Bayesian Estimation of Monotone Single Index Models

Snigdha Das

Overview

The package `monotoneSIM` performs Bayesian estimation using B-Spline basis approximation of a Single Index Model where the unknown link function is assumed to be monotonically increasing. Consider a Single index model of the form:

$$y = g(x^\top \beta) + \epsilon$$

where y: response, x: p-dimensional predictors, β: p-dimensional coefficient vector, ϵ: independent N(0, σ^2_ϵ) random errors, g: unknown monotone link function. To construct a prior on g, we consider the basis expansion: $g(x) = \Sigma_{l=0}^{L} \xi_l \, \psi_l(x)$, where $\psi_l(x) = \int_{-1}^{x} h_l(t) \, \mathrm{d}t$, h_l is a B-Spline basis function of order 2. Then, g is monotonically increasing if and only if $\xi_l \geq 0 \; \forall \; l$, thereby enforcing the monotonicity constraint in an equivalent way.

Note: Here $g(x) = 0$, when $x \leq -1$ and $g(x) = g(1)$, when $x \geq 1$. The matrix of predictors is scaled so that every row has euclidean norm ≤ 1 and euclidean norm of β is taken to be 1 so that $|x^\top \beta| \leq 1$.

Functionality

The package provides two functions.

- The main function `monotoneSIM` performs a Markov Chain Monte Carlo algorithm to generate samples from the conditional posterior distribution of unknown parameters in a monotone Single Index Model - the unknown parameters being B-Spline basis coefficients that approximate the unknown link function, the single index parameter and the error variance of the model.
- The other function `monotoneFIT` calculates fitted values of the response using posterior mean of the single index parameter after generating an MCMC sample using the function `monotoneSIM`. It also returns the estimated link function of the model based on basis coefficients for a grid of supplied values.

Installation

To install this package from Github, run the following in your R console:

```
# install.packages("devtools")
devtools::install_github("das-snigdha/monotoneSIM")
```

An Example Demonstrating Usage

This package shall be implemented on the following simulated data.

```
n = 100; p = 3; L = 20

# We take 2 continuous variables and 1 dichotomous attribute as predictors.
X = matrix(rnorm(n*(p-1)), nrow = n, ncol = (p-1))
X = cbind(X, rbinom(n, 1, 0.5))

# True Value of the parameter (having unit euclidean norm).
true.beta = rnorm(p); true.beta = true.beta/ norm(true.beta, "2")

# True monotone increasing link function
true.g = function(x){
y = (x+1)/2
5*(pnorm(y, mean=0.5, sd=0.1) - pnorm(0, mean = 0.5, sd = 0.1))
}

# Generate the response
y.true = true.g(X%*%true.beta) + rnorm(n, 0, sqrt(0.01))
```

Choose values of hyperparameters and starting values of parameters to apply `monotoneSIM` as follows:

```
beta.start = rnorm(p)  #Starting value of beta
xi = abs(rnorm((L+1), 0, 5))   #Starting value of xi
S_xi = 5*diag(L+1) #Prior Variance of xi
sigma.sq.eps.start = 0.01 #Starting value of sigma.sq.eps

library(monotoneSIM)
MCMC.sample = monotoneSIM(y = y.true, X = X, beta.init = beta.start,
                          xi.init = xi, Sigma.xi =  S_xi, monotone = TRUE,
                          sigma.sq.eps.init = sigma.sq.eps.start,
                          Burn.in = 100, M = 500)
```

Using the MCMC sample generated by `monotoneSIM`, we may draw several inferences on the unknown components of the model. For example, the single index parameter β can be estimated using the posterior mean of the MCMC sample on β, as shown below:

```
#Posterior sample mean of beta
beta.estimated = colMeans(MCMC.sample$beta);
beta.estimated ; true.beta  # Compare estimated beta with true beta
```

```
## [1] -0.95439267  0.29643986  0.03109262
```

```
## [1] -0.95885929  0.28267429  0.02615528
```

```
#Posterior Standard Deviation of beta
sd.beta = apply(MCMC.sample$beta, 2, sd); sd.beta
```

```
## [1] 0.002478689 0.007916307 0.014935644
```

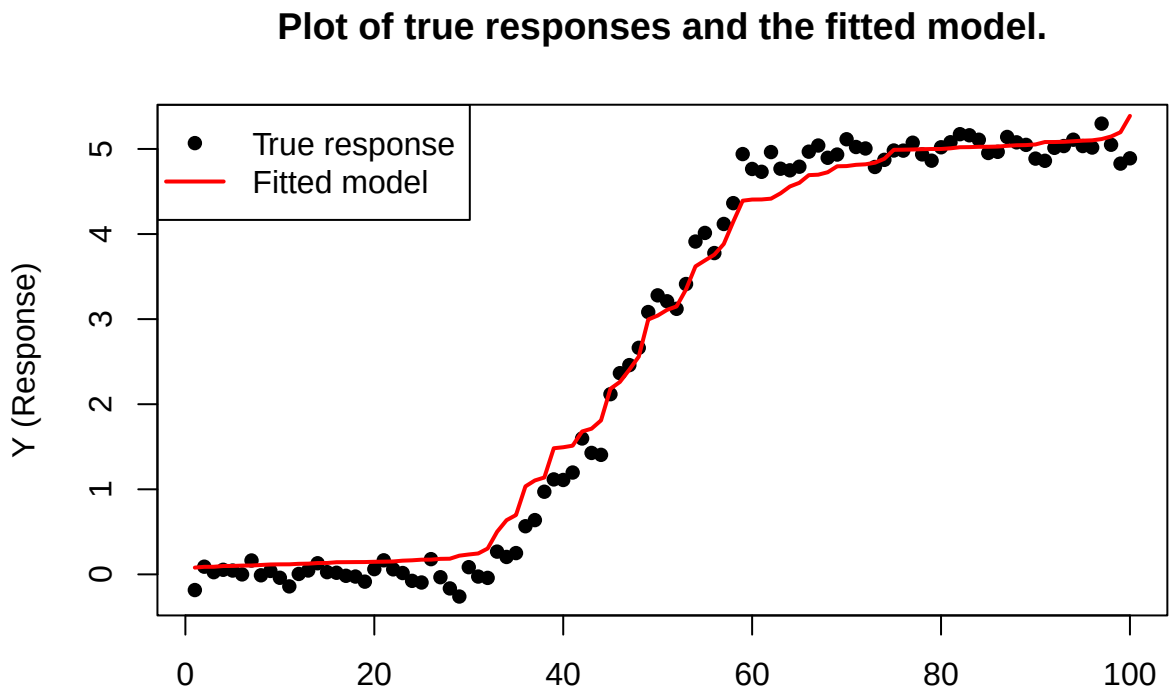
Once the MCMC sample has been obtained, the function `monotoneFIT` is used to extract fitted values of the response and to estimate the link function for a grid of supplied values.

```
fit = monotoneFIT(MCMC.sample, size.grid.x = 200)

# Obtain the fitted response
y.fit = colMeans(fit$Y.fitted)
```

Having obtained the fitted responses, the Mean Sqaured Error may be calculated to comment on the goodness of fit or one may graphically visualize the true response and the fitted model as shown below:

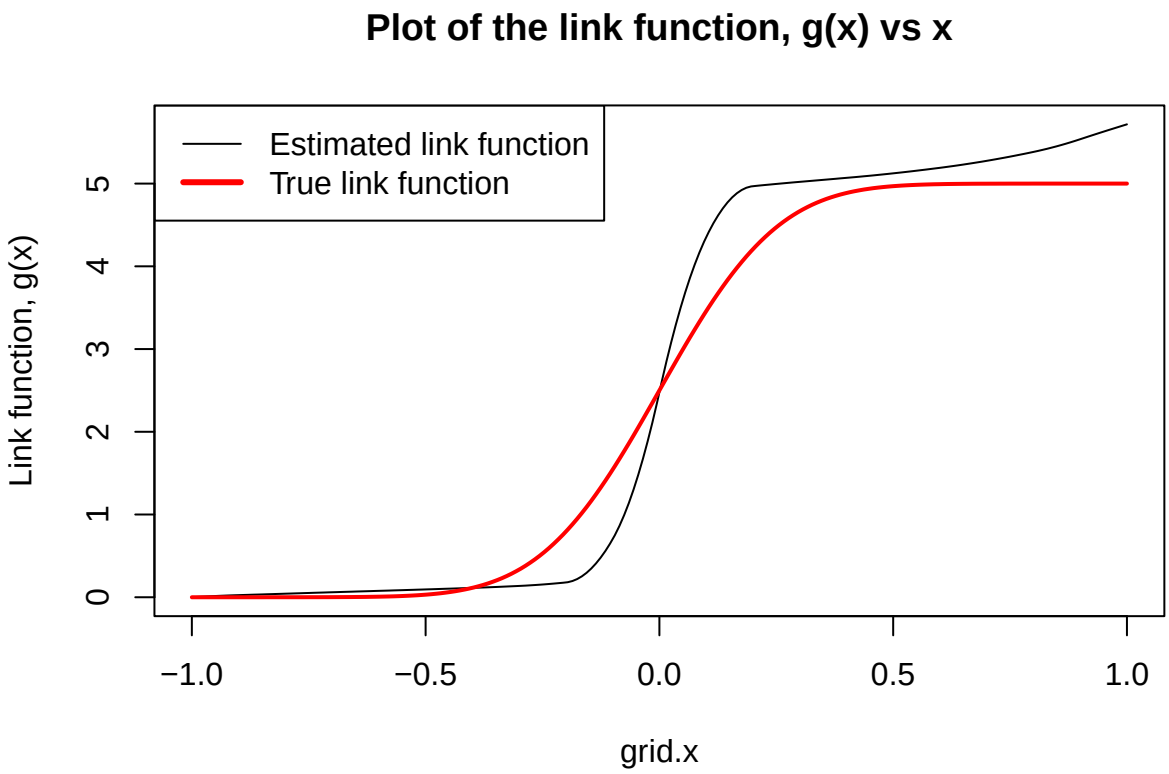
```
## [1] "MSE =  0.0512254663041063"
```



Finally, an estimate of the link function is obtained as :

```
# Obtain the estimated value of the link function g(x)
est.func = colMeans(fit$link.estimated)
```

One can graphically look at the functional form of the the estimated link function by plotting it against `grid.x`


Plot of the link function, g(x) vs x
Estimated link function
True link function
Link function, g(x)
0
1
2
3
4
5
−1.0
−0.5
0.0
0.5
1.0
grid.x