Penalized Regression in the Age of Big Data

Gabriel Ackall[1]*, Connor Shrader[2]*
Mentor: Dr. Ty Kim[3]

[1]Georgia Tech, Civil and Environmental Engineering

[2]University of Central Florida, Mathematics

[3]NCA&T University, Mathematics and Statistics

*Authors contributed equally

July 21, 2021

Abstract

With the prevalence of big data in the modern age, the importance of modeling high dimensional data and selecting influential features has increased greatly. High dimensional data is common in many fields such as genome decoding, rare disease identification, economic modeling, and environmental modeling. However, most traditional regression machine learning models are not designed to handle high dimensional data or conduct variable selection. In this paper, we investigate the use of penalized regression methods instead of, or in conjunction with, the traditional machine learning methods. We focus on lasso, ridge, elastic net, SCAD, MCP, and adaptive versions of lasso, ridge, and elastic net models. For traditional machine learning models, we focus on random forest models, gradient boosting models in the form of XGBoost, and support vector machines. We evaluate these models using factorial design methods for Monte Carlo simulations under various data environments. We conduct tests for 270 environments, with factors being the number of predictors, number of samples, signal to noise ratio, covariance matrix, and correlation strength. This process serves to identify the strengths and weaknesses of different penalization techniques in different environments. We also compare different models using empirical datasets to test their viability in real-world scenarios. Since our models are regression models, we evaluate the models using the test mean squared error and variable selection accuracy. From our investigation, our findings indicate that penalized regression models outperform more traditional machine learning algorithms in most high-dimensional situations or in situations with a low number of data observations. Machine learning models are not often compared to penalized regression methods and so our analysis helps to expand the scope of how penalized regression is used to help model data. Additionally, the analysis helps to create a greater understanding of the strengths and weaknesses of each model type and provide a reference for other researchers on which machine learning techniques they should use, depending on a range of factors and data environments.

Keywords: penalized regression, variable selection, classification, machine learning, large p little n problem, Monte Carlo simulations

1 Introduction

In the modern world, machine learning techniques such as random forest, gradient boosting, and support vector machines are often touted as one-size-fits-all solutions when it comes to modeling big data. While this is frequently the case, an increasingly common type of data set where there are more predictors than observations can pose challenges for these machine learning algorithms. In these situations, lesser known statistical modeling techniques that perform variable selection can potentially perform equivalently or even better than these machine learning techniques. However, there is a distinct lack of academia focusing on comparing these variable selection techniques with the more traditional machine learning techniques. This paper serves to help bridge that gap.

In these situations where there are more predictors, p, than observations, n, many traditional machine learning techniques fail to give good predictions. The large number of predictors and small number of observations make it easy for such models to **overfit**, meaning that the models become fine tuned to the exact training data and instead of finding generalized patterns for a population of data, they find specific occurrences in the training data. Because of this, overfitted models are sensitive to new data which causes them to perform extremely well on the training data, but poorly on testing data or when deployed in the real world. Because a model's predictions in real world scenarios and on new data is the entire purpose of a model, it is very important to reduce overfitting so that predictive accuracy in these scenarios is maximized.

[Citations below]

This paper investigates various methods used to handle the large p, small n problem. We considered subset selection methods such as forward selection, backward selection, stepwise forward selection and stepwise backward selection using both Akaike information criterion (AIC) and Bayesian information criterion (BIC) as the stopping criteria for the models. In addition, we studied penalized regression models such as ridge regression [17], LASSO, elastic-net, SCAD, and MCP. These models were compared to random forest, gradient boosting in the form of XGBoost, and support vector machine models. In order to compare these models, models were trained and evaluated using both generated Monte Carlo simulations data and empirical genomic data.

1.1 Modeling Background

Suppose that we have p predictor variables $X_1, X_2, \ldots, X_p$ and one response variable Y that depends on some (or all) of the predictors. We assume that Y can be expressed as

$$Y = f(X_1, X_2, \ldots, X_p) + \epsilon \tag{1}$$

where f is a function and ϵ is an independent random error with mean zero. The goal of supervised modeling is to find a function $\hat{f}$ that is a suitable approximation for f. To find $\hat{f}$, we use a **training set**, a set of observations where the response variable Y is already known. Then, using the fitted model, we can predict the value of the response variable $\hat{Y}$ for new observations, even if Y is unknown. Model performance can be evaluated using a **test set**, which is a set of observations that were not used to train the model.

There are two broad types of supervised models. **Regression modeling** is used when the response variable Y takes numerical values on a continuous interval. For example, a model that predicts the value of a home is a regression model. On the other hand, if Y can only take discrete values, then **classification modeling** is used. For instance, a model used to predict whether or not a patient has a disease is classification problem. This paper focuses on regression modeling.

1.2 Linear Regression and Ordinary Least Squares

In practice, the function f that relates the predictors to the response is complex. Most statistical models assume that f takes some particular form and estimates a function $\hat{f}$ of that form. For example, many regression models assume that f is a linear function of the predictors; that is, linear models assume that

$$f(X_1, X_2, \ldots, X_p) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p \tag{2}$$

where $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ are coefficients. Notice that the coefficient β_0 is not multiplied with any predictor; it represents an intercept value. Fitting a linear model will give estimates for these coefficient values.

The most common method to approximate the coefficients in a linear model is by **ordinary least squares**. Suppose that we have n observations in our training set. Let x_{ij} represent the value of predictor j for observation i, and let y_i be the response for observation i. For some coefficient estimates $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$, the expression

$$y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \tag{3}$$

is called the **residual** for observation i; it is the difference between the true response value and the predicted response variable using the given coefficient values. Ordinary least squares chooses the coefficients $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ that minimize the **residual sum of squares**

$$\text{RSS} = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}))^2 \tag{4}$$

Intuitively, if the residual sum of squares is low, then the differences between the response variable and its estimates is low. Thus, by minimizing the residual sum of squares, the function obtained from ordinary least squares is a relatively good approximation for f. Figure 1 demonstrates a model fitted with ordinary least squares when there is a single predictor variable.

One reason that ordinary least squares is popular is because it is very easy to compute. Let $\beta = [\beta_0, \beta_1, \beta_2, \ldots, \beta_p]^\top$ be a $(p+1) \times 1$ vector of coefficient values and let $\mathbf{X}$ be a $n \times (p+1)$ matrix where each row contains the predictor values for one observation, with an additional value of 1 in the first entry (this extra value corresponds to the intercept coefficient β_0, which is not truly a predictor). Then $\mathbf{X}\beta$ is a vector of the estimated response values for our choice of β. Let $\mathbf{y}$ represent the true response values. Then $\mathbf{y} - \mathbf{X}\beta$ is a vector of residuals. To choose coefficient estimates that minimize the residual sum of squares, we compute

$$\hat{\beta}^{\text{OLS}} = \arg\min_{\beta} \left\{ (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta) \right\} \tag{5}$$

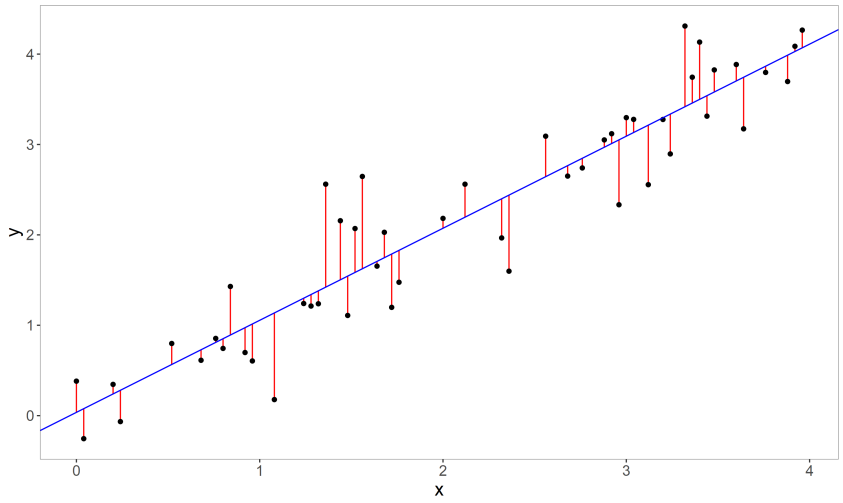


Figure 1: Ordinary least squares fitting with one predictor using simulated data. The blue line represents the line found by ordinary least squares, and the red line segments are the residuals.

where $(\mathbf{y} - \mathbf{X}\beta)^\top(\mathbf{y} - \mathbf{X}\beta)$ is the same residual sum of squares seen in Equation 4. From [15], this gives us the solution

$$\hat{\beta}^{\text{OLS}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} \tag{6}$$

Another advantage of ordinary least squares is that it is an unbiased linear model. This means that if the relationship between the response variable and the predictors is actually linear (as given in Equation 2), then the expected value of the coefficient vector $\hat{\beta}^{\text{OLS}}$ is equal to the actual coefficient vector β. Furthermore, the **Gauss-Markov theorem** states that if the random error ϵ is independent and has constant variance, then ordinary least squares has the lowest variance among all linear unbiased estimators. In other words, the coefficient estimates given by ordinary least squares are relatively close to the actual coefficient values when compared to other unbiased linear estimators. This makes ordinary least squares a relatively consistent model.

If ordinary least squares is unbiased and has the lowest variance among all unbiased models, why should we use any other type of linear model? Despite having a lower variance than other unbiased models, ordinary least squares can still have a high variance. This is especially an issue when the number of predictors p is large compared to the number of observations n. As p gets closer to n, a model fitted with ordinary least squares will typically **overfit** to the training set. This means that the fitted model makes very good predictions with the training data, but performs poorly when given test data that wasn't used to fit the model. Overfitting occurs because of the random error from Equation 1. Ordinary least squares is unable to distinguish between signal and noise, so it will tend to assume predictors are more strongly related to the response than they actually are.

In the extreme case where p exceeds n, the matrix $\mathbf{X}^\top\mathbf{X}$ from Equation 6 becomes non-invertible. This means that there are many coefficient estimates that minimize the residual

sum of squares. In fact, any of these coefficient estimates creates a perfect fit to the training data, which will result in very bad predictions with test data.

By using models that have a small amount of bias, the high variance of ordinary least squares can be mitigated. Liu et. al. in [20] describe three types of variable selection algorithms. **Filter methods** work by evaluating the ability for each individual predictors to predict the response; then, a model is fit using the predictors selected [23, 9]. **Wrapper methods** fit models using different subsets of predictors and choose the model that has the best performance. Finally, **embedded methods** perform variable selection during the model training process. This paper focuses on wrapper methods and embedded methods. In addition, we also used several non-linear machine learning methods to draw a comparison between linear regression models and non-linear models.

1.3 Subset Selection Methods

Subset selection methods are wrapper methods that attempt to find a subset of the predictors $X_1, X_2, \ldots, X_p$ that are most correlated with the response variable Y. These algorithms usually fit models for many different subsets and choose the subset of predictors that results in the best model. Although subset selection techniques can be applied to many types of models, we will focus on subset selection with linear regression.

There are two main benefits to using subset selection methods. By reducing the set of available predictors to just those that are strongly related to the response, overfitting can be mitigated by ignoring predictors that provide little improvement to model performance. Another benefit of subset selection is that it creates a more interpretable model. If a data set includes thousands of predictors but only a few are related to the response, a model found using subset selection will be easier to understand than a model that relies on all of the parameters.

Best subset selection is a subset selection method that considers every possible combination of predictors. For every possible value of k between 0 and p, best subset selection will fit the $\binom{p}{k}$ possible models using k predictors. Then, the best model for each value of k is chosen based on some performance metric. Finally, a final model can be selected from the $(p+1)$ remaining models, ranging from an empty model with no predictors to a full model with all of the predictors. If p is larger than n, then models are only fit for subsets with n or less predictors since ordinary least squares cannot be used when there are more predictors than observations.

Although best subset selection is guaranteed to find the subset of predictors that optimize the chosen metric, this method is computationally expensive. For a data set with p predictors, 2^p possible combinations must be considered. This makes best subset selection infeasible when the number of predictors is too large.

Two alternative methods to best subset selection are **forward selection** and **backward selection**. Forward selection begins by fitting a model with no predictors (only the intercept is non-zero) and iteratively adds predictors into the model. The predictor added at each step is chosen to best increase the model fit. Conversely, backward selection starts from the full (ordinary least squares) model with all p predictors and repeatedly removes predictors. Then, like best subset selection, the final model is chosen from the candidate models fitted at each step. Note that backward selection can only be used when $p \leq n$ since ordinary

least squares cannot be used when $p > n$. Forward selection can always be used.

Although forward and backward selection will not always encounter the best possible model, these methods avoid the exponential runtime of best subset selection. Consequently, forward and backward selection can be used for larger values of p.

The models produced by forward and backward selection can be improved by allowing predictors to be added and removed in the same algorithm. **Forward stepwise selection** begins with an empty model and iteratively improves the model by either adding a new predictor or removing an obsolete one. **Backward stepwise selection** works in the same way but starts with the full model. Like backward selection, backward stepwise selection can only be used when $p \geq n$. These techniques take longer to run than ordinary forward and backward selection, but they are more likely to find the best possible model.

When fitting a model using any of the subset selection methods, the performance metric used when selecting the best model is very important. At first, it may seem reasonable to choose a metric such as the residual sum of squares from Equation 4. However, many metrics, including the residual sum of squares, only describe a model's performance on training data. This is problematic because including more predictors will always decrease the residual sum of squares on the training data. If $p \leq n$, then the model fitted with all p predictors is exactly the same model produced by ordinary least squares, which by definition minimizes the residual sum of squares! If $p > n$, then a model with the maximum possible number of predictors would be selected.

If we wish to produce a model that makes reliable predictions on test data, we must use a different performance metric. Two of the most common metrics used for this purpose are the **Akaike Information Criterion** (AIC) and the **Bayesian Information Criterion** (BIC) [1, 25]. These metrics can be expressed in terms of the log-likelihood function. In the special cases where we have a linear model where the random error ϵ is Gaussian, the Akaike Information Criterion can instead be expressed as

$$\text{AIC} = \frac{1}{n\hat{\sigma}^2}(\text{RSS} + 2p\hat{\sigma}^2) \tag{7}$$

up to a constant, where $\hat{\sigma}^2$ is the estimated value of the variance of ϵ and RSS is the residual sum of squares from Equation 4 [18]. The Bayesian Information Criterion is

$$\text{BIC} = \frac{1}{n\hat{\sigma}^2}(\text{RSS} + \ln(n)p\hat{\sigma}^2) \tag{8}$$

$$\text{BIC} = -2l + \log(n)p \tag{9}$$

up to a constant. These metrics work by using the residual sum of squares plus some additional penalty that increases when p is large. As a result, models that minimize AIC or BIC will have fewer predictors than models chosen just by minimizing the residual sum of squares. This can result in a model that has both a good training error and test error. If $n > 7$, then $\ln(n) > 2$ and so the penalty for BIC is larger than the penalty for AIC. Hence, a model selected using BIC will typically have fewer parameters than a model selected by AIC.

In addition to AIC and BIC, there are several other metrics that modify training error to estimate test error, such as C_p and adjusted R^2 [18]. However, this paper will focus on AIC and BIC.

1.4 Penalized Regression

In general, **penalized regression** works by fitting a model that punishes large coefficient estimates. By forcing coefficient values to shrink, the resulting model will have relatively low variance. Most, but not all, of these methods can perform variable selection. For the remainder of this section, let β represent a single arbitrary coefficient value, rather than a vector of coefficient values.

All of the penalized regression methods in this paper solve an optimization problem of the form

$$\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})) + \lambda \sum_{j=1}^{p} P(\beta_j) \right\} \tag{10}$$

where the first summation is the usual residual sum of squares, $\lambda \geq 0$ is a hyperparameter that controls the strength of the penalty, and $P(\beta)$ is a penalty function applied to each of the coefficients (but not the intercept term). In general, $P(\beta)$ is an even function that is non-decreasing as $|\beta|$ increases. If $\lambda = 0$, then we have the usual ordinary least squares estimator. As λ increases, a stronger penalty is applied which will decrease the coefficient values. As λ approaches ∞, the model becomes an empty model where only the intercept term is non-zero.

A similar way to express penalized regression is with the form

$$\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})) \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} P(\beta_j) \leq t \tag{11}$$

where t is a hyperparameter. In this form, penalized regression fits the ordinary least squares such that the sum of the penalties of the coefficients are kept below some threshold. It can be shown that for every value of λ from Equation 10, there is a value of t from 11 that gives equivalent coefficient estimates [18]. Most algorithms to fit penalized regression models use the form given in Equation 10, but the form in Equation 11 may be considered more intuitive.

When fitting penalized regression models, the choice of λ is very important. If λ is too small, then models may be overfit just like ordinary least squares; on the other hand, if λ is large, then the resulting models are highly biased and the resulting models may be underfit. In practice, the best value of λ can be selected using **cross-validation**. This process involves splitting the training set into k disjoint subsets of equal size called **folds**. Then, k models are fitted for different values of λ, where one fold is used as a testing set and the other $(k-1)$ folds are used for training. The value of λ that results in the lowest test error can then be selected.

Ridge regression is a penalized regression model that uses the penalty function $P(\beta) = \beta^2$ [17]. One advantage of ridge regression is that it can handle highly correlated data better than ordinary least squares. When predictors are correlated with each other, some algorithms have a hard time distinguishing which predictors are actually related to the response. One other benefit of ridge regression is that it can be used when $p > n$.

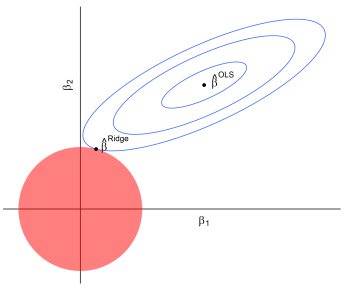
One drawback of ridge regression is that it cannot perform variable selection. It can shrink coefficients towards zero, but it cannot set coefficients to be exactly zero.

The **least absolute shrinkage and selection operation**, commonly referred to as **lasso**, is a shrinkage method with a very similar form to ridge regression [26, 18]. The penalty function for lasso is $P(\beta) = |\beta|$.

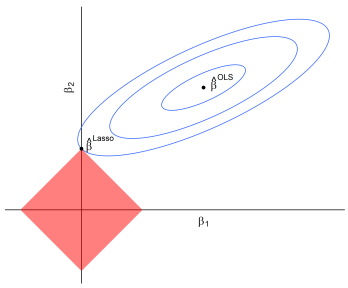
Unlike ridge regression, lasso regression can perform variable selection by setting coefficients to zero. This makes lasso regression favorable when most predictors are not related to the response variable. On the other hand, if the response truly does depend on all of the predictors, then lasso regression may incorrectly set some coefficients to zero. In addition, lasso regression does not have a closed-form solution, but computing the coefficients is still efficient.

Figure 2 provides a visual explanation for why ridge regression cannot perform variable selection while lasso regression can. Suppose that there are $p = 2$ predictors. The red circle in Figure 2a represents the circle $\beta_1^2 + \beta_2^2 \leq 1$, which is the penalty boundary for ridge regression in the form given in Equation 11 when $t = 1$. The red square in Figure 2b represents the boundary $|\beta_1| + |\beta_2| \leq 1$ for lasso regression. The point in the center of the blue ellipses represents the values of β_1 and β_2 that ordinary least squares would produce. Because this point is not within either of the red regions, ridge regression and lasso regression will give different coefficient estimates. The blue ellipses around this point represent contour curves for the residual sum of squares function, which is a quadratic function of β_1 and β_2. The first interception of these ellipses with the red regions represents the coefficient values selected by ridge and lasso regression, since it represents the point within the red region with the lowest residual sum of squares. Because of the round boundary for ridge regression, the intersection in Figure 2a occurs when β_1 and β_2 are both positive. For lasso regression in Figure 2b, the intersection occurs at a corner where $\beta_1 = 0$. Thus, lasso regression has set β_1 to zero, while β_1 is non-zero for ridge regression.

Elastic-net regression uses both the ridge penalty and the lasso penalty at the same



(a) RSS contours and the ridge penalty boundary.



(b) RSS contours and the lasso penalty boundary.

Figure 2: RSS contours and penalty bounds for the ridge and lasso models when $p = 2$ and $t = 1$. The red regions represent the coefficient values allowed by ridge and lasso regression, respectively. The blue ellipses represent contours of the residual sum of squares, with $\hat{\beta}^{\text{OLS}}$ being the point where the residual sum of squares is minimized. The intersection of the ellipse with the red region represents the point selected by lasso and ridge.

time [31]. Elastic-net solves the optimization problem

$$\hat{\beta}^{\text{E-net}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right) + \lambda_2 \sum_{j=1}^{p} \beta_j^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| \right\} \tag{12}$$

where λ_1 and λ_2 are both tuning parameters to be determined later.

An important limitation to note is that elastic net performs best when it close to either ridge or lasso regression, meaning that either λ_1 greatly exceeds λ_2 or λ_2 greatly exceeds λ_1 [31]. Additionally, because elastic net requires two tuning parameters, this makes it much more difficult to determine the best combination of tuning parameters to minimize error in the regression. However, this problem has been largely solved through by the LARS-EN algorithm developed by Zou et. al. which efficiently solves for the tuning parameters [31].

One major flaw of the lasso method is that the penalty punishes large coefficients, even if those coefficients should be large. One way to modify the lasso method is to use the **smoothly clipped absolute deviation** (SCAD) penalty [13]. The goal of this method is to punish large coefficients less severely, which can help mitigate some of the bias introduced by the lasso method. The penalty function $P(\beta)$ for SCAD satisfies

$$\frac{dP}{d\beta} = \text{sign}(\beta) \left[I(|\beta| < \lambda) + \frac{\max(a\lambda - |\beta|, 0)}{(a-1)\lambda} I(|\beta| > \lambda) \right] \tag{13}$$

where $a \geq 2$ is a new hyperparameter and I is the indicator function ($I(Q)$ equals 1 if a statement Q is true, and equals 0 if Q is false). If the hyperparameter a is large, then SCAD behaves like lasso for larger coefficient values. Also, notice that λ is an argument for the penalty function P.

An equivalent way to write the expression in Equation 13 is

$$\frac{dP}{d\beta} = \begin{cases} 1, & |\beta| \leq \lambda \\ \frac{a\lambda - |\beta|}{(a-1)\lambda}, & \lambda < |\beta| < a\lambda \\ 0, & \alpha\lambda < |\beta| \end{cases} \tag{14}$$

This penalty function does not punish coefficients with large magnitude as heavily as the lasso method. In fact, if the magnitude of a coefficient is larger than $a\lambda$, then the penalty becomes constant since the derivative becomes zero.

By integrating with respect to β [2] and choosing $P(\beta) = 0$, we see that

$$P(\beta) = \begin{cases} |\beta|, & |\beta| \leq \lambda \\ \frac{2a\lambda|\beta| - \beta^2 - \lambda^2}{2(a-1)\lambda}, & \lambda < |\beta| \leq a\lambda \\ \frac{\lambda(a+1)}{2}, & a\lambda < |\beta| \end{cases} \tag{15}$$

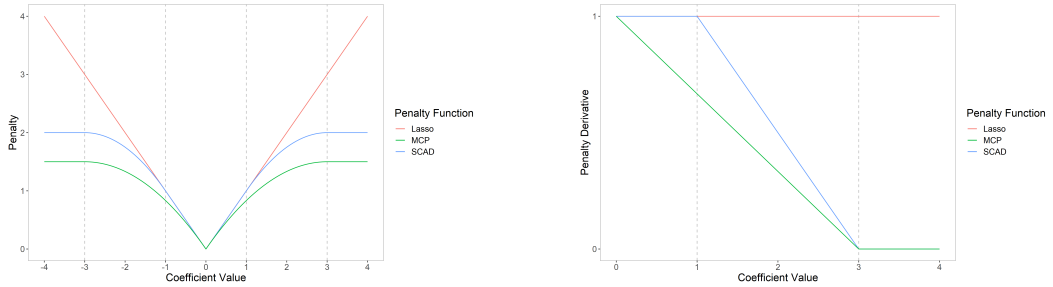
The **minimax concave penalty** (MCP) method is very similar to SCAD [29]. Both methods are used to avoid the high bias caused by the lasso method. MCP uses a penalty function that satisfies

$$\frac{dP}{d\beta} = \begin{cases} \text{sign}(\beta) \left(1 - \frac{|\beta|}{a\lambda} \right), & |\beta| \leq a\lambda \\ 0, & a\lambda < |\beta| \end{cases} \tag{16}$$

where, like SCAD, $a > 1$ is a hyperparameter. Integrating [2], we see that

$$P(\beta) = \begin{cases} |\beta| - \frac{\beta^2}{2a\lambda}, & |\beta| \leq a\lambda \\ \frac{1}{2}a\lambda, & a\lambda < |\beta| \end{cases} \tag{17}$$

Figure 3 below shows the penalty functions (and their derivatives) for lasso, SCAD, and MCP as a function of a coefficient value β. We see that lasso applies a much stronger penalty to large coefficients than SCAD or MCP. Also, note that SCAD starts with a derivative equal to that of the lasso for small values of β; on the other hand, the derivative of the penalty function for MCP starts decreasing immediately.



(a) Penalty functions for lasso, SCAD, and MCP.

(b) Derivatives of the penalty functions for lasso, SCAD, and MCP.

Figure 3: Penalty functions for lasso, SCAD, and MCP, as well as their derivatives. These plots use $\lambda = 1$ and $a = 3$. The dashed vertical lines are the knots for SCAD and MCP.

Note that of the penalized regression methods discussed above, only ridge regression has a general closed-form solution. Although lasso, elastic-net, SCAD and MCP do not have closed-form solutions, their solutions can be efficiently approximated. In some special cases, however, a closed-form solution exists.

Let $\mathbf{X}$ be a $n \times p$ matrix where each row contains the predictor values for one observation. To simplify things, we will assume that our data is centralized so that the coefficient β_0 is 0; that way, we do not need to include an extra entry of 1 in each row of $\mathbf{X}$. In an **orthonormal design**, we assume that $\mathbf{X}$ is orthonormal, meaning that the magnitude of each column of $\mathbf{X}$ is one and every pair of columns of $\mathbf{X}$ is orthogonal. In this special case, the matrix $(\mathbf{X}^\top \mathbf{X})^{-1}$ is the $p \times p$ identity matrix. As a result, each coefficient is independent of the other coefficients; in other words, we can compute the coefficient estimate for each predictor without needing to use the values for the other predictors. For example, in an orthonormal design, the general solution to ordinary least squares regression given in Equation 6 simplifies to

$$\hat{\beta}^{\text{OLS}} = \mathbf{X}^\top \mathbf{y} \tag{18}$$

To compute the coefficient estimates for lasso, SCAD, and MCP in an orthonormal design, we first compute the vector $\mathbf{z} = \mathbf{X}^\top \mathbf{y}$ [13]. Each entry of $\mathbf{z}$ corresponds to one predictor. We then apply a **thresholding function** to each entry of $\mathbf{z}$ to get the coefficient estimate for that predictor. For lasso, elastic-net, SCAD, and MCP, a closed-form for this thresholding function exists [26, 13, 31, 29]. The input to the thresholding function is the actual coefficient value, and the output is the estimated value for that coefficient in an

orthonormal design. Figure 4 shows the threshold functions for lasso, SCAD and MCP when $\lambda = 1$. For SCAD and MCP, we used the value $a = 3$. For reference, the identity line is included, which can be considered as the threshold function for ordinary least squares.

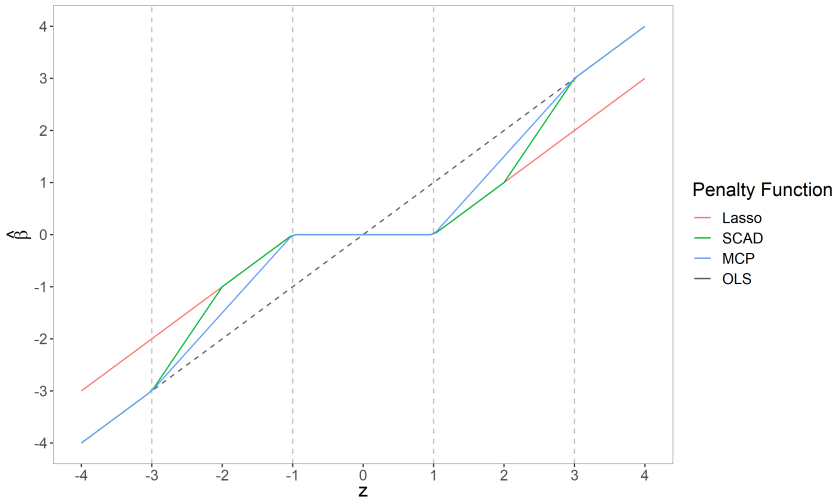


Figure 4: Thresholding function for lasso, SCAD, and MCP when $\lambda = 1$ and $a = 3$ in an orthonormal design. The input (z) represents an entry from the vector $\mathbf{z} = \mathbf{X}^\top \mathbf{y}$; the output ($\hat{\beta}$) is the coefficient estimate. The dashed vertical lines are the knots for SCAD and MCP.

We see that when $|\beta|$ is small, the thresholding function for lasso, SCAD, and MCP are all zero. This matches with the assumption that these methods should set small coefficient values to zero. As $|\beta|$ increases, the estimated value for that coefficient gets larger. Notice that ordinary least squares doesn't decrease the estimated coefficient value for any choice of β; this is because of the fact that ordinary least squares is unbiased, whereas the other methods are biased.

In this figure, we also see that the threshold function for lasso is parallel to the identity line for large values of β. This means that even if a predictor is very influential on the response, lasso will predict a coefficient estimate that is less than the true coefficient value. This makes lasso a very biased model. This issue was one of the main motivations for SCAD and MCP. Unlike lasso, SCAD and MCP converge to the identity line as β increases, which makes these methods less likely to decrease large coefficient estimates.

Another feature of SCAD and MCP is their oracle-like properties [13, 29]. The term **oracle** in this context was first proposed by Donohe and Johnstone in [10]. Suppose that a linear model could be fitted with the aid of an oracle. This oracle could tell you the subset of predictors that are truly related to the response so that a model can be fitted using only the important predictors. Although such a model is not possible in practice, this oracle procedure serves an ideal that can be worked towards.

A linear model is said to have **oracle-like properties** if two asymptotic conditions hold [30]. Let p^* be the number of predictors that have non-zero coefficients. First, as $n \to \infty$, the model must be able to correctly identify the correct subset of non-zero predictors, which

we will denote by $\mathcal{A}$. The second condition is that as $n \to \infty$, we must have

$$\sqrt{n}(\hat{\beta}_{\mathcal{A}} - \beta_{\mathcal{A}}) \to_d \mathcal{N}_p^*(\mathbf{0}, \mathbf{\Sigma}^*) \tag{19}$$

where $\hat{\beta}_{\mathcal{A}}$ are the coefficient estimates for the non-zero predictors, $\beta_{\mathcal{A}}$ are the true coefficient values for the non-zero predictors, and $\mathcal{N}_p^*(\mathbf{0}, \mathbf{\Sigma}^*)$ is the multivariate normal distribution of the p^* non-zero predictors with mean zero and covariance matrix $\mathbf{\Sigma}^*$. In other words, as $n \to \infty$, the coefficient estimates for the non-zero predictors is normally distributed around the true coefficient values. Importantly, this means that the expected value for the coefficient estimates are the true coefficient values.

Both SCAD and MCP have oracle-like properties under certain conditions [13, 29]. Hence, SCAD and MCP can perform as well as the oracle estimator as n approaches ∞. On the other hand, lasso does not have oracle-like properties, meaning that its predictions cannot perform as well as a model found with the aid of an oracle.

1.5 Non-linear models

We next discuss several non-linear methods for regression: random forests, gradient boosting, and support vector machines.

Both random forest and gradient boosting models use **decision trees** to make predictions. A decision tree is a binary tree where each non-leaf node represents a condition and each leaf node represents a prediction value. To make a prediction, start at the root node and check whether the condition at that node is true or false. If true, move down to the node's first child; if false, move to the second child. This process is repeated until a leaf node is reached. This node will give a value that the decision tree predicts.

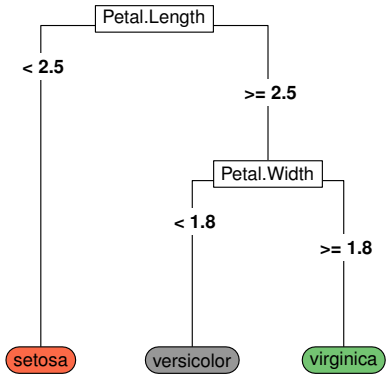


Figure 5: Example of a decision tree. This tree was generated using the `iris` data set and attempts to predict the species of a flower based on its petal length and petal width. The decision tree was fitted and plotted with the `rpart` and `rpart.plot` libraries in `R`.

Although decision trees can be used as machine learning models on their own, it is more common to use decision trees in **ensemble methods**, which combine many different

decision trees into a single model. A single decision tree will usually have high variance since a small change in the training set can lead to a completely different decision tree [18].

Random Forests (RF) solve the issue of high variance by aggregating the predictions of many independent decision trees [5]. Each tree is fit using a subset of the observations and a subset of the predictors, so that the trees are relatively independent from one another.

To fit each decision tree within a random forest model, first select a random sample of the n observations with replacement, in a process called **bootstrapping** [12]. The number of observations chosen for each tree is a hyperparameter that can be changed. After selecting a set of observations, a random sample of predictors are chosen out of the p predictors without replacement. Again, this helps decorrelate the trees. The number of predictors used in each tree is also a hyperparameter that can be changed. Then, a decision tree is generated using the available observations and predictors.

A random forest model aggregates all of the individual decision trees into a single model. The number of trees generated is a hyperparameter that can be changed; usually, a random forest model will contain at least several hundred trees. To make predictions with a random forest, a test observation is passed into each decision tree and the predictions from each tree are aggregated. For regression, the results are normally aggregated using the mean; for classification, the prediction chosen most often by the trees is usually used as the final prediction.

This idea of fitting multiple models with bootstrapping and aggregating the results can be used with other models besides decision trees. In general, this process is called **bagging** (a combination of the words "bootstrap" and "aggregating") [4].

Boosting is the technique of sequentially improving a weak learner until it becomes a strong learner [24]. A **gradient boosting machine** (GBM) is a boosting technique that uses gradient descent to minimize error in a model and correct the shortcomings of the previous weak learner model [16]. This is done through fitting a model, whether that is a linear regression or decision tree model, to a set of data points. From there, the residual of each data point is calculated and another linear regression or decision tree model is fitted to those residuals. A new model is fitted to the residual data of the residual data fitted model, and so on. These models are then all added together to result in a strong GBM model.

Gradient boosting can be used for both regression and classification if they use decision trees, or if they use linear regression models, they can only perform regression. When using a gradient boosting model with decision trees, relative variable importance and pruning can be used as a sort of pseudo-variable selection method to lower complexity and prevent over-fitting. However, this does not yield the same interpretability or bias trade off benefits that true variable selection yields. When using a gradient boosting model with linear regression, there is the ability to use lasso, ridge, and elastic net penalized regression as the weak learner and perform variable selection, although this is much less commonly used than decision trees in XGBoost.

Gradient boosting models often suffer from slow computation speeds due to the large number of sequential models that need to be trained. **Extreme Gradient Boosting** (XGBoost) is a faster version of gradient boosting that utilizes parallel computing as well as different optimization techniques to speed up computation [6]. For these reasons, XGBoost is often preferred over standard gradient boosting models and is very commonly used in

many machine learning applications. In this study, we will use the XGBoost algorithm for gradient boosting.

For our study, we considered a few different hyperparameters. The learning rate controls how quickly the gradient boosting model learns. If this learning rate is high, then the model learns quickly, but it may not learn as efficiently. If the learning rate is low, then the model takes longer but typically makes better predictions. The maximum tree depth determines how high each tree can be. Using smaller tree sizes may oversimplify the model, but it could also mitigate overfitting.

The final non-linear model that we considered is the **Support vector machine** [8]. In the ideal case, a support vector machine is a binary classier whose decision boundary is a $(p-1)$-dimensional hyperplane in p-dimensional space that perfectly separates all of the observations for one class on one side and the observations for the other class lie on the opposite side. Moreover, the hyperplane chosen by a support vector machine will maximize the distance between this hyperplane and any of the observation points.

However, most data sets cannot be split perfectly into two sides. Furthermore, this model has very high variance as changing the points near the hyperplane can significantly alter the hyperplane. Finally, this model cannot handle cases where the true boundary is non-linear. Luckily, support vector machines in practice can handle such issues. Typically, support vector machines are allowed to misclassify some of the training data, which address the cases where the data cannot be split perfectly by a hyperplane. Also, the predictor space can be enlarged to handle non-linear decision boundaries; this is typically done by using **kernels**. For example, using a radial kernel can create decision boundaries that enclose regions of the p-dimensional space.

Although support vector machines are usually used for classification, they can be generalized to perform regression as well [11]. We will be using support vector machines for regression in our study.

Like the other non-linear methods, there are many hyperparameters that can be tuned. The two hyperparameters that we considered are ϵ and the cost function. ϵ defines how tolerant the algorithm is of small errors. If ϵ is large, then the support vector machine will tolerate larger errors; if ϵ is small, then even small errors will be punished. The cost hyperparameter C determines how strongly the model punishes incorrect predictions. If C is large, then the model punishes incorrect predictions more, making it fit more tightly to the training set. If C is smaller, then the model is likely to allow more incorrect predictions in the training data.

2 Methods

2.1 Models

This section gives the specific details how we fit each model for either the simulated data or the empirical data. For a conceptual overview of each of the models, refer back to the previous section.

All of our simulations were run using version 4.1.0 of `R`. Several different libraries were

used to fit machine learning models using our simulated data. Table 1 summarizes the libraries used to fit models.

Table 1: `R` Libraries used and the models used from each library

Library	Models used	Version
`stats` [22]	Ordinary least squares	4.1.0
`MASS` [27]	Forward and backward selection	7.3-54
`glmnet` [14]	Ridge, lasso, elastic-net	4.1-1
`ncvreg` [3]	SCAD and MCP	3.13.0
`xgboost` [7]	Gradient boosting	1.4.1.1
`ranger` [28]	Random forest (simulations)	0.12.1
`randomForest` [19]	Random forest (empirical data)	4.6-14
`e1071` [21]	Support vector machine	1.7-7

Ordinary least squares models were fitted using the `lm` function from the `stats` package in base `R`.

Subset selection models using forward selection, backward selection, stepwise forward selection, and stepwise backward selection were fitted using the `MASS` library. For each of these four algorithms, we fit models using both AIC and BIC, giving up to eight models total.

For ridge, lasso, and elastic-net regression using `glmnet`, we used the `cv.glmnet` function. This function uses cross-validation grid search to determine the value of λ that minimizes the cross-validation error. Using cross-validation can help generate a model that performs well on both training and testing data. We used the default value of 10 folds to fit models using `glmnet`. For elastic-net regression, we used the hyperparameter $\alpha = 0.8$ in our simulation study. This means that the elastic-net model is more similar to lasso (where $\alpha = 1$) than ridge (where $\alpha = 0$). This values of $\alpha = 0.8$ was chosen so that the elastic-net model could emphasize the variable selection provided by lasso while also allowing it to handle multicollinearity from ridge. For the empirical data analysis, we used $\alpha = 0.5$. The remaining hyperparameters were given their default values.

For SCAD and MCP models, we used the `cv.ncvreg` function from the `ncvreg` library. Both SCAD and MCP depend on an additional hyperparameter a. We used the default values of a for both models: 3 for MCP and 3.7 for SCAD (note that the `ncvreg` documentation calls this hyperparameter γ instead of a). All other arguments were given their default values.

For the three non-linear models (gradient boosting, random forests, and support vector machines), we used cross-validation and grid search to find suitable hyperparameters, and then fit a model using the full training set using the hyperparameters selected. Because many the data sets used had large values of n and p, only the most important hyperparameters were tuned. This ensured that the models could be fit within a reasonable amount of time. All other hyperparameters were given their default values.

For gradient boosting with `xgboost`, we used different values for the learning rate (0.1, 0.3, and 0.5) and maximum tree depth (1, 3, and 7). A maximum of 1000 trees were generated, with an early stopping condition if the model failed to improve for several iterations in a row. We used five folds in the cross-validation.

For random forests, we used `ranger` for the simulated data and `randomForest` for the empirical data. We did not use `ranger` for the empirical data because it could not handle the large number of predictors, resulting in stack overflow errors. For both `ranger` and `randomForest`, we tuned the number of predictors used per decision tree ($\lfloor\sqrt{p}\rfloor$, $\lfloor p/3\rfloor$, and $\lfloor p/2\rfloor$) and the number of trees (300, 400, 500 and 600). The best model was selected based on the out-of-bag error, which represents the average error for each observation using only the trees that did not include that observation.

Finally, for support vector machines using `e1071`, we varied ϵ (0.1, 0.5, 2), which affects the model's sensitivity to small errors. We also controlled the cost value C (0.5, 1, 2), which affects how much the model punishes wrong predictions.

Some of the models used could only be used for certain values of n and p. This is because either the runtime becomes unreasonable when n or p are large, or the model simply cannot be used when p is too large. Ordinary least squares was used when $p \leq n$, since it cannot be used at all when $p > n$. For the same reason, the forward subset selection algorithms were also used when $p \leq n$. However, the backward subset selection algorithms were only used when $p \leq 40$. When $p > 40$, the runtime was very long, so we chose to discard backward selection when p was too large. Ridge regression was used when $p \leq n$ as well as it is not designed for variable selection or for $p \geq n$ scenarios.

Lasso, SCAD, MCP, GBM, and Random Forest models were used for all data sets. Support vector machine models were made for all of the simulated data but was not used for the empirical data. The empirical data has too many predictors which leads to unmitigable stack overflow errors. Simply put, the support vector machine model has trouble converging upon a local minimum and due to the need to store very large kernel matrices, support vector machines are not feasible for large datasets.

2.2 Monte Carlo Simulations

Monte Carlo simulations use randomly generated data to fit and test our regression and classification models. There are several benefits to using simulated data rather than experimental data. For one, the true relationship between the predictor variables and the response is known. Simulations can also be iterated many times, giving sturdier results about the effectiveness of each model. Finally, Monte Carlo simulations give us full control over how our data is distributed. This enables us to evaluate the models under various conditions.

For the simulated data, we assumed that the relationship between the response variable Y and the predictors $X_1, X_2, \ldots, X_p$ was linear. That is, we assumed that

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon \tag{20}$$

where β_0 is some intercept, $\beta_1, \ldots, \beta_p$ are coefficient values and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is a normally distributed random error with mean 0 and variance σ^2. We chose the values $\beta_0 = 1$, $\beta_1 = 2$, $\beta_2 = -2$, $\beta_5 = 0.5$ and $\beta_6 = 3$, while the remaining coefficient values were set to 0.

To generate the data, we first defined $\beta = [\beta_0, \beta_1, \ldots, \beta_p]^\top$, a $(p+1) \times 1$ vector of coefficient values. Next, we generated $\mathbf{X}$, a $n \times (p+1)$ matrix where each row contains the predictor values for one observation. The column of $\mathbf{X}$ contained all ones, which correspond

to the intercept β_0. The entries for the remaining columns of $\mathbf{X}$ were generated using the p-dimensional multivariate normal distribution $\mathcal{N}_p(0, \mathbf{\Sigma})$ with mean zero and covariance matrix $\mathbf{\Sigma}$. We assumed that every predictor had a standard deviation of 1, making the covariance matrix equivalent to a correlation matrix. Four different correlation matrix structures were considered in our study, which are discussed later.

We then generated an $n \times 1$ error vector $\mathbf{e} \sim \mathcal{N}(0, \sigma^2)$ with mean zero and variance σ^2. The response $\mathbf{y}$ can then be computed by

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{e} \tag{21}$$

We used a **factorial design** for our simulations. This means that we considered several factors that affect the simulated data in $\mathbf{X}$, each having multiple possible values. We then generated data using every possible combination of factor values, giving us a comprehensive assessment of model performance under various conditions. The factors that we controlled in our simulation are:

- n, the number of observations (50, 200, and 1000),
- p, the number of predictors (10, 100, and 2000),
- σ, the standard deviation of the random error (1, 3, and 6),
- The correlation matrix structure (independent, symmetric compound, autoregressive, and blockwise), and
- ρ, the correlation between predictors (0.2, 0.5, and 0.9)

By taking every possible combination of these factors, we obtain $3 \times 3 \times 3 \times 4 \times 3 = 324$ different settings for the simulations. However, because an independent correlation matrix does not have any correlation between predictors, the value of ρ is not used. Hence, we only needed to run 270 different settings. For each combination of factors, we ran 100 simulations. In each simulation, we generated two data sets: one to train the various models, and one to test the models and evaluate performance. Both data sets contained n observations, meaning that a total of $2n$ observations were generated for each simulation.

As mentioned earlier, we considered four different covariance matrix structures. These structures determine the correlation between different predictors. If $\mathbf{\Sigma}$ is a correlation matrix, then $\mathbf{\Sigma}_{ij}$, the entry at the i-th row and j-th column, represents the correlation between predictors i and j. If $\mathbf{\Sigma}_{ij} = 0$, there is no correlation; but if $\mathbf{\Sigma}_{ij} = 1$, then predictors i and j are perfectly correlated. Note that a correlation matrix is always symmetric, so $\mathbf{\Sigma}_{ij} = \mathbf{\Sigma}_{ji}$ for all indices i and j. This correlation can severely impact the performance of statistical models; if several predictors are highly correlated, then machine learning algorithms are less able to determine which predictors are actually related to the response.

The first correlation structure we considered is **independent correlation**. This means that the correlation matrix $\mathbf{\Sigma}$ has the form

$$\mathbf{\Sigma} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{22}$$

In other words, there is no correlation between different predictors, since $\mathbf{\Sigma}_{ij} = 0$ whenever $i \neq j$. Although this is a very simple case, it is very unrealistic.

The next covariance structure is called **symmetric compound**. This structure has the form

$$\mathbf{\Sigma} = \begin{bmatrix} 1 & \rho & \cdots & \rho \\ \rho & 1 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & 1 \end{bmatrix} \tag{23}$$

where $\rho \in [0, 1]$ is some correlation value. A symmetric compound covariance structure assumes that $\mathbf{\Sigma}_{ij} = \rho$ whenever $i \neq j$, meaning that all predictors are equally correlated with one another. By introducing correlation between different predictors, the data generated in our simulations is more realistic. However, a symmetric compound covariance matrix is still relatively simplistic. In real data sets, it is unrealistic to assume that all of the predictors have the exact same correlation with one another.

An autoregressive covariance structure assumes that

$$\mathbf{\Sigma} = \begin{bmatrix} 1 & \rho & \cdots & \rho^{p-1} \\ \rho & 1 & \cdots & \rho^{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho^{p-1} & \rho^{p-2} & \cdots & 1 \end{bmatrix} \tag{24}$$

For any indices i and j, we have $\mathbf{\Sigma}_{ij} = \rho^{|i-j|}$. Consequently, each predictor is strongly correlated with nearby predictors and weakly correlated with more distant predictors. This form of covariance is commonly seen when using time series, since observed values at nearby times are likely to be highly correlated with one another.

Finally, a blockwise correlation matrix has the block-diagonal form

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{B}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{B}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{B}_k \end{bmatrix} \tag{25}$$

where 0 represents a block containing all zeroes, and each block $\mathbf{B}_i$ has a form identical to the symmetric compound matrix in Equation 23. This implies that predictors within the same block have correlation $\rho \in [0, 1]$, whereas predictors in different blocks have zero correlation. This type of correlation is more realistic than independent or symmetric compound correlation, since only certain groups of predictors are correlated with one another. One important consideration when using blockwise correlation is the size of each block. For our simulations, we used a block size of 5 when $p = 10$, a block size of 25 when $p = 100$, and a block size of 100 when $p = 2000$.

2.3 Empirical Data

For empirical data, we used the Breast Cancer database from The Cancer Genome Atlas (bcTCGA). This contains the gene expression data of 17323 genes from 536 patients. One of

these genes is the BRCA1 gene which is among the first genes discovered that can increase the risk of breast cancer. Because the BRCA1 gene interacts with other genes, it is useful to find genes that interact with BRCA1 to test in further studies. The distribution of the BRCA1 gene expression levels in the bcTCGA database can be seen in Figure 6. The BRCA1 gene expression level will act as the output value in our regression analysis and the other 17322 genes will serve as predictor values.

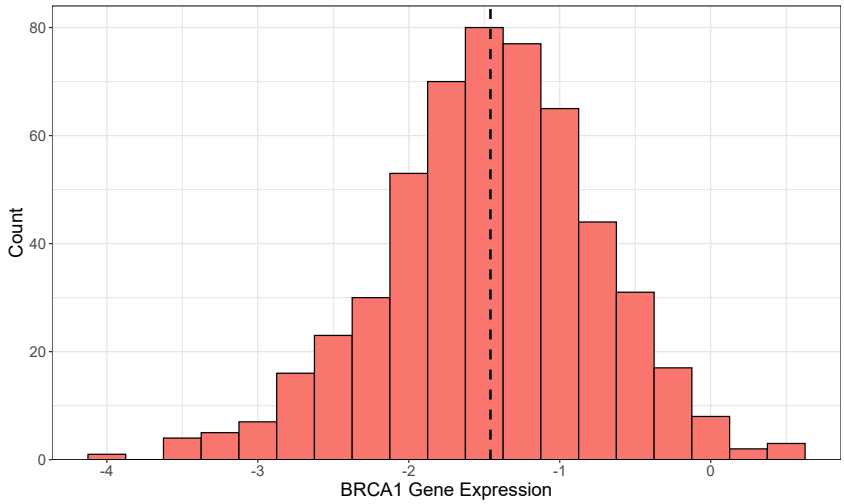


Figure 6: Distribution of BRCA1 gene expression levels. This is the variable that we used as the response. The mean gene expression is -1.459.

This data is a prime example of the $p >> n$ problem where there are many more predictors than data samples. Because of this, only penalized regression and machine learning techniques can be utilized. This is because there are more predictors than samples which makes least squares linear regression impossible and due to the high number of predictors, subset and stepwise regression becomes too computationally expensive to be feasible. Additionally, support vector machines struggle at such a high number of predictors and resulted in stack overflow errors which made fitting support vector machines on this data infeasible.

For analyzing the error of models fitted using this empirical data, we performed nested cross validation. Many of the models fitted used cross validation for tuning purposes to find the best input parameters. Examples of these parameters include λ for penalized regression techniques, the number of trees and number of predictors for random forest, as well as max depth and learning rate for XGBoost. However, we also perform 5-fold cross validation before we even allow the models to tune their parameters. By doing nested cross validation, testing data is not used for training the model **or** tuning model parameters. This results in a more accurate error calculation. If nested cross validation was not used, models could tune their parameters in a way that overfits the model to the testing data which would lead to artificially decreased error and invalid results.

Proposed alternative to the above paragraph: To evaluate the models, we used **nested cross validation**. This means that we first split the data into five folds. For each of these folds, we used the selected fold as a test set while the other four folds were used as a training set. We then fitted the models using cross-validation on this training set, where one interior

fold was used as a validation set while the other folds were used to train a model. The role of the validation set in the interior cross validation is different from the test set used in the exterior cross validation. In the interior cross validation, the validation set is used to tune hyperparameters; the model that performs best against the validation set is then chosen. On the other hand, the test set in the exterior cross validation is not used to tune hyperparameters; its only purpose is to evaluate the performance of the models chosen in the inner cross validation. Because the test set is not used in the model fitting or selection process, it gives an unbiased evaluation of each model's performance.

We chose to use nested cross validation because it produces five models that were fitted using different subsets of the data for training and testing. If we had only fit one model, the subset of the data we choose for training and testing can have a huge impact on our findings. By using five models that are fit with different subsets of the data, we get a more accurate view of how each model performs in general. Cross validation also allows us to get an idea of how much variance each of these models has by comparing the results between different folds.

The hyperparameters tuned in each of the models were the same as those tuned in the Monte Carlo simulations. For ridge, lasso, elastic-net, SCAD, and MCP, we tuned the penalty strength λ; for elastic-net, we used the hyperparameter $\alpha = 0.5$, meaning that the penalty is in between that of lasso and ridge.

3 Results

3.1 Monte Carlo Simulations

We used four metrics to evaluate the performance of each model: **train mean squared error**, **test mean squared error**, β**-sensitivity** and β**-specificity**. The mean squared error is computed by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{26}$$

where y_i is the value of the response and $\hat{y}_i$ is the predicted response value for observation i. In other words, the mean squared error is the average of the squared errors. The mean squared error was computed on both the n observations used to train the models and the n observations that were not used for training, giving us both a training error and a test error.

Because we are using simulated data, where the true coefficient values are known, we can measure the β-sensitivity and β-specificity for each model [20]. For each of the models that performs variable selection, we first measured the number of true positive, true negatives, false positives, and false negatives for the coefficient estimates. For any model, a coefficient estimate is a **true positive** the coefficient is predicted to be non-zero when it is actually non-zero. The estimate is a **true negative**if the coefficient was correctly predicted to be zero. A **false positive** happens when the coefficient is predicted to be non-zero but is actually zero. Finally, a coefficient estimate is a **false negative** if it was estimated to be zero but is actually non-zero.

For a particular model, let TP be the number of true positives, TN the number of true negatives, FP the number of false positives, and FN the number of false negatives. Then the β-sensitivity model the model can be computed by

$$\beta\text{-sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{27}$$

The β-sensitivity is a measure of a model's ability to correctly identify non-zero coefficients. If the β-sensitivity is close to 1, then the model recognizes almost all of the non-zero coefficients; if the β-sensitivity is close to 0, then the model cannot identify non-zero coefficients well. Similarly, the β-specificity can be computed by

$$\beta\text{-specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{28}$$

The β-specificity measures a model's ability to recognize coefficients that are actually zero.

Because we ran simulations using 270 different combinations of factors, we will just highlight the results for the two extremes: when $n = 1000$ and $p = 10$, and when $n = 50$ and $p = 2000$. All of the plots in this section use facet plots to convey as much information as possible. Each plot measures the average value for one of the four metrics discussed above over 100 simulations. Each row represents a different value of σ, the standard deviation of the random error. Each column represents a correlation structure. The different shapes and colors for each points represent the strength of the correlation between predictors.

To save space in the plots, each model is given a shortened label. Many of these labels have been used throughout this paper; for example, ordinary least squares is labelled as OLS. The labels for the wrapper methods start with AIC or BIC, followed by either B for backward, SB for stepwise backward, F for forward or SF for stepwise forward. As an example, the stepwise forward model evaluated with BIC is labelled as "AIC SF."

We will first consider the case where $n = 1000$ and $p = 10$. Figures 7 and 8 show the average mean squared error for each model for the training set and test set, respectively. Tables 4 and 5 in Appendix A contain all of the data shown in these figures as well as the standard deviation for each point.

We see that the linear models have very similar train and test mean squared errors. For these models, the mean squared error is approximately equal to σ^2, the square of the random error. We also see that for all of the linear models except for ridge regression, the type and strength of correlation has little effect on model performance. However, ridge regression performs worse when the correlation is high.

The non-linear models all have lower training mean squared errors compared to the linear models. However, they also have higher test mean squared errors. We see that gradient boosting is not affected too strongly by the amount of correlation; on the other hand, we see that random forest and support vector machine models have better test performance when there is a strong correlation between predictors.

Figure 9 displays the average β-sensitivity for each model that performs variable selection when $n = 1000$ and $p = 10$. We see that almost all the models were able to correctly identify all of the non-zero coefficients when $n = 1000$ and $p = 10$. The only exceptions were lasso and elastic-net models when the correlation structure was blockwise and the strength of the correlation was 0.9.

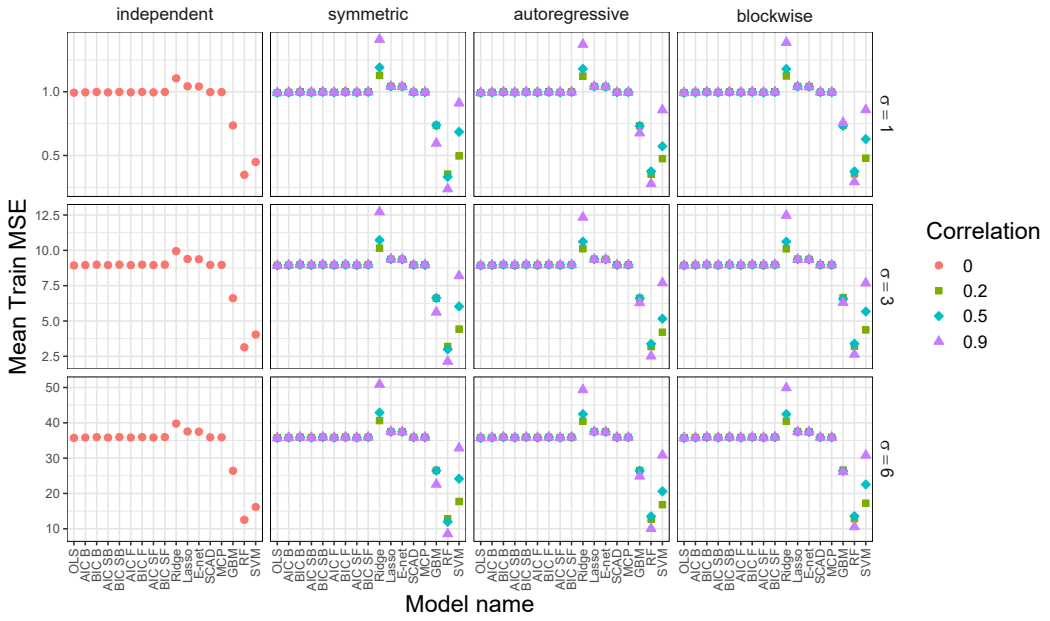


Figure 7: Average mean squared error using the training data for all simulated models when $n = 1000$ and $p = 10$.

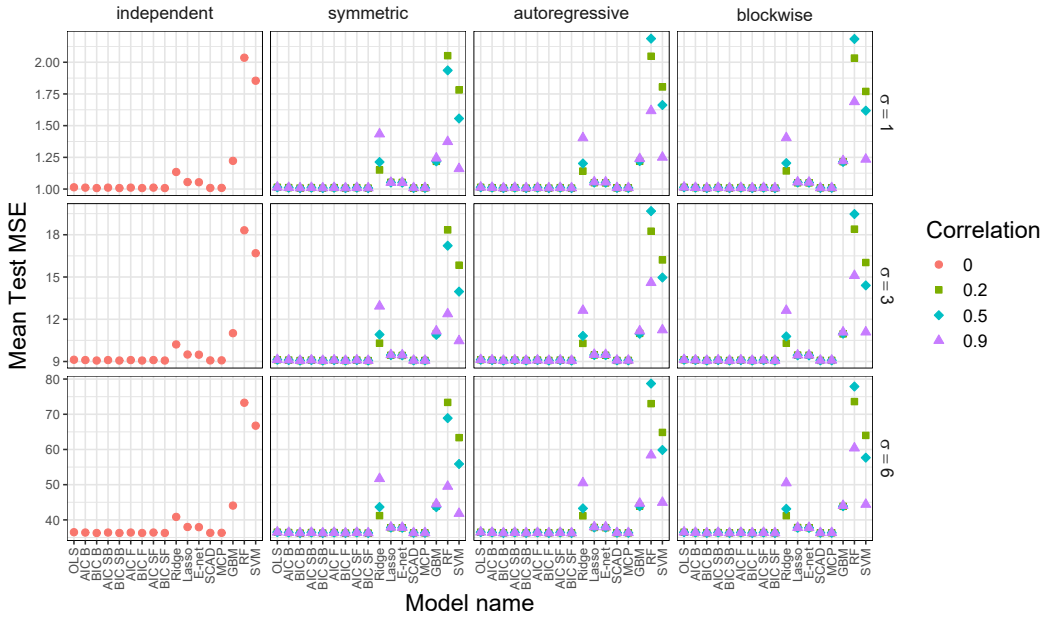


Figure 8: Average mean squared error using the test data for simulated models when $n = 1000$ and $p = 10$.

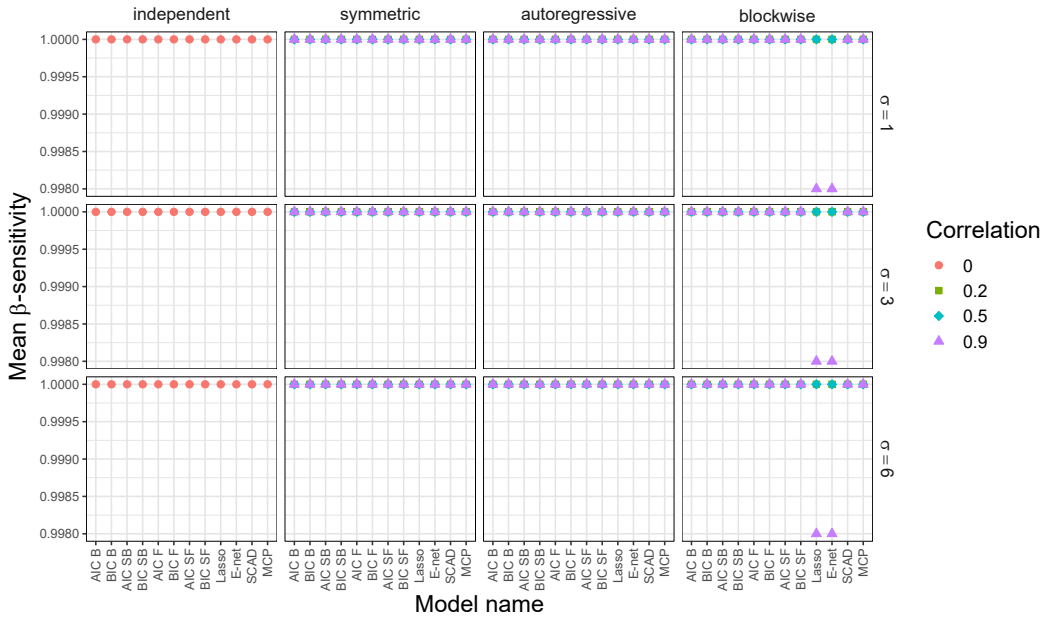


Figure 9: Average β-sensitivity for simulated models when $n = 1000$ and $p = 10$.

Figure 10 shows the average β-specificity for each model. We see that the subset selection models that used BIC made almost zero mistakes when identifying predictors as zero. In comparison, wrapper models fitted using AIC made more mistakes. We see that lasso and elastic-net made almost no mistakes when the correlation was low; in fact, when the correlation structure was independent, lasso made the fewest incorrect estimates out of all the models. As the correlation increases, lasso and elastic-net made more incorrect predictions. SCAD and MCP made more incorrect predictions than lasso and elastic-net in general. However, SCAD and MCP were also somewhat more resilient to an increase in correlation.

Now, we will consider the opposite extreme where $n = 50$ and $p = 2000$. Figures 11 and 12 display the average mean squared error for simulations where $n = 50$ and $p = 2000$. Because less models were fitted for this combination of n and p, less models are shown in this figure. See table 6 and table 7 for a table of these results, including the standard deviations for each point.

We see that the mean squared error for lasso and elastic-net are generally larger than SCAD and MCP for both the training data and test data. Gradient boosting has almost zero training mean squared error under all conditions, but has a relatively large test error. Random forest and support vector machine models have a moderate training error but a large test error. Interestingly, we see that the non-linear models all perform better when there is a strong correlation between predictors. On the other hand, the linear models are somewhat less affected by the correlation.

Figure 13 measures the average β-sensitivity for each model. Recall that the subset selection methods were not considered here because $p < n$. We see that all of the models

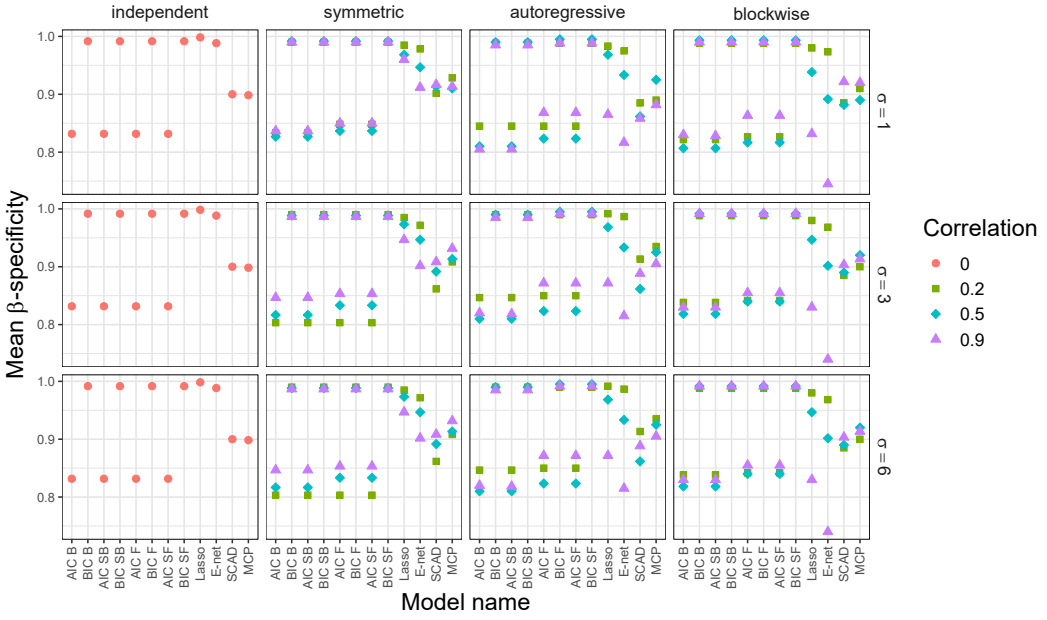


Figure 10: Average β-specificity for simulated models when $n = 1000$ and $p = 10$.

predict most of the non-zero coefficients when the correlation is low. When the correlation is high, all of the models struggle to identify the correct predictors. SCAD and MCP perform the best when the correlation is low but perform the worst when the correlation is high. Elastic-net performs particularly well compared to the other models when the correlation is high, especially when the correlation structure is autoregressive.

Figure 14 shows the average β-specificity $n = 50$ and $p = 2000$. MCP appears to make the fewest mistakes when choosing zero coefficients. The performance of the other models depends heavily on the type of correlation and the correlation strength. Lasso and elastic-net perform the worst when the correlation structure is symmetric compound, whereas SCAD performs poorly when the correlation structure is autoregressive or blockwise. We also see that the models generally made less mistakes as the correlation increases.

3.2 Empirical Data

Recall that we used nested cross-validation when fitting models on the bcTCGA data set. This means that we fitted five models using different subsets of the data for training and testing. Figure 15 below shows a plot with the training and test mean squared error for every fold of every model. The bars show the average mean squared error for the five folds. In addition, Table 2 show the aggregated results for the train and test mean squared error.

The models can also be compared to each other by comparing the most important predictors selected by each model. Figure 16 shows five Venn diagrams. Each diagram shows the number of variables selected by lasso, elastic-net, MCP, and random forest as

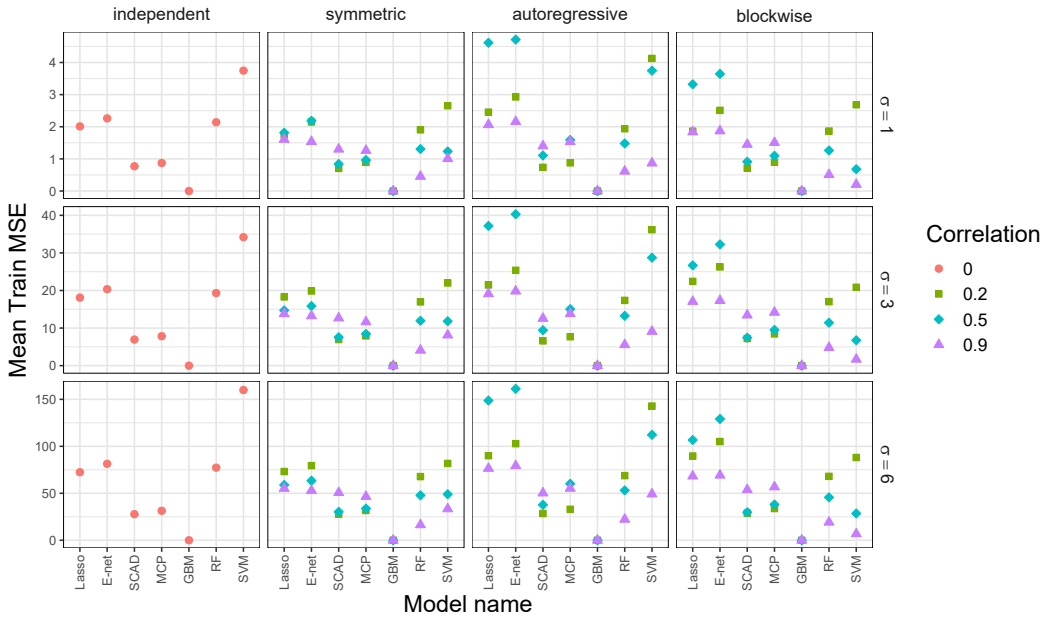


Figure 11: Average mean square error using training data for simulated models when $n = 50$ and $p = 2000$.

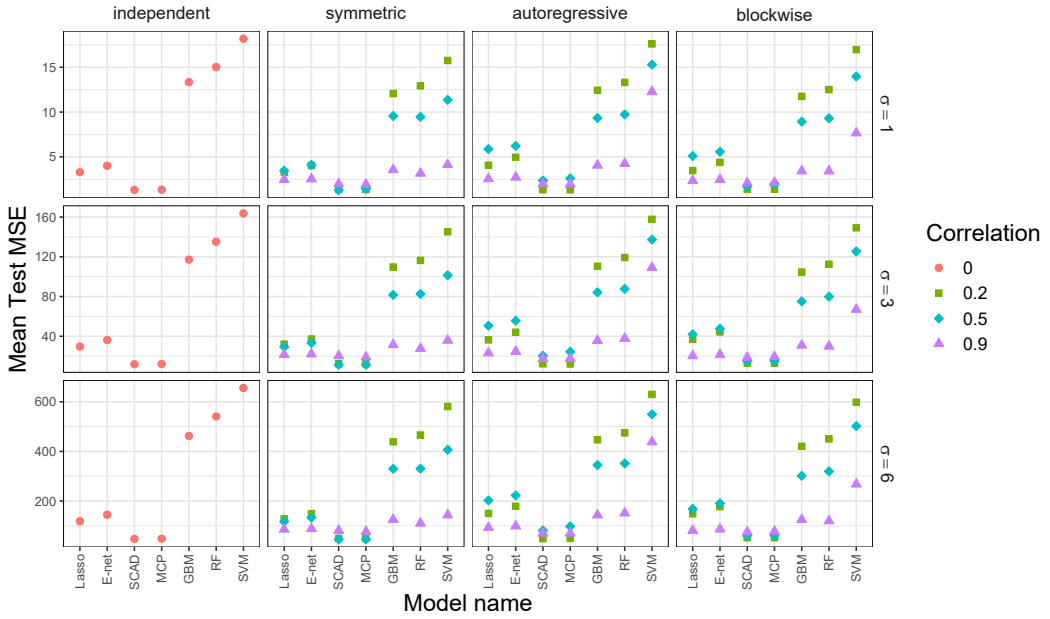


Figure 12: Average mean square error using testing data for simulated models when $n = 50$ and $p = 2000$.

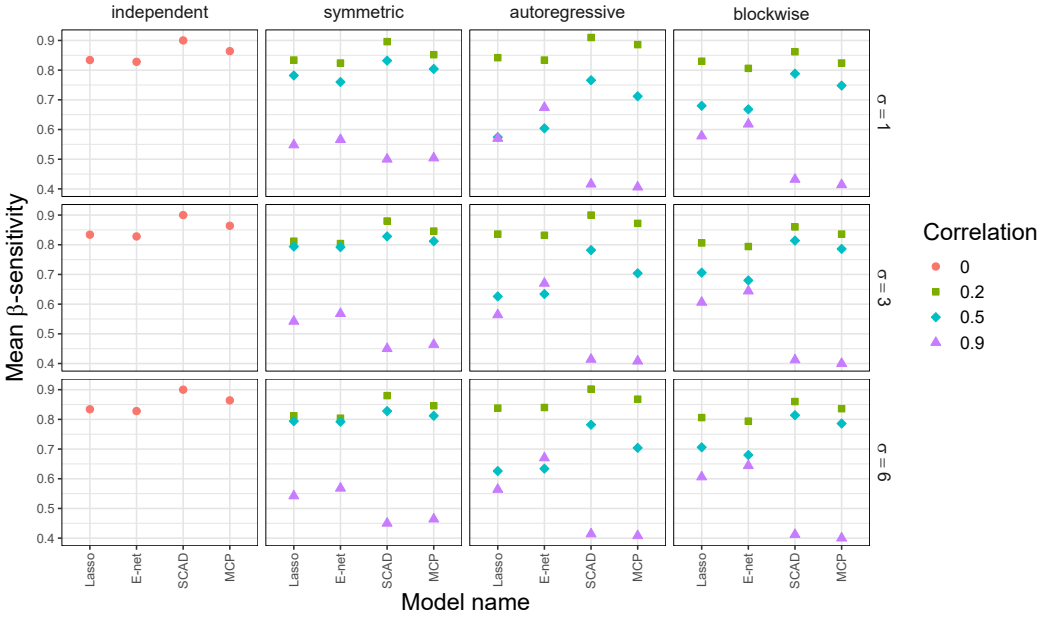


Figure 13: Average β-sensitivity for simulated models when $n = 50$ and $p = 2000$

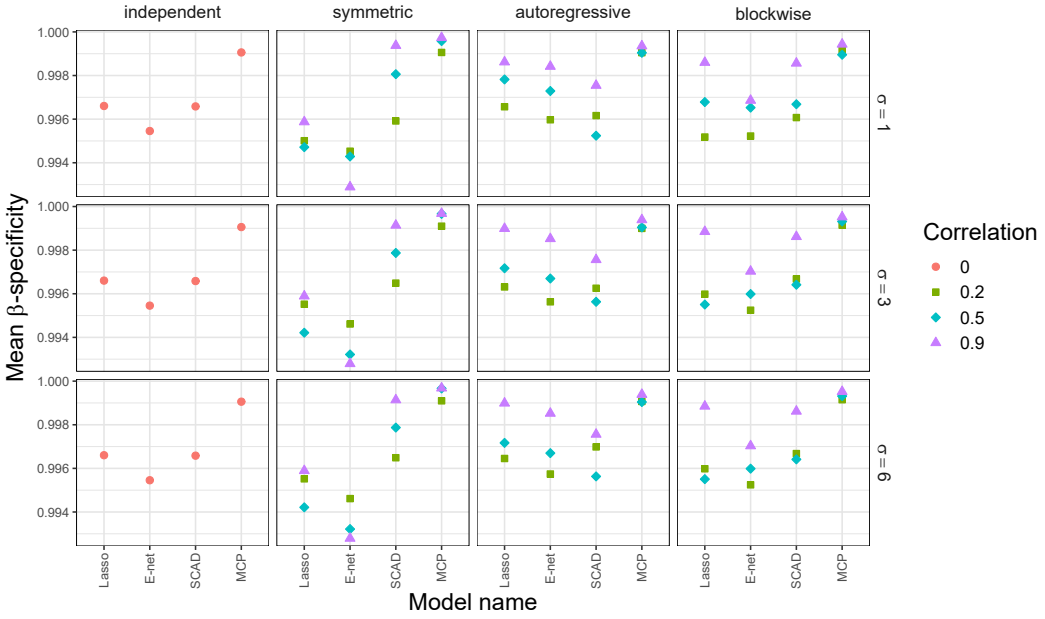


Figure 14: Average β-specificity for simulated models when $n = 50$ and $p = 2000$.

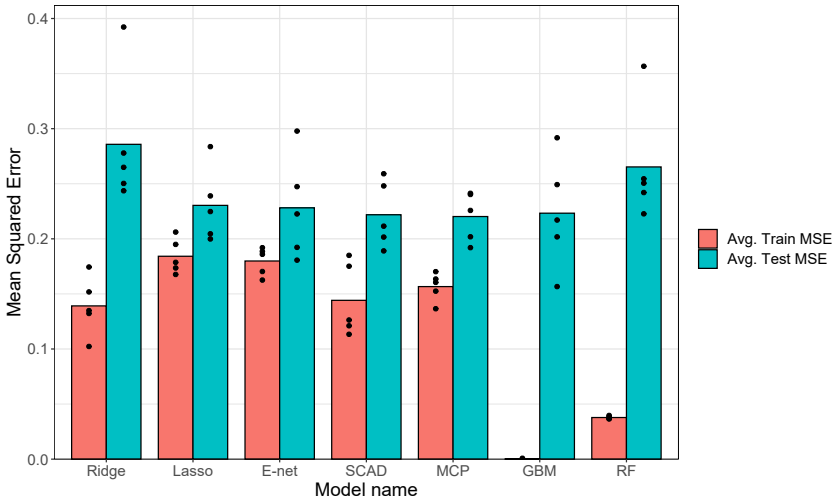


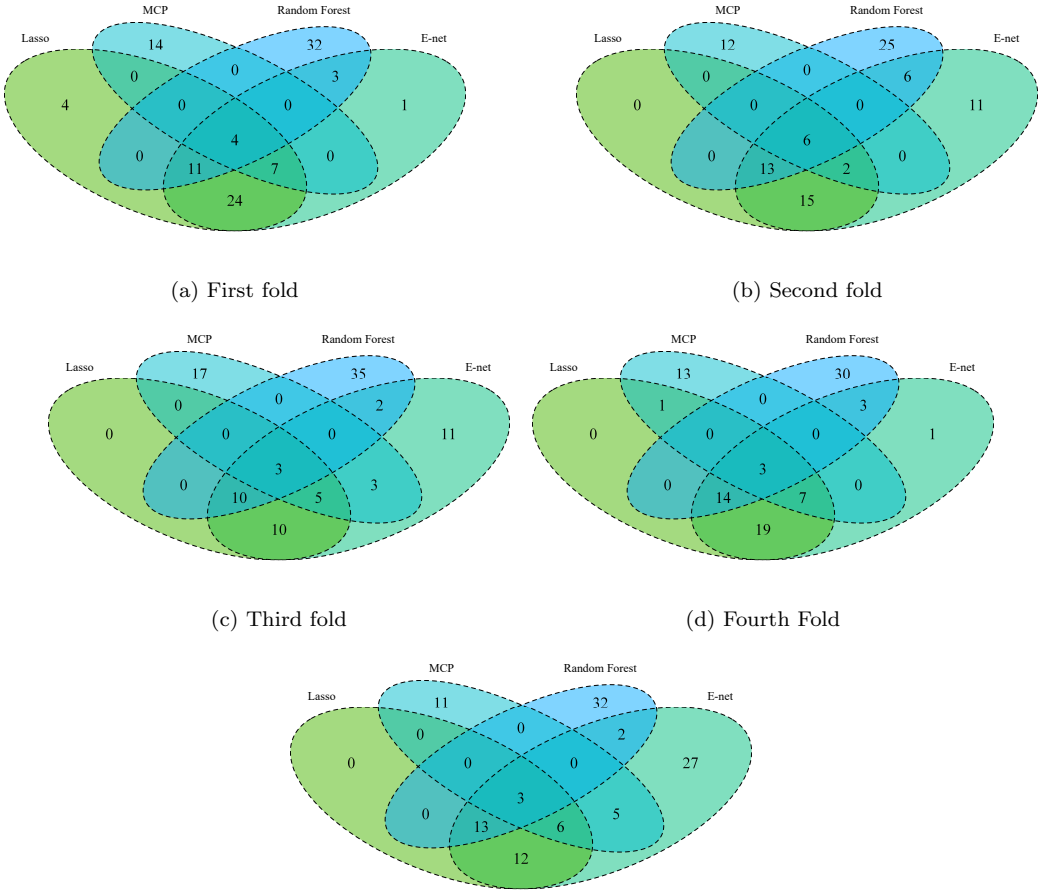
Figure 15: Mean squared error of the models fit on the bcTCGA data set. Each point represents the mean squared error for one fold, while the bars represent the average for the five folds.

Table 2: Average and standard deviation of the mean squared error of the models fit on the bcTCGA data set.

	Train MSE		**Test MSE**	
Model	**Average**	**SD**	**Average**	**SD**
Ridge	0.1391	0.0266	0.2858	0.0610
Lasso	0.1842	0.0159	0.2304	0.0337
E-net	0.1799	0.0127	0.2281	0.0469
SCAD	0.1442	0.0333	0.2218	0.0303
MCP	0.1566	0.0129	0.2202	0.0224
GBM	0.0002	0.0004	0.2233	0.0507
RF	0.0378	0.0013	0.2653	0.0525

the most important for each of the folds. Figure 16a, for example, shows that four of the predictors were chosen by all four models as important. For lasso, elastic-net, and MCP, a predictor is considered important if its coefficient is non-zero. For random forest, we used the 50 predictors with the highest importance score (as computed by `randomForest`). We see that the predictors chosen by each model varies slightly among different folds, however, there still are common trends that are upheld between the different folds. The second fold in 16b had six predictors common to all four models, whereas the models tested in the third fold in 16c only had three.

Table 3 shows the average runtime to fit each of the models on the bcTCGA data set.



(a) First fold

(b) Second fold

(c) Third fold

(d) Fourth Fold

(e) Fifth Fold

Figure 16: Venn diagrams of the predictors selected by lasso, elastic-net, MCP, and random forest models for each of the five folds. Each number represents the number of important predictors chosen by the models that overlap the number.

Table 3: Average Runtimes for Empirical Data Models

Model	**Average Runtime** (seconds)
Ridge	29.29
Lasso	9.57
E-net	9.41
SCAD	17.28
MCP	15.15
GBM	538.12
RF	4906.59

4 Discussion

4.1 Findings

4.1.1 Monte Carlo Simulations

Looking at the simulations, we see that the wrapper methods minimize the test mean squared error, along with SCAD and MCP. In addition to having low test error, these models are also fairly resilient to large correlation values.

Ridge regression performed far worse than the other linear models when $n = 1000$ and $p = 10$. Furthermore, the strength of correlation had a more significant effect on ridge than the other linear models. This increase in error is likely because ridge regression cannot set any coefficients to zero; as a result, the predictors that are related to the response are given smaller coefficient values. Recall that four out of the five true coefficient values are larger than 1. Because the ridge penalty function squares each coefficient, it will heavily punish coefficient estimates close to those true coefficient values.

The non-linear models all appear to overfit to the training data. In particular, when $n = 1000$ and $p = 10$, random forest models had almost zero training error. When $n = 50$ and $p = 2000$, gradient boosting machine models had almost zero training error. The non-linear models performed very poorly on the test data, which indicates that these models likely became overfit to the training data.

Interestingly, the mean squared error for the non-linear models generally improved as the correlation between predictors was increased. There are a few possible reasons for this. For random forest and support vector machine models, the increased correlation may have mitigated the amount of overfitting and better generalized the model. When $n = 1000$ and $p = 10$, Figure 7 shows that the training mean-squared error for support vector machines increased as the correlation increased. However, the training error for gradient boosting and random forest models were less affected by the strength of correlation in this case. Furthermore, when $n = 50$ and $p = 1000$, both training error and test error decreased as correlation increased.

Another possible explanation for this phenomenon is that the non-linear models are less influenced by the random error when the correlation is large. For example, consider random forest models when $n = 50$ and $p = 2000$. When the correlation is zero, only a fraction of the decision trees generated in each model will contain any of the important predictors; many of the trees will instead be fit using only predictors that are unrelated to the response. This means that the random forest model is fitting with a large amount of noise. On the other hand, when the correlation is high, all of the trees will contain predictors that somewhat correlated to some of the important predictors that are related to the response. As a result, none of the trees will truly be fitting on just random error. Similarly, for gradient boosting and support vector machine models, the strong correlation may make it more difficult to fit using just the random noise.

Although wrapper methods and penalized regression models were the clear winners in this simulation study, it is worth noting that the non-linear models suffered from a few unfair disadvantages. For one, the simulated data assumed that there was a linear relationship between the predictors and response, which meant that the linear models were almost

unbiased. The non-linear models were relatively more biased because they did not assume a linear relationship.

Another disadvantage for the non-linear models is that they can be controlled by many different hyperparameters, but we only considered a couple for each model. This was out of necessity; it would have been infeasible to fit models using more hyperparameters given the number of simulations that had to be run. If more hyperparameters could be included, then the performance of the non-linear models may have been more comparable to the linear models. At the same time, the more comprehensive tuning of several hyperparameters could act as a double edged sword in that it could increase the overfitting of these models. More hyperparameter tuning could lead a model to pick hyperparameters that fits the training data very precisely, but at the expense of increased testing error and increased computational cost.

4.1.2 Empirical Data

In our empirical data study, SCAD and MCP also maintained the lowest testing mean squared error among the tested models. This can be seen in Figure 15 and Table 2 which showcase MCP as having the lowest average MSE and standard deviation. This may be due in part to the fact that MCP consistently selected the fewest number of predictors compared to Lasso and Elastic Net as visible in Figure 16. Because this empirical data most likely contains multicollinearity, the fact that MCP selects fewer predictors may have helped to reduce overfitting. This is consistent with results from the Monte Carlo simulations which indicate that MCP performed better than its counterparts in multicollinear environments.

An interesting result was how few predictors were selected between all four models. It would be expected that the most important predictors would be chosen by all four models, but only three to six predictors were selected in common by all the models. Between MCP, Lasso, and Elastic Net, there were slightly more selected predictors in common, but these still were much less than expected. This is most likely because MCP is very different from the other regression techniques in that it uses non-convex optimization and thus selected many predictors that were different from all the other models. Since MCP had a much lower false negative rate compared to the other models as determined in the Monte Carlo simulations, it can be assumed that many of these predictors solely selected by MCP were significant predictors that were missed by the other models. This could have been another reason why MCP performed the best out of all the fitted models on the empirical data.

MCP also maintained the lowest testing MSE standard deviation for the empirical data as seen in Figure 2. This is to be expected given the bias/variance tradeoff. Because MCP selects fewer predictors than the other models, it incurs greater bias but while also lessening variance. By definition, this means that MSP is less sensitive to changes in training data, such as changes in training data due to 5-fold cross validation.

The penalized regression models performed exceptionally faster than random forest and XGBoost as documented in Table 3. On average, Lasso and Elastic Net ran approximately 56x faster than XGBoost and 510x faster than random forest. MCP and SCAD ran approximately 30x faster than XGBoost and 290x faster than random forest. This provides a significant advantage to the penalized regression techniques, especially given that MCP and other penalized techniques performed better than random forest and XGBoost.

4.2 Contributions

There is a severe lack of comprehensive testing comparing traditional machine learning methods such as random forest, gradient boosting, and support vector machines with penalized selection. Our paper bridges the divide between the machine learning and statistical fields in which these two types of models exist in. Testing using Monte Carlo simulations and empirical data has not been tested by other researchers with as many different environments and models.

4.3 Future Work

A Full Results

Table 4: Mean and standard deviation of the mean squared error on training data when $n = 1000$ and $p = 10$.

	Type	independent		symmetric						autoregressive						blockwise					
	Correlation	0		0.2		0.5		0.9		0.2		0.5		0.9		0.2		0.5		0.9	
Error	Model	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
	OLS	0.99	0.04	0.99	0.04	0.99	0.04	0.99	0.04	0.99	0.04	0.99	0.04	0.99	0.04	0.99	0.04	0.99	0.04	0.99	0.04
	AIC B.	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	0.99	0.04	1.00	0.04	0.99	0.04	1.00	0.04
	BIC B	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04
	AIC SB	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	0.99	0.04	1.00	0.04	0.99	0.04	1.00	0.04
	BIC SB	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04
	AIC F	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04
	BIC F	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04
	AIC SF	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04
1	BIC SF	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04
	Ridge	1.11	0.05	1.13	0.05	1.19	0.05	1.41	0.05	1.12	0.05	1.18	0.05	1.37	0.05	1.12	0.05	1.18	0.05	1.39	0.05
	Lasso	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05
	E-net	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05	1.04	0.05
	SCAD	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04
	MCP	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04	1.00	0.04
	GBM	0.74	0.04	0.74	0.04	0.74	0.04	0.60	0.33	0.73	0.04	0.73	0.05	0.68	0.27	0.74	0.04	0.73	0.04	0.76	0.14
	RF	0.35	0.01	0.35	0.01	0.33	0.01	0.24	0.01	0.35	0.02	0.38	0.01	0.28	0.01	0.36	0.01	0.37	0.02	0.29	0.01
	SVM	0.45	0.03	0.50	0.05	0.69	0.11	0.91	0.05	0.47	0.06	0.57	0.08	0.86	0.06	0.48	0.03	0.63	0.10	0.86	0.06
3	OLS	8.93	0.39	8.93	0.39	8.93	0.39	8.93	0.39	8.93	0.39	8.93	0.39	8.93	0.39	8.93	0.39	8.93	0.39	8.93	0.39
	AIC B.	8.96	0.39	8.95	0.39	8.95	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39
	BIC B	8.99	0.40	8.99	0.39	8.99	0.39	8.98	0.39	8.98	0.39	8.99	0.39	8.98	0.39	8.99	0.39	8.99	0.39	8.99	0.39
	AIC SB	8.96	0.39	8.95	0.39	8.95	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39
	BIC SB	8.99	0.40	8.99	0.39	8.99	0.39	8.98	0.39	8.98	0.39	8.99	0.39	8.98	0.39	8.99	0.39	8.99	0.39	8.99	0.39
	AIC F	8.96	0.39	8.95	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39
	BIC F	8.99	0.40	8.99	0.39	8.99	0.39	8.98	0.39	8.98	0.39	8.99	0.39	8.99	0.39	8.99	0.39	8.99	0.39	8.99	0.39
	AIC SF	8.96	0.39	8.95	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39	8.96	0.39
	BIC SF	8.99	0.40	8.99	0.39	8.99	0.39	8.98	0.39	8.98	0.39	8.99	0.39	8.99	0.39	8.99	0.39	8.99	0.39	8.99	0.39
	Ridge	9.95	0.42	10.16	0.44	10.72	0.44	12.72	0.49	10.11	0.43	10.62	0.43	12.34	0.47	10.11	0.42	10.61	0.42	12.47	0.50
	Lasso	9.39	0.42	9.38	0.42	9.38	0.42	9.38	0.42	9.38	0.41	9.38	0.41	9.37	0.41	9.38	0.42	9.38	0.42	9.36	0.42
	E-net	9.37	0.44	9.37	0.43	9.36	0.43	9.36	0.43	9.36	0.43	9.36	0.43	9.35	0.43	9.37	0.43	9.36	0.43	9.34	0.43
	SCAD	8.97	0.39	8.97	0.40	8.97	0.39	8.97	0.39	8.97	0.39	8.97	0.40	8.97	0.39	8.97	0.40	8.97	0.40	8.97	0.39
	MCP	8.97	0.39	8.98	0.40	8.97	0.39	8.97	0.39	8.97	0.39	8.98	0.39	8.97	0.40	8.97	0.39	8.97	0.39	8.97	0.39
	GBM	6.61	0.34	6.61	0.34	6.63	0.38	5.62	2.82	6.62	0.33	6.61	0.38	6.28	2.09	6.65	0.34	6.57	0.41	6.31	2.09
	RF	3.14	0.13	3.20	0.14	2.99	0.14	2.14	0.09	3.19	0.12	3.38	0.13	2.51	0.11	3.21	0.13	3.39	0.14	2.64	0.12
	SVM	4.04	0.26	4.43	0.28	6.03	0.86	8.19	0.45	4.20	0.27	5.16	0.79	7.70	0.53	4.37	0.50	5.67	0.88	7.68	0.47
6	OLS	35.73	1.56	35.73	1.56	35.73	1.56	35.73	1.56	35.73	1.56	35.73	1.56	35.73	1.56	35.73	1.56	35.73	1.56	35.73	1.56
	AIC B.	35.83	1.56	35.82	1.56	35.82	1.56	35.84	1.57	35.83	1.57	35.82	1.56	35.82	1.56	35.83	1.56	35.82	1.56	35.83	1.56
	BIC B	35.95	1.60	35.95	1.57	35.94	1.57	35.93	1.58	35.93	1.57	35.95	1.57	35.93	1.57	35.94	1.58	35.95	1.57	35.95	1.57
	AIC SB	35.83	1.56	35.82	1.56	35.82	1.56	35.84	1.57	35.83	1.57	35.82	1.56	35.82	1.56	35.83	1.56	35.82	1.56	35.83	1.56
	BIC SB	35.95	1.60	35.95	1.57	35.94	1.57	35.93	1.58	35.93	1.57	35.95	1.57	35.93	1.57	35.94	1.58	35.95	1.57	35.95	1.57
	AIC F	35.83	1.56	35.82	1.56	35.83	1.56	35.84	1.57	35.83	1.56	35.83	1.56	35.85	1.56	35.83	1.56	35.83	1.57	35.84	1.56
	BIC F	35.95	1.60	35.95	1.57	35.94	1.57	35.93	1.58	35.93	1.57	35.95	1.56	35.94	1.58	35.94	1.58	35.95	1.57	35.95	1.57
	AIC SF	35.83	1.56	35.82	1.56	35.83	1.56	35.84	1.57	35.83	1.56	35.83	1.56	35.85	1.56	35.83	1.56	35.83	1.57	35.84	1.56
	BIC SF	35.95	1.60	35.95	1.57	35.94	1.57	35.93	1.58	35.93	1.57	35.95	1.56	35.94	1.58	35.94	1.58	35.95	1.57	35.95	1.57
	Ridge	39.78	1.68	40.64	1.76	42.89	1.78	50.87	1.96	40.45	1.73	42.46	1.73	49.34	1.89	40.43	1.70	42.46	1.68	49.89	2.00
	Lasso	37.54	1.68	37.51	1.67	37.51	1.66	37.51	1.67	37.52	1.66	37.50	1.66	37.46	1.66	37.52	1.66	37.50	1.67	37.44	1.67
	E-net	37.48	1.75	37.46	1.71	37.45	1.71	37.44	1.72	37.46	1.73	37.44	1.73	37.40	1.71	37.46	1.73	37.44	1.71	37.38	1.73
	SCAD	35.90	1.57	35.89	1.58	35.89	1.57	35.88	1.57	35.90	1.57	35.88	1.58	35.87	1.57	35.89	1.58	35.90	1.58	35.89	1.58
	MCP	35.89	1.57	35.90	1.58	35.89	1.58	35.90	1.58	35.89	1.57	35.91	1.58	35.88	1.58	35.89	1.57	35.90	1.57	35.88	1.56
	GBM	26.42	1.46	26.50	1.30	26.47	1.43	22.57	10.98	26.52	1.35	26.43	1.39	24.88	9.02	26.62	1.33	26.37	1.56	26.19	6.76
	RF	12.54	0.51	12.79	0.56	11.97	0.54	8.55	0.36	12.75	0.50	13.53	0.52	10.04	0.45	12.83	0.53	13.57	0.56	10.55	0.50
	SVM	16.16	1.04	17.72	1.11	24.16	3.62	32.84	1.76	16.81	1.08	20.60	3.25	30.84	2.09	17.26	1.03	22.55	3.45	30.80	1.94

Table 5: Test2

		type																	
		independent		symmetric				autoregressive						blockwise					
		corr		corr				corr						corr					
		0		0.2		0.5		0.9		0.2		0.5		0.9		0.2		0.5		0.9	
		test.mse		test.mse		test.mse		test.mse		test.mse		test.mse		test.mse		test.mse		test.mse		test.mse	
st.dev	model.name	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
1	OLS	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	AIC B.	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	BIC B	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	AIC SB	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	BIC SB	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	AIC F	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	BIC F	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	AIC SF	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	BIC SF	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	Ridge	1.13	0.06	1.15	0.06	1.21	0.06	1.43	0.07	1.14	0.06	1.20	0.07	1.40	0.06	1.14	0.06	1.20	0.06	1.40	0.08
	Lasso	1.06	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.06	0.05	1.05	0.05	1.05	0.05	1.05	0.05
	E-net	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05	1.05	0.05
	SCAD	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04
	MCP	1.01	0.05	1.01	0.04	1.01	0.05	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.04	1.01	0.05
	GBM	1.22	0.07	1.22	0.06	1.24	0.10	1.22	0.06	1.22	0.06	1.24	0.08	1.22	0.06	1.22	0.06	1.21	0.06	1.22	0.07
	RF	2.04	0.15	2.05	0.15	1.94	0.12	1.37	0.06	2.05	0.16	2.19	0.14	1.62	0.08	2.03	0.13	2.18	0.12	1.69	0.08
	SVM	1.85	0.14	1.78	0.12	1.56	0.11	1.16	0.08	1.80	0.12	1.66	0.12	1.25	0.09	1.77	0.12	1.62	0.10	1.23	0.08
3	OLS	9.13	0.40	9.13	0.40	9.13	0.40	9.13	0.40	9.13	0.40	9.13	0.40	9.13	0.40	9.13	0.40	9.13	0.40	9.13	0.40
	AIC B.	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40
	BIC B	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.08	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40
	AIC SB	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40
	BIC SB	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.08	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40
	AIC F	9.10	0.40	9.07	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.09	0.40	9.10	0.40	9.10	0.40	9.10	0.40
	BIC F	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.08	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40
	AIC SF	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.10	0.40	9.09	0.40	9.10	0.40	9.10	0.40	9.10	0.40
	BIC SF	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.08	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40	9.07	0.40
	Ridge	10.21	0.50	10.30	0.51	10.91	0.59	12.92	0.62	10.29	0.55	10.82	0.59	12.62	0.56	10.30	0.55	10.78	0.59	12.62	0.66
	Lasso	9.50	0.45	9.44	0.45	9.45	0.47	9.47	0.44	9.46	0.47	9.46	0.45	9.50	0.44	9.46	0.47	9.44	0.47	9.45	0.44
	E-net	9.48	0.44	9.43	0.44	9.44	0.46	9.46	0.42	9.45	0.45	9.45	0.45	9.49	0.43	9.44	0.46	9.43	0.46	9.45	0.43
	SCAD	9.08	0.40	9.08	0.40	9.08	0.40	9.08	0.40	9.08	0.39	9.08	0.39	9.08	0.40	9.08	0.40	9.08	0.40	9.08	0.40
	MCP	9.08	0.41	9.08	0.40	9.08	0.40	9.08	0.41	9.08	0.39	9.08	0.39	9.08	0.40	9.08	0.40	9.08	0.40	9.09	0.40
	GBM	11.01	0.60	10.98	0.53	10.89	0.54	11.16	0.78	11.00	0.54	10.97	0.53	11.16	0.77	10.96	0.59	10.98	0.57	11.08	0.83
	RF	18.32	1.33	18.35	1.12	17.22	0.99	12.38	0.58	18.25	1.43	19.68	1.31	14.60	0.69	18.40	1.40	19.47	1.18	15.10	0.68
	SVM	16.69	1.28	15.84	0.99	13.96	0.93	10.46	0.77	16.21	1.12	14.96	1.09	11.24	0.77	16.03	1.20	14.40	0.91	11.09	0.67
6	OLS	36.50	1.59	36.50	1.59	36.50	1.59	36.50	1.59	36.50	1.59	36.50	1.59	36.50	1.59	36.50	1.59	36.50	1.59	36.50	1.59
	AIC B.	36.41	1.60	36.41	1.61	36.41	1.60	36.39	1.61	36.41	1.60	36.42	1.59	36.39	1.62	36.39	1.59	36.41	1.61	36.40	1.60
	BIC B	36.28	1.60	36.26	1.62	36.28	1.59	36.26	1.59	36.30	1.58	36.29	1.58	36.29	1.61	36.29	1.60	36.28	1.60	36.28	1.61
	AIC SB	36.41	1.60	36.41	1.61	36.41	1.60	36.39	1.61	36.41	1.60	36.42	1.59	36.39	1.62	36.39	1.59	36.41	1.61	36.40	1.60
	BIC SB	36.28	1.60	36.26	1.62	36.28	1.59	36.26	1.59	36.30	1.58	36.29	1.58	36.29	1.61	36.29	1.60	36.28	1.60	36.28	1.61
	AIC F	36.41	1.60	36.41	1.61	36.40	1.60	36.39	1.61	36.41	1.60	36.41	1.59	36.37	1.60	36.40	1.59	36.40	1.61	36.39	1.61
	BIC F	36.28	1.60	36.26	1.62	36.28	1.59	36.26	1.59	36.30	1.58	36.28	1.59	36.28	1.62	36.29	1.60	36.28	1.60	36.28	1.61
	AIC SF	36.41	1.60	36.41	1.61	36.40	1.60	36.39	1.61	36.41	1.60	36.41	1.59	36.37	1.60	36.40	1.59	36.40	1.61	36.39	1.61
	BIC SF	36.28	1.60	36.26	1.62	36.28	1.59	36.26	1.59	36.30	1.58	36.28	1.59	36.28	1.62	36.29	1.60	36.28	1.60	36.28	1.61
	Ridge	40.85	2.02	41.21	2.02	43.66	2.34	51.70	2.49	41.15	2.18	43.27	2.36	50.50	2.23	41.19	2.21	43.13	2.37	50.49	2.66
	Lasso	37.99	1.80	37.77	1.81	37.82	1.88	37.89	1.77	37.85	1.90	37.85	1.78	37.99	1.76	37.83	1.86	37.76	1.88	37.81	1.75
	E-net	37.93	1.76	37.73	1.75	37.76	1.82	37.82	1.68	37.78	1.78	37.81	1.78	37.95	1.72	37.77	1.83	37.73	1.85	37.78	1.74
	SCAD	36.31	1.60	36.31	1.61	36.32	1.58	36.32	1.60	36.30	1.58	36.34	1.58	36.34	1.61	36.32	1.59	36.33	1.59	36.33	1.59
	MCP	36.33	1.62	36.31	1.61	36.32	1.59	36.30	1.63	36.32	1.57	36.32	1.57	36.33	1.61	36.33	1.60	36.33	1.60	36.34	1.60
	GBM	44.06	2.40	43.89	2.11	43.61	2.18	44.52	2.85	43.94	2.08	43.87	2.18	44.68	3.13	43.86	2.42	43.88	2.17	44.06	2.77
	RF	73.27	5.35	73.37	4.50	68.90	3.97	49.51	2.33	73.02	5.72	78.72	5.21	58.40	2.75	73.60	5.60	77.88	4.72	60.38	2.71
	SVM	66.76	5.12	63.36	3.98	55.88	3.75	41.80	2.98	64.85	4.47	59.86	4.36	44.94	3.12	63.99	4.57	57.67	3.69	44.39	2.74

Table 6: Test3

		type independent corr 0 train.mse		symmetric corr 0.2 train.mse		0.5 train.mse		0.9 train.mse		autoregressive corr 0.2 train.mse		0.5 train.mse		0.9 train.mse		blockwise corr 0.2 train.mse		0.5 train.mse		0.9 train.mse	
st.dev	model.name	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
1	Lasso	2.01	0.84	1.75	1.10	1.81	1.04	1.60	0.48	2.45	1.62	4.61	2.38	2.06	0.74	1.86	1.15	3.32	1.75	1.84	0.47
	E-net	2.26	1.34	2.15	1.42	2.19	1.31	1.54	0.45	2.93	1.83	4.71	2.25	2.16	0.74	2.51	1.81	3.64	1.83	1.87	0.47
	SCAD	0.77	0.34	0.70	0.33	0.84	0.28	1.30	0.43	0.74	0.33	1.11	1.11	1.40	0.56	0.72	0.32	0.91	0.56	1.45	0.47
	MCP	0.87	0.33	0.89	0.34	0.96	0.36	1.26	0.42	0.88	0.27	1.59	1.31	1.53	0.48	0.90	0.30	1.10	0.63	1.51	0.46
	GBM	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	RF	2.14	0.40	1.91	0.38	1.31	0.23	0.45	0.10	1.94	0.43	1.48	0.33	0.61	0.14	1.87	0.41	1.26	0.25	0.51	0.12
	SVM	3.75	3.59	2.65	2.83	1.23	1.44	1.01	0.61	4.13	3.75	3.74	3.26	0.87	1.38	2.69	3.15	0.68	1.02	0.20	0.11
3	Lasso	18.11	7.55	18.29	12.59	14.71	9.87	13.78	4.35	21.47	14.38	37.19	21.15	19.11	6.00	22.42	16.17	26.68	17.69	17.06	3.80
	E-net	20.35	12.10	19.87	12.92	15.86	10.46	13.23	4.64	25.37	17.20	40.30	21.89	19.82	6.51	26.27	18.03	32.27	18.12	17.30	4.35
	SCAD	6.94	3.08	6.95	3.59	7.56	2.44	12.66	3.68	6.59	2.96	9.43	8.15	12.56	4.93	7.22	3.21	7.45	4.30	13.44	3.94
	MCP	7.84	2.93	7.93	2.75	8.43	2.25	11.63	4.00	7.70	2.57	15.04	11.94	13.80	4.18	8.44	3.69	9.52	6.32	14.18	3.40
	GBM	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	RF	19.31	3.60	16.96	3.35	11.94	2.35	4.09	0.85	17.37	3.90	13.25	2.84	5.55	1.23	17.03	3.06	11.41	2.25	4.79	0.91
	SVM	34.19	30.66	22.00	24.86	11.83	15.63	8.12	4.04	36.17	32.02	28.73	29.54	9.09	13.91	20.86	25.79	6.75	9.91	1.69	0.66
6	Lasso	72.43	30.22	73.15	50.35	58.84	39.48	55.14	17.39	90.05	67.36	148.78	84.60	76.43	24.00	89.69	64.66	106.72	70.77	68.24	15.20
	E-net	81.40	48.39	79.47	51.68	63.43	41.85	52.93	18.57	102.83	64.68	161.21	87.58	79.27	26.05	105.07	72.12	129.10	72.49	69.20	17.40
	SCAD	27.77	12.34	27.82	14.35	30.26	9.76	50.65	14.71	28.49	11.19	37.71	32.58	50.22	19.72	28.88	12.84	29.80	17.22	53.75	15.75
	MCP	31.37	11.71	31.72	10.98	33.72	9.00	46.51	16.00	32.96	15.41	60.14	47.76	55.20	16.70	33.78	14.76	38.07	25.28	56.70	13.61
	GBM	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	RF	77.31	14.37	67.78	13.35	47.85	9.60	16.38	3.44	68.81	15.73	53.11	11.43	22.23	4.96	68.02	12.31	45.69	8.92	19.15	3.69
	SVM	159.93	133.99	81.76	94.03	49.00	59.08	33.34	19.99	142.86	132.04	112.23	116.37	49.15	73.90	88.08	105.63	28.52	47.94	6.74	2.65

Table 7: Test4

		type independent corr 0 test.mse		symmetric corr 0.2 test.mse		0.5 test.mse		0.9 test.mse		autoregressive corr 0.2 test.mse		0.5 test.mse		0.9 test.mse		blockwise corr 0.2 test.mse		0.5 test.mse		0.9 test.mse	
st.dev	model.name	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD	Mean	SD
1	Lasso	3.29	1.57	3.30	2.17	3.45	1.61	2.45	0.71	4.07	2.96	5.87	2.27	2.55	0.70	3.47	1.74	5.09	2.24	2.34	0.75
	E-net	4.01	2.40	4.04	2.55	4.11	1.86	2.52	0.74	4.95	3.45	6.22	2.20	2.72	0.81	4.39	2.35	5.56	2.34	2.45	0.76
	SCAD	1.31	0.28	1.36	0.31	1.28	0.34	2.00	0.72	1.33	0.38	2.36	1.93	1.97	0.42	1.36	0.31	1.68	1.15	2.07	0.59
	MCP	1.33	0.30	1.36	0.33	1.43	0.91	1.92	0.74	1.32	0.35	2.60	2.07	1.94	0.39	1.39	0.37	2.02	1.76	2.13	0.62
	GBM	13.34	4.78	12.06	3.79	9.56	2.38	3.56	0.95	12.43	3.65	9.34	2.48	4.06	1.35	11.74	2.83	8.94	2.80	3.42	0.84
	RF	15.01	3.80	12.93	3.20	9.47	1.82	3.16	0.77	13.32	3.61	9.73	2.12	4.25	1.42	12.51	2.81	9.31	2.44	3.41	0.93
	SVM	18.18	4.06	15.75	3.21	11.37	2.32	4.12	1.71	17.60	3.63	15.29	2.61	12.26	2.68	16.97	3.33	13.97	2.73	7.66	1.95
3	Lasso	29.59	14.14	32.13	18.61	29.27	13.88	21.48	7.12	36.38	27.60	50.59	20.76	23.12	6.66	37.10	23.41	41.95	17.77	20.23	5.78
	E-net	36.11	21.63	37.22	20.50	33.26	15.88	22.04	7.40	44.04	33.05	55.61	20.21	24.65	7.24	44.32	25.21	47.57	17.17	21.58	6.23
	SCAD	11.81	2.53	12.32	3.90	11.26	2.58	20.18	8.37	12.23	3.32	20.29	17.06	17.45	3.60	12.99	7.60	15.06	12.71	18.63	5.29
	MCP	11.99	2.71	12.38	3.71	11.33	2.54	19.00	7.35	12.07	3.38	24.41	19.61	17.54	3.77	13.03	7.45	15.20	11.86	19.35	5.96
	GBM	117.20	39.53	109.72	31.28	81.61	22.71	31.49	8.64	110.46	31.26	84.20	22.66	35.56	10.33	104.68	28.43	75.11	18.49	30.93	9.38
	RF	135.18	34.03	116.37	27.61	82.55	19.63	27.53	7.30	119.28	31.01	87.64	19.98	37.79	12.42	112.50	27.47	79.84	19.08	29.90	9.42
	SVM	163.73	36.63	145.13	30.24	101.39	22.45	35.85	14.32	157.70	33.62	137.34	23.79	109.13	24.94	149.19	30.47	125.37	26.07	66.95	15.98
6	Lasso	118.35	56.56	128.51	74.46	117.09	55.54	85.93	28.47	149.85	112.89	202.37	83.04	92.50	26.63	148.40	93.64	167.81	71.09	80.90	23.12
	E-net	144.43	86.54	148.87	82.01	133.02	63.52	88.15	29.61	178.74	126.56	222.45	80.86	98.59	28.97	177.28	100.85	190.29	68.67	86.33	24.92
	SCAD	47.25	10.14	49.26	15.59	45.05	10.33	80.73	33.47	48.21	13.95	81.17	68.26	69.82	14.39	51.96	30.40	60.25	50.84	74.52	21.16
	MCP	47.96	10.83	49.51	14.84	45.32	10.14	75.98	29.39	49.06	17.54	97.64	78.45	70.14	15.10	52.12	29.79	60.81	47.45	77.41	23.86
	GBM	462.42	153.08	438.75	132.30	329.77	90.89	125.16	35.21	446.56	135.46	344.83	94.89	143.48	43.70	420.52	121.39	301.37	73.14	124.61	35.86
	RF	540.97	136.58	466.03	111.18	330.46	78.76	110.13	29.18	474.40	128.25	351.13	80.23	151.07	49.86	450.25	110.50	319.58	77.19	119.52	37.83
	SVM	655.71	146.59	581.13	117.87	406.59	89.66	143.00	57.08	629.70	132.87	549.67	94.82	437.80	101.11	598.30	121.48	502.02	104.82	267.80	63.93

References

[1] Akaike, H. Information theory and an extension of the maximum likelihood principle. In *Selected papers of hirotugu akaike*, pages 199–213. Springer, 1998.

[2] Breheny. Adaptive lasso, mcp, and scad. URL: https://myweb.uiowa.edu/pbreheny/7600/s16/notes/2-29.pdf, 2016.

[3] Breheny, P. and Huang, J. Coordinate descent algorithms for nonconvex penalized regression, with applications to biological feature selection. *Annals of Applied Statistics*, 5(1):232–253, 2011.

[4] Breiman, L. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

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

[6] Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[7] Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., Li, M., Xie, J., Lin, M., Geng, Y., and Li, Y. *xgboost: Extreme Gradient Boosting*, 2021. R package version 1.4.1.1.

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

[9] Ding, C. and Peng, H. Minimum redundancy feature selection from microarray gene expression data. *Journal of bioinformatics and computational biology*, 3(02):185–205, 2005.

[10] Donoho, D. L. and Johnstone, J. M. Ideal spatial adaptation by wavelet shrinkage. *biometrika*, 81(3):425–455, 1994.

[11] Drucker, H., Burges, C. J., Kaufman, L., Smola, A., Vapnik, V., et al. Support vector regression machines. *Advances in neural information processing systems*, 9:155–161, 1997.

[12] Efron, B. and Tibshirani, R. J. *An introduction to the bootstrap*. CRC press, 1994.

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

[14] Friedman, J., Hastie, T., and Tibshirani, R. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

[15] Friedman, J., Hastie, T., Tibshirani, R., et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[16] Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[17] Hoerl, A. E. and Kennard, R. W. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

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

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

[20] Liu, X.-Y., Wu, S.-B., Zeng, W.-Q., Yuan, Z.-J., and Xu, H.-B. Logsum+ l 2 penalized logistic regression model for biomarker selection and cancer classification. *Scientific Reports*, 10(1):1–16, 2020.

[21] Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., and Leisch, F. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2021. R package version 1.7-7.

[22] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2021.

[23] Sánchez-Maroño, N., Alonso-Betanzos, A., and Tombilla-Sanromán, M. Filter methods for feature selection–a comparative study. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 178–187. Springer, 2007.

[24] Schapire, R. E. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[25] Schwarz, G. Estimating the dimension of a model. *The annals of statistics*, pages 461–464, 1978.

[26] Tibshirani, R. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[27] Venables, W. N. and Ripley, B. D. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0.

[28] Wright, M. N. and Ziegler, A. ranger: A fast implementation of random forests for high dimensional data in C++ and R. *Journal of Statistical Software*, 77(1):1–17, 2017.

[29] Zhang, C.-H. et al. Nearly unbiased variable selection under minimax concave penalty. *The Annals of statistics*, 38(2):894–942, 2010.

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

[31] Zou, H. and Hastie, T. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005.