# An Analysis of Penalized Regression in High Dimensional Scenarios

Gabriel Ackall[1]*, Connor Shrader[2]*
Mentor: Dr. Seongtae Kim[3]
[1]Georgia Tech, Civil Engineering
[2]University of Central Florida, Mathematics
[3]NCA&T University, Mathematics and Statistics
*Authors contributed equally

October 9, 2021

**Abstract**

With the prevalence of big data in recent years, the importance of modeling high dimensional data and selecting influential features has increased greatly. High dimensional data is common in many fields such as genome decoding, rare disease identification, economic modeling, and environmental modeling. However, most traditional regression machine learning models are not designed to handle high dimensional data or conduct variable selection. In this paper, we investigate the use of penalized regression methods such as ridge, least absolute shrinkage and selection operation (lasso), elastic net (E-net), smoothly clipped absolute deviation (SCAD), and minimax concave penalty (MCP) compared to traditional machine learning models such as random forest, XGBoost, and support vector machines. We evaluate these models using factorial design methods for Monte Carlo simulations in 270 environments, with factors being the number of predictors, number of samples, signal to noise ratio, covariance matrix, and correlation strength. We also compare different models using empirical data to evaluate their viability in real-world scenarios. Since our models are regression models, we evaluate the models using the test mean squared error, variable selection accuracy, $\beta$-sensitivity, and $\beta$-specificity. From our investigation, our findings indicate that penalized regression models outperform more traditional machine learning algorithms in most high-dimensional situations or in situations with a low number of data observations. Machine learning models are not often compared to penalized regression methods and so our analysis helps to expand the scope of how penalized regression is used to help model data. Additionally, the analysis helps to create a greater understanding of the strengths and weaknesses of each model type and provide a reference for other researchers on which machine learning techniques they should use, depending on a range of factors and data environments.

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

# Contents

# 1 Introduction

In the modern world, machine learning techniques such as random forest, gradient boosting, and support vector machines are often touted as versatile one-size-fits-all solutions when it comes to modeling big data [24]. This is due in part to tree based models such as XGBoost winning numerous machine learning competitions [24]. While this versatility is frequently the case, an increasingly common type of data set where there are more predictors than observations can pose challenges for these machine learning algorithms. In these situations, lesser known statistical modeling techniques that perform variable selection can potentially perform equivalently or even better than these machine learning techniques. However, there is a distinct lack of academia focusing on comparing these variable selection techniques with the more traditional machine learning techniques. This paper serves to help bridge that gap.

In these situations where there are more predictors, $p$, than observations, $n$, many traditional machine learning techniques either become infeasible to use or fail to give good predictions. The large number of predictors and small number of observations make it easy for such models to **overfit**, meaning that the models become fine tuned to the exact training data and instead of finding generalized patterns for a population of data, they find specific occurrences in the training data [18, 14]. Because of this, overfitted models are sensitive to new data which causes them to perform extremely well on the training data, but poorly on testing data or when deployed in the real world. Because a model's predictions in real world scenarios and on new data is the entire purpose of a model, it is very important to reduce overfitting so that predictive accuracy in these scenarios is maximized.

This paper investigates several methods to handle the large $p$, small $n$ problem. First, We considered wrapper methods such as forward selection, backward selection, stepwise forward selection and stepwise backward selection using both Akaike information criterion (AIC) and Bayesian information criterion (BIC) as the stopping criteria for the models [1, 28]. These models fit several linear models using different subsets of predictors and selects the model that optimizes a specific metric. In addition, we studied penalized regression models such as ridge regression [17], least absolute shrinkage and selection operation (lasso) [30], elastic-net [38], smoothly clipped absolute deviation (SCAD) [12], and minimax concave penalty (MCP) [36]. These models simultaneously select important predictors and fit a linear model. Finally, we considered a few machine learning models: random forests (RF) [4], gradient boosting in the form of XGBoost [6], and support vector machine (SVM) models [7]. These models do not assume a linear relationship between a response and its predictors. To compare these different techniques, models were trained and evaluated using both Monte Carlo simulations and empirical genomic data.

Section 2 contains details about each model and details the implementation of these models for our study. Section 3 describes our simulation study design and results, while section 4 explains our empirical data analysis and results. Section 5 is a discussion of our results and Section 6 is the conclusion.

# 2 Methodology

## 2.1 Modeling Background

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

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

where $f$ is a function and $\epsilon$ 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 $f$ using a **training set**. This study focuses on **regression modeling**, where the response $Y$ is a number on a continuous interval.

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 that the models attempt to estimate.

The most common method to estimate the coefficients in a linear model is with **ordinary least squares** (OLS), which selects the values $\beta_0, \beta_1, \ldots, \beta_p$ that minimize the residual sum of squares

$$\text{RSS} = \sum_{i=1}^{n} \left[ y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \cdots + \hat{\beta}_p x_{ip}) \right]^2 \tag{3}$$

OLS is common because it is the best linear unbiased estimator; that is, OLS has a lower variance than any other linear unbiased estimator [16, 14]. However, if the number of predictors $p$ is large compared to the number of observations $n$, OLS will overfit to the training data. Furthermore, if $p$ exceeds $n$, then the OLS has infinitely many solutions that simply interpolate the training data. In these cases, OLS becomes unreliable for making predictions on test data.

By using models that have a small amount of bias, the high variance of OLS can be mitigated. Liu et. al. in [22] describe three types of variable selection algorithms. **Filter methods** work by evaluating the ability for each individual predictors to predict the response; then, a model is fit using the predictors selected [26, 9]. **Wrapper methods** fit models using different subsets of predictors and choose the model that has the best performance. Finally, **embedded methods** perform variable selection during the model training process. This paper focuses on wrapper methods and embedded methods. In addition, we considered several non-linear machine learning methods to draw a comparison between linear regression models and machine learning models.

## 2.2 Subset Selection Methods

**Subset selection methods** are wrapper methods that attempt to find a subset of the predictors $X_1, X_2, \ldots, X_p$ that are most correlated with the response variable $Y$. These

algorithms usually fit models for many different subsets and choose the subset of predictors that results in the best model. Although subset selection techniques can be applied to many types of models, we will focus on subset selection with linear regression.

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

**Forward stepwise selection** starts by fitting a model with none of the predictors (by simply estimating each observation to be the mean of the response). The algorithm then iteratively chooses the predictor that best increases the model fit until a stopping condition is met. **Backward stepwise selection** does the same thing, but starts with a full model and works backwards. In addition, **forward stepwise selection** and **backward stepwise selection** are hybrid techniques that can both add and remove predictors in each iteration. Note that backward selection and backward stepwise selection can only be used when $p < n$, since they rely on starting on a full OLS model.

## 2.3 Penalized Regression

In general, **penalized regression** works by fitting a model that punishes large coefficient estimates. By forcing coefficient values to shrink, the resulting model will have relatively low variance. All of the models discussed here can be used when there are more predictors than observations. Most, but not all, of these methods can also perform variable selection.

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

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^{n} \Big[ y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \Big]^2 + \lambda \sum_{j=1}^{p} P(\beta_j) \right\} \tag{4}$$

where the first summation is the usual residual sum of squares, $\lambda \geq 0$ is a hyperparameter that controls the strength of the penalty, and $P(\beta)$ is a penalty function applied to each of the coefficients. The penalty is not applied to the intercept $\beta_0$.

**Ridge regression** is a penalized regression model that uses the penalty function $P(\beta) = \beta^2$ [17] in Equation 4. Ridge regression benefits from having a relatively simple solution and its ability to handle colinearity; however, it is unable to perform variable selection.

The **least absolute shrinkage and selection operation**, commonly referred to as **lasso**, is a shrinkage method with a very similar form to ridge regression [30, 18]. The penalty function for lasso is $P(\beta) = |\beta|$. Unlike ridge regression, the lasso can perform variable selection.

**Elastic-net** (or E-net) regression uses both the ridge penalty and the lasso penalty at the same time by adding their two penalties (possibly with different tuning parameters $\lambda_1$ and $\lambda_2$) [38].

Two additional penalized regression techniques are **Smoothly-Clipped Absolute De-**

**viation** (SCAD) and **Minimax Concave Penalty** (MCP) [12, 36]. These methods use more complicated penalty functions that punish larger coefficients less severely. This helps to reduce the bias of the resulting models. Another feature of SCAD and MCP is their oracle-like properties [12, 36]. This means that as $n \to \infty$, SCAD and MCP will correctly identify exactly which predictors should have non-zero coefficients, and that the coefficient estimates will be normally distributed with the mean estimate being the true coefficient value [37].

## 2.4 Non-linear models

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

A **random forest model** aggregate the predictions of a large ensemble of decision trees to estimate the response value [4]. Each tree is fitted independently using a subset of predictors and observations. The observations are selected *with replacement*, in a process called **bootstrapping** [11].

**Boosting** is the technique of sequentially improving a weak learner until it becomes a strong learner [27]. A **gradient boosting machine** (GBM) is a boosting technique that uses gradient descent to minimize error in a model and correct the shortcomings of the previous model [15]. Unlike random forest models, where each tree is independent of one another, the trees in a boosting model are fitted sequentially to correct the mistakes made by the previous tree.

The final non-linear model that we considered is the **support vector machine** (SVM) [7, 10]. Support vector machines find a hyperplane that closely fits the data. Unlike linear regression, support vector machines can address non-linear relationships between the response and its predictors.

## 2.5 Implementation

This section gives the specific details how we fit each model for both the simulated data and the empirical data. Everything in our study was run on version 4.1.0 of `R` [25]. Table 1 summarizes the packages used for each model.

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

| Library | Models used | Version |
|---|---|---|
| `stats` [25] | Ordinary least squares | 4.1.0 |
| `MASS` [32] | Forward and backward selection | 7.3-54 |
| `glmnet` [13] | Ridge, lasso, elastic-net | 4.1-1 |
| `ncvreg` [3] | SCAD and MCP | 3.13.0 |
| `xgboost` [6] | Gradient boosting | 1.4.1.1 |
| `ranger` [33] | Random forest (simulations) | 0.12.1 |
| `randomForest` [21] | Random forest (empirical data) | 4.6-14 |
| `e1071` [23] | Support vector machine | 1.7-7 |

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

Subset selection models using forward, backward, stepwise forward, and stepwise backward selections were fitted using the `MASS` library. For each of these four algorithms, we fit models using two criteria that determine when to stop adding and removing predictors: Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) [1, 28]. In general, using the AIC will lead to more predictors getting non-zero coefficient estimates.

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

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

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

For gradient boosting with `xgboost`, we used different values for the learning rate (0.1, 0.3, and 0.5) and maximum tree depth (1, 3, and 7). A maximum of 1000 trees were generated, with an early stopping condition if the model failed to improve for 10 iterations in a row. We used five folds in the cross validation.

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

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

Some of the models used could only be used for certain values of $n$ and $p$. This is because either the runtime becomes unreasonable when $n$ or $p$ are large, or the model simply cannot

be used when $p$ is too large. Ordinary least squares was only used when $p \leq n$, since it cannot be used at all when $p > n$. For the same reason, the backward subset selection algorithms were also used only when $p \leq n$. The forward subset selection algorithms were only used when $p \leq n$ and $p \leq 40$. When $p > 40$, the runtimes for forward selection and forward stepwise selection become unreasonably long due to the exponentially increasing number of possible predictor combinations.

Lasso, SCAD, MCP, GBM, and random forest models were used for all data sets. Support vector machine models were made for all of the simulated data but was not used for the empirical data. Support vector machine models cannot handle the immense number of predictors in our empirical data.

# 3 Monte Carlo Simulations

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

## 3.1 Simulation Design

Our simulation study used two different functions for the response variable $Y$. Our first function assumed a linear relationship between the response and its predictors $X_1, X_2, \ldots, X_p$, while the second response used a non-linear relationship. By considering both linear and non-linear response functions, we obtain a more thorough understanding of how each model performs in different situations.

The linear response function assumes that

$$Y = 1 + 2X_i - 2X_2 + 0.5X_5 + 3X_6 + \epsilon \tag{5}$$

where $\epsilon$ is an independent random error with mean 0 and constant variance. Our non-linear response function uses

$$Y = 6 \times 1_{X_1>0} + X_2^2 + 0.5X_6 + 3X_7 + 2 \times 1_{X_8>0} \times 1_{X_9>0} + \epsilon \tag{6}$$

where $1_{X_i>0}$ is the index function given by

$$1_{X_i>0} = \begin{cases} 0, & X_i \leq 0 \\ 1, & X_i > 0 \end{cases} \tag{7}$$

Note that the non-linear response still includes linear terms.

For each simulation, we generated a random $n \times (p+1)$ matrix $\mathbf{X} \sim \mathcal{N}_p(0, \mathbf{\Sigma})$, where $\mathcal{N}_p(0, \mathbf{\Sigma})$ is the multivariate normal distribution with mean zero and covariance matrix $\Sigma$. The response $\mathbf{y}$ was then computed using one of the response functions. Finally, a normally

distributed random error $\mathbf{e} \sim \mathcal{N}(0, \sigma^2)$ with mean 0 and standard deviation $\sigma$ was added to the response.

We assumed that the variance for each predictor was 1, meaning that the covariance matrix $\mathbf{\Sigma}$ is actually a correlation matrix. For any $i \neq j$, the entry $\mathbf{\Sigma}_{ij}$ represents the correlation between predictors $i$ and $j$. Correlation between predictors can affect the ability for models to identify important predictors and make accurate predictions.

We considered the following correlation structures for our simulation study:

- **Independent** correlation, where $\mathbf{\Sigma} = \mathbf{I}_p$, the $p \times p$ identity matrix;
- **Symmetric compound** correlation, where $\mathbf{\Sigma}_{ij} = \rho \in (0, 1)$ for all $i \neq j$;
- **Autoregressive correlation**, where $\mathbf{\Sigma}_{ij} = \rho^{|i-j|}$, where $\rho \in (0, 1)$; and
- **Blockwise correlation**, where $\mathbf{\Sigma}$ is block diagonal with each block having symmetric compound structure (and a shared value of $\rho$)

Our simulation study uses a **factorial design**, meaning that we ran simulations using every possible combination of different factors. The factors that we varied in our simulation study are

- The choice of response function (linear or non-linear)
- $n$, the number of observations (50, 200, and 1000),
- $p$, the number of predictors (10, 100, and 2000),
- $\sigma$, the standard deviation of the random error (1, 3, and 6),
- The correlation matrix structure (independent, symmetric compound, autoregressive, and blockwise), and
- $\rho$, the correlation between predictors (0.2, 0.5, and 0.9)

By taking every possible combination of these factors, we obtain $2 \times 3 \times 3 \times 3 \times 4 \times 3 = 648$ different settings for the simulations. However, because an independent correlation matrix does not have any correlation between predictors, the value of $\rho$ is not used. Hence, we only needed to run 540 different settings. For each combination of factors, we ran 100 simulations. Each simulation randomly 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.

## 3.2 Evaluating Model Performance

We used four metrics to evaluate the performance of each model on the simulated data: **train mean squared error**, **test mean squared error**, $\beta$**-sensitivity** and $\beta$**-specificity**. The mean squared error (MSE) is computed by

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{8}$$

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

Because we are using simulated data, where the true response function is known, we can measure the $\beta$-sensitivity and $\beta$-specificity for each penalized linear regression model that performs variable selection [22]. A coefficient estimate is a **true positive** (TP) if the coefficient is predicted to be non-zero when that predictor is actually related to the response value. The estimate is a **true negative** (TN) if the coefficient was correctly predicted to be zero when that predictor is not related to the response. A **false positive** (FP) happens when an important coefficient is incorrectly predicted to be non-zero. Finally, a coefficient estimate is a **false negative** (FN) if it was estimated to be zero but that predictor is actually related to the response. A model that perfectly identifies the important and unimportant predictors will have only true positives and true negatives.

The $\beta$-sensitivity of a model is given by

$$\beta\text{-sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{9}$$

while the $\beta$-specificity is given by

$$\beta\text{-specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{10}$$

The $\beta$-sensitivity is a measure of a model's ability to correctly identify predictors that are related to the response. If the $\beta$-sensitivity is close to 1, then the model assigns non-zero coefficients to all the important predictors; if instead the $\beta$-sensitivity is close to 0, then the model cannot identify important predictors well. Similarly, the $\beta$-specificity of a model measures how well it can identify unimportant predictors (i.e. predictors that should be given a coefficient of zero).

### 3.3 Linear Simulation Results

Because we ran simulations using 540 different combinations of factors, we only show the results for $n = 50$ and $p = 2000$ in this report (representing the largest ratio between $p$ and $n$). Each plot measures the average value for one of the four metrics discussed above over 100 simulations. Each row of the plots represent a different value of $\sigma$, the standard deviation of the random error. Each column represents a correlation structure. The different shapes and colors for each point represent the strength of the correlation between predictors.

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

We begin by presenting the results from our simulations that used the linear response function, followed by the results from the non-linear response function.

Figures 1 and 2 display the average mean squared error for simulations when $n = 50$ and $p = 2000$.

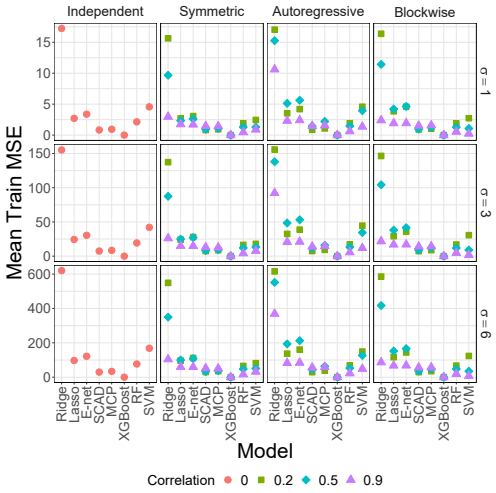


Figure 1: Average mean square error using training data for linear simulations when $n = 50$ and $p = 2000$.

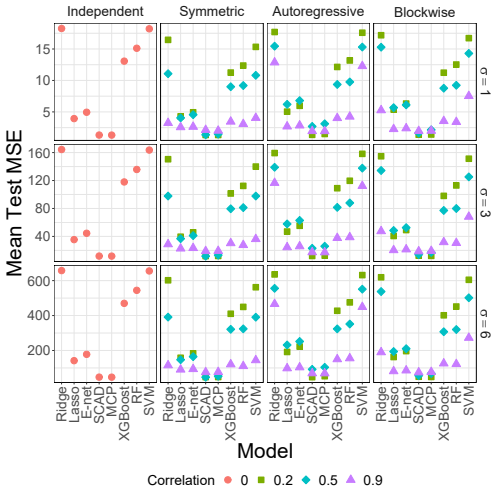


Figure 2: Average mean square error using testing data for linear simulations when $n = 50$ and $p = 2000$.

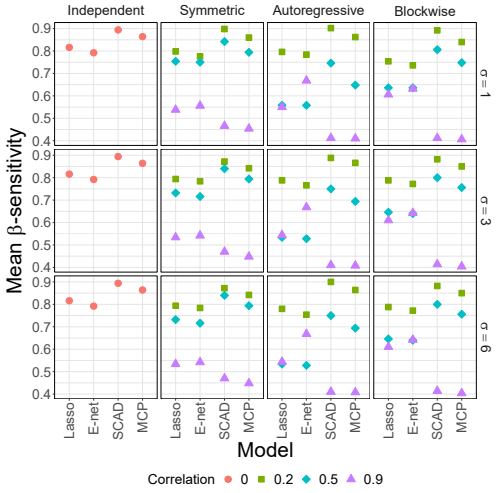


Figure 3: Average $\beta$-sensitivity for linear simulations when $n = 50$ and $p = 2000$.

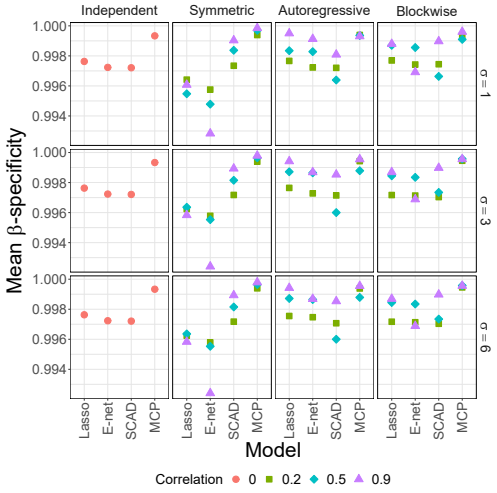


Figure 4: Average $\beta$-specificity for linear simulations when $n = 50$ and $p = 2000$.

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

there is a strong correlation between predictors. On the other hand, the linear models are somewhat less affected by the correlation.

Figure 4 shows the mean $\beta$-sensitivity for each model. We see that all of the models predict most of the non-zero coefficients when the correlation is low. When the correlation is high, all of the models struggle to identify the correct predictors. SCAD and MCP perform the best when the correlation is low but perform the worst when the correlation is high. Elastic-net performs particularly well compared to the other models when the correlation is high, especially when the correlation structure is autoregressive.

Figure 3 shows the average $\beta$-specificity for the models. MCP appears to make the fewest mistakes when choosing zero coefficients. The performance of the other models depends heavily on the type of correlation and the correlation strength. Lasso and elastic-net perform the worst when the correlation structure is symmetric compound, whereas SCAD performs poorly when the correlation structure is autoregressive or blockwise. We also see that the models generally made less mistakes as the correlation increases.

## 3.4 Non-linear Simulation Results

Now, we will highlight some results from the simulations that used the non-linear response function given by Equation 6.

Figures 5 and 6 show the average mean square errors on the training data and test data, respectively. We see that the linear and non-linear models have similar test mean squared errors. Another interesting observation is that the non-linear models all have a noticeably lower test mean squared error when the correlation between predictors is high. The linear models still perform significantly worse on the training data compared to the test data.

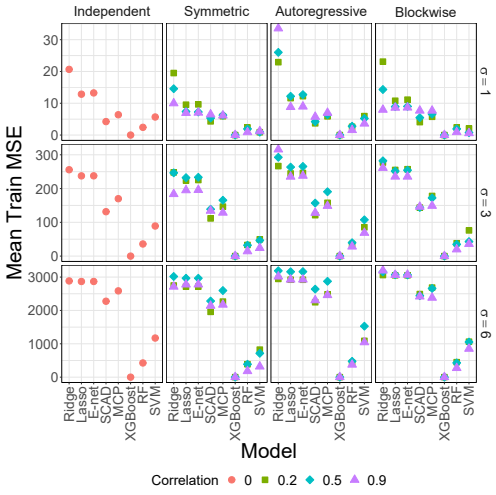


Figure 5: Average mean square error using training data for non-linear simulations when $n = 50$ and $p = 2000$.

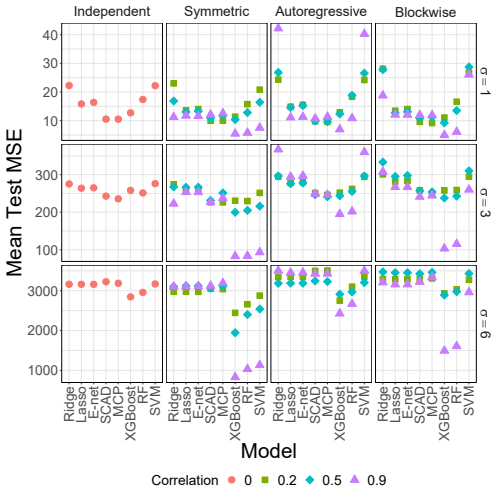


Figure 6: Average mean square error using testing data for non-linear simulations when $n = 50$ and $p = 2000$.

Figures 8 and 7 show the results for the $\beta$-sensitivity and $\beta$-specificity for the non-linear models. We see that all of the linear models estimate almost all of the coefficients as being equal to zero! SCAD and MCP were slightly more likely to correctly estimate non-zero coefficients as being non-zero, but they were also more likely to incorrectly identify unimportant predictors as having non-zero coefficients.

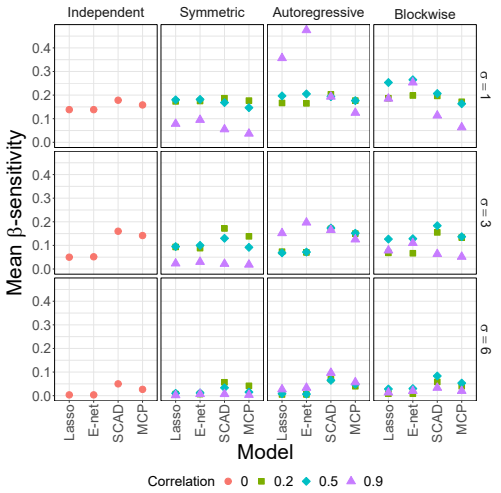


Figure 7: Average $\beta$-sensitivity for non-linear simulations when $n = 50$ and $p = 2000$.

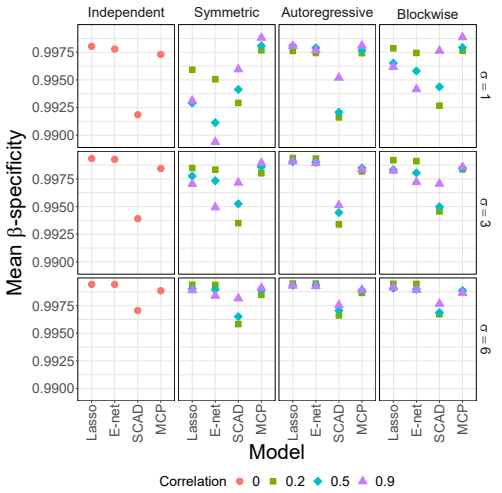


Figure 8: Average $\beta$-specificity for non-linear simulations when $n = 50$ and $p = 2000$.

# 4 Empirical Data Analysis

## 4.1 Details of Empirical Data

For empirical data, we used the Breast Cancer database from The Cancer Genome Atlas (bcTCGA). A cleaned version of the data is provided by the `biglasso R` package [35]. This data set contains the gene expression data of 17323 genes from 536 patients. One of these genes is the BRCA1 gene which is among the first genes discovered that can increase the risk of breast cancer [20, 2]. Mutations in BRCA 1 and BRCA 2, another gene discovered 1 year after BRCA1, are responsible for two-thirds of breast cancer cases in women [8]. Because the BRCA1 gene interacts with other genes, it is useful to find genes that interact with BRCA1 to test in further studies [8]. The BRCA1 gene expression level will act as the output value in our regression analysis and the other 17322 genes will serve as predictor values.

This data is a prime example of the large $p$ small $n$ problem where there are many more predictors than data samples. Because of this, only penalized regression and machine learning techniques can be used. This is because there are more predictors than samples which makes least squares linear regression impossible and due to the high number of predictors, subset and stepwise regression becomes too computationally expensive to be feasible. Addi-

tionally, support vector machines struggle at such a high number of predictors and resulted in stack overflow errors which made fitting support vector machines on this data impossible.

To evaluate the models, we used **nested cross validation**. We first split the data into five folds. For each of these folds, we used the selected fold as a test set while the other four folds were used as a training set. We then fitted the models using cross validation on this training set, where one interior fold was used as a validation set while the other folds were used to train a model. The role of the validation set in the interior cross validation is different from the test set used in the exterior cross validation. In the interior cross validation, the validation set is used to tune hyperparameters; the model that performs best against the validation set is then chosen. On the other hand, the test set in the exterior cross validation is not used to tune hyperparameters; its only purpose is to evaluate the performance of the models chosen in the inner cross validation. Because the outer test set is not used in the model fitting or selection process, it gives an unbiased evaluation of each model's performance.

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

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

## 4.2 Empirical Data Results

Recall that we used nested cross validation when fitting models on the bcTCGA data set. This means that we fitted five models using different subsets of the data for training and testing. Figure 9 below shows a plot with the training and test mean squared error for every fold of every model. The bars show the average mean squared error for the five folds. In addition, Table 2 show the aggregated results for the train and test mean squared error.

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

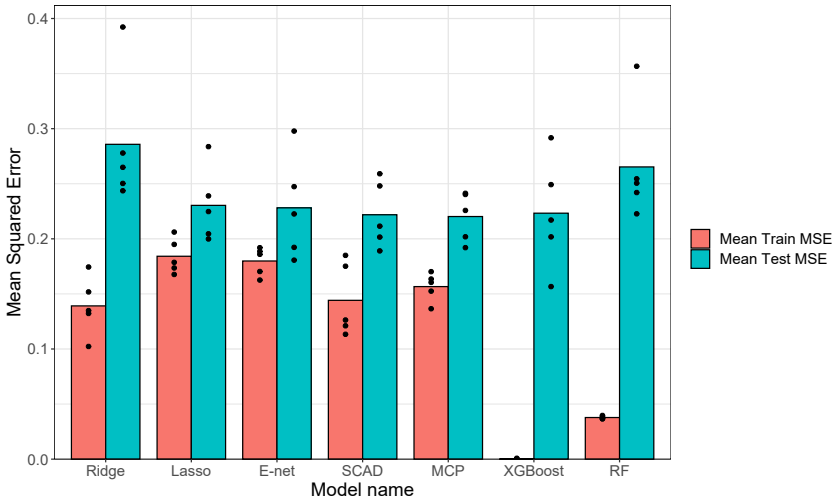


Figure 9: Mean squared error of the models fit on the bcTCGA data set. Each point represents the mean squared error for one fold, while the bars represent the average for the five folds.

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

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

Table 3: Average Runtimes for Empirical Data Models

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

# 5 Discussion

## 5.1 Discussion of Monte Carlo Results

Figures **??** and **??** shows that the best-performing models on test data were SCAD and MCP. The wrapper methods that used BIC performed better than SCAD and MCP when $n = 1000$ and $p = 10$, but could not be used in the high dimensional case where $n = 50$ and $p = 2000$. Lasso, elastic-net, and the wrapper methods with AIC all had low test mean squared errors as well. Ridge performed the worst out of all the linear models. The non-linear models also performed very badly on test data.

Notice that all the linear models except for ridge had a test mean squared error that was approximately equal to $\sigma^2$, the square of the random error. This indicates that these models had low bias and low variance, since most of their error was due to the random noise. On the other hand, ridge regression and the machine learning models had a test mean squared error that was much higher than $\sigma^2$, indicating that these models either had high bias or high variance.

Now, consider the training mean squared error results from Figures **??** and **??**. We see that training error for the linear models was roughly equal to the test error. This indicates that these models did not overfit to the training data. On the other hand, the non-linear models have far lower training mean squared errors than test errors. In particular, random forest models almost perfectly fit to the training data when $n = 1000$ and $p = 10$, while gradient boosting made an almost perfect fit when $n = 50$ and $p = 2000$.

Ridge regression, which performed badly on both training data and test data, was likely restricted by its penalty function and its inability to perform variable selection. Recall that three out of the five non-zero coefficient values chosen were greater than one. Because the penalty function for ridge squares each coefficient, ridge was very strongly discouraged from selecting coefficient estimates close to the true non-zero coefficient values. Furthermore, because ridge regression cannot set coefficients to zero and conduct variable selection, its coefficient estimates for the true non-zero coefficients were likely even smaller because unimportant predictors were given non-zero coefficient estimates. As a result, ridge regression was unable to predict the true coefficient values as well as the other linear models.

Another note about ridge regression is that it was more strongly affected by correlation between predictors when $n = 1000$ and $p = 10$, as seen in Figure **??**. The other linear models, which could set coefficients to zero, were likely able to eliminate unimportant predictors even though they were correlated with the important predictors. Ridge regression, unable to eliminate these unimportant predictors, may have instead given larger coefficient estimates to the incorrect predictors.

Interestingly, the test mean squared error for random forest and support vector machine models improved as the correlation between predictors was increased. Figures **??** and **??** show that the test mean squared error for random forest and support vector machines decreased when correlation was increased. On the other hand, XGBoost was unaffected by an increase in the correlation value.

One possible explanation for this phenomenon is that the random forest models and support vector machines are less influenced by the random error when the correlation is

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

Also note that changing the standard deviation of the random error seems to affect the mean squared errors of all the models equally. In addition, the correlation structure appears to have little influence on the training or test mean squared error.

Now, we will consider the $\beta$-sensitivity results for the linear models that could perform variable selection. These results are shown in Figures **??** and **??**. The $\beta$-sensitivity results when $n = 1000$ and $p = 10$ are rather uninteresting; all of the models correctly identified all of the non-zero coefficients except for a few cases where one coefficient was missed. The results for $n = 50$ and $p = 2000$ tell a richer story: SCAD and MCP tended to correctly identify more of the non-zero coefficients when the correlation between coefficients was low; however, when the correlation is high, SCAD and MCP found less than half of the non-zero coefficients, on average. Lasso and elastic-net performed slightly better when the correlation was high.

Looking at the $\beta$-specificity results from Figures **??** and **??**, we see that lasso, elastic-net, and the BIC wrapper methods had the highest $\beta$-specificity when $n = 1000$ and $p = 10$. When $n = 50$ and $p = 2000$, lasso and elastic-net had must lower $\beta$-specificity values compared to SCAD and MCP.

Another interesting result to note is that for both low-dimensional and high-dimensional settings, when the correlation between predictors increased, the $\beta$-specificity for SCAD and MCP typically increased. For elastic-net, a higher correlation caused the $\beta$-specificity to decrease when $n = 1000$ and $p = 10$; however, when $n = 50$ and $p = 2000$, the $\beta$-specificity of elastic-net depends heavily on the correlation structure. For a symmetric compound correlation structure, elastic-net had a much lower $\beta$-specificity as correlation increased. For an autoregressive or blockwise correlation structure, a higher correlation increased the $\beta$-specificity for elastic-net. This means that lasso and elastic-net have trouble identifying non-zero coefficients when all $p = 2000$ predictors are correlated to each other, while SCAD and MCP could handle this situation well.

Overall, we see that the ability for the penalized models to correctly identify whether coefficients are non-zero depends on how predictors are correlated with one another. However, the standard deviation of the random error does not appear to be too impactful on $\beta$-sensitivity or $\beta$-specificity.

Although wrapper methods and penalized regression models were the evident winners in this simulation study, it is worth noting that the non-linear models suffered from a few unfair disadvantages. For one, the simulated data assumed that there was a linear relationship between the predictors and response, which meant that the linear models were almost unbiased. The non-linear models were relatively more biased because they did not assume a linear relationship.

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

## 5.2 Discussion of Empirical Data Results

In our empirical data study, SCAD and MCP also maintained the lowest testing mean squared error among the tested models. This can be seen in Figure 9 and Table 2. Furthermore, the mean squared error MCP had a relatively low standard deviation across the five folds. This may be due in part to the fact that MCP consistently selected the fewest number of predictors compared to Lasso and Elastic Net as shown by the Venn diagrams in Figure **??**. Because this empirical data most likely contains multicollinearity, the fact that MCP selects fewer predictors may have helped to reduce overfitting. This is consistent with results from the Monte Carlo simulations which indicate that MCP performed better than its counterparts in multicollinear environments.

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

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

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

# 6 Conclusion

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

These comparisons have shown that penalized regression should be added to the toolbox of any data scientist. In addition to performing better with less error than traditional machine learning techniques, penalized regression is much less computationally expensive. Additionally, in scenarios such as the empirical data study outlined earlier, penalized regression techniques can help determine the relationship between predictors and a response value. In such cases, the ability to determine these relationship can be more important than the predictive performance of a model.

However, there is still room for further comparison between these models. Our Monte Carlo simulation data was generated linearly, but we hope to also analyze non-linear Monte Carlo simulations and consider variable interaction. This would allow for comprehensive analysis of penalized regression even in unfavorable conditions that provide a clear advantage to non-linear machine learning techniques.

We could also run Monte Carlo simulations where the response is categorical rather than numerical. This could be used to study how penalized regression performs when used for classification data, which is the most common case for high dimensional data sets.

We are working on developing a hybrid technique between random forest and penalized regression. This method would harness the power of ensemble learning, while still being able to perform variable selection. This would hopefully perform better than either random forest or penalized regression methods individually. There are some models such as SREEM-forest [5] which conduct variable selection, however, these models use wrapper methods and variable importance which can be computationally expensive and do not inherently eliminate insignificant variables the same way penalized regression does. Given that random forest models are already very slow, performing additional stepwise selection may result in an exponentially slower runtime. Thus, it is important that inherent variable selection methods such as penalized regression methods are utilized in ensemble methods.

Lastly, we have developed a penalty function that has the potential to conduct variable selection while also handling multicollinearity well. The penalty function follows the L1/4 + L2 regularization and can be seen in Equation 11. We are interested in implementing this penalty function and analyzing its results. Given the power of penalized regression techniques seen in this study, it is important to develop new penalized regression techniques that improve on the drawbacks of previous designs.

$$\hat{\beta}^{\text{proposed}} = \arg\min_{\beta} \left\{ \sum_{i=1}^{n} \Big[ y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \Big]^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j|^{\frac{1}{4}} + \lambda_2 \sum_{j=1}^{p} \beta_j^2 \right\} \tag{11}$$

# 7 Acknowledgments

This research was conducted as a part of the North Carolina A&T State University and Elon University Joint Summer REU Program in Mathematical Biology and was funded by the National Science Foundation DMS# 1851957/1851981.

The results here are in part based upon data generated by the TCGA Research Network: https://www.cancer.gov/tcga.

We would also like to thank Dr. Luke and Dr. Yokley and Yang Xue for their guidance throughout the research process.

# References

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

[2] Antoniou, A., Pharoah, P. D., Narod, S., Risch, H. A., Eyfjord, J. E., Hopper, J. L., Loman, N., Olsson, H., Johannsson, O., Borg, Å., et al. Average risks of breast and ovarian cancer associated with brca1 or brca2 mutations detected in case series unselected for family history: a combined analysis of 22 studies. *The American Journal of Human Genetics*, 72(5):1117–1130, 2003.

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

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

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

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

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

[8] Deng, C.-X. and Brodie, S. G. Roles of brca1 and its interacting proteins. *Bioessays*, 22(8):728–737, 2000.

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

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

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

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

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

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

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

[16] Greene, W. H. *Econometric analysis*. Pearson Education India, 2003.

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

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

[19] Kobayashi, H., Komatsu, S., Ichikawa, D., Kawaguchi, T., Hirajima, S., Miyamae, M., Okajima, W., Ohashi, T., Kosuga, T., Konishi, H., et al. Overexpression of denticleless e3 ubiquitin protein ligase homolog (dtl) is related to poor outcome in gastric carcinoma. *Oncotarget*, 6(34):36615, 2015.

[20] Kuchenbaecker, K. B., Hopper, J. L., Barnes, D. R., Phillips, K.-A., Mooij, T. M., Roos-Blom, M.-J., Jervis, S., Van Leeuwen, F. E., Milne, R. L., Andrieu, N., et al. Risks of breast, ovarian, and contralateral breast cancer for brca1 and brca2 mutation carriers. *Jama*, 317(23):2402–2416, 2017.

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

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

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

[24] Nielsen, D. Tree boosting with xgboost-why does xgboost win "every" machine learning competition? 2016.

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

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

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

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

[29] Shao, L., Cui, Y., Li, H., Liu, Y., Zhao, H., Wang, Y., Zhang, Y., Ng, K. M., Han, W., Ma, D., et al. Cmtm5 exhibits tumor suppressor activities and is frequently silenced by methylation in carcinoma cell lines. *Clinical cancer research*, 13(19):5756–5762, 2007.

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

[31] Vaccari, T. and Bilder, D. The drosophila tumor suppressor vps25 prevents nonautonomous overproliferation by regulating notch trafficking. *Developmental cell*, 9(5):687–698, 2005.

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

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

[34] Xiao, Z.-D., Liu, X., Zhuang, L., and Gan, B. Nbr2: a former junk gene emerges as a key player in tumor suppression. *Molecular & cellular oncology*, 3(4):e1187322, 2016.

[35] Zeng, Y. and Breheny, P. The biglasso package: A memory- and computation-efficient solver for lasso model fitting with big data in r. *ArXiv e-prints*, 2017.

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

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

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