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 9, 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 $X_1, X_2, \ldots, X_p$ 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.

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.

1.2 Ordinary Least Squares

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.

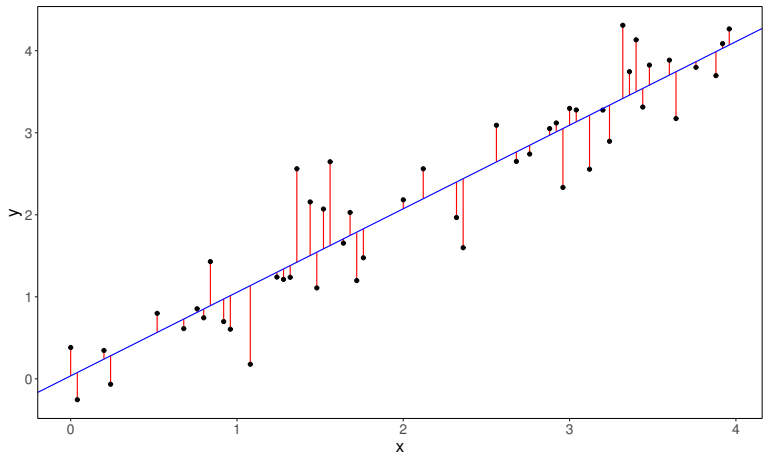


Figure 1: Ordinary least squares fitting with one predictor using simulated data. The blue line represents the line found by ordinary least squares, and the red line segments are the residuals.

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 extra value of 1 in the first entry). Then $\mathbf{X}\beta$ is a vector of the estimated response values. 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

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

where $(\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta)$ is the same residual sum of squares seen in Equation 4. From [3], 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 truly is 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. In most applications, there are more observations than predictors, and so ordinary least squares works well. However, as p gets closer to n, a model fitted with ordinary least squares will begin to **overfit**. Because the response variable Y depends on a random error ϵ, a model that fits too closely to the training data will perform poorly when given new data to test on.

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.

1.3 Literature Review

Liu et. al. in [7] describe three types of variable selection algorithms: filter methods, which evaluate the ability for each variable to predict the response individually; wrapper methods, which find a subset of predictors highly correlated to the response using feature assessment metrics; and embedded methods, which perform variable selection during the model training process. This paper focuses on the second and third types.

Subset selection methods attempt to find a subset of the predictors that are most correlated with the response variable. 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 unimportant predictors. Another benefit of subset selection is that it creates a more interpretable model, since the fitted model only depends on a few of the predictors.

Best subset selection is a subset selection method that fits considers every possible combination of predictors. For every possible subset size k between 0 and p, best subset selection will fit the $\binom{p}{k}$ possible models using k predictors and chooses the best model. Then, a final model can be selected from the $(p+1)$ remaining models. The best model is selected using a metric such as the residual sum of squares. 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 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 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. Although forward and backward selection are not guaranteed to 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 for predictors to

be added and removed in the same algorithm. **Forward stepwise selection** begins with a model 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. These techniques take longer to run than ordinary forward and backward selection, but they are more likely to find the best possible model.

[Discuss AIC/BIC]

Ridge regression helps to solve multicollinearity in predictors while also minimizing insignificant predictors [4]. While it does not minimize these insignificant predictors completely to 0 and thus cannot be considered a variable selection method, it still proves very useful in large datasets.

Ridge regression works by minimizing Residual Sum Squared (RSS) plus a penalty as seen in Equation 7. λ is a tuning parameter and can be used to determine how much of an effect the penalty has on the regression. if $\lambda = 0$, then the regression acts exactly like ordinary least squares regression, but if $\lambda \to \infty$, then $\beta_j \to 0$ and the regression line will be a horizontal line at the intercept, β_0.

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right\} \tag{7}$$

An alternative way to express ridge regression is with the equation

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq t \tag{8}$$

for some tuning parameter t.

The **least absolute shrinkage and selection operation**, often referred to as LASSO, is a shrinkage method with a very similar form to lasso regression [8, 5, 6]. The coefficient estimates satisfy

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right) + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{9}$$

If $\lambda = 0$, then the lasso model is equivalent to the ordinary least squares model; if $\lambda \to \infty$, then the coefficients for all predictors will be set to 0. An equivalently way to define lasso regression is by

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq t \tag{10}$$

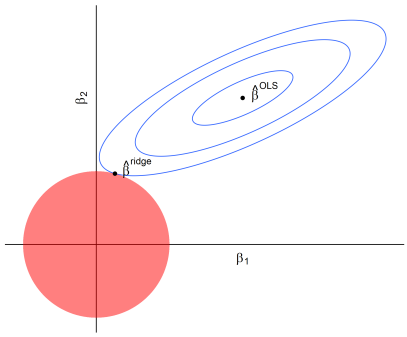
where t is a tuning parameter.

One useful property of the lasso method is that it can perform variable selection by setting some coefficients to zero. To understand why lasso regression can perform variable selection whereas ride regression cannot, consider Figure 2 below. This figure demonstrates the case when $p = 2$ and $t = 1$. The circle in 2a represents the condition $\beta_1^2 + \beta_2^2 < 1$ for ridge regression, while the red diamond in Figure 2b represents the condition $|\beta_1| + |\beta_2| < t$ for lasso regression. The blue curves represent contours of the residual sum of squares for values of β_1 and β_2. The black point in the center of these curves is where the RSS is minimized; this represents the values of β_1 and β_2 that would be selected by ordinary least squares.

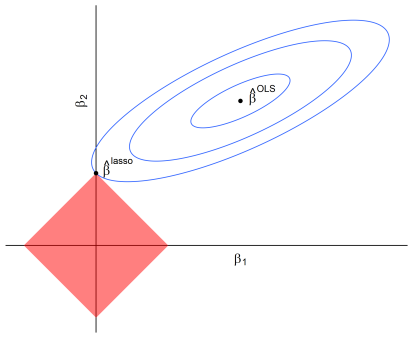
In 2a, the intersection of the black curve and the red diamond represents the parameter values chosen by ridge regression; this point minimizes the RSS under the condition $\beta_1^2 + \beta_2^2 \leq 1$. Because the red region is a circle, this intersection cannot occur exactly at $\beta_1 = 0$; hence, the ridge method cannot remove the predictor β_1. On the other hand, the square shape of the constrained region for lasso regression can perform variable selection because the intersection occurs at one of the axes.

Figure 2: RSS contours and penalty bounds for the ridge and lasso models when $p = 2$ and $t = 1$.

(a) RSS contours and the ridge penalty boundary.



(b) RSS contours and the lasso penalty boundary.



The lasso method is particularly useful in the case where $p > n$ because of its ability to select variables; a model with fewer variables has less variance and is more interpretable. One major downside of lasso regression is that it does not handle multicollinearity as nicely as ridge regression. Another downside of lasso regression is that it does not have a closed-form solution, which can lead to instability in the model.

Elastic-net regression serves as a combination between ridge and lasso regression. It can handle multicollinearity as well as perform variable selection. The coefficients for elastic net regression can be determined by

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

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 $\lambda_1 >> \lambda_2$ or vice versa [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.

Normally in lasso regression, each predictor is weighted the same in the penalty function. **Adaptive lasso regression** is different in that a weight, $\hat{w}_j$ is multiplied to the penalty function. The coefficients for adaptive lasso regression as designed by Zou et. al. [10] can be defined by

$$\hat{\beta}^{\text{adaptive}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right) + \lambda \sum_{j=1}^{p} \hat{w}_j |\beta_j| \right\} \tag{12}$$

where λ is a tuning parameter to be determined later and $\hat{w}_j$ is defined as $\frac{1}{|\hat{\beta}|^{\gamma}}$ with γ being a chosen parameter greater than 0.

Because of the weight that is implemented in adaptive lasso regression, zero-coefficients have a weight that is inflated up to infinity, and thus are punished much more harshly than large coefficients whose weight is much smaller in comparison. This is a similar rationale to SCAD and helps to reduce some of the bias from lasso regression. Bridge regression is the general form of lasso regression from which adaptive lasso originates from. When $\gamma < 1$, bridge regression as shown in Equation 13 is not continuous, which results in model prediction instability. However, adaptive lasso regression is completely continuous and thus has much more consistent coefficients when fitted.

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right) + \lambda \sum_{j=1}^{p} |\beta_j|^{\gamma} \right\} \tag{13}$$

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 [2]. 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.

$$\hat{\beta}^{\text{SCAD}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right) + \lambda \sum_{j=1}^{p} J_a(\beta_j, \lambda) \right\} \tag{14}$$

Here, $J_a(\beta, \lambda)$ is a penalty function that satisfies

$$\frac{dJ_a(\beta,\lambda)}{d\beta} = \lambda \cdot \text{sign}(\beta) \left[I(|\beta| < \lambda) + \frac{(a\lambda - |\beta|)_+}{(a-1)\lambda} I(|\beta| > \lambda) \right] \tag{15}$$

where $\lambda \geq 0$ and $a \geq 2$ are tuning parameters. An equivalent way to write this is

$$\frac{dJ_a(\beta,\lambda)}{d\beta} \begin{cases} \lambda, & |\beta| \leq \lambda \\ \frac{a\lambda - |\beta|}{a-1}, & \lambda < |\beta| < a\lambda \\ 0, & \alpha\lambda < |\beta| \end{cases} \tag{16}$$

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. See Figure 3a for a plot of the SCAD penalty as a function of the coefficient value.

Integrating with respect to β [1], we see that

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

The **minimax concave penalty** (MCP) method is very similar to smoothly clipped absolute deviation [9, 1]. Both methods are used to avoid the high bias caused by the lasso method. MCP uses a penalty function that satisfies

$$\frac{dJ_a(\beta,\lambda)}{d\beta} = \begin{cases} \text{sign}(\beta)\left(\lambda - \frac{|\beta|}{a}\right), & |\beta| \leq a\lambda \\ 0, & a\lambda < |\beta| \end{cases} \tag{18}$$

where $\lambda \geq 0$ and $a > 0$ are tuning parameters. Integrating [1], we see that

$$J_a(\beta,\lambda) = \begin{cases} \lambda|\beta| - \frac{\beta^2}{2a}, & |\beta| \leq a\lambda \\ \frac{1}{2}a\lambda^2, & a\lambda < |\beta| \end{cases} \tag{19}$$

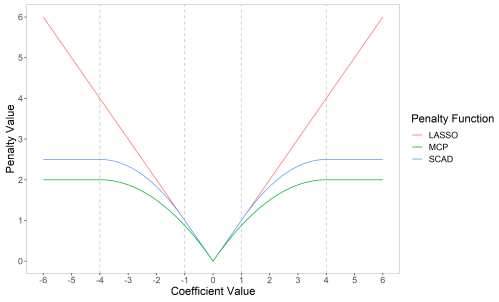
Figure 3 below shows the penalty functions (and their derivatives) for LASSO, SCAD, and MCP as a function of a coefficient value β. We see that LASSO applies a much stronger penalty to large coefficients than SCAD or MCP. Also, note that SCAD starts with a derivative equal to that of the lasso for small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

Now, consider the case where $p = 1$ (there is only one predictor). Figure 4 shows the solutions given when using LASSO, SCAD, and MCP on such a model. The x-axis gives the actual coefficient for the single variable, and the y-axis represents the coefficient estimate produced using each of the algorithms. We used the particular values $\lambda = 2$ and $a = 3$. The gray line is the identity function, which also equals the solution obtained using ordinary least squares.

We see that all three methods set the predicted value to zero when the actual coefficient is small. Also, note that the LASSO method is always off from the identity function when the coefficient is large. On the other hand, SCAD and MCP merge with the identity function when the coefficient is sufficiently large. This shows that both SCAD and MCP can avoid the high bias that LASSO introduces.

Figure 3: Penalty functions for LASSO, SCAD, and MCP, as well as their derivatives. These plots use $\lambda = 2$ and $a = 3$.

(a) Penalty functions for LASSO, SCAD, and MCP



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

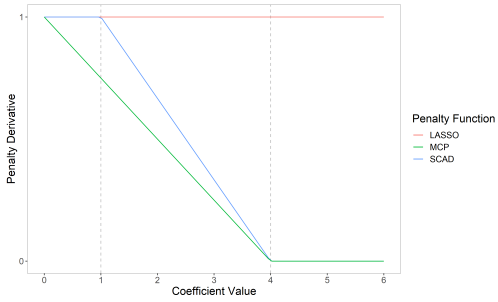
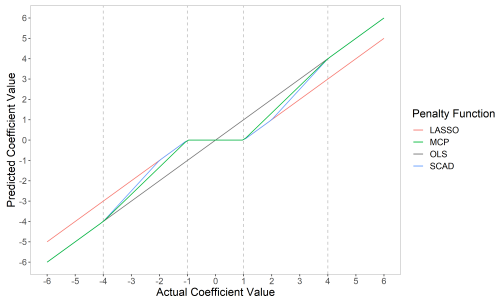


Figure 4: Solutions for LASSO, SCAD, and MCP for a single predictor when $\lambda = 2$, and $a = 3$.



2 Methods

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

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p + \epsilon \tag{20}$$

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.

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

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

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 Σ 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 $i \neq j$.

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.

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. 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`	Stepwise 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

3 Results

3.1 Regression Models

3.2 Classification Models

4 Discussion

4.1 Findings

4.2 Contributions

4.3 Future Work

References

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

[2] 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.

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

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

[5] Gareth James, Daniela Witten, Trevor Hastie, and Rob Tibshirani. *ISLR: Data for an Introduction to Statistical Learning with Applications in R*, 2017. R package version 1.2.

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

[7] 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.

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

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

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

[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.