasso(x, y, beta = beta, ncores = 2)
```

`predict_pemultinom` *Make predictions on new predictors based on fitted l1-penalized multinomial regression model.*

Description

Make predictions on new predictors based on fitted l1-penalized multinomial regression model, by using the fitted beta.

Usage

```
predict_pemultinom(beta, ref, xnew, type = c("class", "prob"))
```

Arguments

`beta`	the beta estimate from l1-penalized multinomial regression. Should be in the same format as `beta.min` or `beta.1se` in output of function `cv.pemultinom`. The user is recommended to directly pass `beta.min` or `beta.1se` from the output of function `cv.pemultinom` to parameter `beta`.
`ref`	the reference level, which should be the same reference level as used in obtaining `beta`. An input is required.
`xnew`	new observations to predict labels for. Should be a matrix or a data frame, where each row and column represents an observation and predictor, respectively.
`type`	the type of prediction output. Can be 'class' or 'prob'. Default = 'class'. • class: the predicted class/label. • prob: the predicted probability for each class.

Value

When `type` = 'class', return a vector. When `type` = 'prob', return a matrix where each row and column represent an observation and a probability of that class, respectively. Default = 'class'.

References

Tian, Y., Rusinek, H., Masurkar, A. V., & Feng, Y. (2023). L1-penalized Multinomial Regression: Estimation, inference, and prediction, with an application to risk factor identification for different dementia subtypes. arXiv preprint arXiv:2302.02310.

See Also

`cv.pemultinom`, `debiased_lasso`.

Examples

```
# generate training data from Model 1 in Tian et al. (2023) with n = 50 and p = 50
set.seed(1, kind = "L'Ecuyer-CMRG")
n <- 50
p <- 50
K <- 3

Sigma <- outer(1:p, 1:p, function(x,y) {
  0.9^(abs(x-y))
})
R <- chol(Sigma)
s <- 3
beta_coef <- matrix(0, nrow = p+1, ncol = K-1)
beta_coef[1+1:s, 1] <- c(1.5, 1.5, 1.5)
beta_coef[1+1:s+s, 2] <- c(1.5, 1.5, 1.5)

x <- matrix(rnorm(n*p), ncol = p) %*% R
y <- sapply(1:n, function(j){
  prob_i <- c(sapply(1:(K-1), function(k){
    exp(sum(x[j, ]*beta_coef[-1, k]))
  }), 1)
  prob_i <- prob_i/sum(prob_i)
  sample(1:K, size = 1, replace = TRUE, prob = prob_i)
})

# fit the l1-penalized multinomial regression model
fit <- cv.pemultinom(x, y, ncores = 2)

# generate test data from the same model
x.test <- matrix(rnorm(n*p), ncol = p) %*% R
y.test <- sapply(1:n, function(j){
  prob_i <- c(sapply(1:(K-1), function(k){
    exp(sum(x.test[j, ]*beta_coef[-1, k]))
  }), 1)
  prob_i <- prob_i/sum(prob_i)
  sample(1:K, size = 1, replace = TRUE, prob = prob_i)
})

# predict labels of test data and calculate the misclassification error rate (using beta.min)
ypred.min <- predict_pemultinom(fit$beta.min, ref = 3, xnew = x.test, type = "class")
mean(ypred.min != y.test)

# predict labels of test data and calculate the misclassification error rate (using beta.1se)
ypred.1se <- predict_pemultinom(fit$beta.1se, ref = 3, xnew = x.test, type = "class")
mean(ypred.1se != y.test)

# predict posterior probabilities of test data
ypred.prob <- predict_pemultinom(fit$beta.min, ref = 3, xnew = x.test, type = "prob")
```

Index