A Comparison of Two-Stage Variable Selection Methods for Ultra-High Dimensional Data

Abstract: Ultra-high dimensional data has been a source for modern problems in statistics due to the rapid growth in technology. Advancements in computers and electronics have increased the detail and speed at which data are collected. Ultra-high dimensional data appear in diverse scientific fields such as high frequency trading, genomics, and brain imaging. Much research has already been focused on variable selection for these fields. In this paper, we conduct a simulation experiment to compare three two-stage variable screening methods to evaluate their screening ability as well as their final variable selection performance.

1 Introduction

1.1 Background

Ultra-high dimensional data, due to rapid developments in computers and technology, is a growing problem area of statistical analysis. Advancements in electronics have made collection of data faster and easier [Donoho 2000]. Diverse scientific fields, such as high frequency trading and genomics, depend on these data [Fan and Lv 2009]. See [Fan and Li 2006] for detailed examples of high dimensional data. Working in such a large dimension, difficulties in statistical analyses are amplified. In such problems, it is likely to find high collinearity among the predictors even when they are completely unrelated. Even with penalization, estimation of model parameters is unstable and unreliable. Implementing classical techniques for variable selection, such as the lasso [Tibshirani 1996], SCAD [Fan and Li 2001] or elastic net [Zou and Hastie 2005], is unreasonable due to large computational costs. The focus in ultra-high dimensional data analysis is development of fast and crude methods for eliminating unimportant variables in the data [Fan and Lv 2009]. These techniques are known as screening methods.

When approaching an ultra-high dimensional problem, most assume the parameter space is sparse. With this assumption, the goal is to efficiently eliminate as many unimportant predictors as possible with a screening algorithm. Post screening, the dimension of the data is greatly reduced and classical methods for variable selection and modeling are appropriate and can provide sensible results. This combination of a screening method with a variable selection technique is known as a two-stage variable selection [Fan and Lv 2009]. It is important to note that the performance of the selection at the second stage is highly dependent on the quality of the screening method used.

To gauge the performance of some two-stage algorithms, we will use a Monte-Carlo type simulation experiment to analyze two-scale methods both post screening and at the final stage. Section 2 will define notation and describe the three algorithms we will be comparing. Details of the simulation study are outlined in section 3 with presentation of the results in section 4. Finally, in section 5, we will discuss the results and offer some conclusions.

2 Two-Stage Algorithms

First, standard notation and some assumptions about the data will be established to be used throughout. Notation may also be added as necessary in subsequent sections.

Let Y denote the $n \times 1$ vector of responses and Y_i be a reference to the i^{th} element. The matrix of covariate information will be represented by X, an $n \times p$ matrix. Let $X_{i,j}$ be a scalar denoting the element in the i^{th} row and j^{th} column of X. The i^{th} row will be indicated by $X_{i,\cdot}$ and similarly $X_{\cdot,j}$ is the j^{th} column. With a subset of predictors $\mathcal{S} \subset \{1, \ldots, p\}$, the matrix constructed from the columns $X_{\cdot,j}$ for $j \in \mathcal{S}$ is denoted by $X_{\cdot,\mathcal{S}}$. The pairs $(Y_i, X_{i,\cdot})_{i=1}^{n}$ are independent and identically distributed. Also, the design matrix X is assumed to be centered and scaled meaning the sample standard deviation of each $X_{\cdot,j}$ is 1 and the sample mean of each $X_{\cdot,j}$ is 0.

The set of all predictors is $\mathcal{F} = \{1, \ldots, p\}$ and the true set of important variables is $\mathcal{M}$ with $s = |\mathcal{M}|$. For a linear model $Y = X\beta + \epsilon$, the true set of predictors would be $\mathcal{M} = \{j : \beta_j \neq 0\}$.

To rank the variables in a screening method, let ω denote the vector of rankings or utilities. These may wear additional accents to indicate estimations.

The operator $\lfloor \cdot \rfloor$ returns the integer part of a real valued number.

2.1 Sure Independence Screening with the Lasso

Sure Independence Screening (SIS) was developed by Fan and Lv [2008]. SIS is based on correlation learning focusing on the sample correlation between the response and a given predictor. SIS is a simple intuitive screening algorithm with low computational cost. SIS will be used as the screening method and the lasso will be used for the second stage variable selection.

SIS ranks based on the magnitude of the sample correlation between a given predictor and the response. With the assumption that $X_{\cdot,j}$ has sample mean 0 and sample variance 1 for all j, the sample correlation between the j^{th} predictor and the response is equivalent to $X_{\cdot,j}^{\intercal}Y$ scaled by the sample variance of Y. Since the missing scalar does not depend on j, this provides an equivalent ranking. Taking the absolute value of these inner products we obtain the rankings $\omega_j = |X_{\cdot,j}^{\intercal}Y|, \ 1 \leq j \leq p$. Sort the rankings in descending order and choose the predictors corresponding to the largest $d = \lfloor n/\log(n) \rfloor$ rankings as the set of screened variables. With this set of screened predictors use the lasso for the second stage variable selection.

2.2 Model-Free Screening Procedure with the Lasso

Making a model assumption about the data is a common approach for variable selection and screening. Zhu et al. [2011] propose a screening procedure which is essentially model-free (MFP). The method uses empirical estimates for conditional densities of the response given the predictors. MFP will be used as a first stage screening algorithm coupled with the lasso as the second stage variable selection method.

Define $\Omega(y) = \mathbb{E}[X_{1,\cdot}F(Y_1|X_{1,\cdot})] = \text{Cov}(X_{1,\cdot}, \mathbb{1}_{\{Y_1<y\}})$. Let $\Omega_j(y)$ be the j^{th} element of the vector $\Omega(y)$. Define $\omega_j = \mathbb{E}[\Omega_j^2(Y_1)]$ as the proposed value to be used for ranking predictors. Estimate these rankings using the empirical expectation

$$\tilde{\omega}_j = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{n}\sum_{i'=1}^{n} X_{i',j}\mathbb{1}_{\{Y_{i'}<Y_i\}}\right]^2. \tag{1}$$

For screening, rank the predictors in descending order according to $\tilde{\omega}$ and select the first $d = \lfloor n/\log(n)\rfloor$ predictors.

2.3 Nonparametric Independence Screening with Lasso

Some independent screening methods assume that a linear model is correct, while in practice, there is not always evidence to support these assumptions. It is of interest to allow for these assumptions to be broken when screening for important predictors. Since SIS uses marginal correlation between a predictor and the response, this is comparable to fitting a simple linear model. To add flexibility, Fan et al. [2011] expand on this idea by using a nonparametric model which they call Nonparametric Independence Screening (NIS). NIS will be used as the first stage screening algorithm followed by the lasso for the second stage variable selection.

In NIS the problem of identifying important predictors is to solve the marginal regressions obtaining $g_j = \arg\min_{f\in L_2(P)}\ \mathbb{E}[Y_1 - f(X_{1,j})]^2$ for each j where P is the joint distribution of $X_{1,\cdot}$ and Y_1 and $L_2(P)$ is the class of square-integrable functions under P. We will use a B-spline polynomial basis for estimation of g_j as suggested by Fan et al. [2011]. Assume g_j to have the form $g_j(x) = \sum_{\ell=1}^{q} \beta_\ell \phi_{j,\ell}(x)$ where β_ℓ are estimated using least squares and $\phi_{j,\ell}(x)$ is the value of the ℓ^{th} spline basis function for predictor j evaluated at x. Now, rank the variables in decreasing order by $\hat{\omega}_j = \frac{1}{n}\sum_{i=1}^{n}[\hat{g}_j(X_{i,j})]^2$ and select the top $d = \lfloor n/\log(n) \rfloor$ variables. With the d variables left use the lasso for the second-stage variable selection.

3 Simulation Study

The goal of the simulation study is to compare the quality of variable selection for the three two-stage screening methods above. We will look at results from the first screening stage and after the second stage variable selection to critique the algorithm in its entirety.

We will generate pairs $(Y_i, X_{i,\cdot})_{i=1}^n$ from two different models. The first is from a linear model $Y = X\beta + \epsilon$ where $\epsilon \sim \mathcal{N}_n(0, \Sigma_\epsilon)$. The second is a non-linear model $Y_i = \sum_j g_{k_j}(X_{i,j}) + \epsilon_i$ where ϵ is the same as in the linear model. The individual functions are

$$\begin{array}{lll} g_1(x) = x & g_2(x) = \log(|\frac{x}{4}| + 10^{-3}) & g_3(x) = \sin(\frac{\pi x}{4}) + \cos(\frac{\pi x}{4})^2 \\ g_4(x) = \frac{\cos(\pi x/4)}{2+\cos(\pi x/4)^2} & g_5(x) = \cos(\frac{\pi x}{4})\text{expit}(\frac{x}{4})^2 & g_6(x) = \cos(\frac{\pi x}{4})(1 - \text{expit}(\frac{x}{4}))^2 \\ g_7(x) = \cos(\frac{\pi x}{4})(\frac{|x/4|}{1+|x/4|})^2 & g_8(x) = \frac{\cos(\pi x/4)}{(1+|x/4|)^2} & \end{array} \tag{2}$$

and $k_j = [(j-1) \mod 8] + 1$. Plots of each g_k can be found in the appendix. The dimension p is a function of n of the form $p = \lfloor \exp(n^\alpha) \rfloor$. We explore two values for α to gauge performance as dimension and sample size vary. The sparsity assumption will hold true with $s \ll p$, thus the vast majority of the β_j will be zero. For simplicity, we let $\beta_1, \ldots, \beta_s$ be the non-zero coefficients. To generate these values, we set $\beta_j = (-1)^{v_j} u_j$, where $v_1, \ldots, v_s \overset{iid}{\sim} Binomial(1, .25)$ and $u_1, \ldots, u_s \overset{iid}{\sim} \chi^2_2$ where all u and v are pairwise independent [Fan and Lv 2008]. We fix the random seed for β for all data sets. We generate X as independent standard normal random variables, so $X_{i,j} \overset{iid}{\sim} \mathcal{N}(0, 1)$ for all i and j.

In the simulation, we will use a full factorial experiment comparing multiple different settings. The explored values are $(n, \alpha) \in \{(100, .35), (200, .35), (200, .375)\}$, $s \in \{10, 15, 25\}$, and $\Sigma_\epsilon \in \{25.0I, 100.0I, H(X)\}$. For the error variance, we look to see how the magnitude of the noise affects performance. The final value of Σ_ϵ is a function of the data to violate the constant variance assumption [Wang et al. 2012]. Here, we are comparing the effect of heteroscedasticity. The final variance matrix $H(X)$ is a diagonal matrix where

$$H_{i,i}(X) = 100.0 \left[.05 + .95 \frac{f(X_{i,\cdot}) - \min_{i'} f(X_{i',\cdot})}{\max_{i'} f(X_{i',\cdot}) - \min_{i'} f(X_{i',\cdot})} \right]^2 . \tag{3}$$

and $f(X_{i,\cdot}) = \|X_{i,\cdot}\|_2^2 / (1 + \|X_{i,\cdot}\|_2^2)$. The purpose of $H_{i,i}$ is to spread out the values between 0.25 and 100.0 keeping the sequential ordered differences of $f(X_{i,\cdot})$ proportional.

We compare all 27 combinations of n, α, s and Σ_ϵ using a Monte Carlo type study with 200 replications. After the screening process, we record the number of variables needed to keep in order to retain *all* relevant predictors. Then, selecting the d predictors with the highest rankings, we create a set of screened variables. As a reminder, we set $d = \lfloor n / \log(n) \rfloor$ for all three methods. With the set of screened variables, we use the lasso for variable selection using a 10-fold cross validation to determine the optimal penalty size. Finally, we record how many variables left are relevant and the total number of variables remaining.

4 Numerical Results

The results are divided into three tables. In tables 1 and 2, for a linear and non-linear model respectively, we display the mean number of variables needed to keep in order to retain all important variables along with the estimated standard deviation of the mean in parentheses. Table 3 displays the mean number of relevant predictors kept after the two-stage variable selection as well as the mean total number of variables left. Table 3 does not display standard errors since the largest standard error for any estimate in this table is less than 0.75.

5 Conclusion

For the first stage screening with a linear model, SIS and MFP perform better than NIS as expected. With $\alpha = .35$ SIS does as well as or better than MFP, but for $\alpha = .375$ MFP outperforms SIS. This trend tends to be consistent for varying levels of s and Σ_ϵ. Under the non-linear model, the clear winner is NIS since the other two methods assume a linear relationship between X and Y. The performance difference becomes more noticeable for larger s and an increase in α.

After the two-stage variable selection we see a much greater separation between the three methods. When comparing SIS and MFP, the number of true positives is similar for most settings. However, MFP tends to have lower false positives since the total number of variables kept tends to be lower than with SIS. In comparison of true positives, NIS performs marginally worse than SIS and MFP, but NIS has dramatically less false positives. The number of variables kept is much lower for NIS compared to SIS and MFP.

Depending on the ultimate goal of the analysis, there is not a clear winner among the three methods. While NIS fails to identify as many relevant predictors as the other two methods, with the more parsimonious model perhaps the tendency to overfit is much lower. An interesting future comparison would be to analyze the fit of the model with the final set of predictors. These are suggested topics of investigation for a future simulation study.

n	s	Σ_ϵ	SIS	MFP	NIS
100 [.35]	10	$25.0I$	125.735 (1.59)	125.415 (1.6)	125.93 (1.434)
		$100.0I$	129.73 (1.258)	130.775 (1.239)	130.385 (1.217)
		$H(X)$	126.9 (1.491)	127.87 (1.428)	127.345 (1.373)
	15	$25.0I$	128.6 (1.416)	130.705 (1.315)	129.995 (1.187)
		$100.0I$	134.57 (0.988)	135.875 (0.824)	135.49 (0.859)
		$H(X)$	131.05 (1.274)	132.155 (1.149)	131.83 (1.119)
	25	$25.0I$	141.29 (0.566)	142.405 (0.516)	142.465 (0.549)
		$100.0I$	142.52 (0.501)	142.25 (0.52)	143.455 (0.51)
		$H(X)$	142.275 (0.554)	142.375 (0.547)	142.81 (0.507)
200 [.35]	10	$25.0I$	465.94 (6.987)	461.785 (7.182)	486.805 (6.737)
		$100.0I$	494.15 (6.309)	485.24 (6.144)	511.0 (5.696)
		$H(X)$	474.83 (6.996)	471.585 (7.088)	491.71 (6.786)
	15	$25.0I$	501.715 (5.883)	489.82 (6.731)	499.525 (5.92)
		$100.0I$	507.455 (5.491)	512.51 (5.366)	519.23 (4.969)
		$H(X)$	502.17 (6.017)	490.235 (6.661)	499.24 (5.701)
	25	$25.0I$	546.76 (3.337)	549.075 (2.675)	549.1 (3.091)
		$100.0I$	550.31 (2.957)	556.435 (2.474)	549.74 (2.844)
		$H(X)$	549.9 (3.165)	552.135 (2.682)	549.355 (2.95)
200 [.375]	10	$25.0I$	1150.11 (17.304)	1139.295 (17.887)	1202.22 (16.847)
		$100.0I$	1219.225 (15.559)	1198.45 (15.155)	1262.435 (14.12)
		$H(X)$	1170.84 (17.412)	1174.32 (18.088)	1214.885 (16.458)
	15	$25.0I$	1237.255 (14.749)	1206.92 (16.967)	1233.365 (14.841)
		$100.0I$	1251.54 (13.817)	1264.53 (13.536)	1281.62 (12.346)
		$H(X)$	1224.96 (15.598)	1204.37 (16.718)	1231.71 (14.201)
	25	$25.0I$	1348.39 (8.335)	1355.125 (6.715)	1355.505 (7.877)
		$100.0I$	1357.615 (7.427)	1373.81 (6.273)	1357.49 (7.169)
		$H(X)$	1349.585 (8.327)	1362.52 (6.618)	1362.585 (7.557)

Table 1: Results after the first stage screening. For each method, the number of variables needed to keep in order to retain all relevant predictors are displayed with the associated standard error estimate. Data are generated under a linear model.

n	s	Σ_ϵ	SIS	MFP	NIS
100 [.35]	10	$25.0I$	132.96 (1.034)	134.475 (0.954)	130.63 (1.242)
		$100.0I$	133.285 (1.022)	135.67 (0.932)	132.91 (1.147)
		$H(X)$	133.275 (1.062)	134.98 (0.925)	130.235 (1.186)
	15	$25.0I$	139.45 (0.644)	140.02 (0.67)	139.1 (0.778)
		$100.0I$	139.875 (0.694)	140.05 (0.659)	139.655 (0.674)
		$H(X)$	139.73 (0.704)	139.81 (0.7)	139.35 (0.696)
	25	$25.0I$	144.16 (0.401)	144.425 (0.388)	144.15 (0.391)
		$100.0I$	144.46 (0.412)	145.32 (0.347)	144.085 (0.386)
		$H(X)$	144.34 (0.442)	145.035 (0.383)	144.415 (0.37)
200 [.35]	10	$25.0I$	529.97 (4.043)	527.23 (4.486)	515.125 (5.191)
		$100.0I$	531.79 (3.906)	533.15 (3.965)	519.91 (4.96)
		$H(X)$	530.305 (4.213)	528.995 (4.182)	515.47 (5.173)
	15	$25.0I$	547.505 (2.81)	548.88 (2.735)	544.785 (3.573)
		$100.0I$	551.695 (2.659)	549.655 (3.066)	550.115 (2.916)
		$H(X)$	544.155 (3.243)	553.42 (2.707)	546.455 (3.506)
	25	$25.0I$	562.66 (2.074)	565.85 (1.917)	567.125 (1.821)
		$100.0I$	565.945 (2.025)	568.88 (1.54)	570.17 (1.446)
		$H(X)$	567.055 (1.871)	566.945 (1.694)	566.985 (1.836)
200 [.375]	10	$25.0I$	1309.035 (10.061)	1301.14 (11.277)	1272.73 (12.966)
		$100.0I$	1312.87 (9.696)	1316.585 (9.887)	1285.15 (12.474)
		$H(X)$	1298.485 (11.336)	1318.04 (10.363)	1266.825 (12.119)
	15	$25.0I$	1351.775 (7.091)	1355.54 (6.86)	1345.925 (9.051)
		$100.0I$	1362.435 (6.732)	1357.555 (7.647)	1359.775 (7.304)
		$H(X)$	1339.685 (7.678)	1358.64 (6.946)	1346.815 (9.246)
	25	$25.0I$	1389.085 (5.195)	1397.535 (4.853)	1401.71 (4.577)
		$100.0I$	1396.82 (5.153)	1404.535 (3.978)	1409.315 (3.661)
		$H(X)$	1398.32 (4.109)	1402.27 (4.602)	1405.81 (4.115)

Table 2: Results after the first stage screening. For each method, the number of variables needed to keep in order to retain all relevant predictors are displayed with the associated standard error estimate. Data are generated under a nonlinear model.

$n[\alpha]$	s	Σ_ϵ	SIS		MFP		NIS	
100 [.35]	10	$25.0I$	4.905	15.785	4.73	13.545	4.02	8.62
		$100.0I$	3.35	17.07	2.695	12.91	1.345	4.655
		$H(X)$	4.615	16.48	4.335	12.96	3.265	7.38
	15	$25.0I$	7.825	17.47	7.495	15.67	6.29	11.605
		$100.0I$	5.59	17.885	4.94	14.92	3.31	8.17
		$H(X)$	7.26	17.815	6.94	15.695	5.545	11.065
	25	$25.0I$	8.44	15.31	7.905	13.925	6.135	9.99
		$100.0I$	6.895	16.095	6.22	13.71	4.5	8.405
		$H(X)$	8.11	15.54	7.65	14.175	5.815	9.875
200 [.35]	10	$25.0I$	5.23	30.275	5.18	26.08	4.64	15.58
		$100.0I$	4.19	34.83	3.965	31.29	3.095	16.155
		$H(X)$	5.11	32.315	5.06	25.675	4.47	14.48
	15	$25.0I$	9.0	29.29	8.74	26.525	7.91	18.185
		$100.0I$	7.1	34.685	6.655	31.3	5.4	18.12
		$H(X)$	8.75	31.295	8.51	27.075	7.595	18.03
	25	$25.0I$	9.95	26.23	9.54	23.8	7.93	16.0
		$100.0I$	8.24	31.795	7.61	27.98	5.925	16.52
		$H(X)$	9.715	27.55	9.12	24.005	7.59	16.005
200 [.375]	10	$25.0I$	4.825	32.695	4.73	29.115	4.29	16.835
		$100.0I$	3.465	35.635	3.27	32.72	2.45	17.86
		$H(X)$	4.66	33.48	4.6	28.99	4.095	16.68
	15	$25.0I$	8.26	32.14	7.885	29.315	7.085	19.57
		$100.0I$	5.89	35.575	5.375	32.83	4.04	19.42
		$H(X)$	7.99	33.085	7.705	29.725	6.685	19.155
	25	$25.0I$	8.415	28.665	7.715	25.12	6.35	16.255
		$100.0I$	6.465	33.625	5.87	30.34	4.6	16.915
		$H(X)$	8.115	29.665	7.525	25.725	6.07	16.305

Table 3: Results after the two-stage variable selection. For each method, the number of true positives is on the left and the number of total variables kept is on the right. Data are generated under a linear model.

6 Appendix

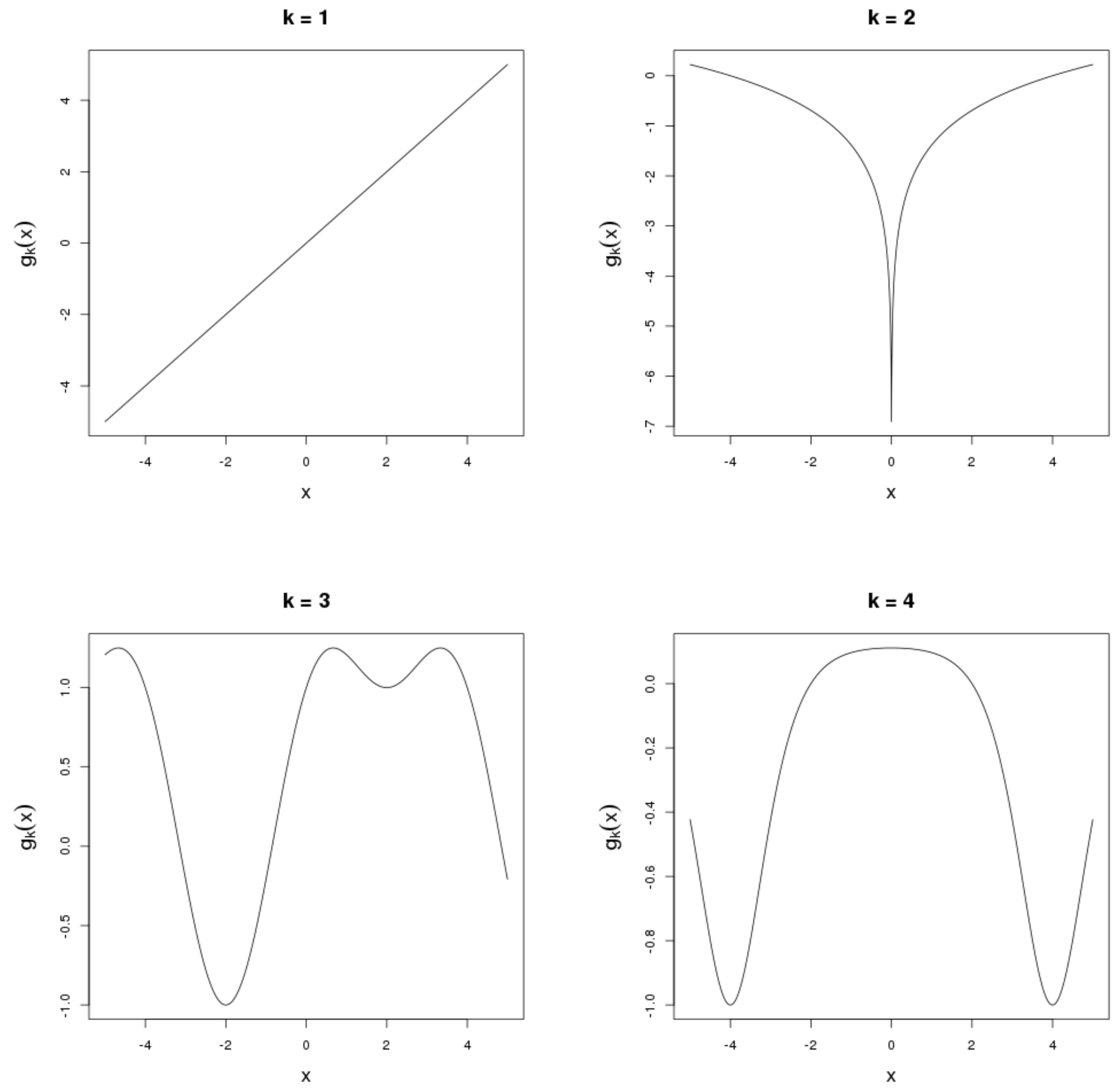
The code used to run the simulation is below. The code is broken into six sections corresponding to six separate files. Some of the code produces results not displayed in the paper such as a cauchy distribution for generating the design matrix.

6.1 methods.R

This file houses the three variable selection methods, functions used to generate data, and some miscellaneous functions. For convenience, after the code the 8 functions used for the non-linear model have been plotted.

```
library(splines)
library(foreach)
library(doMC)
library(boot)



## g? are functions used for the nonlinear model
g1=function(x)
    return(x)
g2=function(x)
    return(log(abs(x/4)+1e-3))
g3=function(x)
    return(sin(pi*x/4)+cos(pi*x/4)^2)
g4=function(x)
    return(cos(pi*x/4)/(2+cos(pi*x/4))^2)
g5=function(x)
    return(cos(pi*x/4)*inv.logit(x/4)^2)
g6=function(x)
    return(cos(pi*x/4)*(1-inv.logit(x/4))^2)
g7=function(x)
    return(cos(pi*x/4)*(abs(x/4)/(1+abs(x/4)))^2)
g8=function(x)
    return(cos(pi*x/4)/((1+abs(x/4)))^2)

if(FALSE){

    x = seq(-5,5,.0001)
    for(i in 1:maxG){
        main = paste("k","=",i)
        xlab = "x"
        ylab = expression(g[k](x))
        png(paste("g",i,".png",sep=""))
        par(mar=c(6,5,4,2))
        plot(x,g(i,x),xlab=xlab,ylab=ylab,main=main,type="l",cex.lab=1.5,cex.main=1.5)
        dev.off()
    }

}



## obtain the maximum function defined
maxG=length(grep("g[0-9]+",ls(),perl=TRUE))



## @fn define a general g that will use one of the above g? functions.
## It rotates through them in order.
## @param (num) the index of the predictor variable (decides which g to use)
## @param (x) the value of the predictor varible
g=function(num,x){
    fn=get(paste("g",(num-1)%%maxG+1,sep=""))
    return(fn(x))
}



## @fn make the design matrix
## @param (dist) the distribution to generate the predictors with
## @param (n) the number of observations
## @param (p) the dimension of the data
## @param (seed) the random seed
makeX=function(dist,n,p,seed=0){
    set.seed(seed)
    if(dist==1){## normal
        return(matrix(rnorm(n*p),nrow=n,ncol=p))
    }
    else if(dist==2){## cauchy
        return(matrix(rcauchy(n*p),nrow=n,ncol=p))
    }
    else{
        stop("invalid *dist* in makeX")
    }
}



## @fn make the error vector
## @param (epsSd) the index of the error variance type
## @param (n) the number of observations
## @param (X) the design matrix
## @param (seed) one minus the random seed
makeEps=function(epsSd,n,X,seed=0){
    set.seed(seed+1) ## add one so its not the same as in makeX()

    if(epsSd==1){## sigma=5.0
        return(rnorm(n,sd=5.0))
    }
    else if(epsSd==2){## sigma=10.0
        return(rnorm(n,sd=10.0))
    }
    else if(epsSd==3){## sigma=10.0*||X_i||/(1+||X_i||)
        sd=sqrt(rowSums(X^2))
        sd=sd/(1+sd)
        sd=sd-min(sd)
        sd=sd*.95/max(sd)
        sd=sd+.05
        sd=10.0*sd
        return(rnorm(n,sd=sd))
    }
    else{
        stop("invalid *sd* in makeEps")
    }
}



## @fn make X and Y
## @param (mod) the index of the model to use to make Y
## @param (dist) the index of the distribution to make X
## @param (epsSd) the index of the error variance type
## @param (n) the number of observations
## @param (p) the dimension of the data
## @param (beta) the coefficients used to make Y
## @param (seed) determines the random seed for making X and eps
makeData=function(mod,dist,epsSd,n,p,beta,seed=0){
    X=makeX(dist,n,p,seed)
    eps=makeEps(epsSd,n,X,seed)

    if(mod==1){## linear
        Y=X%*%beta + eps
    }
    else if(mod==2){## non linear
        V=foreach(j=1:p,.combine=cbind)%do%{
            return(g(j,X[,j]))
        }
        Y=V%*%beta + eps
    }
    else{
        stop("invalid *mod* in makeX")
    }
    return(list(Y=Y,X=X))
}



## @fn calculate the weights according to the SIS algorithm
## @param (Y) the response
## @param (X) the predictors
SIS=function(Y,X){
    p=ncol(X)
    omega=abs(t(X)%*%Y)
    return(omega)
}



## @fn calculate the weights according to the "Model Free Procedure"
## @param (Y) the response
## @param (X) the predictors
MFP=function(Y,X){
    n=nrow(X)
    p=ncol(X)
    omega=foreach(k=1:p,.combine=c)%do%{
        res0=foreach(j=1:n,.combine=c)%do%{
            res1=X[,k]*(Y<Y[j])
            return(mean(res1)^2)
        }
        return(mean(res0))
    }
    return(omega)
}
```

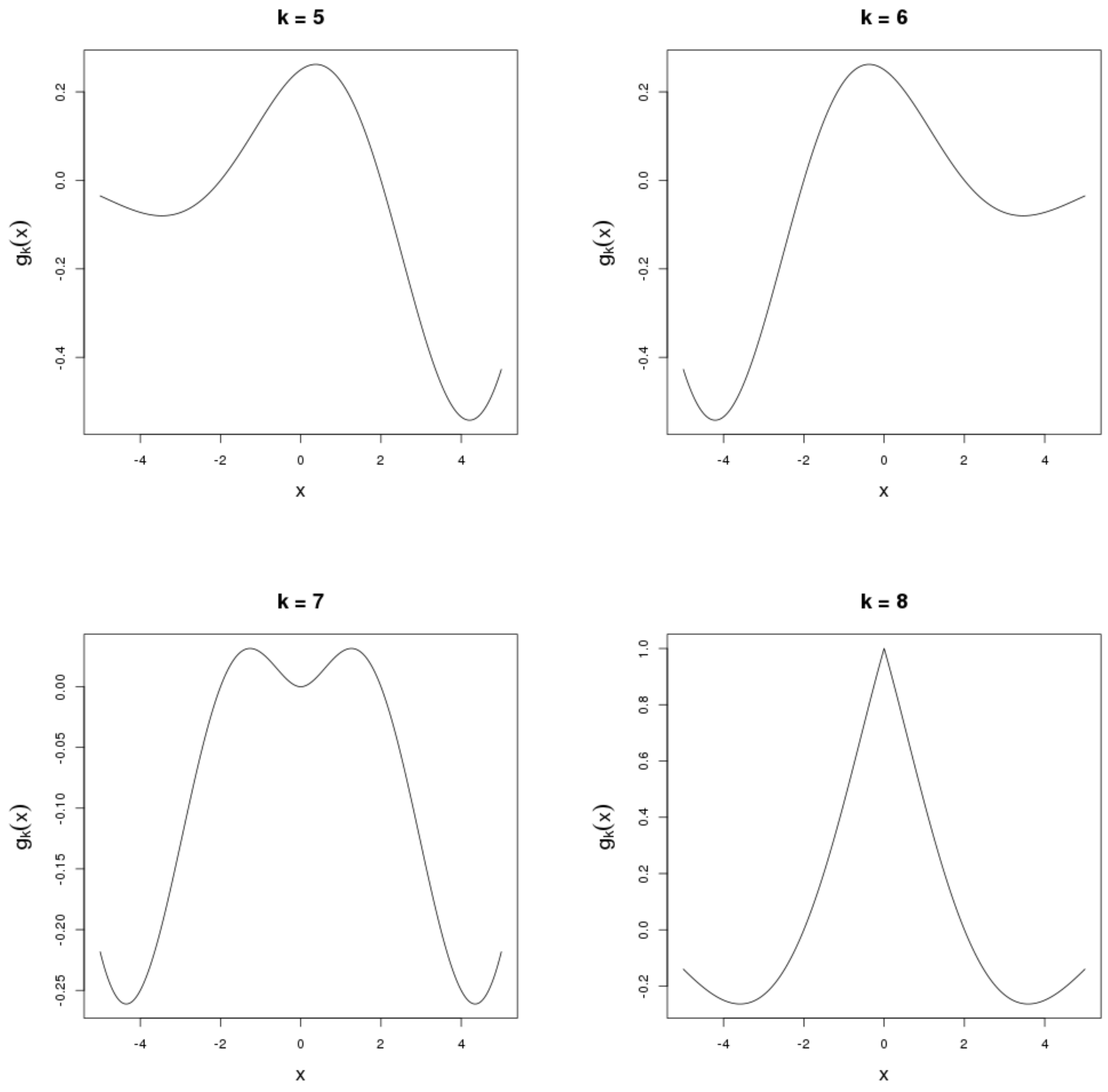
```
## @fn same as above but written in C
## @comment this function overwrites the above function
MFP=function(Y,X){
    dyn.load("MFP.so")
    out=.C("MFP",omega=rep(0,ncol(X)),Y=Y,X=X,n=nrow(X),p=ncol(X))
    dyn.unload("MFP.so")
    return(out$omega)
}



## @fn calculate the weights according to the NIS algorithm
## @param (Y) the response
## @param (X) the predictors
## @param (ell) dimension of the B-splines
NIS=function(Y,X,ell=5,knots=3){
    p=ncol(X)
    omega=foreach(j=1:p,.combine=c)%do%{
        mod=lm(Y~1+bs(X[,j],knots=knots,degree=ell))
        return(mean(mod$fitted^2))
    }
    return(omega)
}



## @fn calculates random orders of the variables
## @param (p) the number of predictors
RandScreen=function(p){
    return(sample(1:p))
}



## @fn obtains the bootstrap standard deviation
## @param (x) the data
## @param (fn) the function to obtain the standard error of
## @param (rep) the number of bootstrap replications
bsSd=function(x,fn,rep=500){
    num=length(x)
    vals=foreach(r=1:rep,.combine=c)%do%{
        fn(sample(x,num,replace=TRUE))
    }

    return(sd(vals))
}



## @fn saves the simulations results to separate files
## @param (sim) an integer indicating which sim is to be saved
## @param (simResults) the results of the simulation
## @param (nVals) the vector of n values
## @param (sVals) the vector of s values
## @param (epsSdVals) the vector of error standard deviation values
## @param (modVals) the vector of values for the different models
## @param (distVals) the vector of values for different design matrix
## distributions
saveResults=function(sim,simResults,nVals,sVals,epsSdVals,modVals,distVals){
    ## every combination of the different settings
    combos=expand.grid(
        n=1:length(nVals),
        s=1:length(sVals),
        epsSd=1:length(epsSdVals),
        mod=1:length(modVals),
        dist=1:length(distVals)
        )

    ## iterate through each setting combination
    for(i in 1:nrow(combos)){
        ## retrieve the index of the current settings
        nInd=combos[i,1]
        sInd=combos[i,2]
        epsSdInd=combos[i,3]
        modInd=combos[i,4]
        distInd=combos[i,5]

        ## save the file
        file=paste("results/","sim",sim,"_n",nInd,"_s",sInd,"_epsSd",epsSdInd,
            "_mod",modInd,"_dist",distInd,".csv",sep="")
        write.table(simResults[nInd,sInd,epsSdInd,modInd,distInd,,],file=file,
                    col.names=FALSE,row.names=FALSE,sep=",")
    }
    cat(paste("Results for sim",sim,"have been saved.\n"))
}
```



6.2 simulation1.R

Code used to obtain the results in table 1.

```
source("methods.R")

## @fn calculate the results for simulation 1.  This is just evaluating the
## screening algorithms.  The results are the minimum number needed to keep to
## retain all relevant predictors.
## @param (Y) the response vector
## @param (X) the design matrix
## @param (s) the number of relevant predictors
```



```
## @param (seed) the random seed
getRes1=function(Y,X,s,seed){
    set.seed(seed)
    minNum=rep(0,4)

    ## SIS
    ord=order(SIS(Y,X),decreasing=TRUE)
    minNum[1]=max(which(ord%in%(1:s)))

    ## MFP
    ord=order(MFP(Y,X),decreasing=TRUE)
    minNum[2]=max(which(ord%in%(1:s)))

    ## NIS
    ord=order(NIS(Y,X,ell=3,knots=5),decreasing=TRUE)
    minNum[3]=max(which(ord%in%(1:s)))

    ## Random
    ord=order(RandScreen(ncol(X)),decreasing=TRUE)
    minNum[4]=max(which(ord%in%(1:s)))

    return(minNum)
}


## @fn run the simulation for one setting
## @param (mod) the index of the model to use to make Y
## @param (dist) the index of the distribution to make X
## @param (epsSd) the index of the error variance type
## @param (n) the number of observations
## @param (a) specifies the relationship between n and p
## @param (s) the number of important predictors
## @param (seed) determines the random seed for making X, making eps, and
## obtaining results
runSim1=function(mod,dist,epsSd,n,a,s,seed=0){
    p=floor(exp(n^a))


    ## generate beta
    set.seed(0) ## for ensuring consistent beta
    beta=c((-1)^rbinom(s,1,.25)*rgamma(s,1,1/2),rep(0,p-s))

    ## make the data set
    dat=makeData(mod,dist,epsSd,n,p,beta,seed)

    ## return the vector of minimum number of predictors needed to
    ## keep all important predictors
    return(getRes1(dat$Y,scale(dat$X),s,seed))
}



## @fn replicate the simulation and obtain mean and sd for results
## @param (mod) the index of the model to use to make Y
## @param (dist) the index of the distribution to make X
## @param (epsSd) the index of the error variance type
## @param (n) the number of observations
## @param (a) specifies the relationship between n and p
## @param (s) the number of important predictors
## @param (seed) an optional shift of the random seeds
repSim1=function(numReps,mod,dist,epsSd,n,a,s,seed=0){
    res=foreach(r=1:numReps,.combine=rbind)%dopar%{
        runSim1(mod,dist,epsSd,n,a,s,r*10+seed)
    }

    means=colMeans(res)
    medians=apply(res,2,median)

    meanSd=apply(res,2,sd)/sqrt(nrow(res))
    medianSd=apply(res,2,bsSd,fn=median)

    return(matrix(c(means,meanSd,medians,medianSd),nrow=ncol(res),ncol=4))
}


## @fn run simulation 1
## @param (numReps) the number of replications
## @param (nVals) the vector of n values
## @param (aVals) the vector of a values
## @param (sVals) the vector of s values
## @param (epsSdVals) the vector of error standard deviation values
## @param (modVals) the vector of values for the different models
## @param (distVals) the vector of values for different design matrix
## distributions
simulation1=function(numReps,nVals,aVals,sVals,epsSdVals,modVals,distVals){
    ## every combination of the different settings
    combos=expand.grid(
        n=1:length(nVals),
        s=1:length(sVals),
        epsSd=1:length(epsSdVals),
        mod=1:length(modVals),
        dist=1:length(distVals)
        )

    ## initialize container for results
    simResults=array(NA,
        dim=c(
            length(nVals),
            length(sVals),
            length(epsSdVals),
            length(modVals),
            length(distVals),
            4,
            4)
        )

    ## iterate through each setting combination
    for(i in 1:nrow(combos)){
        print(i)

        ## obtain index of current settings
        nInd=combos[i,1]
        sInd=combos[i,2]
        epsSdInd=combos[i,3]
        modInd=combos[i,4]
        distInd=combos[i,5]

        ## obtain value of current settings
        n=nVals[nInd]
        a=aVals[nInd] ## a uses nInd
        s=sVals[sInd]
        epsSd=epsSdVals[epsSdInd]
        mod=modVals[modInd]
        dist=distVals[distInd]

        ## obtain and store results
        simResults[nInd,sInd,epsSdInd,modInd,distInd,,] =
            repSim1(numReps,mod,dist,epsSd,n,a,s,0)
    }
    return(simResults)
}
```

6.3 simulation2.R

Code used to obtain the results in table 3.

```
source("methods.R")

library(glmnet)


## @fn variable selection with the lasso
## @param (Y) the response vector
## @param (X) the design matrix
selectWithLasso=function(Y,X){
    fit=cv.glmnet(X,Y)
    return(which(coef(fit)[-1]!=0))
}


## @fn peform the two-stage variable selection recording how many remaining
## predictors are relevant and how many total are left.
## @param (Y) the response vector
## @param (X) the design matrix
## @param (s) the number of relevant predictors
## @param (omega) the vector of rankings
varSelection=function(Y,X,s,omega){
    n=nrow(X)
    p=ncol(X)
    cutOff=floor(n/log(n))

    ## obtain the final set of variables after screening and the lasso
    vars=order(omega,decreasing=TRUE)[1:cutOff]
    vars=vars[selectWithLasso(Y,X[,vars])]

    ## which are correct
    numCorrect=sum(vars %in% (1:s))
    numTotal=length(vars)

    return(c(numCorrect,numTotal))
}


## @fn obtain variable selection results for the listed methods.  These results
## are after the complete two-stage selection methods.  For each method the
## number of relevant variables remaining are recorded as well as the total
## number left
## @param (Y) the response vector
## @param (X) the design matrix
## @param (s) the number of relevant predictors
## @param (seed) the random seed
getRes2=function(Y,X,s,seed){
    set.seed(seed)
    res=matrix(0,nrow=4,ncol=2)

    ## SIS
    res[1,]=varSelection(Y,X,s,SIS(Y,X))

    ## MFP
    res[2,]=varSelection(Y,X,s,MFP(Y,X))
    ## NIS
    res[3,]=varSelection(Y,X,s,NIS(Y,X,ell=3,knots=5))

    ## Random
    res[4,]=varSelection(Y,X,s,RandScreen(ncol(X)))

    return(res)
}


## @fn run the simulation for one setting
## @param (mod) the index of the model to use to make Y
## @param (dist) the index of the distribution to make X
## @param (epsSd) the index of the error variance type
## @param (n) the number of observations
## @param (a) specifies the relationship between n and p
## @param (s) the number of important predictors
## @param (seed) determines the random seed for making X, making eps, and
## obtaining results
runSim2=function(mod,dist,epsSd,n,a,s,seed=0){
    p=floor(exp(n^a))

    ## generate beta
    set.seed(0) ## for ensuring consistent beta
    beta=c((-1)^rbinom(s,1,.25)*rgamma(s,1,1/2),rep(0,p-s))

    ## make the data set
    dat=makeData(mod,dist,epsSd,n,p,beta,seed)

    ## return the vector of minimum number of predictors needed to
    ## keep all important predictors
    return(getRes2(dat$Y,scale(dat$X),s,seed))
}



## @fn replicate the simulation and obtain mean and sd for results
## @param (mod) the index of the model to use to make Y
## @param (dist) the index of the distribution to make X
## @param (epsSd) the index of the error variance type
## @param (n) the number of observations
## @param (a) specifies the relationship between n and p
## @param (s) the number of important predictors
## @param (seed) an optional shift of the random seeds
```

```
repSim2=function(numReps,mod,dist,epsSd,n,a,s,seed=0){
    res=foreach(r=1:numReps,.combine=list)%dopar%{
        runSim2(mod,dist,epsSd,n,a,s,r*10+seed)
    }
    res=aperm(array(unlist(res),dim=c(4,2,numReps)),c(3,1,2))

    correctMeans=rep(0,4)
    correctSds=rep(0,4)
    for(i in 1:4){
        correctMeans[i]=mean(res[,i,1])
        correctSds[i]=sd(res[,i,1])/sqrt(numReps)
    }

    totalMeans=rep(0,4)
    totalSds=rep(0,4)
    for(i in 1:4){
        totalMeans[i]=mean(res[,i,2])
        totalSds[i]=sd(res[,i,2])/sqrt(numReps)
    }

    return(matrix(c(correctMeans,correctSds,totalMeans,totalSds),nrow=4,ncol=4))
}



## @fn run simulation 2
## @param (numReps) the number of replications
## @param (nVals) the vector of n values
## @param (aVals) the vector of a values
## @param (sVals) the vector of s values
## @param (epsSdVals) the vector of error standard deviation values
## @param (modVals) the vector of values for the different models
## @param (distVals) the vector of values for different design matrix
## distributions
simulation2=function(numReps,nVals,aVals,sVals,epsSdVals,modVals,distVals){
    ## every combination of the different settings
    combos=expand.grid(
        n=1:length(nVals),
        s=1:length(sVals),
        epsSd=1:length(epsSdVals),
        mod=1:length(modVals),
        dist=1:length(distVals)
        )

    ## initialize container for results
    simResults=array(NA,
        dim=c(
            length(nVals),
            length(sVals),
            length(epsSdVals),
            length(modVals),
            length(distVals),
            4,
            4)
        )

    ## iterate through each setting combination
    for(i in 1:nrow(combos)){
        print(i)

        ## obtain index of current settings
        nInd=combos[i,1]
        sInd=combos[i,2]
        epsSdInd=combos[i,3]
        modInd=combos[i,4]
        distInd=combos[i,5]

        ## obtain value of current settings
        n=nVals[nInd]
        a=aVals[nInd] ## a uses nInd
        s=sVals[sInd]
        epsSd=epsSdVals[epsSdInd]
        mod=modVals[modInd]
        dist=distVals[distInd]

        ## obtain and store results
        simResults[nInd,sInd,epsSdInd,modInd,distInd,,] =
            repSim2(numReps,mod,dist,epsSd,n,a,s,0)
    }
    return(simResults)
}
```

6.4 MFP.c

The MFP method was expensive in `R` so it was rewritten in `C` to alleviate this burden.

```
// @fn model-free screening procedure.
// @param (omega) a pointer to an array of doubles for variable rankings
// @param (Y) a pointer to an array of doubles for the response
// @param (X) a pointer to an array of doubles for the design matrix
// @param (n) a pointer to an integer specifying sample size
// @param (p) a pointer to an integer specifying data dimension
void MFP(double * omega, double * Y, double * X, int * n, int * p){
  int i,j,k;
  double res0;
  double res1;

  for(k=0; k<*p; k++){ // repeat for each predictor
    res0=0;
    for(j=0; j<*n; j++){ // outer empirical expectation
      res1=0;
      for(i=0; i<*n; i++){ // inner empirical expectation
        res1+=(Y[i]<Y[j] ? X[i+k*(*n)] : 0);
      }
      res1/=(double)(*n);
      res0+=res1*res1;
    }
    res0/=(double)(*n);

    // store values
    omega[k]=res0;
  }
}
```

6.5 runSim1.R

A short file to run the simulation for table 1 and save all results to individual `csv` files.

```
rm(list=ls(all=TRUE))
source("simulation1.R")

## settings
nVals=c(100,200,200)
aVals=c(.35,.35,.375)
sVals=c(10,15,25)
epsSdVals=1:3
modVals=1:2
distVals=1:1


numReps=200

## obtain results
```

```
registerDoMC(64)
simResults=simulation1(numReps,nVals,aVals,sVals,epsSdVals,modVals,distVals)

## save results to files
saveResults(1,simResults,nVals,sVals,epsSdVals,modVals,distVals)

## save container results as Rdata file
save("simResults",file="simResults1.Rdata")
```

6.6 runSim2.R

A short file to run the simulation for table 3 and save all results to individual `csv` files.

```
rm(list=ls(all=TRUE))
source("simulation2.R")

## settings
nVals=c(100,200,200)
aVals=c(.35,.35,.375)
sVals=c(10,15,25)
epsSdVals=1:3
modVals=1:2
distVals=1:1


numReps=200

## obtain results
registerDoMC(64)
simResults=simulation2(numReps,nVals,aVals,sVals,epsSdVals,modVals,distVals)

## save results to files
saveResults(2,simResults,nVals,sVals,epsSdVals,modVals,distVals)

## save container results as Rdata file
save("simResults",file="simResults2.Rdata")
```

References

D. L. Donoho. High-dimensional data analysis: the curses and blessings of dimensionality. In *American Mathematical Society Conf. Math Challenges of the 21st Century*, 2000.

J. Fan and R. Li. Statistical Challenges with High Dimensionality: Feature Selection in Knowledge Discovery. *ArXiv Mathematics e-prints*, February 2006.

J. Fan and J. Lv. A Selective Overview of Variable Selection in High Dimensional Feature Space (Invited Review Article). *ArXiv e-prints*, October 2009.

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. doi: 10.1198/016214501753382273.

Jianqing Fan and Jinchi Lv. Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(5): 849–911, 2008. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2008.00674.x.

Jianqing Fan, Yang Feng, and Rui Song. Nonparametric independence screening in sparse ultra-high-dimensional additive models. *Journal of the American Statistical Association*, 106(494):544–557, 2011. doi: 10.1198/jasa.2011.tm09779.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):pp. 267–288, 1996. ISSN 00359246.

Lan Wang, Yichao Wu, and Runze Li. Quantile regression for analyzing heterogeneity in ultra-high dimension. *Journal of the American Statistical Association*, 107(497):214–222, 2012. doi: 10.1080/01621459.2012.656014.

Li-Ping Zhu, Lexin Li, Runze Li, and Li-Xing Zhu. Model-free feature screening for ultrahigh-dimensional data. *Journal of the American Statistical Association*, 106(496):1464–1475, 2011. doi: doi:10.1198/jasa.2011.tm10563.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005. ISSN 1467-9868. doi: 10.1111/j.1467-9868.2005.00503.x.