

Penalized Regression in the Age of Big Data

Gabriel Ackall^{1*}, Connor Shrader^{2*}

Mentor: Dr. Ty Kim³

¹Georgia Tech, Civil and Environmental Engineering

²University of Central Florida, Mathematics

³NCA&T University, Mathematics and Statistics

*Authors contributed equally

July 5, 2021

Abstract

With the prevalence of big data in the modern age, the importance of modeling high dimensional data and selecting influential features has increased greatly. High dimensional data is common in many fields such as genome decoding, rare disease identification, economic modeling, and environmental modeling. However, most traditional regression and classification machine learning models are not designed to handle high dimensional data or conduct variable selection. In this paper, we investigated the use of penalized regression methods instead of, or in conjunction with, the traditional machine learning methods. We focused on lasso, ridge, elastic net, SCAD, MCP, and adaptive versions of lasso, ridge, and elastic net models. For traditional machine learning models, we focused on random forest models, gradient boosting models in the form of XGBoost, and support vector machines. These models were evaluated using factorial design methods for Monte Carlo simulations under various data environments. Tests were conducted for 270 environments, with factors being the number of predictors, number of samples, signal to noise ratio, covariance matrix, and correlation strength. This served to identify the strengths and weaknesses of different penalization techniques in different environments. We also compared different models using empirical datasets to test their viability in real-world scenarios. Additionally, we considered penalization methods outlined earlier in logistic regression models for classifying data. These results were compared to random forest, gradient boosting, and support vector machine classification models using both Monte Carlo data generation methods and empirical data. For regression, we evaluated the models using the test mean squared error and variable selection accuracy; for classification, we considered test prediction accuracy and variable selection accuracy. We found that for both regression and classification, penalized regression models outperformed more traditional machine learning algorithms in most high-dimensional situations or in situations with a low number of data observations. By comparing traditional machine learning methods with penalized regression, we hope to expand the scope of machine learning methods for big data to include the various penalized regression techniques we tested. Additionally, we hope to create a greater understanding of the strengths and weaknesses of each model type and provide a reference for other researchers on which machine learning techniques they should use, depending on a range of factors.

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

1 Introduction

1.1 Background

1.2 Literature Review

2 Methods

2.1 Models

2.2 Monte Carlo Simulations

Monte Carlo simulations use randomly generated data to fit and test our regression and classification models. There are several benefits to using simulated data rather than experimental data:

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

For the simulated data, we assumed that the relationship between the response variable y and the predictors x_1, x_2, \dots, x_p was linear. That is, we assumed that

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p + \epsilon \quad (1)$$

where β_0 is some intercept, β_1, \dots, β_p are coefficient values and $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is a normally distributed random error with mean 0 and variance σ^2 .

To generate the data, we first defined $\beta = [\beta_0, \beta_1, \dots, \beta_p]^\top$, a $(p+1) \times 1$ vector of coefficient values. For our simulations, we used $\beta_0 = 1$, $\beta_1 = 2$, $\beta_2 = -2$, $\beta_5 = 0.5$ and $\beta_6 = 3$; the remaining coefficient values were set to 0.

Next, we generated \mathbf{X} , a $n \times (p+1)$ matrix of predictor variables. The first column contains 1 in all of its entries; this corresponds to the intercept of our linear model. Column i of \mathbf{X} contains the variable values for predictor x_{i-1} , for $1 \leq i \leq p$. These values were generated using the p -dimensional multivariate normal distribution $\mathcal{N}_p(0, \mathbf{\Sigma})$ with mean zero and covariance matrix $\mathbf{\Sigma}$. We assumed that every predictor had a standard deviation of 1, making the covariance matrix equivalent to a correlation matrix. Four different correlation matrix structures were considered in our study.

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

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{E} \quad (2)$$

For binary classification, the response was computed by

$$TODO \quad (3)$$

We used a **factorial design** for our simulations. This means that we considered several factors that affect the data generation process, each having multiple possible values. We then generated data using every possible combination of factor values, giving us a comprehensive assessment of model performance under various conditions.

- n , the number of observations: 50, 200, 1000.
- p , the number of predictors: 10, 100, 2000.
- σ , the standard deviation of the random error: 1, 3, 6.
- The correlation matrix structure: independent, symmetric compound, autoregressive, blockwise.
- ρ , the correlation between predictors: 0.2, 0.5, 0.9.

By taking every possible combination of these factors, we obtain $3 \times 3 \times 3 \times 4 \times 3 = 324$ different settings for the simulations. However, because an independent correlation matrix does not use the correlation value ρ , we actually only used 270 combinations. For each combination of factors, we ran 100 simulations. In each simulation, we generated two data sets: one to train the various models, and one to test the models and evaluate performance. Both data sets contained n observations, meaning that a total of $2n$ observations were generated for each simulation.

As mentioned earlier, we considered four different covariance matrix structures. These structures determine the correlation between different predictors. If Σ is a correlation matrix, then Σ_{ij} , the entry at the i -th row and j -th column, represents the correlation between predictors i and j . If $\Sigma_{ij} = 0$, there is no correlation; but if $\Sigma_{ij} = 1$, then predictors i and j are perfectly correlated. Note that a correlation matrix is always symmetric, so $\Sigma_{ij} = \Sigma_{ji}$ for all indices i and j . This correlation can severely impact the performance of statistical models; if several predictors are highly correlated, then machine learning algorithms are less able to determine which predictors are actually related to the response.

The first correlation structure we considered is **independent correlation**. This means that the correlation matrix Σ has the form

$$\Sigma = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \quad (4)$$

In other words, there is no correlation between different predictors, since $\Sigma_{ij} = 0$ whenever $i \neq j$.

The next covariance structure is called **symmetric compound**. This structure has the form

$$\Sigma = \begin{bmatrix} 1 & \rho & \cdots & \rho \\ \rho & 1 & \cdots & \rho \\ \vdots & \vdots & \ddots & \vdots \\ \rho & \rho & \cdots & 1 \end{bmatrix} \quad (5)$$

where $\rho \in [0, 1]$ is some correlation value. A symmetric compound covariance structure assumes that $\Sigma_{ij} = \rho$ whenever $i \neq j$, meaning that all predictors are equally correlated with one another.

An autoregressive covariance structure assumes that

$$\Sigma = \begin{bmatrix} 1 & \rho & \cdots & \rho^{p-1} \\ \rho & 1 & \cdots & \rho^{p-2} \\ \vdots & \vdots & \ddots & \vdots \\ \rho^{p-1} & \rho^{p-2} & \cdots & 1 \end{bmatrix} \quad (6)$$

For any indices i and j , we have $\Sigma_{ij} = \rho^{|i-j|}$. Consequently, each predictor is strongly correlated with nearby predictors and weakly correlated with more distant predictors. This form of covariance is commonly seen when using time series, since observed values at nearby times are likely to be highly correlated with one another.

Finally, a blockwise correlation matrix has the block-diagonal form

$$\Sigma = \begin{bmatrix} \mathbf{B}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{B}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{B}_k \end{bmatrix} \quad (7)$$

where 0 represents a block containing all zeroes, and each block \mathbf{B}_i has a form identical to the symmetric compound matrix in 5. This implies that predictors within the same block have correlation $\rho \in [0, 1]$, whereas predictors in different blocks have zero correlation. One important consideration when using blockwise correlation is the size of each block. For our simulations, we used a block size of 5 when $p = 10$, a block size of 25 when $p = 100$, and a block size of 100 when $p = 2000$.

All of our simulations were run using version 4.1.0 of R. Several different libraries were used to fit machine learning models using our simulated data. Table 1 summarizes the libraries used to fit models.

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

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

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

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

For SCAD and MCP models, we used the `cv.ncvreg` function from the **ncvreg** library. We used the default values of a for both models: 3 for MCP and 3.7 for SCAD (note that the **ncvreg** documentation calls this hyperparameter γ instead of a).

For the three non-linear models (gradient boosting, random forests, and support vector machines), we used cross-validation and grid search to find suitable hyperparameters, and then fit a model using the full training set using those hyperparameters. For gradient boosting with **xgboost**, we used different values for the learning rate (0.1, 0.3, and 0.5) and maximum tree depth (1, 3, and 7). A maximum of 1000 trees were generated, with an early stopping condition if the model failed to improve for several iterations in a row. The cross-validation function used five folds.

For random forests using **ranger**, we tuned the number of predictors used per decision tree ($\lfloor \sqrt{p} \rfloor$, $\lfloor p/3 \rfloor$, and $\lfloor p/2 \rfloor$) and the number of trees (300, 400, 500 and 600). The cross-validation function used five folds.

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

2.3 Empirical Data

3 Results

3.1 Regression Models

3.2 Classification Models

4 Discussion

4.1 Findings

4.2 Contributions

4.3 Future Work