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 13, 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 and classification machine learning models are not designed to handle high dimensional data or conduct variable selection. In this paper, we investigated the use of penalized regression methods instead of, or in conjunction with, the traditional machine learning methods. We focused on lasso, ridge, elastic net, SCAD, MCP, and adaptive versions of lasso, ridge, and elastic net models. For traditional machine learning models, we focused on random forest models, gradient boosting models in the form of XGBoost, and support vector machines. These models were evaluated using factorial design methods for Monte Carlo simulations under various data environments. Tests were conducted for 270 environments, with factors being the number of predictors, number of samples, signal to noise ratio, covariance matrix, and correlation strength. This served to identify the strengths and weaknesses of different penalization techniques in different environments. We also compared different models using empirical datasets to test their viability in real-world scenarios. Additionally, we considered penalization methods outlined earlier in logistic regression models for classifying data. These results were compared to random forest, gradient boosting, and support vector machine classification models using both Monte Carlo data generation methods and empirical data. For regression, we evaluated the models using the test mean squared error and variable selection accuracy; for classification, we considered test prediction accuracy and variable selection accuracy. We found that for both regression and classification, penalized regression models outperformed more traditional machine learning algorithms in most high-dimensional situations or in situations with a low number of data observations. By comparing traditional machine learning methods with penalized regression, we hope to expand the scope of machine learning methods for big data to include the various penalized regression techniques we tested. Additionally, we hope 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.

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

1 Introduction

[Better intro paragraph]

Typically, data sets are represented as a table of values. Most columns represent **predictors** (also called variables, attributes, or features), while the rows represent **observations** (also called instances). The value in the i-th row and j-th column represents the value for predictor j in observation i. At least one column is designated as a **response**, which is assumed to be related to some of the predictors in some way. Machine learning models attempt to predict the value of this response from the values of the predictors.

Let n be the number of observations for a data set, and let p be the number of predictors. In most situations, the number of observations greatly exceeds the number of predictors. However, as data collection becomes easier and as statistical modeling techniques are introduced to new disciplines, situations can arise where there are more predictors than observations. For instance, in the field of genomics, there may be thousands of genes that could cause a disease and only a few samples to train from.

In situations where there are more predictors than observations, many traditional machine learning techniques fail to give good predictions. The large number of predictors makes it easy for such models to **overfit**, meaning that the models make good predictors from the data used to train the model, but perform badly when given new data.

To resolve this "large p, little n" problem, many algorithms have been introduced to address situations where there are more predictors than observations. Many, but not all, of these techniques use **variable selection**, meaning that they select the predictors that are most correlated with the response. By ignoring predictors that are not strongly related to the response, the negative consequences of overfitting can be greatly reduced.

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. In addition, we studied penalized regression models such as ridge regression, LASSO, elastic-net, adaptive LASSO, SCAD, and MCP. Models were trained and evaluated using both Monte Carlo simulations and empirical genomic data.

1.1 Background

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

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

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

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

where $\beta_0, \beta_1, \beta_2, \dots, \beta_p$ 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**

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

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.

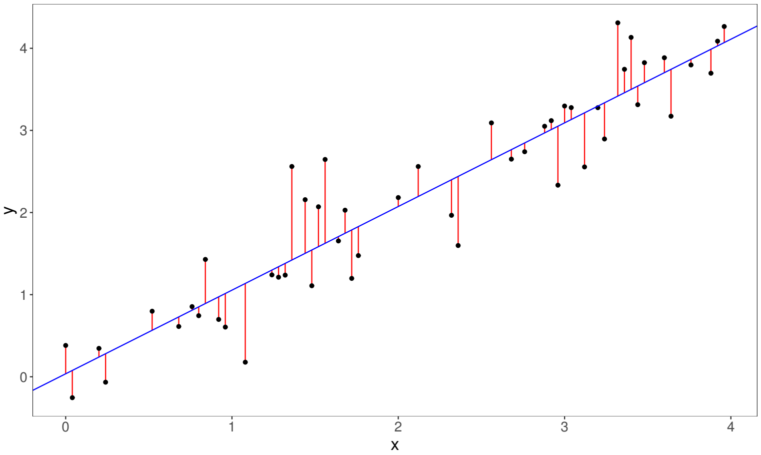


Figure 1: Ordinary least squares fitting with one predictor using simulated data. The blue line represents the line found by ordinary least squares, and the red line segments are the residuals.

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

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

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

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

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 $\hat{\beta}^{\mathrm{OLS}}$ 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 [8] 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. **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). 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 [7]. The Bayesian Information Criterion is

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

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 [7]. 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

$$\hat{\beta} = \arg\min_{\beta} \left\{ \sum_{i=1}^{N} (y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})) + \lambda \sum_{j=1}^{p} P(\beta_j) \right\} \tag{9}$$

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

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

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 9, there is a value of t from 10 that gives equivalent coefficient estimates [7]. Most algorithms to fit penalized regression models use the form given in Equation 9, but the form in Equation 10 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.

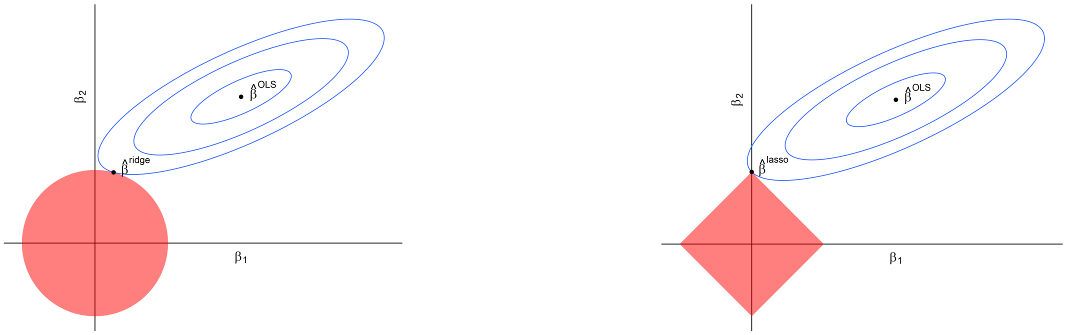
Ridge regression is a penalized regression model that uses the penalty function $P(\beta) = \beta^2$ [5]. One advantage of ridge regression is that it can handle highly correlated data better than ordinary least squares. When predictors are correlated with each other, some algorithms have a hard time 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** (LASSO) is a shrinkage method with a very similar form to ridge regression [9, 7, 6]. 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 10 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.



(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.

Elastic-net regression uses both the ridge penalty and the LASSO penalty at the same time [11]. Elastic-net solves the optimization problem

$$\hat{\beta}^{\text{E-net}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right) + \lambda_2 \sum_{j=1}^{p} \beta_j^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j| \right\} \tag{11}$$

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 [11]. 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 [11].

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 [3]. 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

$$\frac{dP}{d\beta} = \cdot\text{sign}(\beta) \left[I(|\beta| < \lambda) + \frac{\max(a\lambda - |\beta|, 0)}{(a-1)\lambda} I(|\beta| > \lambda) \right] \tag{12}$$

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 12 is

$$\frac{dP}{d\beta} = \begin{cases} 1, & |\beta| \leq \lambda \\ \frac{a\lambda - |\beta|}{(a-1)\lambda}, & \lambda < |\beta| < a\lambda \\ 0, & a\lambda < |\beta| \end{cases} \tag{13}$$

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 [1] and choosing $P(\beta) = 0$, we see that

$$P(\beta) = \begin{cases} |\beta|, & |\beta| \leq \lambda \\ \frac{2a\lambda|\beta| - \beta^2 - \lambda^2}{2(a-1)\lambda}, & \lambda < |\beta| < a\lambda \\ \frac{\lambda(a+1)}{2}, & a\lambda < |\beta| \end{cases} \tag{14}$$

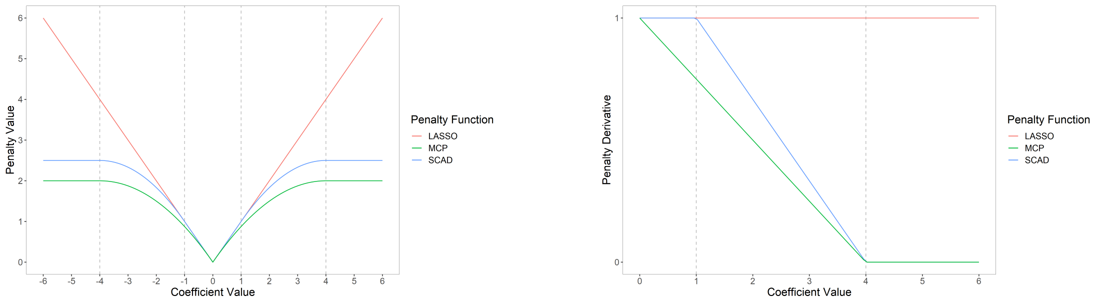
The **minimax concave penalty** (MCP) method is very similar to SCAD [10, 1]. Both methods are used to avoid the high bias caused by the LASSO method. MCP uses a penalty function that satisfies

$$\frac{dP}{d\beta} = \begin{cases} \text{sign}(\beta)\left(1 - \frac{|\beta|}{a\lambda}\right), & |\beta| \leq a\lambda \\ 0, & a\lambda < |\beta| \end{cases} \tag{15}$$

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

$$P(\beta) = \begin{cases} |\beta| - \frac{\beta^2}{2a\lambda}, & |\beta| \leq a\lambda \\ \frac{1}{2}a\lambda, & a\lambda < |\beta| \end{cases} \tag{16}$$

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 $\lambda = 2$ 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 value for each coefficient is independent of the other coefficients; in other words, we can compute each coefficient value as if it were the only predictor in the data set.

We can apply a **thresholding function** to each predictor to get the coefficient estimate for that predictor. For LASSO, elastic-net, SCAD, and MCP, a closed-form for this thresholding function exists [9, 3, 11, 10].

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 $\lambda = 2$. 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.

Another feature of SCAD and MCP is that they

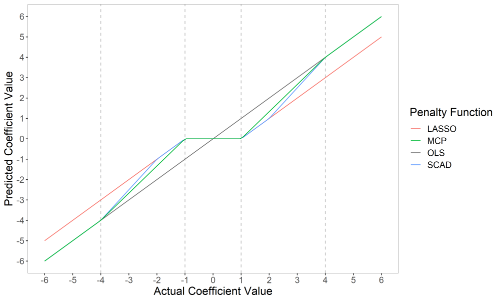


Figure 4: Thresholding function for LASSO, SCAD, and MCP when $\lambda = 2$ and $a = 3$. In an orthonormal design, this function can be applied to the true coefficient values to get the predicted coefficient values. The dashed vertical lines are the knots for SCAD and MCP.

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**. In this context, the term **oracle** comes from a paper by Donohe and Johnstone about spatially variable estimators [2]. They consider an ideal situation where a model could be fitted with the aid of an oracle. This oracle could give information about the underlying behavior of the function f relating a response variable to the predictors without giving the function f itself.

In the context of linear regression, an oracle estimator would know which predictors are truly correlated with the response. With the aid of an oracle, one could then fit an ordinary least squares model using only the coefficients that are actually non-zero and ignoring the predictors that are actually zero. Fan and Li in [3] and Zhang in [10] prove that SCAD and MCP both have oracle-like properties. Under certain conditions, 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.

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 [6].

Random Forests solve the issue of high variance by aggregating the predictions of many independent decision trees. 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 generate a tree within a random forest model, first select a random sample of the n observations with replacement. This is a process called **bootstrapping**. 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. The number of trees generated is another hyperparameter; 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.

Boosting is the technique of sequentially improving a weak learner until it becomes a strong learner. A **gradient boosting model** (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. 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, variable importance and pruning can be used as a sort of pseudo-variable selection method to lower complexity and prevent over-fitting. However, 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.

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. For these reasons, XGBoost is often preferred over standard gradient boosting models and is very commonly used in many machine learning applications.

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.

Support vector machines are versatile statistical models that can be used for both regression and classification. 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. 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

To test and evaluate each of the models discussed above, we considered both Monte Carlo simulations and empirical data. Monte Carlo simulations allow us to test the models under different environments to get a comprehensive idea of how each model performs. Furthermore, by repeating each simulation many times, our results will be more consistent than if we just relied on one data set to test each model.

For our empirical data analysis, we used

2.1 Models

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:

- The true relationship between the predictor variables and the response is known.
- Simulations can be iterated many times, giving sturdier results about the effectiveness of each model.
- We have full control over factors such as the number of predictors and the amount of correlation between predictors.

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

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \epsilon \tag{17}$$

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.

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. For our simulations, we used $\beta_0 = 1$, $\beta_1 = 2$, $\beta_2 = -2$, $\beta_5 = 0.5$ and $\beta_6 = 3$; the remaining coefficient values were set to 0.

Next, we generated $\mathbf{X}$, a $n \times (p+1)$ matrix of predictor variables. The first column contains 1 in all of its entries; this corresponds to the intercept of our linear model. Column i of $\mathbf{X}$ contains the variable values for predictor x_{i-1}, for $1 \leq i \leq p$. These values 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.

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

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{e} \tag{18}$$

We used a **factorial design** for our simulations. This means that we considered several factors that affect the data generation process, 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.

- n, the number of observations: 50, 200, 1000.
- p, the number of predictors: 10, 100, 2000.
- σ, the standard deviation of the random error: 1, 3, 6.
- The correlation matrix structure: independent, symmetric compound, autoregressive, blockwise.
- ρ, the correlation between predictors: 0.2, 0.5, 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 use the correlation value ρ, we actually only used 270 combinations. 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

$$\mathbf{\Sigma} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{19}$$

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

$$\mathbf{\Sigma} = \begin{bmatrix} 1 & \rho & \cdots & \rho \\ \rho & 1 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & 1 \end{bmatrix} \tag{20}$$

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

$$\mathbf{\Sigma} = \begin{bmatrix} 1 & \rho & \cdots & \rho^{p-1} \\ \rho & 1 & \cdots & \rho^{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho^{p-1} & \rho^{p-2} & \cdots & 1 \end{bmatrix} \tag{21}$$

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

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{B}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{B}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{B}_k \end{bmatrix} \tag{22}$$

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 20. 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 $p = 2000$.

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`	Ordinary least squares	4.1.0
`MASS`	Forward and backward selection	7.3-54
`glmnet`	Ridge, lasso, elastic-net	4.1-1
`gcdnet`	Adaptive ridge, adaptive lasso, adaptive elastic-net	1.0.5
`ncvreg`	SCAD and MCP	3.13.0
`xgboost`	Gradient boosting	1.4.1.1
`ranger`	Random forest	0.12.1
`e1071`	Support vector machine	1.7-7

For ridge, lasso, and elastic-net regression using `glmnet`, we used the `cv.glmnet` function. This function uses cross-validation to determine the value of λ that minimizes the cross-validation error. We used ten folds with `cv.glmnet`. Using cross-validation can help generate a model that has a good test performance. For elastic-net regression, we used the hyperparameter $\alpha = 0.8$. This means that the elastic-net model is more similar to lasso (where $\alpha = 1$) than ridge (where $\alpha = 0$). The remaining hyperparameters were given their default values.

We used the `cv.gcdnet` function from the `gcdnet` library for the adaptive versions of ridge, lasso, and elastic-net. Again, ten folds were used for the cross-validation, and all hyperparameters were given their default values.

For SCAD and MCP models, we used the `cv.ncvreg` function from the `ncvreg` library. 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).

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 those hyperparameters. 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. The cross-validation function used five folds.

For random forests using `ranger`, 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 cross-validation function used five folds.

Finally, for support vector machines using `e1071`, we varied ϵ (TODO: explain this) and the cost function (0.5, 1, and 2).

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 5. 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.

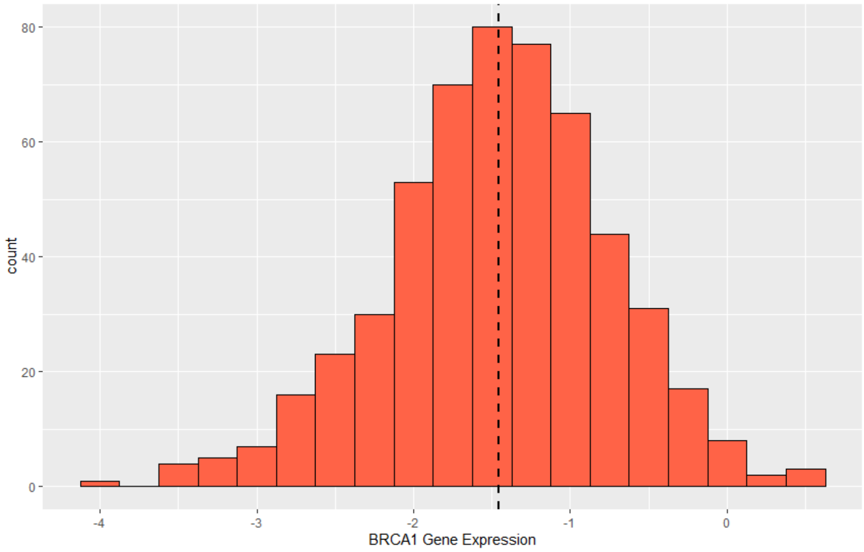


Figure 5: Distribution of BRCA1 gene expression levels

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.

3 Results

3.1 Monte Carlo Simulations

3.2 Empirical Data

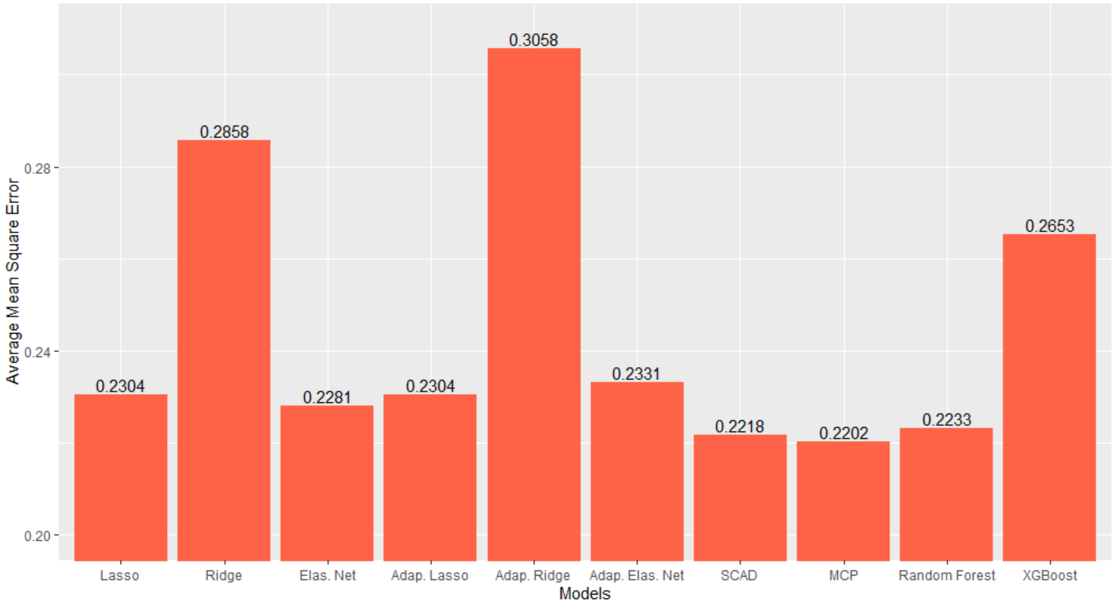


Figure 6: Average Mean Square Error for Empirical Data Models

4 Discussion

4.1 Findings

4.2 Contributions

4.3 Future Work

		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	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251
	AIC back.	1.218	0.249	1.206	0.240	1.231	0.262	1.207	0.254	1.216	0.235	1.209	0.237	1.235	0.253	1.221	0.256	1.223	0.249	1.215	0.243
	BIC back.	1.164	0.243	1.168	0.261	1.196	0.259	1.170	0.249	1.162	0.245	1.174	0.232	1.186	0.247	1.191	0.267	1.199	0.248	1.180	0.248
	AIC step. back.	1.218	0.249	1.207	0.239	1.231	0.262	1.207	0.254	1.216	0.235	1.209	0.237	1.238	0.256	1.221	0.256	1.223	0.249	1.216	0.243
	BIC step. back.	1.164	0.243	1.168	0.261	1.196	0.259	1.169	0.249	1.163	0.245	1.174	0.232	1.188	0.248	1.191	0.267	1.199	0.248	1.180	0.248
	AIC for.	1.209	0.247	1.197	0.243	1.231	0.263	1.195	0.245	1.201	0.233	1.204	0.243	1.248	0.321	1.223	0.260	1.221	0.248	1.225	0.292
	BIC for.	1.159	0.252	1.163	0.261	1.195	0.255	1.196	0.308	1.156	0.241	1.171	0.236	1.396	0.402	1.188	0.263	1.199	0.244	1.230	0.368
	AIC step. for.	1.209	0.247	1.197	0.243	1.231	0.263	1.196	0.246	1.201	0.233	1.202	0.243	1.244	0.321	1.223	0.260	1.221	0.248	1.225	0.292
	BIC step. for.	1.159	0.252	1.163	0.261	1.195	0.255	1.196	0.308	1.156	0.241	1.171	0.236	1.396	0.402	1.188	0.263	1.199	0.244	1.230	0.368
	Ridge	1.586	0.343	1.593	0.408	1.695	0.499	1.843	0.385	1.643	0.333	1.749	0.398	1.866	0.504	1.663	0.351	1.665	0.424	1.801	0.406
	Lasso	1.388	0.322	1.390	0.397	1.402	0.349	1.395	0.362	1.399	0.308	1.410	0.319	1.387	0.418	1.418	0.326	1.360	0.301	1.395	0.383
	E-net	1.385	0.319	1.391	0.386	1.413	0.373	1.407	0.331	1.408	0.313	1.423	0.306	1.390	0.413	1.431	0.321	1.371	0.317	1.411	0.412
	Adap. ridge	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.276	0.251	1.275	0.251	1.276	0.251	1.276	0.251	1.276	0.251
	Adap. lasso	1.382	0.334	1.395	0.406	1.422	0.380	1.399	0.333	1.399	0.315	1.417	0.327	1.382	0.417	1.424	0.327	1.373	0.324	1.389	0.387
	Adap e-net	1.372	0.320	1.390	0.405	1.408	0.373	1.397	0.345	1.390	0.300	1.414	0.312	1.380	0.403	1.427	0.334	1.369	0.316	1.385	0.401
	SCAD	1.200	0.249	1.207	0.262	1.230	0.256	1.185	0.257	1.201	0.274	1.196	0.249	1.201	0.249	1.205	0.254	1.219	0.254	1.174	0.262
	MCP	1.184	0.244	1.193	0.262	1.219	0.257	1.172	0.251	1.205	0.269	1.204	0.253	1.205	0.268	1.196	0.247	1.217	0.237	1.174	0.260
	GB	3.728	1.140	3.696	1.221	3.823	1.140	2.926	0.696	4.023	1.169	3.848	1.154	2.934	0.815	3.693	1.197	3.747	1.015	2.907	0.727
	RF	6.899	1.779	6.581	1.856	5.279	1.275	2.592	0.534	6.952	1.671	5.883	1.390	2.779	0.658	6.807	1.848	5.704	1.279	2.867	0.683
	SVM	5.770	1.710	5.321	1.816	4.345	1.686	2.991	1.473	5.567	1.619	5.084	1.606	3.257	1.327	5.438	1.818	4.451	1.332	3.194	1.313
3	OLS	11.483	2.258	11.483	2.258	11.483	2.258	11.483	2.258	11.483	2.258	11.483	2.258	11.483	2.258	11.483	2.258	11.483	2.258	11.483	2.258
	AIC back.	10.964	2.239	10.932	2.448	11.113	2.243	10.911	2.117	10.918	2.153	10.878	2.131	11.088	2.282	11.050	2.499	11.046	2.386	11.131	2.176
	BIC back.	10.472	2.190	10.576	2.501	10.877	2.114	10.648	2.065	10.450	2.087	10.562	2.092	10.762	2.189	10.737	2.573	10.916	2.390	10.713	2.142
	AIC step. back.	10.964	2.239	10.932	2.448	11.095	2.257	10.911	2.117	10.918	2.153	10.885	2.137	11.072	2.249	11.050	2.499	11.067	2.375	11.131	2.176
	BIC step. back.	10.472	2.190	10.576	2.501	10.877	2.114	10.649	2.066	10.450	2.087	10.568	2.090	10.762	2.189	10.737	2.573	10.916	2.390	10.713	2.142
	AIC for.	10.882	2.223	10.874	2.487	11.018	2.153	10.903	2.093	10.831	2.217	10.833	2.183	11.154	2.631	11.010	2.523	10.998	2.333	11.025	2.312
	BIC for.	10.430	2.272	10.569	2.438	10.822	2.084	10.659	2.160	10.434	2.102	10.542	2.120	12.520	3.689	10.759	2.600	10.825	2.313	11.147	2.973
	AIC step. for.	10.882	2.223	10.874	2.487	11.015	2.151	10.903	2.093	10.831	2.217	10.816	2.190	11.148	2.678	11.010	2.523	11.022	2.355	11.032	2.309
	BIC step. for.	10.430	2.272	10.569	2.438	10.822	2.084	10.674	2.187	10.434	2.102	10.537	2.125	12.520	3.690	10.759	2.600	10.812	2.296	11.144	2.978
	Ridge	14.277	3.091	15.172	4.146	14.822	3.577	16.744	3.571	14.733	2.883	15.737	3.583	16.693	4.239	14.737	4.027	15.851	3.455	16.429	4.332
	Lasso	12.491	2.899	12.648	3.262	12.484	2.851	12.557	3.285	12.500	2.852	12.693	2.869	12.366	3.500	12.618	3.418	12.834	2.919	12.849	3.744
	E-net	12.461	2.874	12.868	3.302	12.424	2.817	12.677	3.273	12.664	2.936	12.810	2.755	12.620	3.569	12.695	3.302	12.918	2.802	12.808	3.678
	Adap. ridge	11.482	2.257	11.483	2.258	11.482	2.257	11.482	2.256	11.482	2.257	11.483	2.257	11.477	2.256	11.482	2.258	11.482	2.257	11.481	2.257
	Adap. lasso	12.442	3.003	12.793	3.416	12.469	2.929	12.659	3.286	12.592	2.977	12.753	2.945	12.457	3.749	12.743	3.599	12.906	2.843	12.787	3.655
	Adap e-net	12.344	2.884	12.689	3.276	12.423	2.875	12.742	3.309	12.578	2.802	12.728	2.804	12.487	3.549	12.708	3.515	12.822	2.821	12.883	3.698
	SCAD	10.804	2.240	10.827	2.465	11.096	2.163	10.872	2.215	10.718	2.301	10.768	2.245	10.749	2.297	10.804	2.313	10.912	2.156	10.713	2.255
	MCP	10.657	2.192	10.745	2.484	10.982	2.177	10.802	2.341	10.680	2.327	10.838	2.281	10.848	2.263	10.752	2.315	10.824	2.137	10.706	2.249
	GB	35.185	12.026	35.724	10.740	34.796	11.994	25.465	6.593	34.832	10.704	33.856	8.918	26.327	7.057	34.641	10.940	34.518	10.047	26.503	7.561
	RF	61.937	15.929	58.397	14.389	47.302	11.784	23.032	4.826	61.755	15.868	52.935	12.555	24.747	6.186	58.857	15.627	52.980	13.097	25.289	6.085
	SVM	51.927	15.388	49.173	15.338	39.745	13.980	26.920	13.226	50.239	15.211	45.480	14.164	29.461	11.979	47.105	15.030	41.676	14.355	29.467	12.088
6	OLS	45.930	9.030	45.930	9.030	45.930	9.030	45.930	9.030	45.930	9.030	45.930	9.030	45.930	9.030	45.930	9.030	45.930	9.030	45.930	9.030
	AIC back.	43.855	8.957	43.726	9.793	44.453	8.971	43.642	8.470	43.673	8.613	43.511	8.525	44.354	9.128	44.198	9.994	44.186	9.546	44.524	8.705
	BIC back.	41.889	8.759	42.304	10.004	43.509	8.456	42.592	8.259	41.802	8.347	42.247	8.367	43.046	8.756	42.950	10.291	43.663	9.560	42.854	8.570
	AIC step. back.	43.855	8.957	43.726	9.793	44.381	9.026	43.642	8.470	43.673	8.613	43.541	8.548	44.288	8.998	44.198	9.994	44.267	9.500	44.524	8.705
	BIC step. back.	41.889	8.759	42.304	10.004	43.509	8.456	42.596	8.265	41.802	8.347	42.272	8.359	43.046	8.756	42.950	10.291	43.663	9.560	42.854	8.570
	AIC for.	43.528	8.892	43.497	9.947	44.071	8.611	43.611	8.374	43.325	8.867	43.334	8.731	44.618	10.525	44.039	10.092	43.992	9.332	44.101	9.247
	BIC for.	41.719	9.087	42.276	9.754	43.288	8.336	42.638	8.640	41.735	8.409	42.168	8.481	50.079	14.758	43.036	10.400	43.298	9.251	44.588	11.894
	AIC step. for.	43.528	8.892	43.497	9.947	44.059	8.603	43.611	8.374	43.325	8.867	43.266	8.760	44.592	10.714	44.039	10.092	44.088	9.421	44.127	9.238
	BIC step. for.	41.719	9.087	42.276	9.754	43.288	8.336	42.697	8.747	41.735	8.409	42.147	8.498	50.079	14.761	43.036	10.400	43.246	9.183	44.576	11.910
	Ridge	57.109	12.364	60.687	16.583	59.289	14.307	66.974	14.284	58.930	11.533	62.949	14.330	66.771	16.957	58.947	16.108	63.405	13.819	65.715	17.329
	Lasso	49.962	11.595	50.593	13.047	49.936	11.403	50.230	13.138	50.002	11.410	50.772	11.476	49.464	13.999	50.472	13.673	51.335	11.675	51.397	14.976
	E-net	49.845	11.496	51.472	13.209	49.694	11.269	50.708	13.092	50.656	11.743	51.241	11.019	50.481	14.278	50.779	13.210	51.672	11.210	51.231	14.713
	Adap. ridge	45.929	9.029	45.931	9.031	45.927	9.028	45.928	9.024	45.929	9.029	45.931	9.030	45.910	9.026	45.930	9.030	45.929	9.029	45.924	9.027
	Adap. lasso	49.766	12.013	51.171	13.663	49.877	11.717	50.634	13.144	50.368	11.908	51.011	11.779	49.827	14.997	50.971	14.396	51.624	11.372	51.147	14.622
	Adap e-net	49.376	11.536	50.756	13.102	49.693	11.501	50.969	13.234	50.312	11.208	50.913	11.215	49.949	14.195	50.830	14.061	51.289	11.285	51.532	14.793
	SCAD	43.214	8.960	43.307	9.860	44.382	8.653	43.487	8.861	42.873	9.203	43.071	8.978	42.996	9.187	43.217	9.251	43.650	8.623	42.850	9.022
	MCP	42.628	8.769	42.979	9.937	43.929	8.709	43.210	9.365	42.719	9.307	43.351	9.125	43.394	9.052	43.009	9.262	43.297	8.547	42.823	8.998
	GB	135.812	40.816	140.921	40.573	135.510	43.275	102.763	23.849	137.560	43.883	137.808	40.068	105.821	29.447	141.906	45.697	137.212	37.811	105.834	28.599
	RF	248.001	64.295	233.826	57.679	188.905	46.879	92.071	19.257	246.706	63.527	211.647	50.441	99.022	24.832	235.415	62.415	211.856	52.405	101.307	24.452
	SVM	207.957	61.720	196.815	61.630	159.388	56.783	106.807	53.412	200.958	60.842	182.386	56.868	116.606	48.592	188.419	60.121	166.756	57.132	118.048	50.595

References

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

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

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

[4] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

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

[6] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[7] Witten D. Hastie T. Tibshirani R. James, G. *ISLR: Data for an Introduction to Statistical Learning with Applications in R*, 2017. R package version 1.2.

[8] Xiao-Ying Liu, Sheng-Bing Wu, Wen-Quan Zeng, Zhan-Jiang Yuan, and Hong-Bo Xu. Logsum+ l 2 penalized logistic regression model for biomarker selection and cancer classification. *Scientific Reports*, 10(1):1–16, 2020.

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

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

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