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 years, 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 the model as an additive function. These high-dimensional additive models serve for two purposes: variable selection and outcome prediction. In high-dimensional statistics, it is common to assume there is only a small subset of predictors that have effects on the outcome, also known as the signal sparsity assumption. (Bühlmann and van de Geer 2011) The high-dimensional additive models select not only the predictors who

have linear associations with the outcome, but also those who inform the outcome prediction with nonlinearity. As a result, they provide more flexible effect modeling and improve prediction accuracy compared to high-dimensional linear models.

There are many proposals on high-dimensional additive models. The main idea of these proposals focus on the application of grouped sparse penalties, for example, group LASSO penalty (Ravikumar, Lafferty, Liu, and Wasserman 2009; Huang, Horowitz, and Wei 2010) and group SCAD penalty (Wang, Chen, and Li 2007; Xue 2009), on the coefficients of additive functions. There methods are developed primarily for variable selection and may provide inaccurate estimation of the underlying functions due to the excess shrinkage of the sparsity penalty (Scheipl, Kneib, and Fahrmeir 2013). Thus, the prediction performance will be affected. In addition, these methods take an "all-in-all-out" approach for variable selection, and fails to answer if the underlying signals are linear or nonlinear. To address these shortcomings, Guo and his colleagues proposed the two-part spike-and-slab LASSO prior for generalized additive models (Guo, Jaeger, Rahman, Long, and Yi 2022) and additive Cox proportional hazards models (Guo and Yi 2022). Instead of using the computationally prohibitive Markov chain Monte Carlo approximations, optimization-based EM-Coordinate Descent algorithms are developed for model fitting. Monte Carlo studies and real data analysis demonstrate improved prediction and computation performances compare to the state-of-the-art additive models.

In this article, we introduce an R package `BHAM` that implements the spike-and-slab LASSO additive models and computationally efficient algorithms. Notably, `BHAM` provides functions for setting up and fitting various spike-and-slab LASSO additive models, including generalized additive models for various continuous and discrete outcomes and Cox proportional hazards models for censored survival outcomes. The specification of additive functions follows a popular syntax implemented in `mgcv` (Wood 2018). We provide a parser function that translates high-dimensional predictors names and their corresponding additive functions to model formulas, rendering convenience to model large datasets with hundreds and thousands of predictors. Other ancillary functions include cross-validation, model summary, and effect visualization. Our objective with `BHAM` is to offer a friendly user experience that emphasizes statistical validity, computational scalability and utility flexibility for high-dimension additive models.

There are other R packages that facilitate flexible modeling of complex signals via additive models for high-dimensional data analysis. The R package `COSSO` (Zhang and Lin 2013) implements smoothing spline ANOVA models with the component selection and smoothing operator to analyze generalized and survival outcomes. The packages `spikeSlabGAM` (Scheipl 2011) and `sparseGAM` (Bai 2021a) fits generalized additive regression models with spike-and-slab and spike-and-slab LASSO priors respectively; nevertheless, both packages does not offer analytic support to model time-to-event outcome. The package `tfCox` (Wu and Witten 2019b) implements additive Cox proportional hazards models with trend filtering. Scheipl *et al.* (2013) summarized some other scripts or packages to fit additive models published before 2013, including `spam` (Ravikumar *et al.* 2009), `hgam` (Meier, van de Geer, and Bühlmann 2009) and `hypergsplines` (Bové, Held, and Kauermann 2011); unfortunately, these tools are hardly available now due to maintenance issues. One inconvenience shared by the packages is the limited ability to customize additive functions due to the difficulty to formulate the high-dimensional model. In the proposed `BHAM`, we address this challenge by providing an interface that parses a data frame of spline function specification to model formula, and hence provide

greater flexibility compared to previous packages.

The remainder of this paper is as follows. In Section 2, we briefly describe the spike-and-slab LASSO prior of smooth functions and the computationally efficient EM-Coordinate Descent algorithm for model fitting. Section 3 demonstrates the analytic pipeline to analyze high-dimensional data with the R package `BHAM`. We deliver the conclusion in Section 4. Fore more details and examples about `BHAM`, we encourage the readers to visit https://boyiguo1.github.io/BHAM/.

2. Models and Algorithms

In this section, we describe the Bayesian hierarchical additive model that `BHAM` implements. The key idea is to impose the two-part spike-and-slab LASSO prior Guo *et al.* (2022) on each additive function in generalized models or Cox proportional hazards models. For the additive function $B_j(X_j)$ of the jth variable, the proposed two-part spike-and-slab LASSO prior consists of a spike-and-slab LASSO prior for the linear space coefficient β_j and a modified group spike-and-slab LASSO prior for the nonlinear space coefficients $\beta^*_{jk}, k = 1, ..., K_j$,

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

To note, the model matrix of each additive function undergoes a reparameterization process in advance, which eigendecomposes the smoothing penalty matrix to isolate the linear and nonlinear spaces of the additive function (Wood 2017). The reparameterization greatly reduces the complexity to formulate the sparsity-smoothness penalty (Meier *et al.* 2009) and allows different shrinkage on the two spaces. The shrinkage on the linear space manages the variable selection, while the shrinkage on the nonlinear space emphasizes the adequate smoothing of nonlinear effect interpolation. In addition, the isolation of linear and nonlinear spaces motivates the bi-level functional selection, i.e. the selection of additive functions and the selection of nonlinear effects. In the proposed prior, each additive function has two indicators γ_j and γ^*_j, controlling the linear and nonlinear component selection. Effect hierarchy was implemented via the conditional priors of γ^*_j to ensure the the linear component is more likely to be selected than the nonlinear components.

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

We further impose a beta prior on the inclusion probability parameter θ_j to allow locally adaptive shrinkage. For simplicity, θ_j follows a uniform(0, 1) prior. Compared to previous spike-and-slab priors (Scheipl, Fahrmeir, and Kneib 2012; Bai 2021b) for additive functions, the proposed prior provides three advantages. First of all, the proposed prior allows bi-level functional selection instead of an "all-in-all-out" approach for variable selection. Secondly, the proposed prior offers a natural selection procedure by shrinking unnecessary coefficients to exactly 0, contrasting to soft-thresholding the inclusion probability. Last but not least, the proposed prior is easily applicable to model different types of outcomes, including time-to-event outcomes via Cox proportional hazards models.

To fit the proposed models in a efficient and scalable fashion, we develop the EM-coordinate descent algorithm. The EM-coordinant descent algorithm estimates maximum a posteriori

of coefficients by optimizing the log joint posterior density function. The algorithm formulates the spike-and-slab LASSO prior as a double exponential distribution with a conditional scale parameter. It further leverages the relationship between double exponential prior and l_1 penalty and expresses the log joint posterior density function as the summation of a l_1 penalized likelihood function (and l_1 penalized partial likelihood function for Cox proportional hazards models) and log beta posterior densities. Because the nuisance parameters $\boldsymbol{\gamma}$ are unknown, we instead optimize the conditional expectation of log joint posterior density function via an EM procedure (Dempster, Laird, and Rubin 1977). In each iterations of the EM procedure, we update the expectation of the log joint posterior density function with respect to the nuisance 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 Guo *et al.* (2022); Guo and Yi (2022) for more detail on the GAM algorithm and the additive Cox model algorithm.

3. Analytic Pipline Using BHAM

In this section, we demonstrate how to fit Bayesian hierarchical additive model with two-part spike-and-slab LASSO prior using the package `BHAM`. Specifically, we introduce how to 1) prepare the high-dimensional design matrix for fitting the proposed model, 2) fit generalized additive model, 3) tune models and assess model performance, and 4) visualize the bi-level variable selection.

3.1. Installation

To install the latest development version of the `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 (2021b) 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 $B_j(x_j) = 0, j = 5, \dots, p$.

$$
\begin{aligned}
B_1(x_1) &= 5\sin(2\pi x_1) & B_2(x_2) &= -4\cos(2\pi x_2 - 0.5) \\
B_3(x_3) &= 6(x_3 - 0.5) & B_4(x_4) &= -5(x_4^2 - 0.3)
\end{aligned}
$$

Using this data generating mechanism, we simulate two datasets of binary outcomes with the logit link function from Bernoulli trials. 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 observations of the training 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 provide convenient way to specify the high-dimensional model with enough flexibility to customize the additive functions. Our solution here is to use a data frame to accommodate each predictor in the raw data set and allow each predictor have their spline function specified respectively. There are three columns for this model 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 syntax from `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. Then we bind the design matrices of all spline functions together with a systematic naming convention. The linear component of each 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) set up linear constraints for identifiability, 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 has two arguments, the model specification data frame and the raw data. It returns 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 prediction.

```
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. 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, by organizing 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 scale parameters of the spike and slab densities. To recall, the spike-and-slab LASSO prior can be formulated as the mixture of two double exponential distributions of mean 0, and hence has two scale parameters. 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. These scale parameters is a general starting value based on empirical evidence (Tang, Shen, Li, Zhang, Wen, Qian, Zhuang, Shi, and Yi 2018; Tang, Lei, Zhang, Yi, Guo, Chen, Shen, and Yi 2019).

```
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 model. Nevertheless, the fitted model may not be 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 increments. Currently, we don't consider the examination of the slab density scale parameter s_1 for computational economy. Previously literature (Tang, Shen, Zhang, and Yi 2017a,b) 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: 0.011 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.031 minutes
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.03 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: 0.005 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
Fitting ncv*nfolds = 5 models:
1 2 3 4 5
 Cross-validation time: 0.008 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: 0.005 minutes
```

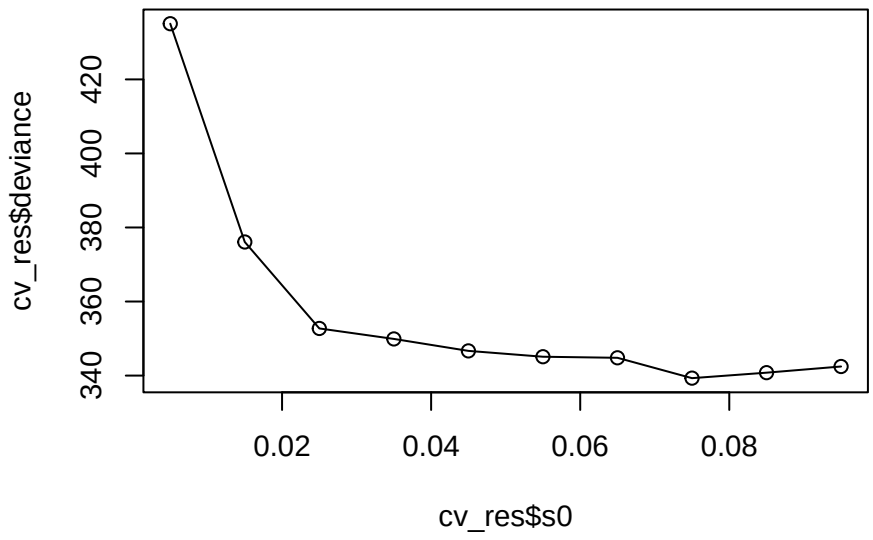
The cross-validation tuning function returns different performance metrics, including deviance, mean squared error, mean absolute error, area under the curve, misclassifcation for binary outcome, and concordance statistics for survival outcome. The following shows the cross-validated performance metrics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

Here we want to caution the reader, if the performance metric varies monotonically with the candidate s_0 values, it would be better to examine a broader range of candidate s_0 values, as the sequence contains a local optimal where the global optimal is not reached yet. Using some visual aid to examine the s_0 and performance metric relationship would be more helpful.

```
R> plot(cv_res$s0, cv_res$deviance)
R> lines(cv_res$s0, cv_res$deviance)
```



With the cross-validation results, we can choose from all the candidate values of s_0 and select the one with the best performance using the preferred metrics. For example, we can use the s_0 value that gives the minimum cross-validated deviance and re-fit the model. Hence, this would be 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 = make_group(names(train_smooth_data)),
+                         ss = c(s0_min, 0.5))
```

To note, it is a convention to use some predictive metrics to select the best performed model among all the candidate values for both predictive purpose and variable selection purpose. However, previous literature (Wu and Witten 2019a) shows that when using predictive metrics to select model for variable selection purpose, the variable selection performance may not be optimal.

3.5. Varible Selection and Curve Intropolation

Variable Selecrtion

We provide a function to summarize the variable selection result of a produced model, namely `bamlasso_var_selection`. The input of the function is a fitted BHAM model, and the output is a list containing two components, `parametric` and `non-parametric`. The `parametrc` component is a vector contains the selected variables that were fitted in the model in their parameteric form, i.e. not specified via additive functions. The `non-parametric` component

contains a data frame with 3 columns, `Variable`, `Linear`, `Nonlinear`. While `Variable` column includes the variable names of selected additive functions, `Linear` and `Nonlinear` columns are logical vectors indicating if the linear and nonlinear components of additive functions are included in the model respectively.

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

Here, we shows the variable selection result from previously tuned model. Since, the model didn't include any variables in their parametric form. Hence, the `parametric` is an empty vector. Meanwhile, the `nonparametric` data frame contains the bi-level selection result.

```
R> bamlasso_vs_part
```

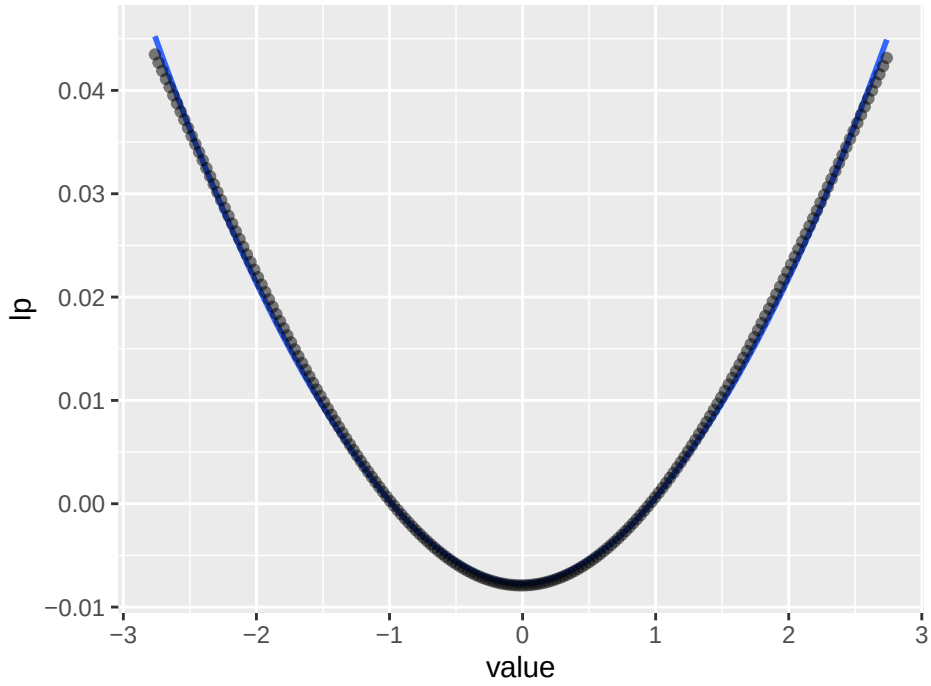
```
$Parametric
character(0)

$`Non-parametric`
  Variable Linear Nonlinear
1       x1  FALSE      TRUE
2       x2  FALSE      TRUE
3       x3   TRUE     FALSE
4       x4  FALSE      TRUE
5       x5  FALSE      TRUE
6       x7  FALSE      TRUE
7       x9  FALSE      TRUE
8      x10  FALSE      TRUE
```

Curve Plotting

We also provide a utility function `plot_smooth_term` to plot the estimated functions. The function takes in the fitted model, the variable name, the previously constructed smooth objective to construct the design matrix, minimum and maximum of the range of the predictors. The function outputs a `ggplot` object to show the estimated curve.

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

```
`geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



3.6. Prediction

To predict new datasets, we need to go through the same two-step procedure to produce the data matrix as previously when building the model. First of all, we need to translate the new dataset to their matrix form using the function `make_predict_dat`. This step is necessary because of the reparameterization of the design matrix. The function `make_predict_dat` is based on the function `PredictMat` from `mgcv`. The function asks for an additional input argument besides the new dataset, which is the Smooth object when constructing the design matrix for the training data. The output of the function is the new data matrix of the new dataset with conformable dimension and variable name. We show the first six columns of the first five observations in the following example.

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

     x1.pen1     x1.pen2    x1.pen3     x1.pen4    x1.pen5   x1.null1
1  0.2105822 -0.51339049 -0.7087016  1.48599984 -1.6282845 -0.6187856
2  0.1401345 -0.04244866  0.4101826 -2.49600416 -0.5234851  1.2762868
3 -0.1157559 -0.28695472  0.3280699  0.08111065  9.0232660  3.6778234
4 -0.1551403 -0.57575563  1.1032603 -2.71644734  1.1193043  1.8444025
5  0.1841429 -0.50899800 -0.7593021  1.41065378 -1.6744694 -0.5829544
```

With the new dataset in the conformable design matrix format, we can easily produce the prediction using the function `predict`. Under the hood, we use `predict.glmnet` to produce the prediction, and hence, it is robust. For the GLM, we can produce the linear predictors using `type = "link"` and the fitted probability/mean using `type = "response"`.

```
R> bham_final$offset = 0
R> pred_res <- predict(bham_final, newx = as.matrix(test_sm_dat),
+                      newoffset = 0, type = "link")
```

To note, we suggest to use `BhGLM::measure.bh` to provide a quick prediction performance evaluation for the new dataset.

```
R> if(!require("devtools")) install.packages("devtools")
R> if(!require("BhGLM")) devtools::install_github("nyiuab/BhGLM")
R>
R> BhGLM::measure.bh(bham_final, as.matrix(test_sm_dat), test_dat$y)
```

4. Discussion

In this article, we introduce the R package `BHAM` to fit Bayesian Hierarchical additive models with two-part spike-and-slab LASSO prior for high-dimensional data analysis. The R package can be widely used to analyze large-scale molecular and clinical data with the flexibility to model both linear and nonlinear signals, and hence provide improved prediction accuracy. Meanwhile, compared to the more complicated machine learning method, the additive models can provide more interpretable inference of the underlying signals. In addition, the two-part spike-and-slab LASSO prior for smooth function and the EM-CD algorithm provides a natural solution to the bi-level selection problem, without further requirement of thresholding or hypothesis testing. Fitting a high-dimensional Bayesian model is normally computationally intensive. We provide an economic solution by integrate coordinate descent algorithm with the EM procedure. The implementation of the algorithm leverage some commonly used modeling interface form the standard R packages and hence granting robustness.

To help the users to familiarize the utilities of `BHAM`, we provide a analysis pipeline in this manuscript. We demonstrate the construction of the design matrix, model fitting and tuning, signal selection and visualization, and prediction via the analysis of a simulated data set. Due to the space constraint, we can't showcase all the functionality offered by `BHAM` for example fitting a Cox proportional hazard model, time-varying effect model, or fitting the model with the EM-Newton or EM-IWLS algorithms. We recommand the user to visit an interactive website for more details via https://boyiguo1.github.io/BHAM/.

Optimal goal is to provide interface for optimal customizability.

References

Bai R (2021a). *sparseGAM: Sparse Generalized Additive Models*. R package version 1.0.99, URL https://CRAN.R-project.org/package=sparseGAM.

Bai R (2021b). "Spike-and-Slab Group Lasso for Consistent Estimation and Variable Selection in Non-Gaussian Generalized Additive Models." *arXiv:2007.07021 [stat]*. ArXiv: 2007.07021, URL http://arxiv.org/abs/2007.07021.

Bové D, Held L, Kauermann G (2011). "Mixtures of g-Priors for Generalised Additive Model Selection with Penalised Splines." doi:10.48550/ARXIV.1108.3520. URL https://arxiv.org/abs/1108.3520.

Bühlmann P, van de Geer S (2011). *Statistics for High-Dimensional Data*. Springer Berlin Heidelberg. doi:10.1007/978-3-642-20192-9. URL http://dx.doi.org/10.1007/978-3-642-20192-9.

Dempster AP, Laird NM, Rubin DB (1977). "Maximum Likelihood from Incomplete Data Via the*EM*Algorithm." *Journal of the Royal Statistical Society: Series B (Methodological)*, **39**(1), 1–22. doi:10.1111/j.2517-6161.1977.tb01600.x. URL http://dx.doi.org/10.1111/j.2517-6161.1977.tb01600.x.

Guo B, Jaeger BC, Rahman AKMF, Long DL, Yi N (2022). "Spike-and-Slab LASSO Generalized Additive Models and Scalable Algorithms for High-Dimensional Data Analysis." *arXiv:2110.14449 [stat]*. ArXiv: 2110.14449, URL http://arxiv.org/abs/2110.14449.

Guo B, Yi N (2022). "A scalable and flexible Cox proportional hazards model for high-dimensional survival prediction and functional selection." (arXiv:2205.11600). ArXiv:2205.11600 [stat], URL http://arxiv.org/abs/2205.11600.

Huang J, Horowitz JL, Wei F (2010). "Variable selection in nonparametric additive models." *The Annals of Statistics*, **38**(4). doi:10.1214/09-aos781. URL http://dx.doi.org/10.1214/09-aos781.

Meier L, van de Geer S, Bühlmann P (2009). "High-dimensional additive modeling." *The Annals of Statistics*, **37**(6B). doi:10.1214/09-aos692. URL http://dx.doi.org/10.1214/09-AOS692.

Ravikumar P, Lafferty J, Liu H, Wasserman L (2009). "Sparse additive models." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **71**(5), 1009–1030. doi:10.1111/j.1467-9868.2009.00718.x. URL http://dx.doi.org/10.1111/j.1467-9868.2009.00718.x.

Scheipl F (2011). "spikeSlabGAM: Bayesian Variable Selection, Model Choice and Regularization for Generalized Additive Mixed Models in R." *Journal of Statistical Software*, **43**(14), 1–24. doi:10.18637/jss.v043.i14.

Scheipl F, Fahrmeir L, Kneib T (2012). "Spike-and-Slab Priors for Function Selection in Structured Additive Regression Models." *Journal of the American Statistical Association*, **107**(500), 1518–1532. doi:10.1080/01621459.2012.737742. URL http://dx.doi.org/10.1080/01621459.2012.737742.

Scheipl F, Kneib T, Fahrmeir L (2013). "Penalized likelihood and Bayesian function selection in regression models." *AStA Advances in Statistical Analysis*, **97**(4), 349–385. doi:10.1007/s10182-013-0211-3. URL http://dx.doi.org/10.1007/s10182-013-0211-3.

Tang Z, Lei S, Zhang X, Yi Z, Guo B, Chen JY, Shen Y, Yi N (2019). "Gsslasso Cox: A Bayesian hierarchical model for predicting survival and detecting associated genes by incorporating pathway information." *BMC Bioinformatics*, **20**(1), 1–15. ISSN 14712105. doi:10.1186/s12859-019-2656-1.

Tang Z, Shen Y, Li Y, Zhang X, Wen J, Qian C, Zhuang W, Shi X, Yi N (2018). "Group spike-And-slab lasso generalized linear models for disease prediction and associated genes detection by incorporating pathway information." *Bioinformatics*, **34**(6), 901–910. ISSN 14602059. doi:10.1093/bioinformatics/btx684.

Tang Z, Shen Y, Zhang X, Yi N (2017a). "The spike-and-slab lasso Cox model for survival prediction and associated genes detection." *Bioinformatics*, **33**(18), 2799–2807. ISSN 14602059. doi:10.1093/bioinformatics/btx300.

Tang Z, Shen Y, Zhang X, Yi N (2017b). "The spike-and-slab lasso generalized linear models for prediction and associated genes detection." *Genetics*, **205**(1), 77–88. ISSN 19432631. doi:10.1534/genetics.116.192195.

Wang L, Chen G, Li H (2007). "Group SCAD regression analysis for microarray time course gene expression data." *Bioinformatics*, **23**(12), 1486–1494. doi:10.1093/bioinformatics/btm125. URL http://dx.doi.org/10.1093/bioinformatics/btm125.

Wood S (2018). *Mixed GAM computation vehicle with automatic smoothness estimation*. R package version 1.8-40, URL https://cran.r-project.org/web/packages/mgcv/index.html.

Wood SN (2017). *Generalized additive models: an introduction with R*. Chapman & Hall/CRC texts in statistical science, second edition edition. CRC Press/Taylor & Francis Group, Boca Raton.

Wu J, Witten D (2019a). "Flexible and Interpretable Models for Survival Data." *Journal of Computational and Graphical Statistics*, **28**(4), 954–966. ISSN 1061-8600. doi:10.1080/10618600.2019.1592758. URL https://www.tandfonline.com/doi/full/10.1080/10618600.2019.1592758.

Wu J, Witten D (2019b). *tfCox: Fits Piecewise Polynomial with Data-Adaptive Knots in Cox Model*. R package version 0.1.0, URL https://CRAN.R-project.org/package=tfCox.

Xue L (2009). "CONSISTENT VARIABLE SELECTION IN ADDITIVE MODELS." *Statistica Sinica*, **19**(3), 1281–1296. ISSN 10170405, 19968507. URL http://www.jstor.org/stable/24308957.

Zhang HH, Lin CY (2013). *cosso: Fit Regularized Nonparametric Regression Models Using COSSO Penalty*. R package version 2.1-1, URL https://CRAN.R-project.org/package=cosso.

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