Variable Selection Techniques

Gabriel Ackall and Connor Shrader

June 10, 2021

1 Introduction

Linear regression is one of the simplest forms of statistical learning models. It assumes that some predictor variable y can be modeled as a linear combination of a set of predictor variables $x_1, x_2, \ldots, x_p$. More precisely, a linear regression model assumes that

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon \tag{1}$$

where $\beta_0, \beta_1, \ldots, \beta_p$ are coefficients and ϵ is a random error with mean zero and variance σ^2. We also assume that this error has no correlation between different observations. When working with real data sets, the coefficient values are usually unknown; the goal of linear regression is to compute coefficient estimates $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$ that are good estimates for the actual coefficients.

1.1 Ordinary Least Squares

Let $\beta = [\beta_0, \beta_1, \ldots, \beta_p]^T$ be a vector of coefficient values, and let $\hat{\beta} = [\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p]^T$ represent a vector of coefficient estimates. Let $\mathbf{X}$ be a $n \times (p+1)$ matrix containing n observations in p variables. In order to account for the constant β_0 term in Equation 1, we include an extra column in $\mathbf{X}$ with each entry equal to 1; the coefficient estimate for this additional "variable" will be $\hat{\beta}_0$. Finally, let $\mathbf{y}$ represent the vector of response values for each observation.

The most common way to compute $\hat{\beta}$ is with ordinary least squares. This method computes $\hat{\beta}$ by minimizing the residual sum of squares:

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

$$= \arg\min_{\beta} \left\{ (\mathbf{y} - \mathbf{X}\beta)^\top (\mathbf{y} - \mathbf{X}\beta)^\top \right\} \tag{3}$$

Ordinary least squares is favored for a few reasons. For one, the solution is very easy to compute; the coefficient estimate is given by

$$\hat{\beta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \tag{4}$$

Another advantage of ordinary least squares comes from the Gauss-Markov Theorem. This theorem states that of all the linear unbiased estimators, ordinary least squares has the lowest variance. Alternative unbiased estimators will necessarily have a larger variance, which can lead to more inaccurate predictions.

1.2 Linear Regression with High Dimensionality Data

If ordinary least squares minimizes variance among all unbiased estimators, why would we use anything else? One issue is that the matrix $\mathbf{X}^\top \mathbf{X}$ is not always invertible. In fact, this matrix is never invertible in the case when $p > n$. Hence, in some situations, we cannot use ordinary least squares.

Even if the matrix is invertible, other methods may lead to better model performance. Despite having the least variance among unbiased estimators, ordinary least squares may still have high variance. Consequently, it may be worthwhile to sacrifice some bias in order to reduce variance.

Many biased linear regression models perform *variable selection*, meaning that some coefficient estimates will be set to zero. In many applications, not all of the predictors will be related to the response, but ordinary least squares still attempts to use every predictor when computing coefficient estimates. Variable selection techniques can identify the predictors most strongly correlated to the data and eliminate the rest. This results in a simpler model that uses p^* of the p available predictors. This not only helps reduce variance, but it also creates more interpretable models.

The rest of this document examines many approaches to fitting linear regression models. These techniques fall under two categories: subset selection and penalized regression. Subset selection algorithms attempt to find a subset of predictors that is most strongly correlated to the response and fits an ordinary least squares model to that subset. Penalized regression techniques punish large coefficient estimates, which encourages models that have smaller coefficient values. Depending on the particular algorithm used, penalized regression can also perform variable selection.

1.3 Alternative Techniques with High Dimensionality Data

In addition to penalized and subset selection linear regression methods used for variable selection, there are other techniques that can achieve similar results. One of these such methods includes using boosting. Boosting achieves accurate predictions by sequentially updating inaccurate models, each time correcting on the weaknesses of the predecessor. While boosting is traditionally used with decision trees and classification, its premise has shown success with linear models and regression, as demonstrated by Buehlmann et. al. [3].

Random forest techniques have also shown promise and perform very well for high dimensionality data. However, they are limited in that they do not perform as well when the number of predictors is greater than the number of data samples. This is because random forests work by using the majority vote of thousands of decision trees on bootstrapped data. Due to the lack of samples, tree pruning can lead to increased error, and this can contribute to overfitting where the model fits the training data well, but cannot capture the true relationships of the data. This will lead to decreased accuracy when exposed to new testing data [7]. Still, random forest is a common model used when the number of predictors is greater than the number of samples and recently, some statisticians have proposed new random forest methods that perform much better than traditional methods [4].

2 Subset Selection

First, we discuss subset selection techniques. In general, these techniques are discrete algorithms that attempt to find a good model using only a subset of the available predictors.

2.1 Best Subset Selection

Best subset selection is a method for selecting the most influential predictors that minimize error in a least squares linear regression. It does this by fitting a linear regression model to every possible combination of predictors and then choosing the best model of all the possible combinations. The best model is determined through the use of a test error estimate. The most common examples of test error estimation indicators are Akaike information criterion (AIC), Bayesian information criterion (BIC), Adjusted R^2, and cross validation.

While best subset selection results in the best possible model given the predictors, it is very computationally expensive. As the number of predictors increases, the number of linear models that best subset selection has to fit increases exponentially. This can be seen in Table 1. Thus, for models with more than 40 predictors, this can become infeasible for most computers to compute [9]. Given that in many scenarios,

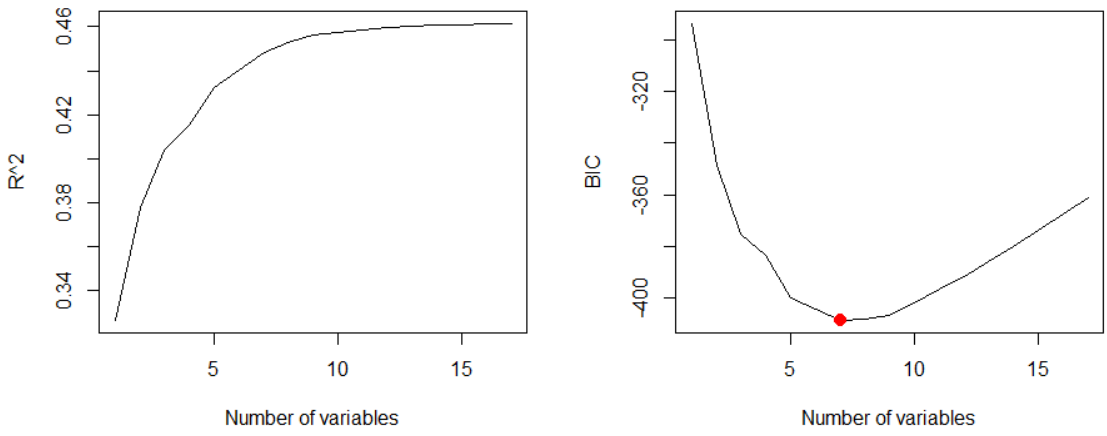
especially those seen in medicine with genomic data or in scenarios where there are more predictors than data samples, there can be many thousands of predictors, and best subset selection becomes impossible.

Table 1: Number of fitted models depending on number of predictors (p)

p	Fitted Models
2	$2^2 = 4$
10	$2^{10} = 1024$
100	$2^{100} > 10^{30}$
k	2^k

Figure 1 below demonstrates how the number of predictors can affect R^2 and BIC when using best subset selection. This plot was created by using the `leaps` library (which provides a function to run best subset selection) [1]. We used the `College` dataset provided by the `ISLR` library [8], and fit linear regression models using `Grad.Rate` as the response. We see that as the number of predictors increases, R^2 always increases (as expected). On the other hand, BIC is minimized with a moderate number of variables (between seven and nine). According to the BIC statistic, the best model has seven variables. The code used for this figure is in the `r` folder of the REU GitHub repository.

Figure 1: R^2 and BIC when applying best subset selection



2.2 Forward Stepwise Selection

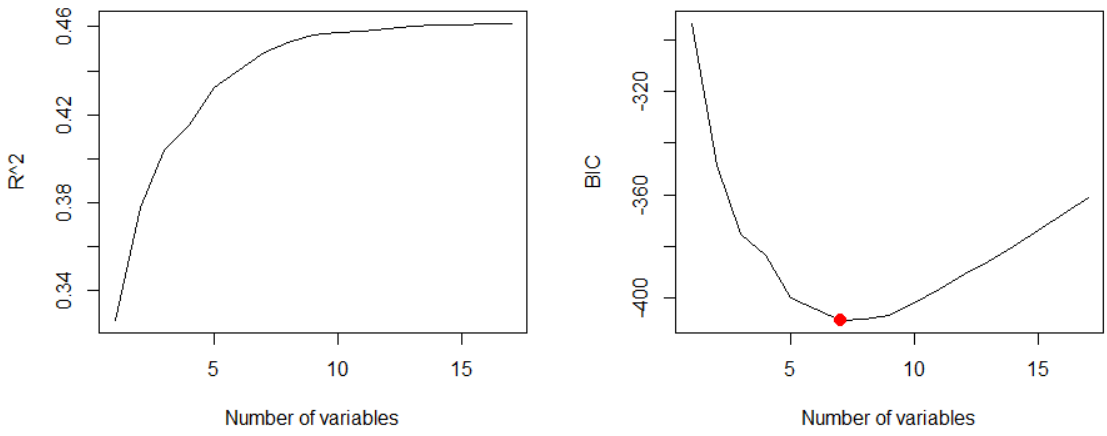
Forward stepwise selection aims to approximate the best combination of predictors in a linear regression model, but with a more computationally efficient method than best subset selection. Forward stepwise selection starts without using any predictors. It then slowly begins adding the most important predictor to the model. The predictor is chosen to minimize statistics such as p-value, AIC, BIC, or Adjusted R^2, to name a few. This is repeated until a stopping point is reached which can be defined by p-value, AIC, BIC, and more.

This process is much more computationally efficient than best subset selection, but it does not necessarily result in the best combination of parameters in the linear regression and is not guaranteed to result in the best model.

Figure 2 shows the R^2 and BIC statistics when fitting models using forward stepwise selection. Again, we predicted `Grad.Rate` using the `College` data set using the `leaps` library. The results are almost identical

to what we saw for best subset selection. Even though the plots are similar, the specific model chosen by forward stepwise selection is actually different than the one found using best subset selection.

Figure 2: R^2 and BIC when applying forward stepwise selection



2.3 Backward Stepwise Selection

Backwards stepwise selection works very similarly to forward stepwise selection, except that it starts with every single predictor included in the least squares linear regression. Instead of adding predictors like in forward stepwise selection, backward stepwise selection removes the least important predictor in each iteration. Similar to the forward method, the importance of a predictor can be determined by its p-value, AIC, BIC, or Adjusted R^2. This is repeated until a pre-determined stopping point is reached.

Backward stepwise selection can often result in better models than forward stepwise selection because it is guaranteed to test all the predictors together. This is different from forward stepwise selection that can sometimes suppress predictors, especially those that are collinear. For these reasons, when its use is possible, backward stepwise selection is preferred to forward stepwise selection. However, in cases where the number of predictors are greater than the number of samples, backward stepwise selection is impossible. In these case, forward stepwise selection must be used.

Figure 3 shows R^2 and BIC after applying backward stepwise selection to the `College` data set. Again, the results are very similar to Figures 1 and 2, but the particular models chosen by the algorithm were slightly different.

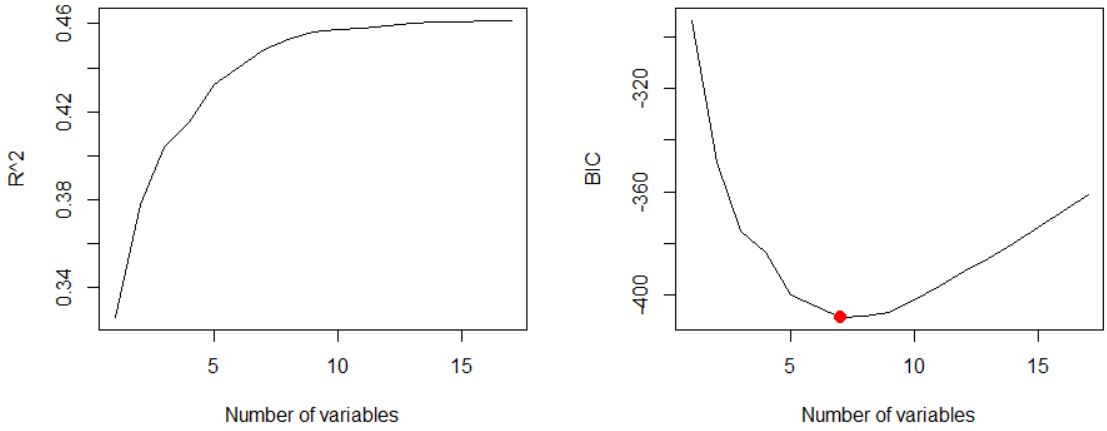
2.4 Hybrid Stepwise Selection

One weakness of the forward stepwise and backward stepwise methods is that they are greedy algorithms; in general, they will not find the best model for a given number of predictors. One way to improve model accuracy is to use hybrid stepwise selection, which allows for both forward steps and backward steps [6].

The algorithm could start with either zero predictors or all predictors. In each iteration, the method would either add a new predictor to the model or remove a predictor that does not increase performance. Like the forward and backward stepwise selection methods, this algorithm terminates when the model cannot be improved further; measuring the accuracy of the model can be determined using the AIC or BIC.

Although this strategy is slightly more computationally expensive than forward stepwise or backward

Figure 3: R^2 and BIC when applying backward stepwise selection



stepwise selection, a hybrid approach may improve model results while still avoiding the unrealistic runtime of best subset selection.

2.5 Forward Stagewise Selection

One last method for feature selection is called forward stagewise regression. Like forward stepwise selection, forward stagewise selection starts by fitting a model using none of the predictors. In each iteration, the method chooses the predictor most closely correlated to the residuals of the current model, and fits a simple linear regression using the predictor against the residuals. The coefficient for this predictor in the simple model is then added to the corresponding coefficient in the other model. This process is repeated until none of the predictors are correlated with the residuals.

Note that in each iteration of this algorithm, only one of the coefficients is changed. As a result, this method has a long runtime. In the long run, forward stagewise selection is still competitive compared to the strategies previously discussed.

3 Penalized Regression

The following sections discuss several modifications to the ordinary least squares model that penalize large coefficient estimates. Recall that an ordinary least squares model assumes that some response variable y can be modeled by

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon \tag{5}$$

where $x_1, x_2, \ldots, x_p$ are predictors, $\beta_0, \beta_1, \ldots, \beta_p$ are coefficients, and ϵ is some random error with mean 0 and variance σ^2. An ordinary least squares model estimates the coefficients as

$$\hat{\beta}^{\text{OLS}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \tag{6}$$

In general, a penalized regression imposes some additional restriction that punishes large coefficient estimates. The coefficient estimates are then given by

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

where $p(\beta_j)$ is some penalty function. Generally, this function should be large when the coefficient is large so that a model with smaller coefficients is favored. Penalized regression is especially useful when $p > n$ because the penalties can reduce variance and perform variable selection at the cost of some bias.

Because these models place penalties on large coefficient estimates, it is common to standardize the predictors to have a mean of 0 and a variance of 1. This ensures that none of the predictors have a disproportionate effect on the estimated coefficients.

3.1 Ridge Regression

Ridge regression helps to solve multicollinearity in predictors while also minimizing insignificant predictors. 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 8. 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.

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

An alternative way to express ridge regression is with the equation

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} \beta_j^2 \leq t \tag{9}$$

for some tuning parameter t.

3.2 Lasso Regression

The least absolute shrinkage and selection operation, often referred to as *lasso*, is a shrinkage method with a very similar form to lasso regression [10, 8, 9]. The coefficient estimates satisfy

$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right) + \lambda \sum_{j=1}^{p} |\beta_j| \right\} \tag{10}$$

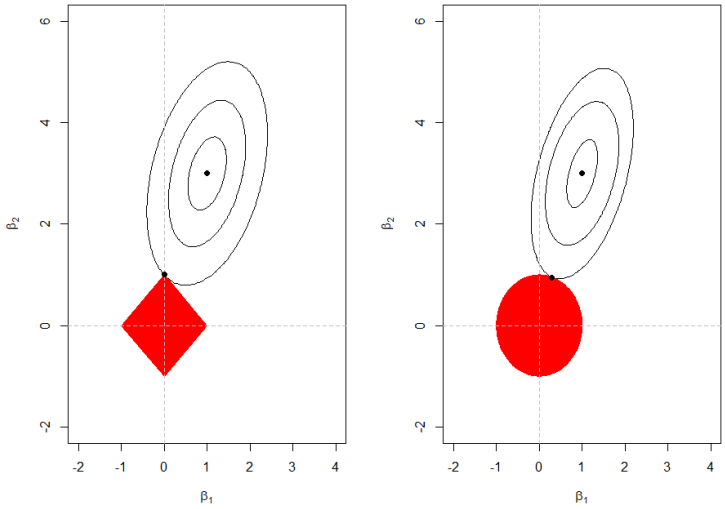
If $\lambda = 0$, then the lasso model is equivalent to the ordinary least squares model; if $\lambda \to \infty$, then the coefficients for all predictors will be set to 0. An equivalently way to define lasso regression is by

$$\hat{\beta}^{\text{ridge}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq t \tag{11}$$

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 4 below. This figure demonstrates the case when $p = 2$ and $t = 1$. The red diamond on the left represents the condition $|\beta_1| + |\beta_2| < 1$ for ridge regression, while the circle on the right represents the condition $\beta_1^2 + \beta_2^2 < 1$ for lasso regression. The black curves represent contours of the residual sum of squares error 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.

Figure 4: Error and constant curves for the lasso and ridge models when $p = 2$.



In the left plot, the intersection of the black curve and the red diamond represents the parameter values chosen by lasso regression; this point minimizes the RSS under the condition $|\beta_1| + |\beta_2| < 1$. Because the red region is a square, this intersection occurs at a point where $\beta_1 = 0$; hence, the lasso method removes the predictor β_1. On the other hand, the circular shape of the constrained region for ridge regression cannot perform variable selection because the intersection does not occur at one of the axes.

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. This can be demonstrated by the greater spread of β coefficients compared to ridge regression, as visible in Figure 5.

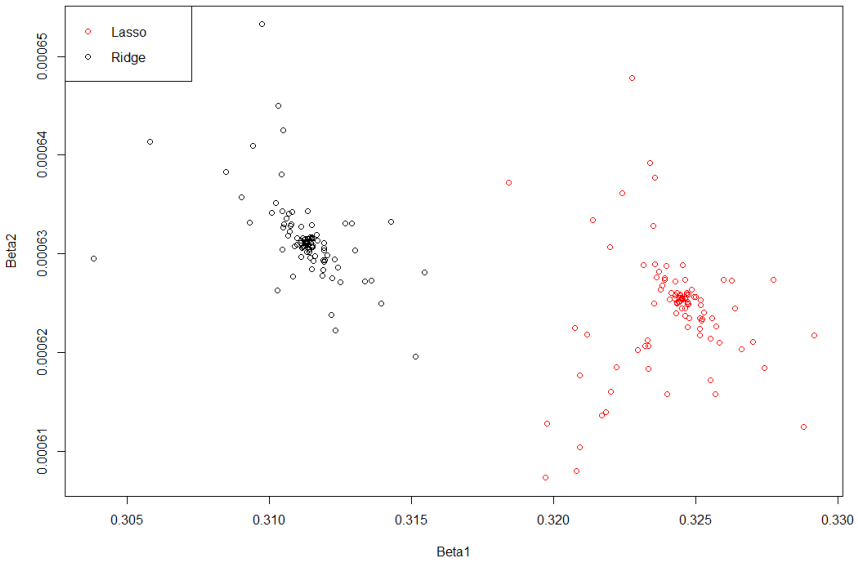
3.3 Elastic Net Regression

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

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

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

Figure 5: Instability of Lasso and Ridge Regression to Changes in Training Data



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 [13]. 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.

3.4 Adaptive Lasso Regression

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. [12] can be defined by

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

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 14 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.

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

3.5 Smoothly Clipped Absolute Deviation Regression

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 [5]. 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.

$$\hat{\beta}^{\text{SCAD}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{i=1}^{p} \beta_j x_{ij} \right) + \lambda \sum_{j=1}^{p} J_a(\beta_j, \lambda) \right\} \tag{15}$$

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

$$\frac{dJ_a(\beta,\lambda)}{d\beta} = \lambda \cdot \text{sign}(\beta) \left[I(|\beta| < \lambda) + \frac{(a\lambda - |\beta|)_+}{(a-1)\lambda} I(|\beta| > \lambda) \right] \tag{16}$$

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

$$\frac{dJ_a(\beta,\lambda)}{d\beta} \begin{cases} \lambda, & |\beta| \leq \lambda \\ \frac{a\lambda - |\beta|}{a-1}, & \lambda < |\beta| < a\lambda \\ 0, & \alpha\lambda < |\beta| \end{cases} \tag{17}$$

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 6a for a plot of the SCAD penalty as a function of the coefficient value.

Integrating with respect to β [2], we see that

$$J_a(\beta,\lambda) = \begin{cases} \lambda|\beta|, & |\beta| \leq \lambda \\ \frac{2a\lambda|\beta| - \beta^2 - \lambda^2}{2(a-1)}, & \lambda < |\beta| < a\lambda \\ \frac{\lambda^2(a+1)}{2}, & a\lambda < |\beta| \end{cases} \tag{18}$$

3.6 Minimax Concave Penalty Regression

The minimax concave penalty (MCP) method is very similar to smoothly clipped absolute deviation [11, 2]. Both methods are used to avoid the high bias caused by the lasso method. MCP uses a penalty function that satisfies

$$\frac{dJ_a(\beta,\lambda)}{d\beta} = \begin{cases} \text{sign}(\beta)\left(\lambda - \frac{|\beta|}{a}\right), & |\beta| \leq a\lambda \\ 0, & a\lambda < |\beta| \end{cases} \tag{19}$$

where $\lambda \geq 0$ and $a > 0$ are tuning parameters. Integrating [2], we see that

$$J_a(\beta,\lambda) = \begin{cases} \lambda|\beta| - \frac{\beta^2}{2a}, & |\beta| \leq a\lambda \\ \frac{1}{2}a\lambda^2, & a\lambda < |\beta| \end{cases} \tag{20}$$

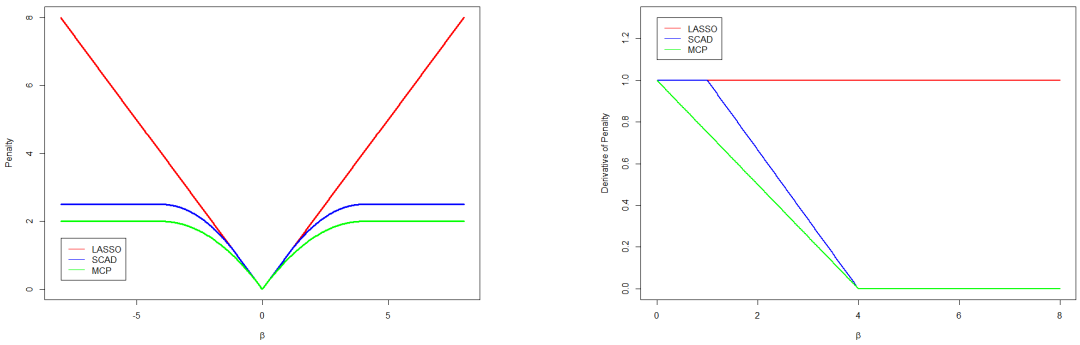
Figure 6 below shows the penalty functions (and their derivatives) for LASSO, SCAD, and MCP as a function of a coefficient value β. We see that LASSO applies a much stronger penalty to large coefficients than SCAD or MCP. Also, note that SCAD starts with a derivative equal to that of the lasso for small 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 7 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.

Figure 6: Penalty functions for LASSO, SCAD, and MCP, as well as their derivatives. These plots use $\lambda = 2$ and $a = 3$.

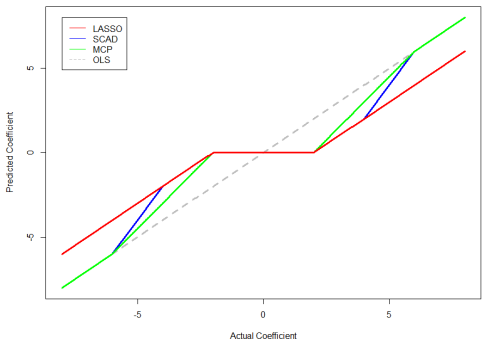
(a) Penalty functions for LASSO, SCAD, and MCP

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



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



4 Evaluating Methods

4.1 Monte Carlo Data Generation

Using Monte Carlo simulations to generate data is extremely useful for evaluating linear regression models and comparing them against each other. Generated data is beneficial because it allows us to know the true coefficient values, which allows accuracy, bias, and more to be calculated. It additionally allows for control of the data and situations. For example, this can be used to test different models in environments where predictors are completely independent, which is impossible with real data, or on the opposite spectrum when predictors are heavily correlated with each other. This can allow for the testing of model strengths and weaknesses. Because datasets can be generated thousands of times, Monte Carlo simulations can be used to evaluate randomness and variation of different models.

Data with n samples from p predictors is created through Monte Carlo simulations by first defining a vector M such that

$$M = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix}$$

with the true coefficient values β_0 through β_p that will later be compared to the fitted coefficient values.

Secondly, a n x p matrix X is generated such that

$$X = \begin{bmatrix} \mathcal{N}(\mu, \sigma^2)_{1,1} & \mathcal{N}(\mu, \sigma^2)_{1,2} & \dots & \mathcal{N}(\mu, \sigma^2)_{1,p} \\ \mathcal{N}(\mu, \sigma^2)_{2,1} & \mathcal{N}(\mu, \sigma^2)_{2,2} & & \\ \vdots & & \ddots & \\ \mathcal{N}(\mu, \sigma^2)_{n,1} & & & \mathcal{N}(\mu, \sigma^2)_{n,p} \end{bmatrix}.$$

This serves as the predictor data that will be used to determine the output result.

Another vector, E, is generated such that

$$E = \begin{bmatrix} \mathcal{N}(\mu, \sigma^2)_1 \\ \mathcal{N}(\mu, \sigma^2)_2 \\ \vdots \\ \mathcal{N}(\mu, \sigma^2)_n \end{bmatrix}.$$

This serves as the error between the predicted values using the true coefficients with predictor data and the output values.

Lastly, the output data, Y, is a vector such that

$$Y = XM + E$$

$$Y = \begin{bmatrix} \mathcal{N}(\mu, \sigma^2)_{1,1} & \mathcal{N}(\mu, \sigma^2)_{1,2} & \dots & \mathcal{N}(\mu, \sigma^2)_{1,p} \\ \mathcal{N}(\mu, \sigma^2)_{2,1} & \mathcal{N}(\mu, \sigma^2)_{2,2} & & \\ \vdots & & \ddots & \\ \mathcal{N}(\mu, \sigma^2)_{n,1} & & & \mathcal{N}(\mu, \sigma^2)_{n,p} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \mathcal{N}(\mu, \sigma^2)_1 \\ \mathcal{N}(\mu, \sigma^2)_2 \\ \vdots \\ \mathcal{N}(\mu, \sigma^2)_n \end{bmatrix}.$$

Completely independent predictor data and their corresponding outputs can be generated thousands of times, each time yielding different values. A model can be fit to the data from each iteration to determine the variance of the model and evaluate randomness.

4.2 Variance

4.3 Accuracy

4.4 Bias

References

[1] Thomas Lumley based on Fortran code by Alan Miller. *leaps: Regression Subset Selection*, 2020. R package version 3.1.

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

[3] Peter Buehlmann et al. Boosting for high-dimensional linear models. *The Annals of Statistics*, 34(2):559–583, 2006.

[4] Louis Capitaine, Robin Genuer, and Rodolphe Thiébaut. Random forests for high-dimensional longitudinal data. *Statistical Methods in Medical Research*, 30(1):166–184, 2021.

[5] 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.

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

[7] Robin Genuer, Jean-Michel Poggi, and Christine Tuleau. Random forests: some methodological insights. *arXiv preprint arXiv:0811.3619*, 2008.

[8] 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.

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

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

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

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

[13] 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.