Variable Selection Techniques

Gabriel Ackall and Connor Shrader

June 20, 2021

Contents

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.

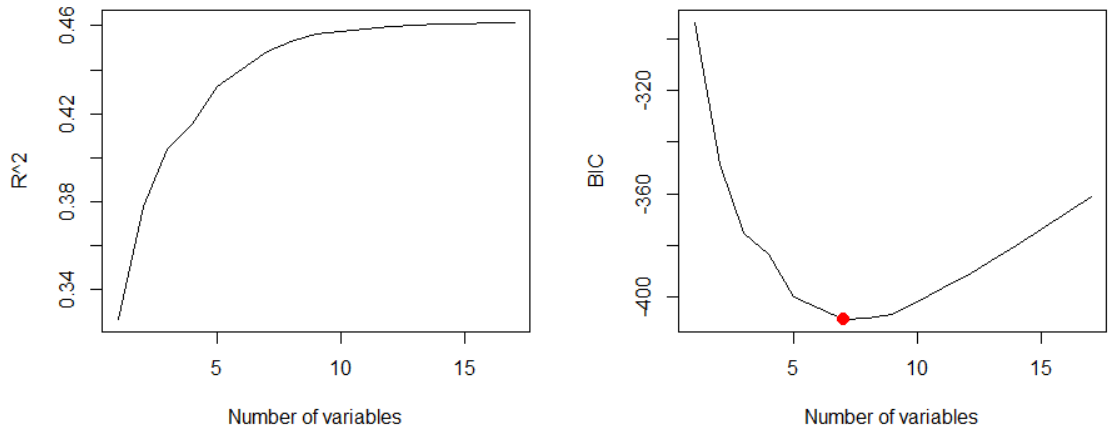
While best subset selection results in the best possible model given the predictors, it is very computationally expensive. As the number of predictors increases, the number of linear models that best subset selection has to fit increases exponentially. This can be seen in Table 1. Thus, for models with more than 40 predictors, this can become infeasible for most computers to compute [10]. 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 [9], 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



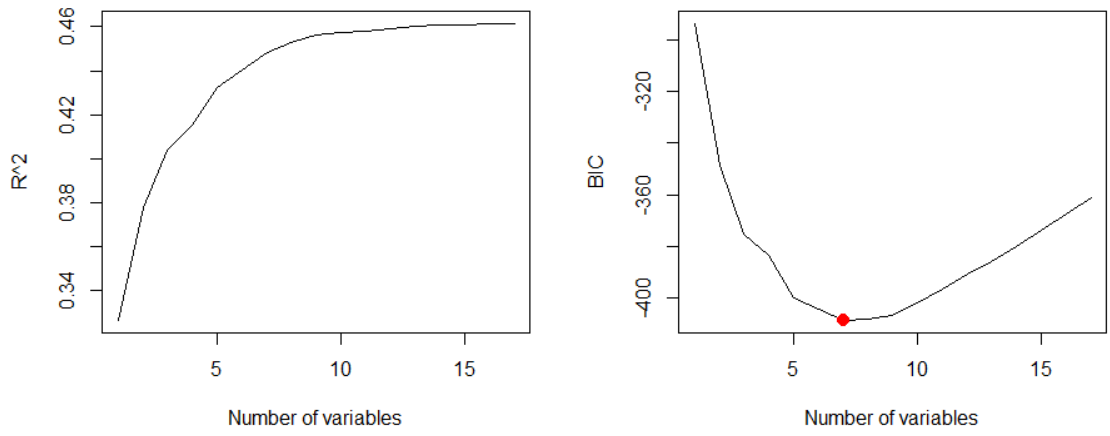
2.2 Forward Stepwise Selection

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

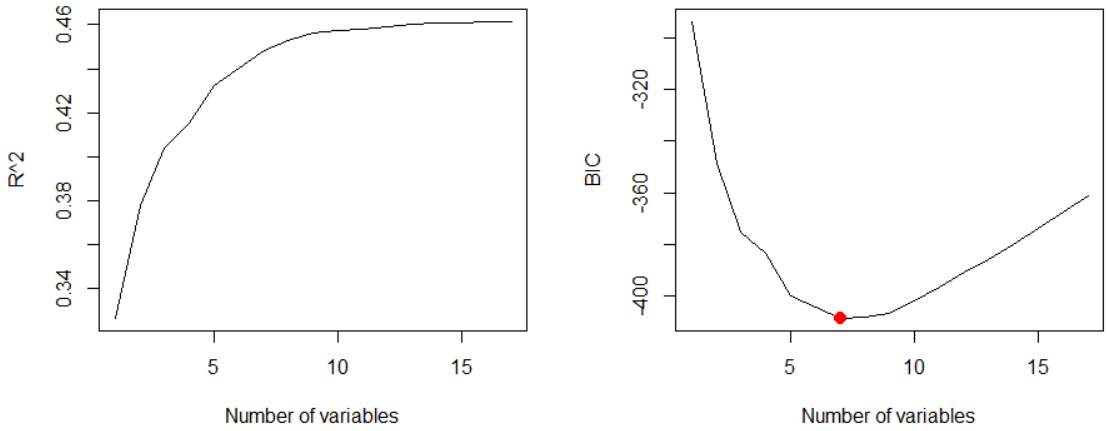
Backwards stepwise selection works very similarly to forward stepwise selection, except that it starts with every single predictor included in the least squares linear regression. Instead of adding predictors like in forward stepwise selection, backward stepwise selection removes the least important 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.

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



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 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 [8]. 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 [11, 9, 10]. The coefficient estimates satisfy

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

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

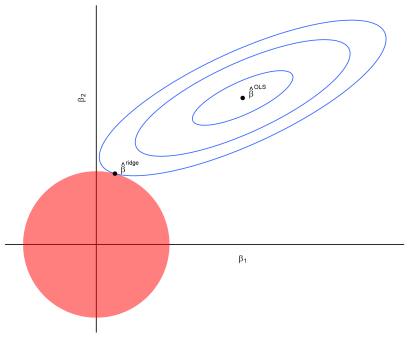
$$\hat{\beta}^{\text{lasso}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{p} |\beta_j| \leq t \tag{11}$$

where t is a tuning parameter.

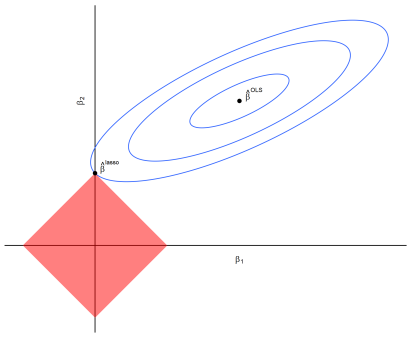
One useful property of the lasso method is that it can perform variable selection by setting some coefficients to zero. To understand why lasso regression can perform variable selection whereas ride regression cannot, consider Figure 4 below. This figure demonstrates the case when $p = 2$ and $t = 1$. The circle in 4a represents the condition $\beta_1^2 + \beta_2^2 < 1$ for ridge regression, while the red diamond in Figure 4b 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.

Figure 4: 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.



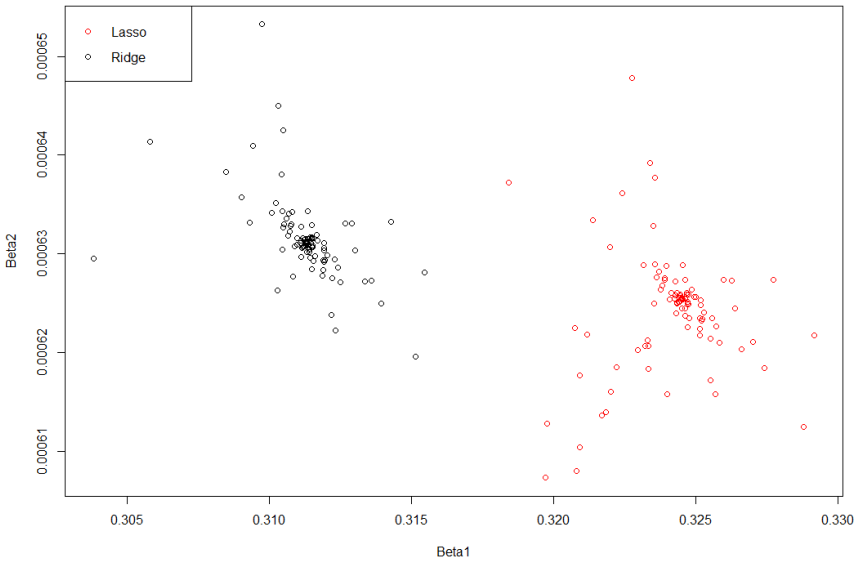
(b) RSS contours and the lasso penalty boundary.



In 4a, 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.

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.

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



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.

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 [14]. 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. [13] 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 [12, 2]. Both methods are used to avoid the high bias caused by the lasso method. MCP uses a penalty function that satisfies

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

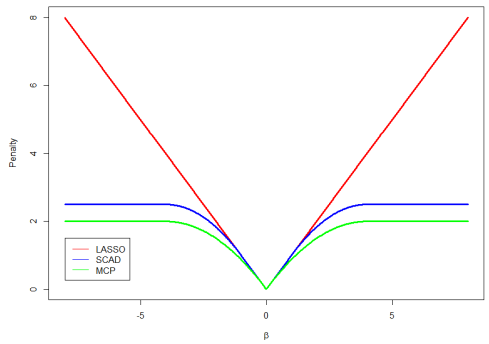
where $\lambda \geq 0$ and $a > 0$ are tuning parameters. Integrating [2], we see that

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

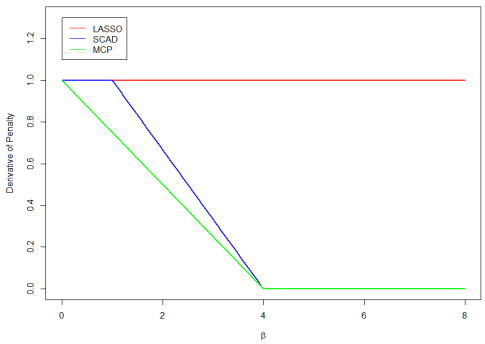
Figure 6 below shows the penalty functions (and their derivatives) for LASSO, SCAD, and MCP as a function of a coefficient value β. We see that LASSO applies a much stronger penalty to large coefficients than SCAD or MCP. Also, note that SCAD starts with a derivative equal to that of the lasso for small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

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



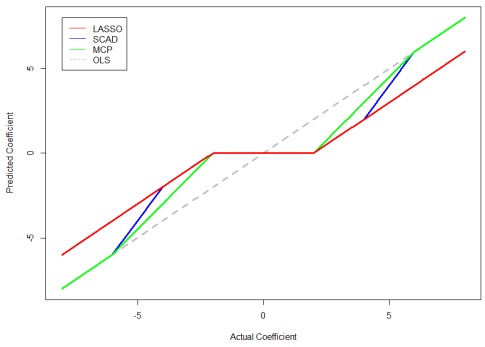
(b) Derivatives of the penalty functions for LASSO, SCAD, and MCP



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.

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 Non-linear Models

We next discuss several non-linear methods for regression and classification.

4.1 Decision Tree Regression

A decision tree is a machine learning algorithm of nodes and branches in a tree-like structure. They work from top to bottom by iteratively splitting into groups based on input values in order to minimize residual sum squared. This creates a flowchart that starts from the top and can be followed down. The direction of movement is determined by a boolean condition on the node.

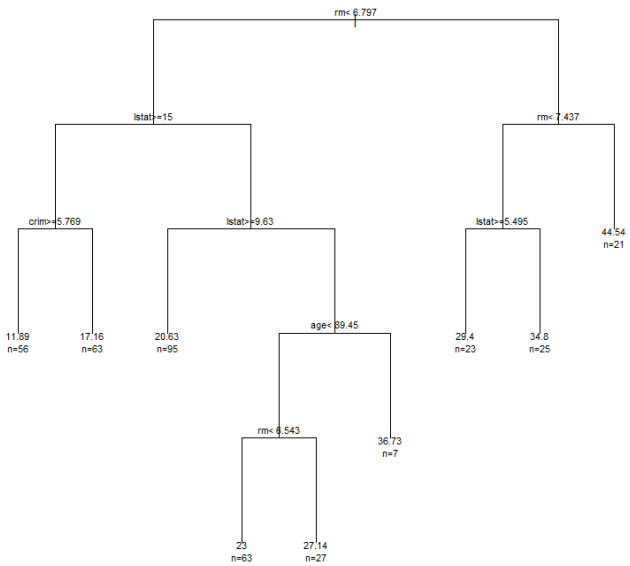
An example decision tree can be seen in Figure 8. An example row of predictor data points can be seen in Table 2. In this illustration, if the condition on the node is true, then the left direction is taken, and if the condition is false, then the right direction is taken. For these example data points, the decision tree predicts a median housing value of $20,630, which is slightly over the correct value of $18,900.

Decision trees have many other uses and can be used for both classification and regression. Additionally, decision trees can prune branches that use a variable with low importance. This can serve as a type of variable selection and helps to reduce overfitting as well as the complexity of the model. This will inherently cause some bias, but because of the reduction in variance, overall error is decreased.

Table 2: Example predictor data points of house values in Boston Suburbs

crim (crime per capita)	zn (lot zone proportion)	indus (proportion of non-retail business)	chas (bounds Charles River)	nox (nitrogen oxides concentration)	rm (rooms per dwelling)	age (proportion of units built before 1940)
0.11747	12.5	7.87	0	0.524	6.009	82.9
tax (tax rate)	black (proportion of black population)	dis (distance to employment centers)	ptratio (pupil-teacher ratio)	rad (accessibility to radial highways)	lstat (lower status of population)	
311	396.9	6.2267	15.2	5	13.27	

Figure 8: Decision tree of Boston suburbs median home value (in $1000s)



4.2 Random Forests

One of the flaws of decision tree models is that they generally have high variance. A small change in some of the data points can completely change the tree's structure. Random forest models can overcome the high variance of decision tree models by aggregating the predictions of many separate trees. Each individual tree is fitted using only a subset of the available observations and predictors. This means that each tree will be significantly different from any of the other trees. To make a prediction with a random forest model, the predictions of each individual tree are calculated and aggregated (usually using the mean for regression and the mode for classification).

By only considering a subset of the predictors for each tree, there will be relatively little correlation different trees. In the case where all p predictors are used for each tree (which is called bagging), many of the trees will have similar structures because they all use the predictors most correlated with the response. This can cause high variance. In many random forest models, it is common to use $\sqrt{p}$ predictors for each individual tree.

4.3 Gradient Boosting Model

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.

GBMs 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 GBM 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 GBM with linear regression, there is the ability to use lasso, ridge, and elastic net penalized regression as the weak learner and perform variable selection.

4.3.1 XGBoost

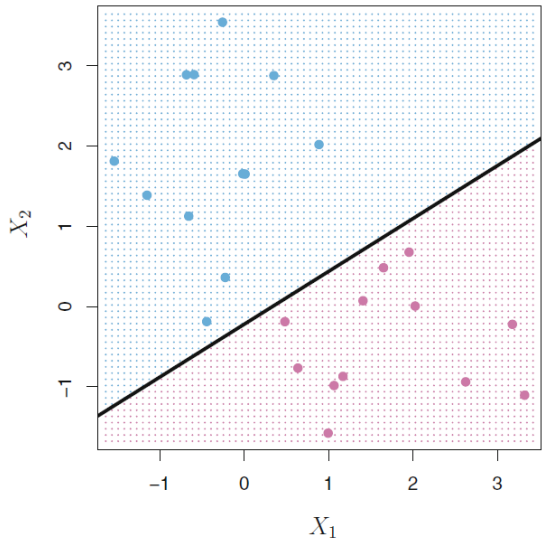
GBMs often suffer from slow computation speeds due to the large number of sequential models that need to be trained. Extreme Gradient Boosting (XGBoost) is a faster version of GBM that utilizes parallel computing as well as different optimization techniques to speed up computation. For these reasons,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

4.4 Support Vector Machines

Support vector machines are versatile statistical models; they can be used for both regression and classification. In the most basic case, a support vector machine is a binary classier whose decision boundary is a $(p-1)$-dimensional hyperplane in p-dimensional space. This hyperplane 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. Figure 9 demonstrates the line chosen by a support vector machine in the case where there are two predictors.

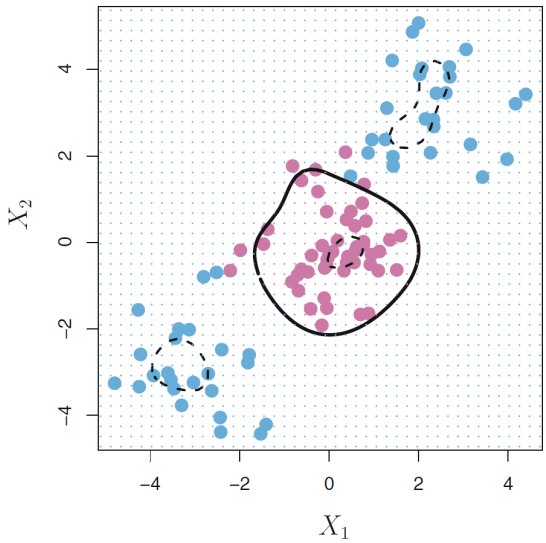
This description of a support vector machine is simple but incomplete. Most datasets are not able to be split perfectly into two sides like the example shown in 9. Furthermore, this model has very high variance changing the points near the hyperplane can significantly alter the hyperplane. Finally, this model cannot handle cases where the true boundary is non-linear.

Figure 9: The decision line for a support vector machine maximizes the margin between itself and any observations. Image source: [10], page 340



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. Figure 10 shows an example of this.

Figure 10: Using a radial kernel to create a decision boundary. Image source: [10], page 353



Support vector machines can be generalized even further to handle more than two classes, or even regression.

5 Evaluating Methods

5.1 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 the $(p+1) \times 1$ vector $\beta = [\beta_0, \beta_1, \ldots, \beta_p]^\top$ of coefficient values.

Next, a $n \times (p+1)$ matrix $\mathbf{X}$ is generated such that the entries of the first column are all 1 and the remaining $n \times p$ submatrix is distributed according to the p-dimensional multivariate normal distribution $\mathcal{N}_p(0, \mathbf{\Sigma})$. Entries have mean zero and covariance determined by the covariance matrix $\mathbf{\Sigma}$. The next section discusses different choices for the covariance matrix and how they can reflect real data.

Finally, a $n \times 1$ error vector $E \sim \mathcal{N}(0, \sigma^2)$ is generated with mean zero and variance σ^2. This error term reflects the random error for the general linear model from equation 1.

The response $\mathbf{y}$ can then be computed as

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

We can then fit statistical models using the predictor variables from $\mathbf{X}$ and the corresponding response $\mathbf{y}$. This process can then be repeated many times to evaluate model effectiveness.

5.2 Choice of the Covariance Matrix

Recall that data generated using Monte Carlo simulations is distributed according to the p-dimensional multivariate normal distribution $\mathcal{N}_p(0, \mathbf{\Sigma})$. The choice for the covariance matrix $\mathbf{\Sigma}$ can have a significant effect on the performance of statistical models. Here, we describe several different types of covariance matrices.

5.2.1 Independent Covariance

Independent covariance assumes that the covariance matrix is diagonal; that is, it can be written as

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_p^2 \end{bmatrix} \tag{22}$$

where σ_i^2 is the variance of predictor i. Because $\mathbf{\Sigma}_{ij}$ whenever $i \neq j$, each pair of predictors are independent.

An independent covariance matrix is easy to use since it is equivalent to just taking p single-variable normal distributions. However, the assumption that each pair of variables is independent is unrealistic; in most real-world situations, there is some correlation between different variables. As such, considering other covariance matrices can lead to results that better reflect real-world data.

5.2.2 Symmetric Compound

With a symmetric compound covariance matrix, we assume that $\mathbf{\Sigma}$ takes the form

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho & \cdots & \rho \\ \rho & \sigma_2^2 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & \sigma_p^2 \end{bmatrix} \tag{23}$$

where σ_i^2 is the variance of predictor i and $\rho \in (0, 1)$ is the covariance between every pair of predictors. This type of covariance is more general than independent covariance, and it can also be more realistic. However, in most real-world scenarios, the correlations between pairs of parameters will likely not be the same.

5.2.3 Blockwise

In a blockwise covariance distribution, we assume that $\mathbf{\Sigma}$ is can be written as a block-diagonal matrix

$$\mathbf{\Sigma} = \begin{bmatrix} S_1 & O & \cdots & O \\ O & S_2 & \cdots & O \\ \vdots & \vdots & \ddots & \vdots \\ O & O & \cdots & S_k \end{bmatrix} \tag{24}$$

Blocks denoted as O are zero blocks. Each block S_l has the form

$$S_l = \begin{bmatrix} \sigma_r^2 & \rho & \cdots & \rho \\ \rho & \sigma_{r+1}^2 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & \sigma_s^2 \end{bmatrix} \tag{25}$$

where σ_i^2 is the variance of predictor i and $\rho \in (0, 1)$ is the covariance of predictors in the same block.

5.2.4 Unstructured Covariance

Finally, unstructed covariance assumes that $\mathbf{\Sigma}$ takes no structured form. We have

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1p} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{p1} & \sigma_{p2} & \cdots & \sigma_p^2 \end{bmatrix} \tag{26}$$

where σ_i^2 is the variance of predictor i and σ_{ij} is the covariance of predictors i and j. The covariance matrix is always symmetric, so $\sigma_{ij} = \sigma_{ji}$.

5.2.5 Autoregressive Covariance

Autoregressive covariance (also called AR(1)), assumes that

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho & \cdots & \rho^{p-1} \\ \rho & \sigma_2^2 & \cdots & \rho^{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho^{p-1} & \rho^{p-2} & \cdots & \sigma_p^2 \end{bmatrix} \tag{27}$$

where σ_i^2 is the variance of predictor i and $\rho \in (0, 1)$. In general, for $i \neq j$, we have $\mathbf{\Sigma}_{ij} = \rho^{|i-j|}$. This implies that predictors that are near each other are more correlated than predictors far apart. AR(1) is a very useful type of covariance matrix because it is not as simple as independent or symmetric compound covariance. AR(1) is particularly useful when used for applications such as time-series analysis, since nearby predictors are usually close together in time. Consequently, it makes sense for predictors close in time to have the highest correlation.

5.3 Factorial Design

Using Monte Carlo simulations, we can study the effectiveness of regression models using a factorial design. For each simulation, we have control over parameters such as the number of predictors, the correlation of predictors, and the amount of random error. For each parameter, we chose two or three different levels. our simulations go through every possible combination of these parameter values.

Below is a list of the parameters we tuned and the values that we used:

- n, the number of observations.
- p, the number of predictors.
- p^*, the number [proportion?] of predictors that are non-zero.
- σ, the variance of the random error in each observation.
- The type of covariance among the predictors [do we include blockwise and unstructured?].
- ρ, the covariance between predictors (the exact meaning of ρ depends on the type of covariance used).

Table 3 shows the values chosen for each parameter in this study.

Table 3: Choice of parameter values for the simulation study

Parameter	Value 1	Value 2	Value 3
n	50	200	1000
p	10	100	2000
p^*	?	?	?
σ	1	3	6
Covariance type	Independent	Symmetric Compound	Autoregressive
ρ	0.2	0.5	0.9

Because of the factorial design, every possible combination of these parameters will be considered.

5.4 Fitting Models

5.4.1 Subset Selection Models

We fitted subset selection models (such as stepwise forward and stepwise backward selection) using the `stepAIC` function from the `MASS` package. For each subset selection algorithm, we fitted two different models: one using the Akaike Information Criterion (AIC) and one using the Bayesian Information Criterion (BIC). This gave us a total of eight subset selection models. We also evaluated the full OLS model (containing all predictors) and the null model (containing no predictors).

In the case where $p > n$, the full model cannot be computated. Furthermore, due to the functionality of `stepAIC`, we did not compute the subset selection models in the case where $2p > n$. Hence, when $2p > n$, the only models computed were the null model, the penalized regression models, and the non-linear models.

5.4.2 Penalized Regression Models

The lasso, ridge, and elastic-net models were computed using the `glmnet` library. We used the `cv.glmnet` function, which automatically selects the optimal value of λ that minimizes the test mean-squared error (using cross-validation). However, `glmnet` only tests λ in a certain range, so it is possible that the function misses the best possible value of λ. To avoid this issue, we manually checked the values of λ chosen by `glmnet` to make sure that none of them were corner cases.

For elastic-net, we used a value of $\alpha = 0.8$. This means that the penalty function more closely resembles the lasso penalty than the ridge penalty.

For SCAD and MCP, we used the `ncvreg` library. Like `glmnet`, `ncvreg` contains a cross-validation function `cv.ncvreg` to find the value of λ that minimizes the mean-squared error. By default, `ncvreg` uses $\gamma = 3.7$ for SCAD and $\gamma = 3$ for MCP.

5.4.3 Non-Linear Models

To fit models using random forests and gradient boosting, we used grid search. This means that we generated models using many different combinations of hyperparameters and selected the combination that made the best predictions for a validation set. The resulting model was then used to make predictions on the test set to evaluate performance. Tables 4 and 5 below show the choices of hyperparameters we turned with grid search.

Table 4: Choice of hyperparameter values for random forest grid search.

Parameter	Value 1	Value 2	Value 3
something	something	something	something

Table 5: Choice of hyperparameter values for gradient boosting grid search.

Parameter	Value 1	Value 2	Value 3
something	something	something	something

5.5 Confusion Matrix

In classification, a confusion matrix, as visible in Table 6, can tell us how well a model classifies different objects. For regression, there is no classification, and so a confusion matrix cannot be used in its traditional way. However, it can be used to identify how well a variable selection linear regression model chooses its predictors. In this case, it can be viewed as classification of whether the model chooses the predictor or not, compared to the truth of whether the predictor has a coefficient > 0 or is not significant and thus is 0. This results in 4 possibilities for each model coefficient: True Positive meaning that the model correctly included the predictor, True Negative meaning that the model correctly didn't include the predictor, False Positive meaning that the model incorrectly included the predictor, and False Negative meaning that the model incorrectly didn't include the predictor.

The values in a confusion matrix can be used to calculate a variety of different statistics. Some of the most common of these statistics are accuracy, sensitivity, and specificity. The importance of each statistic depends on the scenario, but the higher the accuracy, sensitivity, or specificity; the better.

Accuracy represents the total percentage of classifications that are classified correctly. This is by far the most commonly used statistic and can be calculated by

$$\text{Accuracy (ACC)} = \frac{TP + TN}{Total}. \tag{28}$$

		Predicted Condition	
		Positive	Negative
True Condition	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Table 6: Confusion Matrix

Sensitivity is the true positive rate. In other words, it is the rate of True Positive classifications against the total amount of actual Positive classifications. Sensitivity is often very important in tests for a disease or illness. This is because a patient can always be tested again if their positive result is false, but if a positive patient is mistakenly diagnosed as negative, the illness can go undetected and can lead to injuries or mortality. In these scenarios, specificity will often be sacrificed for increased sensitivity. Sensitivity can be calculated by

$$\text{Sensitivity (SEN)} = \frac{TP}{TP + FN}\,. \tag{29}$$

Specificity is the true negative rate. It is the rate of True Negative classifications against the total number of actual Negative classifications. Specificity is very important when False Positive results are very detrimental. In many cases, False Positive results are not as detrimental as False Negatives and thus specificity is often traded for sensitivity. However, in cases such as the court system, having a very high specificity is important so that innocent people are not wrongly convicted of crimes they did not commit. Specificity, also known as true negative rate, can be calculated by

$$\text{Specificity (TNR)} = \frac{TN}{TN + FP}\,. \tag{30}$$

Another commonly used confusion matrix statistic is F-score. F-score also helps measure the accuracy of classifications. It is determined from the precision and sensitivity of a model and helps to portray the balance between the two. Precision, also known as positive predictive value, can be calculated by

$$\text{Precision (PPV)} = \frac{TP}{TP + FP}. \tag{31}$$

With the equation for precision and sensitivity, the F-score can be calculated and simplified to

$$F_1 = \frac{2 \cdot PPV \cdot SEN}{PPV + SEN} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}\,. \tag{32}$$

5.6 Bias

Bias is the difference between the estimated value of a coefficient and its true value. Because it necessitates the true value of a coefficient, bias can only be calculated when data is generated using Monte Carlo methods. Bias can be calculated as a function of the euclidean distance between the predictor coefficients through the equation

$$\text{bias(model)} = \sqrt{\sum_{i=1}^{p} \left(\beta_i - \hat{\beta}_i\right)^2}\,. \tag{33}$$

Variable selection linear regression methods increase bias, but still lead to overall decreases in error because they decrease error from variance. This trade-off must be carefully balanced and depends greatly on the environment of the data.

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

[9] 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.

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

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

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

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

[14] 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.