Sparse Additive Interaction Learning

Sahir R Bhatnagar[1,2], Yi Yang[4], Amanda Lovato[2], and

Celia MT Greenwood[1,2,5]

[1]Department of Epidemiology, Biostatistics and Occupational Health, McGill University

[2]Lady Davis Institute, Jewish General Hospital, Montréal, QC

[4]Department of Mathematics and Statistics, McGill University

[5]Departments of Oncology and Human Genetics, McGill University

May 30, 2018

1 Introduction

Computational approaches to variable selection have become increasingly important with the advent of high-throughput technologies in genomics and brain imaging studies, where the data has become massive, yet where it is believed that the number of truly important variables is small relative to the total number of variables. Although many approaches have been developed for main effects, there is a consistent interest in powerful methods for estimating interactions, since interactions may reflect important modulation of a genomic system by an external factor. Accurate capture of interactions may hold the potential to better understand biological phenomena and improve prediction accuracy.

Furthermore, the manifestations of disease are often considered to be the result of changes in entire biological networks whose states are affected by a complex interaction of genetic and environmental factors [1]. However, there is a general deficit of such replicated interactions in the literature [2]. Indeed, power to detect interactions is always lower than for main effects, and in high-dimensional settings ($p >> n$), this lack of power to detect interactions is exacerbated, since the number of possible interactions could be enormous and their effects may be non-linear. Hence, analytic methods that may improve power are essential.

You need a non-genetic example - i.e. gene expression or proteomic or ...

Interactions may occur in numerous types and of varying complexities. In this paper, we consider one specific type of interaction models, where one (exposure) variable is involved in possibly non-linear interactions with a high-dimensional set of measures $\mathbf{X}$ leading to effects on a response variable, Y. We propose a multivariable penalization procedure for detecting non-linear interactions $\mathbf{X}$ and E.

1.1 Sparse additive interaction model

Let $Y = (Y_1, \ldots, Y_n) \in \mathbb{R}^n$ be a continuous or binary outcome variable, $X_E = (E_1, \ldots, E_n) \in \mathbb{R}^n$ a binary or continuous environment vector, and $\boldsymbol{X} = (X_1, \ldots, X_p) \in \mathbb{R}^{n \times p}$ a matrix of predictors, possibly high-dimensional. Furthermore let $f_j : \mathbb{R} \to \mathbb{R}$ be a smoothing method for variable X_j by a projection on to a set of basis functions:

$$f_j(X_j) = \sum_{\ell=1}^{m_j} \psi_{j\ell}(X_j)\beta_{j\ell} \tag{1}$$

Here, the $\{\psi_{j\ell}\}_1^{m_j}$ are a family of basis functions in X_j [3]. Let $\boldsymbol{\Psi}_j$ be the $n \times m_j$ matrix of evaluations of the $\psi_{j\ell}$ and $\boldsymbol{\theta}_j = (\beta_{j1}, \ldots, \beta_{jm_j}) \in \mathbb{R}^{m_j}$ for $j = 1, \ldots, p$, i.e., $\boldsymbol{\theta}_j$ is a m_j-dimensional column vector of basis coefficients for the jth main effect. In this article we

consider an additive interaction regression model of the form

$$g(\boldsymbol{\mu}) = \beta_0 \cdot \mathbf{1} + \sum_{j=1}^{p} \boldsymbol{\Psi}_j \boldsymbol{\theta}_j + \beta_E X_E + \sum_{j=1}^{p} (X_E \circ \boldsymbol{\Psi}_j) \boldsymbol{\tau}_j \tag{2}$$

where $g(\cdot)$ is a known link function, $\boldsymbol{\mu} = \mathsf{E}[Y|\boldsymbol{\Psi}, X_E]$, β_0 is the intercept, β_E is the coefficient for the environment variable, $\boldsymbol{\tau}_j = (\tau_{j1}, \ldots, \tau_{jm_j}) \in \mathbb{R}^{m_j}$ are the basis coefficients for the jth interaction term, and $(X_E \circ \boldsymbol{\Psi}_j)$ is the $n \times m_j$ matrix formed by the component-wise multiplication of the column vector X_E by each column of $\boldsymbol{\Psi}_j$. For a continuous response, we use the squared-error loss:

$$\mathcal{L}(\boldsymbol{\Theta}|\mathbf{D}) = \frac{1}{2n} \left\| Y - \beta_0 \cdot \mathbf{1} - \sum_{j=1}^{p} \boldsymbol{\Psi}_j \boldsymbol{\theta}_j - \beta_E X_E - \sum_{j=1}^{p} (X_E \circ \boldsymbol{\Psi}_j) \boldsymbol{\tau}_j \right\|_2^2 \tag{3}$$

and for binary response $Y_i \in \{-1, +1\}$ we use the logistic loss:

$$\mathcal{L}(\boldsymbol{\Theta}|\mathbf{D}) = \frac{1}{n} \sum_i \log \left(1 + \exp \left\{ -Y_i \left(\beta_0 \cdot \mathbf{1} - \sum_{j=1}^{p} \boldsymbol{\Psi}_j \boldsymbol{\theta}_j - \beta_E X_E - \sum_{j=1}^{p} (X_E \circ \boldsymbol{\Psi}_j) \boldsymbol{\tau}_j \right) \right\} \right) \tag{4}$$

where $\boldsymbol{\Theta} \coloneqq (\beta_0, \beta_E, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p, \boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_p)$ and $\mathbf{D} \coloneqq (Y, \boldsymbol{\Psi}, X_E)$ is the working data.

Due to the large number of parameters to estimate with respect to the number of observations, one commonly-used approach is to shrink the regression coefficients by placing a constraint on the values of $(\beta_E, \boldsymbol{\theta}_j, \boldsymbol{\tau}_j)$. Certain constraints have the added benefit of producing a sparse model in the sense that many of the coefficients will be set exactly to 0. This reduced predictor set can lead to a more interpretable model with smaller prediction variance, albeit at the cost of having biased parameter estimates. In light of these goals, we consider the following objective function:

$$\arg\min_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}|\mathbf{D}) + \lambda(1-\alpha) \left(w_E |\beta_E| + \sum_{j=1}^{p} w_j \|\boldsymbol{\theta}_j\|_2 \right) + \lambda\alpha \sum_{j=1}^{p} w_{jE} \|\boldsymbol{\tau}_j\|_2 \tag{5}$$

where $\|\boldsymbol{\theta}_j\|_2 = \sqrt{\sum_{k=1}^{m_j} \beta_{jk}^2}$, $\|\boldsymbol{\tau}_j\|_2 = \sqrt{\sum_{k=1}^{m_j} \tau_{jk}^2}$, $\lambda > 0$ and $\alpha \in (0, 1)$ are tuning parameters, w_E, w_j, w_{jE} are adaptive weights for $j = 1, \ldots, p$. These weights serve as a way of allowing parameters to be penalized differently.

An issue with (5) is that since no constraint is placed on the structure of the model, it is possible that an estimated interaction term is nonzero while the corresponding main effects are zero. While there may be certain situations where this is plausible, statisticians have generally argued that interactions should only be included if the corresponding main effects are also in the model [4]. This is known as the strong heredity principle [5]. Indeed, large main effects are more likely to lead to detectable interactions [6]. In the next section we discuss how a simple reparametrization of the model (5) can lead to this desirable property.

1.2 Strong and weak heredity

The strong heredity principle states that the interaction term can only have a non-zero estimate if its corresponding main effects are estimated to be non-zero. The weak heredity principle allows for a non-zero interaction estimate as long as one of the corresponding main effects are estimated to be non-zero [5]. In the context of penalized regression methods, these principles can be formulated as structured sparsity [7] problems. Several authors have proposed to modify the type of penalty in order to achieve the heredity principle [8, 9**? ?**]. We take an alternative approach. Following Choi et al. [10], we introduce a parameter $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p) \in \mathbb{R}^p$ and reparametrize the coefficients for the interaction terms $\boldsymbol{\tau}_j$ in (2) as a function of γ_j and the main effect parameters $\boldsymbol{\theta}_j$ and β_E. This reparametrization for both strong and weak heredity is summarized in Table 1.

To perform variable selection in this new parametrization, we penalize $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$

Table 1: Reparametrization for strong and weak heredity principle for `sail` model

Type	Feature	Reparametrization
Strong heredity	$\hat{\boldsymbol{\tau}}_j \neq 0 \Rightarrow \hat{\boldsymbol{\theta}}_j \neq 0$ and $\hat{\beta}_E \neq 0$	$\boldsymbol{\tau}_j = \gamma_j \beta_E \boldsymbol{\theta}_j$
Weak heredity	$\hat{\boldsymbol{\tau}}_j \neq 0 \Rightarrow \hat{\boldsymbol{\theta}}_j \neq 0$ or $\hat{\beta}_E \neq 0$	$\boldsymbol{\tau}_j = \gamma_j(\beta_E \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j)$

instead of penalizing $\boldsymbol{\tau}$ as in (5), leading to the following objective function:

$$\arg\min_{\boldsymbol{\Theta}} \mathcal{L}(\boldsymbol{\Theta}|\mathbf{D}) + \lambda(1-\alpha)\left(w_E\beta_E + \sum_{j=1}^{p} w_j \|\boldsymbol{\theta}_j\|_2\right) + \lambda\alpha \sum_{j=1}^{p} w_{jE}|\gamma_j| \tag{6}$$

This penalty allows for the possibility of excluding the interaction term from the model even if the corresponding main effects are non-zero.

1.3 Toy example

We begin with a toy example to better illustrate our method. With a sample size of $n = 100$, we sample $p = 20$ covariates $X_1, \ldots X_p$ independently from a $N(0, 1)$ truncated to the interval [0,1]. We generated data from a model which follows the strong heredity principle, but where only one covariates, X_2, is involved in the interaction with E:

$$Y = f_1(X_1) + f_2(X_2) + 1.75E + 1.5E \cdot f_2(X_2) + \varepsilon \tag{7}$$

Function $f_1(\cdot)$ is assumed to be linear, whereas function $f_2(\cdot)$ is non-linear: $f_1(x) = -3x$, $f_2(x) = 2(2x-1)^3$. The error term ε is generated from a normal distribution with variance chosen such that the signal-to-noise ratio (SNR) is 2. We generated a single simulated dataset and used the strong heredity `sail` method with cubic B-splines to estimate the functional forms. 10-fold CV was used to choose the optimal value of λ. We used $\alpha = 0.5$ and default values were used for all other arguments. We plot the solution path for both main effects

and interactions in Figure 1 and we color the lines corresponding to the selected model. We see that our method is able to correctly identify the true model. We can also visually see the effect of the penalty and strong heredity principle working in tandem, i.e., the interaction term $E \cdot f_2(X_2)$ (orange lines in the bottom panel) can only be nonzero if the main effects E and $f_2(X_2)$ (black and orange lines respectively in the top panel) are nonzero, while nonzero main effects doesn't necessarily imply a nonzero interaction.

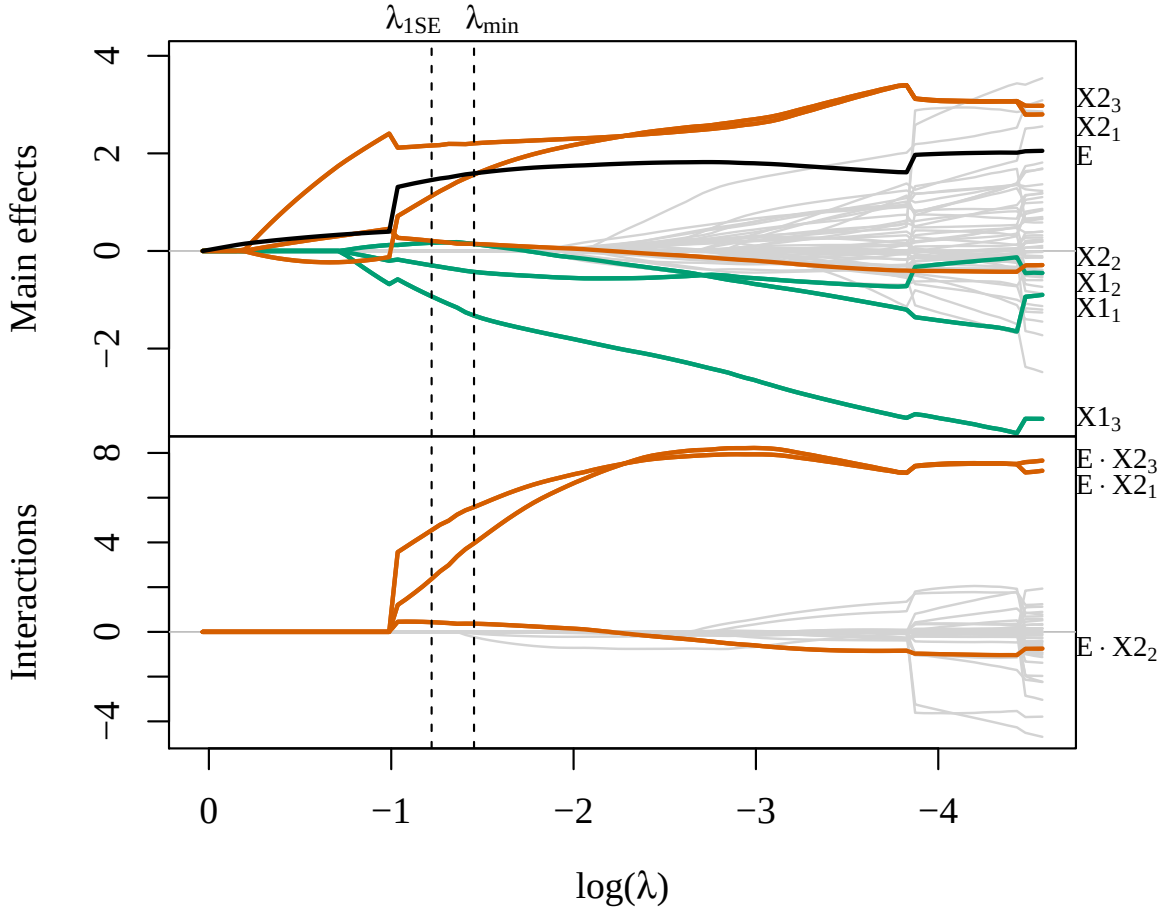


Figure 1: Toy example solution path

In Figure 2, we plot the true and estimated component functions $\hat{f}_1(X_1)$ and $E \cdot \hat{f}_2(X_2)$, and their estimates with `sail`. We are able to capture the shape of the correct functional form, but our means are not well aligned with the data. Lack-of-fit for $f_1(X_1)$ can be partially

explained by acknowledging that `sail` is trying to fit a cubic spline to a linear function. Nevertheless, this example demonstrates that it can still identify linear associations with reasonable sensitivity and specificity.

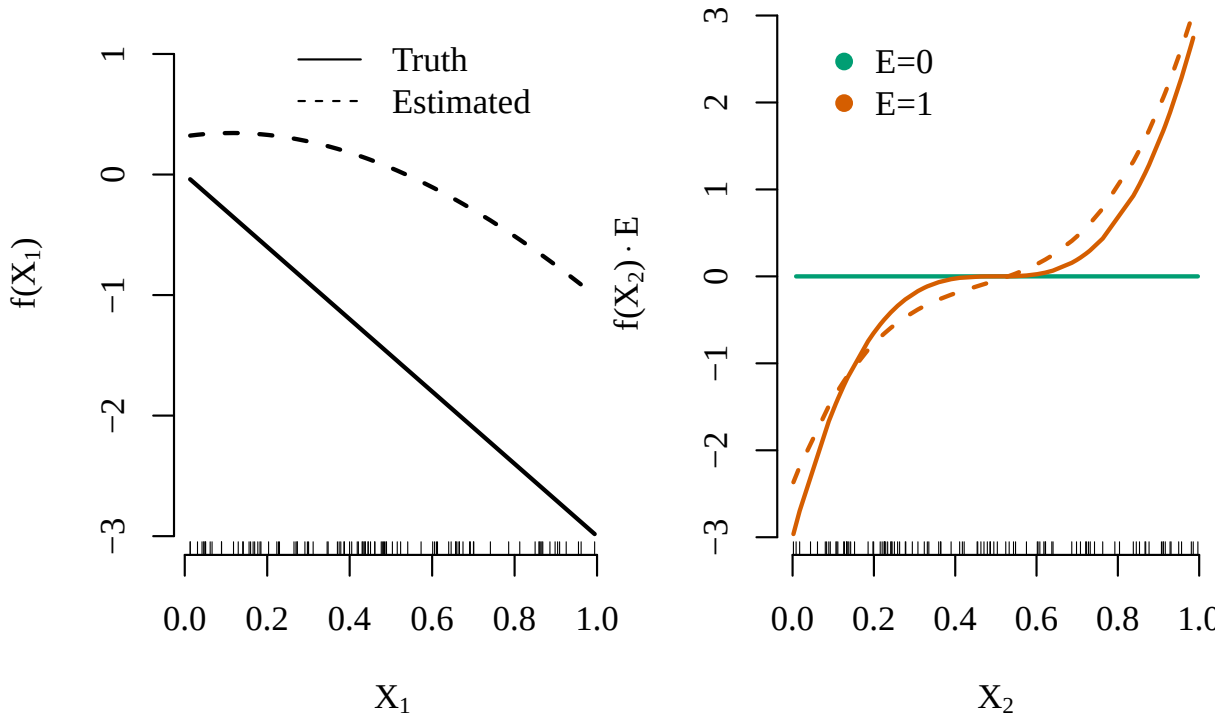


Figure 2: Estimated smooth functions by `sail` method based on λ_{min}.

1.4 Related Work

Methods for interaction selection can be broken down into two categories: linear and non-linear interaction effects. Many of the linear effect methods consider all pairwise interactions in $\boldsymbol{X}$ [9, 10, 11, 12] which can be computationally prohibitive when p is large. This is evidenced by the relatively small number of variables used in their simulations and real data analysis. More recent proposals allow the user to restrict the search space to interaction candidates [13, 14] which is useful when the researcher wants to impose some prior information on the model. Two-stage procedures, where interactions candidates are considered from

an original screen of main effects, have shown good performance when p is large [15, 16] in the linear setting. There are many fewer methods avaliable for non-linear interactions. For example, Radchenko and James (2010) [8] proposed a model of the form

$$Y = \beta_0 + \sum_{j=1}^{p} f_j(X_j) + \sum_{j>k} f_{jk}(X_j, X_k) + \varepsilon$$

where $f(\cdot)$ are smooth component functions. This method is computationally expensive however, as it involves a complex penalty function and considers all pairwise interactions. Furthermore, it's effectiveness in both simulations and real-data applications is unknown as there is no software implementation.

While working on this paper, we were made aware of the recently proposed pliable lasso [17] which considers the interactions between $\boldsymbol{X}_{n\times p}$ and another matrix $\mathbf{Z}_{n\times K}$ and takes the form

$$Y = \beta_0 + \sum_{j=1}^{p} \beta_j X_j + \sum_{j=1}^{K} \theta_j Z_j + \sum_{j=1}^{p} (X_j \circ \mathbf{Z})\boldsymbol{\alpha}_j + \varepsilon$$

where α_j is a K-dimensional vector. Our proposal is most closely related to this method with $\mathbf{Z}$ being a single column matrix; the key difference being the non-linearity of the predictor variables. As pointed out by the authors of the pliable lasso, these methods can be seen as a varying coefficient model, i.e., the effect of the predictors vary as a function of the exposure variable E.

The main contributions of this paper are fourfold. First, we develop a model for non-linear interactions with a key exposure variable following either the weak or strong heredity principle that is computationally efficient and scales to the high-dimensional setting ($n << p$). Second, through simulation studies, we show improved performance over existing methods that only consider linear interactions or additive main effects. Third, we show that our method possesses the oracle property [18], i.e., it performs as well as if the true model were known in advance. Fourth, all of our algorithms are implemented in the `sail` R package

hosted on GitHub with extensive documentation (http://sahirbhatnagar.com/sail/). In particular, our implementation also allows for linear interaction models, user-defined basis expansions, a cross-validation procedure for selecting the optimal tuning parameter, and differential shrinkage parameters to apply the adaptive lasso [19] idea.

The rest of the paper is organized as follows. Sections 2 and 3 describe our optimization procedure and some details about the algorithm used to fit the `sail` model for the least squares and logistic case, respectively. In Section 4, we compare the performance of our proposed approach and demonstrate the scenarios where it can be advantageous to use over existing methods through simulation studies. Section 5 contains some real data examples and Section 6 discusses some limitations and future directions.

2 Algorithm and Computational Details

In this section we describe a blockwise coordinate descent algorithm for fitting both the least-squares and logistic version of the `sail` model in (6). We fix the value for α and minimize the objective function over a decreasing sequence of λ values ($\lambda_{max} > \cdots > \lambda_{min}$). We use the subgradient equations to determine the maximal value λ_{max} such that all estimates are zero. Due to the heredity principle, this reduces to finding the largest λ such that all main effects $(\beta_E, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p)$ are zero. Following Friedman et al. [20], we construct a λ-sequence of 100 values decreasing from λ_{max} to $0.001\lambda_{max}$ on the log scale, and use the warm start strategy where the solution for λ_ℓ is used as a starting value for $\lambda_{\ell+1}$.

2.1 Blockwise coordinate descent for least-squares loss

The strong heredity `sail` model with least-squares loss has the form

$$\hat{Y} = \beta_0 \cdot \mathbf{1} + \sum_{j=1}^{p} \mathbf{\Psi}_j \boldsymbol{\theta}_j + \beta_E X_E + \sum_{j=1}^{p} \gamma_j \beta_E (X_E \circ \mathbf{\Psi}_j)\boldsymbol{\theta}_j \tag{8}$$

and the objective function is given by

$$Q(\mathbf{\Theta}) = \frac{1}{2n}\left\|Y - \hat{Y}\right\|_2^2 + \lambda(1-\alpha)\left(w_E|\beta_E| + \sum_{j=1}^{p} w_j \|\boldsymbol{\theta}_j\|_2\right) + \lambda\alpha \sum_{j=1}^{p} w_{jE}|\gamma_j| \tag{9}$$

Solving (9) in a blockwise manner allows us to leverage computationally fast algorithms for ℓ_1 and ℓ_2 norm penalized regression. The objective function simplifies to a modified lasso problem when holding all θ_j fixed, and a modified group lasso problem when holding β_E and all γ_j fixed.

Denote the n-dimensional residual column vector $R = Y - \hat{Y}$. The subgradient equations are given by

$$\frac{\partial Q}{\partial \beta_0} = \frac{1}{n}\left(Y - \beta_0 \cdot \mathbf{1} - \sum_{j=1}^{p} \mathbf{\Psi}_j \boldsymbol{\theta}_j - \beta_E X_E - \sum_{j=1}^{p} \gamma_j \beta_E (X_E \circ \mathbf{\Psi}_j)\boldsymbol{\theta}_j\right)^\top \mathbf{1} = 0 \tag{10}$$

$$\frac{\partial Q}{\partial \beta_E} = -\frac{1}{n}\left(X_E + \sum_{j=1}^{p} \gamma_j (X_E \circ \mathbf{\Psi}_j)\boldsymbol{\theta}_j\right)^\top R + \lambda(1-\alpha) w_E s_1 = 0 \tag{11}$$

$$\frac{\partial Q}{\partial \boldsymbol{\theta}_j} = -\frac{1}{n}\left(\mathbf{\Psi}_j + \gamma_j \beta_E (X_E \circ \mathbf{\Psi}_j)\right)^\top R + \lambda(1-\alpha) w_j s_2 = \mathbf{0} \tag{12}$$

$$\frac{\partial Q}{\partial \gamma_j} = -\frac{1}{n}\left(\beta_E (X_E \circ \mathbf{\Psi}_j)\boldsymbol{\theta}_j\right)^\top R + \lambda\alpha w_{jE} s_3 = 0 \tag{13}$$

where s_1 is in the subgradient of the ℓ_1 norm:

$$s_1 \in \begin{cases} \operatorname{sign}(\beta_E) & \text{if } \beta_E \neq 0 \\ [-1, 1] & \text{if } \beta_E = 0, \end{cases}$$

s_2 is in the subgradient of the ℓ_2 norm:

$$s_2 \in \begin{cases} \dfrac{\boldsymbol{\theta}_j}{\|\boldsymbol{\theta}_j\|_2} & \text{if } \boldsymbol{\theta}_j \neq \mathbf{0} \\ u \in \mathbb{R}^{m_j} : \|u\|_2 \leq 1 & \text{if } \boldsymbol{\theta}_j = \mathbf{0}, \end{cases}$$

and s_3 is in the subgradient of the ℓ_1 norm:

$$s_3 \in \begin{cases} \text{sign}\,(\gamma_j) & \text{if } \gamma_j \neq 0 \\ [-1, 1] & \text{if } \gamma_j = 0. \end{cases}$$

Define the partial residuals, without the jth predictor for $j = 1, \ldots, p$, as

$$R_{(-j)} = Y - \beta_0 \cdot \mathbf{1} - \sum_{\ell \neq j} \boldsymbol{\Psi}_\ell \boldsymbol{\theta}_\ell - \beta_E X_E - \sum_{\ell \neq j} \gamma_\ell \beta_E (X_E \circ \boldsymbol{\Psi}_\ell) \boldsymbol{\theta}_\ell$$

the partial residual without X_E as

$$R_{(-E)} = Y - \beta_0 \cdot \mathbf{1} - \sum_{j=1}^{p} \boldsymbol{\Psi}_j \boldsymbol{\theta}_j$$

and the partial residual without the jth interaction for $j = 1, \ldots, p$, as

$$R_{(-jE)} = Y - \beta_0 \cdot \mathbf{1} - \sum_{j=1}^{p} \boldsymbol{\Psi}_j \boldsymbol{\theta}_j - \beta_E X_E - \sum_{\ell \neq j} \gamma_\ell \beta_E (X_E \circ \boldsymbol{\Psi}_\ell) \boldsymbol{\theta}_\ell$$

From the subgradient equations (10)–(13) we see that

$$\hat{\beta}_0 = \left(Y - \sum_{j=1}^{p} \boldsymbol{\Psi}_j \hat{\boldsymbol{\theta}}_j - \hat{\beta}_E X_E - \sum_{j=1}^{p} \hat{\gamma}_j \hat{\beta}_E (X_E \circ \boldsymbol{\Psi}_j) \hat{\boldsymbol{\theta}}_j \right)^\top \mathbf{1} \tag{14}$$

$$\hat{\beta}_E = S\left(\frac{1}{n \cdot w_E} \left(X_E + \sum_{j=1}^{p} \hat{\gamma}_j (X_E \circ \boldsymbol{\Psi}_j) \hat{\boldsymbol{\theta}}_j \right)^\top R_{(-E)}, \lambda(1-\alpha) \right) \tag{15}$$

$$\lambda(1-\alpha) w_j \frac{\boldsymbol{\theta}_j}{\|\boldsymbol{\theta}_j\|_2} = \frac{1}{n} \left(\boldsymbol{\Psi}_j + \gamma_j \beta_E (X_E \circ \boldsymbol{\Psi}_j) \right)^\top R_{(-j)} \tag{16}$$

$$\hat{\gamma}_j = S\left(\frac{1}{n \cdot w_{jE}} \left(\beta_E (X_E \circ \boldsymbol{\Psi}_j) \boldsymbol{\theta}_j \right)^\top R_{(-jE)}, \lambda\alpha \right) \tag{17}$$

where $S(x,t) = \text{sign}(x)(|x| - t)$ is the soft-thresholding operator. We see from (14) and (15) that there are closed form solutions for the intercept and β_E. From (17), each γ_j also has a closed form solution and can be solved efficiently for $j = 1, \ldots, p$ using the coordinate descent procedure implemented in the `glmnet` package [20]. While there is no closed form solution for β_j, we can use a quadratic majorization technique implemented in the `gglasso` package [21] to solve (16). From these estimates, we can compute the interaction effects using the reparametrizations presented in Table 1, e.g., $\hat{\boldsymbol{\tau}}_j = \hat{\gamma}_j \hat{\beta}_E \hat{\boldsymbol{\theta}}_j$, $j = 1, \ldots, p$ for the strong heredity `sail` model. We provide an overview of the computations in Algorithm 1. A more detailed version of this algorithm is given in Section A.1 of the Appendix.

Algorithm 1 Blockwise Coordinate Descent for Least-Squares `sail` with Strong Heredity.

For a decreasing sequence $\lambda = \lambda_{max}, \ldots, \lambda_{min}$ and fixed α:

1. Initialize $\beta_0^{(0)}, \beta_E^{(0)}, \boldsymbol{\theta}_j^{(0)}, \gamma_j^{(0)}$ for $j = 1, \ldots, p$ and set iteration counter $k \leftarrow 0$.
2. Repeat the following until convergence:
   (a) update $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$
      i. Compute the pseudo design: $\widetilde{X}_j \leftarrow \beta_E^{(k)}(X_E \circ \boldsymbol{\Psi}_j)\boldsymbol{\theta}_j^{(k)} \qquad$ for $j = 1, \ldots, p$
      ii. Compute the pseudo response $\widetilde{Y}$ by removing the contribution of every term not involving $\boldsymbol{\gamma}$ from Y
      iii. Solve:

$$\boldsymbol{\gamma}^{(k)(new)} \leftarrow \arg\min_{\boldsymbol{\gamma}} \frac{1}{2n}\left\|\widetilde{Y} - \sum_j \gamma_j \widetilde{X}_j\right\|_2^2 + \lambda\alpha \sum_j w_{jE}|\gamma_j| \tag{18}$$

      iv. Set $\boldsymbol{\gamma}^{(k)} = \boldsymbol{\gamma}^{(k)(new)}$
   (b) update $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p)$
      — for $j = 1, \ldots, p$
      i. Compute the pseudo design: $\widetilde{X}_j \leftarrow \boldsymbol{\Psi}_j + \gamma_j^{(k)}\beta_E^{(k)}(X_E \circ \boldsymbol{\Psi}_j)$
      ii. Compute the pseudo response $(\widetilde{Y})$ by removing the contribution of every term not involving $\boldsymbol{\theta}_j$ from Y
      iii. Solve:

$$\boldsymbol{\theta}_j^{(k)(new)} \leftarrow \arg\min_{\boldsymbol{\theta}_j} \frac{1}{2n}\left\|\widetilde{Y} - \widetilde{X}_j\boldsymbol{\theta}_j\right\|_2^2 + \lambda(1-\alpha)w_j\|\theta_j\|_2 \tag{19}$$

      iv. Set $\boldsymbol{\theta}_j^{(k)} \leftarrow \boldsymbol{\theta}_j^{(k)(new)}$
   (c) update β_E
      i. Compute the pseudo design: $\widetilde{X}_E \leftarrow X_E + \sum_j \gamma_j^{(k)}\widetilde{\boldsymbol{\Psi}}_j\boldsymbol{\theta}_j^{(k)}$
      ii. Compute the pseudo response $(\widetilde{Y})$ by removing the contribution of every term not involving β_E from Y
      iii. Soft-threshold update ($S(x,t) = \text{sign}(x)(|x| - t)_+$):

$$\beta_E^{(k)(new)} \leftarrow S\left(\frac{1}{n \cdot w_E}\widetilde{X}_E^\top\widetilde{Y}, \lambda(1-\alpha)\right) \tag{20}$$

      iv. Set $\beta_E^{(k+1)} \leftarrow \beta_E^{(k)(new)}$, $k \leftarrow k+1$

2.2 Lambda max

The subgradient equations (11)–(13) can be used to determine the largest value of λ such that all coefficients are 0. From the subgradient Equation (11), we see that $\beta_E = 0$ is a solution if

$$\frac{1}{w_E}\left|\frac{1}{n}\left(X_E + \sum_{j=1}^{p}\gamma_j(X_E \circ \boldsymbol{\Psi}_j)\boldsymbol{\theta}_j\right)^\top R_{(-E)}\right| \leq \lambda(1-\alpha) \tag{21}$$

From the subgradient Equation (12), we see that $\boldsymbol{\theta}_j = \mathbf{0}$ is a solution if

$$\frac{1}{w_j}\left\|\frac{1}{n}\left(\boldsymbol{\Psi}_j + \gamma_j\beta_E(X_E \circ \boldsymbol{\Psi}_j)\right)^\top R_{(-j)}\right\|_2 \leq \lambda(1-\alpha) \tag{22}$$

From the subgradient Equation (13), we see that $\gamma_j = 0$ is a solution if

$$\frac{1}{w_{jE}}\left|\frac{1}{n}\left(\beta_E(X_E \circ \boldsymbol{\Psi}_j)\boldsymbol{\theta}_j\right)^\top R_{(-jE)}\right| \leq \lambda\alpha \tag{23}$$

Due to the strong heredity property, the parameter vector $(\beta_E, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p, \gamma_1, \ldots, \gamma_p)$ will be entirely equal to $\mathbf{0}$ if $(\beta_E, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p) = \mathbf{0}$. Therefore, the smallest value of λ for which the entire parameter vector (excluding the intercept) is $\mathbf{0}$ is:

$$\lambda_{max} = \frac{1}{n(1-\alpha)}\max\left\{\frac{1}{w_E}\left(X_E + \sum_{j=1}^{p}\gamma_j(X_E \circ \boldsymbol{\Psi}_j)\boldsymbol{\theta}_j\right)^\top R_{(-E)}, \max_j \frac{1}{w_j}\left\|\left(\boldsymbol{\Psi}_j + \gamma_j\beta_E(X_E \circ \boldsymbol{\Psi}_j)\right)^\top R_{(-j)}\right\|_2\right\} \tag{24}$$

which reduces to

$$\lambda_{max} = \frac{1}{n(1-\alpha)}\max\left\{\frac{1}{w_E}\left(X_E\right)^\top R_{(-E)}, \max_j \frac{1}{w_j}\left\|\left(\boldsymbol{\Psi}_j\right)^\top R_{(-j)}\right\|_2\right\}$$

2.3 Weak Heredity

Our method can be easily adopted to enforce the weak heredity property:

$$\hat{\alpha}_{jE} \neq 0 \quad \Rightarrow \quad \hat{\beta}_j \neq 0 \quad \text{or} \quad \hat{\beta}_E \neq 0$$

That is, an interaction term can only be present if at least one of it's corresponding main effects is nonzero. To do so, we reparametrize the coefficients for the interaction terms in (2) as $\boldsymbol{\alpha}_j = \gamma_j(\beta_E \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j)$, where $\mathbf{1}_{m_j}$ is a vector of ones with dimension m_j (i.e. the length of $\boldsymbol{\theta}_j$). We defer the algorithm details for fitting the `sail` model with weak heredity in Section A.3 of the Appendix, as it is very similar to Algorithm 1 for the strong heredity `sail` model.

2.4 Adaptive `sail`

The weights for the environment variable, main effects and interactions are given by w_E, w_j and w_{jE} respectively. These weights serve as a way of allowing a different penalty to be applied to each variable. In particular, any variable with a weight of zero is not penalized at all. This feature can be applied mainly for two reasons:

1. Prior knowledge about the importance of certain variables is known. Larger weights will penalize the variable more, while smaller weights will penalize the variable less
2. Allows users to apply the adaptive `sail`, similar to the adaptive lasso [19]

We describe the adaptive `sail` in Algorithm 2. This is a general procedure that can be applied to the weak and strong heredity settings, as well as both least squares and logistic loss functions. We provide this capability in the `sail` package using the `penalty.factor` argument and provide an example in Section C.6 of the Appendix.

Algorithm 2 Adaptive `sail` algorithm

1. For a decreasing sequence $\lambda = \lambda_{max}, \ldots, \lambda_{min}$ and fixed α run the `sail` algorithm
2. Use cross-validation or a data splitting procedure to determine the optimal value for the tuning parameter: $\lambda^{[opt]} \in \{\lambda_{max}, \ldots, \lambda_{min}\}$
3. Let $\widehat{\beta_E}^{[opt]}, \widehat{\boldsymbol{\theta}}_j^{[opt]}$ and $\widehat{\boldsymbol{\tau}}_j^{[opt]}$ for $j = 1, \ldots, p$ be the coefficient estimates corresponding to the model at $\lambda^{[opt]}$
4. Set the weights to be
$$w_E = \left(\left|\widehat{\beta_E}^{[opt]}\right|\right)^{-1}, \; w_j = \left(\|\widehat{\boldsymbol{\theta}}_j^{[opt]}\|_2\right)^{-1}, \; w_{jE} = \left(\|\widehat{\boldsymbol{\tau}}_j^{[opt]}\|_2\right)^{-1} \text{ for } j = 1, \ldots, p$$
5. Run the `sail` algorithm with the weights defined in step 4), and use cross-validation or a data splitting procedure to choose the optimal value of λ

2.5 Flexible design matrix

The definition of the basis expansion functions in (1) is very flexible in the sense that our algorithms are independent of this choice. As a result, the user can apply any basis expansion they want. In the extreme case, we can apply the identity map, i.e., $f_j(X_j) = X_j$ which leads to a linear interaction model (referred to as `linear sail`). When little information is known a priori about the relationship between the predictors and the response, by default, we choose to apply the same basis expansion to all columns of $\boldsymbol{X}$. This is a reasonable approach when all the variables are continuous. However, there are often instances when our data contains a combination of categorical and continuous variables. In these situations it may be sub-optimal to apply a basis expansion to the categorical variables. Owing to the flexible nature of our algorithm, we can handle this scenario in our implementation by allowing a user-defined design matrix. The only extra information needed is the group membership of each column in the design matrix. We provide such an example in the `sail` package showcase in Section C.7 of the Appendix.

3 Simulation Study

In this section, we use simulated data to understand the performance of `sail` in different scenarios.

3.1 Comparator Methods

Since there is no software that directly addresses our problem, we selected comparator methods based on the following criteria: 1) is a penalized regression method that can handle high-dimensional data ($n < p$) 2) considers linear, non-linear or interaction effects and 3) has a software implementation in `R`. The selected methods can be grouped into three categories:

1. Linear main effects: `lasso` [22], `adaptive lasso` [19]
2. Linear interactions: `lassoBT` [16], `GLinternet` [13]
3. Non-linear main effects: `HierBasis` [23], `SPAM` [24], `gamsel` [25]

For `GLinternet` we specified the `interactionCandidates` argument as to only consider interactions between the environment and all other X variables. For all other methods we supplied $(\boldsymbol{X}, X_E)$ as the data matrix, 100 for the number of tuning parameters to fit, and used the default values otherwise[1]. `lassoBT` considers all pairwise interactions as there is no way for the user to restrict the search space. `SPAM` applies the same basis expansion to every column of the data matrix; we chose 5 basis spline functions. `HierBasis` and `gamsel` selects whether a term in an additive model is nonzero, linear, or a non-linear spline up to a specified max degrees of freedom per variable.

We compare the above listed methods with our main proposal `sail`, as well as `adaptive sail` (Algorithm 2), `sail weak` which has the weak heredity property and `linear sail` as

[1] `R` code for each method available at https://github.com/sahirbhatnagar/sail/blob/master/my_sims/method_functions.R

described in Section 2.5. For each function f_j, we use a B-spline basis matrix with `degree=5` implemented in the `bs` function in `R` [26]. We center the environment variable and the basis functions before running the `sail` method.

3.2 Simulation Design

The covariates are simulated as follows. First, we generate $w_1, \ldots, w_p, u, v$ independently from a standard normal distribution truncated to the interval [0,1] for $i = 1, \ldots, n$. Then we set $x_j = (w_j + t \cdot u)/(1 + t)$ for $j = 1, \ldots, 4$ and $x_j = (w_j + t \cdot v)/(1 + t)$ for $j = 5, \ldots, p$, where the parameter t controls the amount of correlation among predictors. The first four variables are nonzero (i.e. active in the response), while the rest of the variables are zero (i.e. are noise variables). This leads to a compound symmetry correlation structure where $Corr(x_j, x_k) = t^2/(1 + t^2)$, for $1 \leq j \leq 4, 1 \leq k \leq 4$, and $Corr(x_j, x_k) = t^2/(1 + t^2)$, for $5 \leq j \leq p, 5 \leq k \leq p$, but the covariates of the nonzero and zero components are independent [27, 28]. We consider the case when $p = 1000$ and $t = 0$. The outcome Y is then generated following one of the models and assumptions described below.

We evaluate the performance of our method on three of its defining characteristics: 1) the strong heredity property, 2) non-linearity of predictor effects and 3) interactions.

1. **Hierarchy**

   (a) Truth obeys strong hierarchy. In this situation, the true model for Y contains main effect terms for all covariates involved in interactions.

$$Y = \sum_{j=1}^{4} f_j(X_j) + \beta_E \cdot X_E + X_E \cdot f_3(X_3) + X_E \cdot f_4(X_4) + \varepsilon$$

   (b) Truth obeys weak hierarchy. Here, in addition to the interaction, the E variable

has its own main effect but the covariates X_3 and X_4 do not.

$$Y = f_1(X_1) + f_2(X_2) + \beta_E \cdot X_E + X_E \cdot f_3(X_3) + X_E \cdot f_4(X_4) + \varepsilon$$

(c) Truth only has interactions. In this simulation, the covariates involved in interactions do not have main effects as well.

$$Y = X_E \cdot f_3(X_3) + X_E \cdot f_4(X_4) + \varepsilon$$

2. **Non-linearity**

— Truth is linear. `sail` is designed to model non-linearity; here we assess its performance if the true model is completely linear.

$$Y = 5X_1 + 3(X_2 + 1) + 4X_3 + 6(X_4 - 2) + \beta_E \cdot X_E + X_E \cdot 4X_3 + X_E \cdot 6(X_4 - 2) + \varepsilon$$

3. **Interactions**

— Truth only has main effects. `sail` is designed to capture interactions; here we assess its performance when there are non in the true model.

$$Y = \sum_{j=1}^{4} f_j(X_j) + \beta_E \cdot X_E + \varepsilon$$

The true component functions are the same as in [27, 28] and are given by $f_1(t) = 5t$, $f_2(t) = 3(2t-1)^2$, $f_3(t) = 4\sin(2\pi t)/(2 - \sin(2\pi t))$, $f_4(t) = 6(0.1\sin(2\pi t) + 0.2\cos(2\pi t) + 0.3\sin(2\pi t)^2 + 0.4\cos(2\pi t)^3 + 0.5\sin(2\pi t)^3)$. We set $\beta_E = 2$ and draw ε from a normal distribution with variance chosen such that the signal-to-noise ratio is 2. Using this setup, we generated 200 replications consisting of a training set of $n = 200$, a validation set of $n = 200$ and a test set of $n = 800$. The training set was used to fit the model and the

validation set was used to select the optimal tuning parameter corresponding to the minimum prediction mean squared error (MSE). Variable selection results including true positive rate, false positive rate and number of active variables (the number of variables with a non-zero coefficient estimate) were assessed on the training set and MSE was assessed on the test set.

3.3 Results

The test set MSE results for each of the five simulation scenarios are shown in Figure 3, while Figure 4 shows the mean true positive rate (TPR) vs. the mean false positive rate (FPR) ± 1 standard deviation (SD).

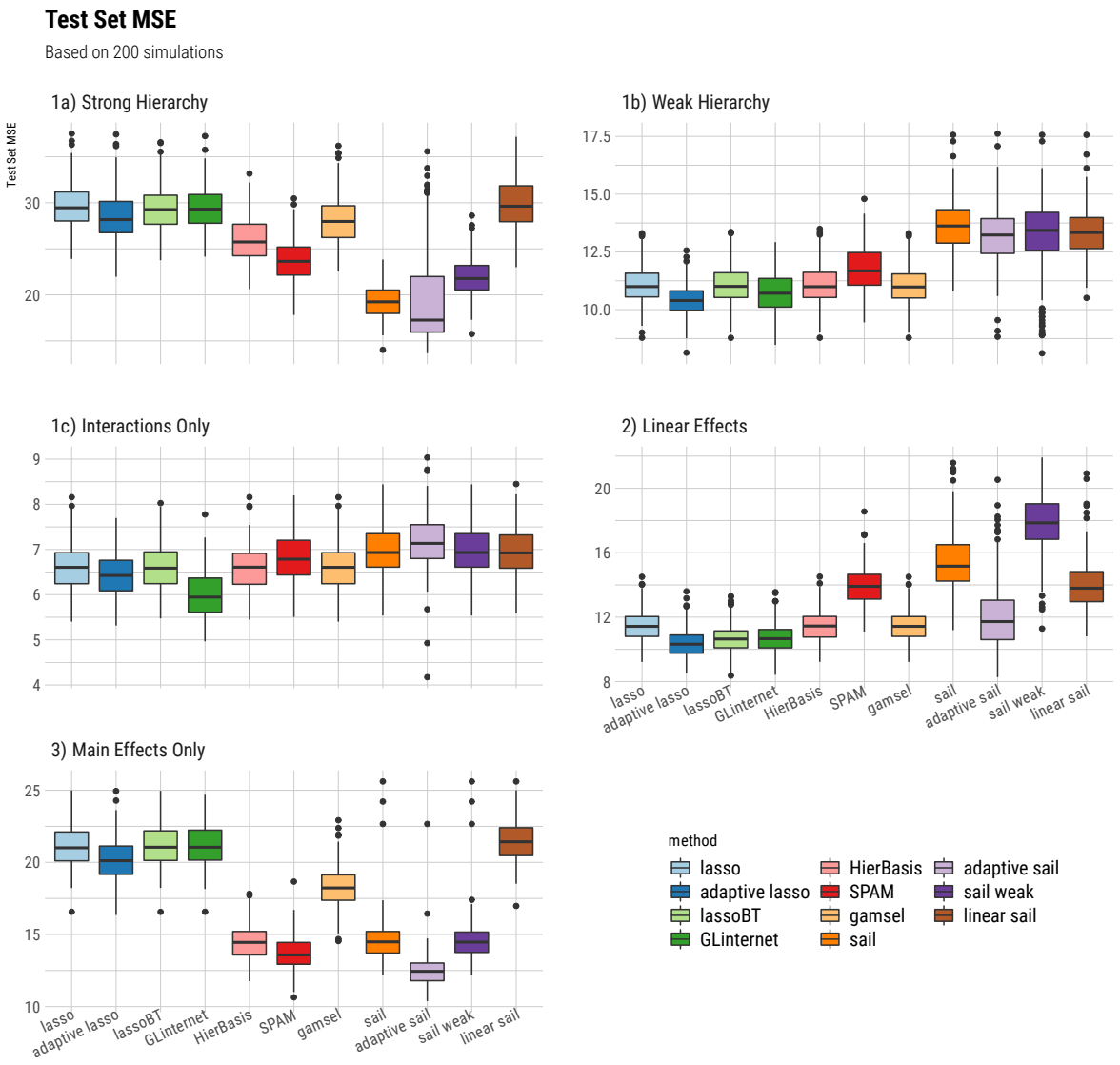


Figure 3: Boxplots of the test set mean squared error from 200 simulations for each of the five simulation scenarios.

We see that `sail` ,`adaptive sail` and `sail weak` have the best performance in terms of both MSE and yielding correct sparse models when the truth follows strong hierarchy (scenario 1a), as we would expect, since this is exactly the scenario that our method is trying to target.

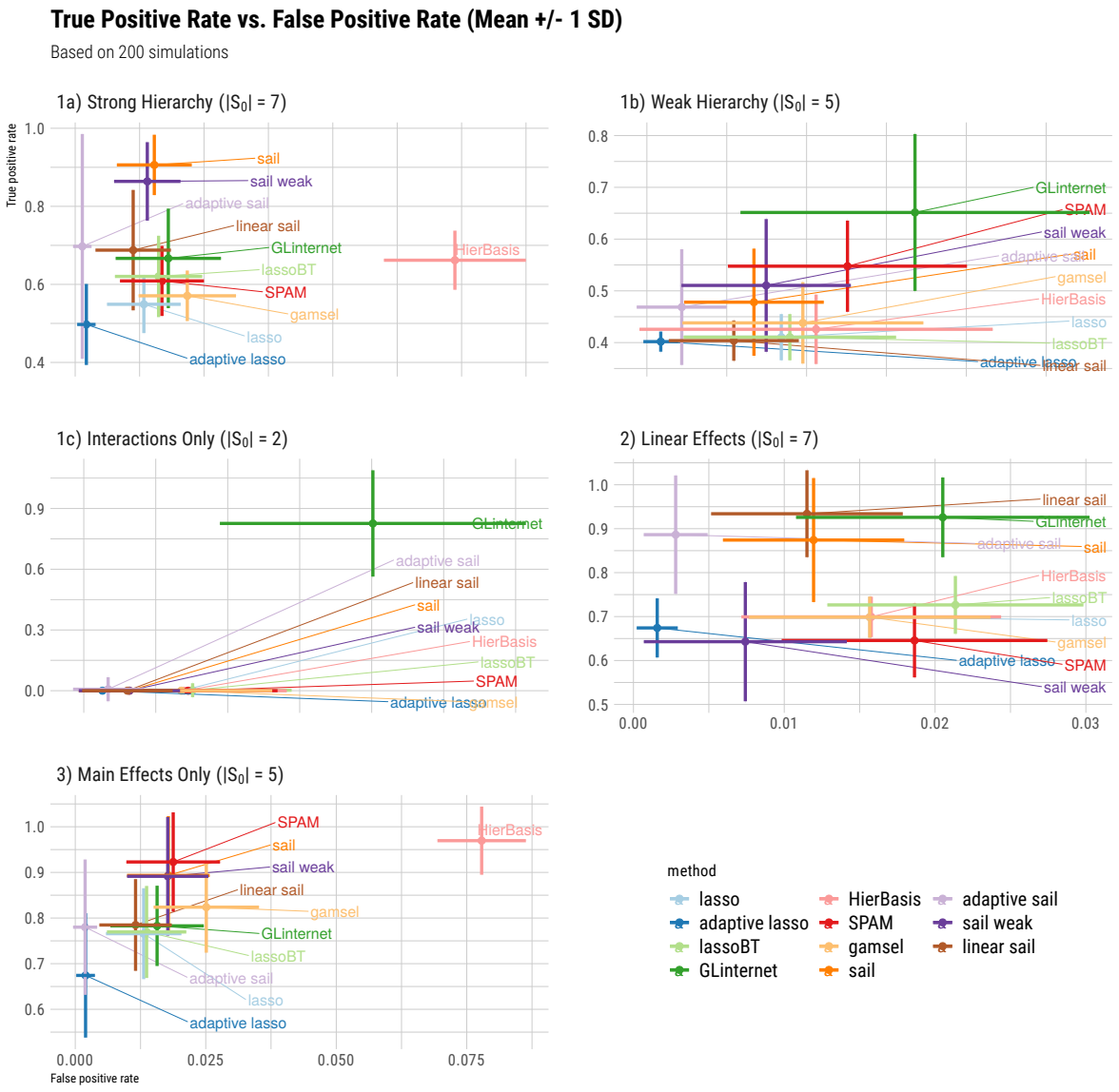


Figure 4: Means ± 1 standard deviation of true positive rate vs. false positive rate from 200 simulations for each of the five scenarios. $|S_0|$ is the number of truly associated variables.

Our method is also competitive when only main effects are present (scenario 3) and performs just as well as methods that only consider linear and non-linear main effects (`HierBasis`, `SPAM`), owing to the penalization applied to the interaction parameter. Due to the heredity property, our method is unable to capture any of the truly associated variables when only interactions are present (scenario 1c). However, the other methods also fail to capture any

signal, with the exception of `GLinternet` which has a high TPR and FPR. When only linear effects and interactions are present (scenario 2), we see that `linear sail` has a high TPR and low FPR as compared to the other linear interaction methods (`lassoBT` and `GLinternet`) though the test set MSE isn't as good. The `lasso` and `adaptive lasso` have good test set MSE performance but poor sensitivity. Additional results are available in Section B of the Appendix. Specifically, in Figure B.1 we plot the mean MSE against the mean number of active variables ± 1 standard deviation (SD). Figures B.2 and B.3 show the true positive and false positive rates, respectively. Figure B.4 shows the number of active variables.

We visually inspected whether our method could correctly capture the shape of the association between the predictors and the response for both main and interaction effects. To do so, we plotted the true and predicted curves for scenario 1a) only. Figure 5 shows each of the four main effects with the estimated curves from each of the 200 simulations along with the true curve. We can see the effect of the penalty on the parameters, i.e., decreasing prediction variance at the cost of increased bias. This is particularly well illustrated in the bottom right panel where `sail` smooths out the very wiggly component function $f_4(x)$.

To visualize the estimated interaction effects, we ordered the 200 simulation runs by the euclidean distance between the estimated and true regression functions. Following Radchenko et al. [8], we then identified the 25th, 50th, and 75th best simulations and plotted, in Figures 6 and 7, their interaction effects of X_E with $f_3(X_3)$ and $f_4(X_4)$, respectively. We see that `sail` does a good job at capturing the true interaction surface for $X_E \cdot f_3(X_3)$. Again, the smoothing and shrinkage effect is apparent when looking at the interaction surfaces for $X_E \cdot f_4(X_4)$

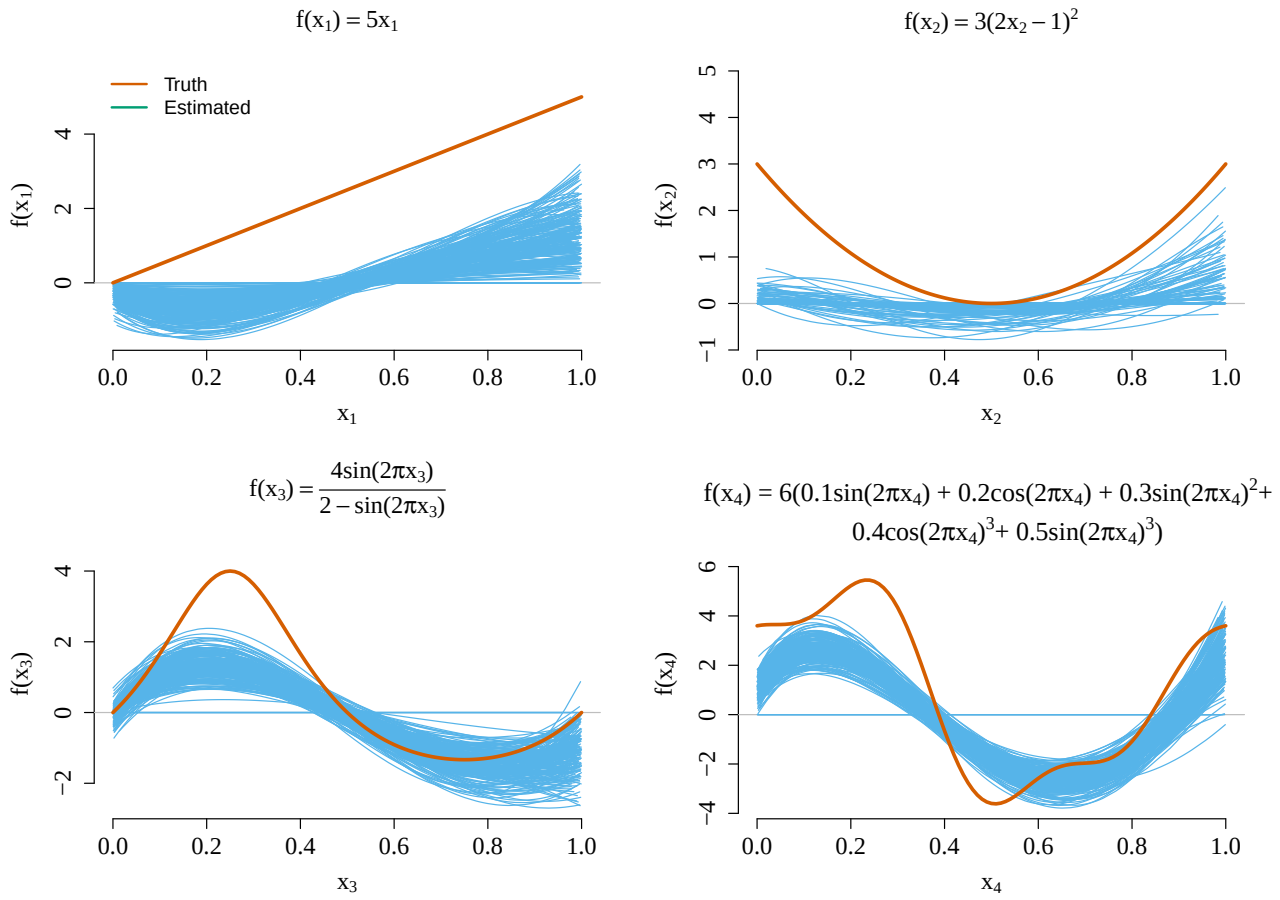


Figure 5: True and estimated main effect component functions for scenario 1a). The estimated curves represent the results from each one of the 200 simulations conducted.

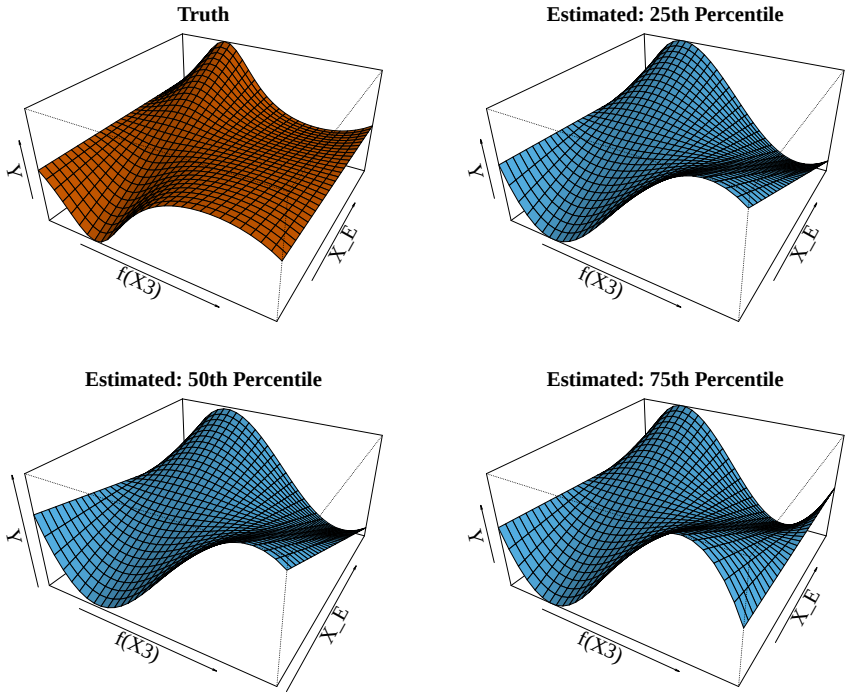


Figure 6: True and estimated interaction effects for $X_E \cdot f_3(X_3)$ in simulation scenario 1a).

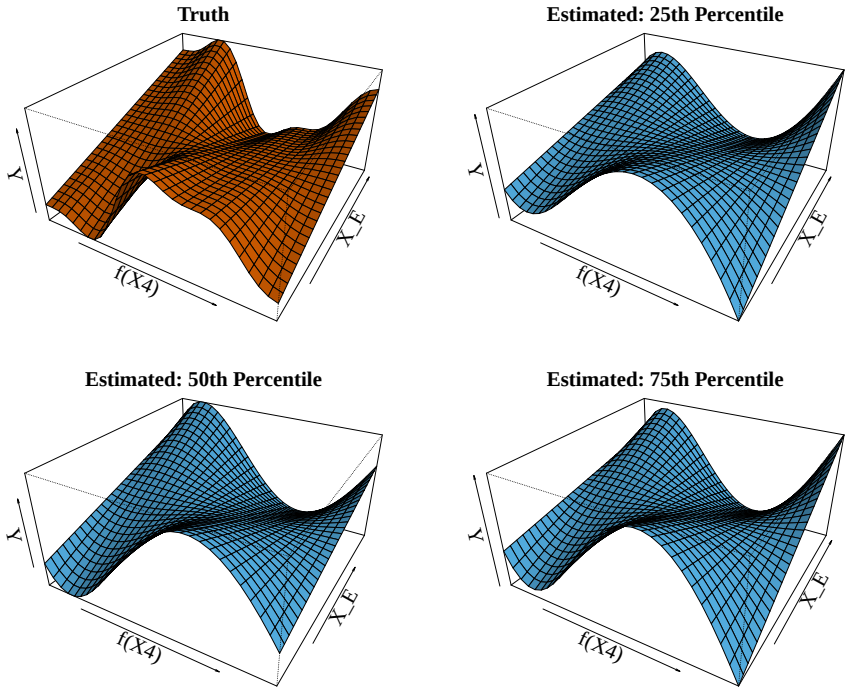


Figure 7: True and estimated interaction effects for $X_E \cdot f_4(X_4)$ in simulation scenario 1a).

4 Real Data Application

In this section we illustrate `sail` on several real data examples.

4.1 Alzheimer's Disease Neuroimaging Initiative

Alzheimer's is an irreversible neurodegenerative disease that results in a loss of mental function due to the deterioration of brain tissue. The overall goal of the Alzheimer's Disease Neuroimaging Initiative (ADNI) is to validate biomarkers for use in Alzheimer's disease clinical treatment trials [29]. The patients were selected into the study based on their clinical diagnosis: controls, mild cognitive impairment (MCI) or Alzheimer's disease (AD). PET amyloid imaging was used to asses amyloid beta (Aβ) protein load in 96 brain regions. The response was general cognitive decline measured by a continuous mini-mental state examination score. We applied `sail` to this data to see if there were any non-linear interactions between clinical diagnosis and Aβ protein in the 96 brain regions on mini-mental state examination.

There were a total of 343 patients who we divided randomly into equal sized training/validation/test splits. We ran the strong heredity `sail` with cubic B-splines and $\alpha = 0.1$. We also applied the `lasso, lassoBT, HierBasis` and `GLinternet` to this data. Using the same default settings and strategy as the simulation study, we ran each method on the training data, determined the optimal tuning parameter on the validation data, and assessed MSE on the test data. We repeated this process 200 times.

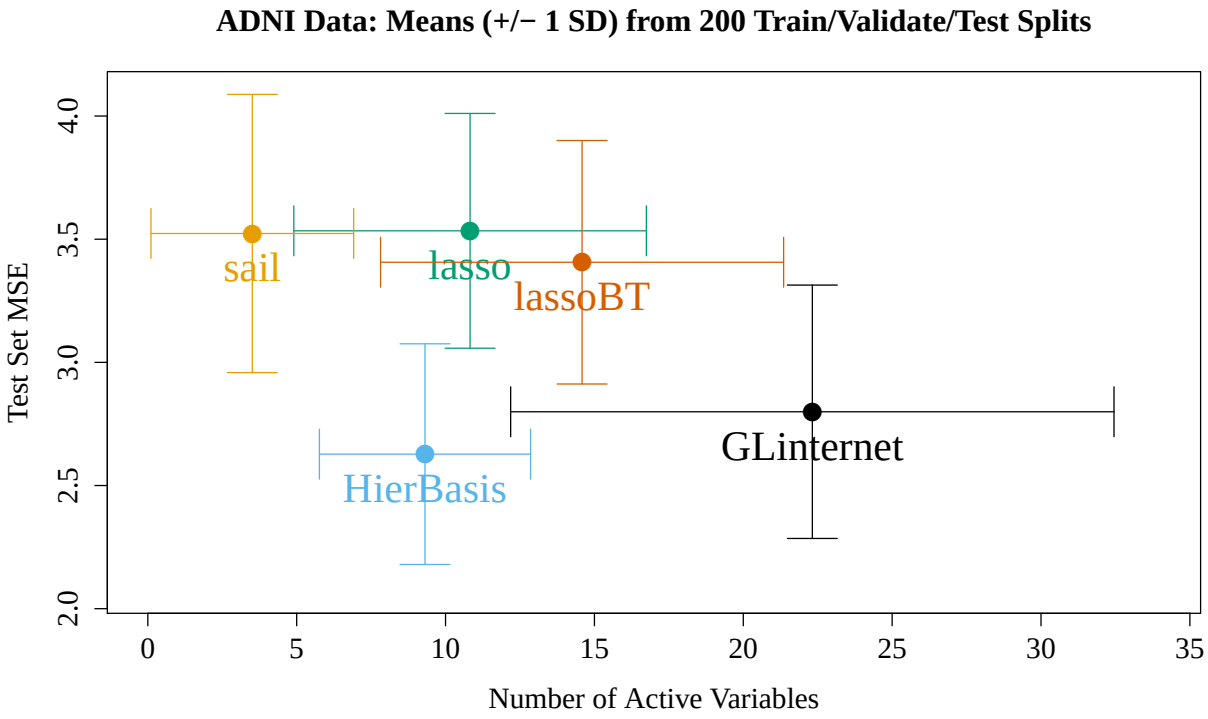


Figure 8: Mean test set MSE vs. mean number of active variables (±1 SD).

In Figure 8 we plot the mean test set MSE vs. the mean number of active variables ±1 SD. We see that `sail` produces the sparsest models but doesn't perform as well as `HierBasis` and `GLinternet` in terms of MSE. `sail` achieves similar MSE to both the `lasso` and `lassoBT` with fewer variables on average. `GLinternet` produces the largest models and seems to be sensitive to the train/validate/test split as evidenced by the large standard deviations.

To visualize the results from the `sail` method, we chose the train/validate/test split which led to the best test set MSE, and then plotted the interaction effects in Figure 9. The left panel shows the middle occipital gyrus left region in the occipital lobe known for visual object perception. We see that more Aβ protein loads leads to a worse cognitive score for the MCI and AD group but not for the controls. The right panel shows the cuneus region known which is known to be involved in basic visual processing. We see that more Aβ proteins leads to better cognitive scores for the MCI and AD group and poorer scores for

the controls.

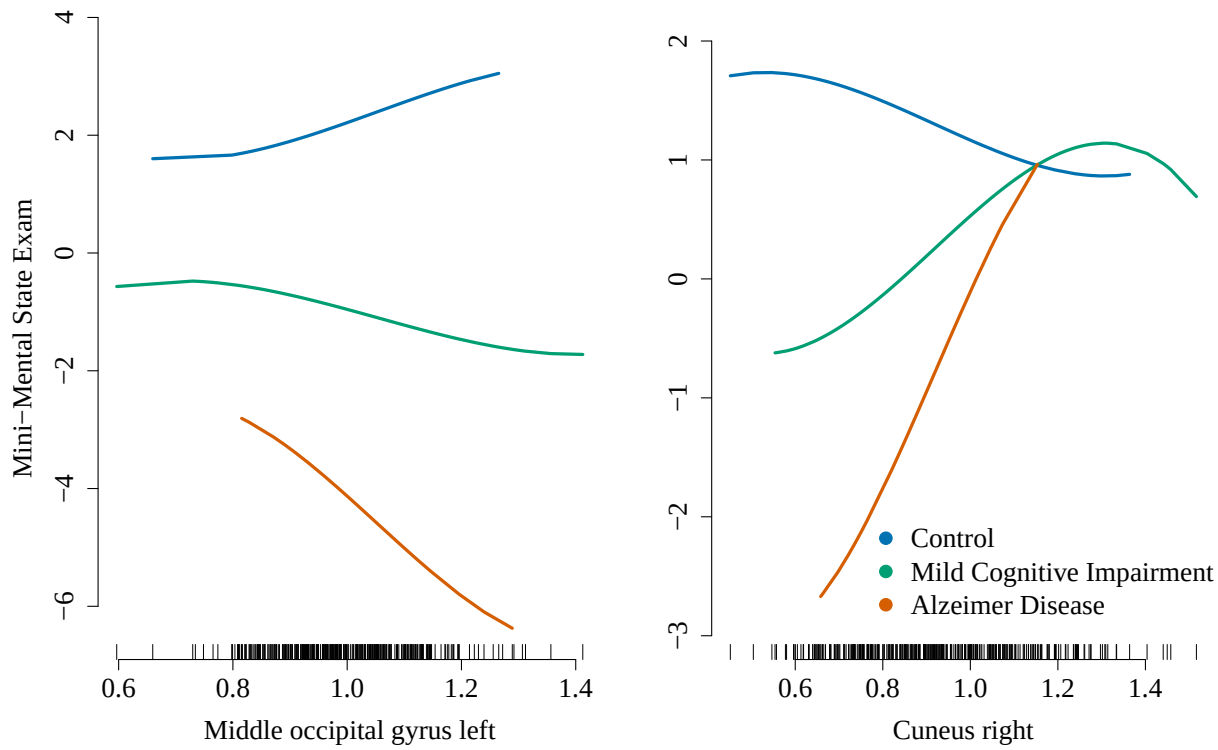


Figure 9: Estimated interaction effects by `sail` for ADNI data.

5 Discussion

In this article we introduce the sparse additive interaction learning model `sail` for detecting non-linear interactions with a key environmental or exposure variable in high-dimensional settings. Using a simple reparametrization, we are able to achieve both the weak and strong heredity property without using a complex penalty function. We use a blockwise coordinate descent algorithm to solve the `sail` objective function for both least-squares and logistic loss functions. We show that the `adaptive sail` has the oracle property. All our algorithms are implemented in a computationally efficient, well-documented and freely available `R` package. Furthermore, our method is flexible to handle any type of basis expansion including the identity map, which allows for linear interactions. Our implementation allows the user to

selectively apply the basis expansions to the predictors, allowing for example, a combination of continuous and categorical predictors. An extensive simulation study shows that `sail`, `adaptive sail` and `sail weak` outperform existing penalized regression methods in terms of prediction error, sensitivity and specificity when there are non-linear main effects only, as well as interactions with an exposure variable.

Our method however does have its limitations. `sail`can currently only handle $X_E \cdot f(X)$ or $f(X_E) \cdot X$ and does not allow for $f(X, X_E)$, i.e., only one of the variables in the interaction can have a non-linear effect and we do not consider the tensor product. The reparametrization leads to a non-convex optimization problem which makes convergence rates difficult to assess, though we did not experience any major convergence issues in our simulations and real data analysis. The memory footprint can also be an issue depending on the degree of the basis expansion and the number of variables.

To our knowledge, our proposal is the first to allow for non-linear interactions with a key exposure variable following the weak or strong heredity property in high-dimensional settings. We also provide a first software implementation for these models.

References

[1] Eric E Schadt. Molecular networks as sensors and drivers of common human diseases. *Nature*, 461(7261):218–223, 2009. 2

[2] Nicholas J Timpson, Celia MT Greenwood, Nicole Soranzo, Daniel J Lawson, and J Brent Richards. Genetic architecture: the shape of the genetic contribution to human traits and disease. *Nature Reviews Genetics*, 19(2):110, 2018. 2

[3] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. CRC Press, 2015. 2

[4] Peter McCullagh and John A Nelder. *Generalized linear models*, volume 37. CRC press, 1989. 4

[5] Hugh Chipman. Bayesian variable selection with related predictors. *Canadian Journal of Statistics*, 24(1):17–36, 1996. 4

[6] David R Cox. Interaction. *International Statistical Review/Revue Internationale de Statistique*, pages 1–24, 1984. 4

[7] Francis Bach, Rodolphe Jenatton, Julien Mairal, Guillaume Obozinski, et al. Structured sparsity through convex optimization. *Statistical Science*, 27(4):450–468, 2012. 4

[8] Peter Radchenko and Gareth M James. Variable selection using adaptive nonlinear interaction structures in high dimensions. *Journal of the American Statistical Association*, 105(492):1541–1553, 2010. 4, 8, 23

[9] Jacob Bien, Jonathan Taylor, Robert Tibshirani, et al. A lasso for hierarchical interactions. *The Annals of Statistics*, 41(3):1111–1141, 2013. 4, 7

[10] Nam Hee Choi, William Li, and Ji Zhu. Variable selection with the strong heredity constraint and its oracle property. *Journal of the American Statistical Association*, 105(489):354–364, 2010. 4, 7

[11] Peng Zhao, Guilherme Rocha, and Bin Yu. The composite absolute penalties family for grouped and hierarchical variable selection. *The Annals of Statistics*, pages 3468–3497, 2009. 7

[12] Yiyuan She and He Jiang. Group regularized estimation under structural hierarchy. *arXiv preprint arXiv:1411.4691*, 2014. 7

[13] Michael Lim and Trevor Hastie. Learning interactions via hierarchical group-lasso regularization. *Journal of Computational and Graphical Statistics*, 24(3):627–654, 2015. 7, 17

[14] Asad Haris, Daniela Witten, and Noah Simon. Convex modeling of interactions with strong heredity. *Journal of Computational and Graphical Statistics*, 25(4):981–1004, 2016. 7

[15] Ning Hao, Yang Feng, and Hao Helen Zhang. Model selection for high-dimensional quadratic regression via regularization. *Journal of the American Statistical Association*, pages 1–11, 2018. 8

[16] Rajen D Shah. Modelling interactions in high-dimensional data with backtracking. *Journal of Machine Learning Research*, 17(207):1–31, 2016. 8, 17

[17] Robert Tibshirani and Jerome Friedman. A pliable lasso. *arXiv preprint arXiv:1712.00484*, 2017. 8

[18] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360, 2001. 8

[19] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429, 2006. 9, 15, 17

[20] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010. 9, 12, 39

[21] Yi Yang and Hui Zou. A fast unified algorithm for solving group-lasso penalize learning problems. *Statistics and Computing*, 25(6):1129–1141, 2015. 12, 39

[22] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. 17

[23] Asad Haris, Ali Shojaie, and Noah Simon. Nonparametric regression with adaptive truncation via a convex hierarchical penalty. *arXiv preprint arXiv:1611.09972*, 2016. 17

[24] Pradeep Ravikumar, John Lafferty, Han Liu, and Larry Wasserman. Sparse additive models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(5):1009–1030, 2009. 17

[25] Alexandra Chouldechova and Trevor Hastie. Generalized additive model selection. *arXiv preprint arXiv:1506.03850*, 2015. 17

[26] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. 18

[27] Yi Lin, Hao Helen Zhang, et al. Component selection and smoothing in multivariate nonparametric regression. *The Annals of Statistics*, 34(5):2272–2297, 2006. 18, 19

[28] Jian Huang, Joel L Horowitz, and Fengrong Wei. Variable selection in nonparametric additive models. *Annals of statistics*, 38(4):2282, 2010. 18, 19

[29] Ronald Carl Petersen, PS Aisen, LA Beckett, MC Donohue, AC Gamst, DJ Harvey, CR Jack, WJ Jagust, LM Shaw, AW Toga, et al. Alzheimer's disease neuroimaging initiative (adni) clinical characterization. *Neurology*, 74(3):201–209, 2010. 26

[30] Yihui Xie. *Dynamic Documents with R and knitr*, volume 29. CRC Press, 2015. 45

A Algorithm Details

In this section we provide more specific details about the algorithms used to solve the `sail` objective function.

A.1 Least-Squares `sail` with Strong Heredity

A more detailed algorithm for fitting the least-squares `sail` model with strong heredity is given in Algorithm 3.

Algorithm 3 Blockwise Coordinate Descent for Least-Squares `sail` with Strong Heredity

function `sail`$(\boldsymbol{X}, Y, X_E, \texttt{basis}, \lambda, \alpha, w_j, w_E, w_{jE}, \epsilon)$ ▷ Algorithm for solving (9)

$\Psi_j \leftarrow \texttt{basis}(X_j)$, $\widetilde{\Psi}_j \leftarrow X_E \circ \Psi_j$ for $j = 1, \ldots, p$

Initialize: $\beta_0^{(0)} \leftarrow \bar{Y}$, $\beta_E^{(0)} = \boldsymbol{\theta}_j^{(0)} = \gamma_j^{(0)} \leftarrow 0$ for $j = 1, \ldots, p$.

Set iteration counter $k \leftarrow 0$

$R^* \leftarrow Y - \beta_0^{(k)} - \beta_E^{(k)} X_E - \sum_j (\boldsymbol{\Psi}_j + \gamma_j^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_j) \boldsymbol{\theta}_j^{(k)}$

repeat

- To update $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$

$\widetilde{X}_j \leftarrow \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_j \boldsymbol{\theta}_j^{(k)}$ for $j = 1, \ldots, p$

$R \leftarrow R^* + \sum_{j=1}^p \gamma_j^{(k)} \widetilde{X}_j$

$$\boldsymbol{\gamma}^{(k)(new)} \leftarrow \arg\min_{\boldsymbol{\gamma}} \frac{1}{2n} \left\| R - \sum_j \gamma_j \widetilde{X}_j \right\|_2^2 + \lambda\alpha \sum_j w_{jE} |\gamma_j|$$

$\Delta = \sum_j (\gamma_j^{(k)} - \gamma_j^{(k)(new)}) \widetilde{X}_j$

$R^* \leftarrow R^* + \Delta$

- To update $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p)$

$\widetilde{X}_j \leftarrow \boldsymbol{\Psi}_j + \gamma_j^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_j$ for $j = 1, \ldots, p$

for $j = 1, \ldots, p$ **do**

$R \leftarrow R^* + \widetilde{X}_j \boldsymbol{\theta}_j^{(k)}$

$$\boldsymbol{\theta}_j^{(k)(new)} \leftarrow \arg\min_{\boldsymbol{\theta}_j} \frac{1}{2n} \left\| R - \widetilde{X}_j \boldsymbol{\theta}_j \right\|_2^2 + \lambda(1-\alpha) w_j \|\theta_j\|_2$$

$\Delta = \widetilde{X}_j (\boldsymbol{\theta}_j^{(k)} - \boldsymbol{\theta}_j^{(k)(new)})$

$R^* \leftarrow R^* + \Delta$

- To update β_E

$\widetilde{X}_E \leftarrow X_E + \sum_j \gamma_j^{(k)} \widetilde{\boldsymbol{\Psi}}_j \boldsymbol{\theta}_j^{(k)}$

$R \leftarrow R^* + \beta_E^{(k)} \widetilde{X}_E$

$$\beta_E^{(k)(new)} \leftarrow S\left(\frac{1}{n \cdot w_E} \widetilde{X}_E^\top R, \lambda(1-\alpha) \right)$$

▷ $S(x, t) = \text{sign}(x)(|x| - t)_+$

$\Delta = (\beta_E^{(k)} - \beta_E^{(k)(new)}) \widetilde{X}_E$

$R^* \leftarrow R^* + \Delta$

- To update β_0

$R \leftarrow R^* + \beta_0^{(k)}$

$$\beta_0^{(k)(new)} \leftarrow \frac{1}{n} R^* \cdot \mathbf{1}$$

$\Delta = \beta_0^{(k)} - \beta_0^{(k)(new)}$

$R^* \leftarrow R^* + \Delta$

$k \leftarrow k + 1$

until convergence criterion is satisfied: $\left| Q(\boldsymbol{\Theta}^{(k-1)}) - Q(\boldsymbol{\Theta}^{(k)}) \right| / Q(\boldsymbol{\Theta}^{(k-1)}) < \epsilon$

A.2 Details on Update for $\boldsymbol{\theta}$

Here we discuss a computational speedup in the updates for the $\boldsymbol{\theta}$ parameter. The partial residual (R_s) used for updating $\boldsymbol{\theta}_s$ ($s \in 1, \ldots, p$) at the kth iteration is given by

$$R_s = Y - \widetilde{Y}^{(k)}_{(-s)} \tag{25}$$

where $\widetilde{Y}^{(k)}_{(-s)}$ is the fitted value at the kth iteration excluding the contribution from $\boldsymbol{\Psi}_s$:

$$\widetilde{Y}^{(k)}_{(-s)} = \beta_0^{(k)} - \beta_E^{(k)} X_E - \sum_{\ell \neq s} \boldsymbol{\Psi}_\ell \boldsymbol{\theta}_\ell^{(k)} - \sum_{\ell \neq s} \gamma_\ell^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_\ell \boldsymbol{\theta}_\ell^{(k)} \tag{26}$$

Using (26), (25) can be re-written as

$$\begin{aligned} R_s &= Y - \beta_0^{(k)} - \beta_E^{(k)} X_E - \sum_{j=1}^{p} (\boldsymbol{\Psi}_j + \gamma_j^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_j) \boldsymbol{\theta}_j^{(k)} + (\boldsymbol{\Psi}_s + \gamma_s^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_s) \boldsymbol{\theta}_s^{(k)} \\ &= R^* + (\boldsymbol{\Psi}_s + \gamma_s^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_s) \boldsymbol{\theta}_s^{(k)} \end{aligned} \tag{27}$$

where

$$R^* = Y - \beta_0^{(k)} - \beta_E^{(k)} X_E - \sum_{j=1}^{p} (\boldsymbol{\Psi}_j + \gamma_j^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_j) \boldsymbol{\theta}_j^{(k)} \tag{28}$$

Denote $\boldsymbol{\theta}_s^{(k)(\text{new})}$ the solution for predictor s at the kth iteration, given by:

$$\boldsymbol{\theta}_s^{(k)(new)} = \arg\min_{\boldsymbol{\theta}_j} \frac{1}{2n} \left\| R_s - (\boldsymbol{\Psi}_s + \gamma_s^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_s) \boldsymbol{\theta}_j \right\|_2^2 + \lambda(1-\alpha) w_s \|\theta_j\|_2 \tag{29}$$

Now we want to update the parameters for the next predictor $\boldsymbol{\theta}_{s+1}$ ($s+1 \in 1, \ldots, p$) at the kth iteration. The partial residual used to update $\boldsymbol{\theta}_{s+1}$ is given by

$$R_{s+1} = R^* + (\boldsymbol{\Psi}_{s+1} + \gamma_{s+1}^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_{s+1}) \boldsymbol{\theta}_{s+1}^{(k)} + (\boldsymbol{\Psi}_s + \gamma_s^{(k)} \beta_E^{(k)} \widetilde{\boldsymbol{\Psi}}_s)(\boldsymbol{\theta}_s^{(k)} - \boldsymbol{\theta}_s^{(k)(new)}) \tag{30}$$

where R^* is given by (28), $\boldsymbol{\theta}_s^{(k)}$ is the parameter value prior to the update, and $\boldsymbol{\theta}_s^{(k)(new)}$ is the updated value given by (29). Taking the difference between (27) and (30) gives

$$\begin{aligned}\Delta &= R_t - R_s \\ &= (\boldsymbol{\Psi}_t + \gamma_t^{(k)}\beta_E^{(k)}\widetilde{\boldsymbol{\Psi}}_t)\boldsymbol{\theta}_t^{(k)} + (\boldsymbol{\Psi}_s + \gamma_s^{(k)}\beta_E^{(k)}\widetilde{\boldsymbol{\Psi}}_s)(\boldsymbol{\theta}_s^{(k)} - \boldsymbol{\theta}_s^{(k)(new)}) - (\boldsymbol{\Psi}_s + \gamma_s^{(k)}\beta_E^{(k)}\widetilde{\boldsymbol{\Psi}}_s)\boldsymbol{\theta}_s^{(k)} \\ &= (\boldsymbol{\Psi}_t + \gamma_t^{(k)}\beta_E^{(k)}\widetilde{\boldsymbol{\Psi}}_t)\boldsymbol{\theta}_t^{(k)} - (\boldsymbol{\Psi}_s + \gamma_s^{(k)}\beta_E^{(k)}\widetilde{\boldsymbol{\Psi}}_s)\boldsymbol{\theta}_s^{(k)(new)} \qquad (31)\end{aligned}$$

Therefore $R_t = R_s + \Delta$, and the partial residual for updating the next predictor can be computed by updating the previous partial residual by Δ, given by (31). This formulation can lead to computational speedups especially when $\Delta = 0$, meaning the partial residual does not need to be re-calculated.

A.3 Least-Squares `sail` with Weak Heredity

The least-squares `sail` model with weak heredity has the form

$$\hat{Y} = \beta_0 \cdot \mathbf{1} + \sum_{j=1}^{p} \boldsymbol{\Psi}_j \boldsymbol{\theta}_j + \beta_E X_E + \sum_{j=1}^{p} \gamma_j (X_E \circ \boldsymbol{\Psi}_j)(\beta_E \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j) \qquad (32)$$

The objective function is given by

$$Q(\boldsymbol{\Theta}) = \frac{1}{2n}\left\|Y - \hat{Y}\right\|_2^2 + \lambda(1-\alpha)\left(w_E|\beta_E| + \sum_{j=1}^{p} w_j \|\boldsymbol{\theta}_j\|_2\right) + \lambda\alpha \sum_{j=1}^{p} w_{jE}|\gamma_j| \qquad (33)$$

Denote the n-dimensional residual column vector $R = Y - \hat{Y}$. The subgradient equations are given by

$$\frac{\partial Q}{\partial \beta_0} = \frac{1}{n}\left(Y - \beta_0 \cdot \mathbf{1} - \sum_{j=1}^{p} \mathbf{\Psi}_j \boldsymbol{\theta}_j - \beta_E X_E - \sum_{j=1}^{p} \gamma_j (X_E \circ \mathbf{\Psi}_j)(\beta_E \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j)\right)^\top \mathbf{1} = 0 \quad (34)$$

$$\frac{\partial Q}{\partial \beta_E} = -\frac{1}{n}\left(X_E + \sum_{j=1}^{p} \gamma_j (X_E \circ \mathbf{\Psi}_j)\mathbf{1}_{m_j}\right)^\top R + \lambda(1-\alpha) w_E s_1 = 0 \quad (35)$$

$$\frac{\partial Q}{\partial \boldsymbol{\theta}_j} = -\frac{1}{n}\left(\mathbf{\Psi}_j + \gamma_j (X_E \circ \mathbf{\Psi}_j)\right)^\top R + \lambda(1-\alpha) w_j s_2 = \mathbf{0} \quad (36)$$

$$\frac{\partial Q}{\partial \gamma_j} = -\frac{1}{n}\left((X_E \circ \mathbf{\Psi}_j)(\beta_E \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j)\right)^\top R + \lambda\alpha w_{jE} s_3 = 0 \quad (37)$$

where s_1 is in the subgradient of the ℓ_1 norm:

$$s_1 \in \begin{cases} \operatorname{sign}(\beta_E) & \text{if } \beta_E \neq 0 \\ [-1, 1] & \text{if } \beta_E = 0, \end{cases}$$

s_2 is in the subgradient of the ℓ_2 norm:

$$s_2 \in \begin{cases} \dfrac{\boldsymbol{\theta}_j}{\|\boldsymbol{\theta}_j\|_2} & \text{if } \boldsymbol{\theta}_j \neq \mathbf{0} \\ u \in \mathbb{R}^{m_j} : \|u\|_2 \leq 1 & \text{if } \boldsymbol{\theta}_j = \mathbf{0}, \end{cases}$$

and s_3 is in the subgradient of the ℓ_1 norm:

$$s_3 \in \begin{cases} \operatorname{sign}(\gamma_j) & \text{if } \gamma_j \neq 0 \\ [-1, 1] & \text{if } \gamma_j = 0. \end{cases}$$

Define the partial residuals, without the jth predictor for $j = 1, \ldots, p$, as

$$R_{(-j)} = Y - \beta_0 \cdot \mathbf{1} - \sum_{\ell \neq j} \mathbf{\Psi}_\ell \boldsymbol{\theta}_\ell - \beta_E X_E - \sum_{\ell \neq j} \gamma_\ell (X_E \circ \mathbf{\Psi}_\ell)(\beta_E \cdot \mathbf{1}_{m_\ell} + \boldsymbol{\theta}_\ell)$$

the partial residual without X_E as

$$R_{(-E)} = Y - \beta_0 \cdot \mathbf{1} - \sum_{j=1}^{p} \mathbf{\Psi}_j \boldsymbol{\theta}_j - \sum_{j=1}^{p} \gamma_j (X_E \circ \mathbf{\Psi}_j) \boldsymbol{\theta}_j$$

and the partial residual without the jth interaction for $j = 1, \ldots, p$

$$R_{(-jE)} = Y - \beta_0 \cdot \mathbf{1} - \sum_{j=1}^{p} \mathbf{\Psi}_j \boldsymbol{\theta}_j - \beta_E X_E - \sum_{\ell \neq j} \gamma_\ell (X_E \circ \mathbf{\Psi}_\ell)(\beta_E \cdot \mathbf{1}_{m_\ell} + \boldsymbol{\theta}_\ell)$$

From the subgradient Equation (35), we see that $\beta_E = 0$ is a solution if

$$\frac{1}{w_E} \left| \frac{1}{n} \left(X_E + \sum_{j=1}^{p} \gamma_j (X_E \circ \mathbf{\Psi}_j) \mathbf{1}_{m_j} \right)^\top R_{(-E)} \right| \leq \lambda(1-\alpha) \tag{38}$$

From the subgradient Equation (36), we see that $\boldsymbol{\theta}_j = \mathbf{0}$ is a solution if

$$\frac{1}{w_j} \left\| \frac{1}{n} \left(\mathbf{\Psi}_j + \gamma_j (X_E \circ \mathbf{\Psi}_j) \right)^\top R_{(-j)} \right\|_2 \leq \lambda(1-\alpha) \tag{39}$$

From the subgradient Equation (37), we see that $\gamma_j = 0$ is a solution if

$$\frac{1}{w_{jE}} \left| \frac{1}{n} \left((X_E \circ \mathbf{\Psi}_j)(\beta_E \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j) \right)^\top R_{(-jE)} \right| \leq \lambda\alpha \tag{40}$$

From the subgradient equations we see that

$$\hat{\beta}_0 = \left(Y - \sum_{j=1}^{p} \mathbf{\Psi}_j \hat{\boldsymbol{\theta}}_j - \hat{\beta}_E X_E - \sum_{j=1}^{p} \hat{\gamma}_j (X_E \circ \mathbf{\Psi}_j)(\hat{\beta}_E \cdot \mathbf{1}_{m_j} + \hat{\boldsymbol{\theta}}_j) \right)^\top \mathbf{1} \tag{41}$$

$$\hat{\beta}_E = S\left(\frac{1}{n \cdot w_E} \left(X_E + \sum_{j=1}^{p} \hat{\gamma}_j (X_E \circ \mathbf{\Psi}_j) \mathbf{1}_{m_j} \right)^\top R_{(-E)}, \lambda(1-\alpha) \right) \tag{42}$$

$$\lambda(1-\alpha) w_j \frac{\boldsymbol{\theta}_j}{\|\boldsymbol{\theta}_j\|_2} = \frac{1}{n} \left(\mathbf{\Psi}_j + \gamma_j (X_E \circ \mathbf{\Psi}_j) \right)^\top R_{(-j)} \tag{43}$$

$$\hat{\gamma}_j = S\left(\frac{1}{n \cdot w_{jE}} \left((X_E \circ \mathbf{\Psi}_j)(\beta_E \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j) \right)^\top R_{(-jE)}, \lambda\alpha \right) \tag{44}$$

where $S(x,t) = \text{sign}(x)(|x|-t)$ is the soft-thresholding operator. As was the case in the strong heredity `sail` model, there are closed form solutions for the intercept and β_E, each γ_j also has a closed form solution and can be solved efficiently for $j = 1, \ldots, p$ using the coordinate descent procedure implemented in the `glmnet` package [20], while we use the quadratic majorization technique implemented in the `gglasso` package [21] to solve (43). Algorithm 4 details the procedure used to fit the least-squares weak heredity `sail` model.

A.3.1 Lambda Max

The smallest value of λ for which the entire parameter vector $(\beta_E, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p, \gamma_1, \ldots, \gamma_p)$ is $\mathbf{0}$ is:

$$\lambda_{max} = \frac{1}{n} \max \left\{ \frac{1}{(1-\alpha)w_E} \left(X_E + \sum_{j=1}^{p} \gamma_j (X_E \circ \boldsymbol{\Psi}_j) \mathbf{1}_{m_j} \right)^\top R_{(-E)}, \right. \\ \max_j \frac{1}{(1-\alpha)w_j} \left\| (\boldsymbol{\Psi}_j + \gamma_j (X_E \circ \boldsymbol{\Psi}_j))^\top R_{(-j)} \right\|_2, \\ \left. \max_j \frac{1}{\alpha w_{jE}} \left((X_E \circ \boldsymbol{\Psi}_j)(\beta_E \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j) \right)^\top R_{(-jE)} \right\} \quad (45)$$

which reduces to

$$\lambda_{max} = \frac{1}{n(1-\alpha)} \max \left\{ \frac{1}{w_E} (X_E)^\top R_{(-E)}, \max_j \frac{1}{w_j} \left\| (\boldsymbol{\Psi}_j)^\top R_{(-j)} \right\|_2 \right\}$$

This is the same λ_{max} as the least-squares strong heredity `sail` model.

Algorithm 4 Coordinate descent for least-squares `sail` with weak heredity

function `sail`$(\boldsymbol{X}, Y, X_E, \texttt{basis}, \lambda, \alpha, w_j, w_E, w_{jE}, \epsilon)$ ▷ Algorithm for solving (33)

  $\Psi_j \leftarrow \texttt{basis}(X_j)$, $\widetilde{\Psi}_j \leftarrow X_E \circ \Psi_j$ for $j = 1, \ldots, p$

  Initialize: $\beta_0^{(0)} \leftarrow \bar{Y}$, $\beta_E^{(0)} = \boldsymbol{\theta}_j^{(0)} = \gamma_j^{(0)} \leftarrow 0$ for $j = 1, \ldots, p$.

  Set iteration counter $k \leftarrow 0$

  $R^* \leftarrow Y - \beta_0^{(k)} - \beta_E^{(k)} X_E - \sum_j \boldsymbol{\Psi}_j \boldsymbol{\theta}_j^{(k)} - \sum_j \gamma_j^{(k)} \widetilde{\boldsymbol{\Psi}}_j (\beta_E^{(k)} \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j^{(k)})$

  repeat

    • To update $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_p)$

      $\widetilde{X}_j \leftarrow \widetilde{\boldsymbol{\Psi}}_j (\beta_E^{(k)} \cdot \mathbf{1}_{m_j} + \boldsymbol{\theta}_j^{(k)})$ for $j = 1, \ldots, p$

      $R \leftarrow R^* + \sum_{j=1}^p \gamma_j^{(k)} \widetilde{X}_j$

$$\boldsymbol{\gamma}^{(k)(new)} \leftarrow \arg\min_{\boldsymbol{\gamma}} \frac{1}{2n} \left\| R - \sum_j \gamma_j \widetilde{X}_j \right\|_2^2 + \lambda\alpha \sum_j w_{jE} |\gamma_j|$$

      $\Delta = \sum_j (\gamma_j^{(k)} - \gamma_j^{(k)(new)}) \widetilde{X}_j$

      $R^* \leftarrow R^* + \Delta$

    • To update $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_p)$

      $\widetilde{X}_j \leftarrow \boldsymbol{\Psi}_j + \gamma_j^{(k)} \widetilde{\boldsymbol{\Psi}}_j$ for $j = 1, \ldots, p$

      for $j = 1, \ldots, p$ **do**

        $R \leftarrow R^* + \widetilde{X}_j \boldsymbol{\theta}_j^{(k)}$

$$\boldsymbol{\theta}_j^{(k)(new)} \leftarrow \arg\min_{\boldsymbol{\theta}_j} \frac{1}{2n} \left\| R - \widetilde{X}_j \boldsymbol{\theta}_j \right\|_2^2 + \lambda(1-\alpha) w_j \|\theta_j\|_2$$

        $\Delta = \widetilde{X}_j (\boldsymbol{\theta}_j^{(k)} - \boldsymbol{\theta}_j^{(k)(new)})$

        $R^* \leftarrow R^* + \Delta$

    • To update β_E

      $\widetilde{X}_E \leftarrow X_E + \sum_j \gamma_j^{(k)} \widetilde{\boldsymbol{\Psi}}_j \mathbf{1}_{m_j}$

      $R \leftarrow R^* + \beta_E^{(k)} \widetilde{X}_E$

$$\beta_E^{(k)(new)} \leftarrow S\left(\frac{1}{n \cdot w_E} \widetilde{X}_E^\top R, \lambda(1-\alpha) \right)$$

▷ $S(x,t) = \text{sign}(x)(|x| - t)_+$

      $\Delta = (\beta_E^{(k)} - \beta_E^{(k)(new)}) \widetilde{X}_E$

      $R^* \leftarrow R^* + \Delta$

    • To update β_0

      $R \leftarrow R^* + \beta_0^{(k)}$

$$\beta_0^{(k)(new)} \leftarrow \frac{1}{n} R^* \cdot \mathbf{1}$$

      $\Delta = \beta_0^{(k)} - \beta_0^{(k)(new)}$

      $R^* \leftarrow R^* + \Delta$

    $k \leftarrow k + 1$

  until convergence criterion is satisfied: $\left| Q(\boldsymbol{\Theta}^{(k-1)}) - Q(\boldsymbol{\Theta}^{(k)}) \right| / Q(\boldsymbol{\Theta}^{(k-1)}) < \epsilon$

B Simulation Results

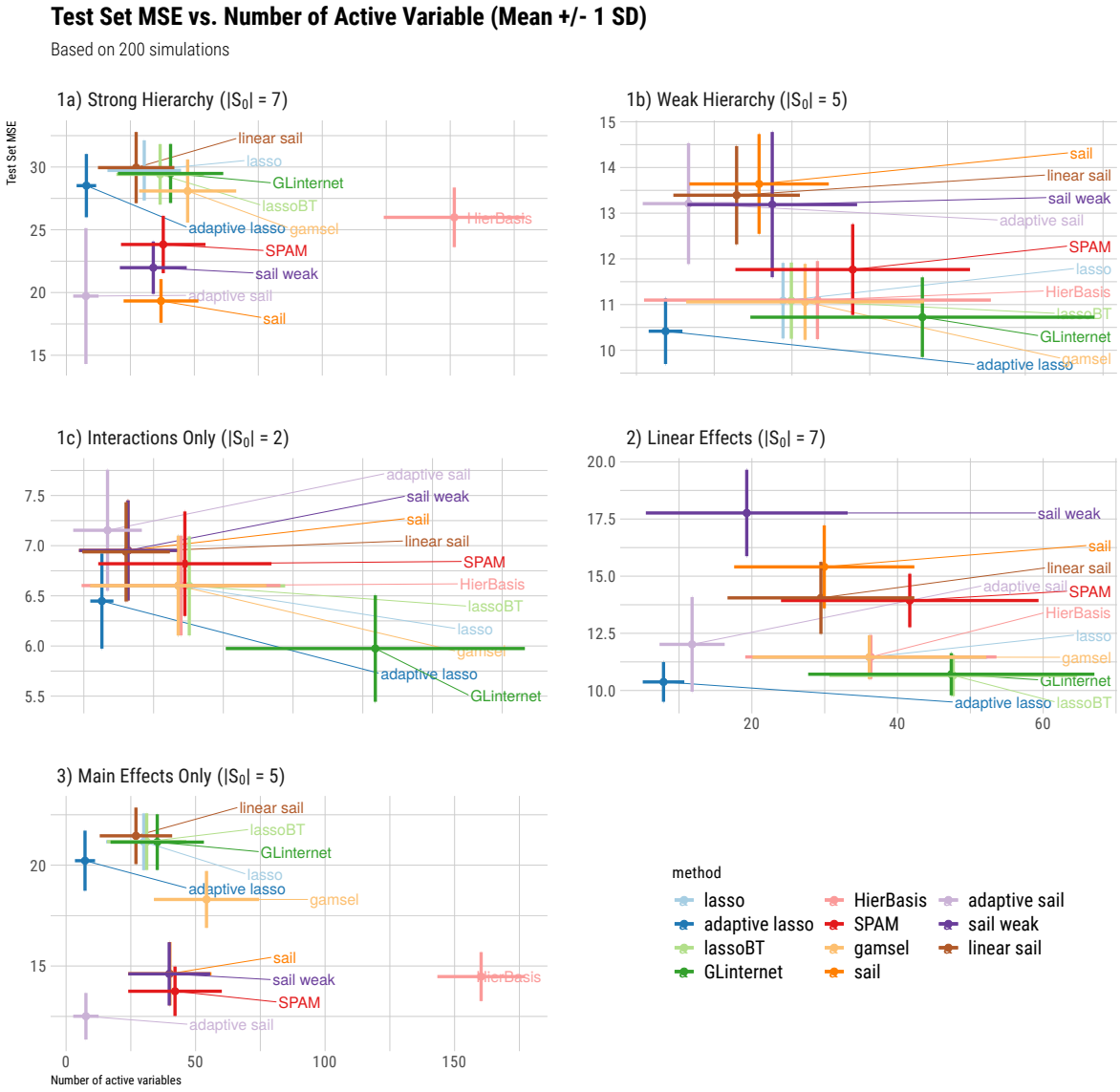


Figure B.1: Test set MSE vs number of active variables results.

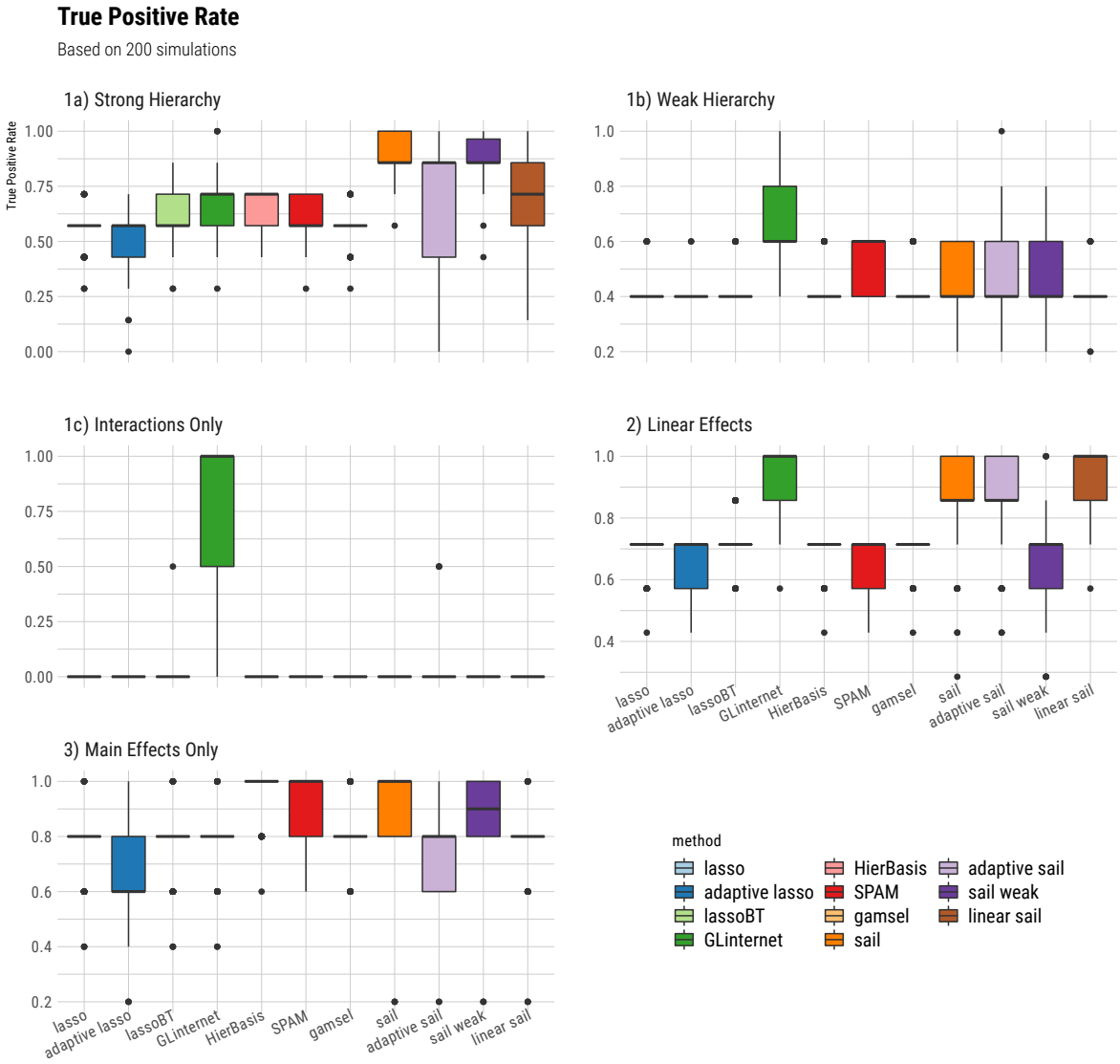


Figure B.2: True positive rate results.

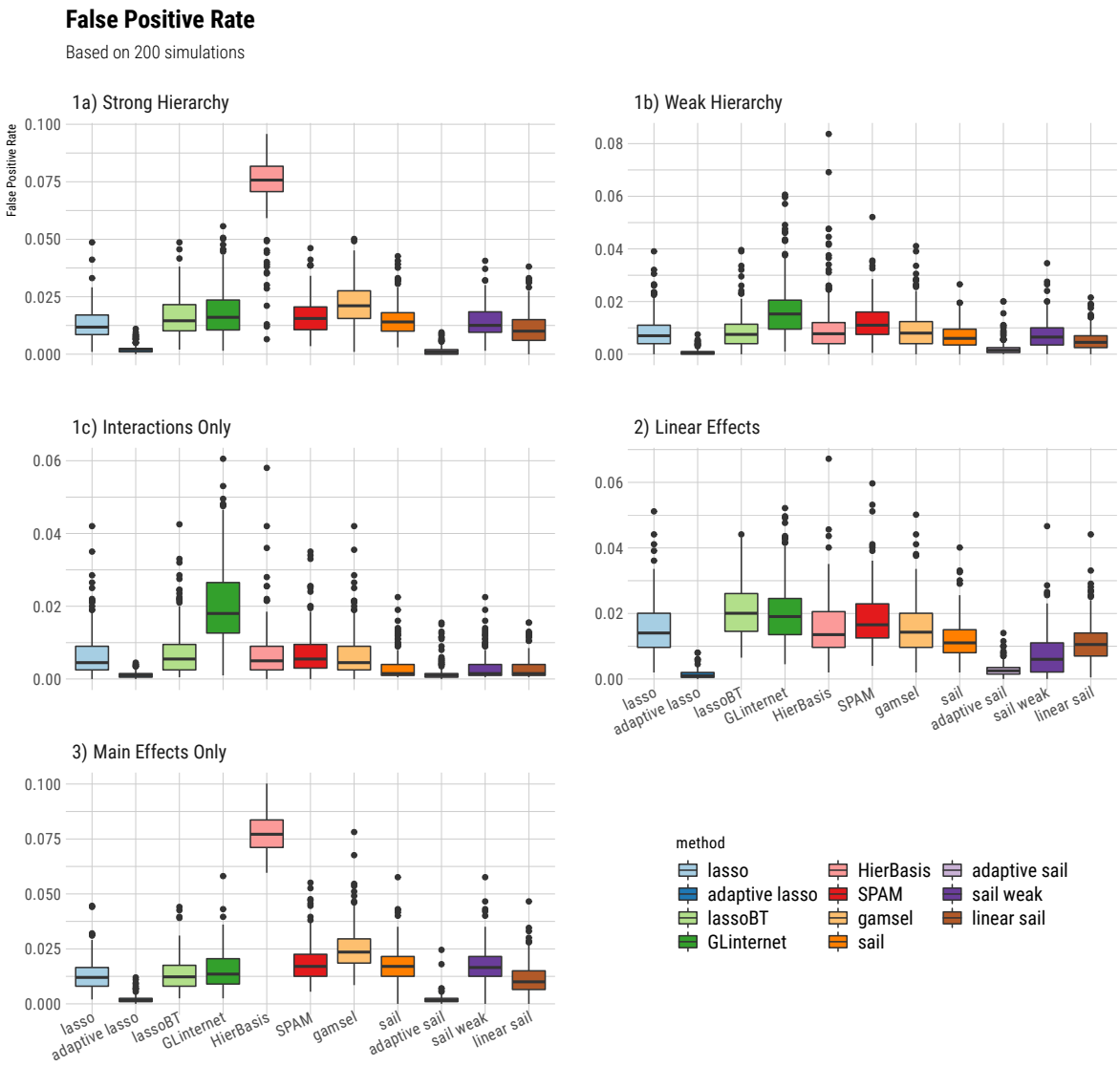


Figure B.3: False positive rate results.

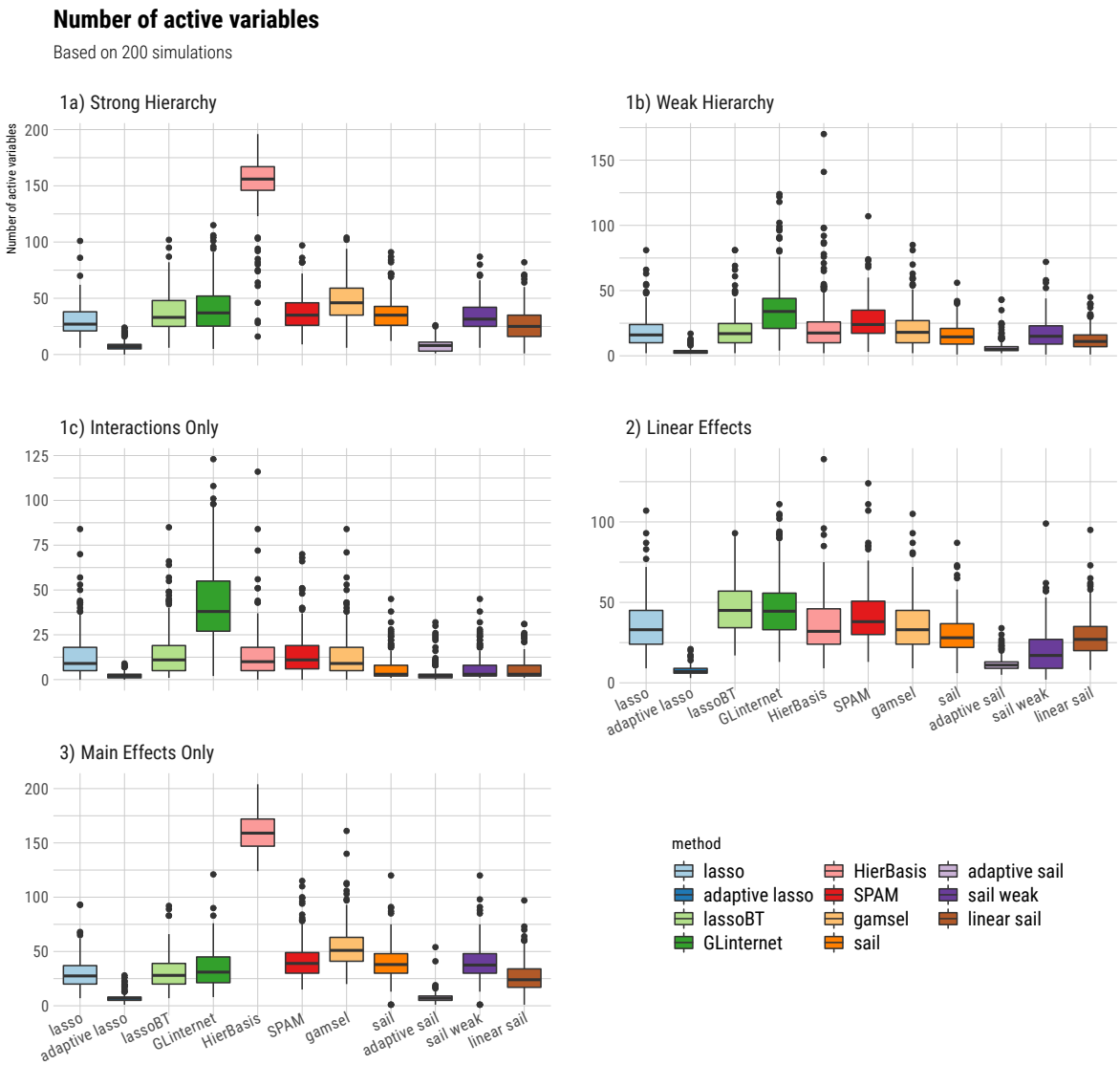


Figure B.4: Number of active variables results.

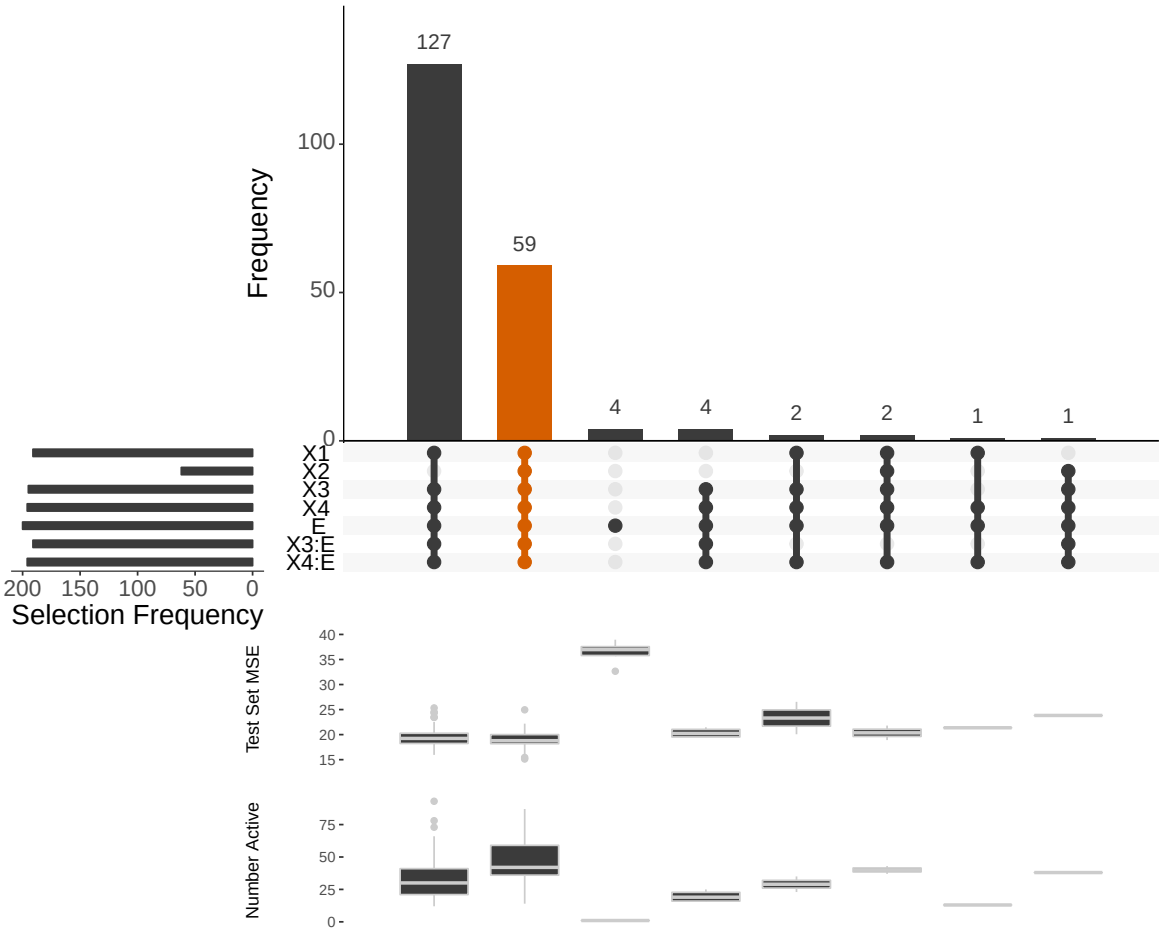


Figure B.5: Selection rates across 200 simulations of scenario 1a) for strong heredity `sail`.

C `sail` Package Showcase

In this section we briefly introduce the freely available and open source `sail` package in `R`. More comprehensive documentation is available at https://sahirbhatnagar.com/sail. Note that this entire section is reproducible; the code and text are combined in an `.Rnw`[1] file and compiled using `knitr` [30].

[1] scripts available at https://github.com/sahirbhatnagar/sail/tree/master/manuscript

C.1 Installation

The package can be installed from GitHub via

```
install.packages("pacman")
pacman::p_load_gh('sahirbhatnagar/sail')
```

C.2 Quick Start

We give a quick overview of the main functions and go into details in other vignettes. We will use the simulated data which ships with the package and can be loaded via:

```
library(sail)
data("sailsim")
names(sailsim)

## [1] "x"         "y"         "e"         "f1"        "f2"        "f3"
## [7] "f4"        "f3.inter" "f4.inter"
```

We first define a basis expansion. In this example we use B-splines with degree 5.

```
library(splines)
f.basis <- function(x) splines::bs(x, degree = 5)
```

Next we fit the model using the most basic call to `sail`

```
fit <- sail(x = sailsim$x, y = sailsim$y, e = sailsim$e, basis = f.basis)
```

`fit` is an object of class `sail` that contains all the relevant information of the fitted model including the estimated coefficients at each value of λ (by default the program chooses its own decreasing sequence of 100 λ values). There are `print`, `plot`, `coef` and `predict` methods of objects of class `sail`. We can call the `print` method:

```
fit

##
## Call:  sail(x = sailsim$x, y = sailsim$y, e = sailsim$e, basis = f.basis)
##
##      df_main df_interaction df_environment     %Dev  Lambda
## s1         0              0              0 0.000000 1.48800
## s2         0              0              1 0.001701 1.42000
```

```
## s3        0             0             1 0.003359 1.35600
## s4        0             0             1 0.004899 1.29400
## s5        0             0             1 0.006354 1.23500
## s6        1             0             1 0.014750 1.17900
## s7        2             0             1 0.045860 1.12600
## s8        2             0             1 0.080070 1.07500
## s9        2             0             1 0.111900 1.02600
## s10       2             0             1 0.141500 0.97900
## s11       2             0             1 0.169000 0.93450
## s12       2             0             1 0.194500 0.89210
## s13       2             0             1 0.218100 0.85150
## s14       2             0             1 0.240100 0.81280
## s15       2             0             1 0.260400 0.77590
## s16       2             0             1 0.279200 0.74060
## s17       3             1             1 0.465100 0.70690
## s18       3             1             1 0.479500 0.67480
## s19       3             1             1 0.493000 0.64410
## s20       3             2             1 0.508300 0.61490
## s21       3             2             1 0.524200 0.58690
## s22       3             2             1 0.536600 0.56020
## s23       3             2             1 0.548100 0.53480
## s24       4             2             1 0.558000 0.51050
## s25       5             2             1 0.568900 0.48730
## s26       5             2             1 0.579900 0.46510
## s27       5             2             1 0.590100 0.44400
## s28       5             2             1 0.599500 0.42380
## s29       5             3             1 0.609800 0.40450
## s30       5             3             1 0.619200 0.38620
## s31       5             3             1 0.628200 0.36860
## s32       6             3             1 0.637200 0.35190
## s33       6             3             1 0.646800 0.33590
## s34       6             3             1 0.655000 0.32060
## s35       6             3             1 0.663300 0.30600
## s36       7             3             1 0.671600 0.29210
## s37       7             3             1 0.679500 0.27880
## s38       7             3             1 0.687400 0.26620
## s39       7             3             1 0.694600 0.25410
## s40       7             4             1 0.702700 0.24250
## s41       7             4             1 0.734100 0.23150
## s42       7             4             1 0.739300 0.22100
## s43       7             5             1 0.745200 0.21090
```

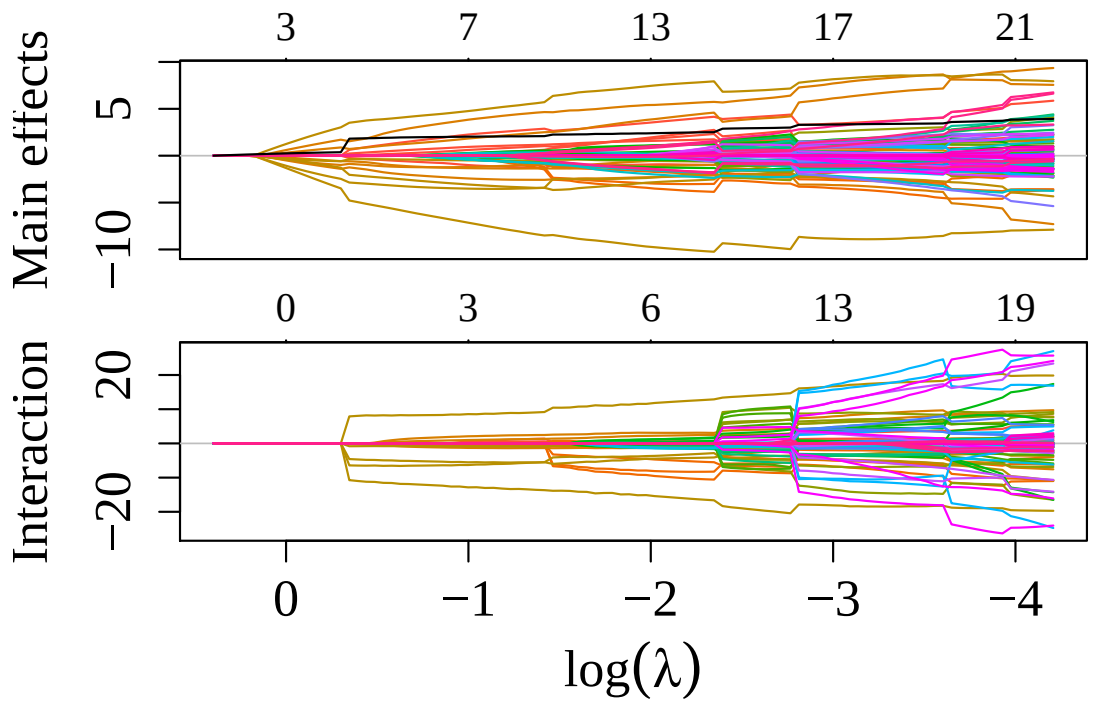
```
## s44       7               6               1 0.754400 0.20130
## s45       7               6               1 0.760700 0.19220
## s46       7               6               1 0.767300 0.18350
## s47       7               6               1 0.771800 0.17510
## s48       7               6               1 0.776100 0.16720
## s49       8               6               1 0.780700 0.15960
## s50       8               6               1 0.784500 0.15230
## s51       9               6               1 0.788700 0.14540
## s52      10               6               1 0.792700 0.13880
## s53      12               6               1 0.797800 0.13250
## s54      13               6               1 0.802600 0.12640
## s55      13               6               1 0.807500 0.12070
## s56      13               6               1 0.812200 0.11520
## s57      14               6               1 0.816800 0.11000
## s58      14               6               1 0.821400 0.10500
## s59      15               6               1 0.826100 0.10020
## s60      15               7               1 0.831200 0.09565
## s61      13              10               1 0.871400 0.09131
## s62      13              10               1 0.873800 0.08716
## s63      13              10               1 0.876200 0.08319
## s64      14              10               1 0.878600 0.07941
## s65      14              10               1 0.881000 0.07580
## s66      14              11               1 0.883400 0.07236
## s67      14              11               1 0.885900 0.06907
## s68      14              11               1 0.888400 0.06593
## s69      14              11               1 0.890600 0.06293
## s70      16              13               1 0.930600 0.06007
## s71      16              13               1 0.932300 0.05734
## s72      16              13               1 0.934200 0.05474
## s73      16              13               1 0.936100 0.05225
## s74      16              13               1 0.938500 0.04987
## s75      16              13               1 0.940000 0.04761
## s76      17              13               1 0.942100 0.04544
## s77      17              13               1 0.943800 0.04338
## s78      17              14               1 0.945600 0.04141
## s79      17              14               1 0.947300 0.03952
## s80      18              14               1 0.949000 0.03773
## s81      18              14               1 0.950700 0.03601
## s82      18              14               1 0.952400 0.03438
## s83      18              14               1 0.954400 0.03281
## s84      18              14               1 0.955800 0.03132
```

```
## s85       18             14             1 0.957200 0.02990
## s86       19             14             1 0.959100 0.02854
## s87       19             15             1 0.960300 0.02724
## s88       19             15             1 0.967900 0.02600
## s89       19             15             1 0.968900 0.02482
## s90       19             15             1 0.970100 0.02369
## s91       19             15             1 0.971300 0.02262
## s92       19             15             1 0.972400 0.02159
## s93       19             15             1 0.973500 0.02061
## s94       19             15             1 0.974600 0.01967
## s95       20             19             1 0.977900 0.01878
## s96       20             19             1 0.978600 0.01792
## s97       20             19             1 0.979300 0.01711
## s98       20             19             1 0.980000 0.01633
## s99       20             19             1 0.980700 0.01559
## s100      20             19             1 0.981400 0.01488
```

When `expand = TRUE` (i.e. the user did not provide their own design matrix), the `df_main` and `df_interaction` columns correspond to the number of non-zero predictors present in the model before basis expansion. This does not correspond to the number of non-zero coefficients in the model, but rather the number of unique variables. In this example we expanded each column of $\mathbf{X}$ to five columns. If `df_main=4`, `df_interaction=2` and `df_environment=1`, then the total number of non-zero coefficients would be $5 \times (4 + 2) + 1$.

The entire solution path can be plotted via the `plot` method for objects of class `sail`. The y-axis is the value of the coefficient and the x-axis is the $\log(\lambda)$. Each line represents a coefficient in the model, and each color represents a variable (i.e. in this example a given variable will have 5 lines when it is non-zero). The numbers at the top of the plot represent the number of non-zero variables in the model: top panel (`df_main` + `df_environment`), bottom panel (`df_interaction`). The black line is the coefficient path for the environment variable.

```
plot(fit)
```



The estimated coefficients at each value of lambda is given by (matrix partially printed here for brevity)

```
coef(fit)[1:6,50:55]

## 6 x 6 sparse Matrix of class "dgCMatrix"
##                     s50        s51        s52        s53        s54
## (Intercept)  5.2908242  5.2837492  5.2803715  5.2753572  5.2717869
## X1_1        -0.9792849 -0.9604046 -0.9449616 -0.9220738 -0.9171304
## X1_2         1.6903252  1.7894886  1.8924485  1.9952094  2.1042358
## X1_3         1.6463057  1.7049842  1.7722916  1.8251613  1.8951562
## X1_4         1.5224653  1.5433528  1.5663095  1.5854332  1.6102159
## X1_5         3.3386403  3.4183219  3.4908074  3.5763781  3.6633809
##                     s55
## (Intercept)  5.2695399
## X1_1        -0.9270958
## X1_2         2.2058453
## X1_3         1.9642875
## X1_4         1.6322047
## X1_5         3.7453708
```

The corresponding predicted response at each value of lambda (matrix partially printed here for brevity):

```
predict(fit)[1:5,50:55]

##            s50       s51       s52       s53       s54       s55
## [1,]  6.244693  6.199302  6.185402  6.177991  6.156173  6.124271
## [2,]  3.002799  2.995418  3.038701  3.079700  3.143715  3.209065
## [3,]  2.073305  2.043476  2.016319  1.997172  1.966900  1.957271
## [4,] 13.488945 13.490766 13.360998 13.384370 13.350671 13.324117
## [5,]  1.225516  1.210346  1.134420  1.156355  1.156696  1.156135
```

The predicted response at a specific value of lambda can be specified by the `s` argument:

```
predict(fit, s = 0.8)[1:5, ]

## [1] 5.624232 4.940944 3.847965 6.687777 3.058125
```

You can specify more than one value for ‘s‘:

```
predict(fit, s = c(0.8, 0.2))[1:5, ]

##             1         2
## [1,] 5.624232  6.523025
## [2,] 4.940944  2.975046
## [3,] 3.847965  2.326672
## [4,] 6.687777 13.956092
## [5,] 3.058125  1.568897
```

You can also extract a list of active variables (i.e. variables with a non-zero estimated coefficient) for each value of lambda:

```
fit[["active"]][50:55]

## [[1]]
##  [1] "X1"    "X2"    "X3"    "X4"    "X8"    "X10"   "X11"   "X20"
##  [9] "X1:E"  "X2:E"  "X3:E"  "X4:E"  "X8:E"  "X11:E" "E"
##
## [[2]]
##  [1] "X1"    "X2"    "X3"    "X4"    "X6"    "X8"    "X10"   "X11"
##  [9] "X20"   "X1:E"  "X2:E"  "X3:E"  "X4:E"  "X8:E"  "X11:E" "E"
##
## [[3]]
##  [1] "X1"    "X2"    "X3"    "X4"    "X6"    "X8"    "X10"   "X11"
##  [9] "X16"   "X20"   "X1:E"  "X2:E"  "X3:E"  "X4:E"  "X8:E"  "X11:E"
```

```
## [17] "E"
##
## [[4]]
##  [1] "X1"    "X2"    "X3"    "X4"    "X6"    "X8"    "X10"   "X11"
##  [9] "X15"   "X16"   "X19"   "X20"   "X1:E"  "X2:E"  "X3:E"  "X4:E"
## [17] "X8:E"  "X11:E" "E"
##
## [[5]]
##  [1] "X1"    "X2"    "X3"    "X4"    "X5"    "X6"    "X8"    "X10"
##  [9] "X11"   "X15"   "X16"   "X19"   "X20"   "X1:E"  "X2:E"  "X3:E"
## [17] "X4:E"  "X8:E"  "X11:E" "E"
##
## [[6]]
##  [1] "X1"    "X2"    "X3"    "X4"    "X5"    "X6"    "X8"    "X10"
##  [9] "X11"   "X15"   "X16"   "X19"   "X20"   "X1:E"  "X2:E"  "X3:E"
## [17] "X4:E"  "X8:E"  "X11:E" "E"
```

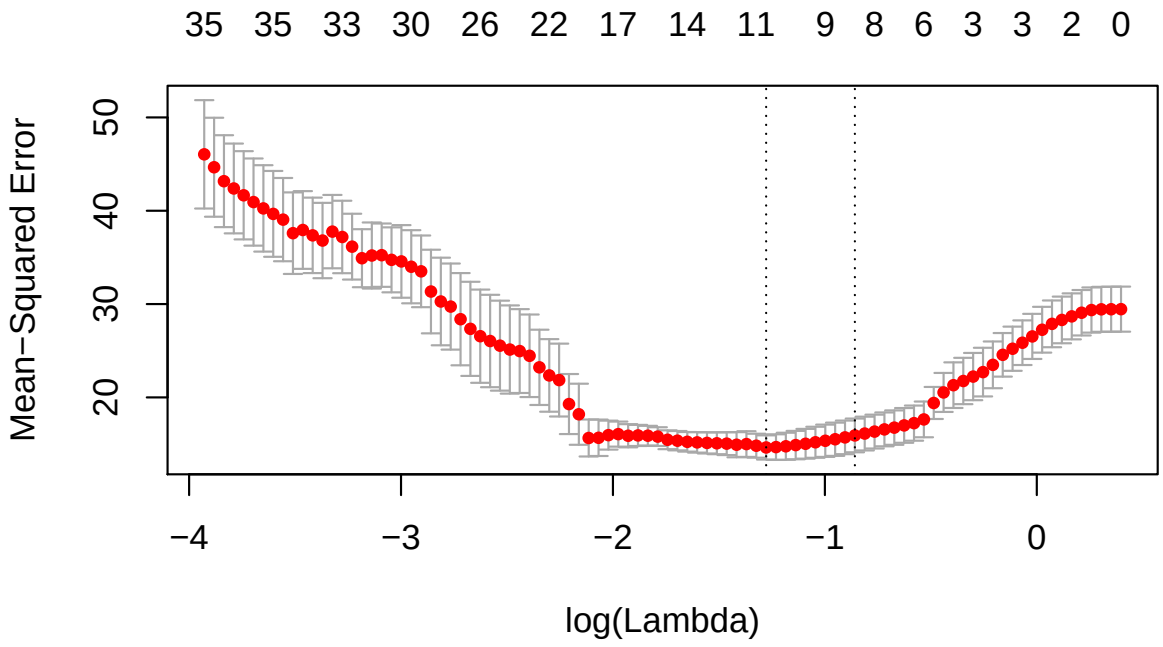
C.3 Cross-Validation

`cv.sail` is the main function to do cross-validation along with `plot`, `predict`, and `coef` methods for objects of class `cv.sail`. We run it in parallel:

```
set.seed(432) # to reproduce results (randomness due to CV folds)
library(doMC)
registerDoMC(cores = 8)
cvfit <- cv.sail(x = sailsim$x, y = sailsim$y, e = sailsim$e, basis = f.basis,
nfolds = 5, parallel = TRUE)
```

We plot the cross-validated error curve which has the mean-squared error on the y-axis and $\log(\lambda)$ on the x-axis. It includes the cross-validation curve (red dotted line), and upper and lower standard deviation curves along the λ sequence (error bars). Two selected λ's are indicated by the vertical dotted lines (see below). The numbers at the top of the plot represent the total number of non-zero variables at that value of λ (`df_main` + `df_environment` + `df_interaction`):

```
plot(cvfit)
```



`lambda.min` is the value of λ that gives minimum mean cross-validated error. The other λ saved is `lambda.1se`, which gives the most regularized model such that error is within one standard error of the minimum. We can view the selected λ's and the corresponding coefficients:

```
cvfit[["lambda.min"]]

## [1] 0.2788355

cvfit[["lambda.1se"]]

## [1] 0.4238052
```

The estimated nonzero coefficients at `lambda.1se` and `lambda.min`:

```
predict(cvfit, type = "nonzero", s="lambda.1se") # lambda.1se is the default

##                       1
## (Intercept)  5.42162330
## X1_1        -0.78259292
## X1_2         0.13434743
## X1_3         0.43872182
## X1_4         0.79452833
```

```
## X1_5          1.92193021
## X3_1          3.45123833
## X3_2          1.56772288
## X3_3         -1.27179135
## X3_4         -2.27792209
## X3_5         -1.23352137
## X4_1          4.47182924
## X4_2         -2.87488058
## X4_3         -6.65073351
## X4_4         -3.44085202
## X4_5         -0.47032703
## X8_1          0.17507051
## X8_2          0.11619435
## X8_3          0.01783624
## X8_4         -0.10505726
## X8_5         -0.24485431
## X11_1         0.13691003
## X11_2         0.02857999
## X11_3        -0.07324928
## X11_4        -0.20944262
## X11_5        -0.22648392
## E             1.99575642
## X3_1:E        1.80711590
## X3_2:E        0.82088128
## X3_3:E       -0.66592746
## X3_4:E       -1.19275137
## X3_5:E       -0.64588877
## X4_1:E        8.31249627
## X4_2:E       -5.34399523
## X4_3:E      -12.36277025
## X4_4:E       -6.39605585
## X4_5:E       -0.87427125
```

```
predict(cvfit, type = "nonzero", s = "lambda.min")
```

```
##                       1
## (Intercept)   5.43619834
## X1_1         -1.23832799
## X1_2          0.48736979
## X1_3          0.88915804
## X1_4          1.10813495
## X1_5          2.68123247
```

```
## X2_1         -0.21985179
## X2_2         -0.95071069
## X2_3         -0.76066212
## X2_4         -0.17413616
## X2_5         -0.06845895
## X3_1          4.42751615
## X3_2          2.28700197
## X3_3         -1.36817303
## X3_4         -2.56642377
## X3_5         -1.03093978
## X4_1          5.37962120
## X4_2         -3.35568118
## X4_3         -8.07355346
## X4_4         -3.50305791
## X4_5         -0.55031385
## X8_1          0.47925233
## X8_2          0.33992532
## X8_3          0.07652614
## X8_4         -0.28114406
## X8_5         -0.66616044
## X11_1         0.34768804
## X11_2         0.06274138
## X11_3        -0.20149241
## X11_4        -0.67859653
## X11_5        -0.69589312
## X20_1         0.10225010
## X20_2        -0.03831915
## X20_3        -0.17781683
## X20_4        -0.18564133
## X20_5         0.09741855
## E             2.06410796
## X1_1:E       -0.53459297
## X1_2:E        0.21040020
## X1_3:E        0.38385439
## X1_4:E        0.47838792
## X1_5:E        1.15750273
## X3_1:E        2.37995847
## X3_2:E        1.22935061
## X3_3:E       -0.73544508
## X3_4:E       -1.37955047
## X3_5:E       -0.55416937
```

```
## X4_1:E        8.87349228
## X4_2:E       -5.53507578
## X4_3:E      -13.31703694
## X4_4:E       -5.77816841
## X4_5:E       -0.90772296
```

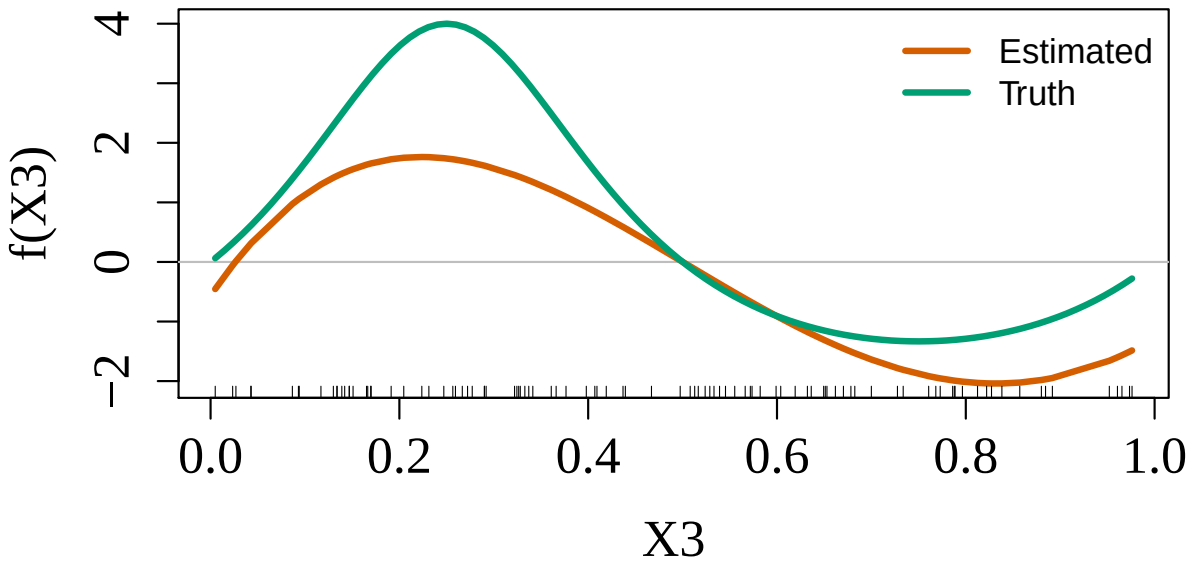
C.4 Visualizing the Effect of the Non-linear Terms

B-splines are difficult to interpret. We provide a plotting function to visualize the effect of the non-linear function on the response.

C.4.1 Main Effects

Since we are using simulated data, we also plot the true curve:

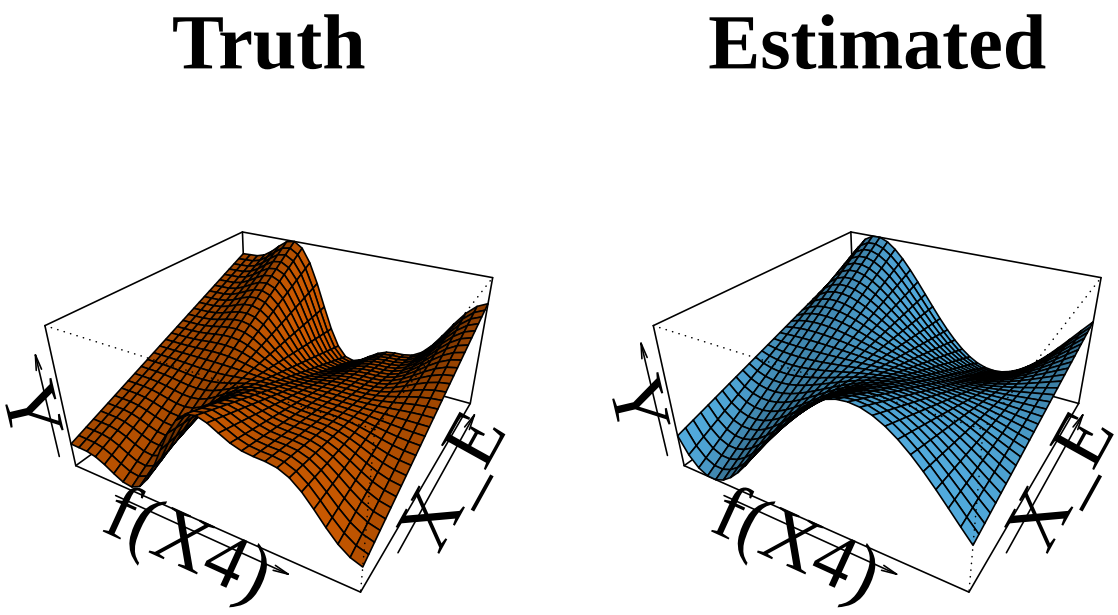
```
plotMain(cvfit$sail.fit, x = sailsim$x, xvar = "X3",
legend.position = "topright",
s = cvfit$lambda.min, f.truth = sailsim$f3)
```



C.4.2 Interaction Effects

Again, since we are using simulated data, we also plot the true interaction:

```
plotInter(cvfit$sail.fit, x = sailsim$x, xvar = "X4",
f.truth = sailsim$f4.inter,
s = cvfit$lambda.min,
title_z = "Estimated")
```



C.5 Linear Interactions

The `basis` argument in the `sail` function is very flexible in that it allows you to apply *any* basis expansion to the columns of $\mathbf{X}$. Of course, there might be situations where you do not expect any non-linear main effects or interactions to be present in your data. You can still use the `sail` method to search for linear main effects and interactions. This can be accomplished by specifying an identity map:

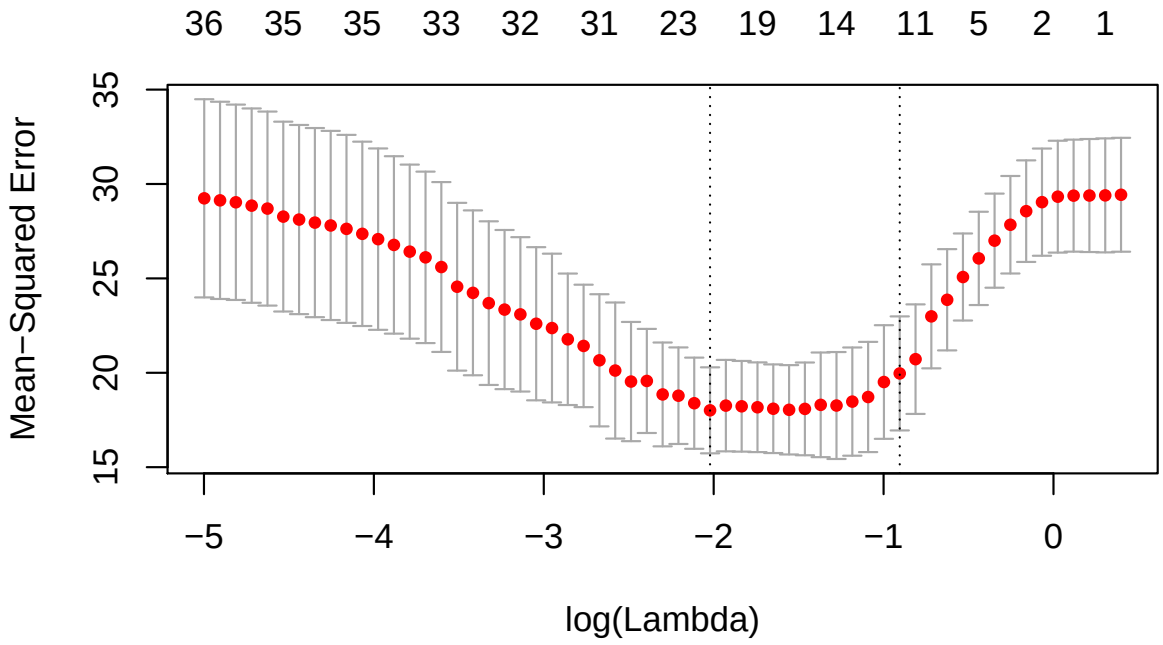
```
f.identity <- function(i) i
```

We then pass this function to the `basis` argument in `cv.sail`:

```
cvfit_linear <- cv.sail(x = sailsim$x, y = sailsim$y, e = sailsim$e,
basis = f.identity, nfolds = 5, parallel = TRUE)
```

Next we plot the cross-validated curve:

```
plot(cvfit_linear)
```



And extract the model at `lambda.min`:

```
predict(cvfit_linear, s = "lambda.min", type = "nonzero")

##                       1
## (Intercept)   5.4385240
## X1_1          2.4568930
## X2_1         -1.6517024
## X3_1         -5.1516226
## X4_1         -7.1356296
## X7_1         -1.1055458
## X8_1         -0.8602620
## X11_1        -2.1990021
## X14_1        -1.9208459
## X16_1         3.2004798
```

```
## X18_1          1.0234994
## X20_1         -0.2161983
## E              2.4466403
## X1_1:E        -4.3995787
## X2_1:E        -2.7115733
## X3_1:E        -4.7198552
## X4_1:E       -12.9777976
## X7_1:E         2.3705634
## X11_1:E       -2.1462985
## X14_1:E       -0.9788318
## X16_1:E        6.9268058
## X18_1:E        1.9005742
```

C.6 Applying a different penalty to each predictor

Recall that we consider the following penalized least squares criterion for this problem:

$$\arg\min_{\boldsymbol{\theta}} \mathcal{L}(Y;\boldsymbol{\theta}) + \lambda(1-\alpha)\left(w_E|\beta_E| + \sum_{j=1}^{p} w_j\|\boldsymbol{\theta}_j\|_2\right) + \lambda\alpha\sum_{j=1}^{p} w_{jE}|\gamma_j| \tag{46}$$

The weights w_E, w_j, w_{jE} are by default set to 1 as specified by the `penalty.factor` argument. This argument allows users to apply separate penalty factors to each coefficient. In particular, any variable with `penalty.factor` equal to zero is not penalized at all. This feature can be applied mainly for two reasons:

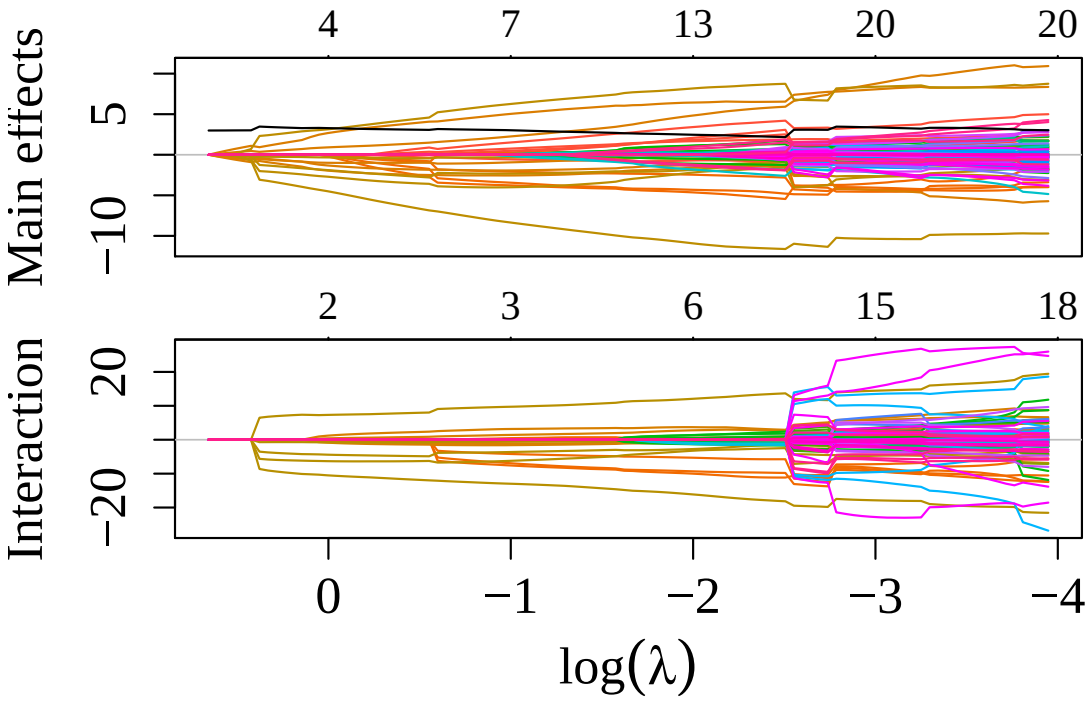
1. Prior knowledge about the importance of certain variables is known. Larger weights will penalize the variable more, while smaller weights will penalize the variable less 2. Allows users to apply the Adaptive `sail`, similar to the Adaptive Lasso

In the following example, we want the environment variable to always be included so we set the first element of `p.fac` to zero. We also want to apply less of a penalty to the main effects for X_2, X_3, X_4:

```
# the weights correspond to E, X1, X2, X3, ... X_p, X1:E, X2:E, ... X_p:E
p.fac <- c(0, 1, 0.4, 0.6, 0.7, rep(1, 2*ncol(sailsim$x) - 4))
```

```
fit_pf <- sail(x = sailsim$x, y = sailsim$y, e = sailsim$e, basis = f.basis,
penalty.factor = p.fac)
```

```
plot(fit_pf)
```



We see from the plot above that the black line (corresponding to the X_E variable with `penalty.factor` equal to zero) is always included in the model.

C.7 User-Defined Design Matrix

A limitation of the `sail` method is that the same basis expansion function $f(\cdot)$ is applied to all columns of the predictor matrix $\mathbf{X}$. Being able to automatically select linear vs. nonlinear components was not a focus of our paper, but is an active area of research for main effects only e.g. `gamsel` and `HierBasis`.

However, if the user has some prior knowledge on possible effect relationships, then they

can supply their own design matrix. This can be useful for example, when one has a combination of categorical (e.g. gender, race) and continuous variables, but would only like to apply $f(\cdot)$ on the continuous variables. We provide an example below to illustrate this functionality.

We use the simulated dataset `sailsim` provided in our package. We first add a categorical variable `race` to the data:

```
set.seed(1234)
library(sail)
x_df <- as.data.frame(sailsim$x)
x_df$race <- factor(sample(1:2, nrow(x_df), replace = TRUE))
table(x_df$race)

##
##  1  2
## 55 45
```

We then use the `model.matrix` function to create the design matrix. Note that the intercept should not be included, as this is computed internally in the `sail` function. This is why we add 0 to the formula. Notice also the flexibility we can have by including different basis expansions to each predictor:

```
library(splines)
x <- stats::model.matrix(~ 0 +  bs(X1, degree = 5) + bs(X2, degree = 3) + ns(X3, df = 8) +
bs(X4, degree = 6) + X5 + poly(X6,2) + race, data = x_df)
head(x)

##   bs(X1, degree = 5)1 bs(X1, degree = 5)2 bs(X1, degree = 5)3
## 1        0.0001654794         0.003945507        0.0470361237
## 2        0.2470181057         0.345144379        0.2411253263
## 3        0.1299195522         0.007832449        0.0002360971
## 4        0.3808392973         0.121815907        0.0194821217
## 5        0.1737663057         0.014898419        0.0006386822
## 6        0.1184145931         0.281407715        0.3343772913
##   bs(X1, degree = 5)4 bs(X1, degree = 5)5 bs(X2, degree = 3)1
## 1        2.803692e-01        6.684809e-01           0.3272340
## 2        8.422768e-02        1.176866e-02           0.3065738
## 3        3.558391e-06        2.145244e-08           0.1896790
## 4        1.557896e-03        4.983113e-05           0.4100900
```

```
## 5        1.368987e-05        1.173746e-07           0.3946500
## 6        1.986587e-01        4.721047e-02           0.3175164
##   bs(X2, degree = 3)2 bs(X2, degree = 3)3 ns(X3, df = 8)1 ns(X3, df = 8)2
## 1          0.41274967         0.173537682      0.06566652               0
## 2          0.04879618         0.002588901      0.00000000               0
## 3          0.01508834         0.000400076      0.00000000               0
## 4          0.12345871         0.012389196      0.00000000               0
## 5          0.35302552         0.105263760      0.00000000               0
## 6          0.05370432         0.003027827      0.00000000               0
##   ns(X3, df = 8)3 ns(X3, df = 8)4 ns(X3, df = 8)5 ns(X3, df = 8)6
## 1     0.000000000    0.000000e+00       0.0000000   -1.589937e-01
## 2     0.000000000    5.775107e-04       0.3179489    5.395130e-01
## 3     0.000000000    4.989926e-03       0.4147696    4.830810e-01
## 4     0.133404268    6.839146e-01       0.1826811    3.022366e-08
## 5     0.000000000    8.944913e-05       0.2775548    5.564842e-01
## 6     0.001578195    3.415384e-01       0.6070588    4.566909e-02
##   ns(X3, df = 8)7 ns(X3, df = 8)8 bs(X4, degree = 6)1 bs(X4, degree = 6)2
## 1    4.436233e-01   -2.846296e-01        0.1820918880        0.3088147022
## 2    1.732713e-01   -3.131078e-02        0.0120101010        0.0000608354
## 3    1.434410e-01   -4.628144e-02        0.0002900763        0.0044075535
## 4    7.673343e-09   -4.923233e-09        0.2978877432        0.0579746877
## 5    1.863219e-01   -2.045032e-02        0.0114895681        0.0645689076
## 6    1.159471e-02   -7.439189e-03        0.0102152807        0.0595722132
##   bs(X4, degree = 6)3 bs(X4, degree = 6)4 bs(X4, degree = 6)5
## 1        2.793213e-01        1.421126e-01        3.856204e-02
## 2        1.643482e-07        2.497444e-10        2.024070e-13
## 3        3.571755e-02        1.628127e-01        3.958163e-01
## 4        6.017595e-03        3.513419e-04        1.094046e-05
## 5        1.935272e-01        3.262743e-01        2.933747e-01
## 6        1.852831e-01        3.241534e-01        3.024572e-01
##   bs(X4, degree = 6)6         X5 poly(X6, 2)1 poly(X6, 2)2 race1 race2
## 1        4.359896e-03 0.51332996  -0.13705545   0.09851639     1     0
## 2        6.835086e-17 0.02643863   0.18835303   0.22584415     0     1
## 3        4.009478e-01 0.76746637  -0.15841216   0.16140597     0     1
## 4        1.419483e-07 0.69077618  -0.03664279  -0.07954100     0     1
## 5        1.099135e-01 0.27718210   0.13128945   0.05620199     0     1
## 6        1.175889e-01 0.48384748   0.08486354  -0.03559388     0     1
```

One benefit of using `stats::model.matrix` is that it returns the group membership as an attribute:

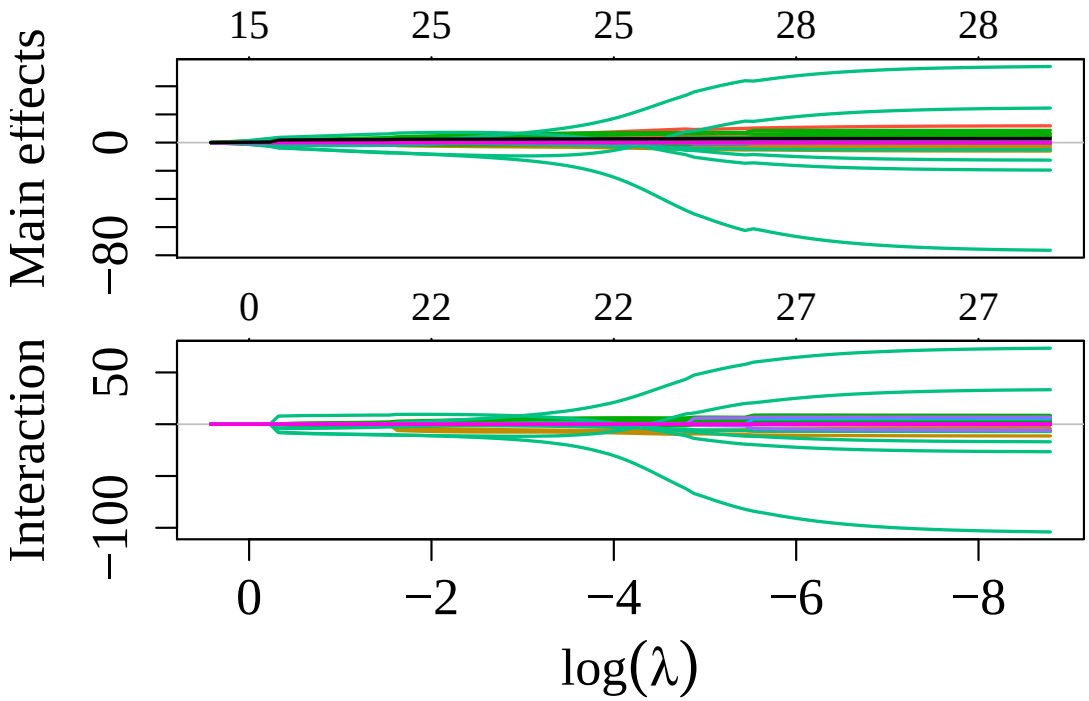
```
attr(x, "assign")

##  [1] 1 1 1 1 1 2 2 2 3 3 3 3 3 3 3 3 4 4 4 4 4 4 5 6 6 7 7
```

The group membership must be supplied to the `sail` function. This information is needed for the group lasso penalty, which will select the whole group as zero or non-zero.

C.7.1 Fit the `sail` Model

We need to set the argument `expand = FALSE` and provide the group membership. The first element of the group membership corresponds to the first column of `x`, the second element to the second column of `x`, and so on.

We can plot the solution path for both main effects and interactions using the `plot` method for objects of class `sail`:

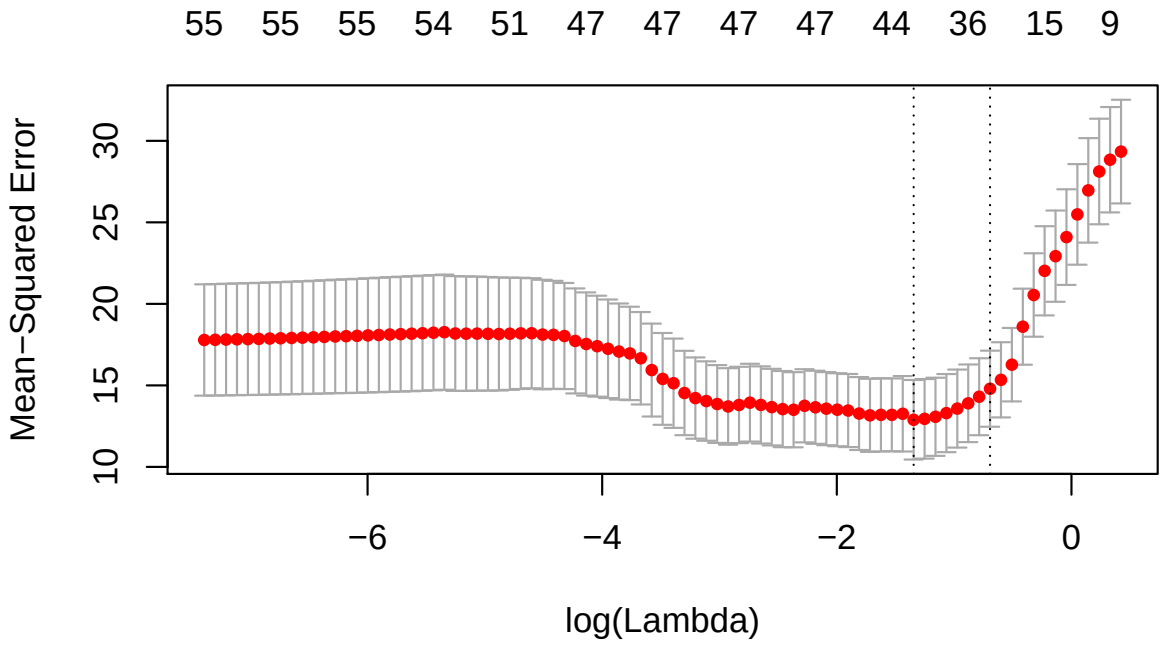


In this instance, since we provided a user-defined design matrix and ‘expand = FALSE‘, the numbers at the top of the plot represent the total number of non-zero coefficients.

C.7.2 Find the Optimal Value for λ

We can use cross-validation to find the optimal value of lambda:

We can plot the cross-validated mean squared error as a function of lambda:



The estimated non-zero coefficients at `lambda.1se`:

```
##                                1
## (Intercept)            5.427451767
## bs(X1, degree = 5)1   -0.525424522
## bs(X1, degree = 5)2    0.052358374
## bs(X1, degree = 5)3    0.271119758
## bs(X1, degree = 5)4    0.554195548
## bs(X1, degree = 5)5    1.330393115
## ns(X3, df = 8)1        2.349455333
## ns(X3, df = 8)2        2.089923982
## ns(X3, df = 8)3        0.666828606
## ns(X3, df = 8)4       -1.200690572
## ns(X3, df = 8)5       -1.662360501
## ns(X3, df = 8)6       -1.365480040
## ns(X3, df = 8)7        0.516186563
```

```
## ns(X3, df = 8)8       -1.215186213
## bs(X4, degree = 6)1    4.466614577
## bs(X4, degree = 6)2   -0.252832683
## bs(X4, degree = 6)3   -4.706674900
## bs(X4, degree = 6)4   -4.868782936
## bs(X4, degree = 6)5   -2.105379737
## bs(X4, degree = 6)6   -0.213392506
## race1                  0.006726548
## race2                 -0.006726548
## E                      1.963105161
## ns(X3, df = 8)1:E      1.170879214
## ns(X3, df = 8)2:E      1.041538657
## ns(X3, df = 8)3:E      0.332322025
## ns(X3, df = 8)4:E     -0.598378532
## ns(X3, df = 8)5:E     -0.828457273
## ns(X3, df = 8)6:E     -0.680503338
## ns(X3, df = 8)7:E      0.257247758
## ns(X3, df = 8)8:E     -0.605602609
## bs(X4, degree = 6)1:E  8.430067510
## bs(X4, degree = 6)2:E -0.477183905
## bs(X4, degree = 6)3:E -8.883145495
## bs(X4, degree = 6)4:E -9.189100187
## bs(X4, degree = 6)5:E -3.973589620
## bs(X4, degree = 6)6:E -0.402746465
```