Package 'SSGL'

March 24, 2025

Type Package

Title Spike-and-Slab Group Lasso for Group-Regularized Generalized Linear Models

Version 2.0

Author Ray Bai [aut, cre]

Maintainer Ray Bai <raybaistat@gmail.com>

Description Fits group-regularized generalized linear models (GLMs) using the spike-and-slab group lasso (SSGL) prior of Bai et al. (2022) <doi:10.1080/01621459.2020.1765784> and extended to GLMs by Bai (2023) <doi:10.48550/arXiv.2007.07021>. This package supports fitting the SSGL model for the following GLMs with group sparsity: Gaussian linear regression, binary logistic regression, and Poisson regression.

License GPL-3

Depends R (>= 3.6.0)

Imports grpreg, stats, caret, parallel, doParallel, doRNG, foreach, MASS, Matrix, GIGrvg, BayesLogit

NeedsCompilation yes

Repository CRAN

Date/Publication 2025-03-24 21:10:02 UTC

Contents

SSGL *Spike-and-Slab Group Lasso for Group-Regularized Generalized Linear Models (GLMs)*

Description

The SSGL function implements maximum a posteriori (MAP) estimation for group-regularized GLMs with the spike-and-slab group lasso (SSGL) penalty of Bai et al. (2022) and Bai (2023). The identity link function is used for Gaussian regression, the logit link is used for binomial regression, and the log link is used for Poisson regression. If the covariates in each x_i are grouped according to known groups $g = 1, ..., G$, then this function can estimate some of the G groups of coefficients as all zero, depending on the amount of regularization.

This function only returns point estimates. Please refer to the SSGL_gibbs function if uncertainty quantification of the model parameters is desired. In general, we recommend using SSGL for estimation and variable selection and SSGL_gibbs for uncertainty quantification.

The SSGL function also has the option of returning the generalized information criterion (GIC) of Fan and Tang (2013) for each regularization parameter in the grid lambda0. The GIC can be used for model selection and serves as a useful alternative to cross-validation. The formula for the GIC and a given λ_0 is

$$DIC(\lambda_0) = \frac{1}{n} Deviance_{\lambda_0} + a_n \times \nu),$$

where $Deviance_{\lambda_0}$ is the deviance computed with the estimate of beta based on spike hyperparameter λ_0, ν_0 is the number of nonzero elements in the estimated beta, and a_n is a sequence that diverges at a suitable rate relative to n. As recommended by Fan and Tang (2013), we set $a_n = \{\log(\log(n))\} \log(p)$.

If cross-validation is preferred for tuning λ_0, please refer to the SSGL_cv function.

Usage

```
SSGL(Y, X, groups, family=c("gaussian","binomial","poisson"),
     X_test, group_weights, n_lambda0=25,
     lambda0, lambda1=1, a=1, b=length(unique(groups)),
     max_iter=100, tol = 1e-6, return_GIC=TRUE, print_lambda0=TRUE)
```

Arguments

Y $n \times 1$ vector of responses for training data.

X $n \times p$ design matrix for training data, where the jth column of X corresponds to the jth overall covariate.

groups p-dimensional vector of group labels. The jth entry in groups should contain either the group number *or* the factor level name that the feature in the jth column of X belongs to. groups must be either a vector of integers or factors.

family exponential dispersion family of the response variables. Allows for "gaussian", "binomial", and "poisson".

`X_test` $n_{test} \times p$ design matrix for test data to calculate predictions. `X_test` must have the *same* number of columns as X, but not necessarily the same number of rows. If *no* test data is provided or if in-sample predictions are desired, then the function automatically sets `X_test=X` in order to calculate *in-sample* predictions.

`group_weights` group-specific, nonnegative weights for the penalty. Default is to use the square roots of the group sizes.

`n_lambda0` number of spike hyperparameters L. Default is `n_lambda0=25`.

`lambda0` grid of L spike hyperparameters λ_0. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.

`lambda1` slab hyperparameter λ_1 in the SSGL prior. Default is `lambda1=1`.

`a` shape hyperparameter for the $Beta(a, b)$ prior on the mixing proportion in the SSGL prior. Default is `a=1`.

`b` shape hyperparameter for the $Beta(a, b)$ prior on the mixing proportion in the SSGL prior. Default is `b=length(unique(groups))`, i.e. the number of groups.

`max_iter` maximum number of iterations in the algorithm. Default is `max_iter=100`.

`tol` convergence threshold for algorithm. Default is `tol=1e-6`.

`return_GIC` Boolean variable for whether or not to return the GIC. Default is `return_GIC=TRUE`.

`print_lambda0` Boolean variable for whether or not to print the current value in `lambda0`. Default is `print_lambda0=TRUE`.

Value

The function returns a list containing the following components:

`lambda0` $L \times 1$ vector of spike hyperpameters `lambda0` used to fit the model. `lambda0` is displayed in descending order.

`beta` $p \times L$ matrix of estimated regression coefficients. The kth column in `beta` corresponds to the kth spike hyperparameter in `lambda0`.

`beta0` $L \times 1$ vector of estimated intercepts. The kth entry in `beta0` corresponds to the kth spike hyperparameter in `lambda0`.

`classifications`
$G \times L$ matrix of classifications, where G is the number of groups. An entry of "1" indicates that the group was classified as nonzero, and an entry of "0" indicates that the group was classified as zero. The kth column of `classifications` corresponds to the kth spike hyperparameter in `lambda0`.

`Y_pred` $n_{test} \times L$ matrix of predicted mean response values $\mu_{test} = E(Y_{test})$ based on the test data in `X_test` (or training data `X` if no argument was specified for `X_test`). The kth column in `Y_pred` corresponds to the predictions for the kth spike hyperparameter in `lambda0`.

`GIC` $L \times 1$ vector of GIC values. The kth entry of `GIC` corresponds to the kth entry in our `lambda0` grid. This is not returned if `return_GIC=FALSE`.

`lambda0_GIC_min`
The value in `lambda0` that minimizes `GIC`. This is not returned if `return_GIC=FALSE`.

`min_GIC_index` The index of `lambda0_GIC_min` in `lambda0`. This is not returned if `return_GIC=FALSE`.

References

Bai, R. (2023). "Bayesian group regularization in generalized linear models with a continuous spike-and-slab prior." *arXiv pre-print arXiv:2007.07021*.

Bai, R., Moran, G. E., Antonelli, J. L., Chen, Y., and Boland, M.R. (2022). "Spike-and-slab group lassos for grouped regression and sparse generalized additive models." *Journal of the American Statistical Association*, **117**:184-197.

Fan, Y. and Tang, C. Y. (2013). "Tuning parameter selection in high dimensional penalized likelihood." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **75**:531-552.

Examples

```
## Generate data
set.seed(12345)
X = matrix(runif(100*10), nrow=100)
n = dim(X)[1]
groups = c(”A”,”A”,”A”,”B”,”B”,”B”,”C”,”C”,”D”,”D”)
groups = as.factor(groups)
beta_true = c(-2.5,1.5,1.5,0,0,0,2,-2,0,0)

## Generate responses from Gaussian distribution
Y = crossprod(t(X), beta_true) + rnorm(n)

## Generate test data
n_test = 50
X_test = matrix(runif(n_test*10), nrow=n_test)

## Fit SSGL model with 10 spike hyperparameters
## NOTE: If you do not specify lambda0, the program will automatically choose a suitable grid.
SSGL_mod = SSGL(Y, X, groups, family=”gaussian”, X_test, lambda0=seq(from=50,to=5,by=-5))

## Regression coefficient estimates
SSGL_mod$beta

## Predicted n_test-dimensional vectors mu=E(Y.test) based on test data, X_test.
## The kth column of 'Y_pred' corresponds to the kth entry in 'lambda.'
SSGL_mod$Y_pred

## Classifications of the 8 groups. The kth column of 'classifications'
## corresponds to the kth entry in 'lambda.'
SSGL_mod$classifications

## Plot lambda vs. GIC
plot(SSGL_mod$lambda0, SSGL_mod$GIC, type='l')

## Model selection with the lambda that minimizes GIC
SSGL_mod$lambda0_GIC_min
SSGL_mod$min_GIC_index
SSGL_mod$classifications[, SSGL_mod$min_GIC_index]
SSGL_mod$beta[, SSGL_mod$min_GIC_index]
```

```
## Example with Poisson regression

## Generate data
set.seed(1234)
X = matrix(runif(100*10), nrow=100)
n = dim(X)[1]
groups = c("A","A","A","B","B","B","C","C","D","D")
groups = as.factor(groups)
beta_true = c(-2.5,1.5,1.5,0,0,0,2,-2,0,0)

## Generate count responses
eta = crossprod(t(X), beta_true)
Y = rpois(n, exp(eta))

## Generate test data
n_test = 50
X_test = matrix(runif(n_test*10), nrow=n_test)

## Fit SSGL model
SSGL_poisson_mod = SSGL(Y, X, groups, family="poisson")

## Regression coefficient estimates
SSGL_poisson_mod$beta

## Predicted n_test-dimensional vectors mu=E(Y.test) based on test data, X_test.
## The kth column of 'Y_pred' corresponds to the kth entry in 'lambda.'
SSGL_poisson_mod$Y_pred

## Classifications of the 8 groups. The kth column of 'classifications'
## corresponds to the kth entry in 'lambda.'
SSGL_poisson_mod$classifications

## Plot lambda vs. GIC
plot(SSGL_poisson_mod$lambda0, SSGL_poisson_mod$GIC, type='l')

## Model selection with the lambda that minimizes GIC
SSGL_poisson_mod$lambda0_GIC_min
SSGL_poisson_mod$min_GIC_index
SSGL_poisson_mod$classifications[, SSGL_mod$min_GIC_index]
SSGL_poisson_mod$beta[, SSGL_mod$min_GIC_index]
```

SSGL_cv — *Cross-Validation for Spike-and-Slab Group Lasso in Group-Regularized Generalized Linear Models (GLMs)*

Description

The SSGL_cv function implements K-fold cross-validation for choosing the regularization parameter λ_0 in group-regularized GLMs with the spike-and-slab group lasso (SSGL) penalty of Bai et al. (2022) and Bai (2023). The default is $K = 10$. The identity link function is used for Gaussian regression, the logit link is used for binomial regression, and the log link is used for Poisson regression.

Although you can choose `lambda0` from cross-validation with this function, it can be time-consuming to do so if the number of groups G and/or the number of total covariantes p is moderate to large. In this case, you may choose to set the argument `parallelize=TRUE`, which will perform K-fold cross-validation in parallel across the K folds. If K cores are used, then this may offer a speed-up of roughly the order of K.

As an alternative to cross-validation, you can also simply use the `SSGL` function on your data and select the final model according to the `lambda0` which minimizes the generalized information criterion (GIC). See description of the `SSGL` function for more details.

Usage

```
SSGL_cv(Y, X, groups,
        family=c("gaussian","binomial","poisson"),
        group_weights, n_folds=10, n_lambda0=25,
        lambda0, lambda1=1, a=1, b=length(unique(groups)),
        max_iter=100, tol=1e-6, parallelize=FALSE, n_cores)
```

Arguments

`Y` $n \times 1$ vector of responses for training data.

`X` $n \times p$ design matrix for training data, where the jth column corresponds to the jth overall feature.

`groups` p-dimensional vector of group labels. The jth entry in `groups` should contain either the group number *or* the factor level name that the feature in the jth column of `X` belongs to. `groups` must be either a vector of integers or factors.

`family` exponential dispersion family of the response variables. Allows for `"gaussian"`, `"binomial"`, and `"poisson"`.

`group_weights` group-specific, nonnegative weights for the penalty. Default is to use the square roots of the group sizes.

`n_folds` number of folds K to use in K-fold cross-validation. Default is `n_folds=10`.

`n_lambda0` number of spike hyperparameters L. Default is `n_lambda0=25`.

`lambda0` grid of L spike hyperparameters λ_0. The user may specify either a scalar or a vector. If the user does not provide this, the program chooses the grid automatically.

`lambda1` slab hyperparameter λ_1 in the SSGL prior. Default is `lambda1=1`.

`a` shape hyperparameter for the $Beta(a, b)$ prior on the mixing proportion in the SSGL prior. Default is `a=1`.

`b` shape hyperparameter for the $Beta(a, b)$ prior on the mixing proportion in the SSGL prior. Default is `b=length(unique(groups))`, i.e. the number of groups.

`max_iter` maximum number of iterations in the algorithm. Default is `max_iter=100`.

`tol` convergence threshold for algorithm. Default is `tol=1e-6`.

`parallelize` Boolean variable for whether or not to parallelize K-fold cross-validation across the K folds. If the number of group G and/or the number of predictors p is moderate or large, then it may be preferable to perform cross-validation in parallel. In this case, the user can set `parallelize=TRUE`.

`n_cores` Number of cores to use for parallelization. If the user does not specify this, the function will use the minimum of either K or the number of available cores minus one.

Value

The function returns a list containing the following components:

`lambda0` $L \times 1$ vector of spike hyperparameters `lambda0` used to fit the model. `lambda0` is displayed in descending order.

`cve` $L \times 1$ vector of mean cross-validation error across all K folds. The kth entry in `cve` corresponds to the kth spike hyperparameter parameter in `lambda0`.

`cvse` $L \times 1$ vector of standard errors for cross-validation error across all K folds. The kth entry in `cvse` corresponds to the kth spike hyperparameter parameter in `lambda0`.

`lambda0_cve_min` The value in `lambda0` that minimizes mean cross-validation error `cve`.

`min_cve_index` The index of `lambda0_cve_min` in `lambda0`.

References

Bai, R. (2023). "Bayesian group regularization in generalized linear models with a continuous spike-and-slab prior." *arXiv pre-print arXiv:2007.07021.*

Bai, R., Moran, G. E., Antonelli, J. L., Chen, Y., and Boland, M.R. (2022). "Spike-and-slab group lassos for grouped regression and sparse generalized additive models." *Journal of the American Statistical Association*, **117**:184-197.

Examples

```
## Generate data
set.seed(12345)
X = matrix(runif(50*6), nrow=50)
n = dim(X)[1]
groups = c(1,1,1,2,2,2)
beta_true = c(-2,1,1.5,0,0,0)

## Generate responses from Gaussian distribution
Y = crossprod(t(X), beta_true) + rnorm(n)

## K-fold cross-validation
## NOTE: If you do not specify lambda0, the function will automatically choose a suitable grid.

ssgl_mods = SSGL_cv(Y, X, groups, family=”gaussian”, n_folds=5, lambda0=seq(from=16,to=4,by=-4))

## Plot cross-validation curve
```

```
plot(ssgl_mods$lambda0, ssgl_mods$cve, type="l", xlab="lambda0", ylab="CVE")

## lambda which minimizes mean CVE
ssgl_mods$lambda0_cve_min
ssgl_mods$min_cve_index


## Example with binary logistic regression

## Generate binary responses
set.seed(123)
X = matrix(runif(50*6), nrow=50)
n = dim(X)[1]
groups = c(1,1,2,2,3,3)
beta_true = c(-2,1.5,0,0,2,-1.5)
eta = crossprod(t(X), beta_true)
Y = rbinom(n, size=1, prob=1/(1+exp(-eta)))

## K-fold cross-validation. Set parallelize=TRUE for potential speed-up
## If n_cores is not specified, then the function will automatically choose
# the minimum of either K or the number of available cores minus one.

ssgl_logistic_mods = SSGL_cv(Y, X, groups, family="binomial", parallelize=TRUE, n_cores=2)

## Plot cross-validation curve
plot(ssgl_logistic_mods$lambda0, ssgl_logistic_mods$cve, type="l", xlab="lambda0", ylab="CVE")

## lambda which minimizes mean CVE
ssgl_logistic_mods$lambda0_cve_min
ssgl_logistic_mods$min_cve_index
```

SSGL_gibbs   *Gibbs sampling for Spike-and-Slab Group Lasso in Group-Regularized Generalized Linear Models (GLMs)*

Description

The SSGL_gibbs function implements Gibbs sampling for group-regularized GLMs with the spike-and-slab group lasso (SSGL) prior of Bai et al. (2022) and Bai (2023). The identity link function is used for Gaussian regression, the logit link is used for binomial regression, and the log link is used for Poisson regression.

For binomial and Poisson regression, Polya-gamma data augmentation (Polson et al., 2013) is used to draw MCMC samples. The details are described in Bai (2023).

Note that the SSGL_gibbs function only returns the posterior mean, the 95 percent posterior credible intervals, and the posterior samples for the elements of the model parameter β and the predicted mean response $\mu_{test} = E(Y_{test})$. This function does not perform variable selection.

It is recommended that you use the SSGL function to perform variable selection and MAP estimation. If uncertainty quantification is *also* desired, then this SSGL_gibbs function can be used.

Usage

```
SSGL_gibbs(Y, X, groups, family=c("gaussian","binomial","poisson"),
           X_test, group_weights, lambda0=5, lambda1=1,
           a=1, b=length(unique(groups)),
           burn=1000, n_mcmc=2000, save_samples=TRUE)
```

Arguments

`Y` $n \times 1$ vector of responses for training data.

`X` $n \times p$ design matrix for training data, where the jth column corresponds to the jth overall feature.

`groups` p-dimensional vector of group labels. The jth entry in `groups` should contain either the group number *or* the factor level name that the feature in the jth column of `X` belongs to. `groups` must be either a vector of integers or factors.

`family` exponential dispersion family of the response variables. Allows for `"gaussian"`, `"binomial"`, and `"poisson"`.

`X_test` $n_{test} \times p$ design matrix for test data to calculate predictions. `X_test` must have the *same* number of columns as X, but not necessarily the same number of rows. If *no* test data is provided or if in-sample predictions are desired, then the function automatically sets `X_test=X` in order to calculate *in-sample* predictions.

`group_weights` group-specific, nonnegative weights for the penalty. Default is to use the square roots of the group sizes.

`lambda0` spike hyperparameter λ_0 in the SSGL prior. Default is `lambda0=5`.

`lambda1` slab hyperparameter λ_1 in the SSGL prior. Default is `lambda1=1`.

`a` shape hyperparameter for the $Beta(a, b)$ prior on the mixing proportion in the SSGL prior. Default is `a=1`.

`b` shape hyperparameter for the $Beta(a, b)$ prior on the mixing proportion in the SSGL prior. Default is `b=length(unique(groups))`, i.e. the number of groups.

`burn` Number of warm-up MCMC samples to discard as burn-in. Default is `burn=1000`.

`n_mcmc` Number of MCMC samples to save for posterior inference. Default is `n_mcmc=2000`.

`save_samples` Boolean variable for whether or not to save the MCMC samples for β and predicted mean response $\mu_{test} = E(Y_{text})$. Default is `save_samples=TRUE`.

Value

The function returns a list containing the following components:

`beta_hat` estimated posterior mean of $p \times 1$ regression coefficient vector β.

`Y_pred_hat` estimated posterior mean of $n_{test} \times 1$ vector of predicted mean response values $\mu_{test} = E(Y_{test})$ based on the test data in `X_test` (or training data `X` if no argument was specified for `X_test`).

`beta_lower` $p \times 1$ vector of lower endpoints of the 95 percent posterior credible intervals for β.

`beta_upper` $p \times 1$ vector of upper endpoints of the 95 percent posterior credible intervals for β.

`Y_pred_lower` $n_{test} \times 1$ vector of lower endpoints of the 95 percent posterior credible intervals for $\mu_{test} = E(Y_{test})$.

`Y_pred_upper` $n_{test} \times 1$ vector of upper endpoints of the 95 percent posterior credible intervals for $\mu_{test} = E(Y_{test})$.

`beta_samples` $p\times$ `n_mcmc` matrix of saved posterior samples for β. The jth row of `beta_samples` consists of the posterior samples for the jth regression coefficient in β. This is not returned if `save_samples=FALSE`.

`Y_pred_samples` $n_{test}\times$ `n_mcmc` matrix of saved posterior samples for β. The ith row of `Y_pred_samples` consists of the posterior samples of the predicted mean response $\mu_{i,test} = E(Y_{i,test})$ for the ith test point. This is not returned if `save_samples=FALSE`.

References

Bai, R. (2023). "Bayesian group regularization in generalized linear models with a continuous spike-and-slab prior." *arXiv pre-print arXiv:2007.07021.*

Polson, N. G., Scott, J. G., and Windle, J. (2013). "Bayesian inference for logistic models using Polya-gamma latent variables." *Journal of the American Statistical Association*, **108**: 1339-1349.

Examples

```
## Generate data
set.seed(1)
X = matrix(runif(200*17), nrow=200)
X_test = matrix(runif(20*17), nrow=20)

n = dim(X)[1]
n_test = dim(X_test)[1]

groups = c(1,1,1,2,2,2,2,3,3,3,4,4,5,5,6,6,6)
true_beta = c(-2,2,2,0,0,0,0,0,0,0,0,0,2.5,-2.5,0,0,0)
Y = crossprod(t(X), true_beta) + rnorm(n)

## Fit SSGL model. You should use the default burn=1000 and n_mcmc=2000

SSGL_mod = SSGL_gibbs(Y, X, groups, family="gaussian", X_test, burn=500, n_mcmc=1000)

## Evaluate results
cbind("True Beta" = true_beta,
      "Posterior Mean" = SSGL_mod$beta_hat,
      "95 CI lower" = SSGL_mod$beta_lower,
      "95 CI upper"= SSGL_mod$beta_upper)

## Predictions on test data
cbind("Predicted E(Y)" = SSGL_mod$Y_pred_hat,
      "95 CI lower" = SSGL_mod$Y_pred_lower,
      "95 CI upper" = SSGL_mod$Y_pred_upper)


## Example with binary logistic regression
## Generate data
set.seed(123)
X = matrix(runif(200*16), nrow=200)
X_test = matrix(runif(50*16), nrow=50)
n = dim(X)[1]
n_test = dim(X)[2]
groups = c(1,1,1,1,2,2,2,2,3,4,4,5,5,6,6,6)
true_beta = c(-2,2,2,-2,0,0,0,0,0,0,0,2.5,-2.5,0,0,0)

## Generate binary responses
eta = crossprod(t(X), true_beta)
Y = rbinom(n, 1, 1/(1+exp(-eta)))

## Fit  SSGL logistic model
SSGL_logistic_mod = SSGL_gibbs(Y, X, groups, family="binomial", X_test)

## Evaluate results
cbind("True Beta" = true_beta,
      "Posterior Mean" = SSGL_logistic_mod$beta_hat,
      "95 CI lower" = SSGL_logistic_mod$beta_lower,
      "95 CI upper"= SSGL_logistic_mod$beta_upper)

## Predictions on test data
cbind("Predicted E(Y)" = SSGL_logistic_mod$Y_pred_hat,
      "95 CI lower" = SSGL_logistic_mod$Y_pred_lower,
      "95 CI upper" = SSGL_logistic_mod$Y_pred_upper)
```

Index