University of Nevada, Reno

An Application of Variable Selection Methods to Identify Influential Factors of ADHD Diagnosis in Children

A thesis submitted in partial fulfillment
of the requirements for the degree of

Bachelor of Science in Mathematics and the Honors Program

by

Matthew R. Dragan

Dr. Mihye Ahn, Thesis Advisor

May, 2017

UNIVERSITY OF NEVADA RENO

THE HONORS PROGRAM

We recommend that the thesis
prepared under our supervision by

Matthew R. Dragan

entitled

An Application of Variable Selection Methods to Identify Influential Factors of ADHD Diagnosis in Children

be accepted in partial fulfillment of the
requirements for the degree of

BACHELOR OF SCIENCE, MATHEMATICS

Mihye Ahn, Ph.D., Thesis Advisor

Tamara Valentine, Ph. D., Director, **Honors Program**

May, 2017

Abstract

In 2011, there was the ADHD-200 Global Competition in which several teams around the world competed to produce a model that best categorized children as having ADHD or not having ADHD based on phenotypic and neuroimaging data for each child. These data were gathered at eight locations in the United States, the Netherlands, and China. The model each team produced was tested and scored based on prediction accuracy. Interestingly, the team that scored the highest used only the phenotypic data. This team was disqualified because it did not use the neuroimaging data, so its model was never published. The results of the competition show that there is a relationship between phenotypic variables and ADHD diagnosis. The goal of this thesis is to identify which variables most greatly influence ADHD diagnosis in children using penalized logistic regression. According to the CDC about 11% of United States children between the ages of 4 and 17 are diagnosed with ADHD. It is important that children are correctly diagnosed. This thesis will help identify which factors are influential in ADHD diagnosis in children.

Contents

List of Tables

List of Figures

1 Introduction

In many experiments and observational studies, scientists look at the relationship between variables. A common method for evaluating relationships is through regression models (Fitzmaurice, 2016). Regression models allow scientists to test for a relationship between one or more explanatory variables and a single response variable.

Linear regression is the most commonly used regression model. The goal of linear regression is to identify a linear relationship between a continuous response variable and one or more explanatory variables. Explanatory variables can be either continuous or categorical. A continuous variable takes an infinite number values, while a categorical variable takes on a finite number of values (Fitzmaurice, 2016). In some cases, however, the response variable must be limited to a finite number of values. For example, consider a study to predict a medical diagnosis of a certain disease. In this example the response variable is the diagnosis. Since there are only two possible outcomes (1 the individual is diagnosed with the disease and 0 the individual is not diagnosed with the disease), the response variable is categorical. Since the response variable is categorical, linear regression cannot be used to produce a model. In this case, a binary logistic regression model works for these data because the predicted value from a binary logistic regression model is between zero and one. In order to categorize a result from the model a cutoff is selected. Continuing the example from above, assume that the binary logistic regression model has been fit and a cutoff of .5 has been selected. Selecting a cutoff of .5 means that if the explanatory data for one individual maps to a value greater than .5 on the model he or she will be identified as having the disease. If an individual's explanatory data maps to a value less than .5, he or she will be identified as not having the disease. In cases where the outcome is categorical but there are more than two possible outcomes, the multinomial logistic regression model can be used. The multinomial logistic regression model is

an extension of the binomial logistic regression model. In the multinomial case, a model is fit for each response variable. The predicted response is based on which of the models gives the highest probability based on the explanatory data being used. For example, consider a study to classify an individual's place of residents based on some data such as highest level of education, yearly income, etc. In this example the response variable, the individual's place of residence, could include house, apartment, dormitory, etc. In this example, there is a finite number of outcomes, but the number of outcomes is greater than two so a multinomial logistic regression model would apply.

This thesis will focus on the attention deficit hyperactivity disorder (ADHD) data available at the 100 Functional Connectomes Project website: `http://fcon_1000.projects.nitrc.org/indi/adhd200`. In 2011, several researchers around the world competed to produce the best model for classifying individuals as a typically developing child or as a child diagnosed with ADHD (Bellec et al., 2016) based on these data. These data were gathered from eight different sites. One of the most interesting outcomes of the competition is that the team that produced the best model created the model using only phenotypic data. This team did not include the neuroimaging data for fitting the model. The phenotypic competition data include variables such as gender, age, handedness, ADHD index, inattentive measure, hyperactive/impulsive measure, verbal IQ, performance IQ, and full4 IQ. Although this team produced the best model, it was disqualified because it did not use the neuroimaging data, and the model produced by this team was never published. Since the model was never published the phenotypic factors that are influential on ADHD diagnosis in children remain unknown. The goal of this thesis is to identify which phenotypic factors influence ADHD diagnosis in children. The phenotypic factors from the competition data are considered. The influential factors are determined by

fitting a binary logistic regression model and a multinomial logistic regression model with the data and subjecting the model to a penalty $P(\boldsymbol{\beta})$. The penalty will reduce the influence or entirely remove a factor from the model depending on the method of variable shrinkage and selection. The remaining factors after applying the penalty are considered to be influential in the model.

Variable shrinkage and selection methods are used to identify the influential factors of ADHD. Traditional methods of selecting a model can be computationally expensive because there are two to the power of the number of explanatory variables different possible models. For example, in a case where there are 16 explanatory variables there are $2^{16} = 65,536$ possible models because there are 65,536 different combinations of variables. Another downside of traditional methods of selecting models is that these methods are discrete, meaning that the variable is either included or excluded from the model. More modern variable shrinkage and selection methods, generally, allow for the selection of influential factors without as much computational overhead and allow the influence of a variable to be increased or decreased without removing the variable. Variable shrinkage and selection methods will be studied through a simulation study and through literature. The shrinkage and selection methods which will be studied are least absolute shrinkage and selection operator (LASSO), adaptive LASSO, Ridge Regression, Elastic Net, along with ordinary least squares (OLS) estimation method.

In order to complete this research, several aspects of statistical modeling as well as the ADHD-200 competition mentioned above are studied. First documentation of the original competition is considered. Unfortunately there is limited documentation of the original competition aside from the website describing the competition as well as the results. The data used for this thesis are available at `http://fcon_1000.projects.nitrc.org/indi/adhd200`. Bellec et al. (2016) pro-

vided a chart of the summary data of the individuals who participated in the study as seen in Figure 1. Bellec et al. (2016) noted that there was a significant difference on how the neuroimaging data were gathered from site to site, which brings into question the reliability of neuroimaging data. Although this thesis will not focus on the neuroimaging data, it is important to note that the results of the competition may be related to how the neuroimaging data were gathered. Some summary statistics of the data collected at New York University (NYU) can be seen in Table 1.

		TDC		ADHD	
Site	Sex	N	Age Range (avg.)	N	Age Range (avg.)
BHBU	F	17*	8 - 18 (13.8)	0	-
	M	9*	12 - 18 (16.1)	0	-
KKI	F	28	8 - 12 (10.3)	10	8 - 13 (9.9)
	M	41	8 - 13 (10.4)	15	8 - 13 (10.1)
NI	F	25	12 - 26 (18.8)	5	12 - 20 (15.2)
	M	12	13 - 25 (17.9)	31	11 - 21 (17.1)
NYU	F	55	7 - 18 (12.2)	34	7 - 17 (10.1)
	M	56	7 - 18 (12.0)	117	7 - 18 (11.2)
OHSU	F	40	7 - 12 (9.0)	13	7 - 11 (8.9)
	M	30	7 - 12 (9.5)	30	7 - 12 (8.9)
PKU	F	59	8 - 15 (10.9)	10	9 - 16 (10.9)
	M	84	8 - 15 (11.8)	92	8 - 17 (12.2)
Pitt	F	44	10 - 20 (15.7)	1	15
	M	50	10 - 19 (14.5)	3	14 - 17 (15.7)
WUSTL	F	28	7 - 22 (11.3)	0	-
	M	33	7 - 22 (11.5)	0	-
Totals	F	279*	7 - 26 (12.3)	73	7 - 20 (10.4)
	M	306*	7 - 25 (12.1)	288	7 - 21 (11.9)

Figure 1: Summary statistics of the participants for the ADHD-200 competition from Bellec et al. (2016).

Table 1: Summary Statistics of NYU Data Set. N is the number of observations and SD is the standard deviation.

	N	mean	SD	min	max
Gender:	204	0.647	0.479	0	1
Age:	204	11.460	2.860	7.17	17.96
Handedness:	204	0.630	0.269	-.20	1.00
ADHD Index:	204	60.221	14.929	40	99
Inattentive Measure:	204	59.721	14.659	40	90
Hyperactive/Impulsive Measure:	204	59.025	14.371	41	90
Verbal IQ:	204	108.691	14.145	65	143
Performance IQ:	204	104.922	14.578	72	137
Full4 IQ:	204	107.808	14.365	73	142

1.1 Selection Criterion

Before discussing variable shrinkage and selection methods, it is important to discuss how a model is selected. When considering a model, it is important to subject the model to some selection criteria. A selection criterion provides a basis for which model candidate will perform the best. For many of the shrinkage and selection methods, a tuning parameter must be chosen. The tuning parameter determines how strong of an effect the penalty term will have on the explanatory variables. The selection criteria provide a basis for selecting a tuning parameter which dictates the strength of the penalty term $P(\boldsymbol{\beta})$. For this thesis, four selection criteria are used in the simulation study for determining the ideal tuning parameter for each variable shrinkage and selection method of regression. These four selection criteria are Akaike Information Criterion (AIC) proposed by Akaike (1973), the Bayesian Information Criterion (BIC) which was proposed by Schwarz (1978), cross validation (CV) which was proposed by Stone (1974), and generalized cross validation (GCV)

which was proposed by Craven and Wahba (1979). The AIC selection statistic involves minimizing $-2\log(L) + 2p$, where L is the Gaussian likelihood function and p is the number of nonzero coefficients. The AIC statistic generally performs better with models with a small number of observations, but selects more complicated models as the number of observations gets large. BIC is similar to AIC and involves minimizing $-2\log(L) + p\log(n)$, where L is the Gaussian likelihood function, p is the number of nonzero coefficients, and n is the number of observations. The BIC performs well when the number of observations is large, but poorly when the number is small. Both AIC and BIC invoke a heavier penalty on models when a large number of explanatory variables are included in the model. The CV method involves dividing the data into a training set where the true response is known and a test set in which the accuracy of the model produced by the training set can be tested. The tuning parameter in the penalty term will be chosen based on which value minimizes the mean of the squared residuals of the test set. The CV method can be very computationally expensive, but can be performed for non-normal data and on non-linear models. The non-linear nature of the models for this thesis makes the CV selection criterion good for selecting a penalty. The GCV selection criterion is a generalized approach of the cross validation method that is not as computationally expensive. This criterion selects a tuning parameter for the penalization term based on which tuning parameter minimizes the GCV value. The GCV values are calculated using the following equation:

$$GCV = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{y}_i}{1 - \frac{trace(\mathbf{S})}{n}}\right)^2,$$

where n is the number of observations and $trace(\mathbf{S})$ is the sum of the diagonal elements of the Hat matrix, $\mathbf{S}$, obtained by $Hat = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$. Each of the four methods is tested in the simulation study. The performance of each variable shrinkage and

selection method is tested at each selection criterion based on some performance measures. The variable shrinkage and selection method and selection criterion that perform the best based on the performance measures is focused on for ADHD data analysis.

1.2 Penalization Methods

As stated earlier, traditional methods of variable selection can be computationally expensive and do not allow for variable coefficients to be shrunk. With traditional methods, a variable is either present or not. In this thesis, the traditional methods, such as best subset selection, are avoided by using more modern methods of variable shrinkage and selection. Tibshirani (1996) points out that best subset selection generally performs the best when there is a small number of variables with large effects meaning that there is a small number of factors that have a large influence on the model. Zou (2006) makes a note of the large computational overhead and the "inherent discreteness" of the best subset selection method. The best subset selection method is not used for this thesis due to its discreteness discussed in Zou (2006).

The Ridge Regression penalization method proposed by Hoerl and Kennard (1970) was an early method of penalizing regression models. This method was intended to improve the linear model to reduce the variance of the estimated coefficients. Ridge Regression shrinks the coefficients of the explanatory variables by subjecting them to the L_2 penalty term $P(\boldsymbol{\beta})$. In other words, Ridge Regression minimizes

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}{\beta_j}^2,$$

where $\lambda\sum_{j=1}^{p}{\beta_j}^2$ is the penalty term $P(\boldsymbol{\beta})$ and λ is the tuning parameter. The tuning parameter dictates the influence of the penalty term. According to Tibshirani (1996),

Ridge Regression performs the best when there are many explanatory variables that have small effects. The short-coming of Ridge Regression is that it cannot shrink any of the variable coefficients to zero even if the true $\boldsymbol{\beta}$ has one or more zero coefficients. This limitation makes discerning influential variables difficult.

Tibshirani (1996) proposed the LASSO penalty for variable shrinkage and selection in models. The LASSO minimizes

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \mid \beta_j \mid,$$

where the L_1 penalty term $P(\boldsymbol{\beta})$ is $\lambda \sum_{j=1}^{p} \mid \beta_j \mid$ and λ is the tuning parameter. The benefit of this penalty term is that the $\mid \beta_j \mid$ allows for regression coefficients to be reduced to zero meaning that the variable $\mathbf{x_j}$ is removed from the model. The influence of explanatory variables with high variability will be lessened or entirely removed from the model. The shrinking of some variables to zero is good for the purpose of this thesis because the factors that remain are those with the greatest influence on the response variable. According to Tibshirani (1996), the LASSO regression performs the best when there are a moderate number of variables with moderate effects. The problem with the LASSO method is that it does not satisfy oracle properties proposed by Fan and Li (2001) because the LASSO cannot achieve both selection consistency and $\sqrt{n}$-consistency simultaneously. The oracle properties include identifying the correct subset model and the model has the optimal estimation rate (Zou, 2006). A penalization approach that satisfies the oracle properties is theoretically better than one that does not satisfy oracle properties. In addition, for models where the number of explanatory factors is greater than the number of observations ($n < p$) the LASSO can select at most n factors limiting the model, and when there is variable grouping the LASSO tends to select only one variable from the group (Zou & Hastie, 2005).

The Elastic Net penalty, proposed by Zou and Hastie (2005), employs the L_1

penalization term from the LASSO and the L_2 from Ridge Regression. The Elastic Net method can be represented by

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda_1 \sum_{j=1}^{p} \mid \beta_j \mid + \lambda_2 \sum_{j=1}^{p} {\beta_j}^2.$$

Similar to the LASSO, the Elastic Net has the ability to shrink explanatory variable coefficients to zero and tends to perform well in cases where the LASSO performs poorly. According to Zou and Hastie (2005) the Elastic Net penalty frequently out-performs the LASSO penalty especially when the number of explanatory variables is much larger than the number of observations and when there are variables with high pairwise correlation. The Elastic Net addresses the case defined by Tibshirani (1996) that Ridge Regression out-performs the LASSO when the number of observations is greater than the number of explanatory variables and the data are highly correlated. According to Zou and Hastie (2005), the Elastic Net is a powerful tool for variable shrinkage and selection.

Zou (2006) proposed the adaptive LASSO penalty. The adaptive LASSO is an extension of the LASSO method but the penalty term is given weight based on some criteria. Adding a weight to the penalty term allows the penalty to be weighted differently for each explanatory variable. Giving a weight to the penalty term is beneficial because very influential factors can be penalized less heavily than factors with less influence. The adaptive LASSO can be solved using the same efficient algorithm of the LASSO and performed by convex optimization on the penalty term of the LASSO penalty (Zou, 2006). Zou (2006) outlines the use of the OLS estimates as weights for the penalty term of the adaptive LASSO. In this thesis, the adaptive LASSO will be represented by

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \hat{w}_j \mid \beta_j \mid .$$

In the simulation study, the OLS estimates will be used for the weights of the penalization term, that is, $\hat{w}_j = 1/|\beta_j^{OLS}|$. The adaptive LASSO satisfies the oracle properties proposed by Fan and Li (2001). Thus this method, theoretically, performs better than the methods that do not.

The previous methods, subjected to some selection criteria, and the OLS method are evaluated through simulation study. The OLS, LASSO, Ridge Regression, adaptive LASSO, and Elastic Net methods are tested using R for linear models and OLS, LASSO, Ridge Regression, and Elastic Net methods are tested for the binary logistic and multinomial logistic models. The adaptive LASSO penalty was not included in the binary and multinomial logistic models because of a lack of literature. The method that performs the best based on the performance measures described in the Methodology section is focused on in the ADHD data analysis.

1.3 Penalized Logistic Regression

Due to the discrete nature of the response variable for this thesis, a binary logistic regression model and a multinomial logistic regression model are used to model the data. For this thesis, the binary and multinomial logistic regression models are penalized based on the selection criterion and variable shrinkage and selection method selected from the simulation study. As stated earlier, a binary logistic regression model has a response variable that can take on values of either zero or one. In cases where there are only two outcomes of the response variable, the logistic regression model can be accurately expressed with this "S" shaped curve (Agresti, 2007) given by

$$\pi(\mathbf{x}) = \frac{e^{\mathbf{x}^T\boldsymbol{\beta}}}{1 + e^{\mathbf{x}^T\boldsymbol{\beta}}},$$

where $\pi(\mathbf{x}) = P(Y = 1|X = \mathbf{x})$. For this thesis, the binary logistic regression model, with the response variables of 0: typically developing children and 1: children diag-

nosed with ADHD, will be used to model ADHD diagnosis based on the phenotypic data from the ADHD-200 competition. The multinomial logistic regression model is an extension of the binary regression model. In this case, the response variable can take on a finite number of values, but there are more than two possible outcomes (Agresti, 2002). The multinomial logistic regression model is given by

$$\pi_i(\mathbf{x}) = \frac{e^{\mathbf{x}^T\boldsymbol{\beta}^{(i)}}}{\sum_{j=1}^{k} e^{\mathbf{x}^T\boldsymbol{\beta}^{(j)}}},$$

where k is the number of response categories, $\boldsymbol{\beta}^{(i)}$ is the model parameters for the ith category, and $\pi_i(\mathbf{x}) = P(Y = i|X = \mathbf{x})$. The multinomial logistic model fits a logistic model for each response outcome. The predicted response in a multinomial model is calculated by applying explanatory variables to each logistic model. The logistic model that produces the highest probability, for that set of explanatory variables determines which category the response falls under. For this thesis, a multinomial logistic regression model with three different response values is used to model ADHD diagnosis with one outcome being typically developing children and the others are two sub-types of ADHD. There is a third sub-type of ADHD that will not be considered in this thesis due to lack of observations within the data. The explanatory data for the multinomial regression model is the phenotypic data from the ADHD-200 competition.

For purpose of identifying the influential factors of ADHD, both the binary logistic regression model and the multinomial logistic regression model must be subjected to some penalty term. The results of the simulation study will determine which model and selection criterion will be used. For binary logistic regression, Tibshirani (1996) first applied the LASSO penalization method to the logistic model. Shi, Yin, Osher, and Sajda (2010) reviewed a comprehensive list of algorithms for sparse logistic regression (sparse logistic regression and penalized logistic regression are the same).

This paper enumerates the application of several variable selection methods to binary logistic regression models. The methodology from Shi et al. (2010) is followed for applying penalty to binary models in the simulations. Park and Hastie (2007) proposed an algorithm for applying the L_1 penalty to generalized linear models. The algorithm that Park and Hastie (2007) proposed can be extended to the multinomial logistic regression model since the multinomial logistic regression model is a special case of generalized linear models. An application of the Elastic Net penalty to generalized linear models is described by Friedman, Hastie, and Tibshirani (2010), which allows the application of the Elastic Net penalty to the multinomial logistic regression model. Li (2011) applied the supSCAD penalty to the multinomial logistic regression model and Tutz, Pobnecker, and Uhlmann (2015) applied variable selection to general multinomial logistic regression models. The methodologies of these papers are followed when applying the variable shrinkage and selection methods to the binary and multinomial logistic regression models.

2 Methodology

The goal of this thesis is to identify the influential factors of ADHD using a penalized binary logistic regression model and a penalized multinomial logistic regression model using the phenotypic data from the ADHD-200 competition. This section outlines the methods for the simulation study and the real data analysis.

The method of variable shrinkage and selection used for determining the influential factors of the ADHD data was selected through simulation study. As stated earlier, traditional methods of variable selection can be computationally expensive and do not allow for changing the influence of explanatory variables. In traditional methods, a variable is either included or excluded, so this study will focus on methods that allow the influence of variables to be changed. The simulation was done

using R software (R Core Team, 2016). The simulation was applied for linear models and logistic regression models with correlated explanatory variables and uncorrelated explanatory simulation data. For each case, 1000 different data sets were generated with 100 observations per data set. The uncorrelated data sets were drawn from a standard normal distribution using the `rnorm` function from R Software. The correlated data sets were generated based on a method used by Zou (2006). These data sets are the explanatory variable $\mathbf{X}$ for the simulation. Additionally a true $\boldsymbol{\beta}$ was selected to be $\boldsymbol{\beta} = (1.5,\ 3,\ 0,\ 0,\ 2,\ 0,\ 0,\ 0)^T$ for the linear and binary logistic simulations. For the multinomial simulations, three true beta's were selected since the simulation is designed to have three possible outcomes. These beta's are $\boldsymbol{\beta}^{(1)}$ =(1.5, 3, 0, 0, 2, 0, 0, 0)^T, $\boldsymbol{\beta}^{(2)}$ =(2, 0, 2, 0, 0, 3, 0, 0)^T, and $\boldsymbol{\beta}^{(3)}$ =(3, 0, 0, 0, 1.5, 0, 2, 0)^T. This selection of $\boldsymbol{\beta}$ allows evaluating each method of shrinkage and selection on the method's ability to select the correct model.

2.1 Penalized Linear Model Simulations

For the linear simulation, a true response variable $\mathbf{y}$ was generated using the linear regression model, $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, in which the error $\boldsymbol{\epsilon}$ was generated using the `rnorm` function in R. After generating explanatory data $\mathbf{X}$ and calculating response $\mathbf{y}$ a linear model was computed using the LASSO penalty, Ridge Regression penalty, the adaptive LASSO penalty, the Elastic Net penalty, and no penalty (ordinary least squares regression). The AIC, BIC, CV, and GCV selection criteria were applied to each method that required a penalty in order to select the optimal penalty for determining the correct model. The accuracy of each model was calculated based on some predetermined performance measures. The performance measures are as follows: 1. average mean squared error (MSE) over 1000 simulations, 2. median MSE over 1000 simulations, 3. the average number of coefficients correctly shrunk to

zero (correctly = 0), 4. the average number of coefficients incorrectly selected for the model (incorrectly $\neq 0$), 5. the average number of coefficients correctly selected for the model(correctly $\neq 0$), 6. the average number of coefficients incorrectly shrunk to zero (incorrectly = 0), and 7. the proportion of correct models over 1000 data sets. The MSE is given by $(\boldsymbol{\beta} - \hat{\boldsymbol{\beta}})^T \mathbf{X}^T \mathbf{X} (\boldsymbol{\beta} - \hat{\boldsymbol{\beta}})$, where $\mathbf{X}^T\mathbf{X}$ gives the covariance matrix. If $\mathbf{X}$ is uncorrelated $\mathbf{X}^T\mathbf{X}$ should give the identity matrix.

Ideally (Oracle) the average MSE and median MSE for uncorrelated $\mathbf{X}$ are 0.0309 and 0.0246 respectively, the average MSE and median MSE for correlated $\mathbf{X}$ are 0.0293 and 0.0227 respectively, the average number of coefficients correctly shrunk to zero is five, the average number of coefficients incorrectly selected for the model is zero, the average number of coefficients correctly selected for the model is three, the average number of coefficients incorrectly shrunk to zero is zero, and the proportion of correct models is one.

The simulation data are used to compute an ordinary linear regression model without the application of a penalty. Table 2 shows the results of the ordinary least squares simulation using uncorrelated data and Table 3 shows the results of the simulation using correlated data. (NOTE: since there is no penalty in OLS regression, no selection criterion is applied to the model.)

Table 2: Linear Regression with Uncorrelated $\mathbf{X}$

Performance Measure:↓	Result	Oracle
Mean MSE:	0.0859	0.0309
Median MSE:	0.0779	0.0246
Correctly = 0:	0	5
Incorrectly $\neq 0$:	5	0
Correctly $\neq 0$:	3	3
Incorrectly = 0:	0	0
% of Correct Models:	0%	100%

Table 3: Linear Regression with Correlated $\mathbf{X}$

Performance Measure:↓	Result	Oracle
Mean MSE:	0.0858	0.0293
Median MSE:	0.0768	0.0227
Correctly = 0:	0	5
Incorrectly ≠ 0:	5	0
Correctly ≠ 0:	3	3
Incorrectly = 0:	0	0
% of Correct Models:	0%	100%

The Ridge Regression method applies the L_2 penalty to the OLS model. The ridge penalty can shrink the influence of variables in the model; however, it is unable to eliminate them entirely. The performance of the Ridge Regression simulation using uncorrelated explanatory variables is given in Table 4 and the performance using the correlated data sets is given in Table 5. The simulation results show that Ridge Regression really suffers from the inability to shrink coefficients to zero.

Table 4: Linear Regression with Uncorrelated $\mathbf{X}$ and Ridge Regression Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean MSE:	0.0854	0.0854	0.0855	0.0861	0.0309
Median MSE:	0.0769	0.0769	0.0768	0.0772	0.0246
Correctly = 0:	0	0	0	0	5
Incorrectly ≠ 0:	5	5	5	5	0
Correctly ≠ 0:	3	3	3	3	3
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	0%	0%	0%	0%	100%

Table 5: Linear Regression with Correlated **X** and Ridge Regression Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean MSE:	0.0855	0.0855	0.0857	0.0861	0.0293
Median MSE:	0.0771	0.0771	0.0773	0.0774	0.0227
Correctly = 0:	0	0	0	0	5
Incorrectly ≠ 0:	5	5	5	5	0
Correctly ≠ 0:	3	3	3	3	3
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	0%	0%	0%	0%	100%

The LASSO regression model applies the L_1 penalty to the model. Similar to Ridge Regression, the LASSO can shrink the influence of some of the variables by shrinking the coefficients. The advantage to using the LASSO over Ridge Regression is that the L_1 penalty can shrink some coefficients entirely to zero. However, as stated earlier, the LASSO has some limitations. The result of the simulation when applying the LASSO to the uncorrelated data sets is given in Table 6 and the result of applying the LASSO to the correlated data is given in Table 7. The simulation shows that the LASSO tends to select the correct model more frequently when subjected to the BIC. The MSE is slightly higher for the BIC for uncorrelated data and correlated data, but the difference in MSE between BIC and other selection criteria is small.

Table 6: Linear Regression with Uncorrelated **X** and LASSO Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean MSE:	0.0707	0.0791	0.0712	0.0705	0.0309
Median MSE:	0.0605	0.0676	0.0621	0.0610	0.0246
Correctly = 0:	2.849	4.037	2.461	2.888	5
Incorrectly ≠ 0:	2.151	0.963	2.539	2.112	0
Correctly ≠ 0:	3	3	3	3	3
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	12.7%	41.4%	8.8%	13.1%	100%

Table 7: Linear Regression with Correlated **X** and LASSO Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean MSE:	0.0656	0.0706	0.0659	0.0660	0.0293
Median MSE:	0.0564	0.0587	0.0564	0.0566	0.0227
Correctly = 0:	2.997	3.981	2.606	2.961	5
Incorrectly ≠ 0:	2.003	1.019	2.093	2.005	0
Correctly ≠ 0:	3	3	3	3	3
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	14.3%	33.7%	9.3%	14.2%	100%

The adaptive LASSO improves upon the LASSO method by adding weights to the coefficients as stated above. Adding weights to the coefficients allows coefficients that are more influential on the model to have more power. For the simulation the OLS coefficients were used to add weight to the model. The results from applying the adaptive LASSO to the uncorrelated data are given in Table 8, and the results from applying the adaptive LASSO to the correlated model are given in Table 9. The linear adaptive LASSO shows the BIC to be performing significantly better than

the other selection criteria in terms of the proportion of models correctly selected and tends to have a smaller MSE compared to other selection criteria. The adaptive LASSO greatly outperforms the LASSO in terms of selecting the correct model and minimizing MSE.

Table 8: Linear Regression with Uncorrelated $\mathbf{X}$ and Adaptive LASSO Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean MSE:	0.0535	0.0420	0.2350	0.0542	0.0309
Median MSE:	0.0424	0.0336	0.0437	0.0432	0.0246
Correctly = 0:	4.089	4.846	3.506	4.039	5
Incorrectly ≠ 0:	0.911	0.154	1.494	0.961	0
Correctly ≠ 0:	3	3	2.986	3	3
Incorrectly = 0:	0	0	0.014	0	0
% of Correct Models:	42.5%	86.2%	19.8%	40.9%	100%

Table 9: Linear Regression with Correlated $\mathbf{X}$ and Adaptive LASSO Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean MSE:	0.0522	0.0422	0.2701	0.0530	0.0293
Median MSE:	0.0429	0.0335	0.0442	0.0448	0.0227
Correctly = 0:	4.155	4.824	3.400	4.103	5
Incorrectly ≠ 0:	0.845	0.176	1.600	0.897	0
Correctly ≠ 0:	3	3	2.976	3	3
Incorrectly = 0:	0	0	0.024	0	0
% of Correct Models:	44.8%	84.3%	18.4%	43.5%	100%

The Elastic Net model applies both the L_1 penalty and the L_2 penalty. The use of the L_1 penalty helps generate a model with the most influential variables selected. The addition of the L_2 penalty helps regulate the shrinkage and selection which allows

the Elastic Net penalty to perform better when data are correlated. The results of applying the Elastic Net to the uncorrelated data set and the correlated data sets are given in Table 10 and Table 11, respectively. Again we see that the BIC tends to perform better than the other criteria when selecting a model and producing the minimum MSE. The Elastic Net has similar performance to the adaptive LASSO and outperforms the LASSO and Ridge Regression method.

Table 10: Linear Regression with Uncorrelated $\mathbf{X}$ and Elastic Net Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean MSE:	0.0524	0.0466	0.3247	0.0531	0.0309
Median MSE:	0.0425	0.0383	0.0724	0.0432	0.0246
Correctly = 0:	4.108	4.866	2.493	4.060	5
Incorrectly ≠ 0:	0.892	0.134	2.507	0.940	0
Correctly ≠ 0:	3	3	3	3	3
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	52.8%	88.4%	10.0%	51.7%	100%

Table 11: Linear Regression with Correlated $\mathbf{X}$ and Elastic Net Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean MSE:	0.0526	0.0468	0.0682	0.0532	0.0293
Median MSE:	0.0423	0.0367	0.0573	0.0427	0.0227
Correctly = 0:	4.064	4.729	2.489	4.041	5
Incorrectly ≠ 0:	0.936	0.271	2.511	0.959	0
Correctly ≠ 0:	3	3	3	3	3
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	48.4%	77.5%	7.7%	48.4%	100%

The simulation results show that the Elastic Net method and the adaptive LASSO

have similar performance and outperform other methods. The Elastic Net method outperforms adaptive LASSO for uncorrelated data, but the adaptive LASSO outperforms the Elastic Net when applied to correlated data. Across all methods tested in the simulation the BIC performed the best producing the highest number of correct model selections and the smallest MSE for a majority of the methods tested. In general, the adaptive LASSO and Elastic Net with the BIC tend to select the nonzero coefficients with high accuracy, but shrinking the zero coefficients is where each method struggles. The CV selection criterion tends to perform the worst overall, but tends to perform better on correlated data than on uncorrelated data for some methods. Ridge Regression and OLS regression struggle in performance due to their inability to shrink coefficients to zero. Also, it is notable that the methods that satisfy the oracle properties (adaptive LASSO and Elastic Net) performed significantly better than the methods that do not satisfy these properties.

2.2 Penalized Binary Logistic Model Simulations

For the binary logistic simulation a true response $\mathbf{y}$ was generated using the Bernoulli distribution.

$$\mathbf{y} \sim Bernoulli\left[\frac{e^{\mathbf{X}\boldsymbol{\beta}}}{1+e^{\mathbf{X}\boldsymbol{\beta}}}\right]$$

The Bernoulli distribution produces a categorical value of 0 or 1 based on the probability given by $(e^{\mathbf{X}\boldsymbol{\beta}})(1+e^{\mathbf{X}\boldsymbol{\beta}})^{-1}$. When $\mathbf{X}\boldsymbol{\beta}$ is large, the resulting $\mathbf{y}$ is more likely to be 1 and when $\mathbf{X}\boldsymbol{\beta}$ is small the resulting $\mathbf{y}$ is more likely to be 0. This allows the production of a categorical response from explanatory variables. The penalty methods applied to the binary logistic regression method are Ridge Regression, LASSO, and Elastic Net. In this thesis, adaptive LASSO is not applied to the binary logistic regression model because there is not sufficient literature supporting a method to do so. The binary logistic regression model without penalty is also considered. For each

method requiring a penalty, the AIC, BIC, CV, and GCV selection criteria are applied to determine the optimal penalty. The performance measures used to measure the accuracy of each method are given as follows: 1. average relative error (RE) over 1000 simulations, 2. median RE over 1000 simulations, 3. the average prediction error (PE), 4. the median PE, 5. the average number of coefficients correctly shrunk to zero, 6. the average number of coefficients incorrectly selected for the model, 7. the average number of coefficients correctly selected for the model, 8. the average number of coefficients incorrectly shrunk to zero, and 9. the proportion of correct models over 1000 data sets. The proportion of models which were over-fit is included because the performance is worse in the binary logistic case than for the linear case. The proportion of over-fitted models is calculated because it is better to over-fit than under-fit models. The RE is

$$\frac{||\boldsymbol{\beta} - \hat{\boldsymbol{\beta}}||_2}{||\boldsymbol{\beta}||_2} = \frac{\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j)^2}}{\sqrt{\sum_{j=1}^{p}\beta_j^2}},$$

where $\hat{\beta}_j$ is the jth estimated coefficient. The PE is the average classification error given by

$$\frac{\sum_{i=1}^{n}(\hat{y}_i \neq y_i)}{n},$$

where $\hat{y}$ is the estimated response, y is the true response, and n is the number of observations.

Ideally (Oracle) the average RE and median RE for uncorrelated $\mathbf{X}$ are 0.2646 and 0.2001 respectively, the average RE and median RE for correlated $\mathbf{X}$ are 0.2477 and 0.1968 respectively, the mean PE for uncorrelated $\mathbf{X}$ should be 0.1244, the median PE for uncorrelated $\mathbf{X}$ should be 0.1200, the mean PE for correlated $\mathbf{X}$ should be 0.1275, the median PE for correlated $\mathbf{X}$ should be 0.1300, the average number of coefficients correctly shrunk to zero is five, the average number of coefficients incorrectly selected

for the model is zero, the average number of coefficients correctly selected for the model is three, the average number of coefficients incorrectly shrunk to zero is zero, and the proportion of correct models is one. The proportion of over-fit models should be zero, but it is better to over-fit than to under-fit.

These simulation data are applied to a binary logistic regression model without application of a penalty term. Since there is no penalty applied, the selection criteria are not included in the tables. The performance of this method in terms of the number of correct models is poor due to the lack of a penalty. The result of the binary logistic simulation without penalty is given for uncorrelated data in Table 12 and correlated data in Table 13.

Table 12: Binary Logistic Regression with Uncorrelated $\mathbf{X}$

Performance Measure:↓	Result	Oracle
Mean RE:	0.7909	0.2646
Median RE:	0.3526	0.2001
Mean PE:	0.1108	0.1244
Median PE:	0.1100	0.1200
Correctly $= 0$:	0	5
Incorrectly $\neq 0$:	5	0
Correctly $\neq 0$:	3	3
Incorrectly $= 0$:	0	0
% of Correct Models:	0%	100%
% of Over-fit Models:	100%	0%

Table 13: Binary Logistic Regression with Correlated $\mathbf{X}$

Performance Measure:↓	Result	Oracle
Mean RE:	0.5497	0.2477
Median RE:	0.3491	0.1968
Mean PE:	0.1124	0.1275
Median PE:	0.1100	0.1300
Correctly $= 0$:	0	5
Incorrectly $\neq 0$:	5	0
Correctly $\neq 0$:	3	3
Incorrectly $= 0$:	0	0
% of Correct Models:	0%	100%
% of Over-fit Models:	100%	0%

The Ridge Regression method applies the L_2 penalty to the model. This method can shrink the influence of variables in the model, but is unable to shrink coefficients to zero. The performance of the Ridge Regression penalty is given in Table 14 for the uncorrelated data and in Table 15 for the correlated data. The simulation shows that Ridge Regression suffers by its inability to shrink coefficients to zero. Ridge Regression performs the same for AIC, BIC, and GCV in terms of RE and PE, but in terms of the proportion of correct models all selection criteria perform the same. Due to the inability to shrink coefficients to zero all models are over-fit.

Table 14: Binary Logistic Regression with Uncorrelated **X** with Ridge Regression Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.4826	0.4826	0.5004	0.4826	0.2646
Median RE:	0.4834	0.4834	0.5010	0.4834	0.2001
Mean PE:	0.1130	0.1130	0.1131	0.1130	0.1244
Median PE:	0.1100	0.1100	0.1100	0.1100	0.1200
Correctly = 0:	0	0	0	0	5
Incorrectly ≠ 0:	5	5	5	5	0
Correctly ≠ 0:	3	3	3	3	3
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	0%	0%	0%	0%	100%
% of Over-fit Models:	100%	100%	100%	100%	0%

Table 15: Binary Logistic Regression with Correlated **X** with Ridge Regression Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.4852	0.4852	0.5024	0.4852	0.2477
Median RE:	0.4851	0.4851	0.5032	0.4851	0.1968
Mean PE:	0.1143	0.1143	0.1145	0.1143	0.1275
Median PE:	0.1100	0.1100	0.1100	0.1100	0.1300
Correctly = 0:	0	0	0	0	5
Incorrectly ≠ 0:	5	5	5	5	0
Correctly ≠ 0:	3	3	3	3	3
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	0%	0%	0%	0%	100%
% of Over-fit Models:	100%	100%	100%	100%	0%

The LASSO regression method applies the L_1 penalty to the model. The LASSO shrinks the influence of coefficients similar to Ridge Regression, but it has the advantage that the L_1 penalty is capable of shrinking some coefficients to zero. The result of the simulation when applying the LASSO method to the uncorrelated data sets is given in Table 16 and the result when applying the LASSO to the correlated data is given in Table 17. The LASSO method, when applied to a binary logistic regression model tends to perform the best when the BIC is applied in terms of the number of correct models. The RE and PE are higher for the BIC than other criteria, but the number of models correctly selected is much larger for BIC than any other selection criterion. AIC and CV perform poorly for selecting the correct model, but produce lower RE and PE than BIC. The models are not under-fit.

Table 16: Binary Logistic Regression with Uncorrelated $\mathbf{X}$ with LASSO Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.4193	0.4253	0.3223	0.4536	0.2646
Median RE:	0.3201	0.3780	0.3099	0.3295	0.2001
Mean PE:	0.1116	0.1161	0.1136	0.1110	0.1244
Median PE:	0.1100	0.1100	0.1100	0.1100	0.1200
Correctly = 0:	1.759	3.320	2.099	1.159	5
Incorrectly ≠ 0:	3.241	1.680	2.100	3.841	0
Correctly ≠ 0:	3	2.999	3	3	3
Incorrectly = 0:	0	0.001	0	0	0
% of Correct Models:	4.8%	26.9%	4.3%	1.6%	100%
% of Over-fit Models:	95.2%	73.0%	95.7%	98.4%	0%

Table 17: Binary Logistic Regression with Correlated **X** with LASSO Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.3870	0.4021	0.3186	0.4196	0.2477
Median RE:	0.3152	0.3829	0.3093	0.3265	0.1968
Mean PE:	0.1131	0.1177	0.1155	0.1126	0.1275
Median PE:	0.1100	0.1200	0.1100	0.1100	0.1300
Correctly = 0:	1.686	3.308	2.027	1.097	5
Incorrectly ≠ 0:	3.314	1.692	2.937	3.903	0
Correctly ≠ 0:	3	2.999	2.997	3	3
Incorrectly = 0:	0	0.001	0.003	0	0
% of Correct Models:	4.6%	25.1%	3.1%	1.1%	100%
% of Over-fit Models:	95.4%	74.8%	96.9%	98.9%	0%

The Elastic Net method applies both the L_1 penalty and the L_2 penalty to the model. The use of the L_1 penalty helps to select only the variables which are influential to the model. The addition of the L_2 penalty helps regulate the shrinkage and selection which allows the Elastic Net to perform better when the data are correlated. The result of applying the Elastic Net to the uncorrelated data is given in Table 18 and the result when applying the Elastic Net penalty to the correlated data is given in Table 19. The Elastic Net penalty performs the best when subjected to the BIC. The BIC selects the correct model the most frequently and has relatively low RE and PE. AIC and CV have poor performance for selecting the correct model, but both produce a lower RE and PE than BIC. The Elastic Net performs similarly to the LASSO because many of the simulations selected the L_1 penalty only. No models are under-fit.

Table 18: Binary Logistic Regression with Uncorrelated **X** with Elastic Net Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.4212	0.4282	0.3280	0.4549	0.2646
Median RE:	0.3183	0.3783	0.3153	0.3305	0.2001
Mean PE:	0.1115	0.1161	0.1134	0.1110	0.1244
Median PE:	0.1100	0.1100	0.1100	0.1100	0.1200
Correctly = 0:	1.763	3.314	1.9848	1.160	5
Incorrectly ≠ 0:	3.237	1.686	3.052	3.840	0
Correctly ≠ 0:	3	2.999	3	3	3
Incorrectly = 0:	0	0.001	0	0	0
% of Correct Models:	4.9%	26.8%	4.2%	1.7%	100%
% of Over-fit Models:	95.1%	73.1%	95.8%	98.3%	0%

Table 19: Binary Logistic Regression with Correlated **X** with Elastic Net Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.3870	0.4056	0.3242	0.4196	0.2477
Median RE:	0.3160	0.3827	0.3218	0.3251	0.1968
Mean PE:	0.1131	0.1176	0.1145	0.1126	0.1275
Median PE:	0.1100	0.1200	0.1100	0.1100	0.1300
Correctly = 0:	1.690	3.303	1.880	1.095	5
Incorrectly ≠ 0:	3.310	1.697	3.120	3.905	0
Correctly ≠ 0:	3	2.999	3	3	3
Incorrectly = 0:	0	0.001	0	0	0
% of Correct Models:	4.8%	25.1%	3.1%	1.0%	100%
% of Over-fit Models:	95.2%	74.8%	96.9%	99.0%	0%

When considering the binary logistic regression model, the LASSO method tends to perform the best. When the LASSO penalty is applied, the selection criterion which

performs the best is BIC. When the BIC is applied to the LASSO, the proportion of correct models selected is significantly higher than all other selection criteria. The Elastic Net performs similarly to the LASSO, but selects the correct model slightly less frequently. AIC and CV have poor performance for selecting the correct model, but have a lower RE and PE. CV is considered in data analysis because of the low RE and PE. The inability of Ridge Regression and the unpenalized model to shrink coefficients to zero hurts their performance of selection.

2.3 Penalized Multinomial Logistic Model Simulations

For the multinomial logistic simulation a true response $\mathbf{y}$ is generated by calculating $t_1 = e^{\mathbf{X}\boldsymbol{\beta}^{(1)}}$, $t_2 = e^{\mathbf{X}\boldsymbol{\beta}^{(2)}}$, and $t_3 = e^{\mathbf{X}\boldsymbol{\beta}^{(3)}}$, assuming that there are three categories. The probabilities for each categorical outcome are generated by $p_1 = t_1/(t_1+t_2+t_3)$, $p_2 = t_2/(t_1+t_2+t_3)$, and $p_3 = t_3/(t_1+t_2+t_3)$. Then using the `rmultinomial` function from `multinomRob` package and `which.max` function in R, a value of 1, 2, or 3 is placed in $\mathbf{y}$. The values in $\mathbf{y}$ are decided by the maximum probability generated in p_1, p_2, and p_3. This method generates categorical values for $\mathbf{y}$, but allows for there to be more than two categories. Defining $\boldsymbol{\gamma}^{(2)} = \boldsymbol{\beta}^{(2)} - \boldsymbol{\beta}^{(1)}$ and $\boldsymbol{\gamma}^{(3)} = \boldsymbol{\beta}^{(3)} - \boldsymbol{\beta}^{(1)}$, the model framework in `glmnet` can be reparameterized in the form of the generalized linear model given by

$$\log\left(\frac{\pi_2(\mathbf{x})}{\pi_1(\mathbf{x})}\right) = \mathbf{x}^T\boldsymbol{\gamma}^{(2)}$$

and

$$\log\left(\frac{\pi_3(\mathbf{x})}{\pi_1(\mathbf{x})}\right) = \mathbf{x}^T\boldsymbol{\gamma}^{(3)}.$$

The penalty methods applied to the multinomial logistic regression method are Ridge Regression, LASSO, and Elastic Net. The adaptive LASSO is not applied to the multinomial logistic regression model because there is not sufficient literature

supporting a method to do so. The multinomial logistic regression model without penalty is also considered. For each method requiring a penalty, the AIC, BIC, CV, and GCV are applied to determine the optimal penalty. The performance measures used to measure the accuracy of each method are given as follows: 1. average relative error (RE) over 1000 simulations, 2. median RE over 1000 simulations, 3. the average prediction error (PE), 4. median PE over 1000 simulations 5. the average number of coefficients correctly shrunk to zero, 6. the average number of coefficients incorrectly selected for the model, 7. the average number of coefficients correctly selected for the model, 8. the average number of coefficients incorrectly shrunk to zero, and 9. the proportion of correct models over 1000 data sets. Additionally the proportion of over-fit models is included since the performance is poor and it is better to over-fit models than to under-fit models.

Ideally (Oracle) the average RE and median RE for uncorrelated $\mathbf{X}$ are 0.4413 and 0.3615 respectively, the average RE and median RE for correlated $\mathbf{X}$ are 0.3835 and 0.3358 respectively, the average PE and median PE for uncorrelated $\mathbf{X}$ are 0.1483 and 0.1500 respectively, the average PE and median PE for correlated $\mathbf{X}$ are 0.1992 and 0.2000 respectively, the average number of coefficients correctly shrunk to zero is four, the average number of coefficients incorrectly selected for the model is three, the average number of coefficients correctly selected for the model is nine, the average number of coefficients incorrectly shrunk to zero is zero, and the proportion of correct models is 0, and the proportion of over-fit models is one. The oracle model is over-fit because nonzero coefficients are not consistent between $\boldsymbol{\beta}^{(1)}$, $\boldsymbol{\beta}^{(2)}$, and $\boldsymbol{\beta}^{(3)}$. Ideally the proportion of over-fit models is zero, however it is better to over-fit than to under-fit. Let $\boldsymbol{\gamma} = (\boldsymbol{\gamma}^{(2)}, \boldsymbol{\gamma}^{(3)})^T$. The RE is

$$\frac{||\boldsymbol{\gamma} - \hat{\boldsymbol{\gamma}}||_2}{||\boldsymbol{\gamma}||_2} = \frac{\sqrt{\sum_{i=2}^{3}\sum_{j=1}^{p}(\gamma_j^{(i)} - \hat{\gamma}_j^{(i)})^2}}{\sqrt{\sum_{i=2}^{3}\sum_{j=1}^{p}(\gamma_j^{(i)})^2}},$$

where $\hat{\gamma}_j^{(i)}$ is the jth estimated coefficient for the ith category. The PE is

$$\frac{\sum_{i=1}^{n}(\hat{y}_i \neq y_i)}{n},$$

where $\hat{y}_i$ is the estimated response and n is the number of observations.

These simulation data are applied to a multinomial logistic regression model with three categorical outcomes without applying a penalty term. Since there is no penalty term applied in this case, the selection criteria are not included in the tables. The unpenalized method suffers when trying to identify the correct coefficients due to the inability to shrink coefficients to zero. The result of applying the unpenalized multinomial logistic model to the uncorrelated data is given in Table 20, and the result from the correlated data is given in Table 21.

Table 20: Multinomial Logistic Regression with Uncorrelated $\mathbf{X}$

Performance Measure:↓	Result	Oracle
Mean RE:	0.6269	0.4413
Median RE:	0.4832	0.3615
Mean PE:	0.1360	0.1483
Median PE:	0.1300	0.1500
Correctly = 0:	0	4
Incorrectly ≠ 0:	7	3
Correctly ≠ 0:	9	9
Incorrectly = 0:	0	0
% of Correct Models:	0%	0%
% of Over-fit Models:	100%	100%

Table 21: Multinomial Logistic Regression with Correlated $\mathbf{X}$

Performance Measure:↓	Result	Oracle
Mean RE:	0.4957	0.3835
Median RE:	0.4296	0.3358
Mean PE:	0.1879	0.1992
Median PE:	0.1900	0.2000
Correctly $= 0$:	0	4
Incorrectly $\neq 0$:	7	3
Correctly $\neq 0$:	9	9
Incorrectly $= 0$:	0	0
% of Correct Models:	0%	0%
% of Over-fit Models:	100%	100%

The Ridge Regression method applies the L_2 penalty to the multinomial logistic regression model. This method can shrink the influence of variables in the model, but suffers in terms of identifying influential factors because the Ridge Regression method is unable to shrink coefficients to zero. The performance of the Ridge Regression on the multinomial logistic regression model when using the correlated data is given in Table 22. For the correlated data, the result is given in Table 23. AIC, BIC and GCV produce the same RE and PE. CV performs worse with a larger RE and PE than the other selection criteria. All of the models are over-fit because coefficients cannot be shrunk to zero.

Table 22: Multinomial Logistic Regression with Uncorrelated **X** and Ridge Regression Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.4455	0.4455	0.4633	0.4455	0.4413
Median RE:	0.4450	0.4450	0.4630	0.4450	0.3615
Mean PE:	0.1432	0.1432	0.1438	0.1432	0.1483
Median PE:	0.1400	0.1400	0.1400	0.1400	0.1500
Correctly = 0:	0	0	0	0	4
Incorrectly ≠ 0:	7	7	7	7	3
Correctly ≠ 0:	9	9	9	9	9
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	0%	0%	0%	0%	0%
% of Over-fit Models:	100%	100%	100%	100%	100%

Table 23: Multinomial Logistic Regression with Correlated **X** and Ridge Regression Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.4428	0.4428	0.4658	0.4428	0.3835
Median RE:	0.4419	0.4419	0.4646	0.4419	0.3358
Mean PE:	0.1958	0.1958	0.1973	0.1958	0.1992
Median PE:	0.2000	0.2000	0.2000	0.2000	0.2000
Correctly = 0:	0	0	0	0	4
Incorrectly ≠ 0:	7	7	7	7	3
Correctly ≠ 0:	9	9	9	9	9
Incorrectly = 0:	0	0	0	0	0
% of Correct Models:	0%	0%	0%	0%	0%
% of Over-fit Models:	100%	100%	100%	100%	100%

The LASSO regression method applies the L_1 penalty to the multinomial logistic regression model. The LASSO penalty shrinks the influence of coefficients similar to the Ridge Regression, but the L_1 penalty is capable of shrinking coefficients to zero. The result of the simulation when applying the LASSO penalty to the multinomial logistic regression model for uncorrelated data is given in Table 24, and for the correlated data the result is given in Table 25. The BIC performs the best when the L_1 is applied to the multinomial logistic regression for uncorrelated and correlated $\mathbf{X}$. The BIC produces the highest proportion of correct models. However, BIC performs pretty poorly selecting less than 1% of models correctly. CV has the lowest RE and PE among the selection criterion. AIC and GCV over-fit the models more frequently than BIC and CV. BIC under-fits over 40% of the models.

Table 24: Multinomial Logistic Regression with Uncorrelated $\mathbf{X}$ and LASSO Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.5119	0.4616	0.3351	0.5449	0.4413
Median RE:	0.3715	0.3933	0.3339	0.3911	0.3615
Mean PE:	0.1366	0.1425	0.1421	0.1366	0.1483
Median PE:	0.1400	0.1400	0.1400	0.1400	0.1500
Correctly = 0:	1.045	2.650	1.894	0.918	4
Incorrectly ≠ 0:	5.955	4.350	5.106	6.082	3
Correctly ≠ 0:	8.847	8.585	8.748	8.872	9
Incorrectly = 0:	0.153	0.415	0.252	0.128	0
% of Correct Models:	0%	0.8%	0%	0%	0%
% of Over-fit Models:	85.6%	65.1%	76.7%	87.9%	100%

Table 25: Multinomial Logistic Regression with Correlated **X** and LASSO Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.4050	0.4178	0.3410	0.4197	0.3835
Median RE:	0.3475	0.3882	0.3387	0.3610	0.3358
Mean PE:	0.1895	0.1961	0.1942	0.1898	0.1992
Median PE:	0.1900	0.2000	0.1900	0.1900	0.2000
Correctly = 0:	1.332	2.917	1.972	1.458	4
Incorrectly ≠ 0:	5.668	4.083	5.028	5.542	3
Correctly ≠ 0:	8.806	8.470	8.712	8.791	9
Incorrectly = 0:	0.194	0.530	0.288	0.209	0
% of Correct Models:	0.1%	0.3%	0.1%	0.1%	0%
% of Over-fit Models:	81.5%	57.6%	73.0%	80.3%	100%

The Elastic Net penalty for multinomial logistic regression applies both the L_1 penalty and the L_2 penalty to the regression model. The L_1 penalty selects the influential variables of the model, while the L_2 penalty regulates the shrinkage and selection which helps the Elastic Net penalty perform better for correlated data. The result of applying the Elastic Net penalty to the uncorrelated data is given in Table 26, and from applying the Elastic Net penalty to the correlated data is given in Table 27. The Elastic Net penalty performs the best for selecting the correct model when subjected to the BIC for both the correlated and uncorrelated cases. However, the Elastic Net subjected to the BIC performs poorly. Fewer than 1% of models are selected correctly under BIC. The performance of the Elastic Net is similar to that of the LASSO. However, the Elastic Net appears to perform better than the LASSO because the RE and PE are smaller for the Elastic Net with BIC than the LASSO with BIC when compared to the other selection criteria. The Elastic Net under CV produces the lowest RE and PE. The Elastic Net over-fits the models similarly to the

LASSO.

Table 26: Multinomial Logistic Regression with Uncorrelated **X** and Elastic Net Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.5106	0.4620	0.3402	0.5439	0.4413
Median RE:	0.3720	0.3930	0.3376	0.3918	0.3615
Mean PE:	0.1366	0.1425	0.1415	0.1366	0.1483
Median PE:	0.1400	0.1400	0.1400	0.1400	0.1500
Correctly = 0:	1.050	2.658	1.602	0.925	4
Incorrectly ≠ 0:	5.950	4.342	5.398	6.075	3
Correctly ≠ 0:	8.846	8.586	8.785	8.871	9
Incorrectly = 0:	0.154	0.414	0.215	0.129	0
% of Correct Models:	0%	0.8%	0%	0%	0%
% of Over-fit Models:	85.5%	65.2%	80.3%	87.8%	100%

Table 27: Multinomial Logistic Regression with Correlated **X** and Elastic Net Penalty

Selection Criterion:→ Performance Measure:↓	AIC	BIC	CV	GCV	Oracle
Mean RE:	0.4052	0.4170	0.3473	0.4192	0.3835
Median RE:	0.3477	0.3872	0.3435	0.3605	0.3358
Mean PE:	0.1896	0.1962	0.1936	0.1898	0.1992
Median PE:	0.1900	0.2000	0.1900	0.1900	0.2000
Correctly = 0:	1.337	2.933	1.652	1.470	4
Incorrectly ≠ 0:	5.663	4.067	5.348	5.530	3
Correctly ≠ 0:	8.807	8.467	8.758	8.790	9
Incorrectly = 0:	0.193	0.533	0.242	0.210	0
% of Correct Models:	0.1%	0.2%	0.1%	0.1%	0%
% of Over-fit Models:	81.6%	57.4%	77.3%	80.2%	100%

The Elastic Net method performs the best for multinomial logistic regression. When subjected to the BIC, the Elastic Net method was able to select the correct model the most frequently. The LASSO correctly selected models similarly to the Elastic Net and produced similar RE and PE, but since the Elastic Net satisfies oracle properties this method will be used for multinomial data analysis. CV produced the lowest RE and PE for Elastic Net. The Elastic Net subjected to BIC and Elastic Net subjected to CV are used for data analysis because these methods performed the best in the simulation study.

2.4 Data Analysis Outline

The results of the simulation study show which penalization method paired with a selection criterion performs the best based on the performance measures. The penalization methods with the best performance based on the results of the simulation study are applied to the NYU data set to fit a binary logistic regression model and a multinomial logistic regression model.

The binary logistic regression model is fit with typically developing children (TDC) and children diagnosed with ADHD as the two possible outcomes. For multinomial logistic regression, the four outcomes are TDC, ADHD-Combined (ADHD-C), ADHD-Hyperactive/Impulsive (ADHD-H), and ADHD-Inattentive (ADHD-I). However, due to minimal observations of ADHD-H diagnosis in the NYU data set these observations will be excluded from the model in this thesis. For both the binary and multinomial logistic regression models, there are initially 9 explanatory variables considered for the model. These variables are gender, age, handedness, ADHD index, inattentive measure, hyperactive/impulsive measure, verbal IQ measure, performance IQ measure, and full4 IQ.

Prior to fitting the data to a model, the distributions of explanatory variables

in each response category are compared using box plots for continuous explanatory data and bar plots for categorical explanatory data. Comparing the distribution of explanatory variables within each category provides an initial look at which variables differ the most between categories. The variables that show significantly different distributions between the two categories are likely to be influential for differentiating between categories within the response. In addition to the plots, statistical tests are used to test for difference between categories.

In the binary case, the significance of the differences of explanatory variables within response categories is tested using a χ^2 test for categorical explanatory variables and a student's t-test for continuous explanatory variables. For the multinomial case, the significance of categorical explanatory variables between response categories is tested using a χ^2 test, and for the continuous explanatory variables an analysis of variance (ANOVA) test is used for testing differences. With all types of tests, a p-value $< .05$ implies that there is a significant difference of explanatory variables among categories. For both multinomial and binary logistic regression models without penalty, a goodness-of-fit test is applied to test if the model is appropriate and the significance of each variable in the model is tested. A t-test is used for the binary model and an ANOVA F test is used in the multinomial model. Again, a p-value $< .05$ means that the model is good.

The accuracy of the models is tested by calculating a classification error. The classification error is computed by comparing the true response to the predicted response from the model and dividing by the number of observations. The error is the average number of mis-classifications over the set of explanatory data that are being tested. The classification error is computed for both the test set and the training set. The training set is the set of data used to fit the model and the test set is an alternative set of data from the NYU data set used for testing the models.

Subjecting the binary and multinomial logistic regression models to a variable shrinkage and selection method will reduce the influence of some of the explanatory variables. Some of the explanatory factors may have their influence reduced to zero removing them from the model. Applying a penalty to the model makes significance testing of the model difficult, so the classification error is the only form of testing that is applied to penalized models. The simulation has shown that the BIC or CV applied to the Elastic Net penalty provides the best results for multinomial logistic regression, and that BIC applied to the LASSO penalty produces the best results for the binary logistic model. However, the classification error is calculated for each penalty subjected to each selection criterion. The explanatory variables that remain present after applying the shrinkage and selection methods are considered to be influential on the model.

3 Results

3.1 Binomial Logistic Data Analysis

Comparing explanatory data between response categories provides basic information on how each variable differs between categories. Figure 2 compares the number of males against the number of females in the TDC versus the ADHD. Figure 2 shows that there are more females in the TDC category than males and that there are more males in the ADHD. Figure 2 shows that there might be a significant difference between the number of males and number of females diagnosed with ADHD.

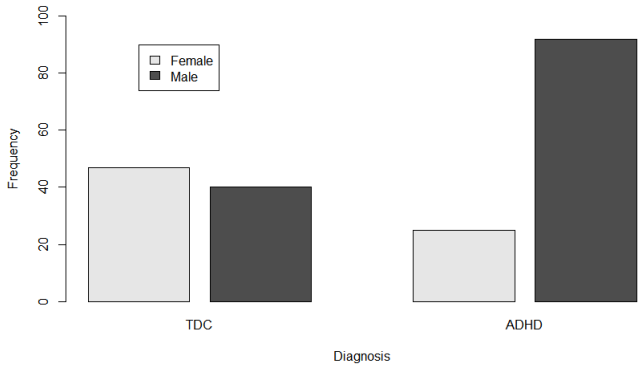

Figure 2: Comparison of gender between TDC and ADHD.

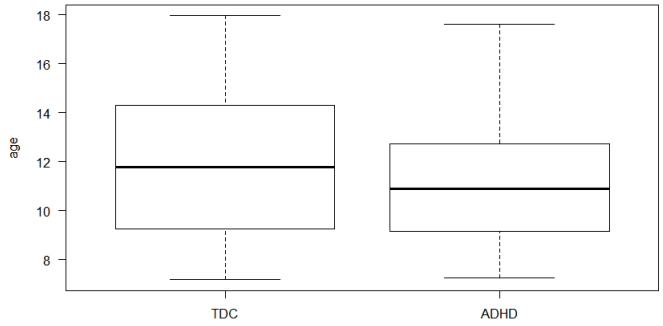

Figure 3: Comparison of age between TDC and ADHD.

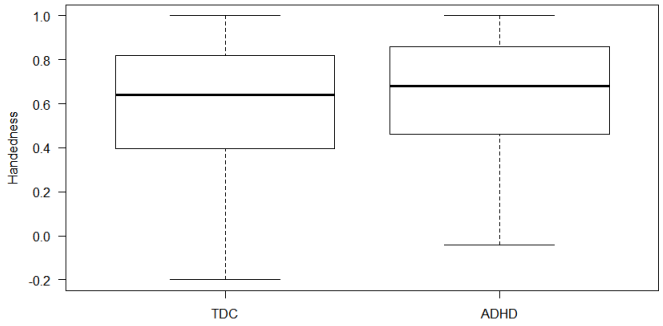

Figure 4: Comparison of handedness between TDC and ADHD.

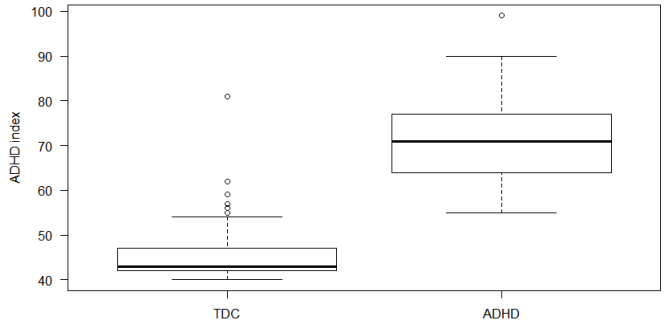

Figure 5: Comparison of ADHD Index measure between TDC and ADHD.

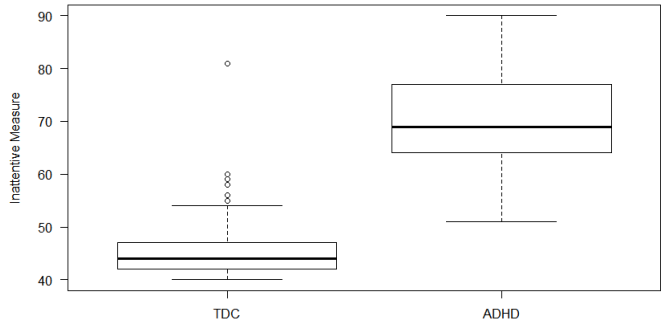

Figure 6: Comparison of Inattentive measure between TDC and ADHD.

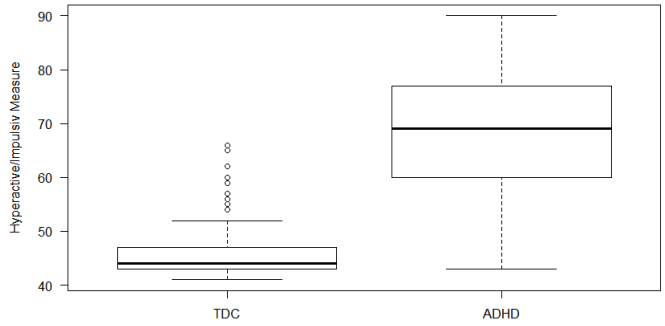

Figure 7: Comparison of Hyperactive/Impulsive measure between TDC and ADHD.

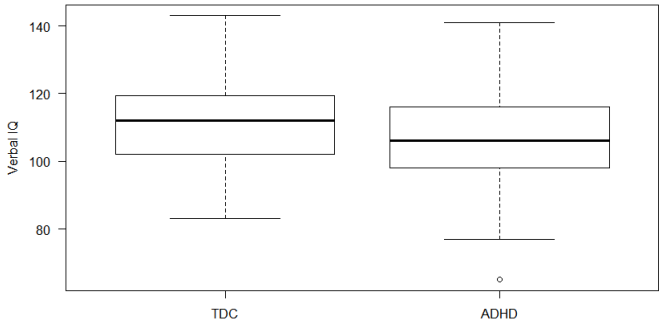

Figure 8: Comparison of Verbal IQ between TDC and ADHD.

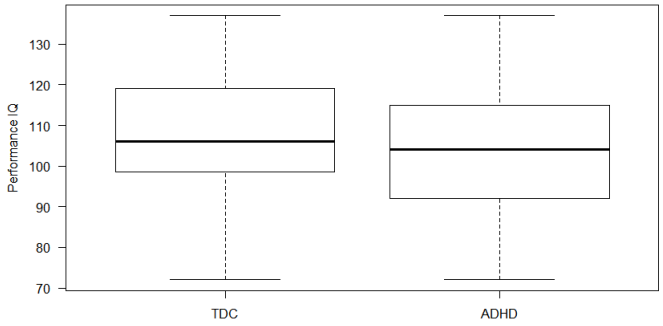

Figure 9: Comparison of Performance IQ between TDC and ADHD.

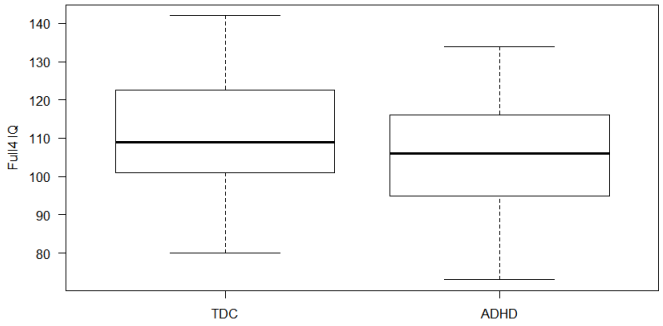

Figure 10: Comparison of Full4 IQ between TDC and ADHD.

Figures 3 through 10 compare the distribution of each of the explanatory variables from the data set between the two categories: TDC and ADHD using box plots. For age and handedness, the distributions between the two categories seem fairly similar. The ADHD group is slightly skewed towards the younger ages and the age range of the middle 50% of the ADHD group is more narrow than that of the TDC group. The distribution of handedness between the two groups is relatively similar. Both the TDC and ADHD groups have their distributions skewed towards being right handed with a mean of about 0.6. The ADHD index, inattentive measure, and hyperactive/impulsive measure show significantly different distributions between TDC and ADHD. For all three variables, the TDC distribution remains entirely between 40 and 60 and is skewed heavily towards the lower measures aside from a few outliers. For these variables, the ADHD is more normal in shape and tends to hold higher values than the TDC. The difference between distributions among categories implies that ADHD index, inattentive measure, and hyperactive/impulsive measure are influential in ADHD diagnosis. The three IQ measures show similar distributions between the two categories. The ADHD group appears to have a slightly lower average within the three groups, but there does not appear to be a significant difference of distribution of the three IQ measures between the categories.

Table 28: Significance Test Results for Variables in TDC vs. ADHD

	Gender	Age	Handedness	ADHD Index	Inattentive
P-Value:	< 0.05	0.0678	0.0858	< 0.05	< 0.05
	Hyperactive /Impulsive	Verbal IQ	Performance IQ	Full 4 IQ	
P-Value:	< 0.05	0.0315	0.1157	0.0475	

Table 28 shows the results of significance tests for each variable. For each variable the test determines if there is a significant difference between the two response categories. If a p-value is $< .05$, then the difference is considered significant. Since gender

is a categorical variable, the significance of gender was tested using a χ^2 test. Since all other explanatory variables are continuous, the significance was tested with a t-test. The p-values for gender, ADHD index, inattentive measure, hyperactive/impulsive measure, verbal IQ, and full4 IQ are considered to be significant. This means that there is a significant difference in gender, ADHD index, inattentive measure, hyperactive/impulsive measure, verbal IQ, and full4 IQ between the TDC and ADHD. The variables that are considered to be significant are likely to be selected in the binary logistic regression model.

Table 29: Significance Test Results from the Binary Logistic Regression Model

	Gender	Age	Handedness	ADHD Index	Inattentive
P-Value:	0.0213	0.9506	0.9352	0.1844	0.1761
	Hyperactive /Impulsive	Verbal IQ	Performance IQ	Full 4 IQ	
P-Value:	0.0849	0.6012	0.4552	0.5128	

Table 29 shows the significance test results for individual $\boldsymbol{\beta}$ values in the model after fitting an unpenalized binary logistic regression model to the data. These beta values are tested against zero. Again, a p-value < 0.05 is considered to be significant. A coefficient that is considered to be significant means that the coefficient is significantly different from zero. This test shows that only the coefficient of gender is considered to be significantly different from zero. However, applying an analysis of variance (ANOVA) goodness-of-fit test to the model, the resulting p-value is < 0.05 which means that the model is appropriate. The classification error is calculated by using the model to predict a response category and calculating the proportions of incorrect classifications. The classification error of the training set (the set of data used to fit the model) is 0.0147 which means that about 1.5% of classifications are incorrect for the training set. The classification error of the test set (a set of data used for testing the model) is 0.0286 which means that about 2.9% of classifications

are made incorrectly. Table 30 shows the predicted response when applying the test set and the training set to the fitted model. The model incorrectly identified one individual in the training set who is in the ADHD category as an individual in the TDC, and the model incorrectly identified three individuals as diagnosed with ADHD when they are in the TDC category. Overall, this model performs quite well. Applying the model to the test set and the training set both showed a small classification error, meaning that the model is likely to classify an observation correctly.

Table 30: Predicted Response for Test Set and Training Set from the Unpenalized Binary Logistic Regression Model

Test Set			Training Set		
	True			True	
Prediction	TDC	ADHD	Prediction	TDC	ADHD
TDC	10	1	TDC	84	0
ADHD	0	24	ADHD	3	117

Applying a penalty to the model helps determine which factors are influential in ADHD diagnosis. Here the LASSO penalty, when subjected to the BIC and CV, are analyzed. The ADHD data analysis for the binary logistic regression model will focus on the LASSO penalty where the penalty is selected by the BIC or the CV because the simulation study showed that the LASSO performs the best when applied to a binary logistic regression model. The `glmnet` function from the `glmnet` library in R is used to fit the model. For binomial models, `glmnet` creates one set of coefficients, $\boldsymbol{\beta}$. Significance tests for penalized models cannot be computed so the analysis will focus on classification error and which variables are selected for the model.

The LASSO penalization method selects gender, ADHD index, inattentive measure, and hyperactive/impulsive measure as influential variables when subjected to the BIC. This means that gender, ADHD index, inattentive measure, and hyperac-

tive/impulsive measure are considered to be influential variables, and the variables that were shrunk to zero such as age and handedness are not influential. It is not surprising that the ADHD measures were selected for the model, but it is interesting that gender was selected. The classification error when subjecting the LASSO to the BIC is 0.0286 when the test set of explanatory variables is applied to the model. The classification error for the training set is 0.0147. Table 31 shows the predicted response for the test set and the training set. This table shows the correct classifications and the mis-classifications for each set of data. It is notable that the mis-classification rates are the same for this penalized model and the unpenalized model above. In addition, each model appears to make the same mis-classification for both sets of data.

Table 31: Predicted Response for Test Set and Training Set with LASSO penalty and BIC

Test Set			Training Set		
	True			True	
Prediction	TDC	ADHD	Prediction	TDC	ADHD
TDC	10	1	TDC	84	0
ADHD	0	24	ADHD	3	117

Subjecting the LASSO penalty to the CV selects ADHD index and inattentive measure as the influential variables. For this particular model selection, the remaining variables are considered not to be influential. This model is simpler than the model selected by the BIC. In general, simpler models are preferred if providing similar performance. The test set gives a classification error of 0.0286, and the training set gives a classification error of 0.0392. The predicted response is compared to the true response in Table 32. Table 32 shows that the model fitted based on the CV has the same mis-classification as the model fitted with BIC and the unpenalized

model for the test set, however, the model fitted based on the CV made more mis-classifications than the other two fitted models. This model seems to have more trouble differentiating between TDC and ADHD than previous models. Although the model selected by CV is simpler than the model selected by the BIC, the model produces more mis-classifications.

Table 32: Predicted Response for Test Set and Training Set with LASSO penalty and CV Selection Criterion

Test Set			Training Set		
	True			True	
Prediction	TDC	ADHD	Prediction	TDC	ADHD
TDC	10	1	TDC	81	2
ADHD	0	24	ADHD	6	115

3.2 Multinomial Logistic Data Analysis

For multinomial data analysis, the goal is to identify which factors are influential for classifying each response category. The three categories are 0 - TDC, 1 - ADHD-C, and 3 - ADHD-I. The gender distribution within the three categories is shown in Figure 11. Figure 11 shows that both ADHD sub-categories are predominantly male. The TDC group is predominantly female. There appears to be a difference in gender between TDC and the other two categories, but there does not appear to be a significant difference in gender between the two ADHD categories. Figure 12 shows the comparison of the distribution of age between the three categories. There does not appear to be a large age difference between the three categories. ADHD-C is skewed towards the lower ages and both the ADHD subcategories have a lower median age than the TDC category. Figure 13 shows the distribution of handedness between the three categories. All three distributions are skewed towards being right

handed. There does not appear to be a significant difference in handedness between the three categories. The ADHD-I category has less variation than the other two categories.

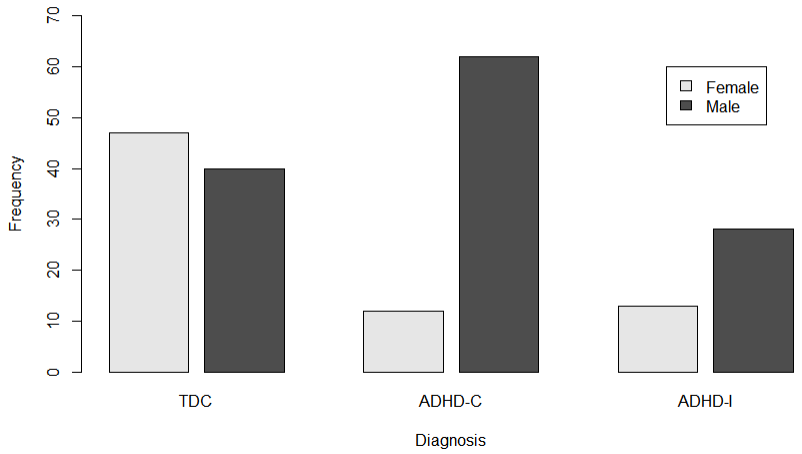

Figure 11: Comparison of gender between TDC, ADHD-C, and ADHD-I.

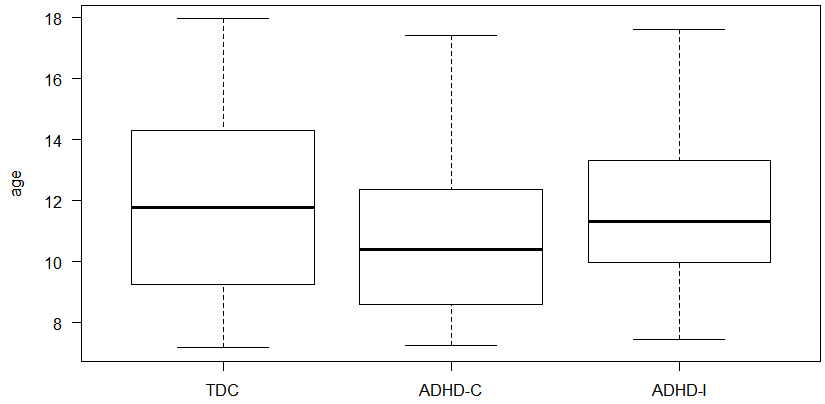

Figure 12: Comparison of age between TDC, ADHD-C, and ADHD-I.

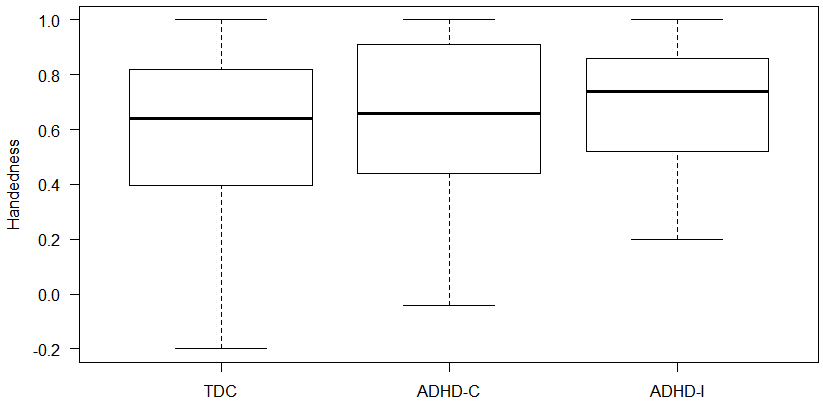

Figure 13: Comparison of handedness between TDC, ADHD-C, and ADHD-I.

Figures 14, 15, and 16 display the distributions within the three categories for ADHD index, inattentive measure, and hyperactive/impulsive measure respectively. For each of these measures, the TDC is heavily skewed towards the lower values, and aside from a few outliers the distribution is entirely under a measure of sixty. There appears to be a significant difference between TDC and the ADHD subtypes for all three of these measures. The subcategories of ADHD show relatively similar distributions for the inattentive measure and the ADHD index measure; however, the distribution of ADHD-C and ADHD-I for the hyperactive/impulsive measure look significantly different. This means that the hyperactive/impulsive measure might be influential for differentiating between ADHD-C and ADHD-I.

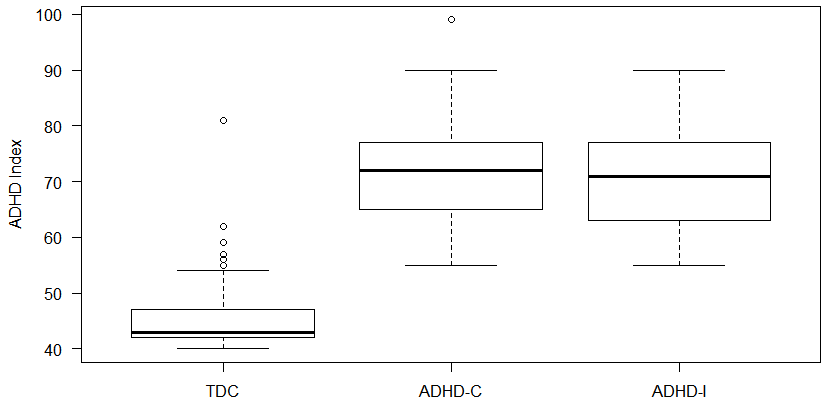

Figure 14: Comparison of ADHD Index measure between TDC, ADHD-C, and ADHD-I.

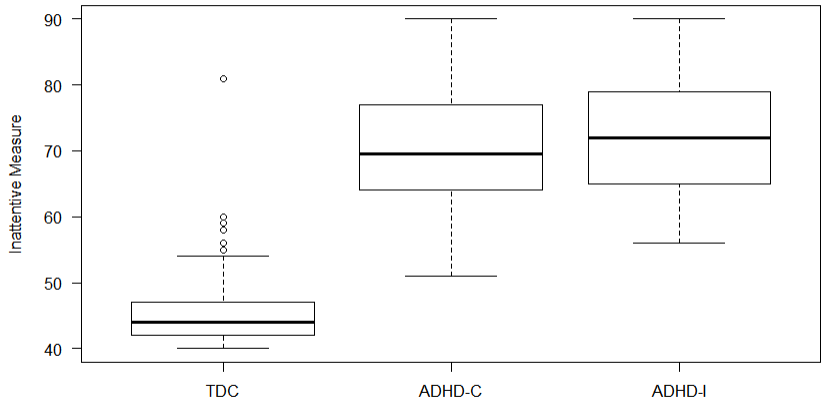

Figure 15: Comparison of inattentive measure between TDC, ADHD-C, and ADHD-I.

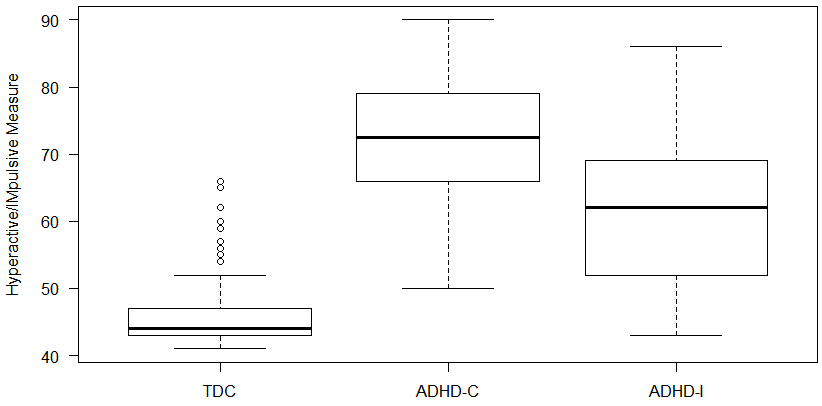

Figure 16: Comparison of Hyperactive/Impulsive measure between TDC, ADHD-C, and ADHD-I.

Figures 17, 18, and 19 show the distribution of the data within each of the three categories for the verbal IQ, performance IQ, and full4 IQ. The distribution of the data within each category is similar for all three IQ measures. Both ADHD-C and ADHD-I show a slightly lower median than TDC, and ADHD-C has a slightly lower median than the ADHD-I, but the difference does not appear to be significant. It does not appear that there is a significant difference in distribution for each category for the three IQ measures.

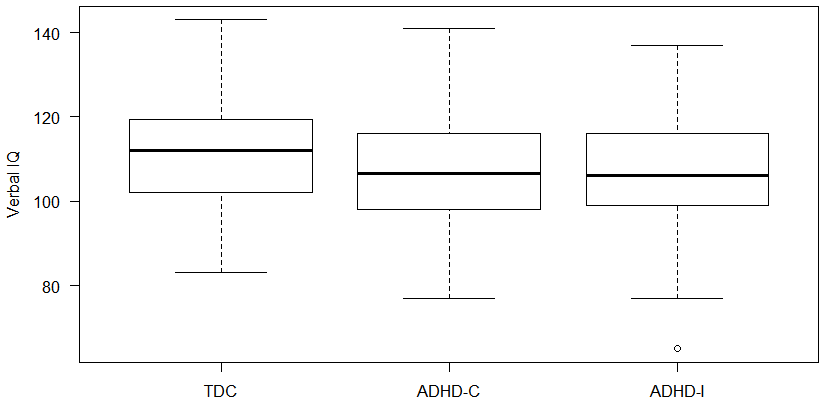

Figure 17: Comparison of Verbal IQ between TDC, ADHD-C, and ADHD-I.

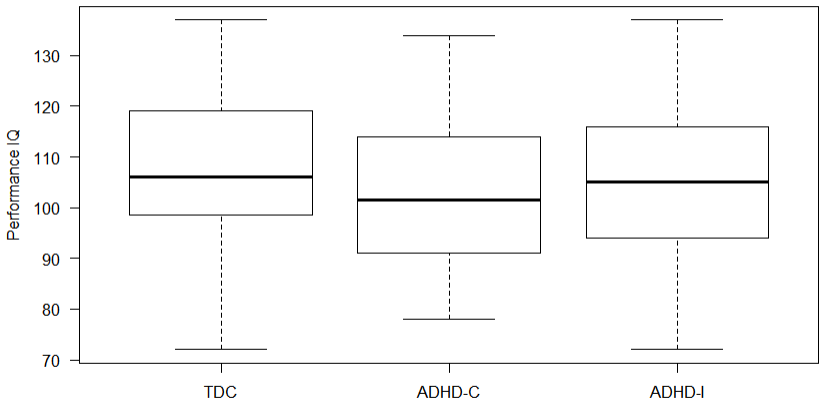

Figure 18: Comparison of Performance IQ between TDC, ADHD-C, and ADHD-I.

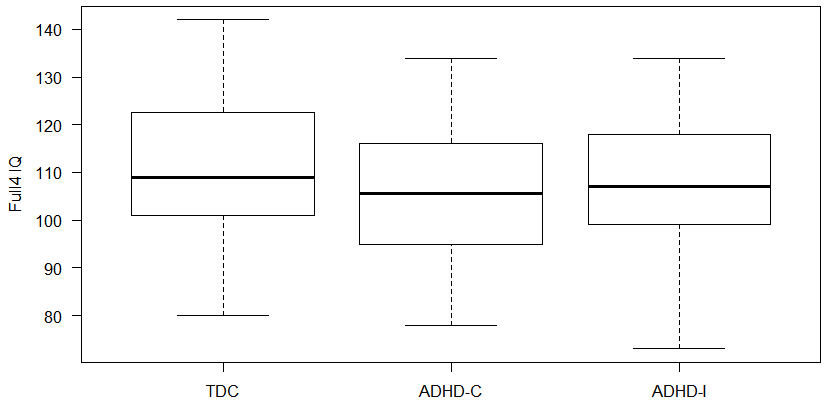

Figure 19: Comparison of Full4 IQ between TDC, ADHD-C, and ADHD-I.

Table 33 shows the result of significance testing to compare the three categories for each explanatory variable. The significance levels were determined using an ANOVA test because there are more than two categories being compared. A p-value $< .05$ is considered significant. The variables that have a significant difference between categories are gender, handedness, ADHD index, inattentive measure, and hyperactive/impulsive measure. This means that each of these variables differ significantly between the three categories. At least one of the variables differ significantly from the other two. The variables that have significant differences within categories will likely be included in the model to fit the data.

Table 33: Significance Test Results for Differences between TDC, ADHD-C and ADHD-I.

	Gender	Age	Handedness	ADHD Index	Inattentive
P-Value:	< 0.05	0.9417	0.0373	< 0.05	< 0.05
	Hyperactive /Impulsive	Verbal IQ	Performance IQ	Full 4 IQ	
P-Value:	< 0.05	0.1044	0.6903	0.2852	

Table 34 shows the significance test results for individual $\boldsymbol{\beta}$ values in the model

after fitting an unpenalized multinomial logistic regression model. The coefficients $\boldsymbol{\beta}$'s are tested against zero. The test determines if the value of $\boldsymbol{\beta}$ in the model is significantly different from zero. If the $\boldsymbol{\beta}$ is considered to be significant, that means that the corresponding variable should be included in the model. If the $\boldsymbol{\beta}$ is deemed insignificant, that means that the $\boldsymbol{\beta}$ is not influential on the model and could be removed. If a p-value < 0.05, then the $\boldsymbol{\beta}$ is considered to be significant. Since there are 3 categories, two different models are fit for the data. Tables 34 and 35 show the result of testing the significance of $\boldsymbol{\beta}$ in the two models. The upper left hand corner identifies which categorical responses the model fits. This test shows that only gender and hyperactive/impulsive measure are influential in the first model, and none of the variables in the second model are considered to be significant. By applying a goodness-of-fit test for multinomial models, we can determine if the model is good. The p-value for the goodness-of-fit test is given to be approximately zero, meaning that the model has an appropriate fit. The classification error when predicting a response using the test set of data is 0.2571, meaning that about 25% of the observations are incorrectly identified. The classification error of the training set is 0.1436 which is relatively low. Table 36 shows how the data were classified within the categories. Table 36 shows that the model quite accurately differentiates between the TDC and the other two categories, but the model has trouble differentiating between ADHD-C and ADHD-I as a majority of the mis-classifications occur between those two categories. The mis-classifications between ADHD-C and ADHD-I are most likely due to there not being significant differences in explanatory variables between the two.

Table 34: Significance Test Results for individual $\boldsymbol{\beta}$ in the TDC vs ADHD-C Model.

	Gender	Age	Handedness	ADHD Index	Inattentive
P-Value:	0.0022	0.3338	0.5804	0.3399	0.2742
	Hyperactive /Impulsive	Verbal IQ	Performance IQ	Full 4 IQ	
P-Value:	0.0026	0.4174	0.3008	0.3469	

Table 35: Significance Test Results for individual $\boldsymbol{\beta}$ in the TDC vs ADHD-I Model.

	Gender	Age	Handedness	ADHD Index	Inattentive
P-Value:	0.1577	0.8798	0.6231	0.1260	0.1675
	Hyperactive /Impulsive	Verbal IQ	Performance IQ	Full 4 IQ	
P-Value:	0.6918	0.6982	0.5852	0.6361	

Table 36: Predicted Response for Test Set and Training Set for Unpenalized Multinomial Logistic regression model

Test Set				Training Set			
	True				True		
Pred.	TDC	ADHD-C	ADHD-I	Pred.	TDC	ADHD-C	ADHD-I
TDC	10	1	1	TDC	84	1	2
ADHD-C	0	13	2	ADHD-C	2	63	13
ADHD-I	0	5	3	ADHD-I	1	10	26

From the simulation study, it is known that the Elastic Net penalty performs the best when applied to multinomial logistic regression models. The selection criteria that had the optimal performance were BIC and CV, so these methods are considered for ADHD data analysis. Using the `glmnet` function from the `glmnet` package in R the data was fit to a multinomial logistic regression model and subjected to the Elastic Net penalty. The `glmnet` function fits three different models, one for each category. When the model is used to find a predicted response, the predicted category is based on which model produces the highest probability based on the equation $\pi(\mathbf{X}) = \exp(\mathbf{X}\boldsymbol{\beta})/(1 + \exp(\mathbf{X}\boldsymbol{\beta}))$. The Elastic Net penalty, when subjected to the BIC, selects gender, ADHD index, and inattentive measure for determining if an individual is in the TDC category. Gender, age, and hyperactive/impulsive measure are selected for determining if an individual is in the ADHD-C category. Handedness, ADHD index, and inattentive measure are selected for determining if an individual is in the ADHD-I category. The variables selected for each response category are considered to be influential for predicting its response category. The predicted response of the test set produces a classification error of 0.2286. The predicted response of the

training set produces a classification error of 0.1535. The test set produces a slightly lower classification error than the unpenalized model, but a slightly higher classification error for the training set than the unpenalized model. Table 37 shows how the data were classified for the test set and the training set. This model is accurate when differentiating between the TDC and the other two categories, but has trouble differentiating between ADHD-C and ADHD-I. This is probably due to minimal differences in distributions of variables between the two categories. The model seems to be able to identify individuals in the ADHD-C category with pretty high accuracy, but the ADHD-I category has poor accuracy only correctly classifying 2/6 in the test set and 22/41 in the training set.

Table 37: Predicted Response for Test Set and Training Set from Multinomial Logistic Regression with Elastic Net Penalty using BIC

Test Set				Training Set			
	True				True		
Pred.	TDC	ADHD-C	ADHD-I	Pred.	TDC	ADHD-C	ADHD-I
TDC	10	1	1	TDC	84	3	3
ADHD-C	0	15	3	ADHD-C	2	65	16
ADHD-I	0	3	2	ADHD-I	1	6	22

The Elastic Net penalty when subjected to the CV selects gender, ADHD index, inattentive measure, and hyperactive/impulsive measure for classifying an individual in the TDC category. Gender, age, handedness, hyperactive/impulsive measure, verbal IQ, and performance IQ were selected for determining if an individual is in the ADHD-C category. Age, handedness, ADHD index, inattentive measure, and hyperactive/impulsive measure were selected for classifying an individual in the ADHD-I category. These measures are considered to be influential in determining if an individual is classified in a certain category. The predicted response of the test set produces a classification error of 0.1714 and the predicted response for the training set produces a classification error of 0.1485. The classification error using the CV is better than

using the BIC selection and seems to perform better than the unpenalized model. CV selects a more complicated model than BIC does in which all variables except full4 IQ were included in the model. The complexity of the model makes it difficult to determine which variables are influential. Table 38 shows the predicted response for the test set and training set when applied to the model. This model proves to be able to differentiate between TDC and the other two categories with high accuracy. However, this model has more difficulty differentiating between the ADHD-C and ADHD-I, but is more accurate for differentiating between ADHD-C and ADHD-I categories than the Elastic Net model using the BIC and the unpenalized model.

Table 38: Predicted Response for Test Set and Training Set from Multinomial Logistic Regression with Elastic Net Penalty using CV

Test Set				Training Set			
	True				True		
Pred.	TDC	ADHD-C	ADHD-I	Pred.	TDC	ADHD-C	ADHD-I
TDC	10	0	1	TDC	84	2	3
ADHD-C	0	16	2	ADHD-C	2	63	13
ADHD-I	0	3	3	ADHD-I	1	9	25

4 Conclusion

In conclusion, the influential factors of ADHD diagnosis, determined from the binary logistic regression model are gender, ADHD index, inattentive measure, and hyperactive/impulsive measure. The ADHD index, inattentive measure, and hyperactive/impuslve measures are not surprising because these measures are the result of tests to diagnose ADHD. The influence of gender does not mean that gender can cause ADHD, but that there is a relationship between gender and ADHD. The tests showed that males are significantly more likely to be diagnosed with ADHD than females. In order to identify why males are significantly more likely to be diagnosed with ADHD, further studies would need to be done, but it would be interesting to discover what

causes this difference. The multinomial logistic regression model showed that it is possible to differentiate between ADHD subtypes from variables in the ADHD data, but the accuracy was not very good. The complexity of the multinomial regression model made it difficult to determine the influential variables.

It may be valuable to try a different method of classification than generalized linear models for classifying the response categories. Future research may consider the linear discriminant analysis (LDA) or quadratic discriminant analysis (QDA) for classifying the different subtypes. Also, it may be valuable to consider an interaction term between the variables as there may be a relationship between explanatory variables that was not considered in this thesis. Future research may consider more explanatory variables such as race, having a history of abuse, location of birth, etc.

The cutoffs for the logistic regression methods could have been varied in order to test the strength of the model. In this experiment, the cutoff for the logistic models was constant at 0.5. Future work could involve using the receiver operating characteristic (ROC) curve to test the model using multiple cutoff values and to compute the sensitivity and specificity. This would allow an optimal cutoff to be used rather than a pre-determined constant. This method could have produced a more accurate model.

References

Agresti, A. (2002). *Categorical data analysis* (2nd ed.). New York: John Wiley and Sons.

Agresti, A. (2007). *An introduction to categorical data analysis* (2nd ed.). New York: John Wiley and Sons.

Akaike, H. (1973). Maximum likelihood identification of Gaussian autoregressive moving average models. *Biometrika*, *60*(2), 255–265. Retrieved from http://www.jstor.org/stable/2334537

Bellec, P., Chu, C., Chouinard-Decorte, F., Benhajali, Y., Margulies, D., & Craddock, R. (2016). The neuro bureau ADHD-200 preprocessed repository. *NeuroImage*. Advance online publication. doi: 10.1016/j.neuroimage.2016.06.034

Craven, P., & Wahba, G. (1979). Smoothing noisy data with spline functions: estimating the correct degree of smoothing by the method of generalized crossvalidation. *Numerische Mathematik*, *31*, 377–403. Retrieved from http://www.stat.washington.edu/courses/stat527/s13/readings/cravenwabha79.pdf

Fan, J., & Li, R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, *96*(456), 1348–1360. doi: 10.1198/016214501753382273

Fitzmaurice, G. M. (2016). Regression. *Diagnostic Histopathology*, *22*(7), 271–278. doi: 10.1016/j.mpdhp.2016.06.004

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*, 1–22. Retrieved from http://web.stanford.edu/~hastie/Papers/glmnet.pdf

Hoerl, A., & Kennard, R. (1970). Ridge regression: application to nonorthogonal

problems. *Technometrics*, *12*(1), 69–82. doi: 10.2307/1267352

Li, N. (2011). *Sparse learning in multiclass problems* (PhD dissertation). North Carolina State University.

Park, M., & Hastie, T. (2007). L1-regularization path algorithm for generalized linear models. *Journal of the Royal Statistical Society. Series B.*, *69*(4), 659–677. doi: 10.1111/j.1467-9868.2007.00607.x

R Core Team. (2016). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from `https://www.R-project.org/`

Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464. Retrieved from `http://www.jstor.org/stable/2958889`

Shi, J., Yin, W., Osher, S., & Sajda, P. (2010). A fast hybrid algorithm for large-scale l_1-regularized logistic regression. *Journal of Machine Learning Research*, *11*, 713–741. Retrieved from `http://www.caam.rice.edu/~wy1/paperfiles/Rice_CAAM_TR08-08.PDF`

Stone, M. (1974). Cross-validation and multinomial prediction. *Biometrika*, *61*(3), 509–515. doi: 10.2307/2334733

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B.*, *58*(1), 267–288. Retrieved from `http://www.jstor.org.unr.idm.oclc.org/stable/2346178`

Tutz, G., Pobnecker, W., & Uhlmann, L. (2015). Variable selection in general multinomial logit models. *Computational Statistics and Data Analysis*, *82*, 207–222. doi: 10.1016/j.csda.2014.09.009

Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, *101*(476), 1418–1429. doi: 10.1198/

016214506000000735

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society. Series B.*, *67*(2), 301–320. doi: 10.1111/j.1467-9868.2005.00503.x

A Appendix

A.1 Uncorrelated Linear Data Generation R Code

```
###############################################
#initialize Packages
###############################################
library(lars)
library(MASS)
library(elasticnet)
###############################################
#initialize values for simulation
###############################################
n=100
p = 8
beta = c(1.5, 3, 0,0,2,0,0,0)
numSims = 1000
#########################################################
# initialize data
#########################################################
bigX = array(0, dim = c(numSims, n, p))
bigY = matrix(0,numSims, n)
for(i in 1:numSims)
{
  epsilon = rnorm(n)
  bigX[i,,] = matrix(rnorm(n*p),n,p)
  bigX[i,,] = scale(bigX[i,,], T,T)
  bigY[i,] = bigX[i,,]%*%beta+epsilon
  bigY[i,] = scale(bigY[i,],T,F)
}
```

A.2 Correlated Linear Data Generation R Code

```
###############################################
#initialize variables
###############################################
library(lars)
library(MASS)
library(elasticnet)
###############################################
#initialize values for simulation
###############################################
n=100
p = 8
beta = c(1.5, 3, 0,0,2,0,0,0)
numSims = 1000
MU <- matrix(0,8,1)
SIGMA <- matrix(0, 8,8)
#########################################################
# initialize data
#########################################################
bigX = array(0, dim = c(numSims, n, p))
bigY = matrix(0,numSims, n)
MU <- matrix(0,8,1)
SIGMA <- matrix(0, 8,8)
for(k in 1:numSims)
{
  epsilon = rnorm(n)
  rho=0.5
  for (i in 1:8){
    for (j in 1:8){
      SIGMA[i,j] =rho^{abs(i-j)}
    }
  }
  X   = mvrnorm(100, mu=MU, Sigma=SIGMA)
  x = scale(x,T,T)
  bigX[k,,] = X
  bigY[k,] = bigX[k,,]%*%beta+epsilon
  bigY[k,] = scale(bigY[k,],T,F)
}
```

A.3 Linear Model Simulation R Code

```
# Linear Model Simulation R Code

#########################################################
# Oracle
#########################################################
oracleBeta = matrix(0,numSims,3)
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0
```

```
result = matrix(0,8,2)

for(i in 1:numSims)
{
  betaHat = lm(bigY[i,]~bigX[i,,c(1,2,5)]+0)
  oracleBeta[i,]  = coef(betaHat)
  #MSE[i] = sum((beta[c(1,2,5)]-oracleBeta[i,])^2)       # for indep X
  MSE[i] = t(beta[c(1,2,5)]-oracleBeta[i,])%*%SIGMA[c(1,2,5),c(1,2,5)]%*%(beta[c(1,2,5)]-oracleBeta[i,]) # for
       correlated X
}
result[1,1] = "oracle_corr"
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", 5)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", 0)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", 0)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", 3)
result[8,] = c( "Proportion of correct models: ", 1)
result
write.csv(result,"C:/Users/Matthew/Documents/oracle_lin_corr.csv",quote = FALSE)

#######################################################
# OLS sim
#######################################################
olsBeta = matrix(0,numSims,p)
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzeroZ= countIncorrectZ = countCorrNonZero = numCorModel = 0
result = matrix(0,8,2)

for(i in 1:numSims)
{
  betaHat = lm(bigY[i,]~bigX[i,,]+0)
  olsBeta[i,]  = coef(betaHat)

  zero = which(beta==0)
  nonzero = which(beta!=0)
  countCorrectZ = countCorrectZ + sum(olsBeta[i,c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ =  countNonzeroZ + sum(olsBeta[i,c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(olsBeta[i,c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(olsBeta[i,c(1,2,5)] != 0)
  numCorModel = numCorModel + sum(all(olsBeta[i,c(3,4,6,7,8)]==0) && all(olsBeta[i,c(1,2,5)] != 0))
  #MSE[i] = sum((beta-olsBeta[i,])^2)              # for indep X
  MSE[i] = t(beta-olsBeta[i,])%*%SIGMA%*%(beta-olsBeta[i,]) # for correlated X
}

result[1,1] = "OLSsim_true_corr"
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzeroZ/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/OLS_true_corr.csv",quote = FALSE)

#######################################################
# initialize AIC BIC CV and GCV functions for lasso and adaptive LASSO
#######################################################
AIC <- function(fit, y, x)
{
  mylambda = seq(0,1, length = 100)
  aic = array(0, length(mylambda))
  index = 1

  for(lambda in mylambda)
  {
    testfit = predict(fit, x, s = lambda, "coefficients", "fraction")$coef
    yhat = x%*%testfit
    SSE = sum((y-yhat)^2)
    p = sum(testfit != 0)
    aic[index] = SSE + 2*p
    index = index +1

  }
  lambda = mylambda[which.min(aic)]
  return(lambda)
}
BIC <- function(fit,y,x,n)
{
  mylambda = seq(0,1, length = 100)
  bic = array(0, length(mylambda))
  index = 1

  for(lambda in mylambda)
  {
    testfit = predict(fit, x, lambda, "coefficients", "fraction")$coef
    yhat = x%*%testfit
    SSE = sum((y-yhat)^2)
```

```
    p = sum(testfit != 0)
    bic[index] = SSE + p*log(n)
    index = index +1

  }
  lambda = mylambda[which.min(bic)]
  return(lambda)
}

CV<-function (x, y, K = 10, index)
{

  all.folds <- cv.folds(length(y), K)

  residmat <- matrix(0, length(index), K)
  for (i in seq(K)) {
    omit <- all.folds[[i]]
    fit <- lars(x[-omit, , drop = FALSE], y[-omit], intercept = F, normalize = F)
    fit <- predict(fit, x[omit, , drop = FALSE], s = index,"fit","fraction")$fit

    residmat[, i] <- apply((y[omit] - fit)^2, 2, mean)
  }
  cv <- apply(residmat, 1, mean)
  cv.error <- sqrt(apply(residmat, 1, var)/K)
  object <- list(index = index, cv = cv, cv.error = cv.error)
  invisible(object)
}

GCV <- function(fit, y, x, n)
{
  mylambda = seq(0,1,length = 100)
  gcv = array(0, length(mylambda))
  index = 1
  for(lambda in mylambda)
  {
    testfit = predict(fit, x, s = lambda, "coefficients", "fraction")$coef
    yhat = x%*%testfit
    p = sum(testfit != 0)
    gcv[index] = (1/n)*(sum(((y-yhat)/(1-(p/n)))^2))
    index = index +1
  }
  lam = mylambda[which.min(gcv)]
  return(lam)
}

#######################################################
# lasso sim with AIC
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0
for( i in 1:numSims)
{
  fit =lars(bigX[i,,],bigY[i,], intercept = F, normalize = F)
  aicFit = predict(fit, bigX[i,,], AIC(fit,bigY[i,],bigX[i,,]), "coefficients", "fraction")$coef

  countCorrectZ = countCorrectZ + sum(aicFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ +sum(aicFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(aicFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(aicFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(aicFit[c(3,4,6,7,8)]==0) && all(aicFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-aicFit)^2)                  # for indep X
  MSE[i] = t(beta-aicFit)%*%SIGMA%*%(beta-aicFit) # for correlated X
}
result[1,] = c("LASSO_lin_corr", "AIC")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzeroZ/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/lassoAIC_lin_corr.csv",quote = FALSE)


#######################################################
# lasso sim with BIC
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  fit =lars(bigX[i,,],bigY[i,], intercept = F, normalize = F)
  bicFit = predict(fit, bigX[i,,], BIC(fit,bigY[i,],bigX[i,,],n), "coefficients", "fraction")$coef

  countCorrectZ = countCorrectZ + sum(bicFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzero = countNonzero + sum(bicFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(bicFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
```

```
  countCorrNonZero = countCorrNonZero + sum(bicFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(bicFit[c(3,4,6,7,8)]==0) && all(bicFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-bicFit)^2)                  # for indep X
  MSE[i] = t(beta-bicFit)%*%SIGMA%*%(beta-bicFit) # for correlated X
}
result[1,] = c("LASSO_lin_corr", "BIC")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/lassoBIC_lin_corr.csv",quote = FALSE)

#########################################################
# lasso sim with CV
#########################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  fit =lars(bigX[i,,],bigY[i,], intercept = F, normalize = F)
  crossValid = CV(bigX[i,,],bigY[i,],10,seq(from = 0, to = 1, length = 100))
  cvFit = predict(fit, bigX[i,,], crossValid$index[which.min(crossValid$cv)], "coefficients", "fraction")$coef

  countCorrectZ = countCorrectZ + sum(cvFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(cvFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(cvFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(cvFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(cvFit[c(3,4,6,7,8)]==0) && all(cvFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-cvFit)^2)                    # for indep X
  MSE[i] = t(beta-cvFit)%*%SIGMA%*%(beta-cvFit)  # for correlated X
}
result[1,] = c("LASSO_lin_corr", "CV")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/lassoCV_lin_corr.csv",quote = FALSE)

#########################################################
# lasso sim with GCV
#########################################################

MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  fit =lars(bigX[i,,],bigY[i,], intercept = F, normalize = F)
  gcvFit = predict(fit, bigX[i,,], s=GCV(fit,bigY[i,],bigX[i,,],n), "coefficients", "fraction")$coef

  countCorrectZ = countCorrectZ + sum(gcvFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(gcvFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(gcvFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(gcvFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(gcvFit[c(3,4,6,7,8)]==0) && all(gcvFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-gcvFit)^2)                    # for indep X
  MSE[i] = t(beta-gcvFit)%*%SIGMA%*%(beta-gcvFit)  # for correlated X
}
result[1,] = c("LASSO_lin_corr", "GCV")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/lassoGCV_lin_corr.csv",quote = FALSE)

#########################################################
# set up new x for adaptive lasso
#########################################################
newX = array(0, dim = c(numSims, n, p))
for(i in 1:numSims)
{
  for(j in 1:n)
  {
    newX[i,j,] = bigX[i,j,]*abs(olsBeta[i,])
  }
}
```

```
#######################################################
# Adaptive lasso Aic
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  fit =lars(newX[i,,],bigY[i,], intercept = F, normalize = F)
  aicFit = predict(fit, newX[i,,], AIC(fit,bigY[i,],newX[i,,]), "coefficients", "fraction")
  aicFit = coef(aicFit)*abs(olsBeta[i,])

  countCorrectZ = countCorrectZ + sum(aicFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(aicFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(aicFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(aicFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(aicFit[c(3,4,6,7,8)]==0) && all(aicFit[c(1,2,5)] != 0))

  #MSE[i] = sum((beta-aicFit)^2)                    # for indep X
  MSE[i] = t(beta-aicFit)%*%SIGMA%*%(beta-aicFit)  # for correlated X
}
result[1,] = c("ADAPTIVE LASSO lin corr", "AIC")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/AlassoAIC_lin_corr.csv",quote = FALSE)

#######################################################
# adaptive lasso sim with BIC
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  fit =lars(newX[i,,],bigY[i,], intercept = F, normalize = F)
  bicFit = predict(fit, newX[i,,], BIC(fit,bigY[i,],newX[i,,],n), "coefficients", "fraction")
  bicFit = coef(bicFit)*abs(olsBeta[i,])

  countCorrectZ = countCorrectZ + sum(bicFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(bicFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(bicFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(bicFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(bicFit[c(3,4,6,7,8)]==0) && all(bicFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-bicFit)^2)                    # for indep X
  MSE[i] = t(beta-bicFit)%*%SIGMA%*%(beta-bicFit)  # for correlated X
}
result[1,] = c("ADAPTIVE LASSO corr", "BIC")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/AlassoBIC_lin_corr.csv",quote = FALSE)

#######################################################
# adaptive lasso sim with CV
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  fit =lars(newX[i,,],bigY[i,], intercept = F, normalize = F)
  crossValid = CV(newX[i,,],bigY[i,],10,seq(from = 0, to = 1, length = 100))
  cvFit = predict(fit, newX[i,,], crossValid$index[which.min(crossValid$cv)], "coefficients", "fraction")
  cvFit = coef(cvFit)*abs(olsBeta[i,])

  countCorrectZ = countCorrectZ + sum(cvFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(cvFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(cvFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(cvFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(cvFit[c(3,4,6,7,8)]==0) && all(cvFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-cvFit)^2)                    # for indep X
  MSE[i] = t(beta-cvFit)%*%SIGMA%*%(beta-cvFit)   # for correlated X
}
result[1,] = c("ADAPTIVE LASSO corr", "CV")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
```

```
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/AlassoCV_lin_corr.csv",quote = FALSE)

#####################################################
# Adaptive lasso sim with GCV
#####################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  fit =lars(newX[i,,],bigY[i,], intercept = F, normalize = F)
  gcvFit = predict(fit, bigX[i,,], s=GCV(fit,bigY[i,],newX[i,,],n), "coefficients", "fraction")
  gcvFit = coef(gcvFit)*abs(olsBeta[i,])

  countCorrectZ = countCorrectZ + sum(gcvFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(gcvFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(gcvFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(gcvFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(gcvFit[c(3,4,6,7,8)]==0) && all(gcvFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-gcvFit)^2)                        # for indep X
  MSE[i] = t(beta-gcvFit)%*%SIGMA%*%(beta-gcvFit)   # for correlated X
}
result[1,] = c("ADAPTIVE LASSO corr", "GCV")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/AlassoGCV_lin_corr.csv",quote = FALSE)

#################################################################
# AIC BIC CV and GCV for ridge regression
#################################################################
aicR <- function(y,x)
{
  mylambda = seq(from = 0, to = 5, length = 200)
  aic = array(0,length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    testfit = lm.ridge(y~x+0, lambda = lam)$coef
    yhat = x%*%testfit
    SSE = sum((y-yhat)^2)
    p = sum(testfit != 0)
    aic[index] = SSE +2*p
    index = index +1
  }
  lam = mylambda[which.min(aic)]
  return(lam)
}

bicR<-function(y,x,n)
{
  mylambda = seq(from = 0, to = 5, length = 200)
  bic = array(0,length(mylambda))
  index = 1
  for(lam in mylambda)
  {
    testfit = lm.ridge(y~x+0, lambda = lam)$coef

    yhat = x%*%testfit
    SSE = sum((y-yhat)^2)
    p = sum(testfit != 0)
    bic[index] = SSE + p*log(n)
    index = index+1
  }
  lam = mylambda[which.min(bic)]
  return(lam)
}

cvR<-function (x, y, K = 10, index)
{
  all.folds <- cv.folds(length(y), K)
  residmat <- matrix(0, length(index), K)
  for (j in seq(K)) {
    omit <- all.folds[[j]]
    fit <- lm.ridge(y[-omit]~x[-omit, , drop = FALSE]+0, lambda=index)
    fit <- x[omit, , drop = FALSE]%*%fit$coef    ## fitted = X %*% coef
    residmat[, j] <- apply((y[omit] - fit)^2, 2, mean)
  }
  cv <- apply(residmat, 1, mean)
```

```
  return(index[which.min(cv)])

}

#######################################################
# ridge regression with AIC
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for(i in 1:numSims)
{
  fit = lm.ridge(bigY[i,]~bigX[i,,]+0, lambda = aicR(bigY[i,], bigX[i,,]))$coef

  countCorrectZ = countCorrectZ + sum(fit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(fit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(fit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(fit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(fit[c(3,4,6,7,8)]==0) && all(fit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-fit)^2)                    # for indep X
  MSE[i] = t(beta-fit)%*%SIGMA%*%(beta-fit)   # for correlated X
}
result[1,] = c("Ridge_corr", "AIC")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/RidgeAIC_lin_corr.csv",quote = FALSE)

#######################################################
# ridge regression with BIC
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for(i in 1:numSims)
{
  fit = lm.ridge(bigY[i,]~bigX[i,,]+0, lambda = bicR(bigY[i,], bigX[i,,],n))$coef
  countCorrectZ = countCorrectZ + sum(fit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(fit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(fit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(fit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(fit[c(3,4,6,7,8)]==0) && all(fit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-fit)^2)                    # for indep X
  MSE[i] = t(beta-fit)%*%SIGMA%*%(beta-fit)   # for correlated X
}
result[1,] = c("Ridge_corr", "BIC")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/RidgeBIC_lin_corr.csv",quote = FALSE)

#######################################################
# ridge regression with CV
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for(i in 1:numSims)
{
  fit = lm.ridge(bigY[i,]~bigX[i,,]+0, lambda = cvR(bigX[i,,],bigY[i,],10, seq(from = 0, to = 5, length = 200)))$coef

  countCorrectZ = countCorrectZ + sum(fit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(fit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(fit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(fit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(fit[c(3,4,6,7,8)]==0) && all(fit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-fit)^2)                    # for indep X
  MSE[i] = t(beta-fit)%*%SIGMA%*%(beta-fit)   # for correlated X
}
result[1,] = c("Ridge_corr", "CV")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/RidgeCV_lin_corr.csv",quote = FALSE)
```

```
#######################################################
# ridge regression with GCV
#######################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for(i in 1:numSims)
{
  fit = lm.ridge(bigY[i,]~bigX[i,,]+0, lambda = seq(0,5,length = 1000))
  fit = fit$coef[,which.min(fit$GCV)]

  countCorrectZ = countCorrectZ + sum(fit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(fit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(fit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(fit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(fit[c(3,4,6,7,8)]==0) && all(fit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-fit)^2)                    # for indep X
  MSE[i] = t(beta-fit)%*%SIGMA%*%(beta-fit)   # for correlated X
}
result[1,] = c("Ridge_corr", "GCV")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/RidgeGCV_lin_corr.csv",quote = FALSE)

#######################################################
# AIC BIC CV and GCV for elastic net
#######################################################
AICenet <- function(y, x)
{
  mylambda = seq(0,5, length = 200)
  myS = seq(0,1,length = 100)
  aic = array(0, 200*100)
  index = 1
  tuningParam = matrix(0,100*200,2)

  for(lambda in mylambda)
  {
    fit = enet(x,y,lambda, intercept = F, normalize = F)
    for(s in myS)
    {
      testfit = predict(fit, x, s, "coefficients", "fraction")$coef
      yhat = x%*%testfit
      SSE = sum((y-yhat)^2)
      p = sum(testfit != 0)
      aic[index] = SSE + 2*p
      tuningParam[index,] = c(lambda, s)
      index = index +1
    }
  }
  list("lambda" = tuningParam[which.min(aic),1], "s" = tuningParam[which.min(aic),2] )
}
BICenet <- function(y,x,n)
{
  mylambda = seq(0,5, length = 200)
  myS = seq(0,1,length = 100)
  bic = array(0, 200*100)
  index = 1
  tuningParam = matrix(0,100*200,2)

  for(lambda in mylambda)
  {
    fit = enet(x,y,lambda, intercept = F, normalize = F)
    for(s in myS)
    {
      testfit = predict(fit, x, s, "coefficients", "fraction")$coef
      yhat = x%*%testfit
      SSE = sum((y-yhat)^2)
      p = sum(testfit != 0)
      bic[index] = SSE + p*log(n)
      tuningParam[index,] = c(lambda,s)
      index = index +1
    }

  }
  list("lambda" = tuningParam[which.min(bic),1], "s" =tuningParam[which.min(bic),2] )
}
CVenet<-function (x, y, K = 10)
{
  myS = seq(0,1,length=100)
  myLambda = seq(0,5,length=200)
  tuningParam = matrix(0,200*100,2)
```

```
  all.folds <- cv.folds(length(y), K)

  residmat <- matrix(0, length(myLambda)*length(myS), K)

  for (i in seq(K)) {
    omit <- all.folds[[i]]
    index=1
    for (lambda in myLambda){
      fit0 <- enet(x[-omit, , drop = FALSE], y[-omit], lambda = lambda)
      for (s in myS){
        fit <- predict(fit0, x[omit, , drop = FALSE], s = s,"fit","fraction")$fit
        residmat[index,i] <- mean((y[omit] - fit)^2)
        index = index+1
      }
    }
  }

  index=1
  for (lambda in myLambda){
    for (s in myS){
      tuningParam[index,] = c(lambda,s)
      index = index+1
    }
  }

  cv <- apply(residmat, 1, mean)
  list("lambda" = tuningParam[which.min(cv),1], "s" =tuningParam[which.min(cv),2] )
}
#############################################################################################

GCVenet <- function(y, x, n)
{
  mylambda = seq(0,5,length = 200)
  myS = seq(0,1,length = 100)
  gcv = array(0, 200*100)
  index = 1
  tuningParam = matrix(0,200*100,2)

  for(lambda in mylambda)
  {
    fit = enet(x,y,lambda, intercept = F, normalize = F)
    for(s in myS)
    {
      testfit = predict(fit, x, s, "coefficients", "fraction")$coef
      yhat = x%*%testfit
      p = sum(testfit != 0)
      gcv[index] = (1/n)*(sum(((y-yhat)/(1-(p/n)))^2))
      tuningParam[index,] = c(lambda,s)
      index = index +1
    }
  }
  list("lambda" = tuningParam[which.min(gcv),1], "s" =tuningParam[which.min(gcv),2] )
}
#####################################################
# elastic net with AIC
#####################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  tuningParam = AICenet(bigY[i,],bigX[i,,])

  aicfit = enet(bigX[i,,],bigY[i,], tuningParam$lambda, intercept = F, normalize = F)
  aicFit = predict(aicfit, bigX[i,,], tuningParam$s, "coefficients", "fraction")$coef

  countCorrectZ = countCorrectZ + sum(aicFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(aicFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(aicFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(aicFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(aicFit[c(3,4,6,7,8)]==0) && all(aicFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-aicFit)^2)                    # for indep X
  MSE[i] = t(beta-aicFit)%*%SIGMA%*%(beta-aicFit)   # for correlated X
}
result[1,] = c("ENET_corr", "AIC")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/enetAIC_lin_corr.csv",quote = FALSE)

#####################################################
# elastic net sim with BIC
#####################################################
MSE = matrix(0,numSims,1)
```

```
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  tuningParam = BICenet(bigY[i,],bigX[i,,],n)

  fit =enet(bigX[i,,],bigY[i,], tuningParam$lambda, intercept = F, normalize = F)
  bicFit = predict(fit, bigX[i,,], tuningParam$s , "coefficients", "fraction")$coef

  countCorrectZ = countCorrectZ + sum(bicFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(bicFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(bicFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(bicFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(bicFit[c(3,4,6,7,8)]==0) && all(bicFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-bicFit)^2)                      # for indep X
  MSE[i] = t(beta-bicFit)%*%SIGMA%*%(beta-bicFit)   # for correlated X
}
result[1,] = c("ENET_corr", "BIC")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/enetBIC_lin_corr.csv",quote = FALSE)

#####################################################
# enet sim with CV
#####################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  crossValid = CVenet(bigX[i,,],bigY[i,],10)
  fit =enet(bigX[i,,],bigY[i,], lambda = crossValid$lambda, intercept = F, normalize = F)
  cvFit = predict(fit, bigX[i,,], crossValid$s, "coefficients", "fraction")$coef
  print(i)
  countCorrectZ = countCorrectZ + sum(cvFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(cvFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(cvFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(cvFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(cvFit[c(3,4,6,7,8)]==0) && all(cvFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-cvFit)^2)                      # for indep X
  MSE[i] = t(beta-cvFit)%*%SIGMA%*%(beta-cvFit)   # for correlated X
}
result[1,] = c("ENET_corr", "CV")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
write.csv(result,"C:/Users/Matthew/Documents/enetCV_lin_corr.csv",quote = FALSE)

#####################################################
# elastic net sim with GCV
#####################################################
MSE = matrix(0,numSims,1)
countCorrectZ = countNonzero= countIncorrectZ = countCorrNonZero = numCorModel = 0

for( i in 1:numSims)
{
  tuningParam = GCVenet(bigY[i,],bigX[i,,],n)

  fit =enet(bigX[i,,],bigY[i,], tuningParam$lambda, intercept = F, normalize = F)
  gcvFit = predict(fit, bigX[i,,], tuningParam$s , "coefficients", "fraction")$coef

  countCorrectZ = countCorrectZ + sum(gcvFit[c(3,4,6,7,8)]== beta[c(3,4,6,7,8)])
  countNonzeroZ = countNonzeroZ + sum(gcvFit[c(1,2,5)]==0)
  countIncorrectZ = countIncorrectZ + sum(gcvFit[c(3,4,6,7,8)]!= beta[c(3,4,6,7,8)])
  countCorrNonZero = countCorrNonZero + sum(gcvFit[c(1,2,5)]!=0)
  numCorModel = numCorModel + sum(all(gcvFit[c(3,4,6,7,8)]==0) && all(gcvFit[c(1,2,5)] != 0))
  #MSE[i] = sum((beta-gcvFit)^2)                      # for indep X
  MSE[i] = t(beta-gcvFit)%*%SIGMA%*%(beta-gcvFit)   # for correlated X
}
result[1,] = c("ENET_corr", "GCV")
result[2,] = c( "Mean MSE: ", mean(MSE))
result[3,] = c( "Median MSE: ", median(MSE))
result[4,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[5,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzero/numSims)
result[6,] = c( "Average number of zero coefficients incorrectly selected: ", (countIncorrectZ)/numSims)
result[7,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[8,] = c( "Proportion of correct models: ", numCorModel/numSims)
result
```

```
write.csv(result,"C:/Users/Matthew/Documents/enetGCV_lin_corr.csv",quote = FALSE)
```

A.4 Uncorrelated Binary Data Generation R Code

```
################################################
# initialize libraries
################################################
library(glmnet)

################################################
# initialize data
################################################
n=100
p = 8
beta = c(1.5, 3, 0,0,2,0,0,0)
numSims = 50
bigX = array(0, dim = c(numSims, n, p))
bigY = matrix(0,numSims, n)

for(i in 1:numSims)
{
  bigX[i,,] = matrix(rnorm(n*p),n,p)
  bigX[i,,] = scale(bigX[i,,], T,T)
  for(j in 1:n)
  {
    bigY[i,j] = rbinom(1,1,exp(bigX[i,j,]%*%beta)/(1+exp(bigX[i,j,]%*%beta)))
  }
}
```

A.5 Correlated Binary Data Generation R Code

```
################################################
# initialize libraries
################################################
library(glmnet)
################################################
# initialize data
################################################
n=100
p = 8
beta = c(1.5, 3, 0, 0, 2, 0, 0, 0)
numSims = 50
bigX = array(0, dim = c(numSims, n, p))
bigY = matrix(0,numSims, n)
MU <- matrix(0,8,1)
SIGMA <- matrix(0, 8,8)

for(k in 1:numSims)
{
  rho=0.5
  for (i in 1:8){
    for (j in 1:8){
      SIGMA[i,j] =rho^{abs(i-j)}
    }
  }
  X = mvrnorm(100, mu=MU, Sigma=SIGMA)
  bigX[k,,] = matrix(rnorm(n*p),n,p)
  bigX[k,,] = scale(bigX[k,,], T,T)
  for(l in 1:n)
  {
    bigY[k,l] = rbinom(1,1,exp(bigX[k,l,]%*%beta)/(1+exp(bigX[k,l,]%*%beta)))
  }
}
```

A.6 Binary Logistic Model Simulation

```
K=10
foldid = sample(rep(1:K, length = n))


# Binary Logistic Model Simulation
################################################
# Oracle SIM
################################################

RE = matrix(0,numSims,1)
PE = matrix(0,numSims,1)
countCorrectZ = countNonzeroZ= countIncorrectZ = countCorrNonZero = numCorModel = numOverModel= 0
result = matrix(0,11,2)
for(i in 1:numSims)
{
  glmFit = glmnet(bigX[i,,c(1,2,5)],bigY[i,], family = "binomial", intercept=FALSE ,alpha = 1, lambda = 0, nlambda = 1)
```

```
  yhat = predict(glmFit, bigX[i,,c(1,2,5)], type = "class")

  RE[i] = sqrt(sum((glmFit$beta-beta[c(1,2,5)])^2))/sqrt(sum(beta[c(1,2,5)]^2))
  PE[i] = mean(yhat!=bigY[i,])

}
result[1,1] = "OracleLOGsim"
result[2,] = c( "Mean RE: ", mean(RE))
result[3,] = c( "Median RE: ", median(RE))
result[4,] = c( "Mean PE: ", mean(PE))
result[5,] = c( "Median PE: ", median(PE))
result[6,] = c( "Average number of coefficients correctly shrunk to zero: ", 5)
result[7,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", 0)
result[8,] = c( "Average number of zero coefficients incorrectly selected: ", 0)
result[9,] = c( "Average number of nonzero coefficients correctly selected: ", 3)
result[10,] = c( "Proportion of correct models: ", 1)
result[11,] = c( "Proportion of overfitted models: ", 0)
result

write.csv(result,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/Oracle_LOG_%s.csv",
      casename[case]),quote = FALSE)

################################################
# OLS SIM
################################################

RE = matrix(0,numSims,1)
PE = matrix(0,numSims,1)
countCorrectZ = countNonzeroZ= countIncorrectZ = countCorrNonZero = numCorModel = numOverModel= 0
result = matrix(0,11,2)
for(i in 1:numSims)
{
  glmFit = glmnet(bigX[i,,],bigY[i,], family = "binomial", intercept=FALSE ,alpha = 1, lambda = 0, nlambda = 1)
  yhat = predict(glmFit, bigX[i,,], type = "class")

  RE[i] = sqrt(sum((glmFit$beta-beta)^2))/sqrt(sum(beta^2))
  PE[i] = mean(yhat!=bigY[i,])

  countCorrectZ = countCorrectZ + sum(glmFit$beta[zero]==0)
  countNonzeroZ =  countNonzeroZ + sum(glmFit$beta[nonzero]==0)
  countIncorrectZ = countIncorrectZ + sum(glmFit$beta[zero] != 0)
  countCorrNonZero = countCorrNonZero + sum(glmFit$beta[nonzero] != 0)
  numCorModel = numCorModel + sum(all(glmFit$beta[zero]==0) && all(glmFit$beta[nonzero] != 0))
  numOverModel = numOverModel + sum(all(glmFit$beta[zero]==0)==F && all(glmFit$beta[nonzero] != 0))
}
result[1,1] = "OLSLOGsim"
result[2,] = c( "Mean RE: ", mean(RE))
result[3,] = c( "Median RE: ", median(RE))
result[4,] = c( "Mean PE: ", mean(PE))
result[5,] = c( "Median PE: ", median(PE))
result[6,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[7,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzeroZ/numSims)
result[8,] = c( "Average number of zero coefficients incorrectly selected: ", countIncorrectZ/numSims)
result[9,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[10,] = c( "Proportion of correct models: ", numCorModel/numSims)
result[11,] = c( "Proportion of overfitted models: ", numOverModel/numSims)
result

write.csv(result,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/OLS_LOG_%s.csv", casename
      [case]),quote = FALSE)

#########################################
# AIC BIC CV and GCV for LASSO
#########################################

Criteria <- function(y,x){
  n=length(y)

  glmFit = glmnet(x,y, family = "binomial", intercept=FALSE ,alpha = 1)
  yhat = predict(glmFit, x, type = "response")
  loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)

  aic = -2*loglik + 2*(glmFit$df)
  lambda = glmFit$lambda[which.min(aic)]
  glmFitbeta = predict(glmFit, s=lambda, type = "coef")
  beta.aic = as.vector(glmFitbeta)[-1]
  yhat.aic = predict(glmFit, newx=x, s=lambda, type = "class")

  bic = -2*loglik + log(n)*(glmFit$df)
  lambda = glmFit$lambda[which.min(bic)]
  glmFitbeta = predict(glmFit, s=lambda, type = "coef")
  beta.bic = as.vector(glmFitbeta)[-1]
  yhat.bic = predict(glmFit, newx=x, s=lambda, type = "class")

  gcv = -2*loglik/(n*(1-(glmFit$df/n))^2)
  lambda = glmFit$lambda[which.min(gcv)]
  glmFitbeta = predict(glmFit, s=lambda, type = "coef")
  beta.gcv = as.vector(glmFitbeta)[-1]
  yhat.gcv = predict(glmFit, newx=x, s=lambda, type = "class")
```

```
  cvfit=cv.glmnet(x,y,foldid=foldid,family="binomial", alpha=1,intercept=F)
  lambda=cvfit$lambda[which.min(cvfit$cvm)]
  glmFitbeta = predict(glmFit, s=lambda, type = "coef")
  beta.cv = as.vector(glmFitbeta)[-1]
  yhat.cv = predict(glmFit, newx=x, s=lambda, type = "class")

  list("beta.aic" = beta.aic, "yhat.aic"=yhat.aic, "beta.bic"=beta.bic, "yhat.bic"=yhat.bic, "beta.gcv"=beta.gcv, "yhat.
       gcv"=yhat.gcv, "beta.cv"=beta.cv, "yhat.cv"=yhat.cv)
}

################################################
# LASSO simulation
################################################

savefile <- function(numSims, bigY, bigX, allbeta, zero, nonzero, type, case){

  aic.RE = aic.PE = matrix(0,numSims,1)
  aic.countCorrectZ = aic.countNonzeroZ= aic.countIncorrectZ = aic.countCorrNonZero = aic.numCorModel = aic.numOverModel
       = 0
  aic.result = matrix(0,11,2)

  bic.RE = bic.PE = matrix(0,numSims,1)
  bic.countCorrectZ = bic.countNonzeroZ= bic.countIncorrectZ = bic.countCorrNonZero = bic.numCorModel = bic.numOverModel
       = 0
  bic.result = matrix(0,11,2)

  gcv.RE = gcv.PE = matrix(0,numSims,1)
  gcv.countCorrectZ = gcv.countNonzeroZ= gcv.countIncorrectZ = gcv.countCorrNonZero = gcv.numCorModel = gcv.numOverModel
       = 0
  gcv.result = matrix(0,11,2)

  cv.RE = cv.PE = matrix(0,numSims,1)
  cv.countCorrectZ = cv.countNonzeroZ= cv.countIncorrectZ = cv.countCorrNonZero = cv.numCorModel = cv.numOverModel= 0
  cv.result = matrix(0,11,2)

  for(i in 1:numSims)
  {
    glmFit = Criteria(bigY[i,],bigX[i,,])

    aic.RE[i] = sqrt(sum((glmFit$beta.aic-beta)^2))/sqrt(sum(beta^2))
    aic.PE[i] = mean(glmFit$yhat.aic!=bigY[i,])

    bic.RE[i] = sqrt(sum((glmFit$beta.bic-beta)^2))/sqrt(sum(beta^2))
    bic.PE[i] = mean(glmFit$yhat.bic!=bigY[i,])

    gcv.RE[i] = sqrt(sum((glmFit$beta.gcv-beta)^2))/sqrt(sum(beta^2))
    gcv.PE[i] = mean(glmFit$yhat.gcv!=bigY[i,])

    cv.RE[i] = sqrt(sum((glmFit$beta.cv-beta)^2))/sqrt(sum(beta^2))
    cv.PE[i] = mean(glmFit$yhat.cv!=bigY[i,])

    aic.countCorrectZ = aic.countCorrectZ + sum(glmFit$beta.aic[zero]==0)
    aic.countNonzeroZ =  aic.countNonzeroZ + sum(glmFit$beta.aic[nonzero]==0)
    aic.countIncorrectZ = aic.countIncorrectZ + sum(glmFit$beta.aic[zero]!=0)
    aic.countCorrNonZero = aic.countCorrNonZero + sum(glmFit$beta.aic[nonzero]!=0)
    aic.numCorModel = aic.numCorModel + sum(all(glmFit$beta.aic[zero]==0) && all(glmFit$beta.aic[nonzero] != 0))
    aic.numOverModel = aic.numOverModel + sum(all(glmFit$beta.aic[zero]==0)==F && all(glmFit$beta.aic[nonzero] != 0))

    bic.countCorrectZ = bic.countCorrectZ + sum(glmFit$beta.bic[zero]==0)
    bic.countNonzeroZ =  bic.countNonzeroZ + sum(glmFit$beta.bic[nonzero]==0)
    bic.countIncorrectZ = bic.countIncorrectZ + sum(glmFit$beta.bic[zero]!=0)
    bic.countCorrNonZero = bic.countCorrNonZero + sum(glmFit$beta.bic[nonzero]!=0)
    bic.numCorModel = bic.numCorModel + sum(all(glmFit$beta.bic[zero]==0) && all(glmFit$beta.bic[nonzero] != 0))
    bic.numOverModel = bic.numOverModel + sum(all(glmFit$beta.bic[zero]==0)==F && all(glmFit$beta.bic[nonzero] != 0))

    gcv.countCorrectZ = gcv.countCorrectZ + sum(glmFit$beta.gcv[zero]==0)
    gcv.countNonzeroZ =  gcv.countNonzeroZ + sum(glmFit$beta.gcv[nonzero]==0)
    gcv.countIncorrectZ = gcv.countIncorrectZ + sum(glmFit$beta.gcv[zero]!=0)
    gcv.countCorrNonZero = gcv.countCorrNonZero + sum(glmFit$beta.gcv[nonzero]!=0)
    gcv.numCorModel = gcv.numCorModel + sum(all(glmFit$beta.gcv[zero]==0) && all(glmFit$beta.gcv[nonzero] != 0))
    gcv.numOverModel = gcv.numOverModel + sum(all(glmFit$beta.gcv[zero]==0)==F && all(glmFit$beta.gcv[nonzero] != 0))

    cv.countCorrectZ = cv.countCorrectZ + sum(glmFit$beta.cv[zero]==0)
    cv.countNonzeroZ =  cv.countNonzeroZ + sum(glmFit$beta.cv[nonzero]==0)
    cv.countIncorrectZ = cv.countIncorrectZ + sum(glmFit$beta.cv[zero]!=0)
    cv.countCorrNonZero = cv.countCorrNonZero + sum(glmFit$beta.cv[nonzero]!=0)
    cv.numCorModel = cv.numCorModel + sum(all(glmFit$beta.cv[zero]==0) && all(glmFit$beta.cv[nonzero] != 0))
    cv.numOverModel = cv.numOverModel + sum(all(glmFit$beta.cv[zero]==0)==F && all(glmFit$beta.cv[nonzero] != 0))
  }

  aic.result[1,1] = paste(type,"_AIC_LOG_", casename[case], sep = "")
  bic.result[1,1] = paste(type,"_BIC_LOG_", casename[case], sep = "")
  gcv.result[1,1] = paste(type,"_GCV_LOG_", casename[case], sep = "")
  cv.result[1,1] = paste(type,"_CV_LOG_", casename[case], sep = "")


  aic.result[2:11,1] = bic.result[2:11,1] = gcv.result[2:11,1] = cv.result[2:11,1] =
    c("Mean RE: ", "Median RE: ", "Mean PE: ", "Median PE: ",
```

```
      "Average number of coefficients correctly shrunk to zero: ",
      "AVerage number of nonzero coefficients incorrectly shrunk to zero: ",
      "Average number of zero coefficients incorrectly selected: ",
      "Average number of nonzero coefficients correctly selected: ",
      "Proportion of correct models: ",
      "Proportion of overfitted models: ")

  aic.result[,2] = c(NA, mean(aic.RE), median(aic.RE), mean(aic.PE), median(aic.PE), aic.countCorrectZ/numSims, aic.
       countNonzeroZ/numSims,
                     aic.countIncorrectZ/numSims, aic.countCorrNonZero/numSims, aic.numCorModel/numSims, aic.
                          numOverModel/numSims)
  bic.result[,2] = c(NA, mean(bic.RE), median(bic.RE), mean(bic.PE), median(bic.PE), bic.countCorrectZ/numSims, bic.
       countNonzeroZ/numSims,
                     bic.countIncorrectZ/numSims, bic.countCorrNonZero/numSims, bic.numCorModel/numSims, bic.
                          numOverModel/numSims)
  gcv.result[,2] = c(NA, mean(gcv.RE), median(gcv.RE), mean(gcv.PE), median(gcv.PE), gcv.countCorrectZ/numSims, gcv.
       countNonzeroZ/numSims,
                     gcv.countIncorrectZ/numSims, gcv.countCorrNonZero/numSims, gcv.numCorModel/numSims, gcv.
                          numOverModel/numSims)
  cv.result[,2] = c(NA, mean(cv.RE), median(cv.RE), mean(cv.PE), median(cv.PE), cv.countCorrectZ/numSims, cv.
       countNonzeroZ/numSims,
                    cv.countIncorrectZ/numSims, cv.countCorrNonZero/numSims, cv.numCorModel/numSims, cv.numOverModel/
                         numSims)

  list("aic" = aic.result, "bic"=bic.result, "gcv"=gcv.result, "cv"=cv.result)
}


type="LASSO"

output = savefile(numSims, bigY, bigX, allbeta, zero, nonzero, type, case)

write.csv(output$aic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_AIC_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$bic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_BIC_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$gcv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_GCV_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$cv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_CV_LOG_%s.csv",
     type, casename[case]),quote = FALSE)

################################################
# AIC BIC CV and GCV for RIDGE
################################################

Criteria <- function(y,x){
  n=length(y)

  glmFit = glmnet(x,y, family = "binomial", intercept=FALSE ,alpha = 0)
  yhat = predict(glmFit, x, type = "response")
  loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)

  aic = -2*loglik + 2*(glmFit$df)
  lambda = glmFit$lambda[which.min(aic)]
  glmFitbeta = predict(glmFit, s=lambda, type = "coef")
  beta.aic = as.vector(glmFitbeta)[-1]
  yhat.aic = predict(glmFit, newx=x, s=lambda, type = "class")

  bic = -2*loglik + log(n)*(glmFit$df)
  lambda = glmFit$lambda[which.min(bic)]
  glmFitbeta = predict(glmFit, s=lambda, type = "coef")
  beta.bic = as.vector(glmFitbeta)[-1]
  yhat.bic = predict(glmFit, newx=x, s=lambda, type = "class")

  gcv = -2*loglik/(n*(1-(glmFit$df/n))^2)
  lambda = glmFit$lambda[which.min(gcv)]
  glmFitbeta = predict(glmFit, s=lambda, type = "coef")
  beta.gcv = as.vector(glmFitbeta)[-1]
  yhat.gcv = predict(glmFit, newx=x, s=lambda, type = "class")

  cvfit=cv.glmnet(x,y,foldid=foldid,family="binomial", alpha=0,intercept=F)
  lambda=cvfit$lambda[which.min(cvfit$cvm)]
  glmFitbeta = predict(glmFit, s=lambda, type = "coef")
  beta.cv = as.vector(glmFitbeta)[-1]
  yhat.cv = predict(glmFit, newx=x, s=lambda, type = "class")

  list("beta.aic" = beta.aic, "yhat.aic"=yhat.aic, "beta.bic"=beta.bic, "yhat.bic"=yhat.bic, "beta.gcv"=beta.gcv, "yhat.
       gcv"=yhat.gcv, "beta.cv"=beta.cv, "yhat.cv"=yhat.cv)
}

type="RIDGE"

output = savefile(numSims, bigY, bigX, allbeta, zero, nonzero, type, case)

write.csv(output$aic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_AIC_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$bic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_BIC_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$gcv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_GCV_LOG_%s.csv",
```

```
     type, casename[case]),quote = FALSE)
write.csv(output$cv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_CV_LOG_%s.csv",
     type, casename[case]),quote = FALSE)

########################################
#elastic net AIC BIC CV and GCV
########################################

Criteria <- function(y,x){
  myalpha = seq(0,1,length = 100)
  BigAIC = BigBIC = BigGCV = BigCV = matrix(0, length(myalpha), 3)
  n=length(y)
  index=1
  for(a in myalpha){
    glmFit = glmnet(x,y, family = "binomial", intercept=F ,alpha = a)
    yhat = predict(glmFit, x, type = "response")
    loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)

    aic = -2*loglik + 2*(glmFit$df)
    BigAIC[index,] = c(a, glmFit$lambda[which.min(aic)], min(aic))

    bic = -2*loglik + log(n)*(glmFit$df)
    BigBIC[index,] = c(a, glmFit$lambda[which.min(bic)], min(bic))

    gcv = -2*loglik/(n*(1-(glmFit$df/n))^2)
    BigGCV[index,] = c(a, glmFit$lambda[which.min(gcv)], min(gcv))

    cvfit=cv.glmnet(x,y,foldid=foldid,family="binomial", alpha=a, intercept=F)
    BigCV[index,]=c(a, cvfit$lambda[which.min(cvfit$cvm)], min(cvfit$cvm))

    index = index+1
  }
  Min.aic = which.min(BigAIC[,3])
  Min.bic = which.min(BigBIC[,3])
  Min.gcv = which.min(BigGCV[,3])
  Min.cv = which.min(BigCV[,3])

  glmFit = glmnet(x,y, family = "binomial", intercept=F ,alpha = BigAIC[Min.aic,1])
  glmFitbeta = predict(glmFit, s=BigAIC[Min.aic,2], type = "coef")
  beta.aic = as.vector(glmFitbeta)[-1]
  yhat.aic = predict(glmFit, newx=x, s=BigAIC[Min.aic,2], type = "class")

  glmFit = glmnet(x,y, family = "binomial", intercept=F ,alpha = BigBIC[Min.bic,1])
  glmFitbeta = predict(glmFit, s=BigBIC[Min.bic,2], type = "coef")
  beta.bic = as.vector(glmFitbeta)[-1]
  yhat.bic = predict(glmFit, newx=x, s=BigBIC[Min.bic,2], type = "class")

  glmFit = glmnet(x,y, family = "binomial", intercept=F ,alpha = BigGCV[Min.gcv,1])
  glmFitbeta = predict(glmFit, s=BigGCV[Min.gcv,2], type = "coef")
  beta.gcv = as.vector(glmFitbeta)[-1]
  yhat.gcv = predict(glmFit, newx=x, s=BigGCV[Min.gcv,2], type = "class")

  glmFit = glmnet(x,y, family = "binomial", intercept=F ,alpha = BigCV[Min.cv,1])
  glmFitbeta = predict(glmFit, s=BigCV[Min.cv,2], type = "coef")
  beta.cv = as.vector(glmFitbeta)[-1]
  yhat.cv = predict(glmFit, newx=x, s=BigCV[Min.cv,2], type = "class")

  list("beta.aic" = beta.aic, "yhat.aic"=yhat.aic, "beta.bic"=beta.bic, "yhat.bic"=yhat.bic, "beta.gcv"=beta.gcv, "yhat.
       gcv"=yhat.gcv, "beta.cv"=beta.cv, "yhat.cv"=yhat.cv)
}

type="ENET"

output = savefile(numSims, bigY, bigX, allbeta, zero, nonzero, type, case)

write.csv(output$aic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_AIC_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$bic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_BIC_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$gcv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_GCV_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$cv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_CV_LOG_%s.csv",
     type, casename[case]),quote = FALSE)
```

A.7 Uncorrelated Multinomial Data Generation R Code

```
##############################################
# initialize libraries
##############################################
library(glmnet)
library(multinomRob)
library(lars)
library(MASS)
##############################################
# initialize data
##############################################
n=100
```

```
p = 8
beta1 = c(1.5, 3, 0,0,2,0,0,0)
beta2 = c(2, 0, 2,0,0,3,0,0)
beta3 = c(3, 0, 0,0,1.5,0,2,0)
numSims = 1000
bigX = array(0, dim = c(numSims, n, p))
bigY = matrix(0,numSims, n)


for(i in 1:numSims)
{
  bigX[i,,] = matrix(rnorm(n*p),n,p)
  bigX[i,,] = scale(bigX[i,,], T,T)

  for(j in 1:n)
  {
    t1= exp(bigX[i,j,]%*%beta1)
    t2= exp(bigX[i,j,]%*%beta2)
    t3= exp(bigX[i,j,]%*%beta3)

    p1= t1/sum(t1,t2,t3)
    p2= t2/sum(t1,t2,t3)
    p3= t3/sum(t1,t2,t3)
    # using three probabilities, randomly generate one value: for example, myP will be (0,1,0). Then, y is assigned to
        the second group.
    myP=rmultinomial(1, pr = c(p1, p2, p3), long = FALSE)
    bigY[i,j] = which.max(myP)
  }
}
```

A.8 Correlated Multinomial Data Generation R Code

```
##############################################
# initialize libraries
##############################################
library(glmnet)
library(multinomRob)
library(lars)
library(MASS)
##############################################
# initialize data
##############################################
n=100
p = 8
beta1 = c(1.5, 3, 0,0,2,0,0,0)
beta2 = c(2, 0, 2,0,0,3,0,0)
beta3 = c(3, 0, 0,0,1.5,0,2,0)
numSims = 1000
bigX = array(0, dim = c(numSims, n, p))
bigY = matrix(0,numSims, n)
MU <- matrix(0,8,1)
SIGMA <- matrix(0, 8,8)

for(k in 1:numSims)
{
  rho=0.5
  for (i in 1:8){
    for (j in 1:8){
      SIGMA[i,j] =rho^{abs(i-j)}
    }
  }
  X = mvrnorm(100, mu=MU, Sigma=SIGMA)
  x = scale(x,T,T)
  bigX[k,,] = X

  for(j in 1:n)
  {
    t1= exp(bigX[k,j,]%*%beta1)
    t2= exp(bigX[k,j,]%*%beta2)
    t3= exp(bigX[k,j,]%*%beta3)

    p1= t1/sum(t1,t2,t3)
    p2= t2/sum(t1,t2,t3)
    p3= t3/sum(t1,t2,t3)
    # using three probabilities, randomly generate one value: for example, myP will be (0,1,0). Then, y is assigned to
        the second group.
    myP=rmultinomial(1, pr = c(p1, p2, p3), long = FALSE)
    bigY[k,j] = which.max(myP)
  }
}
```

A.9 Multinomial Logistic Model Simulation

```
# for CV
```

```
K=10
foldid = sample(rep(1:K, length = n))

# Multinomail Logistic Model Simulation
########################################
# Oracle SIM
########################################

a=matrix(allbeta,nrow=2,byrow=T)
a2=apply(a!=0,2,sum)
exclude=which(a2==0)

RE = matrix(0,numSims,1)
PE = matrix(0,numSims,1)
countCorrectZ = countNonzeroZ= countIncorrectZ = countCorrNonZero = numCorModel = numOverModel= 0
result = matrix(0,11,2)
for(i in 1:numSims)
{
  glmFit = glmnet(bigX[i,,], as.factor(bigY[i,]), family = "multinomial", intercept=F ,alpha = 0, lambda = 0, exclude=
        exclude)

  fit = predict(glmFit, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  fit = c(fit2-fit1,fit3-fit1)

  yhat= predict(glmFit, bigX[i,,], type = "class", exclude=exclude)

  RE[i] = sqrt(sum((fit-allbeta)^2))/sqrt(sum(allbeta^2))
  PE[i] = mean(yhat!=bigY[i,])

  countCorrectZ = countCorrectZ + sum(fit[zero]==0)
  countNonzeroZ = countNonzeroZ + sum(fit[nonzero]==0)
  countIncorrectZ = countIncorrectZ + sum(fit[zero]!=0)
  countCorrNonZero = countCorrNonZero + sum(fit[nonzero]!=0)
  numCorModel = numCorModel + sum(all(fit[zero]==0) && all(fit[nonzero] != 0))
  numOverModel = numOverModel + sum(all(fit[zero]==0)==F && all(fit[nonzero] != 0))
}
result[1,1] = "OracleMULsim_corr"
result[2,] = c( "Mean RE: ", mean(RE))
result[3,] = c( "Median RE: ", median(RE))
result[4,] = c( "Mean PE: ", mean(PE))
result[5,] = c( "Median PE: ", median(PE))
result[6,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[7,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzeroZ/numSims)
result[8,] = c( "Average number of zero coefficients incorrectly selected: ", countIncorrectZ/numSims)
result[9,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[10,] = c( "Proportion of correct models: ", numCorModel/numSims)
result[11,] = c( "Proportion of overfitted models: ", numOverModel/numSims)
result

write.csv(result,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/Oracle_MUL_%s.csv",
      casename[case]),quote = FALSE)

########################################
# OLS SIM
########################################

RE = matrix(0,numSims,1)
PE = matrix(0,numSims,1)
countCorrectZ = countNonzeroZ= countIncorrectZ = countCorrNonZero = numCorModel = numOverModel= 0
result = matrix(0,11,2)
for(i in 1:numSims)
{
  glmFit = glmnet(bigX[i,,], as.factor(bigY[i,]), family = "multinomial", intercept=F ,alpha = 0, lambda = 0)

  fit = predict(glmFit, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  fit = c(fit2-fit1,fit3-fit1)

  yhat= predict(glmFit, bigX[i,,], type = "class")

  RE[i] = sqrt(sum((fit-allbeta)^2))/sqrt(sum(allbeta^2))
  PE[i] = mean(yhat!=bigY[i,])

  countCorrectZ = countCorrectZ + sum(fit[zero]==0)
  countNonzeroZ =  countNonzeroZ + sum(fit[nonzero]==0)
  countIncorrectZ = countIncorrectZ + sum(fit[zero]!=0)
  countCorrNonZero = countCorrNonZero + sum(fit[nonzero]!=0)
  numCorModel = numCorModel + sum(all(fit[zero]==0) && all(fit[nonzero] != 0))
  numOverModel = numOverModel + sum(all(fit[zero]==0)==F && all(fit[nonzero] != 0))
}
result[1,1] = "OLSMULsim_corr"
result[2,] = c( "Mean RE: ", mean(RE))
result[3,] = c( "Median RE: ", median(RE))
result[4,] = c( "Mean PE: ", mean(PE))
result[5,] = c( "Median PE: ", median(PE))
result[6,] = c( "Average number of coefficients correctly shrunk to zero: ", countCorrectZ/numSims)
result[7,] = c( "AVerage number of nonzero coefficients incorrectly shrunk to zero: ", countNonzeroZ/numSims)
result[8,] = c( "Average number of zero coefficients incorrectly selected: ", countIncorrectZ/numSims)
```

```
result[9,] = c( "Average number of nonzero coefficients correctly selected: ", countCorrNonZero/numSims)
result[10,] = c( "Proportion of correct models: ", numCorModel/numSims)
result[11,] = c( "Proportion of overfitted models: ", numOverModel/numSims)
result

write.csv(result,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/OLS_MUL_%s.csv", casename
      [case]),quote = FALSE)


#######################################################
# AIC BIC CV and GCV for LASSO
#######################################################

Criteria <- function(y,x){
  n=length(y)
  glmFit = glmnet(x,y, family = "multinomial", intercept=F ,alpha = 1)

  aic = (1-glmFit$dev.ratio)*glmFit$nulldev + 2*(apply(glmFit$dfmat,2,sum)+1)
  lambda = glmFit$lambda[which.min(aic)]
  fit = predict(glmFit, s=lambda, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.aic = c(fit2-fit1,fit3-fit1)
  yhat.aic = predict(glmFit, newx=x, s=lambda, type = "class")

  bic = (1-glmFit$dev.ratio)*glmFit$nulldev + log(n)*(apply(glmFit$dfmat,2,sum)+1)
  lambda = glmFit$lambda[which.min(bic)]
  fit = predict(glmFit, s=lambda, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.bic = c(fit2-fit1,fit3-fit1)
  yhat.bic = predict(glmFit, newx=x, s=lambda, type = "class")

  gcv = (1-glmFit$dev.ratio)*glmFit$nulldev/(n*(1-((apply(glmFit$dfmat,2,sum)+1)/n))^2)
  lambda = glmFit$lambda[which.min(gcv)]
  fit = predict(glmFit, s=lambda, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.gcv = c(fit2-fit1,fit3-fit1)
  yhat.gcv = predict(glmFit, newx=x, s=lambda, type = "class")

  cvfit=cv.glmnet(x,y,foldid=foldid,family="multinomial",intercept=F, alpha=1)
  lambda=cvfit$lambda[which.min(cvfit$cvm)]
  fit = predict(glmFit, s=lambda, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.cv = c(fit2-fit1,fit3-fit1)
  yhat.cv = predict(glmFit, newx=x, s=lambda, type = "class")

  list("beta.aic" = beta.aic, "yhat.aic"=yhat.aic, "beta.bic"=beta.bic, "yhat.bic"=yhat.bic, "beta.gcv"=beta.gcv, "yhat.
       gcv"=yhat.gcv, "beta.cv"=beta.cv, "yhat.cv"=yhat.cv)
}


########################################
# LASSO simulation
########################################


savefile <- function(numSims, bigY, bigX, allbeta, zero, nonzero, type, case){
  aic.RE = aic.PE = matrix(0,numSims,1)
  aic.countCorrectZ = aic.countNonzeroZ= aic.countIncorrectZ = aic.countCorrNonZero = aic.numCorModel = aic.numOverModel
       = 0
  aic.result = matrix(0,11,2)

  bic.RE = bic.PE = matrix(0,numSims,1)
  bic.countCorrectZ = bic.countNonzeroZ= bic.countIncorrectZ = bic.countCorrNonZero = bic.numCorModel = bic.numOverModel
       = 0
  bic.result = matrix(0,11,2)

  gcv.RE = gcv.PE = matrix(0,numSims,1)
  gcv.countCorrectZ = gcv.countNonzeroZ= gcv.countIncorrectZ = gcv.countCorrNonZero = gcv.numCorModel = gcv.numOverModel
       = 0
  gcv.result = matrix(0,11,2)

  cv.RE = cv.PE = matrix(0,numSims,1)
  cv.countCorrectZ = cv.countNonzeroZ= cv.countIncorrectZ = cv.countCorrNonZero = cv.numCorModel = cv.numOverModel= 0
  cv.result = matrix(0,11,2)

  for(i in 1:numSims)
  {
    glmFit = Criteria(bigY[i,],bigX[i,,])

    aic.RE[i] = sqrt(sum((glmFit$beta.aic-allbeta)^2))/sqrt(sum(allbeta^2))
    aic.PE[i] = mean(glmFit$yhat.aic!=bigY[i,])

    bic.RE[i] = sqrt(sum((glmFit$beta.bic-allbeta)^2))/sqrt(sum(allbeta^2))
    bic.PE[i] = mean(glmFit$yhat.bic!=bigY[i,])

    gcv.RE[i] = sqrt(sum((glmFit$beta.gcv-allbeta)^2))/sqrt(sum(allbeta^2))
    gcv.PE[i] = mean(glmFit$yhat.gcv!=bigY[i,])

    cv.RE[i] = sqrt(sum((glmFit$beta.cv-allbeta)^2))/sqrt(sum(allbeta^2))
```

```
    cv.PE[i] = mean(glmFit$yhat.cv!=bigY[i,])

    aic.countCorrectZ = aic.countCorrectZ + sum(glmFit$beta.aic[zero]==0)
    aic.countNonzeroZ =  aic.countNonzeroZ + sum(glmFit$beta.aic[nonzero]==0)
    aic.countIncorrectZ = aic.countIncorrectZ + sum(glmFit$beta.aic[zero]!=0)
    aic.countCorrNonZero = aic.countCorrNonZero + sum(glmFit$beta.aic[nonzero]!=0)
    aic.numCorModel = aic.numCorModel + sum(all(glmFit$beta.aic[zero]==0) && all(glmFit$beta.aic[nonzero] != 0))
    aic.numOverModel = aic.numOverModel + sum(all(glmFit$beta.aic[zero]==0)==F && all(glmFit$beta.aic[nonzero] != 0))

    bic.countCorrectZ = bic.countCorrectZ + sum(glmFit$beta.bic[zero]==0)
    bic.countNonzeroZ =  bic.countNonzeroZ + sum(glmFit$beta.bic[nonzero]==0)
    bic.countIncorrectZ = bic.countIncorrectZ + sum(glmFit$beta.bic[zero]!=0)
    bic.countCorrNonZero = bic.countCorrNonZero + sum(glmFit$beta.bic[nonzero]!=0)
    bic.numCorModel = bic.numCorModel + sum(all(glmFit$beta.bic[zero]==0) && all(glmFit$beta.bic[nonzero] != 0))
    bic.numOverModel = bic.numOverModel + sum(all(glmFit$beta.bic[zero]==0)==F && all(glmFit$beta.bic[nonzero] != 0))

    gcv.countCorrectZ = gcv.countCorrectZ + sum(glmFit$beta.gcv[zero]==0)
    gcv.countNonzeroZ =  gcv.countNonzeroZ + sum(glmFit$beta.gcv[nonzero]==0)
    gcv.countIncorrectZ = gcv.countIncorrectZ + sum(glmFit$beta.gcv[zero]!=0)
    gcv.countCorrNonZero = gcv.countCorrNonZero + sum(glmFit$beta.gcv[nonzero]!=0)
    gcv.numCorModel = gcv.numCorModel + sum(all(glmFit$beta.gcv[zero]==0) && all(glmFit$beta.gcv[nonzero] != 0))
    gcv.numOverModel = gcv.numOverModel + sum(all(glmFit$beta.gcv[zero]==0)==F && all(glmFit$beta.gcv[nonzero] != 0))

    cv.countCorrectZ = cv.countCorrectZ + sum(glmFit$beta.cv[zero]==0)
    cv.countNonzeroZ =  cv.countNonzeroZ + sum(glmFit$beta.cv[nonzero]==0)
    cv.countIncorrectZ = cv.countIncorrectZ + sum(glmFit$beta.cv[zero]!=0)
    cv.countCorrNonZero = cv.countCorrNonZero + sum(glmFit$beta.cv[nonzero]!=0)
    cv.numCorModel = cv.numCorModel + sum(all(glmFit$beta.cv[zero]==0) && all(glmFit$beta.cv[nonzero] != 0))
    cv.numOverModel = cv.numOverModel + sum(all(glmFit$beta.cv[zero]==0)==F && all(glmFit$beta.cv[nonzero] != 0))
  }
  aic.result[1,1] = paste(type,"_AIC_MUL_", casename[case], sep = "")
  bic.result[1,1] = paste(type,"_BIC_MUL_", casename[case], sep = "")
  gcv.result[1,1] = paste(type,"_GCV_MUL_", casename[case], sep = "")
  cv.result[1,1] = paste(type,"_CV_MUL_", casename[case], sep = "")

  aic.result[2:11,1] = bic.result[2:11,1] = gcv.result[2:11,1] = cv.result[2:11,1] =
    c("Mean RE: ", "Median RE: ", "Mean PE: ", "Median PE: ",
      "Average number of coefficients correctly shrunk to zero: ",
      "AVerage number of nonzero coefficients incorrectly shrunk to zero: ",
      "Average number of zero coefficients incorrectly selected: ",
      "Average number of nonzero coefficients correctly selected: ",
      "Proportion of correct models: ",
      "Proportion of overfitted models: ")

  aic.result[,2] = c(NA, mean(aic.RE), median(aic.RE), mean(aic.PE), median(aic.PE), aic.countCorrectZ/numSims, aic.
       countNonzeroZ/numSims,
                     aic.countIncorrectZ/numSims, aic.countCorrNonZero/numSims, aic.numCorModel/numSims, aic.
                          numOverModel/numSims)
  bic.result[,2] = c(NA, mean(bic.RE), median(bic.RE), mean(bic.PE), median(bic.PE), bic.countCorrectZ/numSims, bic.
       countNonzeroZ/numSims,
                     bic.countIncorrectZ/numSims, bic.countCorrNonZero/numSims, bic.numCorModel/numSims, bic.
                          numOverModel/numSims)
  gcv.result[,2] = c(NA, mean(gcv.RE), median(gcv.RE), mean(gcv.PE), median(gcv.PE), gcv.countCorrectZ/numSims, gcv.
       countNonzeroZ/numSims,
                     gcv.countIncorrectZ/numSims, gcv.countCorrNonZero/numSims, gcv.numCorModel/numSims, gcv.
                          numOverModel/numSims)
  cv.result[,2] = c(NA, mean(cv.RE), median(cv.RE), mean(cv.PE), median(cv.PE), cv.countCorrectZ/numSims, cv.
       countNonzeroZ/numSims,
                    cv.countIncorrectZ/numSims, cv.countCorrNonZero/numSims, cv.numCorModel/numSims, cv.numOverModel/
                         numSims)

  list("aic" = aic.result, "bic"=bic.result, "gcv"=gcv.result, "cv"=cv.result)
}


type="LASSO"

output = savefile(numSims, bigY, bigX, allbeta, zero, nonzero, type, case)

write.csv(output$aic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_AIC_MUL_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$bic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_BIC_MUL_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$gcv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_GCV_MUL_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$cv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_CV_MUL_%s.csv",
     type, casename[case]),quote = FALSE)

########################################
# AIC BIC CV and GCV for ridge regression
########################################


Criteria <- function(y,x){
  n=length(y)
  glmFit = glmnet(x,y, family = "multinomial", intercept=F ,alpha = 0)

  aic = (1-glmFit$dev.ratio)*glmFit$nulldev + 2*(apply(glmFit$dfmat,2,sum)+1)
  lambda = glmFit$lambda[which.min(aic)]
```

```
  fit = predict(glmFit, s=lambda, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.aic = c(fit2-fit1,fit3-fit1)
  yhat.aic = predict(glmFit, newx=x, s=lambda, type = "class")

  bic = (1-glmFit$dev.ratio)*glmFit$nulldev + log(n)*(apply(glmFit$dfmat,2,sum)+1)
  lambda = glmFit$lambda[which.min(bic)]
  fit = predict(glmFit, s=lambda, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.bic = c(fit2-fit1,fit3-fit1)
  yhat.bic = predict(glmFit, newx=x, s=lambda, type = "class")

  gcv = (1-glmFit$dev.ratio)*glmFit$nulldev/(n*(1-((apply(glmFit$dfmat,2,sum)+1)/n))^2)
  lambda = glmFit$lambda[which.min(gcv)]
  fit = predict(glmFit, s=lambda, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.gcv = c(fit2-fit1,fit3-fit1)
  yhat.gcv = predict(glmFit, newx=x, s=lambda, type = "class")

  cvfit=cv.glmnet(x,y,foldid=foldid,family="multinomial",intercept=F, alpha=0)
  lambda=cvfit$lambda[which.min(cvfit$cvm)]
  fit = predict(glmFit, s=lambda, type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.cv = c(fit2-fit1,fit3-fit1)
  yhat.cv = predict(glmFit, newx=x, s=lambda, type = "class")

  list("beta.aic" = beta.aic, "yhat.aic"=yhat.aic, "beta.bic"=beta.bic, "yhat.bic"=yhat.bic, "beta.gcv"=beta.gcv, "yhat.
       gcv"=yhat.gcv, "beta.cv"=beta.cv, "yhat.cv"=yhat.cv)
}


type="RIDGE"

output = savefile(numSims, bigY, bigX, allbeta, zero, nonzero, type, case)

write.csv(output$aic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_AIC_MUL_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$bic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_BIC_MUL_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$gcv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_GCV_MUL_%s.csv",
     type, casename[case]),quote = FALSE)
write.csv(output$cv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_CV_MUL_%s.csv",
     type, casename[case]),quote = FALSE)

########################################
#elastic net AIC BIC CV and GCV
########################################

Criteria <- function(y,x){
  myalpha = seq(0,1,length = 100)
  BigAIC = BigBIC = BigGCV = BigCV = matrix(0, length(myalpha), 3)
  n=length(y)
  index=1
  for(a in myalpha){
    glmFit = glmnet(x,y, family = "multinomial", intercept=F ,alpha = a)

    aic = (1-glmFit$dev.ratio)*glmFit$nulldev + 2*(apply(glmFit$dfmat,2,sum)+1)
    BigAIC[index,] = c(a, glmFit$lambda[which.min(aic)], min(aic))

    bic = (1-glmFit$dev.ratio)*glmFit$nulldev + log(n)*(apply(glmFit$dfmat,2,sum)+1)
    BigBIC[index,] = c(a, glmFit$lambda[which.min(bic)], min(bic))

    gcv = (1-glmFit$dev.ratio)*glmFit$nulldev/(n*(1-((apply(glmFit$dfmat,2,sum)+1)/n))^2)
    BigGCV[index,] = c(a, glmFit$lambda[which.min(gcv)], min(gcv))

    cvfit=cv.glmnet(x,y,foldid=foldid,family="multinomial", alpha=a, intercept=F)
    BigCV[index,]=c(a, cvfit$lambda[which.min(cvfit$cvm)], min(cvfit$cvm))

    index = index+1
  }
  Min.aic = which.min(BigAIC[,3])
  Min.bic = which.min(BigBIC[,3])
  Min.gcv = which.min(BigGCV[,3])
  Min.cv = which.min(BigCV[,3])

  glmFit = glmnet(x,y, family = "multinomial", intercept=F ,alpha = BigAIC[Min.aic,1])
  fit = predict(glmFit, s=BigAIC[Min.aic,2], type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.aic = c(fit2-fit1,fit3-fit1)
  yhat.aic = predict(glmFit, newx=x, s=BigAIC[Min.aic,2], type = "class")

  glmFit = glmnet(x,y, family = "multinomial", intercept=F ,alpha = BigBIC[Min.bic,1])
  fit = predict(glmFit, s=BigBIC[Min.bic,2], type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.bic = c(fit2-fit1,fit3-fit1)
  yhat.bic = predict(glmFit, newx=x, s=BigBIC[Min.bic,2], type = "class")

  glmFit = glmnet(x,y, family = "multinomial", intercept=F ,alpha = BigGCV[Min.gcv,1])
  fit = predict(glmFit, s=BigGCV[Min.gcv,2], type = "coef")
```

```
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.gcv = c(fit2-fit1,fit3-fit1)
  yhat.gcv = predict(glmFit, newx=x, s=BigGCV[Min.gcv,2], type = "class")

  glmFit = glmnet(x,y, family = "multinomial", intercept=F ,alpha = BigCV[Min.cv,1])
  fit = predict(glmFit, s=BigCV[Min.cv,2], type = "coef")
  fit1 = as.numeric(fit$"1"[-1]); fit2 = as.numeric(fit$"2"[-1]); fit3 = as.numeric(fit$"3"[-1]);
  beta.cv = c(fit2-fit1,fit3-fit1)
  yhat.cv = predict(glmFit, newx=x, s=BigCV[Min.cv,2], type = "class")

  list("beta.aic" = beta.aic, "yhat.aic"=yhat.aic, "beta.bic"=beta.bic, "yhat.bic"=yhat.bic, "beta.gcv"=beta.gcv, "yhat.
       gcv"=yhat.gcv, "beta.cv"=beta.cv, "yhat.cv"=yhat.cv)
}


type="ENET"

output = savefile(numSims, bigY, bigX, allbeta, zero, nonzero, type, case)

write.csv(output$aic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_AIC_MUL_%s.csv",
      type, casename[case]),quote = FALSE)
write.csv(output$bic,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_BIC_MUL_%s.csv",
      type, casename[case]),quote = FALSE)
write.csv(output$gcv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_GCV_MUL_%s.csv",
      type, casename[case]),quote = FALSE)
write.csv(output$cv,sprintf("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/SimulationOutput/%s_CV_MUL_%s.csv",
      type, casename[case]),quote = FALSE)
```

A.10 NYU Binary Logistic Data Analysis

```
#########################################################
# include required function
#########################################################
library(glmnet)

#########################################################
# get data
#########################################################
training = read.csv("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/491 data/ADHDdata/ADHD200 training set
      phenotypic information/NYU_phenotypic.csv", header = TRUE)
testing = read.csv("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/491 data/ADHDdata/ADHD200 testing set phenotypic
       information/NYUtest.csv", header = TRUE)
x_train = training[,-c(1,2,6,7)]
x_train = as.matrix(x_train)
y_train = training[,7]

x_test = testing[,-c(1,2,3,7,8)]
x_test = as.matrix(x_test)
y_test = testing[,8]

############################################## summary statistics : before standardizing

apply(x_train,2,length) # n
apply(x_train,2,mean)   # mean
apply(x_train,2,sd)     # sd
apply(x_train,2,min)    # min
apply(x_train,2,max)    # max

# gender
mydata <- table(x_train[,1], y_train)
mydata <- matrix(c(mydata), nrow=2, ncol=2, dimnames=list(c("Female", "Male"), c("TDC", "ADHD")))
barplot(mydata,xlab="Diagnosis",ylab="Frequency",ylim=c(0,100),horiz=F,
        beside=TRUE,space=c(0.2,1),col=gray.colors(2)[2:1],legend.text=FALSE)
legend(x=1.5,y=90,legend=c("Female", "Male"),fill=gray.colors(2)[2:1])

# AGE
boxplot(x_train[,2] ~ y_train, names=c("TDC", "ADHD") ,ylab="age", horizontal=F,las=1)
# Handedness
boxplot(x_train[,3] ~ y_train, names=c("TDC", "ADHD") ,ylab="Handedness", horizontal=F,las=1)
# ADHD index
boxplot(x_train[,4] ~ y_train, names=c("TDC", "ADHD") ,ylab="ADHD index", horizontal=F,las=1)
# inattentive measure
boxplot(x_train[,5] ~ y_train, names=c("TDC", "ADHD") ,ylab="Inattentive Measure", horizontal=F,las=1)
# Hyperacive Impulsive measure
boxplot(x_train[,6] ~ y_train, names=c("TDC", "ADHD") ,ylab="Hyperactive/Impulsiv Measure", horizontal=F,las=1)
#  verbal IQ
boxplot(x_train[,7] ~ y_train, names=c("TDC", "ADHD") ,ylab="Verbal IQ", horizontal=F,las=1)
#  Performance IQ
boxplot(x_train[,8] ~ y_train, names=c("TDC", "ADHD") ,ylab="Performance IQ", horizontal=F,las=1)
#  FULL4 IQ
boxplot(x_train[,9] ~ y_train, names=c("TDC", "ADHD") ,ylab="Full4 IQ", horizontal=F,las=1)

############################################## Individual statistical tests
chisq.test(x_train[,1],y_train) # Gender:  TDC vs ADHD
t.test(x_train[y_train==0,2],x_train[y_train==1,2]) # Age:  TDC vs ADHD
t.test(x_train[y_train==0,3],x_train[y_train==1,3]) # Handedness:  TDC vs ADHD
t.test(x_train[y_train==0,4],x_train[y_train==1,4]) # ADHD index:  TDC vs ADHD
```

```
t.test(x_train[y_train==0,5],x_train[y_train==1,5]) # inattentive measure:  TDC vs ADHD
t.test(x_train[y_train==0,6],x_train[y_train==1,6]) # hyperactive impulsive measure:  TDC vs ADHD
t.test(x_train[y_train==0,7],x_train[y_train==1,7]) # verbal IQ:  TDC vs ADHD
t.test(x_train[y_train==0,8],x_train[y_train==1,8]) # Performace IQ:  TDC vs ADHD
t.test(x_train[y_train==0,9],x_train[y_train==1,9]) # Full4 IQ:  TDC vs ADHD

############################################## prepare an scale data
x_train= as.matrix(x_train)
x_test= as.matrix(x_test)

x_train[,-1] = scale(x_train[,-1], T,T)
x_test[,-1] = scale(x_test[,-1], T,T)

#######################################################
# considering a logistic regression model with no penalty
#######################################################

glmFit = glm(y_train ~ x_train, family = binomial)
summary(glmFit) # present p-values

anova(glmFit, test="Chisq") # goodness-of-fit test: if p<0.05, our model is good.

yhat = predict(glmFit, type="response")
yhat

# glm doesn't assign a group for each subject. Only provides its probability
# also, predict.glm requires to use data.frame.
trainset <- data.frame(y_train, x_train)
testset <- data.frame(y_test, x_test)

glmFit = glm(y_train ~ ., data=trainset, family = binomial)
predY0 = predict(glmFit, testset, type="response")
predY = ifelse(predY0>0.5, 1, 0)  # you may change this cut-off value
mean(predY != y_test) # classification error from a test set
table(predY, y_test)  # table for classification

# for training set
predY0 = predict(glmFit, trainset, type="response")
predY = ifelse(predY0>0.5, 1, 0)  # you may change this cut-off value
mean(predY != y_train) # classification error from a training set
table(predY, y_train)  # table for classification
#######################################################
# logistic regression model using lasso penalty determined by aic
#######################################################
### AIC lasso function
AIC <- function(y, x)
{
  mylambda = seq(0,.1, length = 100)
  aic = array(0, length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "binomial", intercept=TRUE ,alpha = 1, lambda = lam,nlambda = 1)
    yhat = predict(glmFit, x, type = "response")
    loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)


    aic[index] = -2*loglik + 2*(glmFit$df)
    index = index +1
  }
  lambda = mylambda[which.min(aic)]
  return(lambda)
}

######################### calculation
lam = AIC(y_train,x_train)
lam
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = 1, lambda = lam,nlambda = 1)

glmFit$beta
lam

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#######################################################
# logistic regression model using lasso penalty determined by bic
#######################################################
############BIC codes
BIC <- function(y,x,n)
```

```
{
  mylambda = seq(0,.1, length = 100)
  bic = array(0, length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "binomial", intercept=TRUE ,alpha = 1, lambda = lam,nlambda = 1)
    yhat = predict(glmFit, newx = x, type = "response")

    loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)
    bic[index] = -2*loglik + log(n)*(glmFit$df)

    index = index +1

  }
  lambda = mylambda[which.min(bic)]
  return(lambda)
}

######################result calculation
lam = BIC(y_train,x_train, length(y_train))
lam
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = 1, lambda = lam,nlambda = 1)

glmFit$beta

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#########################################################
# logistic regression model using lasso penalty determined by cv
#########################################################
######################CV codes
CV<-function (x, y, K = 10, index)
{
  all.folds <- cv.folds(length(y), K)

  residmat <- matrix(0, length(index), K)
  for (i in seq(K)) {
    omit <- all.folds[[i]]

    glmFit <- glmnet(x[-omit, , drop = FALSE], y[-omit], intercept = TRUE, alpha = 1, lambda = index, nlambda = length(
          index))
    fit <- predict(glmFit,newx= x[omit, , drop = FALSE],type = "response")

    residmat[, i] <- apply((y[omit] - fit)^2, 2, mean)
  }
  cv <- apply(residmat, 1, mean)
  cv.error <- sqrt(apply(residmat, 1, var)/K)
  lam = index[which.min(cv)]
  return(lam)
}

##### calculations
lam = CV(x_train,y_train, 10, seq(from = 0,to = .1, length = 100))
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = 1, lambda = lam,nlambda = 1)

lam

glmFit$beta

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#########################################################
# logistic regression model using lasso penalty determined by GCV
#########################################################
######GCV function
GCV <- function(y, x, n)
{
  mylambda = seq(0,1,length = 100)
  gcv = array(0, length(mylambda))
```

```
  index = 1
  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "binomial", intercept=TRUE ,alpha = 1, lambda = lam, nlambda = 1)
    yhat = predict(glmFit, x, type = "response")

    loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)
    gcv[index] = -2*loglik/(n*(1-(glmFit$df/n))^2)
    index = index +1
  }
  lam = mylambda[which.min(gcv)]
  return(mylambda[which.min(gcv)])
}

# data analysis
lam = GCV(y_train,x_train, length(y_train))
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = 1, lambda = lam,nlambda = 1)
lam
glmFit$beta

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#######################################################
# logistic regression model using ridge penalty determined by aic
#######################################################
# ridge aic function
AIC <- function( y, x)
{
  mylambda = seq(0,.1, length = 100)
  aic = array(0, length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "binomial", intercept=TRUE ,alpha = 0, lambda = lam,nlambda = 1)
    yhat = predict(glmFit,newx = x, type = "response")

    loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)

    aic[index] = -2*loglik + 2*(glmFit$df)
    index = index +1
  }
  lambda = mylambda[which.min(aic)]
  return(lambda)
}


lam = AIC(y_train,x_train)
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = 0, lambda = lam,nlambda = 1)
lam

glmFit$beta

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

#######################################################
# logistic regression model using ridge penalty determined by bic
#######################################################
# ridge regression BIC function
BIC <- function(y,x,n)
{
  mylambda = seq(0,.1, length = 100)
  bic = array(0, length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "binomial", intercept=TRUE ,alpha = 0, lambda = lam,nlambda = 1)
    yhat = predict(glmFit, newx = x, type = "response")
```

```
    loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)
    bic[index] = -2*loglik + log(n)*(glmFit$df)
    index = index +1
  }
  lambda = mylambda[which.min(bic)]
  return(lambda)
}

## application of ridge with BIC to binary logistic model
lam = BIC(y_train,x_train, length(x_train))
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = 0, lambda = lam,nlambda = 1)

lam

glmFit$beta

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#######################################################
# logistic regression model using ridge penalty determined by cv
#######################################################
# CV function for ridge regression
CV<-function (x, y, K = 10, index)
{

  all.folds <- cv.folds(length(y), K)

  residmat <- matrix(0, length(index), K)
  for (i in seq(K)) {
    omit <- all.folds[[i]]
    glmFit <- glmnet(x[-omit, , drop = FALSE], y[-omit], intercept = T, alpha = 0, lambda = index, nlambda = length(
          index))
    fit <- predict(glmFit, x[omit, , drop = FALSE],type = "response")

    residmat[, i] <- apply((y[omit] - fit)^2, 2, mean)
  }
  cv <- apply(residmat, 1, mean)
  cv.error <- sqrt(apply(residmat, 1, var)/K)
  lam = index[which.min(cv)]
  return(lam)

}

### Application of the CV selection criterion to the ridge method
lam = CV(x_train,y_train, 10, seq(from = 0,to = .1, length = 1000))
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = 0, lambda = lam,nlambda = 1)

lam
glmFit$beta

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#######################################################
# logistic regression model using ridge penalty determined by GCV
#######################################################
### GCV ridge regression implementation
GCV <- function(y, x, n)
{
  mylambda = seq(0,1,length = 100)
  gcv = array(0, length(mylambda))
  index = 1
  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "binomial", intercept=T ,alpha = 0, lambda = lam,nlambda = 1)
    yhat = predict(glmFit, x, type = "response")

    loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)
    gcv[index] = -2*loglik/(n*(1-(glmFit$df/n))^2)
    index = index +1
  }
  lam = mylambda[which.min(gcv)]
```

```
  return(mylambda[which.min(gcv)])
}

lam = GCV(y_train,x_train, length(y_train))
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = 0, lambda = lam,nlambda = 1)
lam

glmFit$beta

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

#######################################################
# logistic regression model using elastic net penalty determined by aic
#######################################################
AIC <- function(y, x)
{
  mylambda = seq(0,.1, length = 100)
  myalpha = seq(0,1,length = 100)
  aic = matrix(0, length(mylambda)*length(myalpha),3)
  index1 = 1

  for(a in myalpha)
  {
    for(lam in mylambda)
    {
      glmFit = glmnet(x,y, family = "binomial", intercept=T ,alpha = a, lambda = lam,nlambda = 1)
      yhat = predict(glmFit, x, type = "response")

      loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)
      aic[index1,] = c(-2*loglik + 2*(glmFit$df),lam,a)
      index1 = index1 +1
    }
  }
  min = which.min(aic[,1])
  list(alpha = aic[min,3], lambda = aic[min,2])

}

lam = AIC(y_train,x_train)
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = lam$alpha, lambda = lam$lambda,
      nlambda = 1)

glmFit$beta
lam$lambda
lam$alpha

glmFit$beta

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

#######################################################
# logistic regression model using elastic net penalty determined by bic
#######################################################
BIC <- function(y,x,n)
{
  mylambda = seq(0,.1, length = 100)
  myalpha = seq(0,1,length = 100)
  bic = matrix(0, length(mylambda)*length(myalpha),3)
  index1 = 1
  index2 = 1

  for(lam in mylambda)
  {
    for(a in myalpha)
    {
      glmFit = glmnet(x,y, family = "binomial", intercept=T ,alpha = a, lambda = lam,nlambda = 1)
      yhat = predict(glmFit, x, type = "response")

      loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)
```

```
      bic[index1,] = c(-2*loglik + log(n)*(glmFit$df),lam,a)
      index1 = index1 +1
    }
  }
  min = which.min(bic[,1])
  list(alpha = bic[min,3], lambda = bic[min,2])
}

lam = BIC(y_train,x_train, length(x_train))
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = lam$alpha, lambda = lam$lambda,
     nlambda = 1)

glmFit$beta
lam$lambda
lam$alpha

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

#######################################################
# logistic regression model using elasti net penalty determined by cv
#######################################################
CV<-function (x, y, K = 10, index, alph)
{
  tuningParam = NULL
  all.folds <- cv.folds(length(y), K)

  residmat <- matrix(0, length(index), K)
  for (i in seq(K)) {
    omit <- all.folds[[i]]
    glmFit <- glmnet(x[-omit, , drop = FALSE], y[-omit], intercept = T, alpha = alph, lambda = index, nlambda = length(
         index))
    fit <- predict(glmFit, x[omit, , drop = FALSE],type = "response")

    residmat[, i] <- apply((y[omit] - fit)^2, 2, mean)
  }
  cv <- apply(residmat, 1, mean)
  cv.error <- sqrt(apply(residmat, 1, var)/K)
  lam = index[which.min(cv)]

  tuningParam = c(lam, min(cv))

  return(tuningParam)
}
alpha = seq(from = 0, to = 1, length = 100)
lam = matrix(0, length(alpha),2)
index = 1
for( alph in alpha)
{
  lam[index,]=CV(x_train,y_train, 10, seq(from = 0,to = .1, length = 1000), alph)
  index = index +1
}
myLam = lam[which.min(lam[,2]),1]
myAlph = alpha[which.min(lam[,2])]
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = myAlph, lambda = myLam,nlambda =
     1)

glmFit$beta
myLam
myAlph
fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

#######################################################
# logistic regression model using elastic net penalty determined by GCV
#######################################################
GCV <- function(y, x, n)
{
  mylambda = seq(0,.1, length = 100)
  myalpha = seq(0,1,length = 100)
  gcv = matrix(0, length(mylambda)*length(myalpha),3)
```

```
  index1 = 1
  index2 = 1

  for(a in myalpha)
  {
    for(lam in mylambda)
    {
      glmFit = glmnet(x,y, family = "binomial", intercept = T, alpha = a, lambda = lam,nlambda = 1)
      yhat = predict(glmFit, x, type = "response")

      loglik = apply(y*log(yhat)+(1-y)*log(1-yhat),2,sum)
      gcv[index1,] = c(-2*loglik/(n*(1-(glmFit$df/n))^2),lam,a)
      index1 = index1 +1
    }
  }
  min = which.min(gcv[,1])
  list(alpha = gcv[min,3], lambda = gcv[min,2])
}

lam = GCV(y_train,x_train, length(y_train))
glmFit = glmnet(x = x_train, y = y_train, family = "binomial", intercept=TRUE ,alpha = lam$alpha, lambda = lam$lambda,
     nlambda = 1)

glmFit$beta
lam$lambda
lam$alpha

fit = predict(glmFit,newx = x_test, type = "response")

predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
fit = predict(glmFit, newx=x_train, s = 0, type = "response")
predY = ifelse(fit>0.5, 1, 0)
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
```

A.11 NYU Multinomial Logistic Data Analysis

```
#######################################################
# include required function
#######################################################
library(glmnet)
library(nnet)

#######################################################
# get data
#######################################################
training = read.csv("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/491 data/ADHDdata/ADHD200 training set
     phenotypic information/NYU_phenotypic.csv", header = TRUE)
testing = read.csv("C:/Users/Matthew/Documents/Matthew's Nevada/HON 490/491 data/ADHDdata/ADHD200 testing set phenotypic
      information/NYUtest.csv", header = TRUE)

new.training = training[training[,6]!=2,]  # exclude DX=2
new.testing = testing[testing[,7]!=2,]  # exclude DX=2

x_train = new.training[,-c(1,2,6,7)]
y_train = new.training[,6]
x_test = new.testing[,-c(1,2,3,7,8)]
y_test = new.testing[,7]

x_train= as.matrix(x_train)
x_test= as.matrix(x_test)

############################################## summary statistics : before standardizing

apply(x_train,2,length) # n
apply(x_train,2,mean)   # mean
apply(x_train,2,sd)     # sd
apply(x_train,2,min)    # min
apply(x_train,2,max)    # max

#################################################### plots: barplot, boxplot
# gender
mydata <- table(x_train[,1], y_train)
mydata <- matrix(c(mydata), nrow=2, ncol=3, dimnames=list(c("Female", "Male"), c("TDC", "ADHD-C", "ADHD-I")))
barplot(mydata,xlab="Diagnosis",ylab="Frequency",ylim=c(0,70),horiz=F,
       beside=TRUE,space=c(0.2,1),col=gray.colors(2)[2:1],legend.text=FALSE)
legend(x=8,y=60,legend=c("Female", "Male"),fill=gray.colors(2)[2:1])

# other variables
boxplot(x_train[,2] ~ y_train, names=c("TDC", "ADHD-C", "ADHD-I") ,ylab="age", horizontal=F,las=1)  # age
boxplot(x_train[,3] ~ y_train, names=c("TDC", "ADHD-C", "ADHD-I") ,ylab="Handedness", horizontal=F,las=1)  # Handedness
boxplot(x_train[,4] ~ y_train, names=c("TDC", "ADHD-C", "ADHD-I") ,ylab="ADHD Index", horizontal=F,las=1)  # ADHD index
```

```
boxplot(x_train[,5] ~ y_train, names=c("TDC", "ADHD-C", "ADHD-I") ,ylab="Inattentive Measure", horizontal=F,las=1)  #
     inattentive
boxplot(x_train[,6] ~ y_train, names=c("TDC", "ADHD-C", "ADHD-I") ,ylab="Hyperactive/IMpulsive Measure", horizontal=F,
     las=1)  # hyperactive impulsive
boxplot(x_train[,7] ~ y_train, names=c("TDC", "ADHD-C", "ADHD-I") ,ylab="Verbal IQ", horizontal=F,las=1)  # verbal IQ
boxplot(x_train[,8] ~ y_train, names=c("TDC", "ADHD-C", "ADHD-I") ,ylab="Performance IQ", horizontal=F,las=1)  #
     performcanc IQ
boxplot(x_train[,9] ~ y_train, names=c("TDC", "ADHD-C", "ADHD-I") ,ylab="Full4 IQ", horizontal=F,las=1)  # Full4 IQ

##################################### statistical tests individually
chisq.test(x_train[,1],y_train) # Gender:  TDC vs ADHD-C vs ADHD-I
anova(lm(x_train[,2]~y_train)) # Age:  TDC vs ADHD-C vs ADHD-I
anova(lm(x_train[,3]~y_train)) # Handedness:  TDC vs ADHD-C vs ADHD-I
anova(lm(x_train[,4]~y_train)) # ADHD Measure:  TDC vs ADHD-C vs ADHD-I
anova(lm(x_train[,5]~y_train)) # Inattentive:  TDC vs ADHD-C vs ADHD-I
anova(lm(x_train[,6]~y_train)) # Hyperactive/Impulsive:  TDC vs ADHD-C vs ADHD-I
anova(lm(x_train[,7]~y_train)) # Verbal IQ:  TDC vs ADHD-C vs ADHD-I
anova(lm(x_train[,8]~y_train)) # Performance IQ:  TDC vs ADHD-C vs ADHD-I
anova(lm(x_train[,9]~y_train)) # Full4 IQ:  TDC vs ADHD-C vs ADHD-I

x_train[,-1] = scale(x_train[,-1],T,T)
x_test[,-1] = scale(x_test[,-1],T,T)

#######################################################
# considering a logistic regression model with no penalty
#######################################################

glmFit = multinom(as.factor(y_train) ~ x_train, model=T)
summary(glmFit) # doesn't present p-values :(

z <- summary(glmFit)$coefficients/summary(glmFit)$standard.errors  # for calculating p-values for individual coefficient
p <- (1 - pnorm(abs(z), 0, 1)) * 2
p

# there are two coefficients for each variable because each DX has different model parameters.
# p-values above show individual p-values for each coefficient.
# we may simultaneously test the signficance of each "variable" using likelihood ratio test
temp1=anova(multinom(as.factor(y_train) ~ x_train[,-1]), glmFit, test="Chisq")
temp2=anova(multinom(as.factor(y_train) ~ x_train[,-2]), glmFit, test="Chisq")
temp3=anova(multinom(as.factor(y_train) ~ x_train[,-3]), glmFit, test="Chisq")
temp4=anova(multinom(as.factor(y_train) ~ x_train[,-4]), glmFit, test="Chisq")
temp5=anova(multinom(as.factor(y_train) ~ x_train[,-5]), glmFit, test="Chisq")
temp6=anova(multinom(as.factor(y_train) ~ x_train[,-6]), glmFit, test="Chisq")
temp7=anova(multinom(as.factor(y_train) ~ x_train[,-7]), glmFit, test="Chisq")
temp8=anova(multinom(as.factor(y_train) ~ x_train[,-8]), glmFit, test="Chisq")
temp9=anova(multinom(as.factor(y_train) ~ x_train[,-9]), glmFit, test="Chisq")

myoutput = rbind(c(temp1$"   Df"[2], temp1$"LR stat."[2], temp1$"Pr(Chi)"[2]),
                 c(temp2$"   Df"[2], temp2$"LR stat."[2], temp2$"Pr(Chi)"[2]),
                 c(temp3$"   Df"[2], temp3$"LR stat."[2], temp3$"Pr(Chi)"[2]),
                 c(temp4$"   Df"[2], temp4$"LR stat."[2], temp4$"Pr(Chi)"[2]),
                 c(temp5$"   Df"[2], temp5$"LR stat."[2], temp5$"Pr(Chi)"[2]),
                 c(temp6$"   Df"[2], temp6$"LR stat."[2], temp6$"Pr(Chi)"[2]),
                 c(temp7$"   Df"[2], temp7$"LR stat."[2], temp7$"Pr(Chi)"[2]),
                 c(temp8$"   Df"[2], temp8$"LR stat."[2], temp8$"Pr(Chi)"[2]),
                 c(temp9$"   Df"[2], temp9$"LR stat."[2], temp9$"Pr(Chi)"[2]))
myoutput = matrix(myoutput, nrow=9, ncol=3, dimnames=list(
  c("Gender","Age","Handedness","ADHD.Index","Inattentive","Hyper.Impulsive", "Verbal.IQ", "Performance.IQ", "Full4.IQ")
       ,c("Df", "LR test statistics", "p-value")))
myoutput   # this shows df, LR test statistic, p-value for each variable


anova(multinom(as.factor(y_train) ~ 1), glmFit, test="Chisq")  # goodness-of-fit test: if p<0.05, our model is good. (
     see the second row)

# fitted(glmFit)  gives you three probabilities for each subject.  The subject is assigned a group with highest
     probability among three.
# Predict(glmFit) directly assigns a group for each subject.

yhat = predict(glmFit)
yhat

# also, predict function requires to use data.frame.
trainset <- data.frame(y_train, x_train)
testset <- data.frame(y_test, x_test)
glmFit = multinom(as.factor(y_train) ~ ., data=trainset)
predY = predict(glmFit, testset, type="class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, type="class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#######################################################
# multinom regression model using lasso penalty determined by aic
#######################################################
########AIC function
AIC <- function( y, x)
```

```
{
  mylambda = seq(0,.1, length = 100)
  aic = array(0, length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "multinomial", intercept=TRUE ,alpha = 1, lambda = lam, nlambda = 1)
    yhat = predict(glmFit, x, type = "class")

    aic[index] = (1-glmFit$dev.ratio)*glmFit$nulldev + 2*sum(glmFit$dfmat+1)

    index = index +1
  }
  lambda = mylambda[which.min(aic)]
  return(lambda)
}

#####multinom implementation with AIC
lam = AIC(y_train,x_train)
glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = 1, lambda = lam, nlambda = 1)
lam
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#########################################################
# multinomial regression model using lasso penalty determined by bic
#########################################################
BIC <- function(y,x,n)
{
  mylambda = seq(0,.1, length = 100)
  bic = array(0, length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "multinomial", intercept=TRUE ,alpha = 1, lambda = lam ,nlambda = 1)
    yhat = predict(glmFit, x, type = "class")

    bic[index] = (1-glmFit$dev.ratio)*glmFit$nulldev + log(n)*sum(glmFit$dfmat+1)
    index = index +1

  }
  lambda = mylambda[which.min(bic)]
  return(lambda)
}

lam = BIC(y_train,x_train, length(x_train))

glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = 1, lambda = lam, nlambda = 1)
lam
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

#########################################################
# multinomial regression model using lasso penalty determined by cv
#########################################################
CV<-function (x, y, K = 10, index)
{

  all.folds <- cv.folds(length(y), K)

  residmat <- matrix(0, length(index), K)

  for (i in seq(K)) {
    omit <- all.folds[[i]]

    glmFit <- glmnet(x[-omit, , drop = FALSE], y[-omit], family = "multinomial", intercept = TRUE, alpha = 1, lambda =
          index, nlambda = length(index))
    fit <- predict(glmFit,newx= x[omit, , drop = FALSE],type = "class")
    residmat[, i] <- (1=glmFit$dev.ratio)*glmFit$nulldev
  }
  cv <- apply(residmat, 1, mean)
  cv.error <- sqrt(apply(residmat, 1, var)/K)
  lam = index[which.min(cv)]
  return(lam)
```

```
}

lam = CV(x_train,y_train, 10, seq(from = 0,to = .1, length = 1000))
lam
glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = 1, lambda = lam, nlambda = 1)
lam
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#########################################################
# multinomial regression model using lasso penalty determined by GCV
#########################################################
GCV <- function(y, x, n)
{
  mylambda = seq(0,.1,length = 100)
  gcv = array(0, length(mylambda))
  index = 1
  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "multinomial", intercept=TRUE ,alpha = 1, lambda = lam, nlambda = 1)
    yhat = predict(glmFit,newx= x, type = "response")

    gcv[index] = (1-glmFit$dev.ratio)*glmFit$nulldev/(n*(1-(sum(glmFit$dfmat+1)/n))^2)
    index = index +1
  }
  lam = mylambda[which.min(gcv)]
  return(mylambda[which.min(gcv)])
}

lam = GCV(y_train,x_train, length(y_train))

glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = 1, lambda = lam, nlambda = 1)
lam
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#########################################################
# multinom regression model using RIDGE penalty determined by aic
#########################################################
########AIC function
AIC <- function( y, x)
{
  mylambda = seq(0,.1, length = 100)
  aic = array(0, length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "multinomial", intercept=TRUE ,alpha = 0, lambda = lam,nlambda = 1)
    yhat = predict(glmFit, x, type = "class")

    aic[index] = (1-glmFit$dev.ratio)*glmFit$nulldev + 2*sum(glmFit$dfmat+1)

    index = index +1
  }
  lambda = mylambda[which.min(aic)]
  return(lambda)
}
#####multinom implementation with AIC
lam = AIC(y_train,x_train)
glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = 0, lambda = lam, nlambda = 1)
lam
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#########################################################
# multinomial regression model using ridge penalty determined by bic
#########################################################
BIC <- function(y,x,n)
{
```

```
  mylambda = seq(0,.1, length = 100)
  bic = array(0, length(mylambda))
  index = 1

  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "multinomial", intercept=TRUE ,alpha = 0, lambda = lam,nlambda = 1)
    yhat = predict(glmFit, x, type = "class")

    bic[index] = (1-glmFit$dev.ratio)*glmFit$nulldev + log(n)*sum(glmFit$dfmat+1)
    index = index +1

  }
  lambda = mylambda[which.min(bic)]
  return(lambda)
}

lam = BIC(y_train,x_train, length(x_train))

glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = 0, lambda = lam, nlambda = 1)
lam
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

#########################################################
# multinomial regression model using ridge penalty determined by cv
#########################################################
CV<-function (x, y, K = 10, index)
{

  all.folds <- cv.folds(length(y), K)

  residmat <- matrix(0, length(index), K)

  for (i in seq(K)) {
    omit <- all.folds[[i]]

    glmFit <- glmnet(x[-omit, , drop = FALSE], y[-omit], family = "multinomial", intercept = TRUE, alpha = 1, lambda =
          index, nlambda = length(index))
    fit <- predict(glmFit,newx= x[omit, , drop = FALSE],type = "class")
    residmat[, i] <- mean((y[omit] - as.numeric(fit))^2)
  }
  cv <- apply(residmat, 1, mean)
  cv.error <- sqrt(apply(residmat, 1, var)/K)
  lam = index[which.min(cv)]
  return(lam)

}

lam = CV(x_train,y_train, 10, seq(from = 0,to = .1, length = 1000))
lam
glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = 1, lambda = lam, nlambda = 1)
lam
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#########################################################
# logistic regression model using RIDGE penalty determined by GCV
#########################################################
GCV <- function(y, x, n)
{
  mylambda = seq(0,.1,length = 100)
  gcv = array(0, length(mylambda))
  index = 1
  for(lam in mylambda)
  {
    glmFit = glmnet(x,y, family = "multinomial", intercept=TRUE ,alpha = 0, lambda = lam, nlambda = 1)
    yhat = predict(glmFit,newx= x, type = "response")

    gcv[index] = (1-glmFit$dev.ratio)*glmFit$nulldev/(n*(1-(sum(glmFit$dfmat+1)/n))^2)
    index = index +1
  }
  lam = mylambda[which.min(gcv)]
  return(mylambda[which.min(gcv)])
}
```

```
lam = GCV(y_train,x_train, length(y_train))

glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = 0, lambda = lam, nlambda = 1)
lam
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

#########################################################
# multinom regression model using elastic net penalty determined by aic
#########################################################
########AIC function
AIC <- function(y, x)
{
  mylambda = seq(0,.1, length = 100)
  myalpha = seq(0,1,length = 100)
  aic = matrix(0, length(mylambda)*length(myalpha),3)
  index1 = 1

  for(a in myalpha)
  {
    for(lam in mylambda)
    {
      glmFit = glmnet(x,y, family = "multinomial", intercept=T ,alpha = a, lambda = lam,nlambda = 1)
      yhat = predict(glmFit, x, type = "response")

      aic[index1,] = c((1-glmFit$dev.ratio)*glmFit$nulldev + 2*(sum(glmFit$dfmat+1)),lam,a)
      index1 = index1 +1
    }
  }
  min = which.min(aic[,1])
  list(alpha = aic[min,3], lambda = aic[min,2])

}

#####multinom implementation with AIC
lam = AIC(y_train,x_train)
glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = lam$alpha, lambda = lam$lambda
     , nlambda = 1)
lam$alpha
lam$lambda
glmFit$beta

predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#########################################################
# multinomial regression model using ENET penalty determined by bic
#########################################################
BIC <- function(y,x,n)
{
  mylambda = seq(0,.1, length = 100)
  myalpha = seq(0,1,length = 100)
  bic = matrix(0, length(mylambda)*length(myalpha),3)
  index1 = 1
  index2 = 1

  for(lam in mylambda)
  {
    for(a in myalpha)
    {
      glmFit = glmnet(x,y, family = "multinomial", intercept=T ,alpha = a, lambda = lam ,nlambda = 1)
      yhat = predict(glmFit, x, type = "response")

      bic[index1,] = c((1-glmFit$dev.ratio)*glmFit$nulldev + log(n)*(sum(glmFit$dfmat+1)),lam,a)
      index1 = index1 +1
    }
  }
  min = which.min(bic[,1])
  list(alpha = bic[min,3], lambda = bic[min,2])
}


lam = BIC(y_train,x_train, length(x_train))

glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = lam$alpha, lambda = lam$lambda
     , nlambda = 1)
lam$alpha
```

```
lam$lambda
glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification

################################################################
# multinomial regression model using ENET penalty determined by cv
################################################################
CV<-function( x, y, K = 10, index, alph)
{
  tuningparam = matrix(0,length(index), 2)
  all.folds <- cv.folds(length(y), K)

  residmat <- matrix(0, length(index), K)
  for (i in seq(K)) {
    omit <- all.folds[[i]]
    glmFit <- glmnet(x[-omit, , drop = FALSE], y[-omit], intercept = T, alpha = alph, lambda = index, nlambda = length(
          index))
    fit <- predict(glmFit, x[omit, , drop = FALSE], s = .05,type = "response")

    residmat[, i] <- (1-glmFit$dev.ratio)*glmFit$nulldev
  }
  cv <- apply(residmat, 1, mean)
  cv.error <- sqrt(apply(residmat, 1, var)/K)
  lam = index[which.min(cv)]
  tuningparam = c(lam, min(cv))
}
########################################glmnet has a CV function that will be used here
alph = seq(from = 0, to = 1, length = 100)
tuningparam = matrix(0, length(alph),2)
j = 1
for( a in alph)
{
  param = cv.glmnet(x = x_train, y = y_train, family = "multinomial", intercept = T, alpha = a)
  tuningparam[j,] = c(param$lambda[which.min(param$cvm)], min(param$cvm))
  j = j+1
}
tuningparam
lam = tuningparam[which.min(tuningparam[,2])]
al = alph[which.min(tuningparam[,2])]
al
lam

glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = al, lambda = lam, nlambda = 1)

glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
#####################################################
# logistic regression model using ENET penalty determined by GCV
#####################################################
GCV <- function(y, x, n)
{
  mylambda = seq(0,.1, length = 100)
  myalpha = seq(0,1,length = 100)
  gcv = matrix(0, length(mylambda)*length(myalpha),3)
  index1 = 1
  index2 = 1

  for(a in myalpha)
  {
    for(lam in mylambda)
    {
      glmFit = glmnet(x,y, family = "multinomial", intercept = T, alpha = a, lambda = lam,nlambda = 1)
      yhat = predict(glmFit, x, type = "response")

      gcv[index1,] = c((1-glmFit$dev.ratio)*glmFit$nulldev/(n*(1-(sum(glmFit$dfmat+1)/n))^2),lam,a)
      index1 = index1 +1
    }
  }
  min = which.min(gcv[,1])
  list(alpha = gcv[min,3], lambda = gcv[min,2])
}

lam = GCV(y_train,x_train, length(y_train))

glmFit = glmnet(x = x_train, y = y_train, family = "multinomial", intercept=TRUE ,alpha = lam$alpha, lambda = lam$lambda
```

```
     , nlambda = 1)
lam$alpha
lam$lambda

glmFit$beta
predY= predict(glmFit,newx = x_test, s = 0, type = "class")
mean(predY != y_test) # classification error
table(predY, y_test)  # table for classification

# for training set
predY = predict(glmFit, newx=x_train, s = 0, type = "class")
mean(predY != y_train) # classification error
table(predY, y_train)  # table for classification
```