Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

The R Package BHAM: Fast and Scalable Bayeisan Hierarchical Additive Model for High-dimensional Data

Boyi Guo
University of Alabama at Birmingham

Nengjun Yi
University of Alabama at Birmingham

Abstract

\pkg{BHAM} is a freely avaible R pakcage that implments Bayesian hierarchical additive models for high-dimensional clinical and genomic data. The package includes functions that generlized additive model, and Cox additive model with the spike-and-slab LASSO prior. These functions implements scalable and stable algorithms to estimate parameters. \pkg{BHAM} also provides utility functions to construct additive models in high dimensional settings, select optimal models, summarize bi-level variable selection results, and visualize nonlinear effects. The package can facilitate flexible modeling of large-scale molecular data, i.e. detecting succeptable variables and inforing disease diagnostic and prognostic. In this article, we describe the models, algorithms and related features implemented in \pkg{BHAM}. The package is freely avaiable via the public GitHub repository `https://github.com/boyiguo1/BHAM`.

Keywords: additive model, spike-and-slab LASSO, scalable.

1. Introduction

High-dimensional statistics has been an indispensable area of research for its high impact in molecular and clinical data analysis. In recent year, there are continuous efforts to make high-dimensional models more flexible and interpretable, aiming to capture more complex signals. One particular family of such flexible and interpretable models is the additive models where predictors are included in a model in their functional forms. The additive models can help select predictors who have linear or nonlinear effects and provide more accurate prediction when nonlinear effects exist. Guo et al. developed Bayesian hiarchical additive models to analyze continous, categorical and survival outcomes, and demonstrated improved prediction performance compare to the state-of-the-art additive models. In this article, we

introduce the R package `BHAM` that implements the spike-and-slab LASSO additive models and computationally efficient algorithms to fit these models.

The package `BHAM` provides functions for setting up and fitting various spike-and-slab LASSO additive models, including generalized additive models for various continuous and discrete otucoems and Cox survival models for censored survival outcomes. These functions are extended from previously published Bayesian Hierarchical linear models `BhGLM`, and develop upon commonly used R functions `s` in `mgcv` to construct additive functions. Hence, the proposed models shares similar syntax from well-developed packages and provide powerful feasures f these standard tools. The sytax can be easily followed and provide user friendliness. In addition, the algorithms implemented in `BHAM` is easily scalable, particularly suitable for fitting high-dimensional models. In the package, we also provide a series utility functions, for example . Hence, BHAM provides xxxx and is helpful for xxx.

1.1. Literature Review

We enlist current available packages that have similar functionality, i.e. modeling to the best of our knowledge. To note, we don't list packages that are unable of handling high-dimensional data, for example the well known R package `mgcv`, and high-dimensional packages that requires extra steps to construct the design matrix of functional form of predictors (Such implementation can be found with grouped sparse models, for example `SGL`.)

? Summarized the software development of additive models in high-dimensional data analysis before 2013.

Generalized Additive Model

- `COSSO`
- `spikeSlabGAM`
- `sparseGAM`

Additive Cox Proportional Hazard Model

- `COSSO`
- `tfCox`

The **BHAM** package provides a scalable solution for fitting high-dimensional generalized additive model and additive Cox model using spike-and-slab LASSO priors or other regularized priors, including continuous spike-and-slab priors, Student' T priors and double exponential priors. It fits linear, logistic, poisson and Cox regression models. The specification of the additive functions follows a popular syntax implemented in `mgcv`. Ancillary functions are provided, including cross-validation, model summary, and visualization.

In this article, we focus on the packages that can directly construct additive models for high-dimensional data analysis, instead of requiring additional step of constructing design matrix of functional form of the variables before fitting a sparse model.

There are other methods to model survival outcome and provides proporitonal hazards interpretation, for example **?** provides a link-based survival additive model for mixed censoring in package `GJRM`.

2. Models and algorithms

In this section, we describe the Bayeisan hiearchical additive model that `BHAM` implements. The basic idea is to impose the two-part spike-and-slab LASSO prior **?** on each additive function in the model. The choices of model includes generalized additive model and Cox proportional hazard model. The proposed two-part spike-and-slab LASSO prior consists of a spike-and-slab LASSO prior for the linear space coefficient β_j of a additive function $B_j(X_j)$ of the jth variables, and a modified group spike-and-slab LASSO prior for the nonlinear space coefficients $\beta^*_{jk}, k = 1, ..., K_j$ of the jth additive function.

$$\begin{aligned} \beta_j|\gamma_j, s_0, s_1 &\sim (1-\gamma_j)DE(0, s_0) + \gamma_j DE(0, s_1) \\ \beta^*_{jk}|\gamma^*_j, s_0, s_1 &\overset{\text{iid}}{\sim} (1-\gamma^*_j)DE(0, s_0) + \gamma^*_j DE(0, s_1), k = 1, \ldots, K_j. \end{aligned} \tag{1}$$

To note, the model matrix of the additive function undergoes a reparameterization process that absorbs the smoothing penalty via eigendecomposition. Meanwhile, the reparameterization also isolate the linear and nonlinear spaces of the additive function, allowing different shrinkages on the two spaces and motivates signal selection via the linear space and function smoothing of the nonlinear space. The spike-and-slab prior use the binary indicator γ to indicate if the the corresponding variable is included in the model. Nevertheless, this selection finalized based on soft-thresholding. the spike-and-slab LASSO prior makes this selection process easier by shrinking the coefficient to exactly 0. In the two-part SSL prior, each additive function have two indicators γ_j and γ^*_j, controlling the linear and nonlinear component selection. Effect hierarchy was implemented via the conditional priors of to ensure the the linear component is more likely to be selected than the nonlinear components.

$$\gamma_j|\theta_j \sim Bin(1, \theta_j) \qquad\qquad \gamma^*_j|\gamma_j, \theta_j \sim Bin(1, \gamma_j\theta_j). \tag{2}$$

The inclusion probability parameter θ_j have a beta prior to allow adaptive shrinkage.

To fit the model in a efficient and scalable fashion, we implement the EM-coordinate descent algorithm. The EM-coordinant descent algorithm estimates maximum a posteriori of the coefficients by optimizing the log joint posterior density function. The algorithm re-writes the spike-and-slab LASSO prior as a double exponential distribution with conditional scale parameter, and leverages the relationship between double exponential prior and l_1 penalty. Hence, the log joint posterior density function can be expressed as the summation of a l_1 penalized likelihood function and log beta posterior density. Nevertheless, the nuances parameters $\boldsymbol{\gamma}$ are unknown and requires the EM algorithm to address. In each iterations of the EM procedure, we update the expectation of the log joint posterior density function with respect to the nusance parameters, calculate the penalties based on the estimation from previous iteration, and optimize the penalized likelihood and the posterior density with coordinate descent algorithm and closed-form calculation for the coefficients. The process iterates until convergence. Cross-validation is used to choose the optimal model. We defer **??**to for full description of GAM algorithm and Cox additive model algorithm.

3. Features

In this section, we demonstrate how to fit Bayesian hierarchical additive model with two-part spike-and-slab LASSO prior, and introduce the model tuning, diagnostic and other utility functions for visualize additive functions, bi-level selection.

In this section, we describe to the users the workflow for of fitting a high-dimensional additive model with the two-part spike-and-slab LASSO prior in `BHAM`. Specifically, we introduce how to 1) prepare the high-dimensional design matrix for fitting the proposed model, 2) fit generalized additive model, 3) Model tuning and performance assessment, and 4) visualize the bi-level variable selection.

3.1. Installation

To install the latest development version of `BHAM` package from **GitHub**, type the following command in R console:

```
R> if (!require(devtools)) install.packages("devtools")
R> if(!require(BHAM)) devtools::install_github("boyiguo1/BHAM", build_vignettes = FALSE)
```

You can also set `build_vignettes=TRUE` but this will slow down the installation drastically (the vignettes can always be accessed online anytime at boyiguo1.github.io/BHAM/articles).

3.2. Preliminaries

We use a simulated data set to demonstrate our package. The data generating mechanism is motivated by Bai (citation) and programmed in the function `sim_Bai`: we assume there are $p = 10$ predictors where the first four predictors have effects on the outcome (see functions below), and the rest of predictors don't, i.e $B_j(x_j) = 0, j = 5, \ldots, p$. [TODO: Insert functions here on what the equations are]. With the data generating mechanism, we simulate two datasets with the binary outcome from Bernoulli trials with logit link function. To note, the function `sim_Bai` can also simulate Gaussian and Poisson outcomes using the same data generating mechanism. The sample sizes of these two datasets are 500 and 1000 for training and testing respectively. The following code section creates the training and testing datasets.

```
R> library(BHAM)
R> set.seed(1) ## simulate some data...
R> n_train <- 500
R> n_test <- 1000
R> p <- 10
R> # Train Data
R> train_dat <- sim_Bai(n_train, p)
R> dat <- train_dat$dat %>% data.frame
R>
R> # Test Data
R> test_tmp <- sim_Bai(n_test, p)
R> test_dat <- test_tmp$dat %>% data.frame
```

The first ten observation of the data set look like below

```
           x1         x2         x3          x4         x5          x6
1   1.5579537 -1.1346302  0.5205997  0.73911492 -1.8054836 -0.88614959
2  -0.7292970  0.7645571  0.3775619  0.38660873 -0.6780407 -1.92225490
3  -1.5039509  0.5707101 -0.6236588  1.29639717 -0.4733581  1.61970074
4  -0.5667870 -1.3516939 -0.5726105 -0.80355836  1.0274171  0.51926990
5  -2.1044536 -2.0298855  0.3125012 -1.60262567 -0.5973876 -0.05584993
6   0.5307319  0.5904787 -0.7074278  0.93325097  1.1598494  0.69641761
7   1.6176841 -1.4130700  0.5212035  1.80608925 -1.3332269  0.05351568
8   1.1845319  1.6103416  0.4481880 -0.05650363 -0.9257557 -1.31028350
9   1.8763334  1.8404425 -0.5053226  1.88591132 -1.0744951 -2.12306606
10 -0.4557759  1.3682979 -0.2066122  1.57838343 -1.4511165 -0.20807859
           x7          x8          x9         x10 y
1   0.8500435  1.13496509  0.07730312 -0.6264538 0
2  -0.9253130  1.11193185 -0.29686864  0.1836433 1
3   0.8935812 -0.87077763 -1.18324224 -0.8356286 0
4  -0.9410097  0.21073159  0.01129269  1.5952808 0
5   0.5389521  0.06939565  0.99160104  0.3295078 0
6  -0.1819744 -1.66264885  1.59396745 -0.8204684 0
7   0.8917676  0.81083998 -1.37271127  0.4874291 0
8   1.3292082 -1.91234580 -0.24961093  0.7383247 1
9  -0.1034661 -1.24675343  1.15942453  0.5757814 0
10  0.6150646  0.99815445 -1.11422235 -0.3053884 0
```

3.3. Set up Design Matrix of additive functions

Given the raw data, we would like to translate the additive functions to the their matrix form. The challenge here is to allow a customizable and convenient way to specify the high-dimensional model. Our solution here is to use a data frame to accomodate each predictor in the raw data set, and allows each predictor have their spline function specified. There are three columns for this formulat specification data frame, including `Var` `Func`, `Args`. The `Var` column hosts the variable name; the `Func` column hosts the spline function following the commonly used generalized additive model `mgcv`; the `Args` column hosts the detail specification of the spline function. The data frame can be constructed manually for low-dimensional settings and also be manipulated easily when the number of spline components grows to tens or hundreds. See the examples below.

```
R> # Low-dimensional setting
R> mgcv_df <- dplyr::tribble(
+   ~Var, ~Func, ~Args,
+   "X1",  "s",     "bs='cr', k=5",
+   "X2",  "s",     NA,
+   "X3",  "s",    "",
+ )
R>
R> # High-dimensional setting
R> mgcv_df <- data.frame(
```

```
+   Var = setdiff(names(dat), "y"),
+   Func = "s",
+   Args ="bs='cr', k=7"
+ )
```

After having the model specification data frame, the next task is to construct the overall design matrix. We provide a function `construct_smooth_data` to construct the design matrix for each predictor according to their spline specification iteratively, and binding all design matrices together with a systematic naming convention. The linear component of the spline function is named with the suffix `.null` and the nonlinear components are named with the suffix `.pen`. In `construct_smooth_data`, we take three steps of matrix manipulation via the `smoothCon` from the package `mgcv`: 1) linear constraints, 2) eigendecomposition of the smoothing matrix S to isolate linear and nonlinear spaces, 3) scaling of the design matrix such that the coefficients are on the same scale. As we use `mgcv::smoothCon` to decode the spline specification, we carry over the ability to work with user-defined spline functions as long as it follows `mgcv` standard.

The `construct_smooth_data` function have two arguments, the model specification data frame and the raw data, and return the finalized design matrix `data` and the smooth specification functions `Smooth` which will later be used to construct the design matrix of the new datasets for the prediction purpose.

```
R> train_sm_dat <- BHAM::construct_smooth_data(mgcv_df, dat)
R> train_smooth <- train_sm_dat$Smooth
R> train_smooth_data <- train_sm_dat$data
```

3.4. Fitting the Bayesian Hierarchical model

With the additive function design matrix constructed, we are ready to fit the Bayesian hierarchical model with the two-part spike-and-slab LASSO prior for smooth function. The model fitting algorithm, implementing the EM-coordinate descent algorithm, is wrapped in the function `bamlasso`. The necessary arguments are `x` for the design matrix, `y` for the outcome, `family` for the family distribution of the outcome, and `group` for the additive functions. We provide a utility function `make_group` to automate the grouping, taking the column names from the design matrix. It generates a list of vectors containing the bases of each additive function. Another important argument is `ss`, which is a vector of length 2 for the spike and slab components of the spike-and-slab LASSO scale parameters, i.e. the mixture double exponential distribution. The argument `ss` defaults to a spike double exponential density with scale parameter 0.04, and a slab double exponential density with scale parameter 0.5, which is a general starting prior based on empirical evidence.

```
R> bham_mdl <- bamlasso(x = train_smooth_data, y = dat$y,
+                       family = "binomial",
+                       group = make_group(names(train_smooth_data)))
```

Tuning via Cross-validation

With the specified `ss` argument, the function `bamlasso` fit the asked model. Nevertheless, it

is not necessary the optimal model. To select the optimal model, we employ a tuning step via cross validation, which is implemented in the function `tune.bgam`. The main arguments are the previously fitted model where the model data, additive function specifications are stored, a sequence of spike density scale parameter s_0, and number of folds. The following example shows to use five-fold cross validation to examine a vector of s_0 options, from 0.005 to 0.1 with 0.01 increment. Currently, we don't consider to examine values of the slab density scale parameter s_1 for computational economy, as the previously literature shows s_1 has modest impact on the model performance. The tuning function also allows nested cross-validation by allowing running multiple cross-validation via `ncv` and user-specified folds via `foldid`.

```
R> s0_seq <- seq(0.005, 0.1, 0.01)
R> cv_res <- tune.bgam(bham_mdl, nfolds = 5, s0= s0_seq, verbose = FALSE)

Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.035 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.006 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.006 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.011 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.016 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.009 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.012 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.007 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.005 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.005 minutes
```

The cross-validation tuning function returns different performance metrics, including deviance, mean squared error, mean absolute error, area under the curve, misclassifcation for binary outcome, and concordance statistics for survival outcome. The following shows the

cross-validated performance metrics for the first five values of the s_0 sequence using out-of-bag samples.

```
R> head(cv_res, 5)
```

```
     s0 deviance   auc   mse   mae misclassification
1 0.005  435.044 0.809 0.141 0.281             0.212
2 0.015  376.082 0.865 0.120 0.253             0.166
3 0.025  352.728 0.883 0.111 0.238             0.148
4 0.035  349.896 0.882 0.110 0.226             0.154
5 0.045  346.670 0.884 0.109 0.223             0.158
```

Fit the Optimal Model

```
R> s0_min <- cv_res$s0[which.min(cv_res$deviance)]
R> bham_final <- bamlasso(x = train_smooth_data, y = dat$y, family = "binomial", group = m
+                         ss = c(s0_min, 0.5))
```

3.5. Model Summary

Variable Selecrtion

```
R> bamlasso_vs_part <- bamlasso_var_selection(bham_final)
```

Curve Plotting

```
R> plot_smooth_term(bham_final, "x1", train_smooth,
+                       min = min(dat[, "x1"])-0.1,
+                       max = max(dat[, "x1"]) + 0.1)
```

3.6. Prediction

Due to the reparameterization in the design matrix, it creates complication with setting up the design matrix for the test data. Hence, we write an wrapping function for `PredictMat` from `mgcv`.

```
R> train_smooth <- train_sm_dat$Smooth
R> test_sm_dat <- make_predict_dat(train_sm_dat$Smooth, dat = test_dat)

R> if(!require("devtools")) install.packages("devtools")
R> if(!require("BhGLM")) devtools::install_github("nyiuab/BhGLM ")
R>
```

```
R> BhGLM::measure.bh(bham_final, test_sm_dat, test_dat$y)
R>
R> # pred <- predict.glm(.mdl$mdl, newdata = test_sm_dat, type = "response")
R> # measure.glm(test_dat$y, pred, family = fam_fun$family)
```

3.7. Other Functions

4. Discussion

5. Reference

Affiliation:

Boyi Guo
University of Alabama at Birmingham
1665 University Blvd
Birmingham, AL 35294-0002 USA
E-mail: boyiguo1@uab.edu
URL: http://boyiguo1.github.io

Nengjun Yi
University of Alabama at Birmingham
1665 University Blvd
Birmingham, AL 35294-0002 USA
E-mail: nyi@uab.edu

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd