NOISE ACCUMULATION IN HIGH DIMENSIONAL CLASSIFICATION

Miriam R. Elman

A THESIS

Presented to the
Oregon Health & Science University / Portland State University
School of Public Health in partial fulfillment of
the requirements for the degree of

Master of Science

March 2018

Department of Public Health & Preventive Medicine
School of Public Health
Oregon Health & Science University

CERTIFICATE OF APPROVAL

This is to certify that the Master's thesis of
Miriam R. Elman
has been approved

Dongseok Choi, PhD (Mentor/Advisor)

Jessica Minnier, PhD (Committee Member)

Xiaohui Chang, PhD (Committee Member)

TABLE OF CONTENTS

List of Tables

List of Figures

For my parents

Acknowledgements

I am extremely grateful to everyone who contributed to this project, my understanding of statistics, and my life over the last six years. Foremost, I would like to express my gratitude to Dr. Dongseok Choi who has been both a generous advisor and patient teacher. This work could not have been completed without his support and guidance. He contributed a tremendous amount of time, energy, and thought to this project as well as to advancing my knowledge of machine learning, Big Data, and many other areas of statistics. I wish to thank my committee members, Drs Jessica Minnier and Xiaohui Chang, for their generous contribution of time and input on this project. I appreciate their encouragement and insight. I would also like to acknowledge the biostatistics professors at Oregon Health & Science University and Portland State University I have had over the course of this program and my Masters in Public Health. I appreciate all the guidance, support, and wisdom they have provided. A special thanks to Dr. Dawn Peters who kindly and perseveringly shepherded my learning of the foundational elements of statistics. Finally, my family and friends keep me getting up in the morning. They are the unsung heroes of my life.

Abstract

A tremendous amount of attention has been paid to Big Data in recent years. Such data hold promise for scientific discoveries but also pose challenges for analyses. In their 2014 article ”Challenges to Big Data analysis,” Fan and colleagues propose that the high dimensionality of Big Data introduces statistical problems including noise accumulation. This thesis explores noise accumulation in high dimensional two-group classification problems. First, it aims to determine whether noise accumulation threatens the discriminative ability of classifiers developed with three common machine learning approaches – random forest, support vector machine, and boosted classification trees. Four different scenarios with differing amount of signal strength are simulated to evaluate each method. After determining that noise accumulation may impact the performance of these classifiers, the thesis characterizes factors which impact noise accumulation. Simulations varying sample size, signal strength, signal strength proportional to the number predictors, and signal magnitude are conducted with random forest classifiers. Finally, this thesis develops Total Signal Index to summarize the amount of signal relative to noise in a two-group classification problem. Theoretical and empirical versions of this measure are defined and simulations are used to assess them.

1 Introduction

Big Data have become pervasive in the last ten years. Technological advances and increased computing power have enabled data to be continuously produced and cheaply stored, providing researchers with a wealth of information at a scale not previously available. Such data are not new to statistics. The term ”huge” dataset was used in a 1994 article in COMPSTAT – Proceedings in Computational Statistics.[1,2] ”Massive” datasets were described in the proceedings for the Committee on Applied and Theoretical Statistics in the same year.[3] However, such data and corresponding analytics have become increasingly popular. While such Big Data provide opportunities for new discoveries, they may challenge conventional statistical methods not developed for settings in which the number of predictors far exceeds sample size or sparse data is used for simultaneous estimation of a large number of parameters.

A range of definitions has been proposed for Big Data.[4] We define it in accordance with ”Challenges of Big Data analysis,” which motivated this project.[5] The authors Fan et al. characterize Big Data by massive size and high dimensionality. Although they do not explicitly define what massive size entails, high dimensionality occurs when the number of predictors far exceeds sample size – that is, $n \ll p$. The authors posit several unique features of Big Data's dimensionality that they assert pose significant challenges to traditional data analysis and, consequently, necessitate the development of new statistical methods. They include noise accumulation, spurious correlation, and incidental endogeneity. This thesis focuses on noise accumulation.

Noise accumulation occurs when simultaneous estimation or testing of multiple parameters results in estimation error. This can happen when many weak predictors or ones unrelated to the outcome are included in a model. Such noise can concentrate, obstructing true signal and the estimation of corresponding parameters. Noise accumulation is generally not an issue in conventional statistical settings where sample size exceeds the number of predictors. High dimensional data – such as those arising in gene expression studies and biomedical imaging – is highly susceptible to its effect. Fan and Fan (2008) demonstrate that prediction with most classification rules based on linear discriminant rules performs equivalently to guessing with high dimensional data due to noise accumulation.[6] They also assert that projection methods such as principal component analysis (PCA) tend to perform poorly in high dimensional settings. Hall et al. (2008) and Fan (2013) studied distance-based classifiers in high dimensional settings and found performance was adversely affected.[7,8] The impact of noise accumulation on classification using PCA was further explored by simulation Fan et al. (2014) in "Challenges of Big Data analysis."[5]

To illustrate the issue of noise accumulation, Fan et al. explore a classification scenario with data from two classes.[5] Both classes are drawn from standard multivariate normal distributions with identity covariance matrices. Data from the first class is from

$$\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \sim \mathrm{MV}_p(\boldsymbol{\mu}_1, \boldsymbol{I}_p)$$

where $\boldsymbol{\mu}_1 = \boldsymbol{0}$, $n = 100$ from each class, and $p = 1000$. The second class,

$$\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_n \sim \mathrm{MV}_p(\boldsymbol{\mu}_2, \boldsymbol{I}_p)$$

is constructed identically to the first class except the first 10 elements of $\boldsymbol{\mu}_2$ are nonzero with value equal to 3 and all other entries zero. That is,

$$\boldsymbol{\mu}_2 = (3,3,3,3,3,3,3,3,3,3,0,\ldots,0)$$

Thus, the nonzero components of $\boldsymbol{\mu}_2$ comprise signal that differentiates the two classes. Fan et al. computed principal components for the first $p =$ 2, 40, 200, and 1000 predictors then visually assessed how well the two classes can be separated – that is, the discriminative power – by plotting the first two principal components. They report that discriminative power is high when there are a low number of predictors, which they found to be $p_0 < 200$

in their simulations.[*] When the number of predictors is low enough, there is adequate signal to drown out noise and differentiate between the classes. As the number of predictors grows, noise eventually overwhelms signal and predicting the class membership for the observations becomes infeasible. Fan et al. found this threshold to be $p_0 \geq 200$ in their simulations. Since the amount of signal in the scenario they describe is fixed to be 10 nonzero elements, no signal is added after $p_0 > 10$ and further input is noise. The described setting was the only simulation they reported in their article and appeared to be the basis of their conclusions.

We replicated the simulations conducted by Fan et al. and repeated the PCA they describe. Despite careful adherence to their published simulation, our findings were discrepant with those presented in the article. Where Fan and colleagues observed low discriminative ability at $p_0 \geq 200$ in the scenario they describe, we were still able to differentiate between the two groups. Seeking to understand the differences in our results and weigh whether noise accumulation was an issue in high dimensional two-group classification, we explored additional scenarios with PCA in which we varied the number of nonzero elements as well as the value of the entries. Results of these investigations did not resolve the discrepancy but suggested there might be more to discover about noise accumulation. Further, Fan et al. use PCA as means to visually access discriminative ability rather than a direct method to develop and evaluate a classification rule. We thought using a classification approach might provide more objective results about discriminative ability. Noise accumulation is well known in regression[9] but was quantified first in classification by Fan and Fan.[6] Besides this work and that discussed above, little research has been done to characterize noise accumulation in classification. To our knowledge, limited work to-date has directly tested the discriminative ability of classifiers under conditions of noise accu-

[*]Note that we will use p_0 throughout this thesis to designate to specific values of predictors from simulations whereas p is reserved to refer to predictors more generally.

mulation. In addition to the previously cited work done with distance-classifiers, linear discriminant rules, and PCA, Fan et al showed that the independent classification rule was susceptible to noise accumulation but could be overcome with variable selection.[6] Classification approaches commonly used in high dimensional settings such as gene expression analysis, however, have not been explored.

This thesis focuses on noise accumulation in two-group classification, problems where data originate from two distinct classes and a rule is constructed with which to classify new observations into either the first or second class. It will cover three topics related to noise accumulation in the context of these types of classification problems. First, we present our findings from PCA described by Fan et al. in ”Challenges of Big Data analysis” and expand to additional scenarios. We apply these scenarios to three popular classification approaches in high dimensional settings – support vector machines, random forests, and boosted classification trees – and assess the results. Second, we explore characteristics of noise accumulation in two-group classification, using a random forest approach to construct the classification rules. Finally, we develop theoretical and empirical versions of Total Signal Index, a measure to quantify signal compared to background noise in these settings.

2 Comparison of Classification Methods

In ”Challenges of Big Data analysis,” Fan et al. conducted a simulation in which data from two classes are drawn from standard multivariate normal distributions with equal sample size n for each class, p predictors, and an identity covariance matrix. The two classes were defined as:

$$\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \sim \mathrm{MV}_p(\boldsymbol{\mu}_1, \boldsymbol{I}_p)$$

$$\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_n \sim \mathrm{MV}_p(\boldsymbol{\mu}_2, \boldsymbol{I}_p),$$

where $\boldsymbol{\mu}_1 = \mathbf{0}$, $n = 100$ for each class, and $p = 1000$. The first 10 elements of $\boldsymbol{\mu}_2$ are nonzero with value equal to three and all other entries zero:

$\boldsymbol{\mu}_2 = (3,3,3,3,3,3,3,3,3,3,0,\ldots,0)$.

As described previously, these authors used principal component analysis (PCA) for the first $p = 2$, 40, 200, and 1000 predictors to create scatterplots for the first two principal components (Figure 1[†]). They determined that discriminative power was high when the number of predictors was sufficiently small, which they report as $p_0 < 200$ in their simulations. At $p_0 \geq 200$, they found that noise eventually overwhelms signal.

Like Fan et al., we simulated data for two classes from standard multivariate normal distributions with an identity covariance matrix and p predictors

Class 1: $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \sim \mathrm{MV}_p(\boldsymbol{\mu}_1, \boldsymbol{I}_p)$

Class 2: $\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_n \sim \mathrm{MV}_p(\boldsymbol{\mu}_2, \boldsymbol{I}_p)$

where $\boldsymbol{\mu}_1 = \mathbf{0}$, $\boldsymbol{\mu}_2$ was defined to be sparse with m nonzero elements and the remaining entries equal to zero, and $n = 100$ for each class. In our simulations, we extended the total

[†]In the figure the authors use m for the number or predictors rather than p as we use throughout this thesis

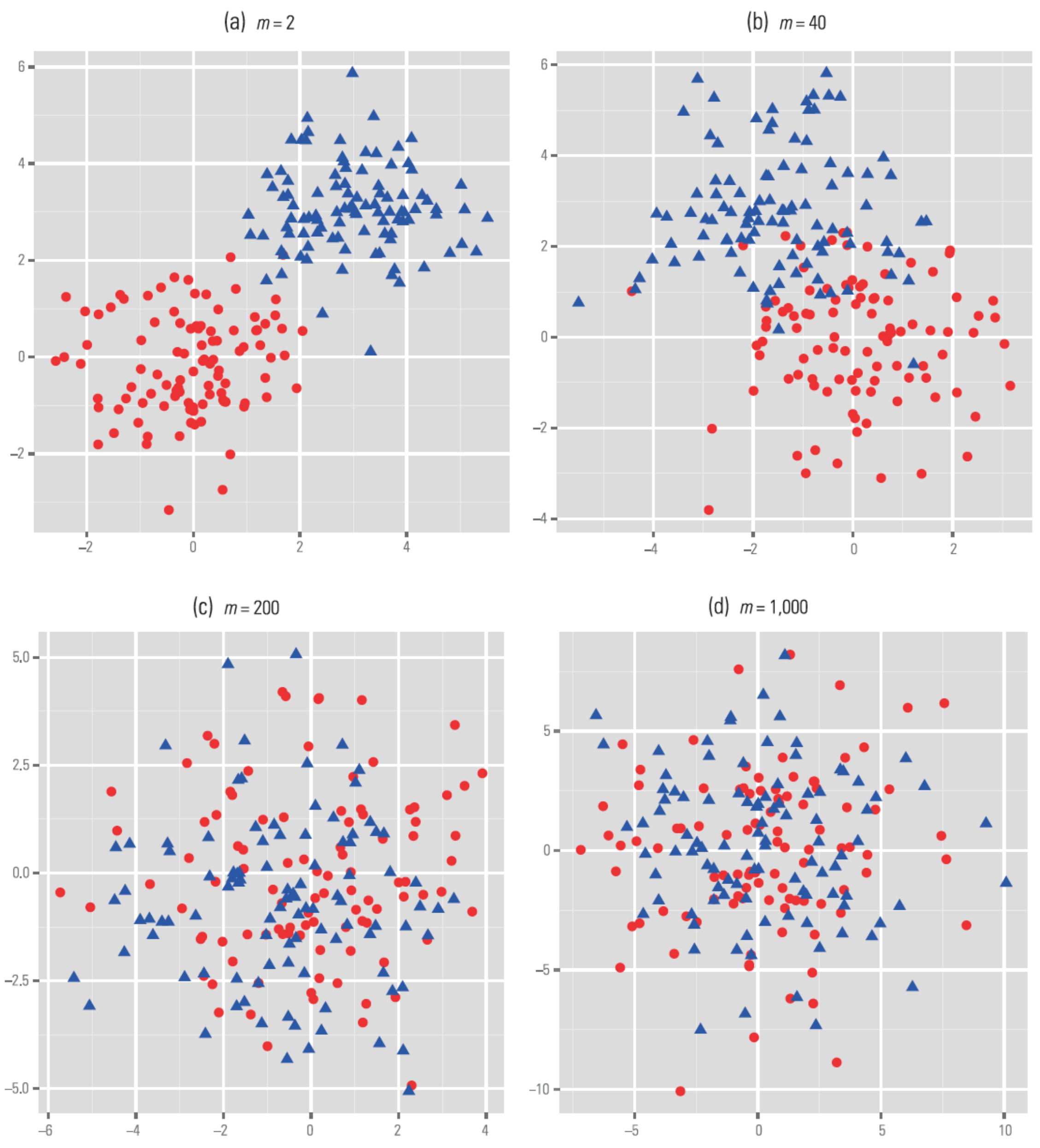

Figure 1. Scatterplots of the projection of observed data (n = 100 for each class) onto the first two principal components of the m-dimensional space. Projected data with the red circles and the blue triangles indicate the first and second classes, respectively.[‡]

[‡]Fan, Jianqing, Fang Han, and Han Liu. "Challenges of big data analysis." *National science review* (2014): 299 by permission from Oxford University Press.

number of predictors used by Fan et al. from $p = 1000$ to 5000 as well as considered four different scenarios for the nonzero elements of $\boldsymbol{\mu}_2$ (Table 1).

Table 1. Scenarios used in simulations for classification methods

Scenario	m	Signal magnitude	Form of $\boldsymbol{\mu}_2$
1	10	3	$(3,3,3,3,3,3,3,3,3,3,0,\ldots,0)$
2	6	3	$(3,3,3,3,3,3,0,\ldots,0)$
3	2	3	$(3,3,0,\ldots,0)$
4	10	1	$(1,1,1,1,1,1,1,1,1,1,0,\ldots,0)$

m, number of nonzero elements in $\boldsymbol{\mu}_2$.

We started by replicating the PCA simulation conducted by Fan et al. described in Scenario 1 then repeated the analyses for each of the other scenarios using a single simulated dataset. We also sought to build classification rules using different methods and assess their discriminative ability for the scenarios initially explored with principal components. We used three common machine learning methods – random forests (RF), support vector machines (SVM), and boosted classification trees (BCT) – to build classifiers and evaluate their performance. For each method and scenario, a classification rule was developed for $p = 2$ to 5000 predictors on a training dataset. This classifier was then applied to a corresponding test dataset and used to predict whether new observations should be categorized into the first or second class. This process was repeated 100 times on training datasets then these classifiers were applied to 100 test datasets to predict class membership for observations from 100 test datasets. Classifiers' discriminative power was assessed by the median classification error for test datasets by comparing the categorization predicted by the classifier to its true class in the test dataset. Along with the median test error, we report 10th and 90th percentile bounds. We evaluated the overall trend of median classification error in the scenarios as well as explicitly examine the maximum classification error for $p_0 < 10$ and $p_0 = 5000$.

Simulations were batch processed in R version 3.4.0 using the OHSU Exacloud computational environment, developed in partnership with Intel and operated by the OHSU Advanced Computing Center.[10] The nodes employed for analyses were running on CentOS Linux 7. PCA was conducted using the prcomp function in base R while randomForest (4.6-12), e1071 (1.6-8), and gbm (2.1.3) packages were used for to run RF, SVM, and BCT procedures, respectively.[11–13] We primarily used the default settings from each package; see code in Appendix A.1 for details.

2.1 Principal Component Analysis

We computed the principal components for $p = 2$, 10, 100, 200, 1000, and 5000 predictors and plotted the projections of the first two components. Figures 2 through 5 show scatterplots showing the results of these simulations with class membership for observations indicated by the black or red filled circles.

In general, our results were analogous to those of Fan et al. in our visual assessment of scatterplots of observed data projected onto the first two principal components (Figures 2–5). That is, high discriminative power appears possible when p is sufficiently low but decreases as p increases. However, the threshold for what Fan et al. deemed *low* differed in our replication of their simulation. They reported that accumulated noise began to exceed signal at $p_0 \geq 200$ yet we found that the threshold for the number of predictors to achieve high discriminative power appeared to be much higher in Scenario 1 (Figure 2). When we replicated their simulation, our data showed high discriminative ability even up through $p_0 = 5000$. In Scenario 2 – where the number of nonzero elements was reduced to $m = 6$ but the value of each element is three, the procedure produced fairly distinct separation of classes up through $p_0 = 1000$ (Figure 3). With $m = 2$ nonzero elements of equal to three (Scenario 3, Figure 4), discriminative ability disappeared more quickly, becoming poor at

$p_0 < 200$. When the number of nonzero elements was $m = 10$ but the value of each element was one in Scenario 4, high discriminative ability appears possible when $p_0 < 1000$ but is otherwise low (Figure 5). Based on these results, it appears that discriminative ability is a factor of both signal magnitude (value of the nonzero elements) as well as its strength (number of the nonzero elements).

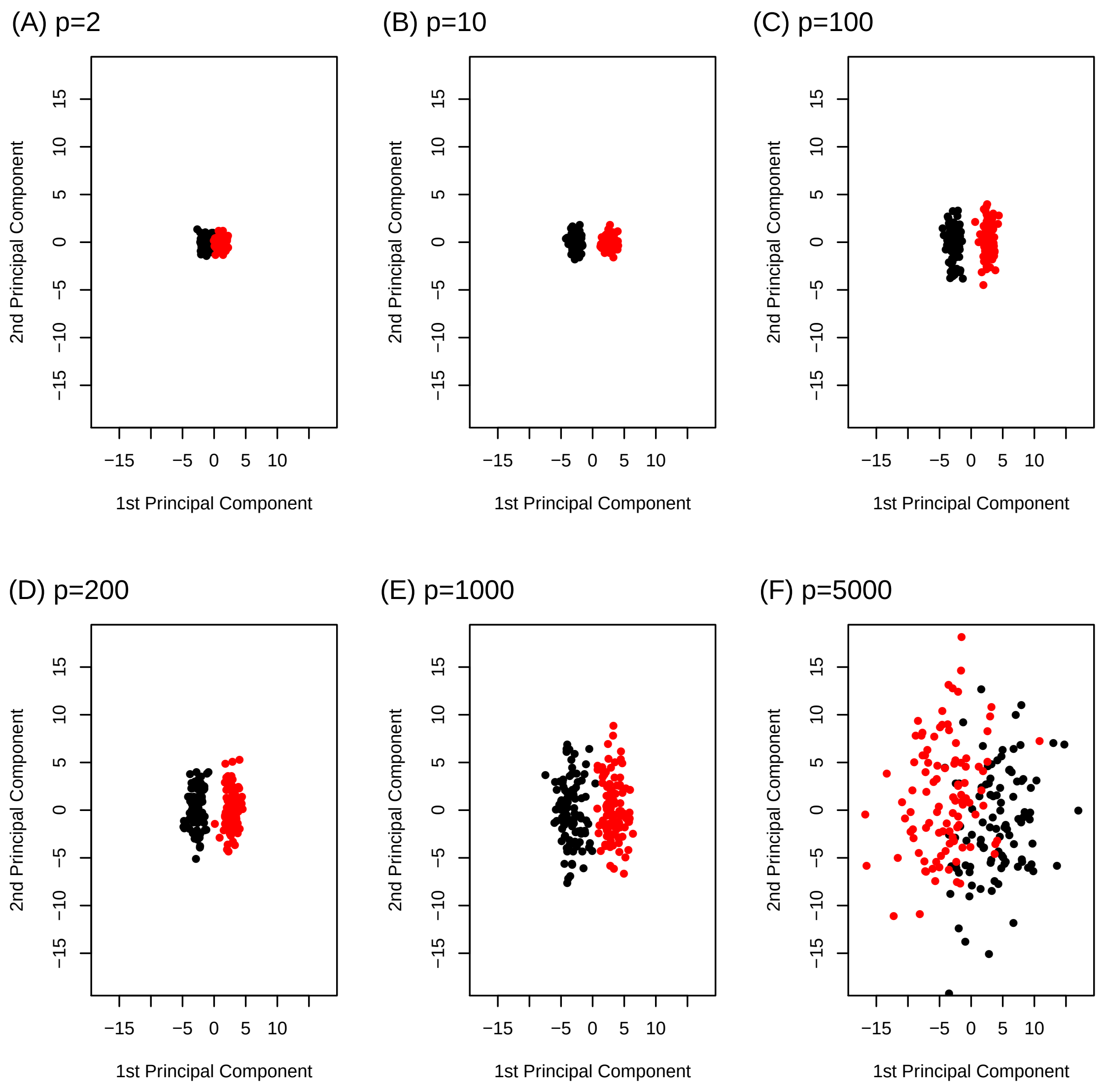

Figure 2. Scatterplots of the projection of observed data from Scenario 1 ($n = 100$ for each class, $m = 10$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to three and $\boldsymbol{\mu}_1 = \mathbf{0}$) onto the first two principal components of the m-dimensional space. Black circles indicate the first class, red circles indicate the second.

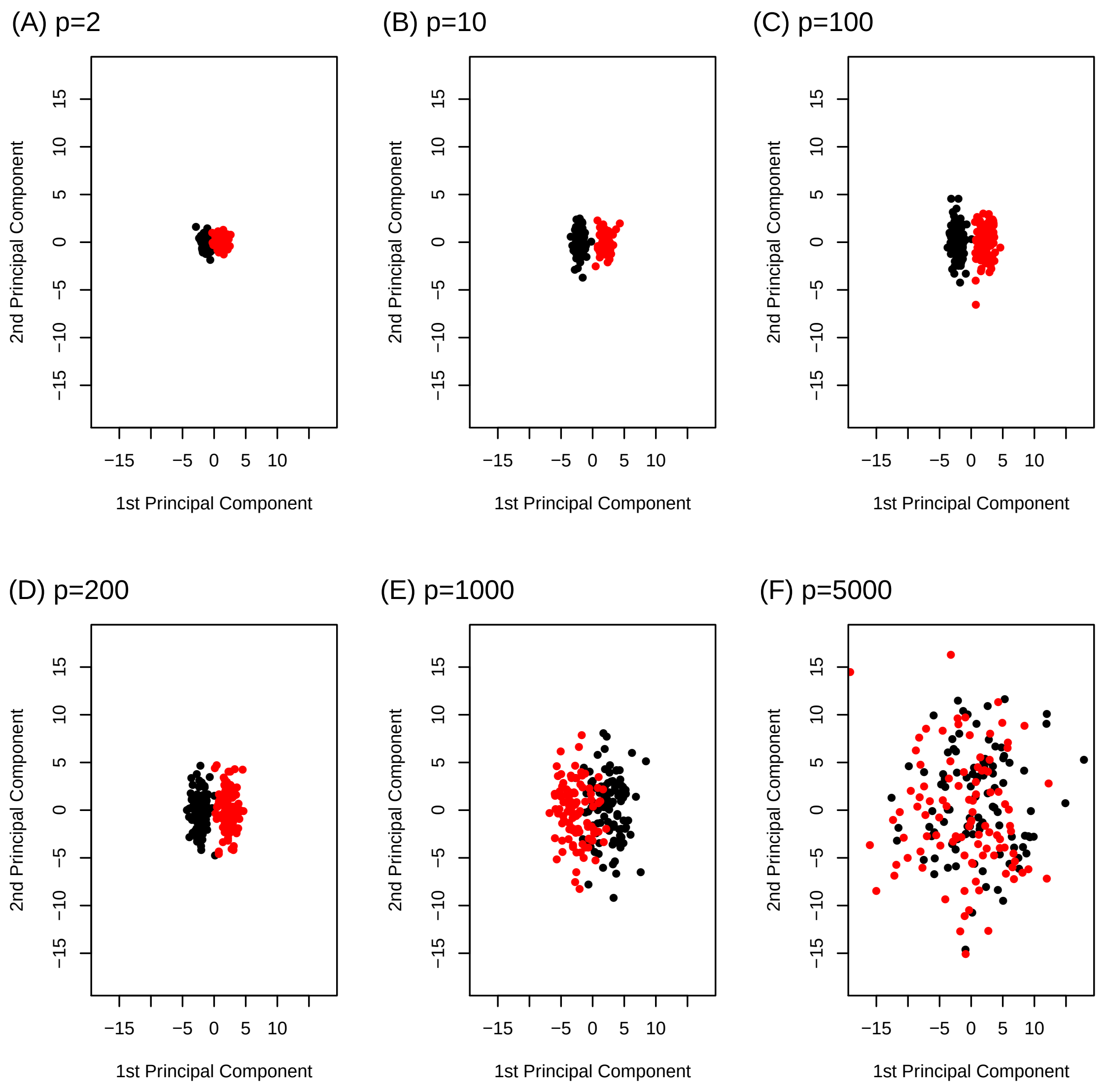

Figure 3. Scatterplots of the projection of observed data from Scenario 2 ($n = 100$ for each class, $m = 6$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to three and $\boldsymbol{\mu}_1 = \mathbf{0}$) onto the first two principal components of the m-dimensional space. Black circles indicate the first class, red circles indicate the second.

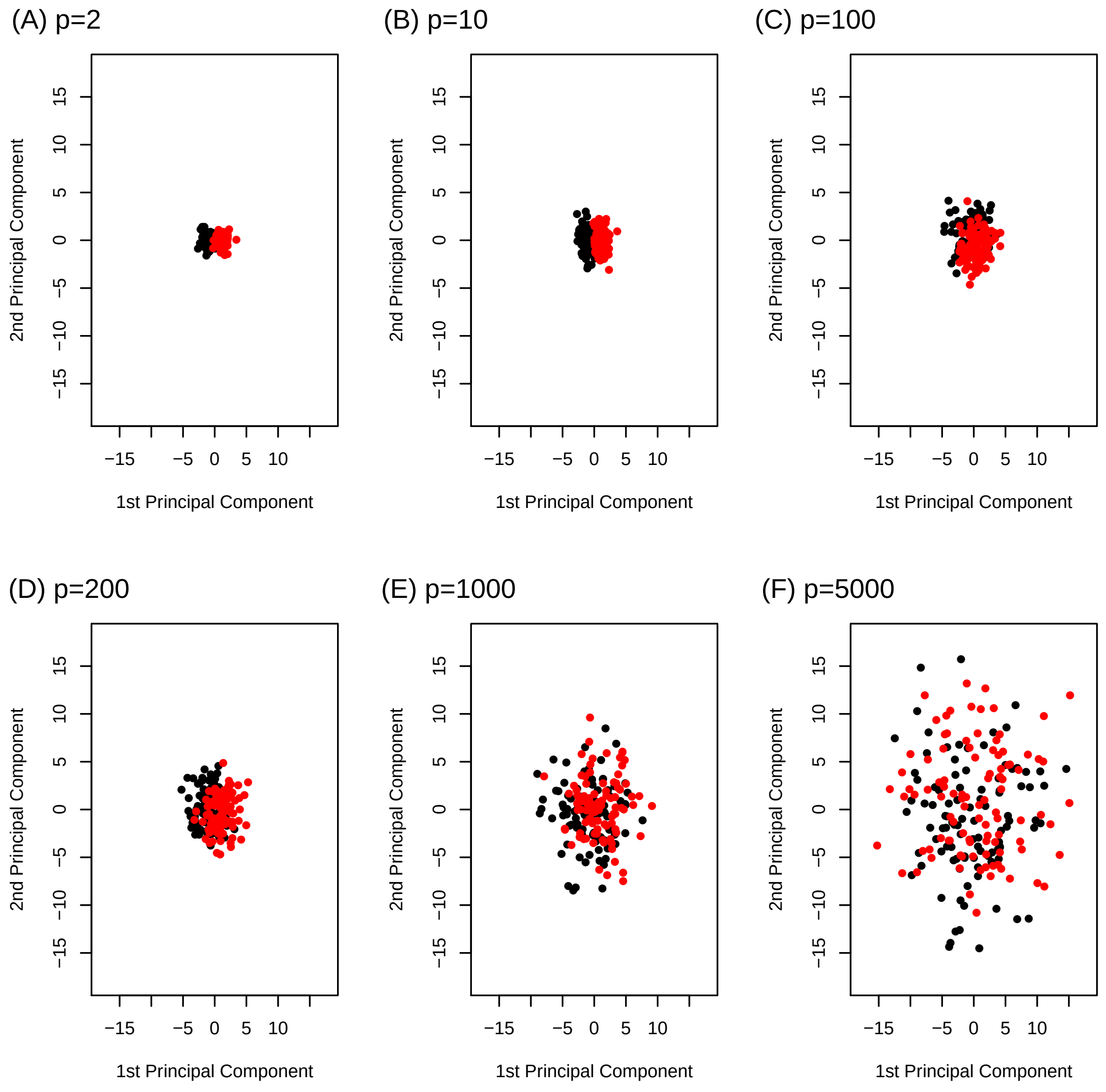

Figure 4. Scatterplots of the projection of observed data from Scenario 3 ($n = 100$ for each class, $m = 2$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to three and $\boldsymbol{\mu}_1 = \mathbf{0}$) onto the first two principal components of the m-dimensional space. Black circles indicate the first class, red circles indicate the second.

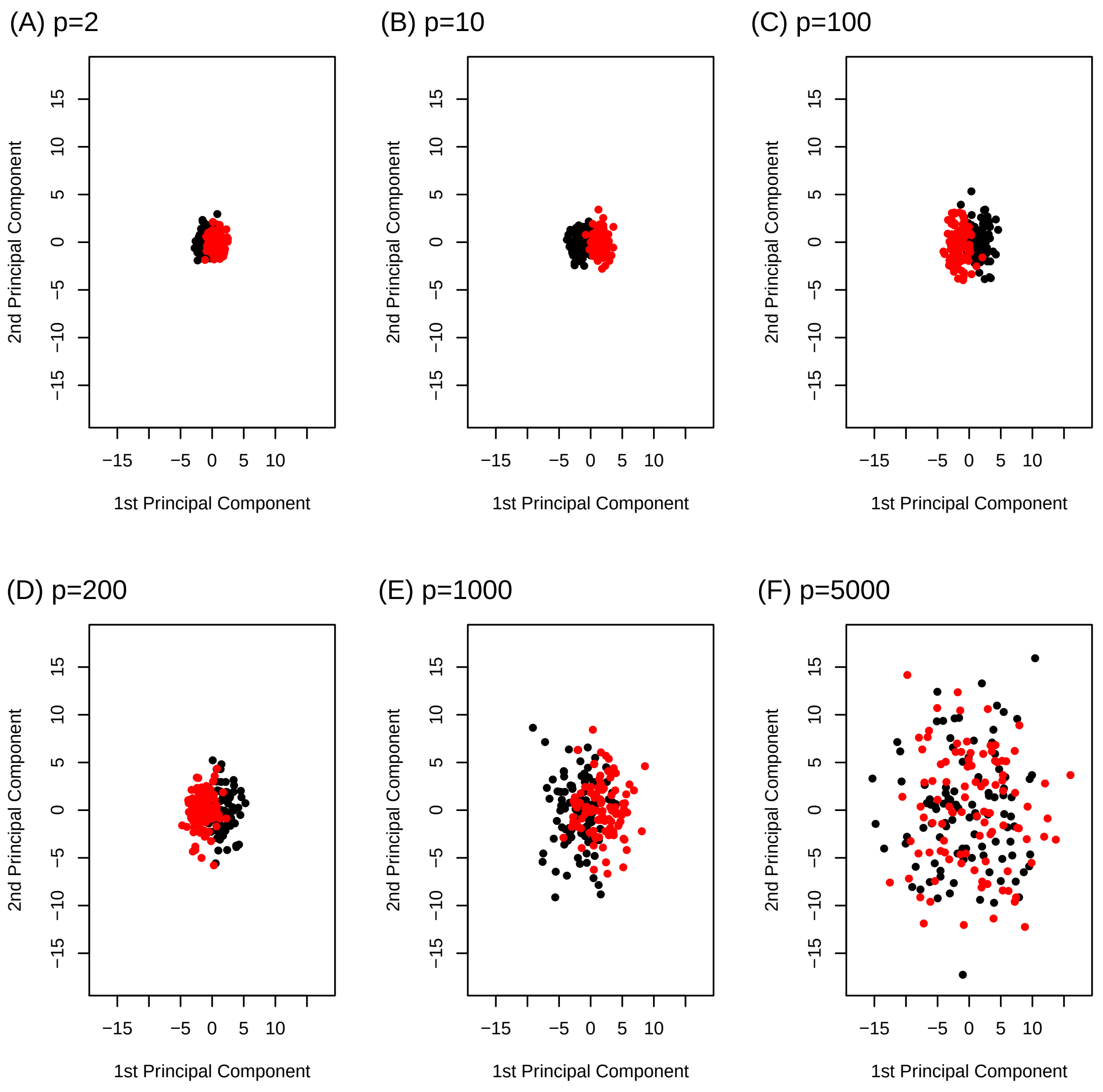

Figure 5. Scatterplots of the projection of observed data from Scenario 4 ($n = 100$ for each class, $m = 10$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to one and $\boldsymbol{\mu}_1 = \mathbf{0}$) onto the first two principal components of the m-dimensional space. Black circles indicate the first class, red circles indicate the second.

2.2 Simulation with Classification Methods

We assessed noise accumulation for the scenarios in Table 1 using RF, BCT, and SVM. Before reviewing the results of these simulations, we present a brief description of each of these methods.

Random forests

RF is a machine learning method developed by Leo Breiman in 2001, expanding upon the idea of decision trees.[14] The algorithm applies bootstrap aggregation (bagging) and random selection of predictors to a set of decision trees. Bagging helps reduce the variation of a single decision tree by repeated, random sampling of a dataset with replacement (that is, bootstrapping) and aggregating the results. In addition to bagging, random forests decorrelate trees by randomly selecting a subset of predictors from all predictors as candidates for the split in the nodes of each tree. A new subset of predictors is chosen for each node. After a specified number of trees have been grown, their results are combined and used for prediction. For classification, majority vote – using the most commonly occurring class from the trees to classify a new observation – is used to aggregate the results.[15] In majority vote, the most commonly occurring class from the collection of trees is used to classify a new observation.

Boosted classification trees

BCT is a tree-based method like RF. The idea behind BCT is to grow simple, sequential trees and slowly improve prediction.[16] In contrast to RF, each successive decision tree is grown off the residuals of the preceding ones rather than being built from an independent, bootstrapped subset of the original data. Like RF, boosting is an ensemble method that synthesizes the results of a large number of trees. Each individual tree, however, may be small with few terminal nodes. As each new decision tree is added to the fitted model, the residuals are updated to inform the next. By fitting small trees, the classifier slowly improves where it performs poorly. Unlike bagging, the construction of each tree depends on the trees that have already been grown. The implementation of boosting used by the R package gbm, gradient boosting, adds a loss function to the described framework.[13,17] The addition of this penalty, which is minimized at each step, improves the performance of the algorithm by reducing overfitting.[15]

Support vector machines

Unlike RF and BCT, SVM is not a tree-method. The original SVM algorithm was developed in the 1960s by Vladmir Vapnik and Alexey Ya Chervonenkis in the 1960s then updated in the 1990s to include nonlinear classifiers and soft margins.[18] SVM originates from the simple, two-dimensional concept of separating observations from two classes by dividing them with a line. A new observation will be assigned a class relative to which side of the line it falls (Figure 6, left). Extending this idea beyond two dimensions, the line used as a classification rule becomes a separating hyperplane. Such straightforward classification may not be possible; if it is, there is an infinite number of hyperplanes that separate the classes. The one equidistant from the observations in either class is called the maximal margin hyperplane and the smallest distance between the observations and the hyperplane is the margin (Figure 6, middle). However, a separating hyperplane – even the maximal margin one – must perfectly divide the classes. In many cases, this does not exist. To address this problem, a support vector classifier uses a soft margin, permitting some observations to be misclassified to the incorrect side of the margin or hyperplane (Figure 6, right). Besides providing a solution to the non-separable case, the support vector classifier helps prevent overfitting training data because it has greater robustness to individual observations. Once the support vector classifier has been chosen, a new observation is still classified by identifying on which side of the hyperplane it lands. Support vector

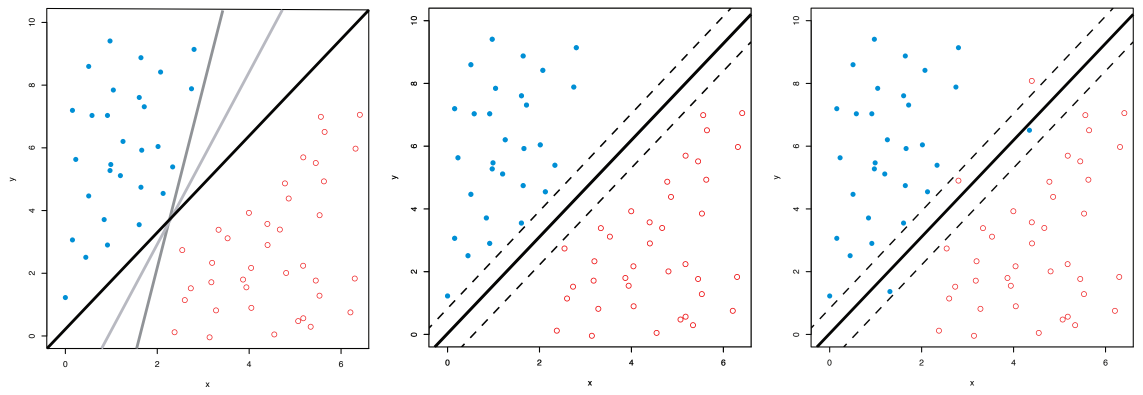

Figure 6. Scatterplots depicting observations from two classes, shown in blue and red. Left: Three of many possible separating hyperplanes are shown by the solid lines. Middle: The maximal margin hyperplane is shown as the solid line while the margin is the distance from either dashed line to the solid one. Right: A support vector classifier equivalent of the adjacent maximal margin hyperplane with points added on the wrong side of the hyperplane and margin. The hyperplane is the solid line and the dashed lines are the margins.

machines generalize support vector classifiers, allowing for a non-linear decision boundary using kernel methods.[15]

2.2.1 Simulations for Scenario 1

All three classification methods demonstrated high discriminative ability in Scenario 1. Overall, the median test error was $< 10\%$ for RF, SVM, and BCT (Figure 7). In particular, RF and BCT performed extremely well with no misclassification when $p_0 > 4$. Below this point, when $2 \leq p_0 \leq 4$, test error reached its maximum for RF and BCT. While the test error dropped substantially for RF and BCT for $p_0 > 10$, it increased for SVM. Table 2 summarizes the highest test error for $p_0 \leq 10$ and its value at $p_0 = 5000$. In Figure 7, RF and BCT do not show evidence of being affected by noise accumulation. After an initial increase, the median test error drops then remains fairly constant for all predictors. The graph for SVM, on the other hand, does show signs of noise accumulation. Around $p_0 = 1000$, the median classification error increases and continues to grow with larger p.

Table 2. Test error for values of p_0 in Scenario 1

p_0	Classification method	Median	10th Percentile	90th Percentile
$p_0 \leq 10$*	Random forests	2.5	1.5	4.0
	Support vector machine	1.5	0.5	3.0
	Boosted classification trees	2.5	1.5	4.5
$p_0 = 5000$	Random forests	0.0	0.0	0.0
	Support vector machine	8.5	5.5	12.1
	Boosted classification trees	0.0	0.0	0.0

*Highest test error.

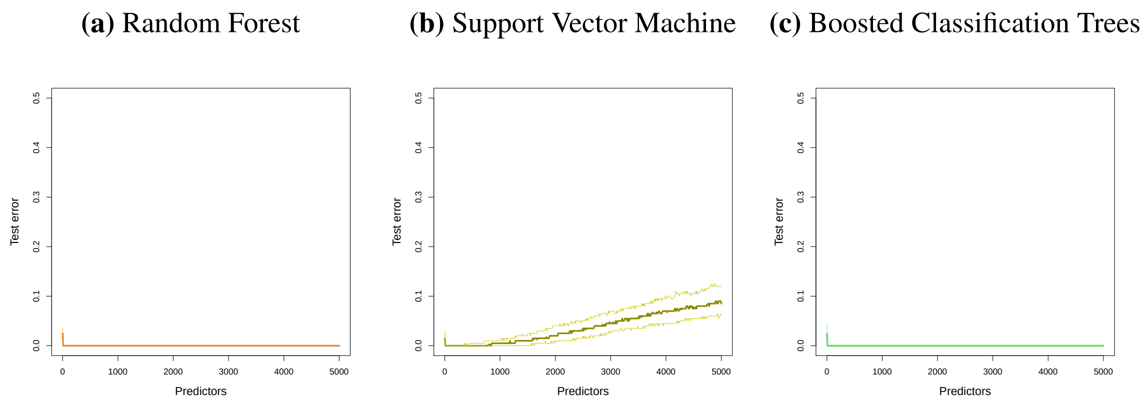

Figure 7. Test error for three classification methods from Scenario 1 ($n = 100$ for each class, $m = 10$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to three and $\boldsymbol{\mu}_1 = \mathbf{0}$) for $p_0 = 2$ to 5000 predictors. Thick lines represent the median classification error from 100 simulations; lighter lines show 10th and 90th percentiles.

2.2.2 Simulations for Scenario 2

Results from the second scenario were similar to the first except SVM performed worse and showed more evidence of noise accumulation (Figure 8). As in Scenario 1, the overall median test error was $< 3\%$ for RF and BCT and the test error for these methods peaked when $2 \leq p_0 \leq 4$ (Table 3). After this point, there was almost no test error for these methods. Contrary to the other two methods, SVM had a small initial peak in test error at $p_0 \leq 3$, which dropped back down then rose even higher as p grew. Table 3 shows the final value of test error for each method at $p_0 = 5000$. Based on a visual inspection of the plots in Figure 8, SVM appears more susceptible to the effects of noise accumulation in this scenario while RF and BCT do not. Once again, the median classification error for RF and BCT remains flat after $p_0 > 10$ whereas it steeply rises for SVM.

Table 3. Test error for values of p_0 in Scenario 2

p_0	Classification method s	Median	10th Percentile	90th Percentile
	Random forests	2.5	1.0	4.0
$p_0 \leq 10$*	Support vector machine	1.5	1.0	3.0
	Boosted classification trees	2.5	1.0	4.1
	Random forests	0.0	0.0	0.5
$p_0 = 5000$	Support vector machine	20.5	17.5	23.5
	Boosted classification trees	0.0	0.0	0.5

*Highest test error.

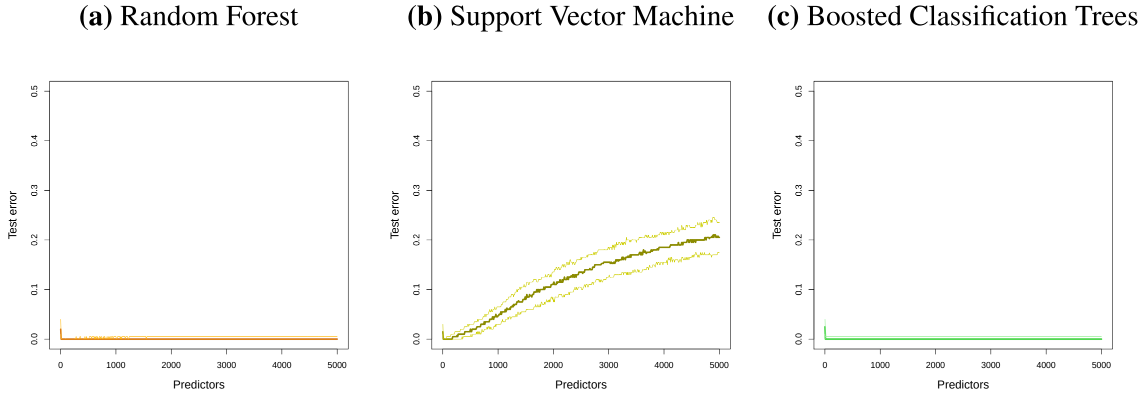

Figure 8. Test error for three classification methods from Scenario 2 ($n = 100$ for each class, $m = 6$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to three and $\boldsymbol{\mu}_1 = \mathbf{0}$) for $p = 2$ to 5000 predictors. Thick lines represent the median classification error from 100 simulations; lighter lines show 10th and 90th percentiles.

2.2.3 Simulations for Scenario 3

There was a decline in discriminative ability and a rise in noise accumulation for RF and especially SVM in Scenario 3 (Figure 9). Despite the increase in test error between this scenario and the previous ones, the RF classifier performed reasonably well with overall median test error $\leq 8\%$. Unlike in the previous scenarios, however, the median test error slowly rises as p increases, indicating noise accumulation is slowly growing as more predictors are added to the classification rule. The impact of noise accumulation is more extreme in the case of SVM as the median classification error precipitously climbs between $0 \leq p_0 \leq 1000$. In terms of test error, SVM did not behave as well; its overall median test error was $> 35\%$. BCT still performed at nearly an equivalent degree as in Scenarios 1 and 2; its overall median test error was $\leq 4\%$. Also, BCT continued to show no evidence of noise accumulation. Unlike previous scenarios, the highest test error did not occur when $p_0 < 5$ for RF and BCT. For $p_0 \leq 10$, SVM outperformed RF and BCT; the highest median test error for this method was 2.0%. The others were 2.5% and 3.0%, respectively (Table 4). When $p_0 > 10$, the test error grew for all simulations as p increased. At $p_0 = 5000$, BCT performed best (median test error, 3.0%) among the three methods, followed closely by RF (median test error, 7.5%) with SVM (median test error, 39.5%) far behind (Table 4).

Table 4. Test error for values of p_0 in Scenario 3

p_0	Classification method	Median	10th Percentile	90th Percentile
$p_0 \leq 10$*	Random forests	2.5	1.5	4.1
	Support vector machine	2.0	1.0	3.5
	Boosted classification trees	3.0	1.5	4.5
$p_0 = 5000$	Random forests	7.5	5.0	10.5
	Support vector machine	39.5	35.5	43.1
	Boosted classification trees	3.0	1.5	5.1

*Highest test error.

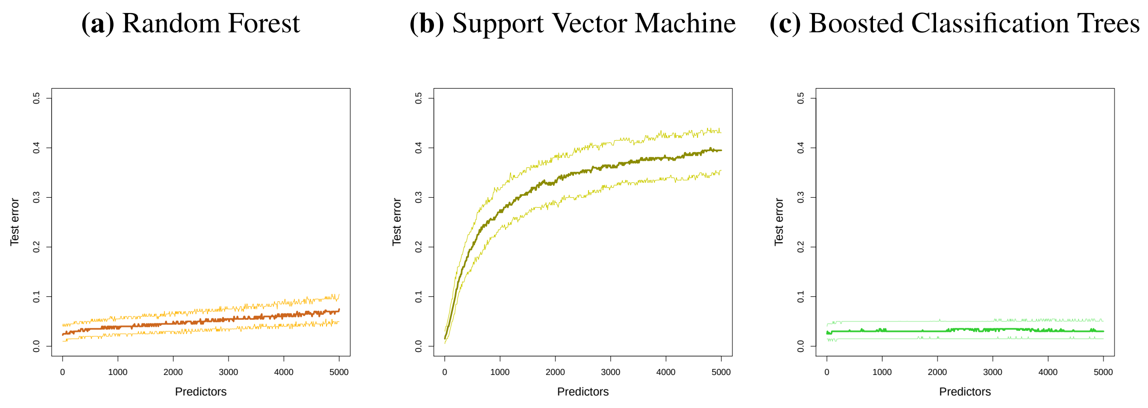

Figure 9. Test error for three classification methods from Scenario 3 ($n = 100$ for each class, $m = 2$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to three and $\boldsymbol{\mu}_1 = \mathbf{0}$) for $p = 2$ to 5000 predictors. Thick lines represent the median classification error from 100 simulations; lighter lines show 10th and 90th percentiles.

2.2.4 Simulations for Scenario 4

Scenario 4 proved to be a difficult simulation for all classification approaches (Figure 10) though the test error for SVM was slightly better in this scenario than the previous one. Overall, the median test error was $< 30\%$ for RF and BCT while it was $> 30\%$ for SVM. The test error peaked again at $2 \leq p_0 \leq 3$ for RF and BCT but at $p_0 = 5000$ for SVM. After the initial increase for $p_0 \leq 10$ (see Table 5 for the maximum test error for this range), test error decreased for all of the methods. The plots of median test error for $p_0 > 10$ differed for the three methods: classification error increased gradually for RF but was not as high as $p_0 = 2$; it escalated quickly for SVM, exceeding the initial jump; and it stayed fairly flat at about 10% for BCT. This behavior is indicative of the impact of noise accumulation in each approach. The BCT classifiers do not seem to be influenced by noise accumulation yet both RF and SVM show signs of its presence as the median classification error for these methods rapidly ascends as the number of predictors increases.

Table 5. Test error for specified values of p_0 in Scenario 4

p_0	Classification method	Median	10th Percentile	90th Percentile
$p_0 \leq 10$*	Random forests	28.0	24.5	33.0
	Support vector machine	24.5	20.5	28.1
	Boosted classification trees	25.5	21.0	29.0
$p_0 = 5000$	Random forests	19.5	15.0	24.1
	Support vector machine	35.0	31.0	39.0
	Boosted classification trees	10.0	8.0	13.5

*Highest test error.

(a) Random Forest **(b)** Support Vector Machine **(c)** Boosted Classification Trees

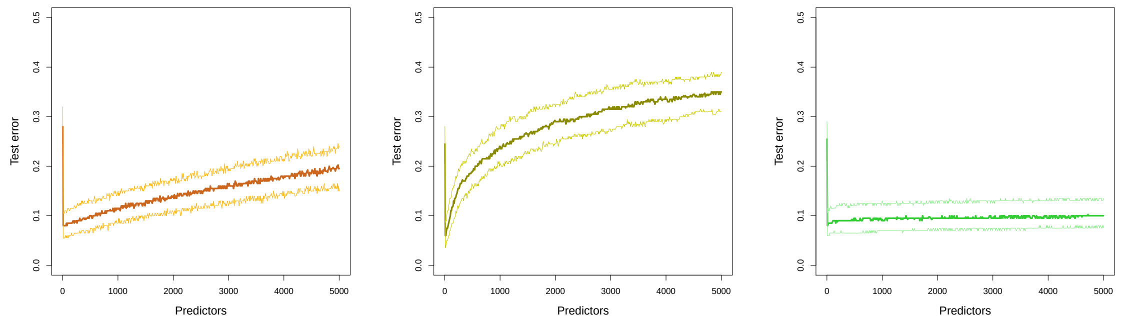

Figure 10. Test error for three classification methods from Scenario 4 ($n = 100$ for each class, $m = 10$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to one and $\boldsymbol{\mu}_1 = 0$) for $p = 2$ to 5000 predictors. Thick lines represent the median classification error from 100 simulations; lighter lines show 10th and 90th percentiles.

Of the classification methods we investigated, SVM appeared to be more susceptible to noise accumulation than RF or GBM. Figure 11 shows the results for SVM comparing Scenarios 3 and 4. Although the median test error for Scenario 4 starts off higher, the one for Scenario 3 has a steeper slope; the error for Scenario 4 catches up to Scenario 3 at about $p_0 = 400$ then exceeds it. This may suggest that signal strength is more important than magnitude for SVM as noise accumulates. For RF and GBM, test error increases in scenarios 1 through 4 such that the medians never cross.

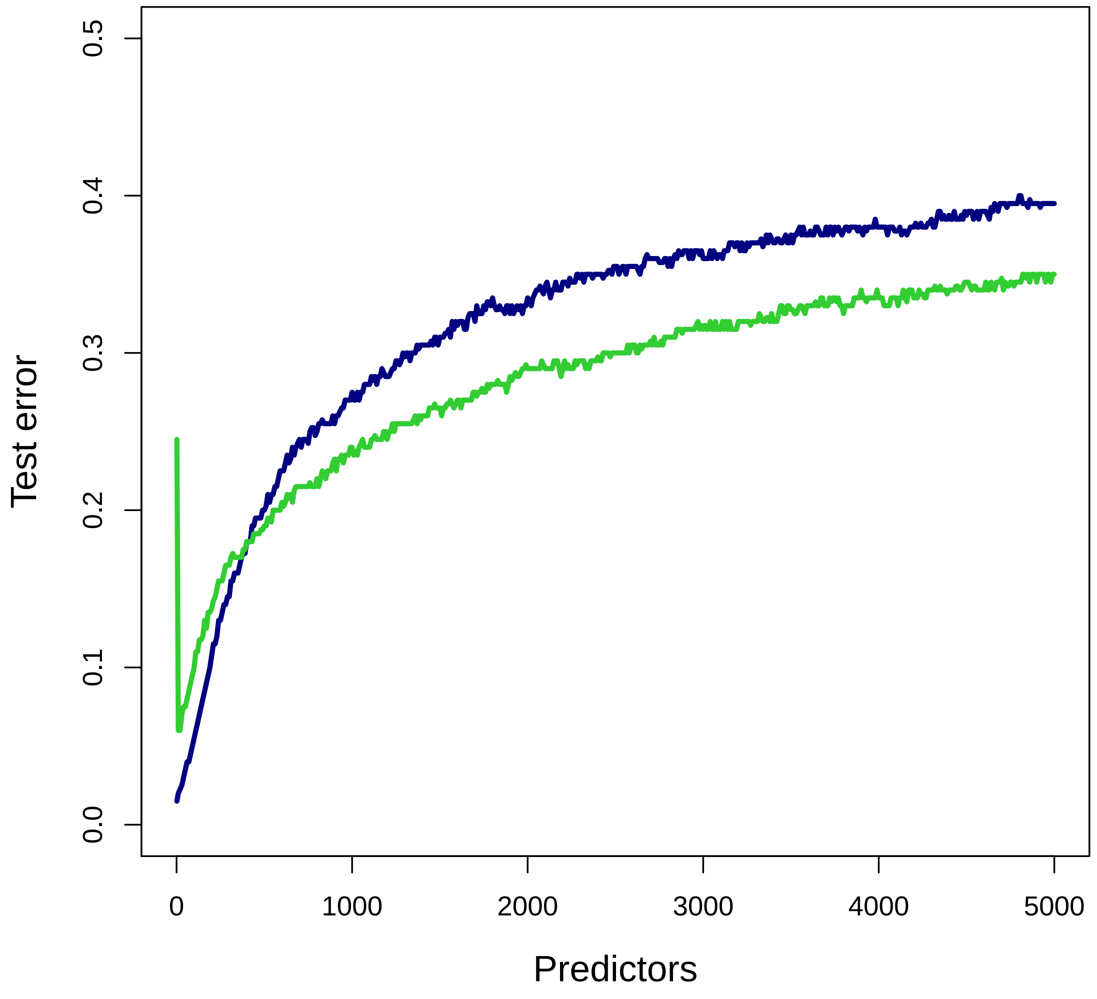

Figure 11. Test error for support vector machines from Scenarios 3 ($n = 100$ for each class, $m = 2$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to three and $\boldsymbol{\mu}_1 = \mathbf{0}$) and 4 ($n = 100$ for each class, $m = 10$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to one and $\boldsymbol{\mu}_1 = 0$) for $p = 2$ to 5000 predictors. The median classification error from 100 simulations is shown by the navy line for Scenario 3 and the green line for Scenario 4.

Previous plots summarize the performance of classifiers for each scenario and approach developed on the training datasets then applied to the test datasets. It is also informative to know how well the classifiers built on the training data fit the underlying distribution of the data. If the classifier follows the noise too closely, it will overfit the data and not produce accurate estimates of the response for new observations. Figure 12 shows the difference in median training minus test error for Scenario 4 for each method used for classification. For RF, the error ranged between 0 and 10% and the classifiers' performance improves on the test compared with the training data. We found the RF classifiers consistently performed as good or better on the test datasets for all scenarios and values of p. It increased from 0.0% at $p_0 = 2$ to 8.5% at $p_0 = 5000$. BCT tended to produce classifiers that worked well on the test data, only slight overfitting Scenarios 3 and 4. Further, the difference between training and test datasets was fairly constant across p; it was about -9.5% for Scenario 4. By contrast, the SVM classifiers overfit the test data in all scenarios, worsened as p increased. For Scenario 4, the difference in median error ranged between -0.5% at $p_0 = 10$ and -30.0% for $p_0 = 5000$.

In "Challenges of Big Data analysis," Fan et al. discuss the impact of the massive size of Big Data on traditional computing infrastructure. The simulated data we used had 200*5000 = 100,000 data points and we ran the simulation 100 times for each method for every 10 predictors, meaning the classifiers developed for each simulation and method used 10,000,000 data points. Even though this magnitude of data seems large, it is not nearly the scale of billions or trillions of data points that the authors consider. Still, the simulations were computationally intensive and the processing time for BCT especially was substantial. The SVM computations took 1.8 days, RF 15.8 days, and BCT over 2 months. For this project, we parallelized creation of the data but not the construction and validation of the classifiers. It is likely that the simulations could have been spread out over nodes in the computing cluster we employed but not all methods could have been sped up this way. For example, the sequential nature of BCT prevents the algorithm from being parallelized.

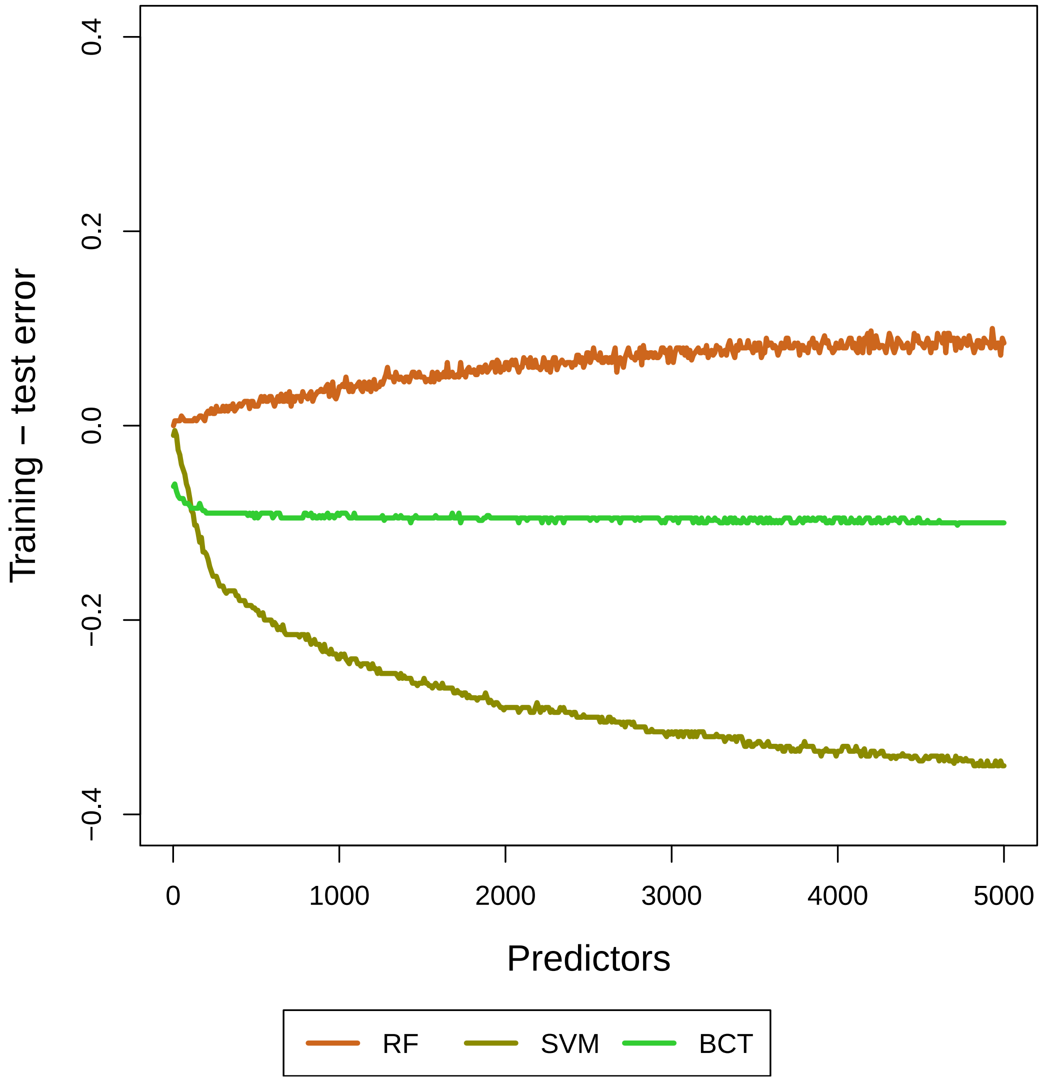

Figure 12. Difference in training and test error for three classification methods for Scenario 4 ($n = 100$ for each class, $m = 10$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to one and $\boldsymbol{\mu}_1 = \mathbf{0}$) for $p = 2$ to 5000 predictors. The brown line shows the median classification error from 100 simulations from random forests, olive line from support vector machine, and green line for boosted classification trees.

3 Characterization of Noise Accumulation

In this chapter, we examine characteristics of noise accumulation using RF to construct the classifiers. We continue our assessment of noise accumulation, now focusing on this approach as our classification method both because of its popularity for analyzing high dimensional data but also due to its performance in simulations from the previous chapter. We wanted a classification method that showed evidence of being impacted by noise accumulation but had reasonable discriminative ability. We also wanted one that could produce results in a realistic amount of time. Unlike the support vector machine approach, RF demonstrated good discriminative ability in the most challenging scenarios and, in contrast to BCT, it produced results in a viable amount of computing time.

To characterize noise accumulation, we conducted simulations to explore different aspects that might impact its magnitude or behavior. We varied sample size, signal strength, and ratio of predictors to signal strength using signal magnitude equal to one. Subsequently, we repeated these simulations, modifying the magnitude of the nonzero elements to be $\frac{1}{\sqrt{m}}$ to see what impact weak signal of the same strength would have. We chose this value for the nonzero elements in our second set of simulations because we wanted to explore more challenging settings in which distance between the mean locations of classes was fixed in all dimensions. See Appendix A.2 for details on how we chose $\frac{1}{\sqrt{m}}$.

As before, simulations were batch processed in R version 3.4.0 using the OHSU Exacloud computational environment with nodes running CentOS Linux 7.[10] RF classification was conducted using randomForest (4.6-12).[11] We used the default settings for the package; see code in Appendix A.1 for details.

As before, we simulated data for two classes from a multivariate normal distribution with equal sample size n in each class, p predictors, and an identity covariance matrix:

Class 1: $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n \sim \mathrm{MV}_p(\boldsymbol{\mu}_1, \boldsymbol{I}_p)$

Class 2: $\boldsymbol{Y}_1, \dots, \boldsymbol{Y}_n \sim \mathrm{MV}_p(\boldsymbol{\mu}_2, \boldsymbol{I}_p)$

where $\boldsymbol{\mu}_1 = \mathbf{0}$ and $\boldsymbol{\mu}_2$ was defined to be sparse with m nonzero elements and the remaining entries equal to zero. For the first set of simulations, we fixed $\boldsymbol{\mu}_1 = \mathbf{0}$ and the value of the m the nonzero elements of $\boldsymbol{\mu}_2$ to be equal to one and sample size n was divided evenly between the two classes. We conducted the following scenarios:

$S_1(1)$ Increase sample size

Sample size was assessed at $n =$200, 500, 1000, and 5000 with $m = 10$.

$S_1(2)$ Modify signal strength

The number of nonzero elements in Class 2 was varied for $m = 5$, 10, 20, and 30. That is, $\boldsymbol{\mu}_2$ = (μ_{21}, μ_{22}, 0, . . . , 0) for $m = 2$; $\boldsymbol{\mu}_2$ = (μ_{21}, μ_{22}, . . ., μ_{29}, μ_{210}, 0, . . ., 0) for $m = 10$; and so on.

$S_1(3)$ Vary signal strength and total predictors

The ratio of m to the maximum number predictors p_{max} ($m : p_{max}$) was fixed while the signal strength was increased. $m : p_{max}$ was assessed for $10 : 5000$, $30 : 15000$, $50 : 25000$, $70 : 35000$, and $90 : 45000$.

For each scenario, a RF-based classification rule was developed for 30 training and test datasets. A classifier was developed for predictors from $p = 2$ to 5000 to categorize observations into one of the two classes except for the third scenario where p_{max} was used. As in Chapter 2, the classifiers developed on the training sets for each value of p were applied to corresponding test datasets to predict class membership. We gauged the discriminative power for each classifier by calculating the median classification error with 10th and 90th percentile bounds by comparing the predicted categorization to its true classification. We repeated these initial simulations changing the value of the nonzero elements of $\boldsymbol{\mu}_2$ from one to $\frac{1}{\sqrt{m}}$ with no additional modifications. This second set of simulations are $S_2(1)$, $S_2(2)$, $S_2(3)$, respectively.

3.1 Simulations increasing sample size

Figure 13 shows results from $S_1(1)$; the discriminative ability of the RF classifier improved as sample size increased. As we saw in the previous section, there was a spike in test error for $p_0 \leq 10$. For all n, the maximum error was highest for $p_0 = 2$, where the average median error was 27.4%. At $p_0 = 5000$, the median test error steadily decreased from 19.5% (10 – 90% percentile: 15.0 – 24.1%) when $n = 200$ to 8.3% (10 – 90% percentile: 7.7 – 8.9%) when $n = 5000$. Likewise, noise accumulation diminished as $n \to 5000$ with the plot almost flat at $n = 5000$. Results from $S_2(1)$ demonstrate much poorer performance of the classifier clearly affected by noise accumulation (Figure 14). When $p_0 \leq 10$, the median test error was greatest at $p_0 = 2$ for each value of n. The median test error for $p_0 > 10$ decreased slightly as n increased yet it was more than 25% higher than in $S_1(1)$. Noise accumulation is present for $n = 200$, 500, and 1000 but recedes somewhat for $n = 5000$. Table 6 shows median test error for $p = 2$ and 5000 with 10th and 90th percentiles for both $S_1(1)$ and $S_2(1)$.

3.2 Simulations modifying signal strength

Results from $S_1(2)$ illustrate both noise accumulation and test error decreased steadily with increasing signal strength (Figure 15). Median test error peaked at approximately 28%

when $p_0 = 2$ (Table 7). Disregarding $p_0 \leq 10$, the slope of the median error was steepest for simulations with the lowest value of m then evened out as signal strength increased. At $p_0 = 5000$ (Table 7), the median test error was highest at $m = 5$ – 31.3% (10 – 90% percentile: 27.5 – 35.5%) – and lowest at $m = 30$, 3.8% (10 – 90% percentile: 1.5 – 5.5%).

Table 6. Test error for values of p_0 for random forest simulations increasing sample size

Scenario	p_0	n	Median	10th Percentile	90th Percentile
$S_1(1)$	$p_0 \leq 10$*	200	28.0	24.5	33.0
		500	27.2	24.2	30.4
		1000	27.4	25.9	29.4
		5000	27.4	26.7	28.2
	$p_0 = 5000$	200	19.5	15.0	24.1
		500	12.4	10.4	14.8
		1000	10.5	9.4	11.4
		5000	8.3	7.7	8.9
$S_2(1)$	$p_0 \leq 10$*	200	44.5	41.7	49.6
		500	45.8	42.6	48.8
		1000	46.3	44.5	48.3
		5000	45.9	45.1	47.2
	$p_0 = 5000$	200	50.0	46.5	53.0
		500	47.6	44.8	50.4
		1000	45.6	43.1	46.9
		5000	39.7	39.0	40.9

*Highest test error.

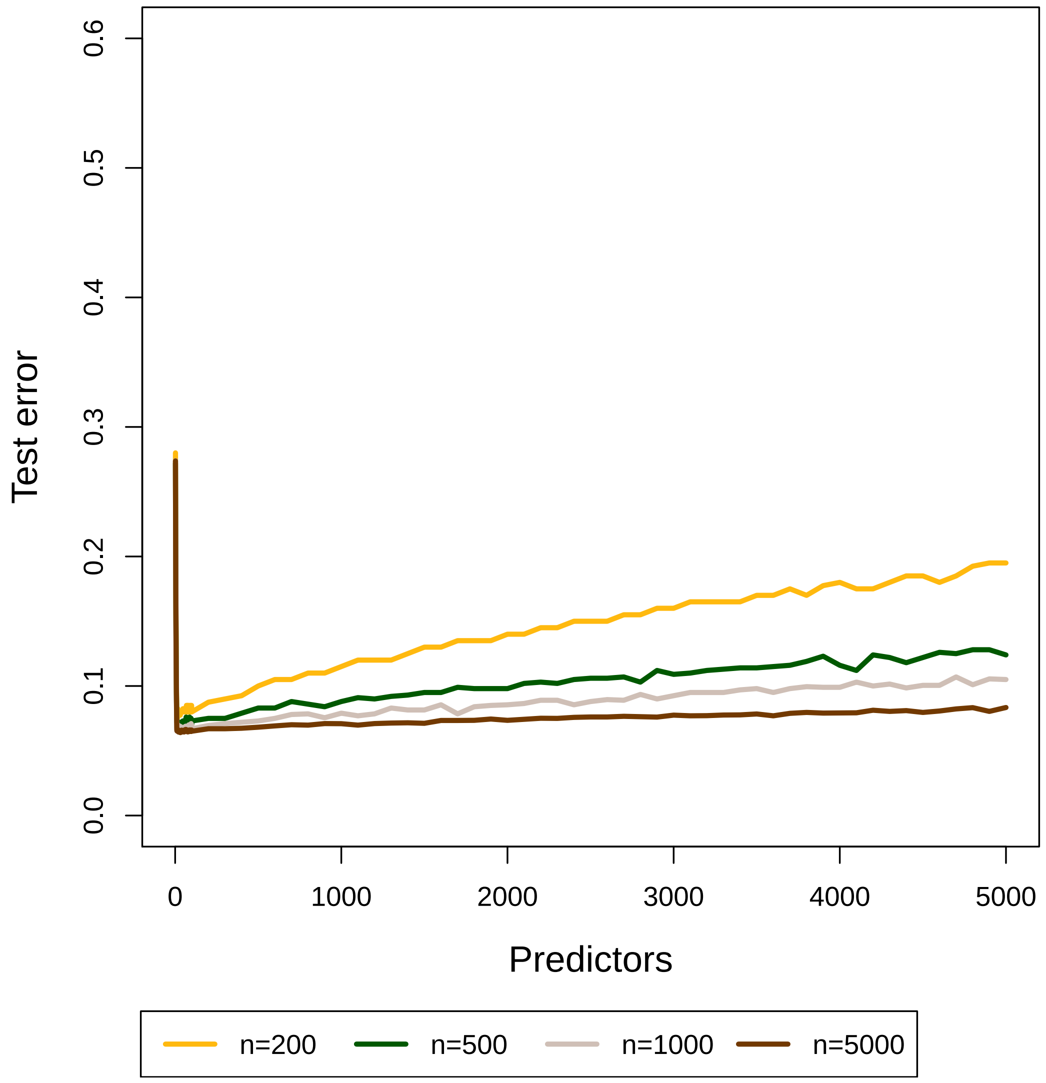

Figure 13. Test error for Scenario $S_1(1)$ (sample size varying between $n = 200$ and 5000 split equally between classes, $m = 10$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to one and $\boldsymbol{\mu}_1 = \mathbf{0}$) for $p = 2$ to 5000 predictors. Lines represent the median classification error from 100 simulations; gold line shows classification for $n = 200$, dark green for $n = 500$, grey for $n = 1000$, and brown for $n = 5000$.

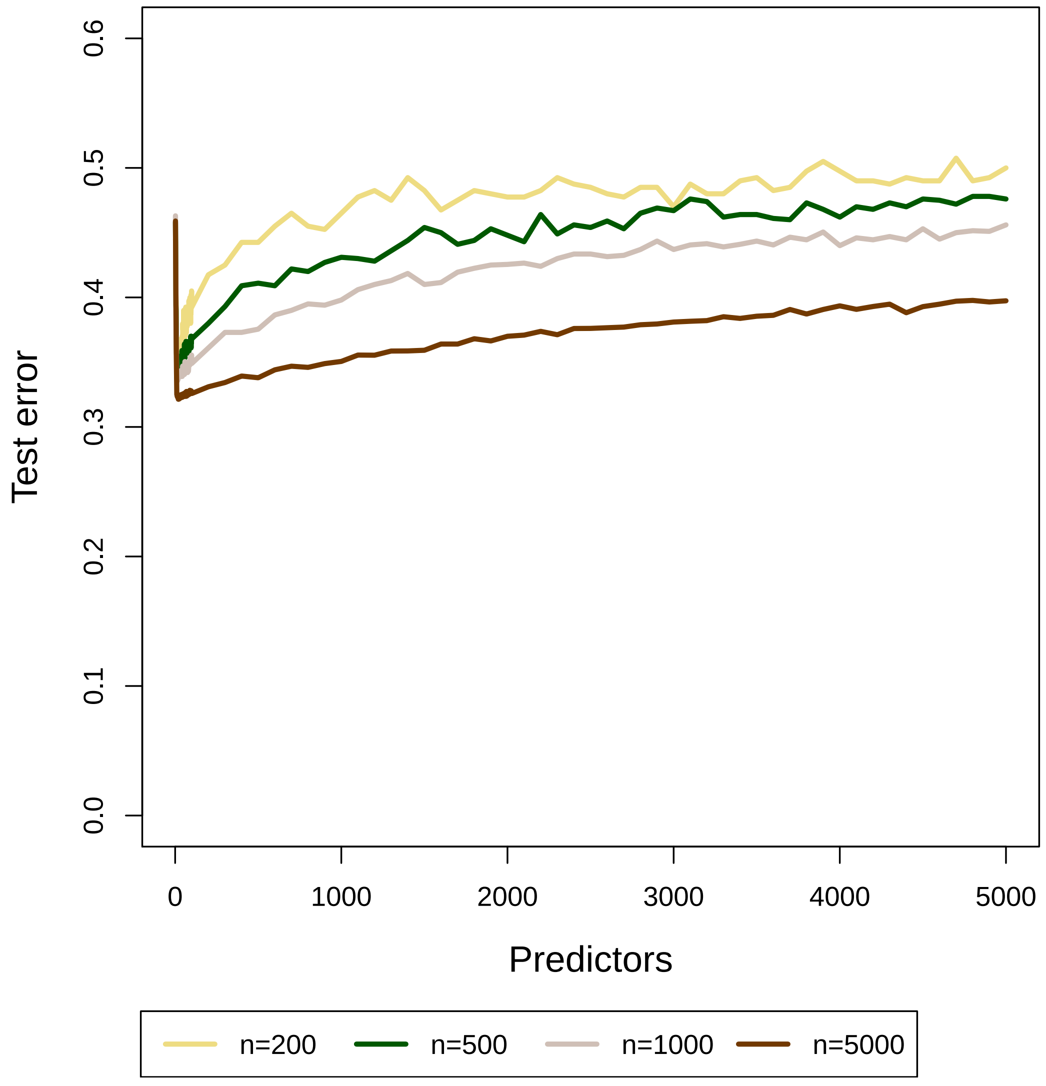

Figure 14. Test error for Scenario $S_2(1)$ (sample size varying between $n = 200$ and 5000 split equally between classes, $m = 10$ nonzero elements for $\boldsymbol{\mu}_2$ each equal to $\frac{1}{\sqrt{m}}$ and $\boldsymbol{\mu}_1 = \mathbf{0}$) for $p = 2$ to 5000 predictors. Lines represent the median classification error from 100 simulations; yellow line shows classification for $n = 200$, dark green for $n = 500$, grey for $n = 1000$, and brown for $n = 5000$.

As before, test error rose when the value of the nonzero elements of $\boldsymbol{\mu}_2$ changed from one to $\frac{1}{\sqrt{m}}$ in $S_2(2)$ (Figure 16). In these simulations, the median test error peaked between 42.8% and 49.0% at $p_0 = 2$ for $p_0 \leq 10$ (Table 7). Between $p_0 > 10$ and $p_0 = 5000$, the slope of the error tended to increase rapidly for all $S_2(2)$ simulations then remain fairly constant above 40%. This does not appear to be due to abatement of noise accumulation, which is evident from the sharply increasing slope after an initial drop for $p < 10$, but from a ceiling effect in the performance of the classifiers. There was not much difference in error or noise acumulation for any of the simulations.

Table 7. Test error for values of p_0 for random forest simulations modifying signal strength

Scenario	p_0	m	Median	10th Percentile	90th Percentile
$S_1(2)$	$p_0 \leq 10$*	5	29.0	24.0	31.5
		10	28.0	24.5	33.0
		20	27.3	24.5	31.2
		30	28.5	23.0	33.5
	$p_0 = 5000$	5	31.3	27.5	35.6
		10	19.5	15.0	24.1
		20	8.0	6.5	10.5
		30	3.8	1.5	5.5
$S_2(2)$	$p_0 \leq 10$*	5	42.8	37.9	46.6
		10	46.8	40.5	51.0
		20	47.8	45.0	51.6
		30	49.0	44.4	54.6
	$p_0 = 5000$	5	49.0	44.9	55.2
		10	49.8	46.5	55.1
		20	49.0	44.0	52.7
		30	49.0	46.0	51.1

*Highest test error.

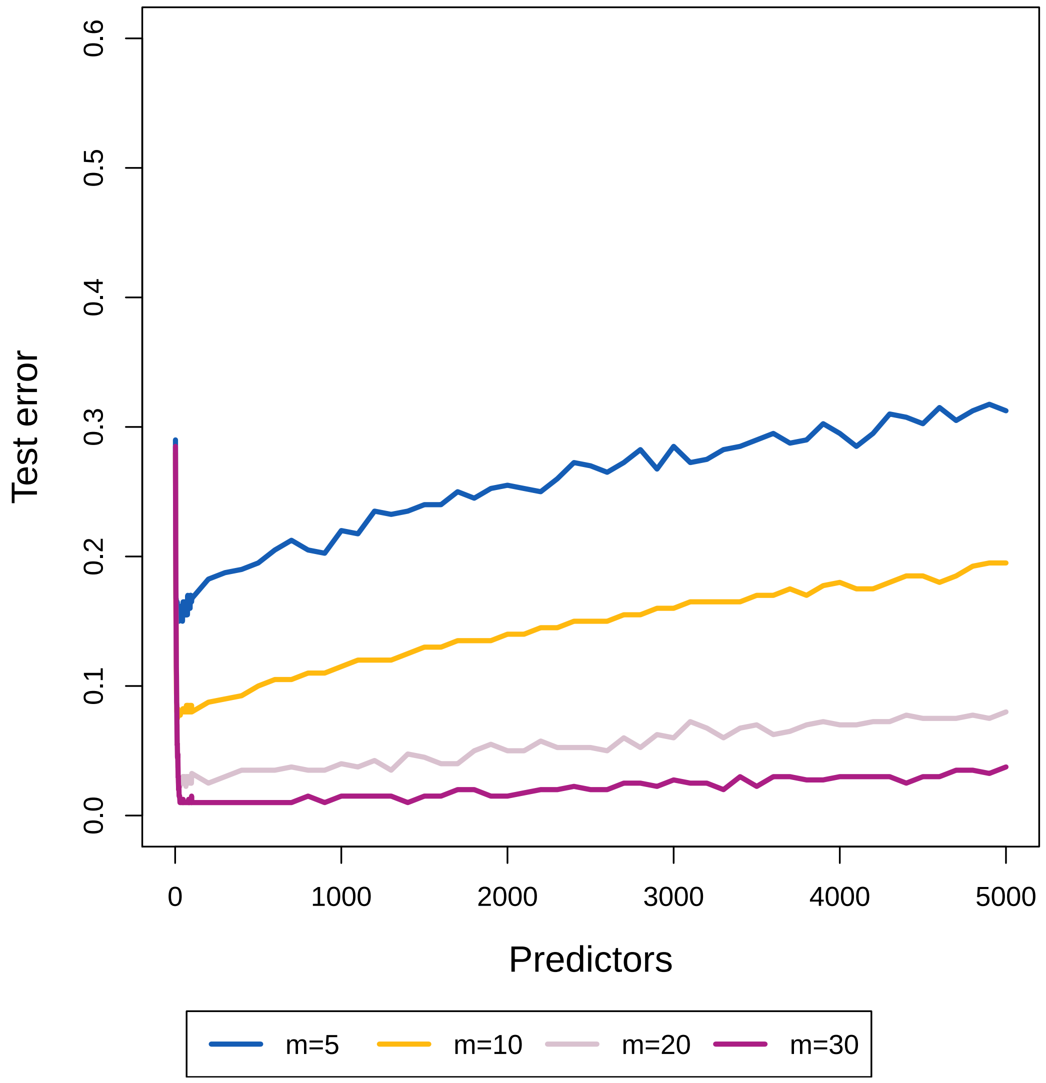

Figure 15. Test error for Scenario $S_1(2)$ ($n = 100$ for each class, signal strength varying between $m = 5$ and 30 nonzero elements for $\boldsymbol{\mu}_2$ each equal to one and $\boldsymbol{\mu}_1 = \mathbf{0}$) for $p = 2$ to 5000 predictors. Lines represent the median classification error from 100 simulations where blue shows classification for $m = 5$, gold for $m = 10$, lilac for $m = 20$, and fuchsia for $m = 30$.

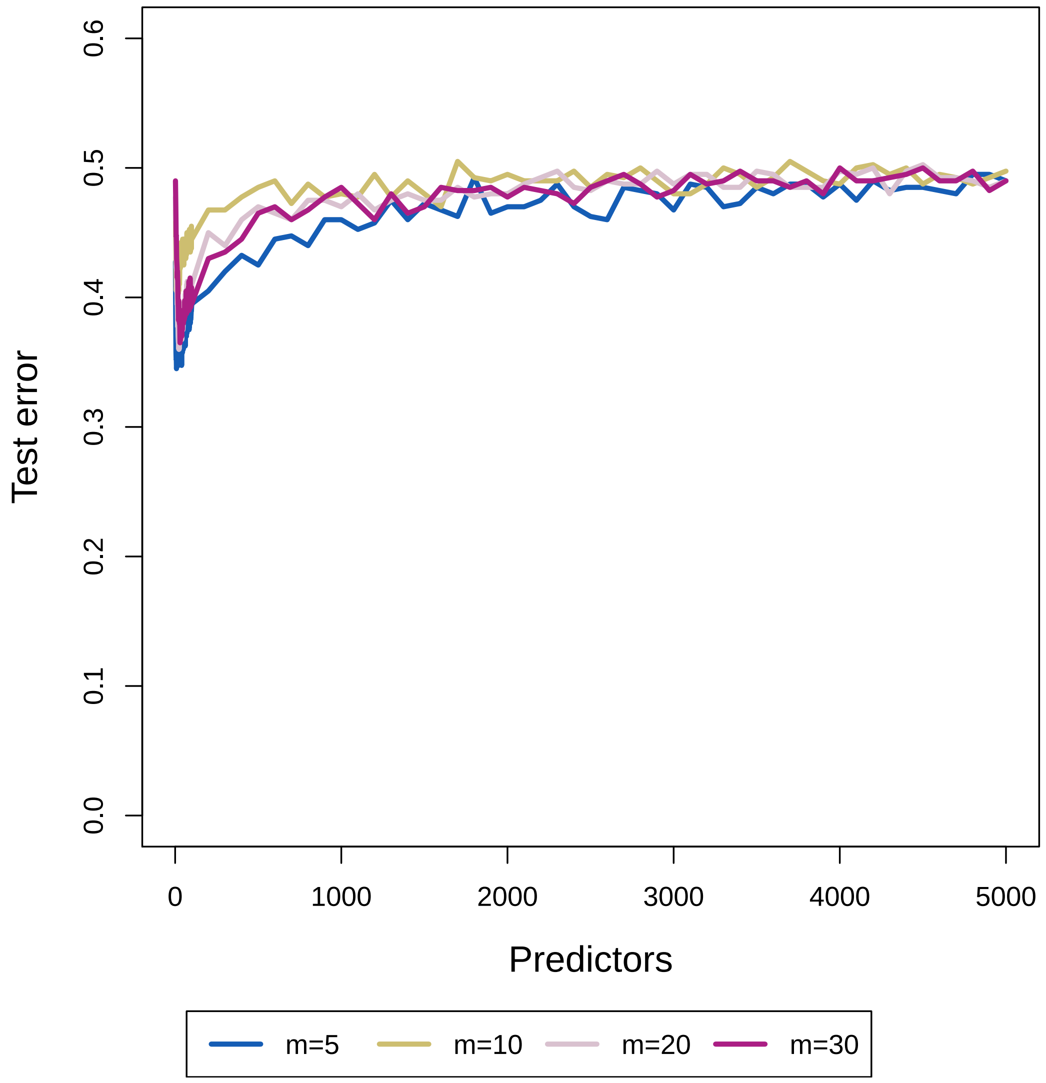

Figure 16. Test error for Scenario $S_2(2)$ ($n = 100$ for each class, signal strength varying between $m = 5$ and 30 nonzero elements for $\boldsymbol{\mu}_2$ each equal to $\frac{1}{\sqrt{m}}$ and $\boldsymbol{\mu}_1 = \mathbf{0}$) for $p = 2$ to 5000 predictors. Lines represent the median classification error from 100 simulations where blue shows classification for $m = 5$, yellow for $m = 10$, lilac for $m = 20$, and fuchsia for $m = 30$.

3.3 Simulations varying signal strength and total predictors

The median test error for simulations in $S_1(3)$ showed similar behavior to $S_1(2)$. That is, it decreased with greater signal strength as p increased (Figure 17). The median error peaked at $p_0 = 2$ where it was approximately 28.5% for all $m : p_{max}$ simulations (Table 8). We showed previously that the median test error for the baseline case ($m = 10$, p_{max} = 5000) at $p_0 = 5000$ was 19.5% (10 – 90% percentile: 15.0 – 24.1%). At $m = 30$ and p_{max} = 15000, it reduced substantially to 9.0% (10 – 90% percentile: 6.4 – 11.5%). As evident in Figure 17), the median error for the remaining simulations gradually reduced towards zero; Table 8 summarizes it for $p_0 = 5000$. The nadir, when $m = 90$ and p_{max} = 45000, had median error of 1.5% (10 – 90% percentile: 0.5 – 3.0%). As in the previous scenario $S_1(2)$, the slope of the median test error tended to be sharpest for simulations with the lower values of m in the $m : p_{max}$ simulations and leveled out as m increased. This pattern of diminishing median classification error with increasing m proportional to p also describes the behavior of noise accumulation in this series of simulations. By contrast, the median error for the $S_2(3)$ simulations initially increased after $p_0 > 10$ then remained steady at about 50% for every simulation except $m : p_{max} = 10 : 5000$ (Figure 18). Also, these classifiers performed uniformly poorly where the median test error was above 30% for all simulations and values of p. Table 8 displays the test error for $p_0 = 2$ where it was highest for $p_0 \leq 10$ and $p_0 = 5000$. Noise accumulation affected all $S_2(3)$ scenarios. Noise accumulation affected classification almost immediately after $p_0 > m$ as in the $S_2(2)$ simulations. As in that case, noise accumulation appears to level out due to a ceiling effect in the median classification error.

Scenarios $S_1(1)$ and $S_1(2)$, where the nonzero elements of $\boldsymbol{\mu}_2$ are equal to one, behaved as conjectured. As sample size and signal increased, the discriminative ability of the RF classifiers improved. For $S_1(1)$, this improvement is anticipated from asymptotic theory.

As $n \to \infty$, the estimator should converge to the true value of the parameter being estimated thus accuracy of classification should rise. Indeed, classification error was less than 10% even for the 90th percentile when sample size was $n = 5000$ but even $n = 500$ performed markedly better than the base scenario of $n = 200$ and comparably to the larger sample cases. As for $S_1(2)$, common sense dictates that classification would improve as signal strength grows since the classifier draws on this information to differentiate between groups. Increasing the number of nonzero elements in of $\boldsymbol{\mu}_2$ seemed to have a greater positive impact on discriminative ability than increasing sample size. For $S_1(2)$, the median classification error dropped to less than 5% when $m = 30$, lower even than $n = 5000$ in $S_1(1)$. Results from the third scenario – increasing the amount of signal while keeping the ratio of predictors constant – were less expected. They demonstrated that discriminative ability improved with signal strength even as the number of predictors grew proportionally. The median test error fell between $m : p_{max} = 10 : 5000$ (the base scenario) and $30 : 15000$. At $m : p_{max} = 90 : 45000$, the error was less than 2%. The result from this simulation is notable because it implies that discriminative ability will be high with sufficient signal regardless of the number of predictors.

When the nonzero elements of $\boldsymbol{\mu}_2$ were reduced to $\frac{1}{\sqrt{m}}$, the classifiers' performance deteriorated considerably. Results from $S_2(1)$, in which the RF classifier was constructed with different sample sizes, showed the best discriminative ability of these scenarios. As in $S_1(1)$, the classifier improved as sample size increased. The test error dropped from classifying half of observations incorrectly at $p_0 = 5000$ when $n = 200$ to 39.7% when $n = 5000$. The test error from scenarios $S_2(2)$ and $S_2(3)$ look equally poor for nearly all simulations where the classifiers do no better than chance.

These simulations suggest that noise accumulation may threaten accurate separation of data into two classes when sample size is small relative to the number of predictors, signal strength is low, and signal magnitude is weak. We explored extreme cases of these

Table 8. Test error for values of p_0 for random forest simulations varying signal strength and total predictors

Scenario	p_0	$m : p_{max}$	Median	10th Percentile	90th Percentile
$S_1(3)$	$p_0 \leq 10$*	10 : 5000	28.0	24.5	33.0
		30 : 15000	29.0	25.4	32.0
		50 : 25000	28.5	26.0	32.1
		70 : 35000	28.5	25.4	32.5
		90 : 45000	27.8	23.9	33.1
	$p_0 = 5000$	10 : 5000	19.5	15.0	24.1
	$p_0 = 15000$	30 : 15000	9.0	6.4	11.6
	$p_0 = 25000$	50 : 25000	4.8	3.0	7.0
	$p_0 = 35000$	70 : 35000	3.0	1.0	4.6
	$p_0 = 45000$	90 : 45000	1.5	0.5	3.0
$S_2(3)$	$p_0 \leq 10$*	10 : 5000	45.8	41.4	49.7
		30 : 15000	49.0	46.8	53.6
		50 : 25000	50.3	44.9	55.8
		70 : 35000	49.8	45.5	53.7
		90 : 45000	49.5	46.0	52.0
	$p_0 = 5000$	10 : 5000	46.5	42.0	53.1
	$p_0 = 15000$	30 : 15000	52.0	47.0	56.0
	$p_0 = 25000$	50 : 25000	49.8	44.4	54.1
	$p_0 = 35000$	70 : 35000	50.5	46.5	54.3
	$p_0 = 45000$	90 : 45000	48.8	44.2	56.0

*Highest test error.

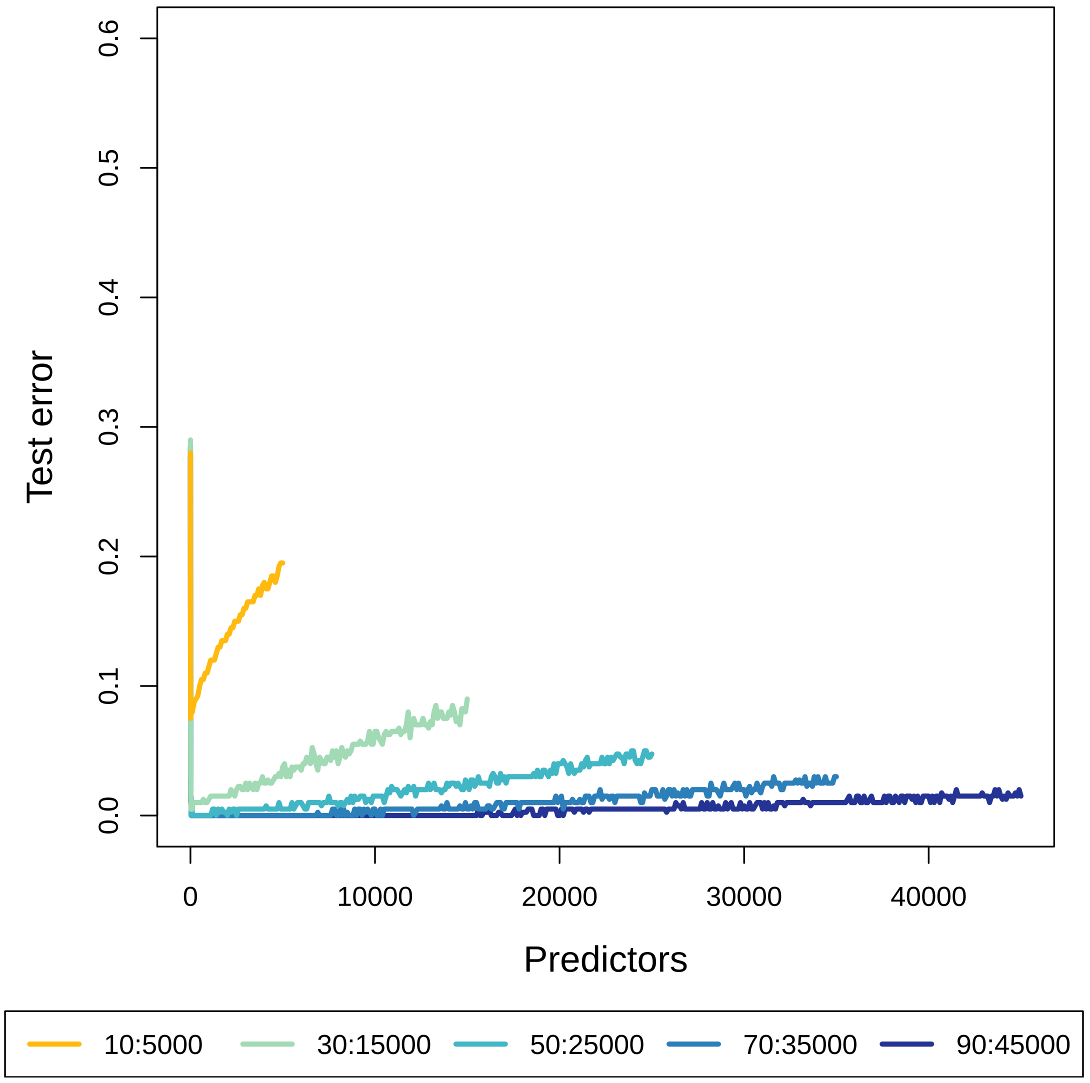

Figure 17. Test error for Scenario $S_1(3)$ ($n = 100$ for each class, $m : p_{max}$ varying between $10 : 5000$ and $90 : 45000$ where m are the nonzero elements for $\boldsymbol{\mu}_2$ each equal to one, p_{max} is the total number of predictors, and $\boldsymbol{\mu}_1 = \mathbf{0}$). Lines represent the median classification error from 100 simulations where gold shows classification for $m : p_{max} = 10 : 5000$, light green for $m : p_{max} = 30 : 15000$, turquoise for $m : p_{max} = 50 : 25000$, blue for $m : p_{max} = 70 : 35000$, and navy for $m : p_{max} = 90 : 45000$.

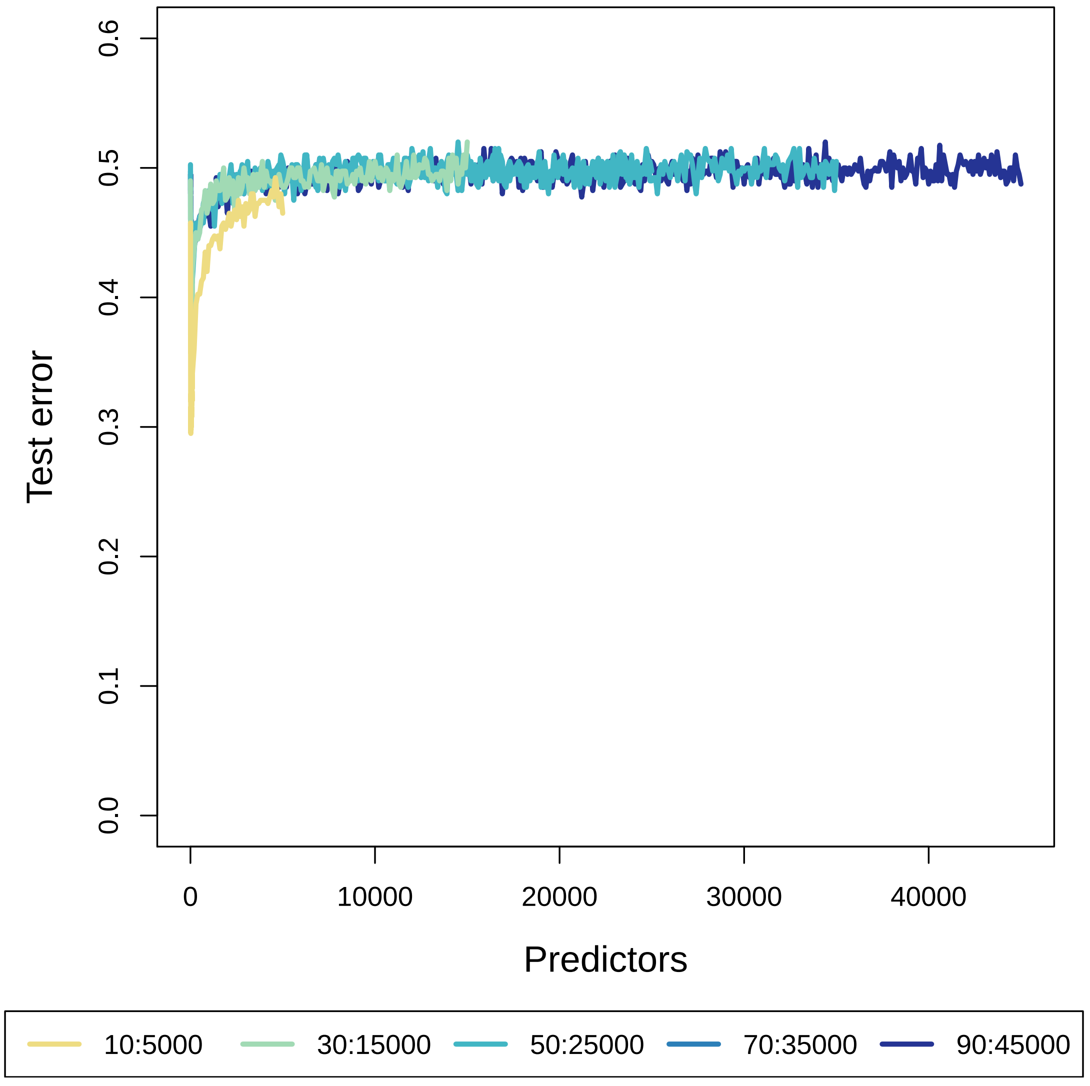

Figure 18. Test error for Scenario $S_2(3)$ ($n = 100$ for each class, $m : p_{max}$ varying between $10 : 5000$ and $90 : 45000$ where m are the nonzero elements for $\boldsymbol{\mu}_2$ each equal to $\frac{1}{\sqrt{m}}$, p_{max} is the total number of predictors, and $\boldsymbol{\mu}_1 = \mathbf{0}$). Lines represent the median classification error from 100 simulations where yellow shows classification for $m : p_{max} = 10 : 5000$, light green for $m : p_{max} = 30 : 15000$, turquoise for $m : p_{max} = 50 : 25000$, blue for $m : p_{max} = 70 : 35000$, and navy for $m : p_{max} = 90 : 45000$.

situations but they may be unlikely in practice. Our findings suggest that as long as the signal magnitude is sufficiently large, it is possible to counteract noise accumulation by collecting large sample, selecting classes that are as different as possible, or – ideally – both. It is likely that increasing sample size is the most modifiable way to avoid this problem. In settings where the magnitude of signal is weak, good discriminative ability may not be possible.

4 Total Signal Index

While running simulations for the previous chapters, we wondered how investigators would evaluate whether noise accumulation was a threat to analysis. In practice, without the luxury of knowing true values, it is difficult to detect. Such concerns led us to consider ways to quantify it. Signal to noise ratio (SNR) is a popular measure in science and engineering to compare the level of signal compared to background noise in order to assess the amount of useful information. Although there are many definitions for this ratio, a common one in statistics is the quotient of the signal mean and the standard deviation of the noise:

$$\mathrm{SNR} = \frac{\mu}{\sigma} \tag{1}$$

where μ is the signal mean and σ is the standard deviation of the noise. Higher ratios mean there is more useful information (i.e., signal) relative to erroneous data (i.e., noise).

One way to interpret signal in the context of classification is as the distance between the means of the two classes. Variance is the noise around each of those means; it indicates how much data will spread out from the mean, thus also whether the classes will overlap. The greater the distance between two means and the smaller their variances, the greater the signal and easier it is to distinguish them. We combined the definition of SNR in (1) with the idea of signal being the distance between class means to develop an index which could summarize the ability of a classifier to differentiate between two groups.

4.1 Theoretical total signal index

Define two independent groups X_1 and X_2 from a multivariate distribution with sample size n_1 and n_2, respectively, such that

$$\begin{aligned} \boldsymbol{X}_1 :& \boldsymbol{x}_{11}, \ldots, \boldsymbol{x}_{1n_1} \text{ with } \boldsymbol{X}_1 \sim \mathrm{MV}(\boldsymbol{\mu}_1, \boldsymbol{\Lambda}_1) \\ \boldsymbol{X}_2 :& \boldsymbol{x}_{21}, \ldots, \boldsymbol{x}_{2n_2} \text{ with } \boldsymbol{X}_2 \sim \mathrm{MV}(\boldsymbol{\mu}_2, \boldsymbol{\Lambda}_2) \end{aligned}$$

where $\boldsymbol{\Lambda}_1$ and $\boldsymbol{\Lambda}_2$ are diagonal covariance matrices; $\boldsymbol{x}_{1k_1}$ and $\boldsymbol{x}_{2k_2}$ are p-dimensional vectors for $k_1 = 1, 2, \ldots, n_1$ and $k_2 = 1, 2, \ldots, n_2$. We define the total signal index (TSI) as the Euclidean distance of the difference in SNR between the two classes:

$$TSI = \sqrt{\sum_{i=1}^{p} \left(\frac{\mu_{1i}}{\sigma_{1i}} - \frac{\mu_{2i}}{\sigma_{2i}} \right)^2} \tag{2}$$

As defined in (2), we expect TSI to increase with greater distance between the two classes and drop as the distance decreases. Equivalently, TSI will be higher with more signal and lower with less signal.

4.2 Empirical total signal index

We propose an empiric version of TSI, TSI$_e$, that can be estimated from sample data. We use the sample mean and variance for each group in lieu of the population parameters. Let Y_1 and Y_2 be independent groups from a multivariate distribution with sample size m_1 and m_2

$$\begin{aligned} \boldsymbol{Y}_1 :& \boldsymbol{y}_{11}, \ldots, \boldsymbol{y}_{1m_1} \text{ with } \boldsymbol{Y}_1 \sim \mathrm{MV}(\bar{\boldsymbol{y}}_1, \boldsymbol{s}_1^2) \\ \boldsymbol{Y}_2 :& \boldsymbol{y}_{21}, \ldots, \boldsymbol{y}_{2m_2} \text{ with } \boldsymbol{Y}_2 \sim \mathrm{MV}(\bar{\boldsymbol{y}}_2, \boldsymbol{s}_2^2) \end{aligned}$$

where $\boldsymbol{s}_1^2$ and $\boldsymbol{s}_2^2$ are diagonal covariance matrices; $\boldsymbol{y}_{1k_1}$ and $\boldsymbol{y}_{2k_2}$ are p-dimensional vectors for $k_1 = 1, 2, \ldots, m_1$ and $k_2 = 1, 2, \ldots, m_2$. Then we define TSI$_e$ as

$$TSI_e = \sqrt{\sum_{i=1}^{p} \left(\frac{\bar{y}_{1i}}{s_{1i}} - \frac{\bar{y}_{2i}}{s_{2i}} \right)^2} \tag{3}$$

As with TSI, the value of TSI$_e$ be higher with more signal and lower with less signal. Although (3) does not directly account for correlation, we will show that TSI$_e$ can be used to indirectly evaluate data with correlated predictors.

4.3 Simulations for total signal indices

Next, we consider simulations to assess the performance of TSI$_e$. We randomly sampled data from multivariate normal distributions such that $\boldsymbol{s}_1 = 1$ and $\boldsymbol{s}_2 = 1$. We considered scenarios with $\bar{\boldsymbol{y}}_1$ and $\bar{\boldsymbol{y}}_2$ summarized in Table 9. Thus,

$\bar{\boldsymbol{y}}_1 = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, \ldots, 0)$ and

$\bar{\boldsymbol{y}}_2 = \mathbf{0} = (0, \ldots, 0)$

in the first scenario and

$\bar{\boldsymbol{y}}_1 = (3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 0, \ldots, 0)$ and

$\bar{\boldsymbol{y}}_2 = (-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, \ldots, 0)$

in the second. For each scenario, we generated 100 datasets and considered sample sizes $n =$200, 500, 1000, 5000, and 10000 split equally between the two groups. We computed minimum, median, and maximum TSI$_e$ for every $p_0 = 1, 2, \ldots, 5000$ predictors. We also calculated the theoretical value of TSI for each scenario for corresponding values of n, p_0, μ_{1i}, μ_{2i}, σ_1, and σ_2. Figures 19 through 23 show median TSI$_e$ plotted for each predictor with bands for minimum and maximum values. The lefthand graph in each figure shows $1 \le p_0 \le 50$, displaying TSI$_e$ tightly fitting TSI for this range of p. This series of figures demonstrates that TSI$_e$ traces TSI well when p is small. As the number of predictors increases (displayed on the righthand side of the figures), TSI$_e$ drifts upwards, overestimating TSI. Ironically, this divergence appears to be due to noise accumulation, which is magnified when each term of TSI$_e$ is summed. When interpreted together, the plots indicate that the difference between the empirical and theoretical indices can be ameliorated either by augmenting the number of samples in the groups or increasing the distance between $\bar{y}_{1i}$ and $\bar{y}_{2i}$. From Figures 19 through 23, the gap between TSI$_e$ and TSI shrinks as n grows and it almost disappears at $n = 10000$. In each of the figures, the separation between TSI$_e$ and TSI is reduced for Scenario 2 compared to Scenario 1, suggesting that a larger distance between $\bar{y}_{1i}$ and $\bar{y}_{2i}$ may help TSI$_e$ better estimate TSI.

Table 9. Scenarios for Total Signal Index simulations

	m_1	m_2	Value of m_1	Value of m_2
1	10	0	1	0
2	10	10	3	-1

m_1, number of nonzero elements in $\bar{\boldsymbol{y}}_1$; m_2, number of nonzero elements in$\bar{\boldsymbol{y}}_2$.

We repeated Scenario 1 as described above with a slight modification. Unlike in the previous simulations of TSI, the nonzero elements in the first group that comprised the signal were *not* defined as the first 10 entries of the vector of means for $\bar{y}_{1i}$. Rather, these elements were scattered randomly throughout the length of $\bar{y}_{1i}$ for TSI$_e$ and, analogously, μ_{1i} for TSI. As before, we calculated the median TSI$_e$ and plotted it with the theoretical TSI. Figures 24 through 28 show the results of these simulations. These plots demonstrate that TSI$_e$ follows the theoretical value of TSI even when signal is not sorted as the first m nonzero elements of the sample mean. Further, the distance between these two measures decreases as n increases.

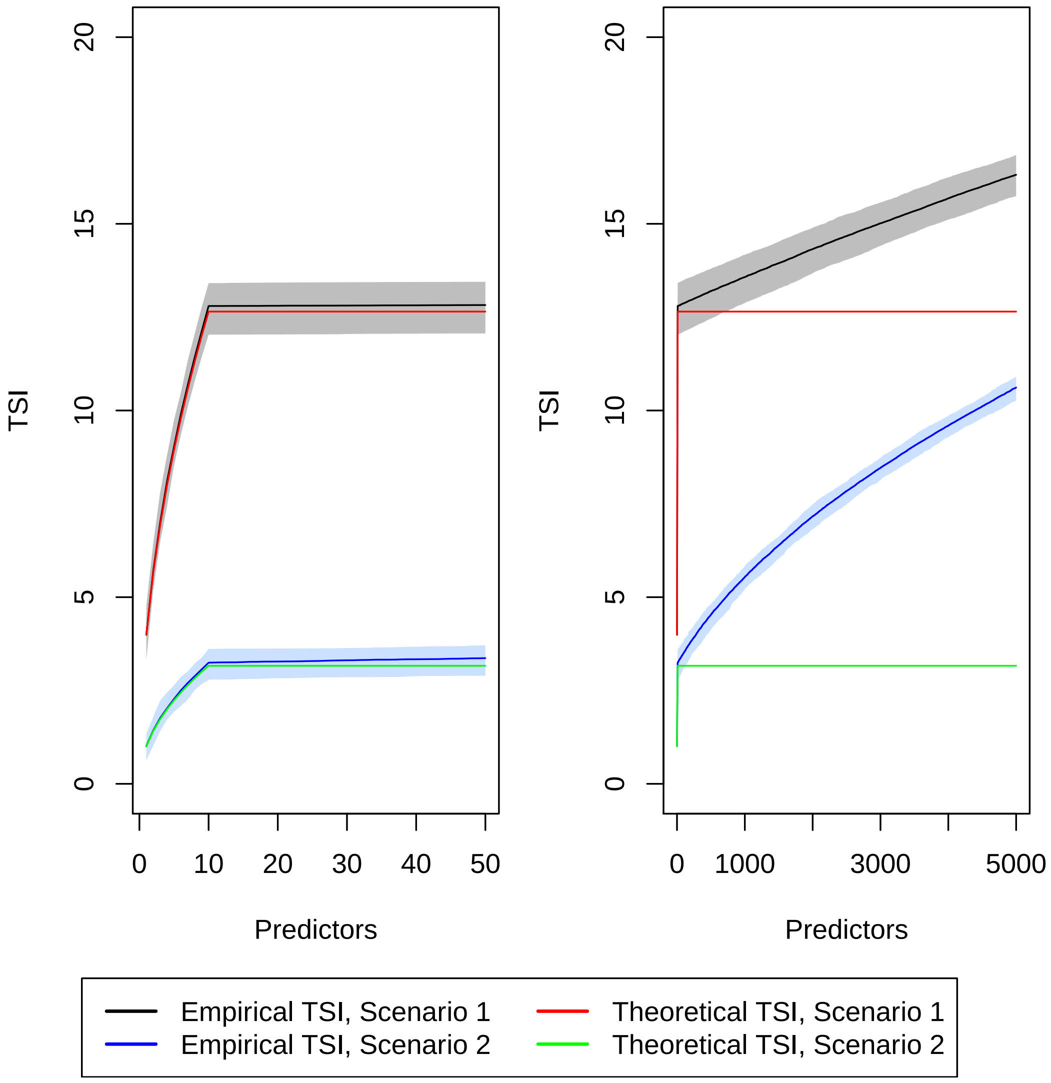

Figure 19. Median empirical total signal index (TSI) by number of predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 200$. Black line is median empirical TSI for Scenario 1 ($\bar{\boldsymbol{y}}_1$ has value one for the first 10 nonzero elements then zero afterward and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); red line is corresponding TSI. Blue line is median empirical TSI for Scenario 2 ($\bar{\boldsymbol{y}}_1 = 3$ and $\bar{\boldsymbol{y}}_2 = -1$ for the first 10 nonzero elements then zero afterward); green line is corresponding TSI. Grey and blue bands show the minimum and maximum values for empirical TSI in Scenarios 1 and 2, respectively.

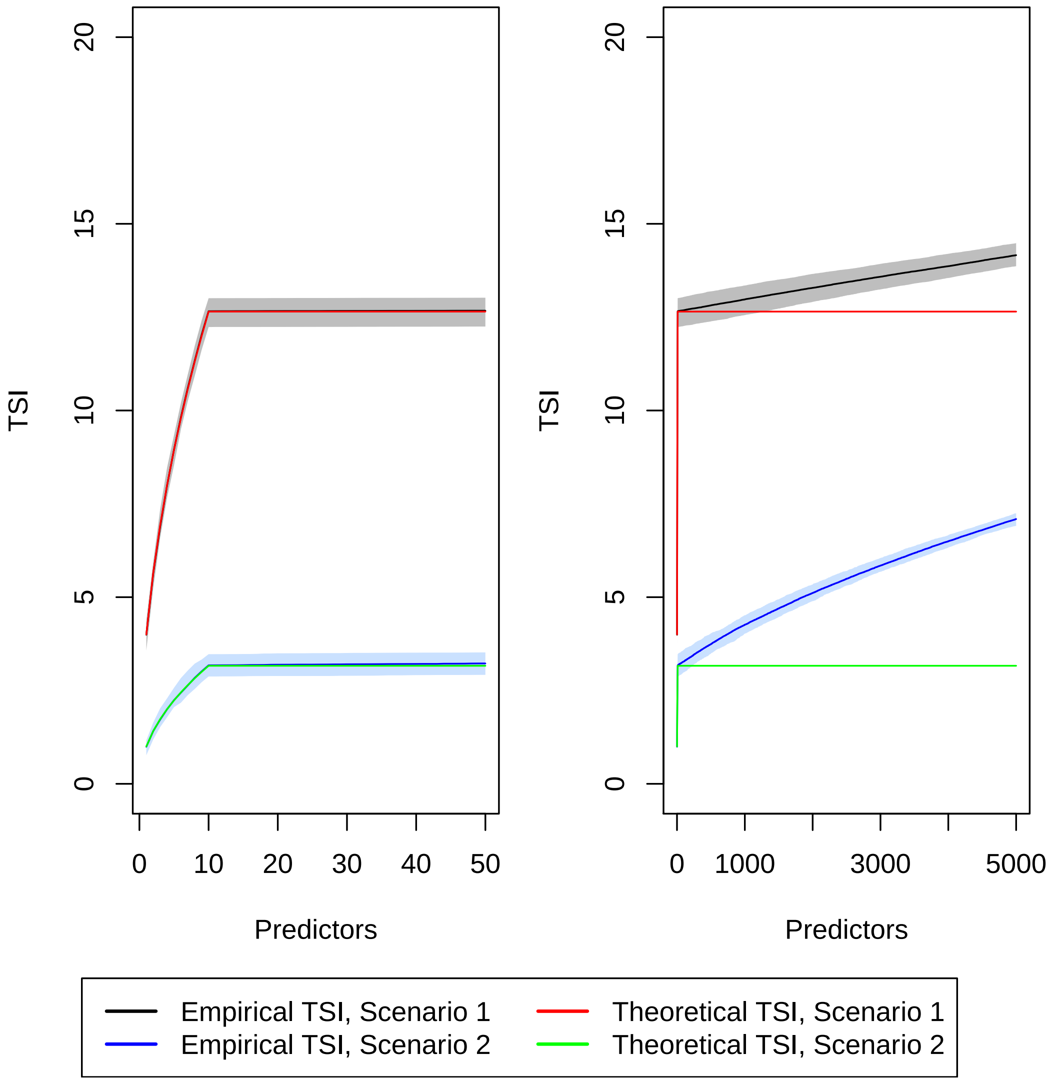

Figure 20. Median empirical total signal index (TSI) by number of predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 500$. Black line is median empirical TSI for Scenario 1 ($\bar{\boldsymbol{y}}_1$ has value one for the first 10 nonzero elements then zero afterward and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); red line is corresponding TSI. Blue line is median empirical TSI for Scenario 2 ($\bar{\boldsymbol{y}}_1 = 3$ and $\bar{\boldsymbol{y}}_2 = -1$ for the first 10 nonzero elements then zero afterward); green line is corresponding TSI. Grey and blue bands show the minimum and maximum values for empirical TSI in Scenarios 1 and 2, respectively.

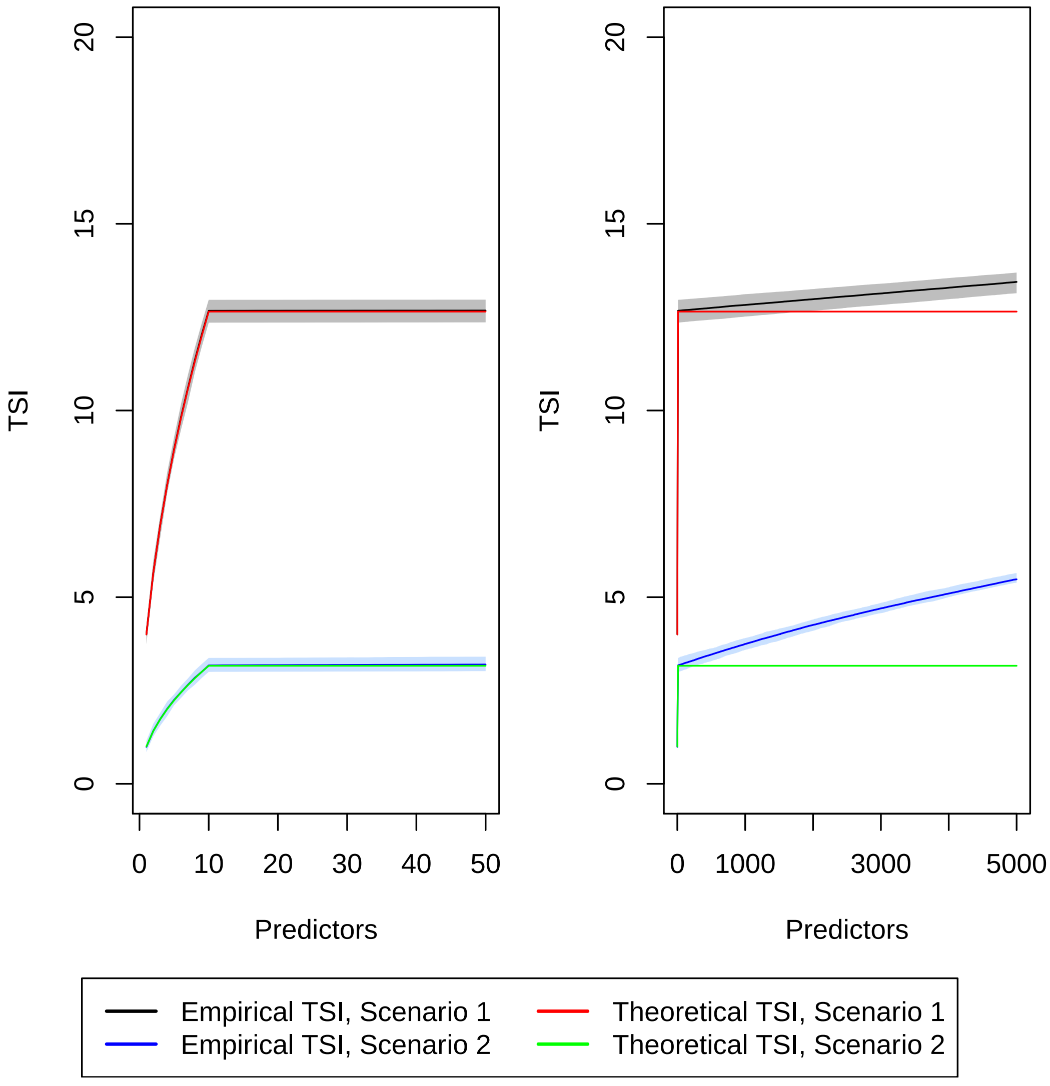

Figure 21. Median empirical total signal index (TSI) by number of predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 1000$. Black line is median empirical TSI for Scenario 1 ($\bar{\boldsymbol{y}}_1$ has value one for the first 10 nonzero elements then zero afterward and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); red line is corresponding TSI. Blue line is median empirical TSI for Scenario 2 ($\bar{\boldsymbol{y}}_1 = 3$ and $\bar{\boldsymbol{y}}_2 = -1$ for the first 10 nonzero elements then zero afterward); green line is corresponding TSI. Grey and blue bands show the minimum and maximum values for empirical TSI in Scenarios 1 and 2, respectively.

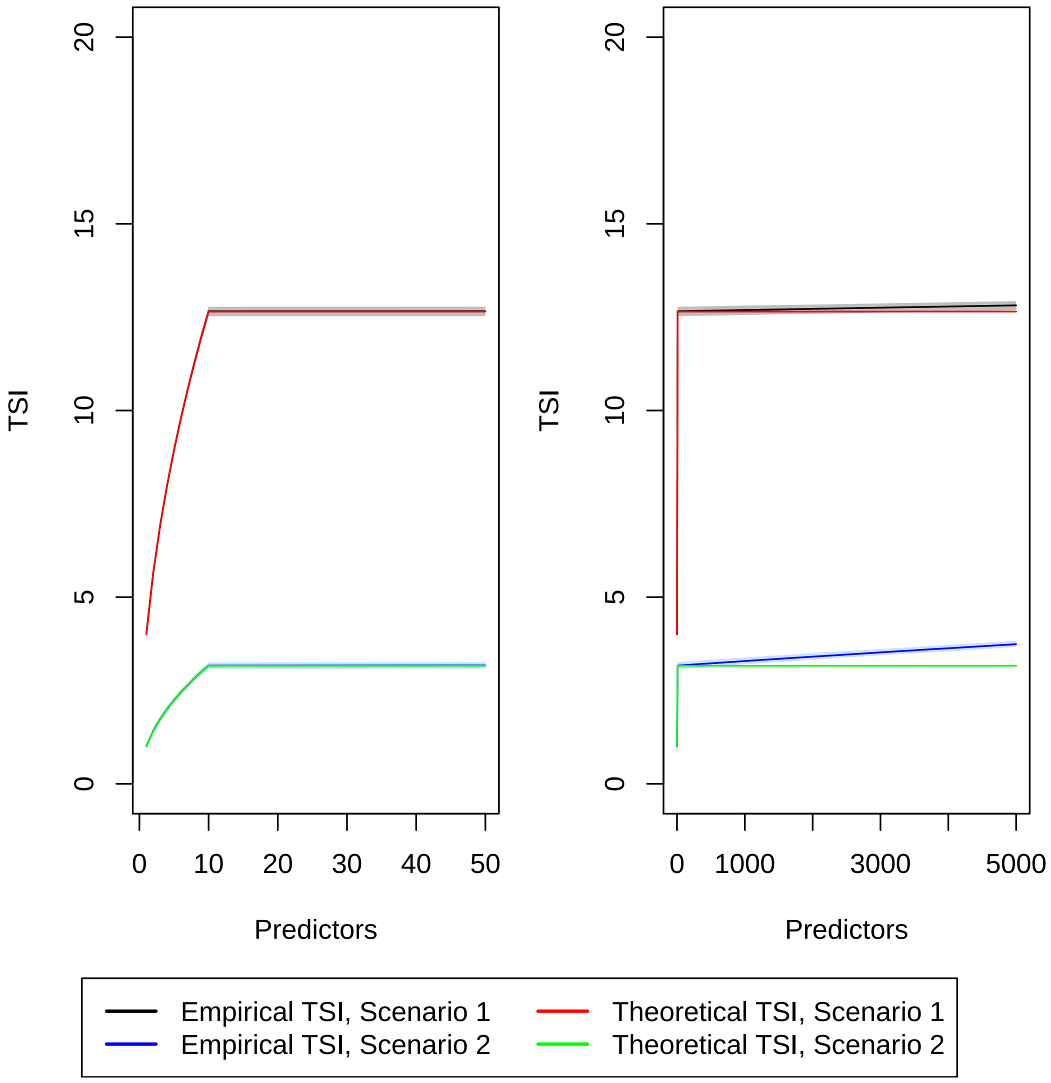

Figure 22. Median empirical total signal index (TSI) by number of predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 5000$. Black line is median empirical TSI for Scenario 1 ($\bar{\boldsymbol{y}}_1$ has value one for the first 10 nonzero elements then zero afterward and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); red line is corresponding TSI. Blue line is median empirical TSI for Scenario 2 ($\bar{\boldsymbol{y}}_1 = 3$ and $\bar{\boldsymbol{y}}_2 = -1$ for the first 10 nonzero elements then zero afterward); green line is corresponding TSI. Grey and blue bands show the minimum and maximum values for empirical TSI in Scenarios 1 and 2, respectively.

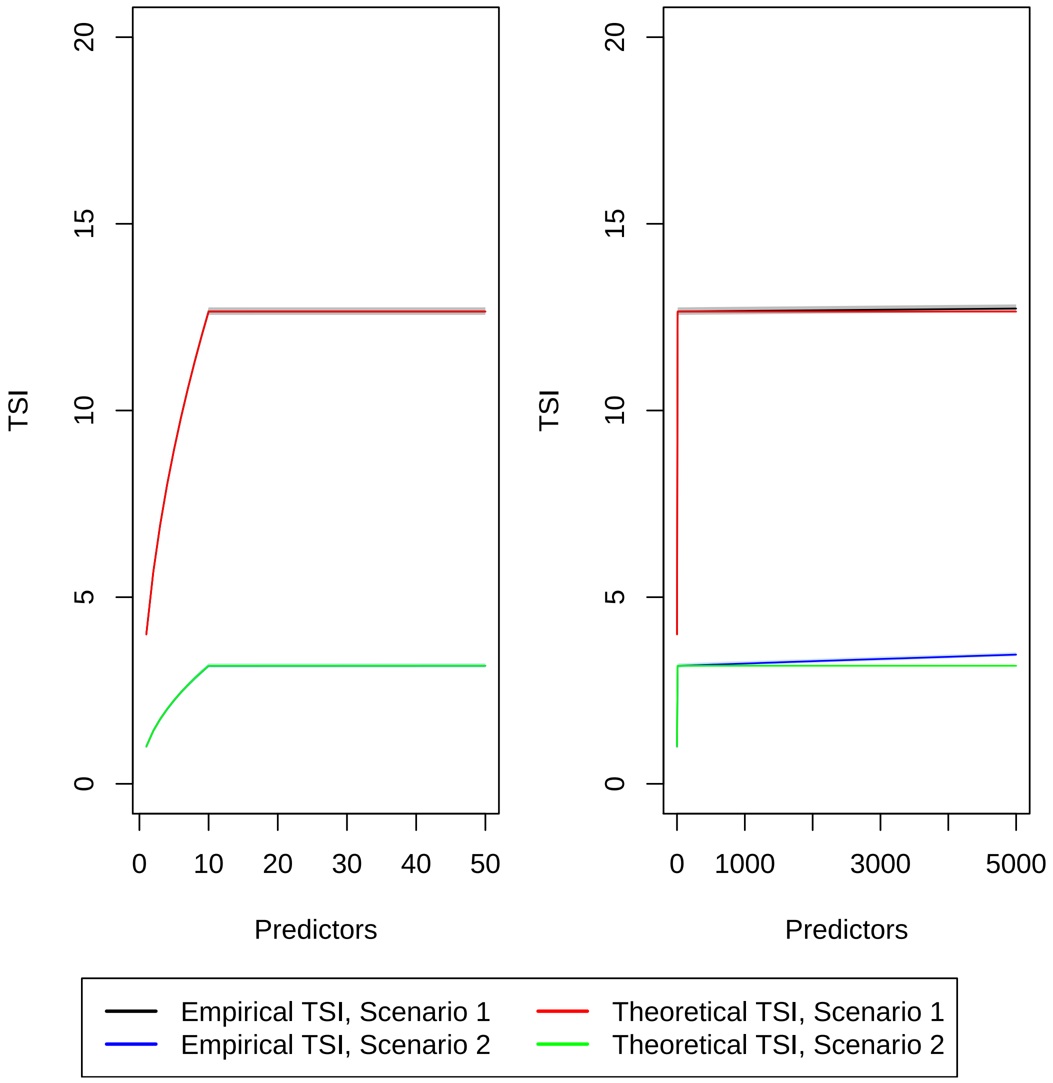

Figure 23. Median empirical total signal index (TSI) by number of predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 10000$. Black line is median empirical TSI for Scenario 1 ($\bar{\boldsymbol{y}}_1$ has value one for the first 10 nonzero elements then zero afterward and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); red line is corresponding TSI. Blue line is median empirical TSI for Scenario 2 ($\bar{\boldsymbol{y}}_1 = 3$ and $\bar{\boldsymbol{y}}_2 = -1$ for the first 10 nonzero elements then zero afterward); green line is corresponding TSI. Grey and blue bands show the minimum and maximum values for empirical TSI in Scenarios 1 and 2, respectively.

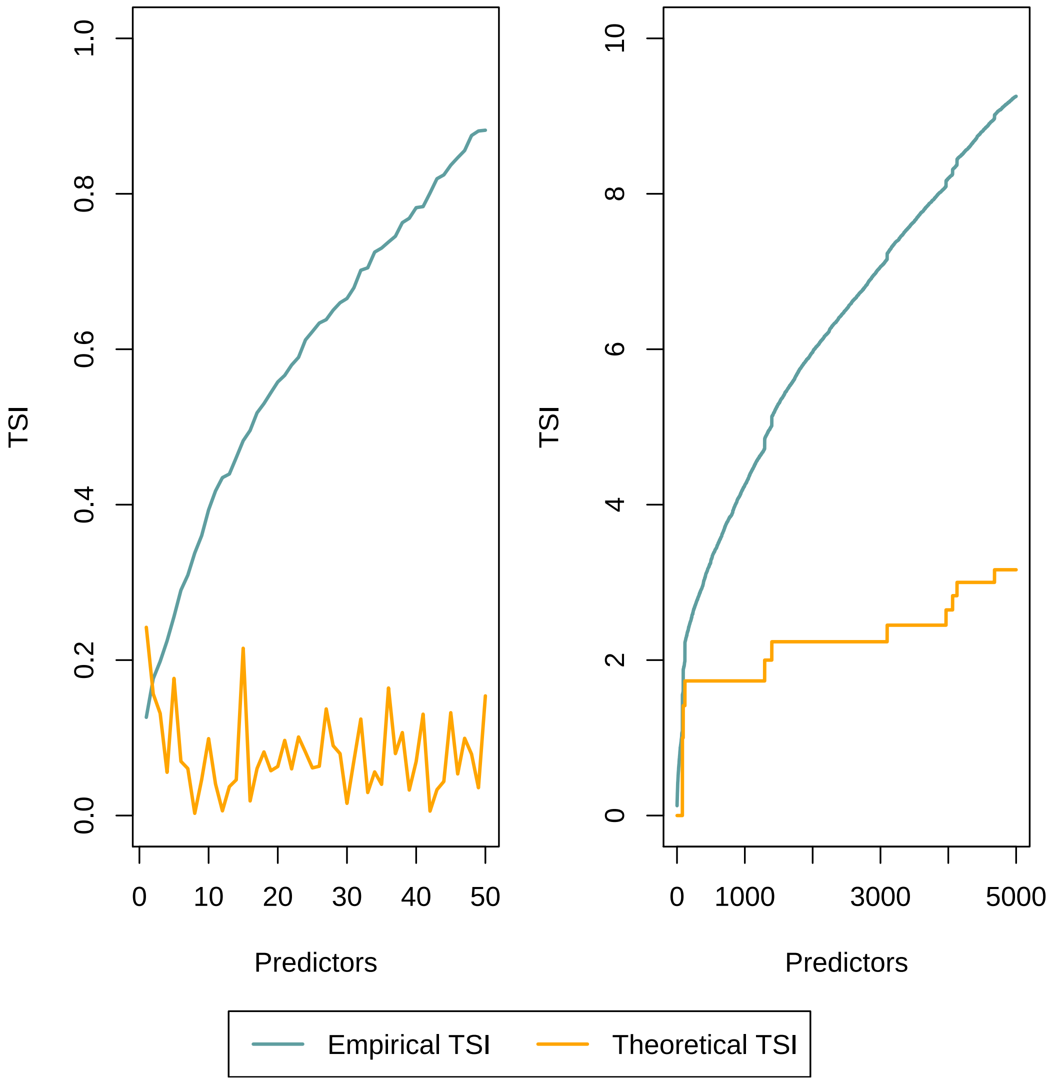

Figure 24. Median empirical total signal index (TSI) for simulations with unsorted signal by number of predictors ($p = 1$ to 50 on left, $p = 1$ to 5000 on right) with theoretical value of TSI overlaid for $n = 200$. Blue line is median empirical TSI ($\bar{\boldsymbol{y}}_1$ has 10 randomly distributed, nonzero elements with other elements equal zero and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); orange line is corresponding TSI.

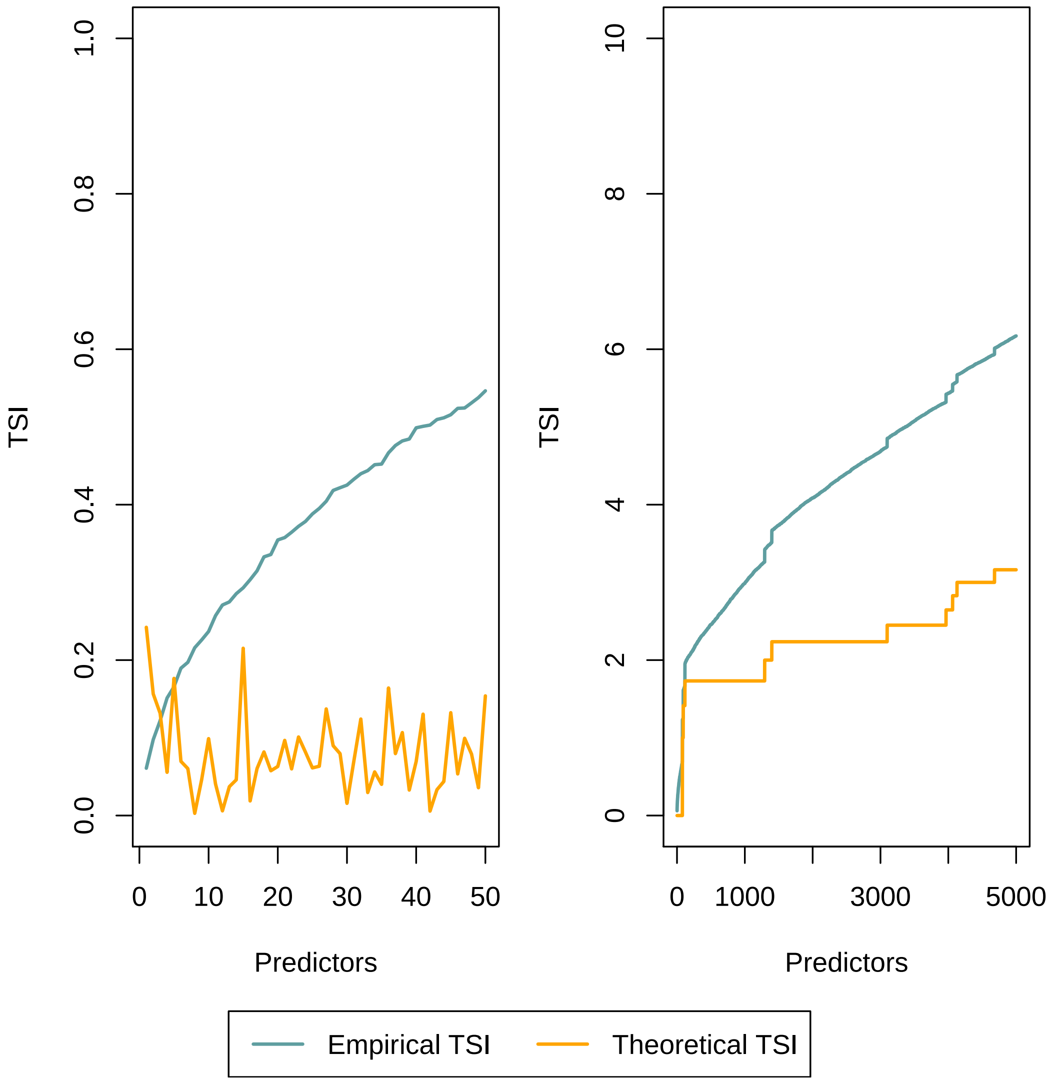

Figure 25. Median empirical total signal index (TSI) for simulations with unsorted signal by number of predictors ($p = 1$ to 50 on left, $p = 1$ to 5000 on right) with theoretical value of TSI overlaid for $n = 500$. Blue line is median empirical TSI ($\bar{\boldsymbol{y}}_1$ has 10 randomly distributed, nonzero elements with other elements equal zero and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); orange line is corresponding TSI.

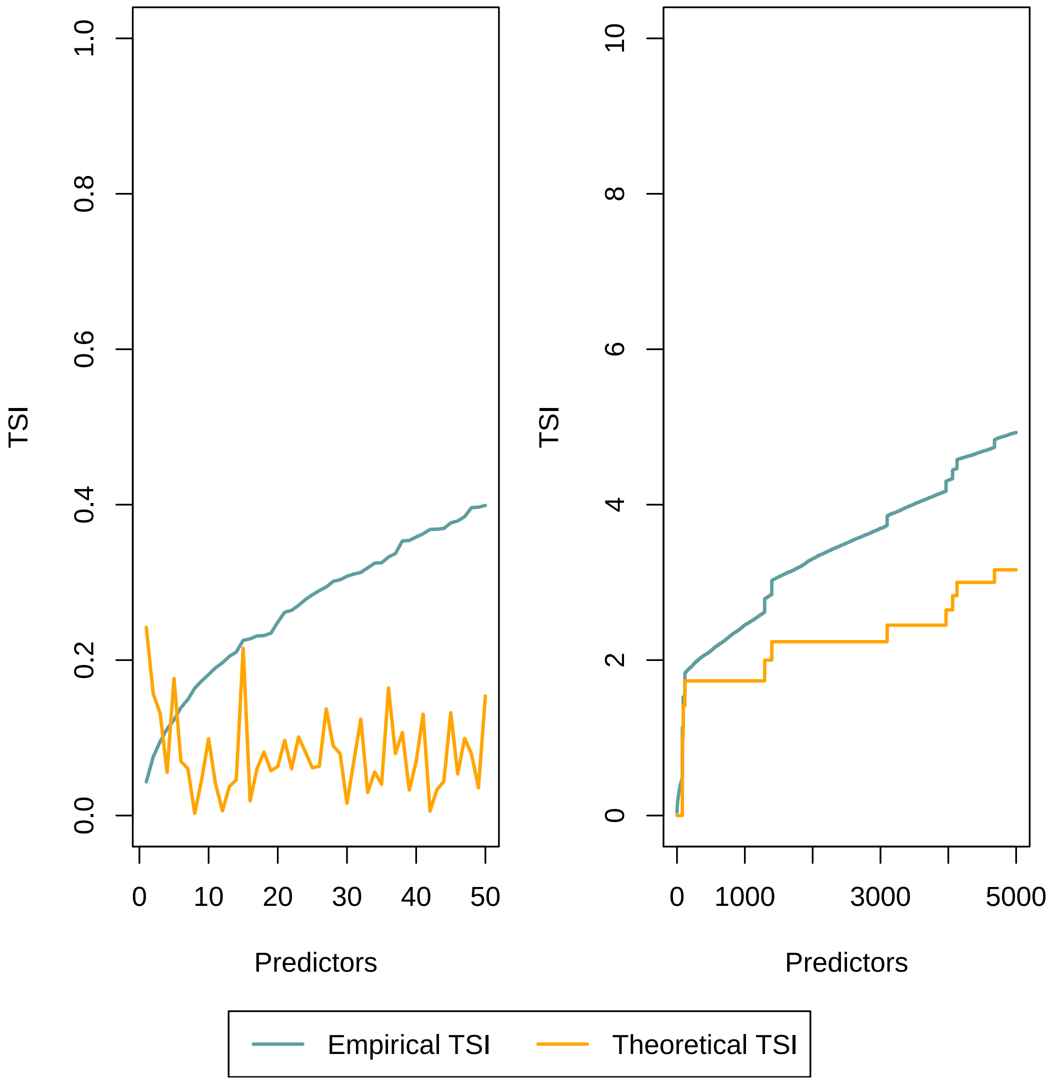

Figure 26. Median empirical total signal index (TSI) for simulations with unsorted signal by number of predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 1000$. Blue line is median empirical TSI ($\bar{\boldsymbol{y}}_1$ has 10 randomly distributed, nonzero elements with other elements equal zero and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); orange line is corresponding TSI.

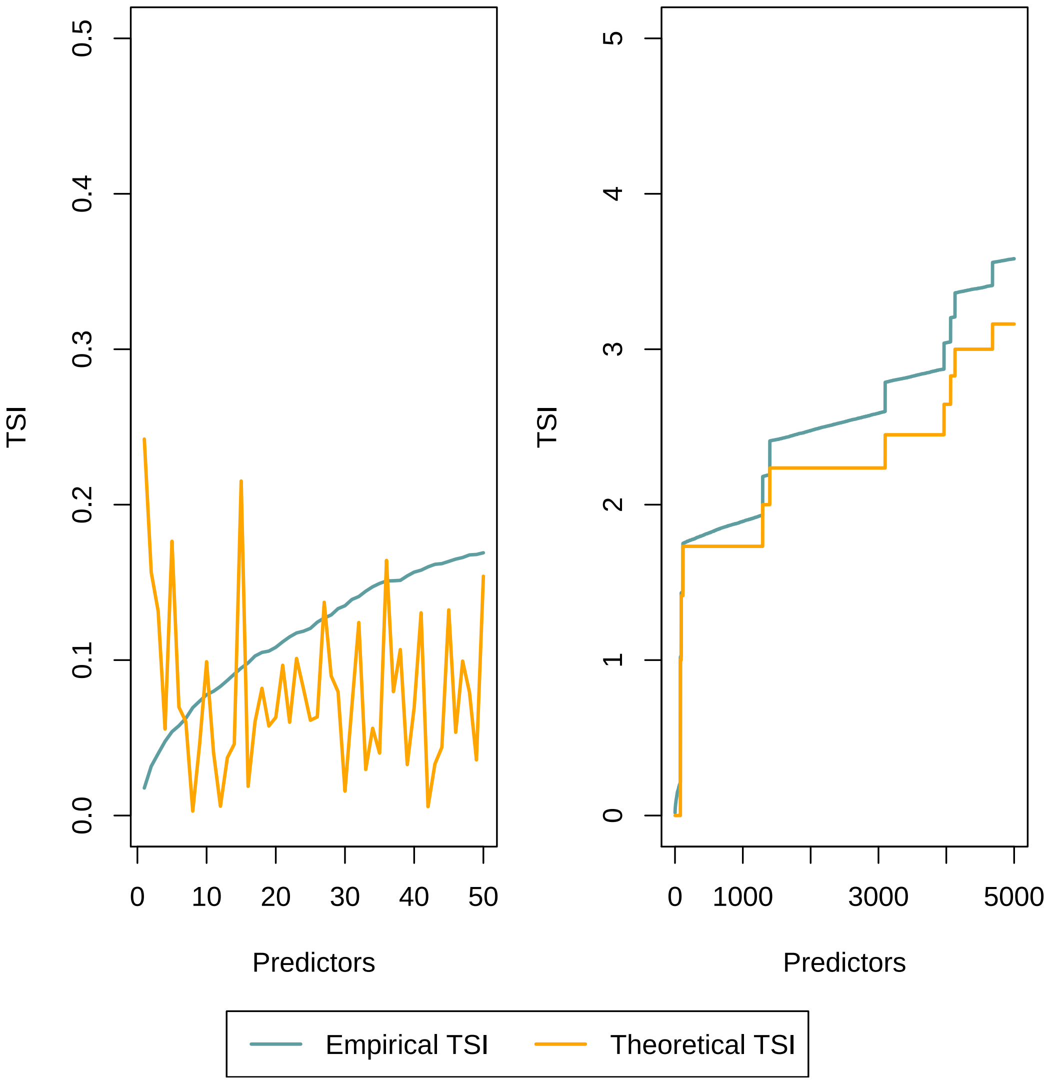

Figure 27. Median empirical total signal index (TSI) for simulations with unsorted signal by number of predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 5000$. Blue line is median empirical TSI ($\bar{\boldsymbol{y}}_1$ has 10 randomly distributed, nonzero elements with other elements equal zero and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$); orange line is corresponding TSI.

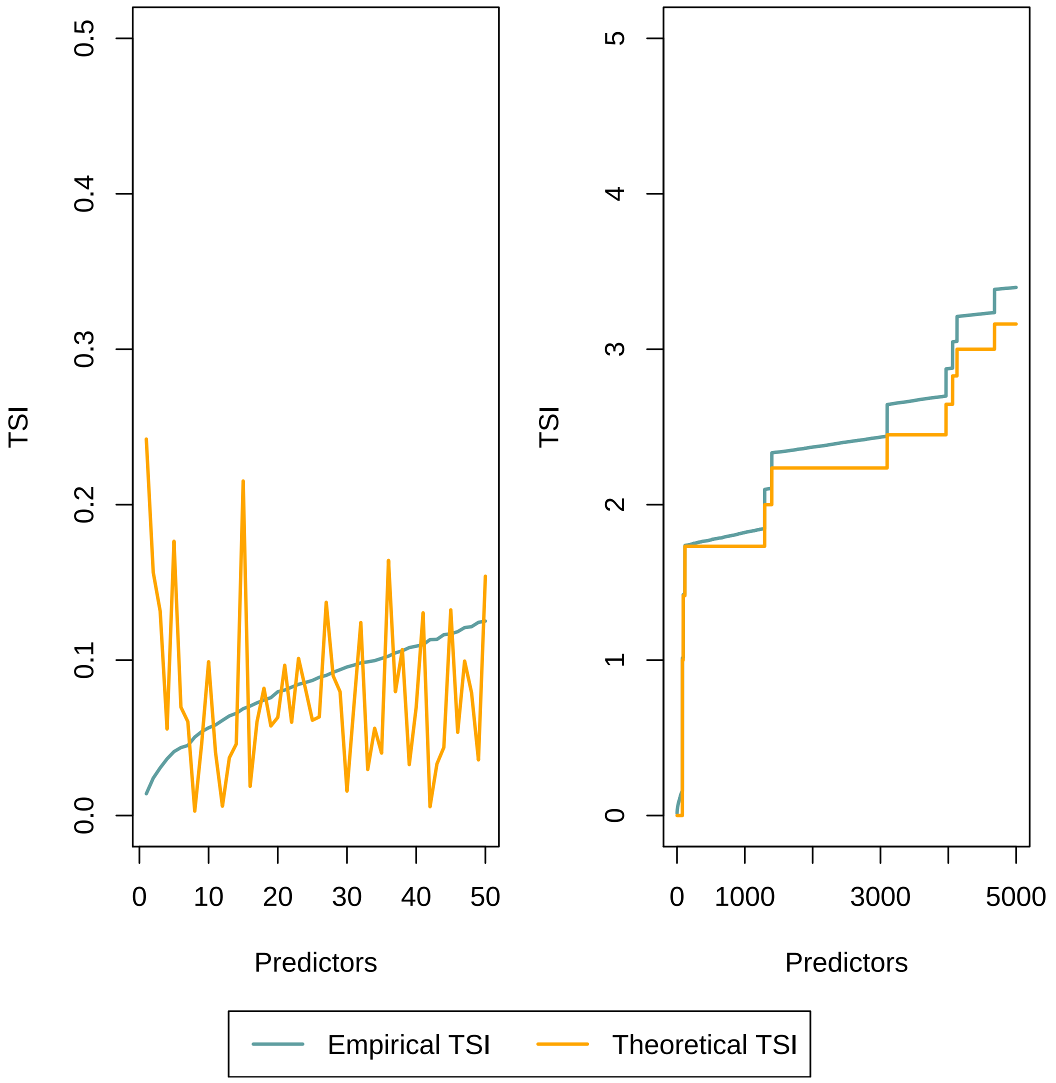

Figure 28. Median empirical total signal index (TSI) for simulations with unsorted signal by number of predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 10000$. Blue line is median empirical TSI ($\bar{\boldsymbol{y}}_1$ has 10 randomly distributed, nonzero elements with other elements equal zero and $\boldsymbol{bary}_2 = \mathbf{0}$); orange line is corresponding TSI.

We also sought to investigate the performance of TSI$_e$ in settings where there was correlation between the predictors. We generated data from a multivariate normal distribution for two classes similar to those in Scenario 1 described in the first row of Table 9. That is, the nonzero elements of $\bar{\boldsymbol{y}}_1$ were one, all elements of $\bar{\boldsymbol{y}}_2 = 0$, and variances for both samples were one. Unlike the previous scenarios, we added covariance terms to the off diagonal entries of the variance-covariance matrix corresponding to the nonzero entries of $\bar{\boldsymbol{y}}_1$ for Λ_1 and Λ_2:

$$\begin{bmatrix}
1 & \rho & \rho & \rho & \rho & \rho & \rho & \rho & \rho & \rho & 0 & \cdots & 0 \\
\rho & 1 & \rho & \rho & \rho & \rho & \rho & \rho & \rho & \rho & 0 & \cdots & 0 \\
\rho & \rho & 1 & \rho & \rho & \rho & \rho & \rho & \rho & \rho & 0 & \cdots & 0 \\
\rho & \rho & \rho & 1 & \rho & \rho & \rho & \rho & \rho & \rho & 0 & \cdots & 0 \\
\rho & \rho & \rho & \rho & 1 & \rho & \rho & \rho & \rho & \rho & 0 & \cdots & 0 \\
\rho & \rho & \rho & \rho & \rho & 1 & \rho & \rho & \rho & \rho & 0 & \cdots & 0 \\
\rho & \rho & \rho & \rho & \rho & \rho & 1 & \rho & \rho & \rho & 0 & \cdots & 0 \\
\rho & \rho & \rho & \rho & \rho & \rho & \rho & 1 & \rho & \rho & 0 & \cdots & 0 \\
\rho & \rho & \rho & \rho & \rho & \rho & \rho & \rho & 1 & \rho & 0 & \cdots & 0 \\
\rho & \rho & \rho & \rho & \rho & \rho & \rho & \rho & \rho & 1 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0
\end{bmatrix}$$

We explored 100 simulations for $\rho = 0.25$, 0.50, and 0.75 where $n = 200$, divided equally between classes. Although we also attempted to include simulations for $\rho =$ -0.25, -0.50, and -0.75, these variance-covariance matrices were not semi-positive definite and could not be computed. We calculated TSI$_e$ as before and graphed the results in Figure 29 along with theoretical and empiric measures assuming independence among the predictors. As seen in this figure, the median TSI$_e$ for the correlated simulations where $\rho = 0.25$ and 0.50 closely match the one without correlation. When $\rho = 0.75$, median TSI$_e$ underestimates TSI until around $p_0 = 100$. At this point, median TSI$_e$ for $\rho = 0.75$ crosses the line for TSI and thereafter exceeds it. Generally, the results for this simulation follow a similar trend to the one with no correlation and are not concerning for TSI$_e$ performance.

Finally, we wanted to gauge the performance of TSI$_e$ vis-à-vis scenarios from the first chapter (Table 1) to see whether the index behaved as expected. In these scenarios, signal strength remains fixed at three for the first three scenarios but its strength decreases from 10 to six to three nonzero elements. Consequently, we expect TSI to decline with dwindling signal between Scenarios 1, 2, and 3. Based on the performance of random forest and boosting classifiers, Scenario 4 appears to have the least amount of signal strength of the scenarios thus we expect this scenario to have the lowest TSI. Figure 30 displays the median TSI for 100 simulations conducted with the same parameter values from Chapter 2 and 100 samples per group. Visual inspection of the graph indicates that TSI$_e$ acts as anticipated: Scenario 1 has the highest TSI$_e$ followed by Scenario 2 then Scenario 3 and, lastly, Scenario 4.

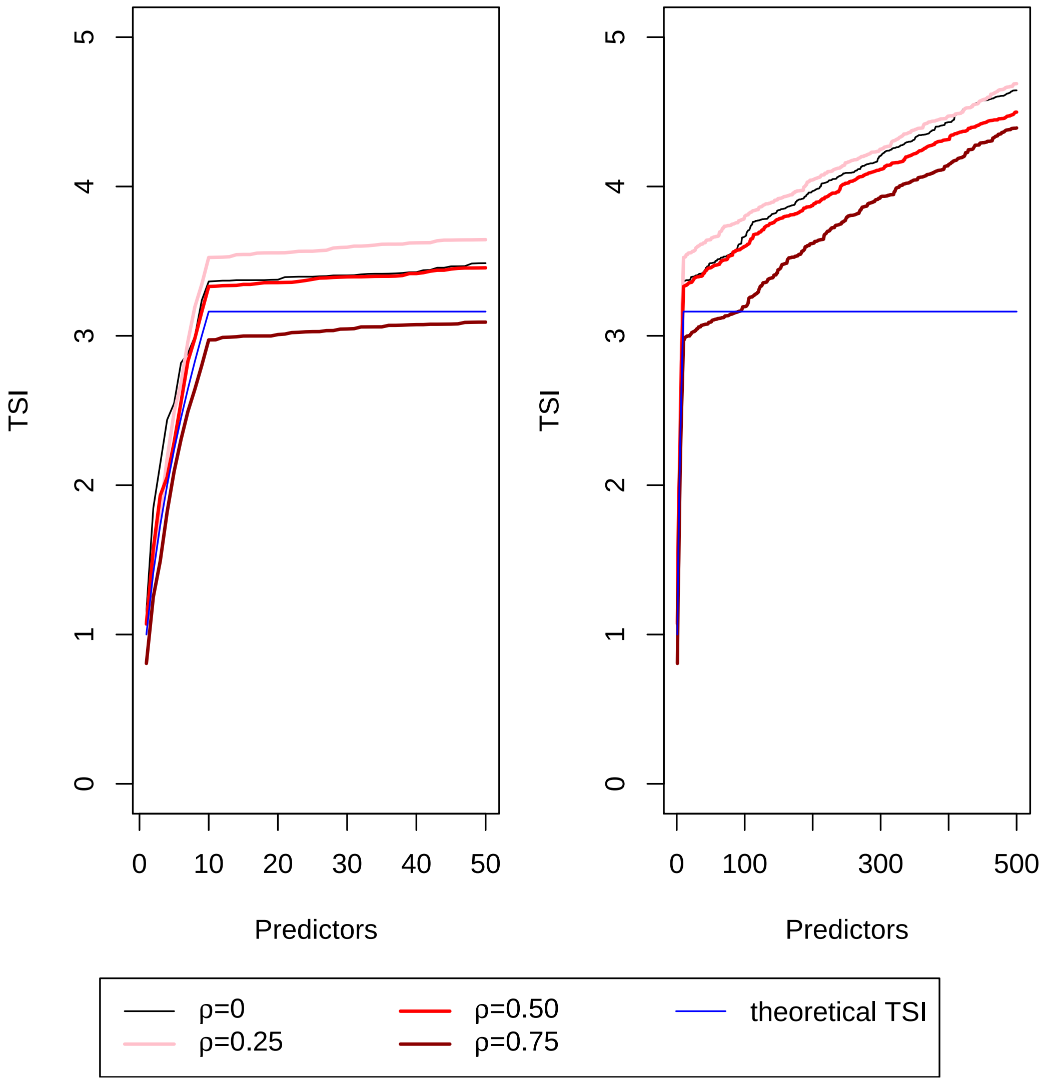

Figure 29. Median empirical total signal index (TSI) for simulations with correlation between predictors (p = 1 to 50 on left, p = 1 to 5000 on right) with theoretical value of TSI overlaid for $n = 200$. All simulations developed for $\bar{\boldsymbol{y}}_1$ equal one for the first 10 nonzero elements then zero afterward and $\bar{\boldsymbol{y}}_2 = \mathbf{0}$; black line is median empirical total signal index for $\rho = 0$, pink for $\rho = 0.25$, red for $\rho = 0.50$, and dark red for $\rho = 0.75$.

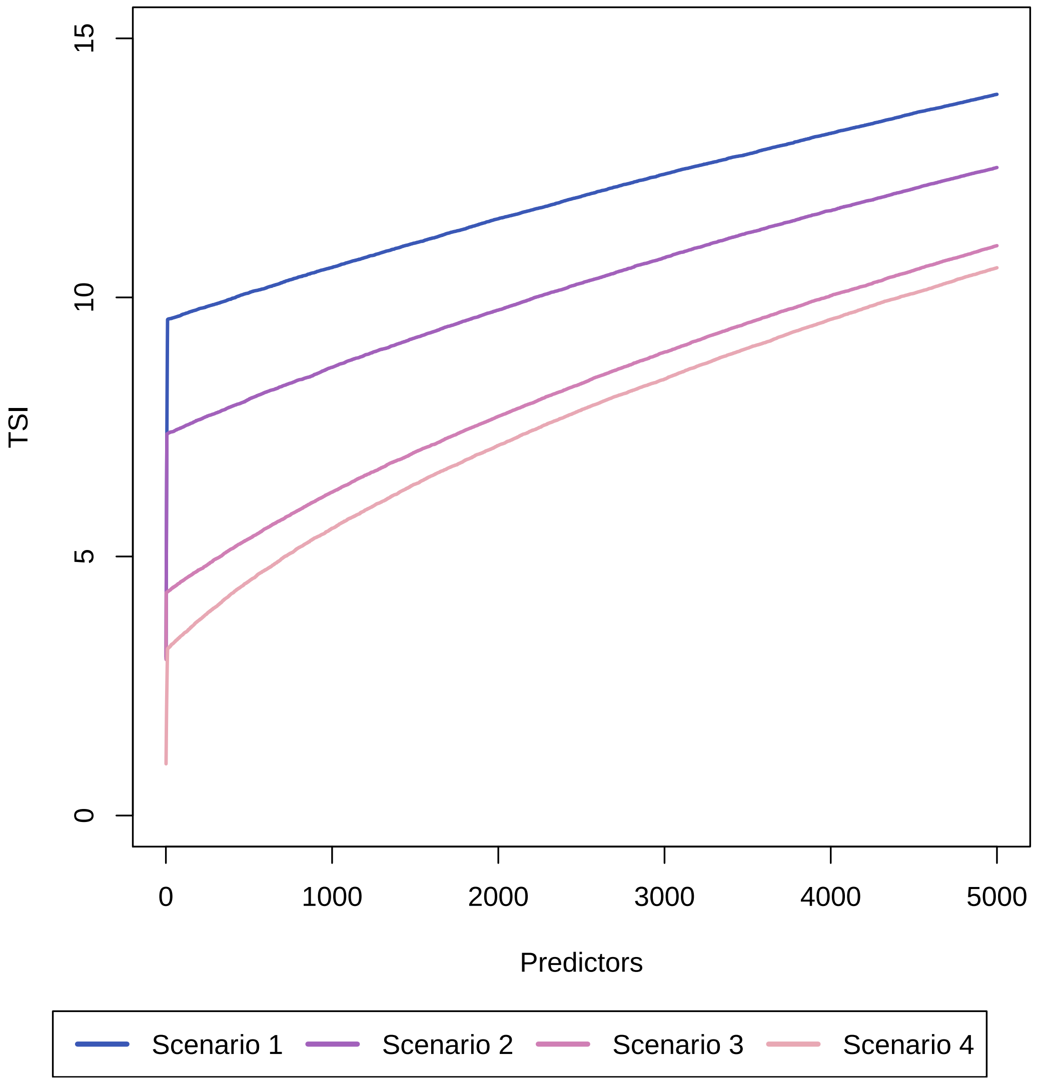

Figure 30. Comparison of total signal index (TSI) from each of four scenarios where $n = 100$ for each class. Thick lines represent the median TSI from 100 simulations with the navy line representing Scenario 1 ($m = 10$ nonzero elements for $\bar{\boldsymbol{y}}_2$ each equal to three and $\bar{\boldsymbol{y}}_1 = \mathbf{0}$), purple Scenario 2 ($m = 6$ nonzero elements for $\bar{\boldsymbol{y}}_2$ each equal to three and $\bar{\boldsymbol{y}}_1 = \mathbf{0}$), rose Scenario 3 ($m = 2$ nonzero elements for $\bar{\boldsymbol{y}}_2$ each equal to three and $\bar{\boldsymbol{y}}_1 = \mathbf{0}$), and pink Scenario 4 ($m = 10$ nonzero elements for $\bar{\boldsymbol{y}}_2$ each equal to one and $\bar{\boldsymbol{y}}_1 = \mathbf{0}$).

5 Summary and conclusion

Over the course of this thesis, we scrutinized noise accumulation when conducting two group classification with high dimensional data using various simulations. In Chapter 2, we explored how noise accumulation impacts the discriminative ability of random forest, support vector machine, and boosted classification in varying conditions of signal strength and magnitude. We showed that the support vector machine algorithm had poor classification ability when signal strength was low or its magnitude limited, at least without additional tailoring of default settings. Boosting performed extremely well in all scenarios but was extremely time intensive. This approach would be the best option in settings of low signal and many predictors as long as time is not constrained or multiple runs of the algorithm could be parallelized. Although boosting outperformed it in the scenarios with less signal, the random forest classifier provided about 80% accurate class predictions in the most challenging settings and took a lot less time. In the third chapter, we sought to understand the behavior of noise accumulation in random forest classification. The results of our simulations indicated that the effects of noise accumulation could be minimized by increasing the dataset's sample size or signal strength. We also discovered that the magnitude of signal needs to be sufficient for these modifications to influence the classifier's discriminative ability. In Chapter 4, we derived theoretical and empirical measures to summarize signal relative to background noise for two classes then tested it with simulations. Although affected by noise accumulation itself, empirical TSI replicated the pattern of classification difficulty ascertained from previous simulations.

For this project, we chose to limit our inquiry to a two-group classification problem with simplistic assumptions about the correlation between predictors. Future research involving multiple groups or more realistic variance-covariance structures would help further characterize noise accumulation. Likewise, our Total Signal Index could be extended to multiple groups or the equation generalized to directly account for correlation among predictors. It would also be informative to see how newer classification methods invented to deal with sparse data such as least absolute shrinkage and selection operator (LASSO) or sparse linear discriminate analyses would handle the scenarios we examined. Also, recent R packages like xgboost may speed up the algorithm for boosted classification trees, making this approach more accessible for further study.

The impetus for this thesis was to explore noise accumulation in classification and elucidate how it challenged statistical analysis of high dimensional data as Fan et al. proposed in ”Challenges of Big Data analysis.” Noise accumulation does appear to constitute a problem for high dimensional classification but it can be addressed with study design as we suggested in Chapter 2 or with variable selection that effectively reduces the dimension of datasets.[5] Finally, noise accumulation may be difficult to detect in practice. We hope Total Signal Index will be a useful tool to detecting noise accumulation in high dimensional data and, in doing so, allow researchers to reflect on whether and how to adapt their analyses.

References

[1] Speed T. Data science, big data and statistics: Can we all live together? http://www.chalmers.se/en/areas-of-advance/ict/calendar/Pages/Terry-Speed.aspx. 2014.

[2] Huber PJ. Huge data sets. In: *Compstat*. Springer. 1994; 3–13.

[3] Council NR. *Massive Data Sets: Proceedings of a Workshop*. Washington, DC: The National Academies Press. 1996.

[4] Gandomi A, Haider M. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*. 2015;35(2):137–144 .

[5] Fan J, Han F, Liu H. Challenges of big data analysis. *National science review*. 2014; 1(2):293–314 .

[6] Fan J, Fan Y. High dimensional classification using features annealed independence rules. *Annals of statistics*. 2008;36(6):2605 .

[7] Hall P, Pittelkow Y, Ghosh M. Theoretical measures of relative performance of classifiers for high dimensional data with small sample sizes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. 2008;70(1):159–173 .

[8] Fan J. Features of big data and sparsest solution in high confidence set. *Past, present, and future of statistical science*. 2013;507–523 .

[9] Fan J, Lv J. A selective overview of variable selection in high dimensional feature space. *Statistica Sinica*. 2010;20(1):101 .

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

[11] Liaw A, Wiener M. Classification and regression by randomforest. *R News*. 2002; 2(3):18–22 .

[12] Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. 2015. R package version 1.6-6.

[13] Ridgeway G. *gbm: Generalized Boosted Regression Models*. 2017. URL https://CRAN.R-project.org/package=gbm. R package version 2.1.3.

[14] Breiman L. Random forests. *Machine learning*. 2001;45(1):5–32 .

[15] James G, Witten D, Hastie T, Tibshirani R. *An introduction to statistical learning*, volume 112. Springer. 2013.

[16] Friedman J, Hastie T, Tibshirani R, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*. 2000;28(2):337–407 .

[17] Friedman JH. Greedy function approximation: a gradient boosting machine. *Annals of statistics*. 2001;1189–1232 .

[18] Cortes C, Vapnik V. Support-vector networks. *Machine learning*. 1995;20(3):273–297 .

A Appendix

A.1 R code

A.1.1 Code for Chapter 2

```
######################################################################

# function to generate data
# use rnorm as data not correlated (significantly faster than mvnorm)
draw3 = function(d,ss1,ss2,nonsparse,mu1,mu2,sd1,sd2){

  n21 = nonsparse

  # class 2
  y1a = matrix(rnorm(n=ss2*n21,
                     mean=mu1,
                     sd=sd1),nrow=ss2,ncol=n21,byrow=FALSE)
  y1b = matrix(rnorm(n=ss2*(d-n21),
                     mean=0,
                     sd=1),nrow=ss2,ncol=d-n21,byrow=FALSE)
  y1 = cbind(y1a,y1b)

  # class 1
  x1a = matrix(rnorm(n=ss1*n21,
                     mean=mu2,
                     sd=sd2),nrow=ss1,ncol=n21,byrow=FALSE)
  x1b = matrix(rnorm(n=ss1*(d-n21),
                     mean=0,
                     sd=1),nrow=ss1,ncol=d-n21,byrow=FALSE)
  x1 = cbind(x1a,x1b)

  sdata = rbind(y1,x1)
  return(sdata=sdata)
}


## generate data for simulations
  # number of simulations
  iters = 100
```

```
# setup parallel backend to use 8 processors
cl <- makeCluster(8)
registerDoParallel(cl)

set.seed(1011)

# all scenarios -- d=5000, ss1=100, ss2=100, mu1=0, sd1=1, sd2=1
# scenario 1 -- mu2 = 3, nonsparse = 10
train_a1 = foreach(icount(iters)) %dopar% {
 draw3(d=5000,ss1=100,ss2=100,nonsparse=10,
    mu1=0,mu2=3,sd=1,sd2=1)
     }
test_a1 = foreach(icount(iters)) %dopar% {
 draw3(d=5000,ss1=100,ss2=100,nonsparse=10,
                   mu1=0,mu2=3,sd=1,sd2=1)
     }

# scenario 2 -- mu2 = 3, nonsparse = 6
train_a2 = foreach(icount(iters)) %dopar% {
 draw3(d=5000,ss1=100,ss2=100,nonsparse=6,
                   mu1=0,mu2=3,sd=1,sd2=1)
     }
test_a2 = foreach(icount(iters)) %dopar% {
 draw3(d=5000,ss1=100,ss2=100,nonsparse=6,
                   mu1=0,mu2=3,sd=1,sd2=1)
     }

# scenario 3 -- mu2 = 3, nonsparse = 2
train_a3 = foreach(icount(iters)) %dopar% {
 draw3(d=5000,ss1=100,ss2=100,nonsparse=2,
                   mu1=0,mu2=3,sd=1,sd2=1)
     }
test_a3 = foreach(icount(iters)) %dopar% {
 draw3(d=5000,ss1=100,ss2=100,nonsparse=2,
                   mu1=0,mu2=3,sd=1,sd2=1)
     }

# scenario 4 -- mu2 = 1, nonsparse = 10
train_a4 = foreach(icount(iters)) %dopar% {
 draw3(d=5000,ss1=100,ss2=100,nonsparse=10,
                   mu1=0,mu2=1,sd=1,sd2=1)
     }
test_a4 = foreach(icount(iters)) %dopar% {
   draw3(d=5000,ss1=100,ss2=100,nonsparse=10,
                   mu1=0,mu2=1,sd=1,sd2=1)
       }
  stopCluster(cl)
  registerDoSEQ()

######################################################################

## principal component analysis simulations

# PCA for Scenario 1
pc_a1_p2=prcomp(train_a1[[1]][,1:2],scale=T,retx=T)
pc_a1_p10=prcomp(train_a1[[1]][,1:10],scale=T,retx=T)
pc_a1_p100=prcomp(train_a1[[1]][,1:100],scale=T,retx=T)
pc_a1_p200=prcomp(train_a1[[1]][,1:200],scale=T,retx=T)
pc_a1_p1000=prcomp(train_a1[[1]][,1:1000],scale=T,retx=T)
pc_a1_p5000=prcomp(train_a1[[1]][,1:5000],scale=T,retx=T)

# PCA for Scenario 2
pc_a2_p2=prcomp(train_a2[[1]][,1:2],scale=T,retx=T)
pc_a2_p10=prcomp(train_a2[[1]][,1:10],scale=T,retx=T)
pc_a2_p100=prcomp(train_a2[[1]][,1:100],scale=T,retx=T)
pc_a2_p200=prcomp(train_a2[[1]][,1:200],scale=T,retx=T)
pc_a2_p1000=prcomp(train_a2[[1]][,1:1000],scale=T,retx=T)
pc_a2_p5000=prcomp(train_a2[[1]][,1:5000],scale=T,retx=T)

# PCA for Scenario 3
pc_a3_p2=prcomp(train_a3[[1]][,1:2],scale=T,retx=T)
pc_a3_p10=prcomp(train_a3[[1]][,1:10],scale=T,retx=T)
pc_a3_p100=prcomp(train_a3[[1]][,1:100],scale=T,retx=T)
pc_a3_p200=prcomp(train_a3[[1]][,1:200],scale=T,retx=T)
pc_a3_p1000=prcomp(train_a3[[1]][,1:1000],scale=T,retx=T)
pc_a3_p5000=prcomp(train_a3[[1]][,1:5000],scale=T,retx=T)

# PCA for Scenario 4
pc_a4_p2=prcomp(train_a4[[1]][,1:2],scale=T,retx=T)
pc_a4_p10=prcomp(train_a4[[1]][,1:10],scale=T,retx=T)
pc_a4_p100=prcomp(train_a4[[1]][,1:100],scale=T,retx=T)
pc_a4_p200=prcomp(train_a4[[1]][,1:200],scale=T,retx=T)
pc_a4_p1000=prcomp(train_a4[[1]][,1:1000],scale=T,retx=T)
pc_a4_p5000=prcomp(train_a4[[1]][,1:5000],scale=T,retx=T)

######################################################################
```

```
### random forest simulations
library(randomForest)

# create classifier for random forest
run.rft = function(train,test,n,d){
 s1 = list()
 s2 = list()
 for (j in 1:n){
        s1[[j]] = matrix(,nrow=1,ncol=length(a))
        s2[[j]] = matrix(,nrow=1,ncol=length(a))
        for (i in 1:length(a)){
                rf = randomForest(y=as.factor(g),
                                  x=train[[j]][,1:a[i]],
                                  ntrees=1000,
                                  importance=T,
                                  proximity=T)
                s1[[j]][,i] = 1 - (sum(rf$confusion[1,1],
                                       rf$confusion[2,2])/
                                       (length(train[[j]][,1])))
                s2[[j]][,i] = 1 - (sum(diag(table(predict(
                                       rf,test[[j]][,1:a[i]]),g)))/
                                       (length(test[[j]][,1])))
        }
 }
 return(list(s1=s1,s2=s2))
}

# make groups
g = c(rep(1,100),rep(2,100))


# every predictor between 2 and 100
a = seq(2,100,1)

rf_a1_detail = run.rft(train=train_a1,test=test_a1,n=100,d=max(a))
rf_a2_detail = run.rft(train=train_a2,test=test_a2,n=100,d=max(a))
rf_a3_detail = run.rft(train=train_a3,test=test_a3,n=100,d=max(a))
rf_a4_detail = run.rft(train=train_a4,test=test_a4,n=100,d=max(a))
```

```
# every 10th predictor between 2 and 5000
a = seq(0,5000,10)
a[1] = 2

rf_a1 = run.rft(train=train_a1,test=test_a1,n=100,d=max(a))
rf_a2 = run.rft(train=train_a2,test=test_a2,n=100,d=max(a))
rf_a3 = run.rft(train=train_a3,test=test_a3,n=100,d=max(a))
rf_a4 = run.rft(train=train_a4,test=test_a4,n=100,d=max(a))

######################################################################

### support vector machine simulations
library(e1071)

# create classifier for SVM
 run.svm = function(train,test,n){
   t1 = list()
   t2 = list()
   for (j in 1:n){
     t1[[j]] = matrix(,nrow=1,ncol=length(a))
     t2[[j]] = matrix(,nrow=1,ncol=length(a))
     for (i in 1:length(a)){
       sv = svm(y=as.factor(g),
                x=train[[j]][,1:a[i]],
                kernel="linear",
                cost=0.01)
       t1[[j]][,i] = 1 - ((sum(diag(table(g, predict(sv)))))/
                         (length(train[[j]][,1])))
       t2[[j]][,i] = 1-(sum(diag(table(predict(
                            sv,test[[j]][,1:a[i]]),g)))/
                            (length(test[[j]][,1])))
     }
   }
   return(list(t1=t1,t2=t2))
 }


 # make groups
 g = c(rep(1,100),rep(2,100))


# every predictor between 2 and 100
a = seq(2,100,1)
```

```
svm_a1_detail = run.svm(train=train_a1,test=test_a1,n=100)
svm_a2_detail = run.svm(train=train_a2,test=test_a2,n=100)
svm_a3_detail = run.svm(train=train_a3,test=test_a3,n=100)
svm_a4_detail = run.svm(train=train_a4,test=test_a4,n=100)


# every 10th predictor between 2 and 5000
a = seq(0,5000,10)
a[1] = 2

svm_a1 = run.svm(train=train_a1,test=test_a1,n=100)
svm_a2 = run.svm(train=train_a2,test=test_a2,n=100)
svm_a3 = run.svm(train=train_a3,test=test_a3,n=100)
svm_a4 = run.svm(train=train_a4,test=test_a4,n=100)

######################################################################

### boosted classification tree simulations
library(gbm)

# create classifier for boosting
run.gbm = function(train,test,n,d){
  u1 = list()
  u2 = list()
  for (j in 1:n){
    u1[[j]] = matrix(,nrow=1,ncol=length(a))
    u2[[j]] = matrix(,nrow=1,ncol=length(a))
    for (i in 1:length(a)){
      bt = gbm.fit(y=g,
                   x=train[[j]][,1:a[i]],
                   n.trees=10000,
                   verbose=FALSE)
      u1[[j]][,i] = 1-(sum(diag(table(ifelse(bt$fit<0,0,1),g)))/
                      (length(train[[j]][,1])))
      u2[[j]][,i] = 1-(sum(diag(table(ifelse((predict(
                       bt,test[[j]][,1:a[i]],n.trees=10000))
                       <0,0,1),g)))/(length(test[[j]][,1])))
    }
  }
  return(list(u1=u1,u2=u2
  ))
}
```

```
# make groups
g = c(rep(0,100),rep(1,100))

# every predictor between 2 and 100
a = seq(2,100,1)

gbm_a1_detail = run.gbm(train=train_a1,test=test_a1,n=100,d=max(a))
gbm_a2_detail = run.gbm(train=train_a2,test=test_a2,n=100,d=max(a))
gbm_a3_detail = run.gbm(train=train_a3,test=test_a3,n=100,d=max(a))
gbm_a4_detail = run.gbm(train=train_a4,test=test_a4,n=100,d=max(a))


# every 10th predictor between 2 and 5000
a = seq(0,5000,10)
a[1] = 2

gbm_a1 = run.gbm(train=train_a1,test=test_a1,n=100,d=max(a))
gbm_a2 = run.gbm(train=train_a2,test=test_a2,n=100,d=max(a))
gbm_a3 = run.gbm(train=train_a3,test=test_a3,n=100,d=max(a))
gbm_a4 = run.gbm(train=train_a4,test=test_a4,n=100,d=max(a))

######################################################################

### Results

## RF
# test error for every predictor between 2 and 100
t2 = do.call(rbind,rf_a1_detail$s2)
u2 = do.call(rbind,rf_a2_detail$s2)
v2 = do.call(rbind,rf_a3_detail$s2)
w2 = do.call(rbind,rf_a4_detail$s2)

t2.p = apply(t2,2,quantile,probs=c(.1,.5,.9))
u2.p = apply(u2,2,quantile,probs=c(.1,.5,.9))
v2.p = apply(v2,2,quantile,probs=c(.1,.5,.9))
w2.p = apply(w2,2,quantile,probs=c(.1,.5,.9))

# scenario 1
# 10, 50, and 90 percentile test error between p=1 and 10
t2.p[1,1:10]
t2.p[2,1:10]
t2.p[3,1:10]
```

```
# scenario 2
# 10, 50, and 90 percentile test error between p=1 and 10
u2.p[1,1:10]
u2.p[2,1:10]
u2.p[3,1:10]

# scenario 3
# 10, 50, and 90 percentile test error between p=1 and 10
v2.p[1,1:10]
v2.p[2,1:10]
v2.p[3,1:10]

# scenario 4
# 10, 50, and 90 percentile test error between p=1 and 10
w2.p[1,1:10]
w2.p[2,1:10]
w2.p[3,1:10]


# test error for every 10th predictor between 2 and 5000
t2 = do.call(rbind,rf_a1$s2)
u2 = do.call(rbind,rf_a2$s2)
v2 = do.call(rbind,rf_a3$s2)
w2 = do.call(rbind,rf_a4$s2)

t2.p_rf = apply(t2,2,quantile,probs=c(.1,.5,.9))
u2.p_rf = apply(u2,2,quantile,probs=c(.1,.5,.9))
v2.p_rf = apply(v2,2,quantile,probs=c(.1,.5,.9))
w2.p_rf = apply(w2,2,quantile,probs=c(.1,.5,.9))

# scenario 1
# 10, 50, and 90 percentile test error at p=5000
t2.p_rf[1,501]
t2.p_rf[2,501]
t2.p_rf[3,501]

# scenario 2
# 10, 50, and 90 percentile test error at p=5000
u2.p_rf[1,501]
u2.p_rf[2,501]
u2.p_rf[3,501]
```

```
# scenario 3
# 10, 50, and 90 percentile test error at p=5000
v2.p_rf[1,501]
v2.p_rf[2,501]
v2.p_rf[3,501]

# scenario 4
# 10, 50, and 90 percentile test error at p=5000
w2.p_rf[1,501]
w2.p_rf[2,501]
w2.p_rf[3,501]



## SVM
# test error for every predictor between 2 and 100
t2 = do.call(rbind,svm_a1_detail$t2)
u2 = do.call(rbind,svm_a2_detail$t2)
v2 = do.call(rbind,svm_a3_detail$t2)
w2 = do.call(rbind,svm_a4_detail$t2)

t2.p = apply(t2,2,quantile,probs=c(.1,.5,.9))
u2.p = apply(u2,2,quantile,probs=c(.1,.5,.9))
v2.p = apply(v2,2,quantile,probs=c(.1,.5,.9))
w2.p = apply(w2,2,quantile,probs=c(.1,.5,.9))

# scenario 1
# 10, 50, and 90 percentile test error between p=1 and 10
t2.p[1,1:10]
t2.p[2,1:10]
t2.p[3,1:10]

# scenario 2
# 10, 50, and 90 percentile test error between p=1 and 10
u2.p[1,1:10]
u2.p[2,1:10]
u2.p[3,1:10]

# scenario 3
# 10, 50, and 90 percentile test error between p=1 and 10
v2.p[1,1:10]
v2.p[2,1:10]
v2.p[3,1:10]
```

```
# scenario 4
# 10, 50, and 90 percentile test error between p=1 and 10
w2.p[1,1:10]
w2.p[2,1:10]
w2.p[3,1:10]


# test error for every 10th predictor between 2 and 5000
t2 = do.call(rbind,svm_a1$t2)
u2 = do.call(rbind,svm_a2$t2)
v2 = do.call(rbind,svm_a3$t2)
w2 = do.call(rbind,svm_a4$t2)

t2.p_svm = apply(t2,2,quantile,probs=c(.1,.5,.9))
u2.p_svm = apply(u2,2,quantile,probs=c(.1,.5,.9))
v2.p_svm = apply(v2,2,quantile,probs=c(.1,.5,.9))
w2.p_svm = apply(w2,2,quantile,probs=c(.1,.5,.9))

# scenario 1
# 10, 50, and 90 percentile test error at p=5000
t2.p_svm[1,501]
t2.p_svm[2,501]
t2.p_svm[3,501]

# scenario 2
# 10, 50, and 90 percentile test error at p=5000
u2.p_svm[1,501]
u2.p_svm[2,501]
u2.p_svm[3,501]

# scenario 3
# 10, 50, and 90 percentile test error at p=5000
v2.p_svm[1,501]
v2.p_svm[2,501]
v2.p_svm[3,501]

# scenario 4
# 10, 50, and 90 percentile test error at p=5000
w2.p_svm[1,501]
w2.p_svm[2,501]
w2.p_svm[3,501]
```

```
## GBM
# test error for every predictor between 2 and 100
t2 = do.call(rbind,gbm_a1_detail$u2)
u2 = do.call(rbind,gbm_a2_detail$u2)
v2 = do.call(rbind,gbm_a3_detail$u2)
w2 = do.call(rbind,gbm_a4_detail$u2)

t2.p = apply(t2,2,quantile,probs=c(.1,.5,.9))
u2.p = apply(u2,2,quantile,probs=c(.1,.5,.9))
v2.p = apply(v2,2,quantile,probs=c(.1,.5,.9))
w2.p = apply(w2,2,quantile,probs=c(.1,.5,.9))

# scenario 1
# 10, 50, and 90 percentile test error between p=1 and 10
t2.p[1,1:10]
t2.p[2,1:10]
t2.p[3,1:10]

# scenario 2
# 10, 50, and 90 percentile test error between p=1 and 10
u2.p[1,1:10]
u2.p[2,1:10]
u2.p[3,1:10]

# scenario 3
# 10, 50, and 90 percentile test error between p=1 and 10
v2.p[1,1:10]
v2.p[2,1:10]
v2.p[3,1:10]

# scenario 4
# 10, 50, and 90 percentile test error between p=1 and 10
w2.p[1,1:10]
w2.p[2,1:10]
w2.p[3,1:10]


# test error for every 10th predictor between 2 and 5000
t2 = do.call(rbind,gbm_a1$u2)
u2 = do.call(rbind,gbm_a2$u2)
v2 = do.call(rbind,gbm_a3$u2)
```

```
w2 = do.call(rbind,gbm_a4$u2)

t2.p_gbm = apply(t2,2,quantile,probs=c(.1,.5,.9))
u2.p_gbm = apply(u2,2,quantile,probs=c(.1,.5,.9))
v2.p_gbm = apply(v2,2,quantile,probs=c(.1,.5,.9))
w2.p_gbm = apply(w2,2,quantile,probs=c(.1,.5,.9))

# scenario 1
# 10, 50, and 90 percentile test error at p=5000
t2.p_gbm[1,501]
t2.p_gbm[2,501]
t2.p_gbm[3,501]

# scenario 2
# 10, 50, and 90 percentile test error at p=5000
u2.p_gbm[1,501]
u2.p_gbm[2,501]
u2.p_gbm[3,501]

# scenario 3
# 10, 50, and 90 percentile test error at p=5000
v2.p_gbm[1,501]
v2.p_gbm[2,501]
v2.p_gbm[3,501]

# scenario 4
# 10, 50, and 90 percentile test error at p=5000
w2.p_gbm[1,501]
w2.p_gbm[2,501]
w2.p_gbm[3,501]

######################################################################

### Graphs

## graph results from PCA

# make groups
g = c(rep(1,100),rep(2,100))

# Scenario1
pdf("graphs/pca_a1.pdf")
par(mfrow=c(2,3))
```

```
plot(pc_a1_p2$x[,1],pc_a1_p2$x[,2],
     xlab='x',ylab='y',
     col=g,pch=16,xlim=c(-3,3),ylim=c(-3,3))
mtext("(A) p=2", at=-25,l=1)
plot(pc_a1_p10$x[,1],pc_a1_p10$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(B) p=10", at=-25,l=1)
plot(pc_a1_p100$x[,1],pc_a1_p100$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(C) p=100", at=-25,l=1)
plot(pc_a1_p200$x[,1],pc_a1_p200$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(D) p=200",at=-25,l=1)
plot(pc_a1_p1000$x[,1],pc_a1_p1000$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(E) p=1000",at=-25,l=1)
plot(pc_a1_p5000$x[,1],pc_a1_p5000$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(F) p=5000",at=-25,l=1)
dev.off()

# Scenario 2
pdf("graphs/pca_a2.pdf")
par(mfrow=c(2,3))
plot(pc_a2_p2$x[,1],pc_a2_p2$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(A) p=2", at=-25,l=1)
plot(pc_a2_p10$x[,1],pc_a2_p10$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(B) p=10", at=-25,l=1)
plot(pc_a2_p100$x[,1],pc_a2_p100$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
plot(pc_a2_p200$x[,1],pc_a2_p200$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
```

```
mtext("(D) p=200",at=-25,l=1)
plot(pc_a2_p1000$x[,1],pc_a2_p1000$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(E) p=1000",at=-25,l=1)
plot(pc_a2_p5000$x[,1],pc_a2_p5000$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(F) p=5000",at=-25,l=1)
dev.off()

# Scenario 3
pdf("graphs/pca_a3.pdf")
par(mfrow=c(2,3))
plot(pc_a3_p2$x[,1],pc_a3_p2$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(A) p=2", at=-25,l=1)
plot(pc_a3_p10$x[,1],pc_a3_p10$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(B) p=10", at=-25,l=1)
plot(pc_a3_p100$x[,1],pc_a3_p100$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(C) p=100",at=-25,l=1)
plot(pc_a3_p200$x[,1],pc_a3_p200$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(D) p=200",at=-25,l=1)
plot(pc_a3_p1000$x[,1],pc_a3_p1000$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(E) p=1000",at=-25,l=1)
plot(pc_a3_p5000$x[,1],pc_a3_p5000$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(F) p=5000",at=-25,l=1)
dev.off()

# Scenario 4
pdf("graphs/pca_a4.pdf")
par(mfrow=c(2,3))
```

```
plot(pc_a4_p2$x[,1],pc_a4_p2$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(A) p=2", at=-25,l=1)
plot(pc_a4_p10$x[,1],pc_a4_p10$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(B) p=10", at=-25,l=1)
plot(pc_a4_p100$x[,1],pc_a4_p100$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(C) p=100",at=-25,l=1)
plot(pc_a4_p200$x[,1],pc_a4_p200$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(D) p=200",at=-25,l=1)
plot(pc_a4_p1000$x[,1],pc_a4_p1000$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(E) p=1000",at=-25,l=1)
plot(pc_a4_p5000$x[,1],pc_a4_p5000$x[,2],
     xlab='1st Principal Component',ylab='2nd Principal Component',
     col=g,pch=16,xlim=c(-18,18),ylim=c(-18,18))
mtext("(F) p=5000",at=-25,l=1)
dev.off()



## graph results from RF
# Scenario 1
pdf("graphs/RF_a4.pdf")
plot(a,t2.p_rf[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="chocolate3",
     cex.lab=1.3)
points(a,t2.p_rf[1,],type='l',lwd=1,col="darkgoldenrod1")
points(a,t2.p_rf[3,],type='l',lwd=1,col="darkgoldenrod1")
dev.off()

# Scenario 2
pdf("graphs/RF_a2.pdf")
plot(a,u2.p_rf[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="chocolate3",
     cex.lab=1.3)
```

```
points(a,u2.p_rf[1,],type='l',lwd=1,col="darkgoldenrod1")
points(a,u2.p_rf[3,],type='l',lwd=1,col="darkgoldenrod1")
dev.off()

# Scenario 3
pdf("graphs/RF_a3.pdf")
plot(a,v2.p_rf[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="chocolate3",
     cex.lab=1.3)
points(a,v2.p_rf[1,],type='l',lwd=1,col="darkgoldenrod1")
points(a,v2.p_rf[3,],type='l',lwd=1,col="darkgoldenrod1")
dev.off()

# Scenario 4
pdf("graphs/RF_a4.pdf")
plot(a,w2.p_rf[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="chocolate3",
     cex.lab=1.3)
points(a,w2.p_rf[1,],type='l',lwd=1,col="darkgoldenrod1")
points(a,w2.p_rf[3,],type='l',lwd=1,col="darkgoldenrod1")
dev.off()



## graph results from SVM
# Scenario 1
pdf("graphs/SVM_a1.pdf")
plot(a,t2.p_svm[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="yellow4",
     cex.lab=1.3)
points(a,t2.p_svm[1,],type='l',lwd=1,col="yellow3")
points(a,t2.p_svm[3,],type='l',lwd=1,col="yellow3")
dev.off()

# Scenario 2
pdf("graphs/SVM_a2.pdf")
plot(a,u2.p_svm[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="yellow4",
     cex.lab=1.3)
points(a,u2.p_svm[1,],type='l',lwd=1,col="yellow3")
points(a,u2.p_svm[3,],type='l',lwd=1,col="yellow3")
dev.off()
```

```
# Scenario 3
pdf("graphs/SVM_a3.pdf")
plot(a,v2.p_svm[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="yellow4",
     cex.lab=1.3)
points(a,v2.p_svm[1,],type='l',lwd=1,col="yellow3")
points(a,v2.p_svm[3,],type='l',lwd=1,col="yellow3")
dev.off()

# Scenario 4
pdf("graphs/SVM_a4.pdf")
plot(a,w2.p_svm[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="yellow4",
     cex.lab=1.3)
points(a,w2.p_svm[1,],type='l',lwd=1,col="yellow3")
points(a,w2.p_svm[3,],type='l',lwd=1,col="yellow3")
dev.off()



## graph results from boosted trees
# Scenario 1
pdf("graphs/GBM_a1.pdf")
plot(a,t2.p_gbm[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="limegreen",
     cex.lab=1.3)
points(a,t2.p_gbm[1,],type='l',lwd=1,col="lightgreen")
points(a,t2.p_gbm[3,],type='l',lwd=1,col="lightgreen")
dev.off()

# Scenario 2
pdf("graphs/GBM_a2.pdf")
plot(a,u2.p_gbm[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="limegreen",
     cex.lab=1.3)
points(a,u2.p_gbm[1,],type='l',lwd=1,col="lightgreen")
points(a,u2.p_gbm[3,],type='l',lwd=1,col="lightgreen")
dev.off()

# Scenario 3
pdf("graphs/GBM_a3.pdf")
plot(a,v2.p_gbm[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="limegreen",
     cex.lab=1.3)
points(a,v2.p_gbm[1,],type='l',lwd=1,col="lightgreen")
points(a,v2.p_gbm[3,],type='l',lwd=1,col="lightgreen")
dev.off()

# Scenario 4
pdf("graphs/GBM_a4.pdf")
plot(a,w2.p_gbm[2,],type='l', lwd=3, ylim=c(0,.5),
     ylab='Test error',xlab='Predictors',col="limegreen"
     ,cex.lab=1.3)
points(a,w2.p_gbm[1,],type='l',lwd=1,col="lightgreen")
points(a,w2.p_gbm[3,],type='l',lwd=1,col="lightgreen")
dev.off()



## compare SVM scenarios 3 & 4
pdf("graphs/SVM_a3a4.pdf")
plot(a,v2.p[2,],type='l', lwd=3, ylim=c(0,.5),xlim=c(0,5000),
     ylab='Test error',xlab='Predictors',col="navy",cex.lab=1.3)
points(a,w2.p[2,],type='l',lwd=3,col="limegreen")
dev.off()



## difference between training and test error
## for RF, SVM, and BCT for scenario 4
reset <- function() {
    par(mfrow=c(1, 1), oma=rep(0, 4), mar=rep(0, 4), new=TRUE)
    plot(0:1, 0:1, type="n", xlab="", ylab="", axes=FALSE)
    }

t.train4 = do.call(rbind,rf_a4$s1)
t.test4  = do.call(rbind,rf_a4$s2)
rf.train4 = apply(t.train4,2,median)
rf.test4 = apply(t.test4,2,median)
rf.diff4 = rf.train4-rf.test4

u.train4 = do.call(rbind,svm_a4$t1)
u.test4  = do.call(rbind,svm_a4$t2)
svm.train4 = apply(u.train4,2,median)
svm.test4 = apply(u.test4,2,median)
svm.diff4 = svm.train4-svm.test4
```

```
v.train4 = do.call(rbind,gbm_a4$u1)
v.test4  = do.call(rbind,gbm_a4$u2)
gbm.train4 = apply(v.train4,2,median)
gbm.test4 = apply(v.test4,2,median)
gbm.diff4 = gbm.train4-gbm.test4

pdf("graphs/Diff_S4.pdf")
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1))
plot(a,rf.diff4,type="l",lwd=3, ylim=c(-.4,.4),
     ylab='Training - test error',xlab='Predictors',
     col="chocolate3",cex.lab=1.3,)
lines(a,svm.diff4,type="l",lwd=3,col="yellow4")
lines(a,gbm.diff4,type="l",lwd=3,col="limegreen")
reset()
legend(x = "bottom",inset = 0,
        legend = c("RF","SVM","BCT"),
        col=c("chocolate3","yellow4","limegreen"),
        lwd=3, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()

######################################################################
```

A.1.2 Code for Chapter 3

```
######################################################################

### generate data for simulations

## vary N where value of nonsparse elements = 1
# number of simulations
  iters = 30

  # setup parallel backend to use 8 processors
  cl <- makeCluster(8)
  registerDoParallel(cl)

  set.seed(1011)
  # vary N  (mu=1,mu2=0,sd1=1,sd2=1,nonsparse=10)
  # n=500
  train_n500_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=250,ss2=250,nonsparse=10,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }
  test_n500_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=250,ss2=250,nonsparse=10,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }

  # n=1000
  train_n1000_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=500,ss2=500,nonsparse=10,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }

  test_n1000_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=500,ss2=500,nonsparse=10,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }

  # n=5000
  train_n5000_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=2500,ss2=2500,nonsparse=10,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }
  test_n5000_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=2500,ss2=2500,nonsparse=10,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }
  stopCluster(cl)
  registerDoSEQ()


## vary N where value of nonsparse elements = 1/sqrt(m)
# number of simulations
  iters = 30

  # setup parallel backend to use 8 processors
  cl <- makeCluster(8)
  registerDoParallel(cl)

  set.seed(1011)
  # vary N  (mu=1,mu2=0,sd1=1,sd2=1,nonsparse=10)
  # n=200
  train_n200_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=10,
          mu1=0,mu2=1/sqrt(10),sd1=1,sd2=1)
  }
  test_n200_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=10,
          mu1=0,mu2=1/sqrt(10),sd1=1,sd2=1)
  }

  # n=500
  train_n500_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=250,ss2=250,nonsparse=10,
          mu1=0,mu2=1/sqrt(10),sd1=1,sd2=1)
  }
  test_n500_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=250,ss2=250,nonsparse=10,
          mu1=0,mu2=1/sqrt(10),sd1=1,sd2=1)
  }

  # n=1000
  train_n1000_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=500,ss2=500,nonsparse=10,
          mu1=0,mu2=1/sqrt(10),sd1=1,sd2=1)
  }
  test_n1000_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=500,ss2=500,nonsparse=10,
          mu1=0,mu2=1/sqrt(10),sd1=1,sd2=1)
  }

  # n=5000
  train_n5000_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=2500,ss2=2500,nonsparse=10,
          mu1=0,mu2=1/sqrt(10),sd1=1,sd2=1)
  }
  test_n5000_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=2500,ss2=2500,nonsparse=10,
          mu1=0,mu2=1/sqrt(10),sd1=1,sd2=1)
  }
  stopCluster(cl)
  registerDoSEQ()



## vary m where value of nonsparse elements = 1
  # number of simulations
  iters = 30

  # setup parallel backend to use 8 processors
  cl <- makeCluster(8)
  registerDoParallel(cl)

  set.seed(1011)

  # vary m  (mu=1,sd1=1,sd2=1,n1=100,n2=100)
  # nonsparse elements=5
  train_s5_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=5,mu1=0,mu2=1
    ,sd1=1,sd2=1)
  }
  test_s5_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=5,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }

  # nonsparse elements=20
  train_s20_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=20,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }
  test_s20_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=20,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }

  # nonsparse elements=30
  train_s30_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=30,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }
  test_s30_b1 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=30,mu1=0,mu2=1,
    sd1=1,sd2=1)
  }

  stopCluster(cl)
  registerDoSEQ()


## vary m where value of nonsparse elements = 1/sqrt(m)
  # number of simulations
  iters = 30

  # setup parallel backend to use 8 processors
  cl <- makeCluster(8)
  registerDoParallel(cl)

  set.seed(1011)

  # vary m  (mu=1,sd1=1,sd2=1,n1=100,n2=100)
  # nonsparse elements=5
  train_s5_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=5,mu1=0,
          mu2=1/sqrt(5),sd1=1,sd2=1)
  }
  test_s5_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=5,mu1=0,
          mu2=1/sqrt(5),sd1=1,sd2=1)
  }

  # nonsparse elements=10
  train_s10_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=5,mu1=0,
          mu2=1/sqrt(10),sd1=1,sd2=1)
  }
  test_s10_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=5,mu1=0,
          mu2=1/sqrt(10),sd1=1,sd2=1)
  }

  # nonsparse elements=20
  train_s20_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=20,mu1=0,
          mu2=1/sqrt(20),sd1=1,sd2=1)
  }
  test_s20_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=20,mu1=0,
          mu2=1/sqrt(20),sd1=1,sd2=1)
  }

  # nonsparse elements=30
  train_s30_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=30,mu1=0,
          mu2=1/sqrt(30),sd1=1,sd2=1)
  }
  test_s30_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=30,mu1=0,
          mu2=1/sqrt(30),sd1=1,sd2=1)
  }

  stopCluster(cl)
  registerDoSEQ()



## vary m : pmax  where value of nonsparse elements = 1
  # number of simulations
  iters = 30

  # setup parallel backend to use 8 processors
  cl <- makeCluster(8)
  registerDoParallel(cl)

  set.seed(1011)
```

```
# vary m and p (mu=1,mu2=0,sd1=1,sd2=1,n=200)

# p=15000, s=30
train_p15000s30_b1 = foreach(i=1:iters) %dorng% {
  draw3(d=15000,ss1=100,ss2=100,nonsparse=30,mu1=0,mu2=1,
  sd1=1,sd2=1)
}
test_p15000s30_b1 = foreach(i=1:iters) %dorng% {
  draw3(d=15000,ss1=100,ss2=100,nonsparse=30,mu1=0,mu2=1,
  sd1=1,sd2=1)
}

# p=25000, s=50
train_p25000s50_b1 = foreach(i=1:iters) %dorng% {
  draw3(d=25000,ss1=100,ss2=100,nonsparse=50,mu1=0,mu2=1,
  sd1=1,sd2=1)
}
test_p25000s50_b1 = foreach(i=1:iters) %dorng% {
  draw3(d=25000,ss1=100,ss2=100,nonsparse=50,mu1=0,mu2=1,
  sd1=1,sd2=1)
}

# p=35000, s=70
train_p35000s70_b1 = foreach(i=1:iters) %dorng% {
  draw3(d=35000,ss1=100,ss2=100,nonsparse=70,mu1=0,mu2=1,
  sd1=1,sd2=1)
}
test_p35000s70_b1 = foreach(i=1:iters) %dorng% {
  draw3(d=35000,ss1=100,ss2=100,nonsparse=70,mu1=0,mu2=1,
  sd1=1,sd2=1)
}

# p=45000, s=90
train_p45000s90_b1 = foreach(i=1:iters) %dorng% {
  draw3(d=45000,ss1=100,ss2=100,nonsparse=90,mu1=0,mu2=1,
  sd1=1,sd2=1)
}
test_p45000s90_b1 = foreach(i=1:iters) %dorng% {
  draw3(d=45000,ss1=100,ss2=100,nonsparse=90,mu1=0,mu2=1,
  sd1=1,sd2=1)
}
```

```
  stopCluster(cl)
  registerDoSEQ()


## vary m : pmax  where value of nonsparse elements = 1/sqrt(m)
  # number of simulations
  iters = 30

  # setup parallel backend to use 8 processors
  cl <- makeCluster(8)
  registerDoParallel(cl)

  set.seed(1011)

  # vary m and P (mu=1,mu2=0,sd1=1,sd2=1,n=200)
  # p=5000, s=10
  train_p5000s10_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=20,mu1=0,
          mu2=1/sqrt(10),sd1=1,sd2=1)
  }



  test_p5000s10_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=5000,ss1=100,ss2=100,nonsparse=20,mu1=0,
          mu2=1/sqrt(10),sd1=1,sd2=1)
  }

  # p=15000, s=30
  train_p15000s30_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=15000,ss1=100,ss2=100,nonsparse=30,mu1=0,
          mu2=1/sqrt(30),sd1=1,sd2=1)
  }
  test_p15000s30_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=15000,ss1=100,ss2=100,nonsparse=30,mu1=0,
          mu2=1/sqrt(30),sd1=1,sd2=1)
  }

  # p=25000, s=50
  train_p25000s50_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=25000,ss1=100,ss2=100,nonsparse=50,mu1=0,
          mu2=1/sqrt(50),sd1=1,sd2=1)
  }
  test_p25000s50_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=25000,ss1=100,ss2=100,nonsparse=50,mu1=0,
          mu2=1/sqrt(50),sd1=1,sd2=1)
  }

  # p=35000, s=70
  train_p35000s70_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=35000,ss1=100,ss2=100,nonsparse=70,mu1=0,
          mu2=1/sqrt(70),sd1=1,sd2=1)
  }
  test_p35000s70_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=35000,ss1=100,ss2=100,nonsparse=70,mu1=0,
          mu2=1/sqrt(70),sd1=1,sd2=1)
  }

  # p=45000, s=90
  train_p45000s90_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=45000,ss1=100,ss2=100,nonsparse=90,mu1=0,
          mu2=1/sqrt(90),sd1=1,sd2=1)
  }
  test_p45000s90_b2 = foreach(i=1:iters) %dorng% {
    draw3(d=45000,ss1=100,ss2=100,nonsparse=90,mu1=0,
          mu2=1/sqrt(90),sd1=1,sd2=1)
  }

  stopCluster(cl)
  registerDoSEQ()

######################################################################

### simulations to vary sample size

## vary N  where value of nonsparse elements = 1

# every 100th predictor between 2 and 5000
a1 = seq(0,100,1)
a2 = seq(0,5000,100)
a = c(a1[3:101],a2[3:51])

g = c(rep(1,250),rep(2,250))
a1_n500_b1 = run.rft(train=train_n500_b1,test=test_n500_b1,
       n=30,d=max(a))
```

```
g = c(rep(1,500),rep(2,500))
a1_n1000_b1  = run.rft(train=train_n1000_b1,test=test_n1000_b1,
       n=30,d=max(a))

g = c(rep(1,2500),rep(2,2500))
a1_n5000_b1  = run.rft(train=train_n5000_b1,test=test_n5000_b1,
       n=30,d=max(a))


## vary N  where value of nonsparse elements = 1/sqrt(m)
# every 100th predictor between 2 and 5000
a1 = seq(0,100,1)
a2 = seq(0,5000,100)
a = c(a1[3:101],a2[3:51])

g = c(rep(1,100),rep(2,100))
a1_n200_b2  = run.rft(train=train_n200_b2,test=test_n200_b2,
       n=30,d=max(a))

g = c(rep(1,250),rep(2,250))
a1_n500_b2  = run.rft(train=train_n500_b2,test=test_n500_b2,
       n=30,d=max(a))

g = c(rep(1,500),rep(2,500))
a1_n1000_b2  = run.rft(train=train_n1000_b2,test=test_n1000_b2,
       n=30,d=max(a))

g = c(rep(1,2500),rep(2,2500))
a1_n5000_b2  = run.rft(train=train_n5000_b2,test=test_n5000_b2,
       n=30,d=max(a))



### simulations to vary signal strength
# every 100th predictor between 2 and 5000
a1 = seq(0,100,1)
a2 = seq(0,5000,100)
a = c(a1[3:101],a2[3:51])

g = c(rep(0,100),rep(1,100))
```

```
## vary m where value of nonsparse elements = 1
a1_s5_b1 = run.rft(train=train_s5_b1,test=test_s5_b1
       ,n=30,d=max(a))
a1_s20_b1 = run.rft(train=train_s20_b1,test=test_s20_b1,
       n=30,d=max(a))
a1_s30_b1 = run.rft(train=train_s30_b1,test=test_s30_b1,
       n=30,d=max(a))

## vary m where value of nonsparse elements = 1/sqrt(m)
a1_s5_b2 = run.rft(train=train_s5_b2,test=test_s5_b2,
       n=30,d=max(a))
a1_s10_b2 = run.rft(train=train_s10_b2,test=test_s10_b2,
       n=30,d=max(a))
a1_s20_b2 = run.rft(train=train_s20_b2,test=test_s20_b2,
       n=30,d=max(a))
a1_s30_b2 = run.rft(train=train_s30_b2,test=test_s30_b2,
       n=30,d=max(a))



### simulations to vary m : pmax  where value of
### nonsparse elements = 1
g = c(rep(0,100),rep(1,100))

# m : pmax = 30 : 15000 for every 100th predictor, p = 2 -15000
a1 = seq(0,100,1)
a2 = seq(0,15000,100)
a = c(a1[3:101],a2[3:151])
a1_p15000s30_b1 = run.rft(train=train_p15000s30_b1,
                          test=test_p15000s30_b1 n=30,d=max(a))

# m : pmax = 50 : 25000 for every 100th predictor, p = 2 - 25000
a1 = seq(0,100,1)
a2 = seq(0,25000,100)
a = c(a1[3:101],a2[3:251])
a1_p25000s50_b1 = run.rft(train=train_p25000s50_b1,
                          test=test_p25000s50_b1,n=30,d=max(a))

# m : pmax = 70 : 35000 for every 100th predictor, p = 2 - 35000
a1 = seq(0,100,1)
a2 = seq(0,35000,100)
a = c(a1[3:101],a2[3:351])
```

```
a1_p35000s70_b1 = run.rft(train=train_p35000s70_b1,
                          test=test_p35000s70_b1,n=30,d=max(a))

# m : pmax = 90 : 45000 for every 100th predictor p = 2 - 45000
a1 = seq(0,100,1)
a2 = seq(0,45000,100)
a = c(a1[3:101],a2[3:451])
a1_p45000s90_b1 = run.rft(train=train_p45000s90_b1,
                          test=test_p45000s90_b1,n=30,d=max(a))


### simulations to vary m : pmax where value of
### nonsparse elements = 1/sqrt(m)
g = c(rep(0,100),rep(1,100))

# m : pmax = 10 : 5000 for every 100th predictor, p = 2 - 5000
a1 = seq(0,100,1)
a2 = seq(0,5000,100)
a = c(a1[3:101],a2[3:151])
a1_p5000s10_b1 = run.rft(train=train_p5000s10_b2,
                          test=test_p5000s10_b2 n=30,d=max(a))

# m : pmax = 30 : 15000 for every 100th predictor,  p = 2 - 25000
a1 = seq(0,100,1)
a2 = seq(0,15000,100)
a = c(a1[3:101],a2[3:151])
a1_p15000s30_b2 = run.rft(train=train_p15000s30_b2,
                          test=test_p15000s30_b2 n=30,d=max(a))

# m : pmax = 50 : 25000 for every 100th predictor, p = 2 - 25000
a1 = seq(0,100,1)
a2 = seq(0,25000,100)
a = c(a1[3:101],a2[3:251])
a1_p25000s50_b2 = run.rft(train=train_p25000s50_b2,
                          test=test_p25000s50_b2,n=30,d=max(a))

# m : pmax = 70 : 35000 for every 100th predictor, p = 2 - 35000
a1 = seq(0,100,1)
a2 = seq(0,35000,100)
a = c(a1[3:101],a2[3:351])
a1_p35000s70_b2 = run.rft(train=train_p35000s70_b2,
                          test=test_p35000s70_b2,n=30,d=max(a))
```

```
# m : pmax = 90 : 45000 for every 100th predictor,  p = 2 - 45000
a1 = seq(0,100,1)
a2 = seq(0,45000,100)
a = c(a1[3:101],a2[3:451])
a1_p45000s90_b2 = run.rft(train=train_p45000s90_b2,
                          test=test_p45000s90_b2,n=30,d=max(a))

######################################################################

### Results

## make baseline for value of nonsparse elements = 1
b1_train1 = do.call(rbind,rf_a4_detail$s1)
b1_test1= do.call(rbind,rf_a4_detail$s2)

b1 = seq(0,5000,10)
rf_a4_foo.test = list()
rf_a4_use.test = list()
for(j in 1:100){
  rf_a4_foo.test[[j]] = rf_a4$s2[[j]][12:501]
  i = 1
  rf_a4_use.test[[j]] = rf_a4_foo.test[[j]][1:(i+9)==(i+9)]
}
b1_test2 = do.call(rbind,rf_a4_use.test)

tq.test2a = apply(b1_test1,2,median)
tq.test2b = apply(b1_test2,2,median)
base.test.50= c(tq.test2a,tq.test2b)

nq.test1.p1 = apply(t.test2a,2,quantile,probs=c(.1,.9))
nq.test1.p2 = apply(t.test2b,2,quantile,probs=c(.1,.9))
base.test.10 = c(nq.test1.p1[1,],nq.test1.p2[1,])
base.test.90 = c(nq.test1.p1[2,],nq.test1.p2[2,])



## Vary N

a1 = seq(0,100,1)
a2 = seq(0,5000,100)
a = c(a1[3:101],a2[3:51])
```

```
## nonsparse elements = 1
s.test2  = do.call(rbind,a1_n500_b1$s2)
s.test3  = do.call(rbind,a1_n1000_b1$s2)
s.test4  = do.call(rbind,a1_n5000_b1$s2)

n500_b1.p = apply(s.test2,2, quantile,probs=c(.1,.5,.9))
n1000_b1.p = apply(s.test3,2, quantile,probs=c(.1,.5,.9))
n5000_b1.p = apply(s.test4,2, quantile,probs=c(.1,.5,.9))

base.test.50[1:9]
base.test.10[1:9]
base.test.90[1:9]
base.test.50[148]
base.test.10[148]
base.test.90[148]

n500_b1.p[2,1:9]
n500_b1.p[1,1:9]
n500_b1.p[3,1:9]
n500_b1.p[2,148]
n500_b1.p[1,148]
n500_b1.p[3,148]

n1000_b1.p[2,1:9]
n1000_b1.p[1,1:9]
n1000_b1.p[3,1:9]
n1000_b1.p[2,148]
n1000_b1.p[1,148]
n1000_b1.p[3,148]

n5000_b1.p[2,1:9]
n5000_b1.p[1,1:9]
n5000_b1.p[3,1:9]
n5000_b1.p[2,148]
n5000_b1.p[1,148]
n5000_b1.p[3,148]


## nonsparse elements = 1/sqrt(m)
t.test1  = do.call(rbind,a1_n200_b2$s2)
t.test2  = do.call(rbind,a1_n500_b2$s2)
t.test3  = do.call(rbind,a1_n1000_b2$s2)
t.test4  = do.call(rbind,a1_n5000_b2$s2)
```

```
n100_b2.p = apply(t.test1,2, quantile,probs=c(.1,.5,.9))
n500_b2.p = apply(t.test2,2, quantile,probs=c(.1,.5,.9))
n1000_b2.p = apply(t.test3,2, quantile,probs=c(.1,.5,.9))
n5000_b2.p = apply(t.test4,2, quantile,probs=c(.1,.5,.9))

n200_b2.p[2,1:9]
n200_b2.p[1,1:9]
n200_b2.p[3,1:9]
n200_b2.p[2,148]
n200_b2.p[1,148]
n200_b2.p[3,148]

n500_b2.p[2,1:9]
n500_b2.p[1,1:9]
n500_b2.p[3,1:9]
n500_b2.p[2,148]
n500_b2.p[1,148]
n500_b2.p[3,148]

n1000_b2.p[2,1:9]
n1000_b2.p[1,1:9]
n1000_b2.p[3,1:9]
n1000_b2.p[2,148]
n1000_b2.p[1,148]
n1000_b2.p[3,148]

n5000_b2.p[2,1:9]
n5000_b2.p[1,1:9]
n5000_b2.p[3,1:9]
n5000_b2.p[2,148]
n5000_b2.p[1,148]
n5000_b2.p[3,148]



## Vary m

## nonsparse elements = 1
u.test1 = do.call(rbind,a1_s5_b1$s2)
u.test3 = do.call(rbind,a1_s20_b1$s2)
u.test4 = do.call(rbind,a1_s30_b1$s2)
```

```
s5_b1.p = apply(u.test1,2,quantile,probs=c(.1,.5,.9))
s20_b1.p = apply(u.test3,2, quantile,probs=c(.1,.5,.9))
s30_b1.p = apply(u.test4,2, quantile,probs=c(.1,.5,.9))

s5_b1.p[1,1:9]
s5_b1.p[3,1:9]
s5_b1.p[2,148]
s5_b1.p[1,148]
s5_b1.p[3,148]

base.test.50[1:9]
base.test.10[1:9]
base.test.90[1:9]
base.test.50[148]
base.test.10[148]
base.test.90[148]

s20_b1.p[1,1:9]
s20_b1.p[3,1:9]
s20_b1.p[2,148]
s20_b1.p[1,148]
s20_b1.p[3,148]

s30_b1.p[1,1:9]
s30_b1.p[3,1:9]
s30_b1.p[2,148]
s30_b1.p[1,148]
s30_b1.p[3,148]


## nonsparse elements = 1/sqrt(m)
v.test1 = do.call(rbind,a1_s5_b2$s2)
v.test2 = do.call(rbind,a1_s10_b2$s2)
v.test3 = do.call(rbind,a1_s20_b2$s2)
v.test4 = do.call(rbind,a1_s30_b2$s2)

s5_b2.p = apply(v.test1,2,quantile,probs=c(.1,.5,.9))
s10_b2.p = apply(v.test3,2, quantile,probs=c(.1,.5,.9))
s20_b2.p = apply(v.test3,2, quantile,probs=c(.1,.5,.9))
s30_b2.p = apply(v.test4,2, quantile,probs=c(.1,.5,.9))

s5_b2.p[1,1:9]
s5_b2.p[3,1:9]
```

```
s5_b2.p[2,148]
s5_b2.p[1,148]
s5_b2.p[3,148]

s10_b2.p[1,1:9]
s10_b2.p[3,1:9]
s10_b2.p[2,148]
s10_b2.p[1,148]
s10_b2.p[3,148]

s20_b2.p[1,1:9]
s20_b2.p[3,1:9]
s20_b2.p[2,148]
s20_b2.p[1,148]
s20_b2.p[3,148]

s30_b2.p[1,1:9]
s30_b2.p[3,1:9]
s30_b2.p[2,148]
s30_b2.p[1,148]
s30_b2.p[3,148]



## Vary m : pmax

## nonsparse elements = 1
w.test2  = do.call(rbind,a1_p15000s30_b1$s2)
w.test3  = do.call(rbind,a1_p25000s50_b1$s2)
w.test4  = do.call(rbind, a1_p35000s70_b1$s2)
w.test5  = do.call(rbind, a1_p45000s90_b1$s2)

s30p15000_b1.p = apply(w.test2,2, quantile,probs=c(.1,.5,.9))
s50p25000_b1.p = apply(w.test3,2, quantile,probs=c(.1,.5,.9))
s70p35000_b1.p = apply(w.test4,2, quantile,probs=c(.1,.5,.9))
s90p45000_b1.p = apply(w.test5,2,quantile,probs=c(.1,.5,.9))

base.test.50[1:9]
base.test.10[1:9]
base.test.90[1:9]
base.test.50[148]
base.test.10[148]
base.test.90[148]
```

```
s30p15000_b1.p [2,1:9]
s30p15000_b1.p [1,1:9]
s30p15000_b1.p [3,1:9]
s30p15000_b1.p [2,248]
s30p15000_b1.p [1,248]
s30p15000_b1.p [3,248]

s50p25000_b1.p [2,1:9]
s50p25000_b1.p [1,1:9]
s50p25000_b1.p [3,1:9]
s50p25000_b1.p [2,248]
s50p25000_b1.p [1,248]
s50p25000_b1.p [3,248]

s70p35000_b1.p [2,1:9]
s70p35000_b1.p [1,1:9]
s70p35000_b1.p [3,1:9]
s70p35000_b1.p [2,248]
s70p35000_b1.p [1,248]
s70p35000_b1.p [3,248]

s90p45000_b1.p [2,1:9]
s90p45000_b1.p [1,1:9]
s90p45000_b1.p [3,1:9]
s90p45000_b1.p [2,248]
s90p45000_b1.p [1,248]
s90p45000_b1.p [3,248]


## nonsparse elements = 1/sqrt(m)
x.test1 = do.call(rbind,a1_p15000s30_b2$s2)
x.test2 = do.call(rbind,a1_p15000s30_b2$s2)
x.test3 = do.call(rbind,a1_p25000s50_b2$s2)
x.test4 = do.call(rbind, a1_p35000s70_b2$s2)
x.test5 = do.call(rbind, a1_p45000s90_b2$s2)

s10p5000_b2.p = apply(x.test1,2, quantile,probs=c(.1,.5,.9))
s30p15000_b2.p = apply(x.test2,2, quantile,probs=c(.1,.5,.9))
s50p25000_b2.p = apply(x.test3,2, quantile,probs=c(.1,.5,.9))
s70p35000_b2.p = apply(x.test4,2, quantile,probs=c(.1,.5,.9))
s90p45000_b2.p = apply(x.test5,2,quantile,probs=c(.1,.5,.9))
```

```
s10p5000_b2.p [2,1:9]
s10p5000_b2.p [1,1:9]
s10p5000_b2.p [3,1:9]
s10p5000_b2.p [2,248]
s10p5000_b2.p [1,248]
s10p5000_b2.p [3,248]

s30p15000_b2.p [2,1:9]
s30p15000_b2.p [1,1:9]
s30p15000_b2.p [3,1:9]
s30p15000_b2.p [2,248]
s30p15000_b2.p [1,248]
s30p15000_b2.p [3,248]

s50p25000_b2.p [2,1:9]
s50p25000_b2.p [1,1:9]
s50p25000_b2.p [3,1:9]
s50p25000_b2.p [2,248]
s50p25000_b2.p [1,248]
s50p25000_b2.p [3,248]

s70p35000_b2.p [2,1:9]
s70p35000_b2.p [1,1:9]
s70p35000_b2.p [3,1:9]
s70p35000_b2.p [2,248]
s70p35000_b2.p [1,248]
s70p35000_b2.p [3,248]

s90p45000_b2.p [2,1:9]
s90p45000_b2.p [1,1:9]
s90p45000_b2.p [3,1:9]
s90p45000_b2.p [2,248]
s90p45000_b2.p [1,248]
s90p45000_b2.p [3,248]

######################################################################
```

```
### Graphs

# function to make space for legend
reset <- function() {
    par(mfrow=c(1, 1), oma=rep(0, 4), mar=rep(0, 4), new=TRUE)
    plot(0:1, 0:1, type="n", xlab="", ylab="", axes=FALSE)
    }



## Vary N
## nonsparse elements = 1
col2 = c("darkgoldenrod1","#005800","#CFBFB6","#723900")
pdf("RF_VaryN_b1.pdf")
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1))
plot(a,base.test.50,type='l', lwd=3, ylim=c(0,.6),
     ylab='Test error',xlab='Predictors',col=col2[1],cex.lab=1.3)
lines(a,n500_b1.p[2,],type='l', lwd=3, col=col2[2])
lines(a,n1000_b1.p[2,],type='l', lwd=3, col=col2[3])
lines(a,n5000_b1.p[2,],type='l', lwd=3, col=col2[4])
reset()
legend(x = "bottom",inset = 0,
        legend = c("n=200","n=500","n=1000","n=5000"),
        col=c(col2[1],col2[2],col2[3],col2[4]),
        lwd=3, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()


## nonsparse elements = 1/sqrt(m)
col2a = c("lightgoldenrod2","#005800","#CFBFB6","#723900")
pdf("RF_VaryN_b2.pdf")
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1))
plot(a,n200_b2.p[2,],type='l', lwd=3, ylim=c(0,.6),
     ylab='Test error',xlab='Predictors',col=col2a[1],cex.lab=1.3)
lines(a,n500_b2.p[2,],type='l', lwd=3, col=col2a[2])
lines(a,n1000_b2.p[2,],type='l', lwd=3, col=col2a[3])
lines(a,n5000_b2.p[2,],type='l', lwd=3, col=col2a[4])
reset()
legend(x = "bottom",inset = 0,
        legend = c("n=200","n=500","n=1000","n=5000"),
        col=c(col2a[1], col2a[2], col2a[3], col2a[4]),
        lwd=3, horiz = TRUE)
```

```
par(oma=rep(0, 4))
dev.off()



## Vary m
## nonsparse elements = 1
col1=c("#155DB5","darkgoldenrod1","#D9C1CF","#AB1E84")
pdf("RF_VaryM_b1.pdf")
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1))
plot(a,s5_b1.p[2,],type='l', lwd=3, ylim=c(0,.6),
     ylab='Test error',xlab='Predictors',col=col1[1],cex.lab=1.3)
lines(a,base.test.50,type='l', lwd=3, col=col1[2],cex.lab=1.3)
lines(a,s20_b1.p[2,],type='l', lwd=3, col=col1[3],cex.lab=1.3)
lines(a,s30_b1.p[2,],type='l', lwd=3, col=col1[4],cex.lab=1.3)
reset()
legend(x = "bottom",inset = 0,
        legend = c("m=5","m=10","m=20","m=30"),
        col=col1, lwd=3, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()


## nonsparse elements = 1/sqrt(m)
col1a=c("#155DB5","lightgoldenrod3","#D9C1CF","#AB1E84")
pdf("RF_VaryM_b2.pdf")
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1))
plot(a,ts5_b2.p,type='l', lwd=3, ylim=c(0,.6),
     ylab='Test error',xlab='Predictors',col=col1a[1],cex.lab=1.3)
lines(a,s10_b2.p,type='l', lwd=3, col=col1a[2],cex.lab=1.3)
lines(a,s20_b2.p,type='l', lwd=3, col=col1a[3],cex.lab=1.3)
lines(a,s30_b2.p,type='l', lwd=3, col=col1a[4],cex.lab=1.3)
reset()
legend(x = "bottom",inset = 0,
           legend = c("m=5","m=10","m=20","m=30"),
           col=col1a,
           lwd=3, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()
```

```
## Vary m : pmax
# sequences
b2a = seq(0,15000,100)
b2 = c(a1[3:101],b2a[3:151])

b3a = seq(0,25000,100)
b3 = c(a1[3:101],b3a[3:251])

b4a = seq(0,35000,100)
b4 = c(a1[3:101],b4a[3:351])

b5a = seq(0,45000,100)
b5 = c(a1[3:101],b5a[3:451])


## nonsparse elements = 1
col3 = c("darkgoldenrod1","#A1DAB4","#41B6C4",
         "#2C7FB8","#253494")
pdf("RF_VarySP_b1.pdf")
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1))
plot(b5,s90p45000_b1.p[2,],type='l', lwd=3, ylim=c(0,.6),
     ylab='Test error',xlab='Predictors',col=col3[5],cex.lab=1.3)
lines(b4,s70p35000_b1.p[2,],type='l', lwd=3, col=col3[4])
lines(b3,s50p25000_b1.p[2,],type='l', lwd=3, col=col3[3])
lines(b2,s30p15000_b1.p[2,],type='l', lwd=3, col=col3[2])
lines(a,base.test.50,type='l', lwd=3, col=col3[1])
reset()
legend(x = "bottom",inset = 0,
        legend = c("10:5000","30:15000","50:25000",
                   "70:35000","90:45000"),
        col=c(col3[1],col3[2],col3[3],col3[4],col3[5]),
        lwd=3, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()


## nonsparse elements = 1/sqrt(m)
col3a = c("lightgoldenrod2","#A1DAB4","#41B6C4",
         "#2C7FB8","#253494")
pdf("RF_VarySP_b2.pdf")
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1))
plot(b5,s90p45000_b2.p[2,],type='l', lwd=3, ylim=c(0,.6),
     ylab='Test error',xlab='Predictors',col=col3a[5],cex.lab=1.3)
```

```
lines(b4,s70p35000_b2.p[2,],type='l', lwd=3, col=col3a[3])
lines(b3,s50p25000_b2.p[2,],type='l', lwd=3, col=col3a[3])
lines(b2,s30p15000_b2.p[2,],type='l', lwd=3, col=col3a[2])
lines(a,s10p5000_b2.p[2,],type='l', lwd=3, col=col3a[1])
reset()
legend(x = "bottom",inset = 0,
        legend = c("10:5000","30:15000","50:25000",
                   "70:35000","90:45000"),
        col=c(col3a[1],col3a[2], col3a[3], col3a[4], col3a[5]),
        lwd=3, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()

######################################################################
```

A.1.3 Code for Chapter 4

```
######################################################################
library(mvtnorm)


### functions

## sample 2 groups from MVN distribution w/independent predictors
draw3 = function(d,ss1,ss2,sd1=1,sd2=1,nonsparse,mu1,mu2){

  n21 = nonsparse
  # class 2
  y1a = matrix(rnorm(n=ss2*n21,mean=mu2,sd=sd2),nrow=ss2,ncol=n21,
        byrow=FALSE)
  y1b = matrix(rnorm(n=ss2*(d-n21),mean=0,sd=1),nrow=ss2,ncol=d-n21,
        byrow=FALSE)
  y1 = cbind(y1a,y1b)

  # class 1
  x1a = matrix(rnorm(n=ss1*n21,mean=mu1,sd=sd1),nrow=ss1,ncol=n21,
        byrow=FALSE)
  x1b = matrix(rnorm(n=ss1*(d-n21),mean=0,sd=1),nrow=ss1,ncol=d-n21,
        byrow=FALSE)
  x1 = cbind(x1a,x1b)

  sdata = rbind(y1,x1)
  return(sdata=sdata)
}


## sample 2 groups from MVN distribution w/correlated predictors,
## variance in groups = 1
draw4 = function(d,ss1,ss2,rho,sd1=1,sd2=1,nonsparse,nz1,nz2){
mu1 = c(rep(nz1,nonsparse),rep(0,d-nonsparse))
mu2 = c(rep(nz2,nonsparse),rep(0,d-nonsparse))

sigma1a = matrix(rho,nrow=nonsparse,ncol= nonsparse,byrow=F)
sigma1b = matrix(0,nrow=nonsparse,ncol=d-nonsparse,byrow=F)
sigma1c = matrix(0,nrow=d-nonsparse,ncol=d,byrow=F)
sigma1 = rbind(cbind(sigma1a,sigma1b),sigma1c)
diag(sigma1) = sd1^2
sigma2a = matrix(rho,nrow=nonsparse,ncol= nonsparse,byrow=F)
sigma2b = matrix(0,nrow=nonsparse,ncol=d-nonsparse,byrow=F)
sigma2c = matrix(0,nrow=d-nonsparse,ncol=d,byrow=F)
sigma2 = rbind(cbind(sigma1a,sigma1b),sigma1c)
diag(sigma2) = sd2^2

y1=rmvnorm(ss1,mean=mu1,sigma=sigma1)
x1=rmvnorm(ss2,mean=mu2,sigma=sigma2)

  sdata = rbind(y1,x1)
  return(sdata=sdata)
}


## theoretical TSI ##
TSI = function(mu1,mu2,sd1,sd2){
  f1 = mu1/sd1
  f2 = mu2/sd2
  f = cumsum((f1-f2)^2)
  tsi = sqrt(f)
  return(tsi)
}

## empirical TSI ##
TSIe = function(g1,g2){
 mu1 = colMeans(g1)
 mu2 = colMeans(g2)
 sd1 = apply(g1,2,sd)
 sd2 = apply(g2,2,sd)

  f1 = mu1/sd1
  f2 = mu2/sd2
  f = cumsum((f1-f2)^2)
  tsi = sqrt(f)
  return(tsi)
}

######################################################################

### theoretical TSI computations where sd1=1 and sd2=1
# 10 nonsparse elements where mu1=1, m1=0
mu1 = c(rep(1,10),rep(0,4990))
```

```
mu2 = c(rep(0,10),rep(0,4990))
sg1=rep(1,5000)
sg2=rep(1,5000)
t1 = TSI(mu1,mu2,sg1,sg2)

# 10 nonsparse elements where mu1 = 3, mu2 = -1
mu1 = c(rep(3,10),rep(0,4990))
mu2 = c(rep(-1,10),rep(0,4990))
sg1=rep(1,5000)
sg2=rep(1,5000)
t3m1 = TSI(mu1,mu2,sg1,sg2)

# 10 scrambled nonsparse elements where mu1=1 and m1=0
mu1 = c(rep(1,10),rep(0,4990))
set.seed(42)
mu1.s = sample(mu1)
mu2 = c(rep(0,10),rep(0,4990))
sg1=rep(1,5000)
sg2=rep(1,5000)
t1s = TSI(mu1.s,mu2,sg1,sg2)

######################################################################

## empirical simulations

## simulations with uncorrelated predictors
## sample size = 200
t1e = matrix(NA,ncol=5000,nrow=100)
t3m1e = matrix(NA,ncol=5000,nrow=100)
t1se = matrix(NA,ncol=5000,nrow=100)
t3n10e = matrix(NA,ncol=5000,nrow=100)
t3n6e = matrix(NA,ncol=5000,nrow=100)
t3n2e = matrix(NA,ncol=5000,nrow=100)
for(i in 1:100){
n1 = 100
n2 = 100
m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=1,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=-1)
g1=m0[1:n1,]
```

```
g2=m0[(n1+1):(n1+n2),]
t3m1e[i,] = TSIe(g1,g2)

m0 = rbind(
   x = sapply(mu1.s,function(x)rnorm(n1,mean=x,sd=1)),
   y = matrix(rnorm(n=n2,mean=0,sd=1),nrow=n2,ncol=5000,byrow=FALSE)
       )
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1se[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n10e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=6,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n6e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=2,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n2e[i,] = TSIe(g1,g2)
}
s1 = data.frame(t1e.min = apply(t1e,2,min),
t3m1e.min = apply(t3m1e,2,min),
t1se.min = apply(t1se,2,min),
t3n10e.min = apply(t3n10e,2,min),
t3n6e.min = apply(t3n6e,2,min),
t3n2e.min = apply(t3n2e,2,min),
t1e.med = apply(t1e,2,median),
t3m1e.med = apply(t3m1e,2,median),
t1se.med = apply(t1se,2,median),
t3n10e.med = apply(t3n10e,2,median),
t3n6e.med = apply(t3n6e,2,median),
t3n2e.med = apply(t3n2e,2,median),
t1e.max = apply(t1e,2,max),
t3m1e.max = apply(t3m1e,2,max),
t1se.max = apply(t1se,2,max),
t3n10e.max = apply(t3n10e,2,max),
t3n6e.max = apply(t3n6e,2,max),
t3n2e.max = apply(t3n2e,2,max))


## sample size = 500
t1e = matrix(NA,ncol=5000,nrow=100)
t3m1e = matrix(NA,ncol=5000,nrow=100)
t1se = matrix(NA,ncol=5000,nrow=100)
t3n10e = matrix(NA,ncol=5000,nrow=100)
t3n6e = matrix(NA,ncol=5000,nrow=100)
t3n2e = matrix(NA,ncol=5000,nrow=100)
for(i in 1:100){
n1 = 250
n2 = 250
m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=1,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=-1)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3m1e[i,] = TSIe(g1,g2)

m0 = rbind(
   x = sapply(mu1.s,function(x)rnorm(n1,mean=x,sd=1)),
   y = matrix(rnorm(n=n2,mean=0,sd=1),nrow=n2,ncol=5000,byrow=FALSE)
      )
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1se[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n10e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=6,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n6e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=2,mu1=3,mu2=0)
g1=m0[1:n1,]
```

```
g2=m0[(n1+1):(n1+n2),]
t3n2e[i,] = TSIe(g1,g2)
}
s2 = data.frame(t1e.min = apply(t1e,2,min),
t3m1e.min = apply(t3m1e,2,min),
t1se.min = apply(t1se,2,min),
t3n10e.min = apply(t3n10e,2,min),
t3n6e.min = apply(t3n6e,2,min),
t3n2e.min = apply(t3n2e,2,min),
t1e.med = apply(t1e,2,median),
t3m1e.med = apply(t3m1e,2,median),
t1se.med = apply(t1se,2,median),
t3n10e.med = apply(t3n10e,2,median),
t3n6e.med = apply(t3n6e,2,median),
t3n2e.med = apply(t3n2e,2,median),
t1e.max = apply(t1e,2,max),
t3m1e.max = apply(t3m1e,2,max),
t1se.max = apply(t1se,2,max),
t3n10e.max = apply(t3n10e,2,max),
t3n6e.max = apply(t3n6e,2,max),
t3n2e.max = apply(t3n2e,2,max))


## sample size = 1000
t1e = matrix(NA,ncol=5000,nrow=100)
t3m1e = matrix(NA,ncol=5000,nrow=100)
t1se = matrix(NA,ncol=5000,nrow=100)
t3n10e = matrix(NA,ncol=5000,nrow=100)
t3n6e = matrix(NA,ncol=5000,nrow=100)
t3n2e = matrix(NA,ncol=5000,nrow=100)
for(i in 1:100){
n1 = 500
n2 = 500
m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=1,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=-1)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3m1e[i,] = TSIe(g1,g2)
```

```
m0 = rbind(
   x = sapply(mu1.s,function(x)rnorm(n1,mean=x,sd=1)),
   y = matrix(rnorm(n=n2,mean=0,sd=1),nrow=n2,ncol=5000,byrow=FALSE)
       )
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1se[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n10e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=6,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n6e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=2,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n2e[i,] = TSIe(g1,g2)
}
s3 = data.frame(t1e.min = apply(t1e,2,min),
t3m1e.min = apply(t3m1e,2,min),
t1se.min = apply(t1se,2,min),
t3n10e.min = apply(t3n10e,2,min),
t3n6e.min = apply(t3n6e,2,min),
t3n2e.min = apply(t3n2e,2,min),
t1e.med = apply(t1e,2,median),
t3m1e.med = apply(t3m1e,2,median),
t1se.med = apply(t1se,2,median),
t3n10e.med = apply(t3n10e,2,median),
t3n6e.med = apply(t3n6e,2,median),
t3n2e.med = apply(t3n2e,2,median),
t1e.max = apply(t1e,2,max),
t3m1e.max = apply(t3m1e,2,max),
t1se.max = apply(t1se,2,max),
t3n10e.max = apply(t3n10e,2,max),
t3n6e.max = apply(t3n6e,2,max),
t3n2e.max = apply(t3n2e,2,max))
```

```
## sample size = 5000
t1e = matrix(NA,ncol=5000,nrow=100)
t3m1e = matrix(NA,ncol=5000,nrow=100)
t1se = matrix(NA,ncol=5000,nrow=100)
t3n10e = matrix(NA,ncol=5000,nrow=100)
t3n6e = matrix(NA,ncol=5000,nrow=100)
t3n2e = matrix(NA,ncol=5000,nrow=100)
for(i in 1:100){
n1 = 2500
n2 = 2500
m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=1,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=-1)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3m1e[i,] = TSIe(g1,g2)

m0 = rbind(
   x = sapply(mu1.s,function(x)rnorm(n1,mean=x,sd=1)),
   y = matrix(rnorm(n=n2,mean=0,sd=1),nrow=n2,ncol=5000,byrow=FALSE)
       )
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1se[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n10e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=6,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n6e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=2,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n2e[i,] = TSIe(g1,g2)
}
```

```
s4 = data.frame(t1e.min = apply(t1e,2,min),
t3m1e.min = apply(t3m1e,2,min),
t1se.min = apply(t1se,2,min),
t3n10e.min = apply(t3n10e,2,min),
t3n6e.min = apply(t3n6e,2,min),
t3n2e.min = apply(t3n2e,2,min),
t1e.med = apply(t1e,2,median),
t3m1e.med = apply(t3m1e,2,median),
t1se.med = apply(t1se,2,median),
t3n10e.med = apply(t3n10e,2,median),
t3n6e.med = apply(t3n6e,2,median),
t3n2e.med = apply(t3n2e,2,median),
t1e.max = apply(t1e,2,max),
t3m1e.max = apply(t3m1e,2,max),
t1se.max = apply(t1se,2,max),
t3n10e.max = apply(t3n10e,2,max),
t3n6e.max = apply(t3n6e,2,max),
t3n2e.max = apply(t3n2e,2,max))


## sample size = 10000
t1e = matrix(NA,ncol=5000,nrow=100)
t3m1e = matrix(NA,ncol=5000,nrow=100)
t1se = matrix(NA,ncol=5000,nrow=100)
t3n10e = matrix(NA,ncol=5000,nrow=100)
t3n6e = matrix(NA,ncol=5000,nrow=100)
t3n2e = matrix(NA,ncol=5000,nrow=100)
for(i in 1:100){
n1 = 5000
n2 = 5000
m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=1,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=-1)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3m1e[i,] = TSIe(g1,g2)
m0 = rbind(
   x = sapply(mu1.s,function(x)rnorm(n1,mean=x,sd=1)),
   y = matrix(rnorm(n=n2,mean=0,sd=1),nrow=n2,ncol=5000,byrow=FALSE)
       )
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t1se[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=10,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n10e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=6,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n6e[i,] = TSIe(g1,g2)

m0 = draw3(d=5000,ss1=n1,ss2=n2,nonsparse=2,mu1=3,mu2=0)
g1=m0[1:n1,]
g2=m0[(n1+1):(n1+n2),]
t3n2e[i,] = TSIe(g1,g2)
}
s5 = data.frame(t1e.min = apply(t1e,2,min),
t3m1e.min = apply(t3m1e,2,min),
t1se.min = apply(t1se,2,min),
t3n10e.min = apply(t3n10e,2,min),
t3n6e.min = apply(t3n6e,2,min),
t3n2e.min = apply(t3n2e,2,min),
t1e.med = apply(t1e,2,median),
t3m1e.med = apply(t3m1e,2,median),
t1se.med = apply(t1se,2,median),
t3n10e.med = apply(t3n10e,2,median),
t3n6e.med = apply(t3n6e,2,median),
t3n2e.med = apply(t3n2e,2,median),
t1e.max = apply(t1e,2,max),
t3m1e.max = apply(t3m1e,2,max),
t1se.max = apply(t1se,2,max),
t3n10e.max = apply(t3n10e,2,max),
t3n6e.max = apply(t3n6e,2,max),
t3n2e.max = apply(t3n2e,2,max))
```

```
## simulation with correlated predictors
tc0e = matrix(NA,ncol=500,nrow=100)
tc25e = matrix(NA,ncol=500,nrow=100)
tc5e = matrix(NA,ncol=500,nrow=100)
tc75e = matrix(NA,ncol=500,nrow=100)
for(i in 1:100){
  n1 = 100
  n2 = 100
  set.seed(3456)
  m0 = draw4(d=500,ss1=100,ss2=100,rho=0,nonsparse=10,nz1=1,nz2=0)
  g1=m0[1:n1,]
  g2=m0[(n1+1):(n1+n2),]
  tc0e[i,] = TSIe(g1,g2)

  m0 = draw4(d=500,ss1=100,ss2=100,rho=.25,nonsparse=10,nz1=1,nz2=0)
  g1=m0[1:n1,]
  g2=m0[(n1+1):(n1+n2),]
  tc25e[i,] = TSIe(g1,g2)

  m0 = draw4(d=500,ss1=100,ss2=100,rho=.5,nonsparse=10,nz1=1,nz2=0)
  g1=m0[1:n1,]
  g2=m0[(n1+1):(n1+n2),]
  tc5e[i,] = TSIe(g1,g2)

  m0 = draw4(d=500,ss1=100,ss2=100,rho=.75,nonsparse=10,nz1=1,nz2=0)
  g1=m0[1:n1,]
  g2=m0[(n1+1):(n1+n2),]
  tc75e[i,] = TSIe(g1,g2)
 }
 s6 = data.frame(tc0e.min = apply(tc0e,2,min),
                tc25e.min = apply(tc25e,2,min),
                tc5e.min = apply(tc5e,2,min),
                tc75e.min = apply(tc75e,2,min),
                tc0e.med = apply(tc0e,2,median),
                tc25e.med = apply(tc25e,2,median),
                tc5e.med = apply(tc5e,2,median),
                      tc75e.med = apply(tc75e,2,median),
                tc0e.max = apply(tc0e,2,max),
                tc25e.max = apply(tc25e,2,max),
                tc5e.max = apply(tc5e,2,max),
                tc75e.max = apply(tc75e,2,max))
```

```
######################################################################

### graphics
library(colorspace)

## overlay empirical with theoretical simulations

# n = 200
pdf('TSI_n200.pdf')
par(mar=c(5.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:50,rev(1:50)),c(s1$t3m1e.min[1:50],
        rev(s1$t3m1e.max[1:50])),col="grey",border=F)
points(s1$t3m1e.med[1:50],type="l",col="black")
points(t3m1[1:50],type="l",col="red")
polygon(c(1:50,rev(1:50)),c(s1$t1e.min[1:50],
        rev(s1$t1e.max[1:50])),col="lightsteelblue1",border=F)
points(s1$t1e.med[1:50],type="l",col="blue")
points(t1[1:50],type="l",col="green")
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:5000,rev(1:5000)),c(s1$t3m1e.min[1:5000],
        rev(s1$t3m1e.max[1:5000])),col="grey",border=F)
points(s1$t3m1e.med[1:5000],type="l",col="black")
points(t3m1[1:5000],type="l",col="red")
polygon(c(1:5000,rev(1:5000)),c(s1$t1e.min[1:5000],
        rev(s1$t1e.max[1:5000])),col="lightsteelblue1",border=F)
points(s1$t1e.med[1:5000],type="l",col="blue")
points(t1[1:5000],type="l",col="green")
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI, Scenario 1",
                   "Empirical TSI, Scenario 2",
                   "Theoretical TSI, Scenario 1",
                   "Theoretical TSI, Scenario 2"),
        col=c("black","blue","red","green"),
        lwd=2,
        ncol=2)
par(oma=rep(0, 4))
dev.off()

# n = 500
pdf('TSI_n500.pdf')
par(mar=c(5.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
```

```
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:50,rev(1:50)),c(s2$t3m1e.min[1:50],
        rev(s2$t3m1e.max[1:50])),col="grey",border=F)
points(s2$t3m1e.med[1:50],type="l",col="black")
points(t3m1[1:50],type="l",col="red")
polygon(c(1:50,rev(1:50)),c(s2$t1e.min[1:50],
        rev(s2$t1e.max[1:50])),col="lightsteelblue1",border=F)
points(s2$t1e.med[1:50],type="l",col="blue")
points(t1[1:50],type="l",col="green")
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:5000,rev(1:5000)),c(s2$t3m1e.min[1:5000],
        rev(s2$t3m1e.max[1:5000])),col="grey",border=F)
points(s2$t3m1e.med[1:5000],type="l",col="black")
points(t3m1[1:5000],type="l",col="red")
polygon(c(1:5000,rev(1:5000)),c(s2$t1e.min[1:5000],
        rev(s2$t1e.max[1:5000])),col="lightsteelblue1",border=F)
points(s2$t1e.med[1:5000],type="l",col="blue")
points(t1[1:5000],type="l",col="green")
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI, Scenario 1",
                   "Empirical TSI, Scenario 2",
                   "Theoretical TSI, Scenario 1",
                   "Theoretical TSI, Scenario 2"),
        col=c("black","blue","red","green"),
        lwd=2,
        ncol=2)
par(oma=rep(0, 4))
dev.off()

# n = 1000
pdf('TSI_n1000.pdf')
par(mar=c(5.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:50,rev(1:50)),c(s3$t3m1e.min[1:50],
        rev(s3$t3m1e.max[1:50])),col="grey",border=F)
points(s3$t3m1e.med[1:50],type="l",col="black")
points(t3m1[1:50],type="l",col="red")
polygon(c(1:50,rev(1:50)),c(s3$t1e.min[1:50],
        rev(s3$t1e.max[1:50])),col="lightsteelblue1",border=F)
points(s3$t1e.med[1:50],type="l",col="blue")
points(t1[1:50],type="l",col="green")
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
```

```
polygon(c(1:5000,rev(1:5000)),c(s3$t3m1e.min[1:5000],
        rev(s3$t3m1e.max[1:5000])),col="grey",border=F)
points(s3$t3m1e.med[1:5000],type="l",col="black")
points(t3m1[1:5000],type="l",col="red")
polygon(c(1:5000,rev(1:5000)),c(s3$t1e.min[1:5000],
        rev(s3$t1e.max[1:5000])),col="lightsteelblue1",border=F)
points(s3$t1e.med[1:5000],type="l",col="blue")
points(t1[1:5000],type="l",col="green")
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI, Scenario 1",
                   "Empirical TSI, Scenario 2",
                   "Theoretical TSI, Scenario 1",
                   "Theoretical TSI, Scenario 2"),
        col=c("black","blue","red","green"),
        lwd=2,
        ncol=2)
par(oma=rep(0, 4))
dev.off()

# n = 5000
pdf('TSI_n5000.pdf')
par(mar=c(5.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:50,rev(1:50)),c(s4$t3m1e.min[1:50],
        rev(s4$t3m1e.max[1:50])),col="grey",border=F)
points(s4$t3m1e.med[1:50],type="l",col="black")
points(t3m1[1:50],type="l",col="red")
polygon(c(1:50,rev(1:50)),c(s4$t1e.min[1:50],
        rev(s4$t1e.max[1:50])),col="lightsteelblue1",border=F)
points(s4$t1e.med[1:50],type="l",col="blue")
points(t1[1:50],type="l",col="green")
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:5000,rev(1:5000)),c(s4$t3m1e.min[1:5000],
        rev(s4$t3m1e.max[1:5000])),col="grey",border=F)
points(s4$t3m1e.med[1:5000],type="l",col="black")
points(t3m1[1:5000],type="l",col="red")
polygon(c(1:5000,rev(1:5000)),c(s4$t1e.min[1:5000],
        rev(s4$t1e.max[1:5000])),col="lightsteelblue1",border=F)
points(s4$t1e.med[1:5000],type="l",col="blue")
points(t1[1:5000],type="l",col="green")
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI, Scenario 1",
                   "Empirical TSI, Scenario 2",
                   "Theoretical TSI, Scenario 1",
                   "Theoretical TSI, Scenario 2"),
        col=c("black","blue","red","green"),
        lwd=2,
        ncol=2)
par(oma=rep(0, 4))
dev.off()


# n = 10000
pdf('TSI_n10000.pdf')
par(mar=c(5.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:50,rev(1:50)),c(s5$t3m1e.min[1:50],
        rev(s5$t3m1e.max[1:50])),col="grey",border=F)
points(s5$t3m1e.med[1:50],type="l",col="black")
points(t3m1[1:50],type="l",col="red")
polygon(c(1:50,rev(1:50)),c(s5$t1e.min[1:50],
        rev(s5$t1e.max[1:50])),col="lightsteelblue1",border=F)
points(s5$t1e.med[1:50],type="l",col="blue")
points(t1[1:50],type="l",col="green")
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,20))
polygon(c(1:5000,rev(1:5000)),c(s5$t3m1e.min[1:5000],
        rev(s5$t3m1e.max[1:5000])),col="grey",border=F)
points(s5$t3m1e.med[1:5000],type="l",col="black")
points(t3m1[1:5000],type="l",col="red")
polygon(c(1:5000,rev(1:5000)),c(s5$t1e.min[1:5000],
        rev(s5$t1e.max[1:5000])),col="lightsteelblue1",border=F)
points(s5$t1e.med[1:5000],type="l",col="blue")
points(t1[1:5000],type="l",col="green")
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI, Scenario 1",
                   "Empirical TSI, Scenario 2",
                   "Theoretical TSI, Scenario 1",
                   "Theoretical TSI, Scenario 2"),
        col=c("black","blue","red","green"),
        lwd=2,
        ncol=2)
par(oma=rep(0, 4))
dev.off()
```

```
## overlay scrambled empirical with scrambled theoretical simulations

# n = 200
pdf('TSI_n200_unsorted.pdf')
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,1))
points(s1$t1se.med[1:50],type="l",col="cadetblue",lwd=2)
points(t1se[1:50],type="l",col="orange",lwd=2)
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,10))
points(s1$t1se.med[1:5000],type="l",col="cadetblue",lwd=2)
points(t1s[1:5000],type="l",col="orange",lwd=2)
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI","Theoretical TSI"),
        col=c("cadetblue","orange"),
        lwd=2, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()

# n = 500
pdf('TSI_n500_unsorted.pdf')
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,1))
points(s2$t1se.med[1:50],type="l",col="cadetblue",lwd=2)
points(t1se[1:50],type="l",col="orange",lwd=2)
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,10))
points(s2$t1se.med[1:5000],type="l",col="cadetblue",lwd=2)
points(t1s[1:5000],type="l",col="orange",lwd=2)
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI","Theoretical TSI"),
        col=c("cadetblue","orange"),
        lwd=2, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()

# n = 1000
pdf('TSI_n1000_unsorted.pdf')
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,1))
```

```
points(s3$t1se.med[1:50],type="l",col="cadetblue",lwd=2)
points(t1se[1:50],type="l",col="orange",lwd=2)
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,10))
points(s3$t1se.med[1:5000],type="l",col="cadetblue",lwd=2)
points(t1s[1:5000],type="l",col="orange",lwd=2)
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI","Theoretical TSI"),
        col=c("cadetblue","orange"),
        lwd=2, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()

# n = 5000
pdf('TSI_n5000_unsorted.pdf')
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,1))
points(s4$t1se.med[1:50],type="l",col="cadetblue",lwd=2)
points(t1se[1:50],type="l",col="orange",lwd=2)
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,10))
points(s4$t1se.med[1:5000],type="l",col="cadetblue",lwd=2)
points(t1s[1:5000],type="l",col="orange",lwd=2)
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI","Theoretical TSI"),
        col=c("cadetblue","orange"),
        lwd=2, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()

# n = 10000
pdf('TSI_n10000_unsorted.pdf')
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,1))
points(s5$t1se.med[1:50],type="l",col="cadetblue",lwd=2)
points(t1se[1:50],type="l",col="orange",lwd=2)
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,10))
points(s5$t1se.med[1:5000],type="l",col="cadetblue",lwd=2)
points(t1s[1:5000],type="l",col="orange",lwd=2)
reset()
legend(x = "bottom",inset = 0,
        legend = c("Empirical TSI","Theoretical TSI"),
        col=c("cadetblue","orange"),
        lwd=2, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()



## TSIe for Scenarios from Chapter 2
col = heat_hcl(4,h=c(0,-100),l=c(75,40),c=c(40,80),power=1)
pdf('TSI_ch1scenarios.pdf')
par(mar=c(4.5,4,1,1), oma=c(2.5,1.75,1.5,1))
plot(c(1:5000),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,15))
points(s1$t3n10e.med[1:5000],type="l",col=col[4],lwd=2)
points(s1$t3n6e.med[1:5000],type="l",col=col[3],lwd=2)
points(s1$t3n2e.med[1:5000],type="l",col=col[2],lwd=2)
points(s1$t1e.med[1:5000],type="l",col=col[1],lwd=2)
reset()
legend(x = "bottom",inset = 0,
        legend = c("Scenario 1",
                   "Scenario 2",
                   "Scenario 3",
                   "Scenario 4"),
        col=c(col[4],col[3],col[2],col[1]),
        lwd=3, horiz = TRUE)
par(oma=rep(0, 4))
dev.off()



## TSIe with correlated predictors
pdf('TSIcorr.pdf')
par(mar=c(5.5,4,1,1), oma=c(2.5,1.75,1.5,1),mfrow=c(1,2))
plot(c(1:50),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,5))
points(s6$tc0e.med[1:50],type="l",col="black")
points(s6$tc25e.med[1:50],type="l",col="pink",lwd=2)
points(s6$tc5e.med[1:50],type="l",col="red",lwd=2)
points(s6$tc75e.med[1:50],type="l",col="darkred",lwd=2)
points(t1[1:50],type="l",col="blue")
plot(c(1:500),type="n",xlab="Predictors",ylab="TSI",ylim=c(0,5))
points(s6$tc0e.med[1:500],type="l",col="black")
points(s6$tc25e.med[1:500],type="l",col="pink",lwd=2)
points(s6$tc5e.med[1:500],type="l",col="red",lwd=2)
points(s6$tc75e.med[1:500],type="l",col="darkred",lwd=2)
points(t1[1:500],type="l",col="blue",)
```

```
reset()
legend(x = "bottom",inset = 0,
        legend = c(expression(paste(rho,"=0")),
               expression(paste(rho,"=0.25")),
               expression(paste(rho,"=0.50")),
               expression(paste(rho,"=0.75")),
               "theoretical TSI" ),
        col=c("black","pink","red","darkred","blue"),
        lwd=c(1,2,2,2,1), ncol=3)
par(oma=rep(0, 4))
dev.off()
######################################################################
```

A.2 Motivation for setting signal magnitude equal to $\frac{1}{\sqrt{m}}$

In one dimension, the distance $d(\cdot)$ between two distributions centered at 0 and c is

$$d(\mu_1, \mu_2) = \sqrt{(0-\mu)^2} = c$$

Assume we wish to fix the Euclidean distance between classes to c for p dimensions as in the setting of this thesis. Define $\boldsymbol{\mu_1}$ and $\boldsymbol{\mu_2}$ for the two distributions as $\boldsymbol{\mu_1} = \mathbf{0}$ and $\boldsymbol{\mu_2}$ sparse with m equal, nonzero elements and remaining entries equal to zero. We have

$$\begin{aligned}
\boldsymbol{\mu_1} &= (0_1, 0_2, \ldots, 0_{p-1}, 0_p) \\
\boldsymbol{\mu_2} &= (\mu_1, \mu_2, \ldots, \mu_m, 0_{m+1}, \ldots, 0_{p-1}, 0_p)
\end{aligned}$$

Now we solve for μ_i for $i \in \{1, 2, \ldots, m\}$ with the distance function equal to c:

$$\begin{aligned}
d(\boldsymbol{\mu_1}, \boldsymbol{\mu_2}) &= c \\
\sqrt{\sum_{i=1}^{p} (\mu_{1i} - \mu_{2i})^2} &= c \\
\sqrt{\mu_1^2 + \mu_2^2 + \ldots + \mu_m^2} &= c \\
\sqrt{m\mu_1^2} &= c \\
m\mu_1^2 &= c^2 \\
\mu_1^2 &= \frac{c^2}{m} \\
\mu_1 &= \frac{c}{\sqrt{m}}
\end{aligned}$$

When c is set to one, the value of the nonzero elements in $\boldsymbol{\mu_2}$ equals $\frac{1}{\sqrt{m}}$.