Contents

1 Introduction

A large part of statistics and machine learning deals with prediction. Predicting sales during the holidays, for example, helps determine the right amount of inventory. Recent research has shown that so-called black-box models, like deep learning, outperforms classical statistical methods substantially. The downside is these models need a lot of data, don't work well in a high dimensional setting, and are not easily interpreted. Classical statistical methods, like the Linear Regression (LR), are easily interpreted and used with little data but are prone to over-fit and don't work well in a high dimensional setting.

One way to bridge the gap between the problems is via regularization or shrinkage methods. Regularization reduces the variance of the estimator by increasing the bias. By applying shrinkage methods to the LR model we keep the interpretability and enhance the prediction quality. Furthermore, a regularized LR model works well in a high dimensional setting. Arguably the most important regularization methods is the Elastic-Net (EN) method introduced by H. Zhou and Hastie (2005).

2 Theoretical Elastic-Net

We discuss the motivation behind developing Elastic-Net by explaining the drawbacks of lasso and ridge and how Elastic-Net overcomes these.

2.1 Model description.

The model is defined as follows. Let y_i be the dependent variable, $x_{i,j}$ the independent variable, and β_j the parameter for $i = 1, 2, ..., n$ and $j = 1, 2, ..., p$. Then the LR model is defined as,

$$Y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}.$$

The general way to estimate the parameter vector β in the LR model with shrinkage is by finding:

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left(\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + f_{\text{pen}}(\beta, \lambda) \right),$$

where $\lambda \in [0, \infty)$ is a regularization parameter and $f_{\text{pen}}(\beta, \lambda)$ a penalty function. The two most commonly chosen penalties functions result in the ridge and lasso estimators and are defined as,

$$\hat{\beta}_{\text{ridge}} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left(\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right),$$

$$\hat{\beta}_{\text{lasso}} \in \underset{\beta \in \mathbb{R}^p}{\arg\min} \left(\sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right).$$

Ridge has the benefit that it keeps the strict convexity in β which gives unique minimum while simultaneously shrinking correlated β_js. Whereas, the lasso is only convex which means that there are multiple solutions for β, but the predictor itself, $X\hat{\beta}$, is unique. Furthermore, the lasso also has the ability to set β_js equal to zero, i.e., performs variable selection but only up to n features which doesn't work well in a high dimensional setting, e.g., when $p \gg n$. Additionally, lasso keeps one of the grouped

variables while ignoring the others.

Zou and Hastie (2005), therefore, proposed a the Elastic-Net penalty. The Elastic-Net penalty is a combination of the ridge and lasso penalties. By combining these, the Elastic-Net penalty removes limitation on the number of selected variables, stabilizes the lasso regularization path, shrinks correlated variables, and does variable selection. The Elastic-Net penalty is defined as

$$f_{\text{EN}}(\beta, \lambda) = \lambda \sum_{j=1}^{p} \left(\alpha\beta_j^2 + (1-\alpha)|\beta_j|\right)$$

for $\alpha \in (0, 1)$. From the penalty function we can directly see that when $p \gg n$, α will be closer to zero such that more variables are set to zero.

2.2 Deriving Elastic-Net

Now we derive the Elastic-Net using the naive Elastic-Net.

2.2.1 Naive Elastic-Net (NEN)

We will briefly introduce NEN. Assume that the data is standardized and the we have n observations and p independent variables. Then the naive Elastic-Netcriterion is defined as,

$$L(\lambda_1, \lambda_2, \beta) = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda_2 \sum_{j=1}^{p} \beta_j^2 + \lambda_1 \sum_{j=1}^{p} |\beta_j|.$$

The naive estimator for β minimized the criterion function s.t.,

$$\hat{\beta}_{\text{EN}} = \underset{\beta \in \mathbb{R}^p}{\arg\min}\, L(\lambda_1, \lambda_2, \beta)$$

By setting $\alpha = \lambda_2/(\lambda_1 + \lambda_2)$ we solve the constrained optimization problem,

$$\begin{aligned} \hat{\beta}_{\text{EN}} =& \underset{\beta \in \mathbb{R}^p}{\arg\min} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \\ \text{s.t.} \quad & (1-\alpha) \sum_{j=1}^{p} |\beta_j| + \alpha \sum_{j=1}^{p} \beta_j^2 \leq t \quad \text{for some } t \in \mathbb{R}. \end{aligned}$$

However, empirical research shows that the NEN performs well when α is either near zero or one, e.g., it's close to ridge or lasso.(H. Zhou & Hastie, 2005) The theoretical reasons for this are, that we first find the ridge regression coefficients and then perform a lasso-type shrinkage following the lasso coefficient solution paths. The double shrinkage does not decrease the variance a lot but introduces unnecessary bias compared the standalone penalties.

2.2.2 Elastic-Net Estimator

For ease of notation we switch to matrix notation to derive the Elastic-Net estimator we switch to matrix notation. The NEN criterion can be written as

$$\hat{\beta}^* = \underset{\beta \in \mathbb{R}^p}{\arg\min} \|y^* - X^*\beta^*\|^2 + \gamma\|\beta^*\|,$$

where $\|\cdot\|$ is a p-norm, $\gamma = \lambda_1/\sqrt{1+\lambda_2}$, and the triple (y^*, X^*, β^*) defined as

$$y^*_{((n+p)\times 1)} = \begin{pmatrix} y_{(n\times 1)} \\ 0_{(p\times 1)} \end{pmatrix}, X^*_{((n+p)\times p)} = \frac{1}{\sqrt{1+\lambda_2}} \begin{pmatrix} X_{(n\times p)} \\ \sqrt{\lambda_2} I_{(p\times p)} \end{pmatrix}, \text{ and } \beta^* = \sqrt{1+\lambda_2}\beta.$$

Then the Elastic-Netestimator becomes

$$\hat{\beta}_{\text{EN}} = \sqrt{1+\lambda_2}\beta^*,$$

since the NEN is defined as

$$\hat{\beta}_{\text{NEN}} = \frac{1}{\sqrt{1+\lambda_2}}\hat{\beta}^*.$$

Thus the Elastic-Netestimator is the rescaled NEN estimator, i.e.,

$$\hat{\beta}_{\text{EN}} = (1+\lambda_2)\hat{\beta}_{\text{NEN}}.$$

After rescaling NEN coefficient, it retains the variable selection property and removes shrinkage. The more refined Elastic-Net estimate achieves a good bias-variance trade-off. Re-scaling allows elastic-net to become a minimax optimal which works well with orthogonal predictors.

2.3 Cartesian Plot

To demonstrate how Elastic-Net penalty is better than lasso and ridge, we plot the penalties on a Cartesian Plane. From Figure 1 shows that the contour for ridge is smooth and, therefore, strict convex. From the lasso contour we can see that when a solution occurs in the corner then one β_j is zero. As dimensions increase it is more likely that a solution occurs in one of the vertices. The Elastic-Net has the sharp vertices and smooth edges which made it able to shrink correlated covariates and set parameters equal to zero.

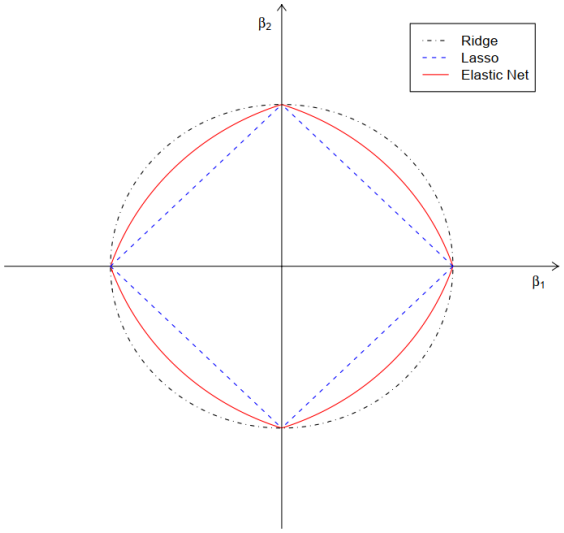


Figure 1: Two-dimensional contour plots for the constraint regions in the optimization problem for the ridge, lasso, and Elastic-Net. For the Elastic-Net $\alpha = 0.5$.(H. Zhou & Hastie, 2005)

3 Tuning Parameters

The Elastic-Net regression has two tuning parameters, namely, λ and α. The former is used for shrinkage and latter for the compromise between the lasso and ridge penalties. Both tuning parameters are

retrieved via cross-validation (CV). DeWitt and Bennett (2019) developed a package `ensr` to determine the tuning parameters simultaneously through CV which allows to compute the optimal λ–α space unlike `glmnet`.

In CV we estimate the model multiple times for a certain range of values for the tuning parameters. Normally tenfold cross-validation is performed on the training set, which splits the training set into 10 equal parts (folds). Then for each fold $k = 1, ..., 10$, we train the model on all data except the kth fold. The kth fold is used as a sort internal test set for which calculate the Mean Squared Error (MSE). At last, we choose the tuning parameters of the model which has the lowest average MSE (average over k internal test).

4 Simulation Example

To show the differences between the models we sample $X_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(0, 3)$ for $i = 1, 2, ..., 125$ and $j = 1, 2, ..., p/2$. Then we create an additional $p/2$ variables by creating a random linear combination of the independent variables and add some noise in the form of a standard normal error. The true β is created by randomly choosing $p/2$ integers between -5 and 5 and the rest are set to zero. The tuning parameters are found via 10-fold CV.

j	β_j	$\hat{\beta}_j^{\text{OLS}}$	$\hat{\beta}_j^{\text{Ridge}}$	$\hat{\beta}_j^{\text{lasso}}$	$\hat{\beta}_j^{\text{EN}}$
1	0.0	0.105	0.106	0.047	0.046
2	−2.0	−2.026	−2.026	−2.005	−2.005
3	2.0	1.990	1.990	1.971	1.971
4	2.0	2.054	2.052	2.005	2.003
5	−4.0	−3.961	−3.948	−3.893	−3.854
6	0.0	−0.043	−0.044	0.0	0.0
7	0.0	0.062	0.057	0.0	0.0
8	0.0	−0.034	−0.032	0.0	0.0
9	0.0	−0.008	−0.012	0.0	−0.013
10	0.0	−0.089	−0.093	−0.088	−0.111

Table 1: Coefficient estimators of the Linear Regression model with no, ridge, lasso, and Elastic-Net penalty.

Table 1 shows the coefficients estimated for the four models with $p = 10$. Here we can see that the lasso and Elastic-Net penalty both force some β_js to be zero to create a sparse model.

p	10	20	50	100
OLS	1.044	1.107	1.419	54.513
Ridge	1.044	1.107	1.415	4.685
lasso	1.028	1.080	1.271	2.287
Elastic-Net	1.033	1.087	1.299	2.693

Table 2: The average Root Mean Squared Error on the test set for 1000 runs for different values of p and the four models.

The data is split into a 80/20 training and test data set. We then run this simulation thousand times and calculate the average RMSE based on the test set for $p = 10, 20, 50, 100$. Table 2 shows the simulation results. Once p increases we see that the performance of the OLS and ridge significantly decrease while the lasso and Elastic-Net are relatively the same.

5 Computing Elastic-Net in R

We will implement OLS, Elastic-Net, lasso and ridge to check their performance.

Regularization methods for regression can easily be computed with the R-packages `caret` and `glmnet`. The former streamlines the model training process, helps with data processing, feature selection, sampling, and tuning parameter selection. The latter is used for fitting generalized linear models via penalized maximum likelihood.

The data we used is Boston in `MASS` package, which contains 506 housing data in Boston.(Harrison & Rubenfield, 1978) We choose the median value of houses in thousands dollar as the dependent variable and include 13 features.

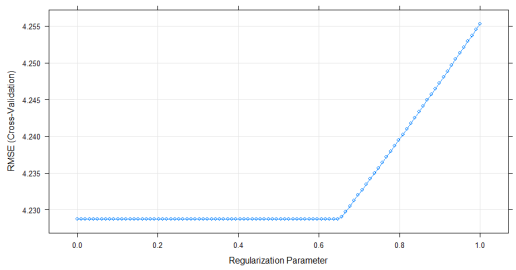
Firstly, we do a 30/70 training-test split, then we apply OLS, ridge, lasso, and Elastic-Net to build the model and prediction. Before building each penalized model, we use 10-fold CV to choose optimal tuning parameters.

The package `glmnet` estimates β through the following formula:

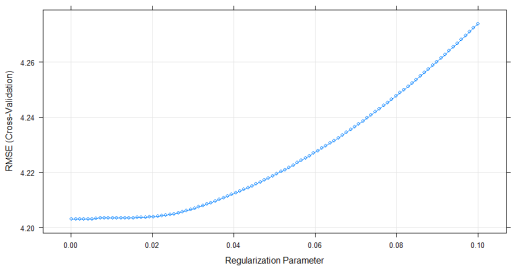
$$\hat{\beta} = \underset{\beta \in \mathbb{R}^p}{\arg\min} \left(\frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 + \lambda \left(\alpha \sum_{j=1}^{p} |\beta_j| + \frac{1-\alpha}{2} \sum_{j=1}^{p} \beta_j^2 \right) \right)$$

There are two tuning parameters λ and α. For ridge regression, α is set to be 0 and for lasso α is set to be 1, and we only use CV for λ. In terms of Elastic-Net, we select both λ and α.

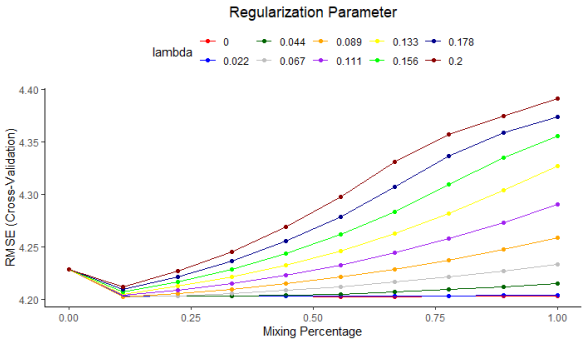
Figure 2 shows the CV RMSE of the three models. The model with the lowest CV RMSE is the best one. We choose $\lambda = 0.646$, $\lambda = 0.00515$ and $(\lambda = 0.0444, \alpha = 0.111)$ as the tuning parameters of ridge, lasso and Elastic-Net respectively.



(a) Root MSE of regularization with ridge penalty.



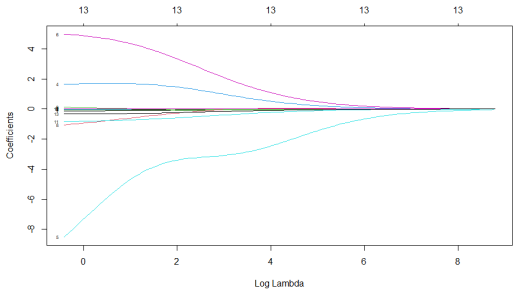
(b) Root MSE of regularization with lasso penalty.



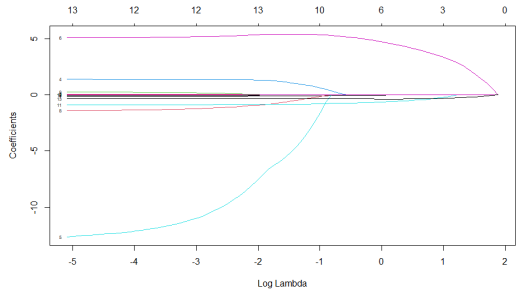
(c) Root MSE of regularization and mixing parameter with Elastic-Net penalty.

Figure 2: Cross-Validation results based on the Root Means Squared Error (RMSE) for the ridge, lasso, and Elastic-Net penalties.

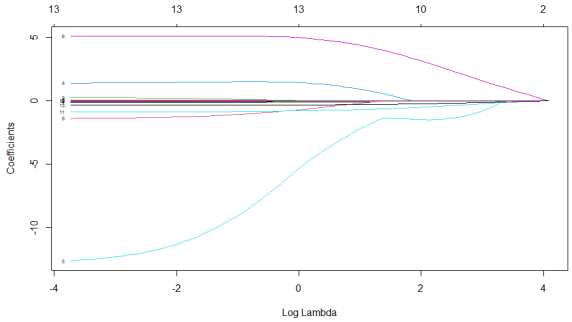
Figure 3 shows the regularization path of the coefficients. As $\log(\lambda)$ increases, all coefficients will shrink towards zero. For ridge regression, the coefficients approach to 0 with the increase in $\log(\lambda)$, known as shrinkage. However, for lasso, if we select a large λ, some coefficients will become 0 known as model selection. When applying Elastic-Net, some coefficients will become 0 with the increase of λ but slowly.



(a) Regularization path for the ridge penalty.



(b) Regularization path for the lasso penalty.



(c) Regularization path for the Elastic-Net penalty.

Figure 3: Coefficient regularization path for the range of the log of λ for the models with the ridge, lasso, and Elastic-Net penalties.

After training, we test the models with best tuning parameters using test data set. We apply Boostrap to resample in both training data and test data 100 times respectively and calculate the average training and test RMSE for 4 models shown in Table 3. Because the number of features is not large enough, the performance of 4 models are quite similar and OLS performs best. However, if we only focus on the regularization models, test RMSE shows that Elastic-Net performs better than other two.

Model	Mean Training RMSE	Mean Test RMSE
OLS	4.10818	6.14500
Ridge	4.16114	6.19819
lasso	4.10859	6.14813
Elastic-Net	4.10955	6.14808

Table 3: The mean of Training and Test RMSE

6 Applications

6.1 Healthcare

Johnsen and Gao (2020) presented an Elastic-Net COVID-19 Forecaster (EN-CoF) which is a multilinear regressor trained on time series data for COVID-19 cases forecasting. CV is used for model

hyperparameters comparison. EN-CoF uses Elastic-Net and seventeen 5-day averages to forecast COVID-19 cases. It is an important tool for logisticians and the error metrics represent how accurate EN-CoF performs by various categories such as country, time-period, daily trend etc. The results prove that EN-CoF is a more generic model compared to other sophisticated models and also works well with any region.

6.2 Finance

Yi (2017) conducted research on probability of loan default using elastic-net as a variable extraction methodology. It enhances the predictive ability of the loan default prediction model due to smaller MSE. With the L_1-norm penalty, the Elastic-Net automatically chooses the relevant variables and makes the unimportant variables' coefficients zero. The L_2-norm penalty "selects the subset of correlated potential variables in the credit risk analysis framework". Furthermore, elastic-net is unaffected by multicollinearity or heteroscedasticity; therefore, it produces parsimonious models, which predicts loan defaults with lower number of predictor variables. These predictor variables differentiate the defaulting firms from the non-defaulting firms.

7 Major developments

Elastic-Net has been developed in the last decade to make it more applicable and efficient. It can be used for some nonlinear models, like Generalize Linear Models (GLMs). Friedman et al. (2010) proposed an efficient approach for constructing the Elastic-Net regularization path for OLS and logistic regression and it extended to Cox models by Simon et al. (2011). Recently, all GLMs have been included in the elastic-net regularized regression's applicability, suggested by Tay et al. (2021). All these papers use cyclical coordinate descent to estimate the parameters. We briefly introduce how to add penalty to GLMs and the principle of coordinate descent.

Take binary logistic regression as an example. Let $y_i = \{0, 1\}$ be the binary dependent variable and x_i a vector with p variables for $i = 1, 2, ..., n$. Then we can write the logistic regression as,

$$\mathbb{P}(Y_i = y_i \mid X_i = x_i') = p_i^{y_i}(1 - p_i)^{1-y_i}$$

where p_i is defined as

$$p_i = \frac{1}{1 + \exp\left(-\beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)}.$$

The log-likelihood function then becomes

$$\begin{aligned}\mathcal{L}(\beta; X) &= \ln\left(\prod_{i=1}^{n} p_i^{y_i}(1 - p_i)^{1-y_i}\right) \\ &= \sum_{i=1}^{n} y_i \ln(p_i) + (1 - y_i)\ln(1 - p_i).\end{aligned}$$

Including the Elastic-Net penalty we get the following penalized log-likelihood,

$$\mathcal{L}_{\text{EN}}(\beta; X) = \mathcal{L}(\beta; X) + \lambda\left(\alpha \sum_{j=1}^{p} |\beta_j| + \frac{1 - \alpha}{2} \sum_{j=1}^{p} \beta_j^2\right).$$

This function can be minimized by solving using cyclical coordinate descent to find β:

$$\hat{\beta}_{\text{EN}} \in \arg\min_{\beta \in \mathbb{R}^p} \mathcal{L}_{\text{EN}}(\beta; X)$$

To reach a function's minimum value, it sequentially minimizes along coordinate axes. It begins with the original value we set. In each iteration, it chooses a coordinate of one direction by setting the partial derivative of that direction to 0, keeping coordinate of other directions fixed.

This example shows how it works and finds the β that can minimize $f(\beta)$.

1. Start with $\beta^{(0)} = (\beta_1^0, ..., \beta_p^0)$
2. Round $k+1$ defines $\beta^{(k+1)}$ from $\beta^{(k)}$ by iteratively solving the single variable optimization problem.
$$\beta_i^{(k+1)} = \arg\min_{\theta \in \mathbb{R}} f(\beta_1^{(k+1)}, ..., \beta_{i-1}^{(k+1)}, \theta, \beta_{i+1}^{(k)}, ..., \beta_p^{(k)})$$
3. Repeat for each variable β_i in β for $i = 1, 2, ..., p$.

Cyclical coordinate descent more efficient than LARS-EN. Statisticians keep improving its efficiency and applicability.

8 Conclusion

To conclude regularization is nowadays one of the most popular approach in modelling and especially prediction. The Elastic-Net penalty is one of the most widely used penalties due to the simultaneously shrinking and doing feature selection simultaneously. It is preferred over lasso and ridge due to sparsity and better prediction accuracy while performing grouping effect. The results and simulations conducted prove that it is obtains the least test RMSE than lasso and ridge. Furthermore, from finance to healthcare, Elastic-Net has been widely applied in all sectors which demonstrates its importance. In the last decade, statisticians have introduced major developments for the Elastic-Net such as implementing it for non-linear models and finding a more efficient way of solving it known as cyclical coordinate descent; however, there is a long way to go since the developments are still in process.

References

DeWitt, P. E., & Bennett, T. D. (2019). Ensr: R package for simultaneous selection of elastic net tuning parameters. https://arxiv.org/pdf/1907.00914.pdf

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1–22.

Harrison, D., & Rubenfield, D. L. (1978). Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, *5*(1), 81–102.

Hastie, T., Tibshirani, R., & Friedman, J. (2011). *The elements of statistical learning* (2nd ed.). Springer.

Johnsen, T. K., & Gao, J. Z. (2020). Elastic net to forecast covid-19 cases. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9311968

Simon, N., Friedman, J., Hastie, T., & Tibshirani, R. (2011). Regularization paths for cox's proportional hazards model via coordinate descent. *Journal of Statistical Software*, *39*(5), 1–13.

Tay, J. K., Narasimhan, B., & Hastie, T. (2021). Elastic net regularization paths for all generalized linear models. https://https://arxiv.org/pdf/2103.03475.pdf

Yi, J. (2017). Prediction of loan default using elastic net. https://www.unsw.edu.au/business/sites/default/files/seminarsconferences/J_Yi_Prediction_of_Loan_Default_Using_Elastic_Net.pdf

Zhou, D.-X. (2013). On grouping effect of elastic net. https://www.researchgate.net/publication/267465100_On_grouping_effect_of_elastic_net

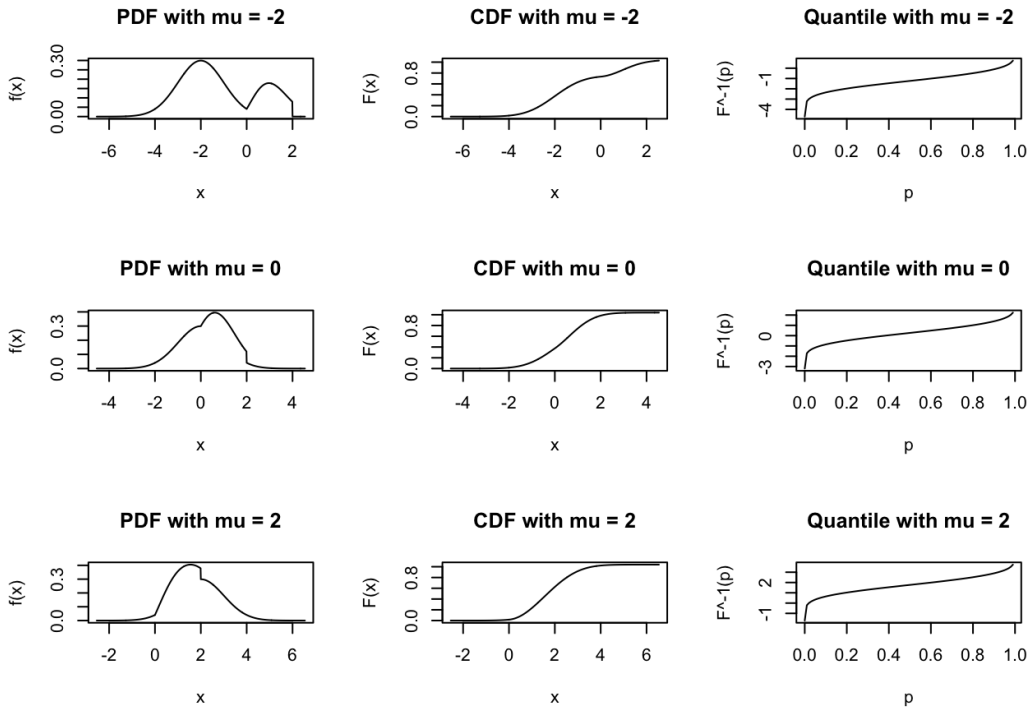
Zhou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society*, *67*(12), 301–320.

A Mixture Distribution

```
################################################################################
# Package import
library(stats4)
library(latex2exp)
library(bbmle)

################################################################################
# Initialize weights for the mixture distribution
w = c(0.25, 0.75)
# Initialize sample size
n = 10000
# Initialize error tolerance for capturing all of the data
tol = 10^(-5)
# For quantile function
p = seq(0+tol, 1-tol, by=0.01)

################################################################################
# Part1: Mixture PDF, CDF, Quantile function, and sampling methods
################################################################################

# Component 1 of the mixture distribution*
# Write PDF, CDF, Quantile function, and sampling methods for the 1st component.
func_pdf <- function(x){
  return(x * exp(-0.5*x^2) / (1 - exp(-2)))
}

func_cdf <- function(x){
  return((1 - exp(-0.5 * x^2)) / (1 - exp(-2)))
}

func_q <- function(p){
  return(sqrt(-2 * log(1 - (1 - exp(-2)) * p)))
}

func_rvs <- function(n){
  # Drawing random variable via inverse transformation technique
  return(func_q(runif(n, min=0, max=1)))
}

################################################################################
# Component 2 of the mixture distribution
# Write PDF, CDF, Quantile function, and sampling methods for the 2nd component.
norm_pdf <- function(x, mu){
  return(dnorm(x, mean=mu, sd=1))
}

norm_cdf <- function(x, mu){
  return(pnorm(x, mean=mu, sd=1))
}

norm_rvs <- function(n, mu){
  return(rnorm(n, mean=mu, sd=1))
}
```

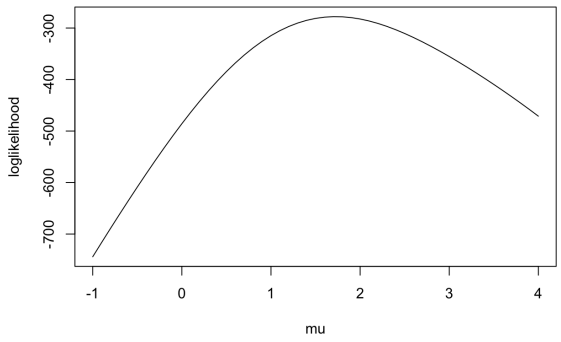
```
################################################################################
# Combine components to get PDF, CDF, Quantile function, and sampling methods
# for the mixture distribution.
mix_pdf <- function(x, mu, w){
  idx = 1*c(0<x & x<2)
  c1_pdf = func_pdf(x)
  c1_pdf = replace(c1_pdf, idx==0, 0)
  pdf = w[1]*c1_pdf + w[2]*norm_pdf(x, mu)
  return(pdf)
}

mix_cdf <- function(x, mu, w){
  idx = 1*c(0<x)
  c1_cdf = func_cdf(x)
  c1_cdf = replace(c1_cdf, idx==0, 0)
  cdf = w[1]*c1_cdf + w[2]*norm_cdf(x, mu)
  return(cdf)
}


mix_q <- function(p, mu, w){
  G = function(x) mix_cdf(x, mu, w) - p
  q = uniroot(G, interval=c(-1000, 1000))
  return(q$root)
}

mix_rvs <- function(n, mu, w){
  component = t(rmultinom(n, 1, w))
  rvs = component[, 1]*func_rvs(n) + component[, 2]*norm_rvs(n, mu)
  return(rvs)
}
```

```
################################################################################
# Part 2: Run plotting part of the assignment
################################################################################

# Initialize Variable for mu
mu = c(-2, 0, 2)

# write function to plot pdf, cdf and quantile function
plot_pdf <- function(x, mu, w){
  y = mix_pdf(x, mu, w)
  plot(x=x, y=y, main = paste0("PDF with mu = ", mu), type='l', xlab="x", ylab="f(x)")
}

plot_cdf <- function(x, mu, w){
  y = mix_cdf(x, mu, w)
  plot(x=x, y=y, main = paste0("CDF with mu = ", mu), type='l', xlab="x", ylab="F(x)")
}

plot_q <- function(p, mu, w){
  y = rep(0, length(p))
  for (i in 1:length(p)){
    y[i] = mix_q(p[i], mu, w)
  }
  plot(x=p, y=y, main = paste0("Quantile with mu = ", mu),
  type='l', xlab="p", ylab="F^-1(p)")
}

# Find coverage for the plots
find_coverage <- function(mu, w, tol){
  x = seq(-1000, 1000, by=0.01)
  y = mix_pdf(x, mu, w)
  idx = y>tol
  x = x[idx]
  return(x)
}

#use for loop to plot graphs for each value of mu
plot_graphs <- function(mu, w, p, tol){
  for(i in 1:length(mu)){
    x = find_coverage(mu[i], w, tol)
    plot_pdf(x, mu[i], w)
    Sys.sleep(0)
    plot_cdf(x, mu[i], w)
    Sys.sleep(0)
    plot_q(p, mu[i], w)
    Sys.sleep(0)
  }
}

# plot the graphs
par(mfrow=c(3,3))
plot_graphs(mu, w, p, tol)
```


PDF with mu = -2
f(x)
0.00
0.30
-6
-4
-2
0
2
x
CDF with mu = -2
F(x)
0.0
0.8
-6
-4
-2
0
2
x
Quantile with mu = -2
F^-1(p)
-4
-1
0.0
0.2
0.4
0.6
0.8
1.0
p
PDF with mu = 0
f(x)
0.0
0.3
-4
-2
0
2
4
x
CDF with mu = 0
F(x)
0.0
0.8
-4
-2
0
2
4
x
Quantile with mu = 0
F^-1(p)
-3
0
0.0
0.2
0.4
0.6
0.8
1.0
p
PDF with mu = 2
f(x)
0.0
0.3
-2
0
2
4
6
x
CDF with mu = 2
F(x)
0.0
0.8
-2
0
2
4
6
x
Quantile with mu = 2
F^-1(p)
-1
2
0.0
0.2
0.4
0.6
0.8
1.0
p

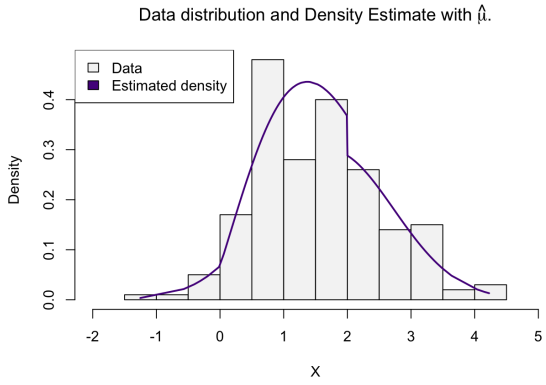
```r
################################################################################
# Part 3: Monte Carlo simulation
################################################################################

# Initialize Variable for mu
mu = -1

#function for calculate the expectation
calc_mean <- function(X){
  return(mean((X[,1] - X[,2]) / (1 + abs(X[,3]))))
}

#function for calculate standard error
calc_se <- function(X, n){
  mean = calc_mean(X)
  se = sqrt(sum(((X[,1] - X[,2]) / (1 + abs(X[,3])) - mean)^2)) / sqrt(n*(n-1))
  return(se)
}

# use for loop to obtain samples from three i.i.d distribution
run_sim <- function(mu, n, w){
  X = matrix(, ncol=3, nrow=n)
  for (i in 1:3){
    X[, i] = mix_rvs(n, mu, w)
  }
  result_list = list("mean" = calc_mean(X), "se" = calc_se(X,n))
  return(result_list)
}

# calculate the estimate for the required expectation and the standard error
result = run_sim(mu, n, w)
print(paste0('Estimate for mu:', round(result$mean, digits=3)))
print(paste0('with standard error:', round(result$se, digits=3)))
```

```r
################################################################################
# Part 4: Estimate mu
################################################################################


# read the file
file = 'mData2022.txt'
x = read.table(file)[[1]]

# Function for the negative loglikelihood
calc_nll <- function(mu, x, w){
  idx = 1*c(0<x & x<2)
  c1_pdf = func_pdf(x)
  c1_pdf = replace(c1_pdf, idx==0, 0)
  pdf = w[1]*c1_pdf + w[2]*norm_pdf(x, mu)
  return(-1*sum(log(pdf)))
}

# Plot the likelihood function
mu_seq = seq(as.integer(min(x)), as.integer(max(x)), by=0.1)
nll = c()

for(i in 1:length(mu_seq)){
  nll[i] = calc_nll(mu_seq[i], x, w)
}
par(mfrow=c(1,1))
plot(x=mu_seq, y=-nll, type='l', xlab='mu', ylab='loglikelihood')
```



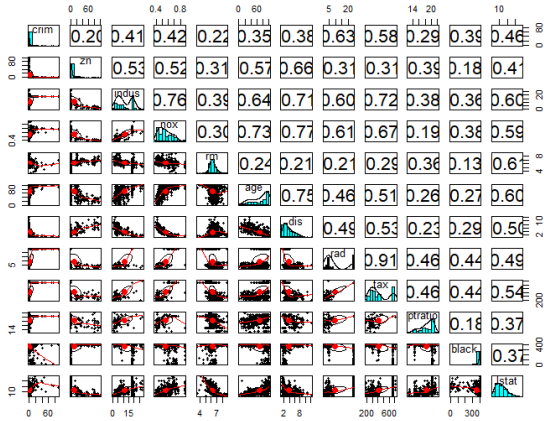

```r
# Estimate mu via the MLE method
mu0 = mean(x)
optim_res = optim(mu0, calc_nll, x=x, w=w, method='BFGS')
mu_hat = optim_res$par

# Therefore, the estimate for mu (mu_hat) is:
print(paste0('Estimate for mu: ', round(mu_hat, digits=3)))
```

```r
# Plot the data and the estimated density line
hist(x, breaks=10, freq=FALSE,
        main=TeX(r"(Data distribution and Density Estimate with \hat{\mu}.)"),
        xlab='X',
        ylab='Density',
     xlim=c(as.integer(floor(min(x))), as.integer(ceiling(max(x)))), c='grey96')
lines(x=sort(x), y=mix_pdf(sort(x), mu_hat, w), col='purple4', lwd=2)
legend(x='topleft', legend=c('Data', 'Estimated density'), fill=c('grey96', 'purple4'))
```



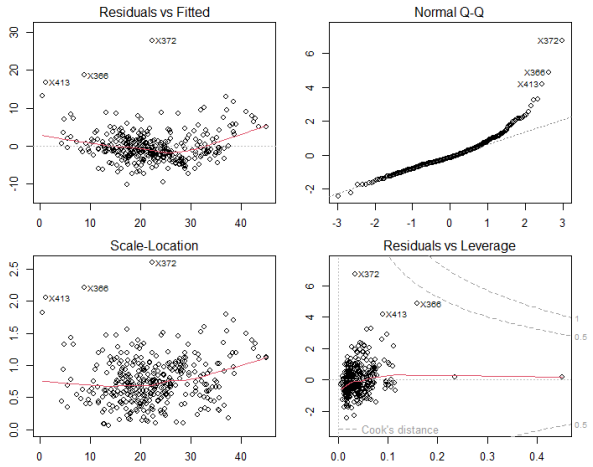
B R code of Part 1

```
#import required libraries
rm(list=ls())
library(caret)
library(glmnet)
library(MASS)
library(mlbench)
library(psych)


# loading the database and checking the structure
data <- Boston
# The structure and some graphical summaries of the data
head(data)
dim(data)
str(data)
pairs.panels(data[c(-4,-14)],cex=2)  #column 4 is a dummy variable and 14 is the response.
```

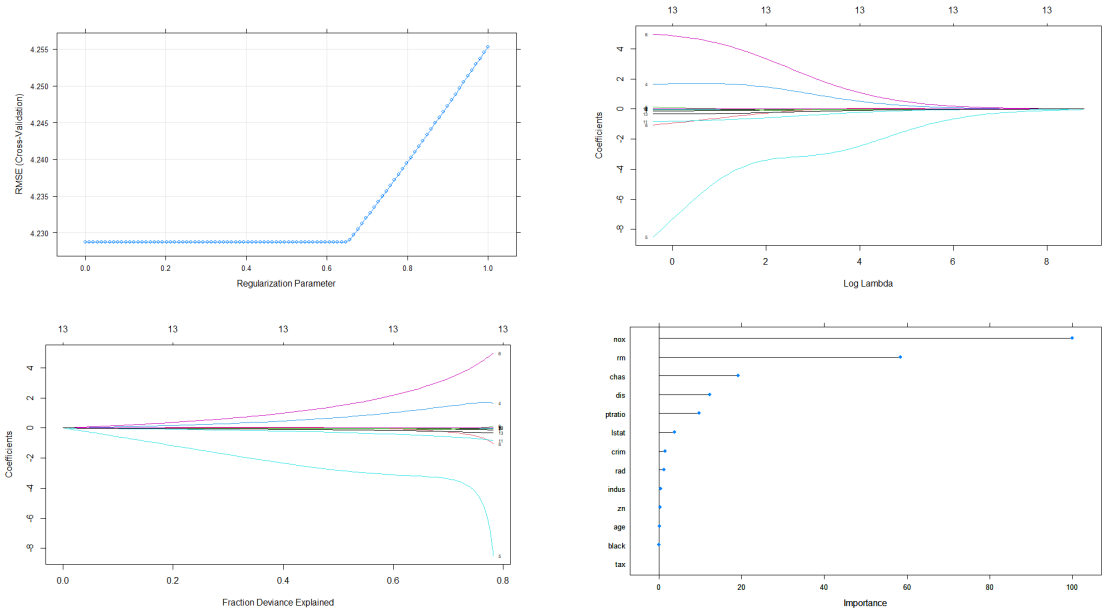



```
# Train-Test split
# set seed
set.seed(222)
n <- nrow(data)
# Generate 1 and 2 randomly for 506 times, and the probability of 1 is 0.7,
# the probability of 1 is 0.3.
ind <- sample(2, n, replace = TRUE, prob = c(0.7, 0.3))
train <- data[ind==1,] # training date
test <- data[ind==2,] # test data
dim(train)
dim(test)

# Custom Control Parameters
custom <- trainControl(method = "cv", number = 10, verboseIter = TRUE)

# Linear Model
set.seed(1234)
lm<- train(medv~., train, method='lm', trControl=custom)
lm$results
```

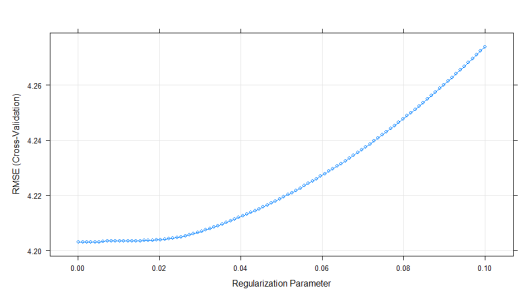
```
lm
summary(lm)
par(mfrow=c(2,2),mar=c(2,2,2,2))
plot(lm$finalModel)
```

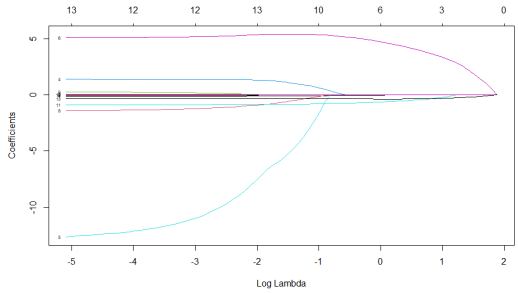


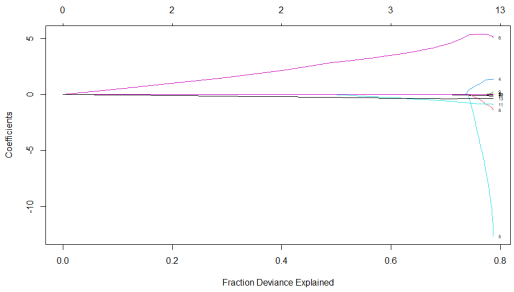

```
# Ridge Rrgression
set.seed(1234)
ridge <- train(medv~.,
               train,
               method = 'glmnet',
               tuneGrid = expand.grid(alpha = 0,
                                      lambda = seq(0, 1, length = 100)),
               trControl = custom)
ridge$results
plot(ridge)
windows(width = 10, height = 6)
plot(ridge$finalModel, xvar = "lambda", label = T)
plot(ridge$finalModel, xvar = 'dev', label=T)
plot(varImp(ridge, scale=T))
```

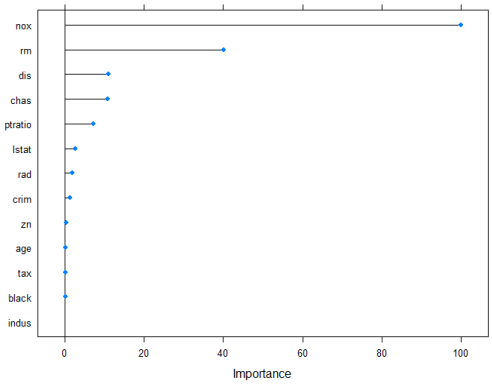



```
# Lasso Rrgression
set.seed(1234)
lasso <- train(medv~.,
               train,
               method = 'glmnet',
               tuneGrid = expand.grid(alpha = 1,
                                      lambda = seq(0.0001, 0.1, length = 100)),
               trControl = custom)

lasso$results
plot(lasso)
windows(width = 10, height = 6)
plot(lasso$finalModel, xvar = 'lambda', label=T)
plot(lasso$finalModel, xvar = 'dev', label=T)
plot(varImp(lasso,scale = T))
```

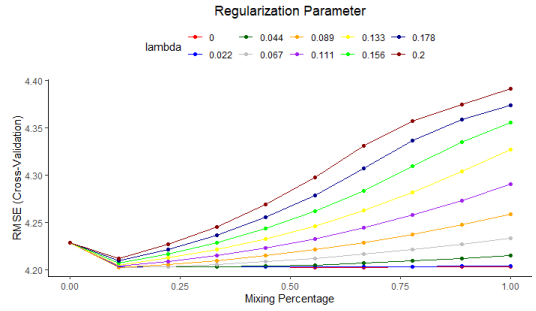


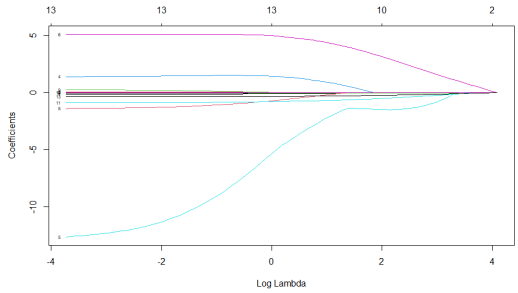


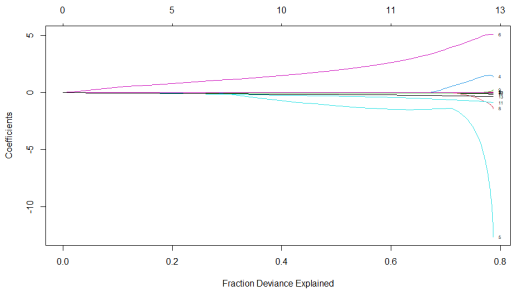


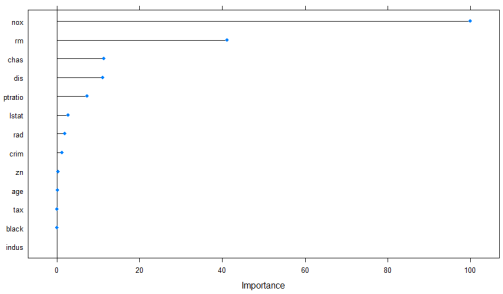



```
# Elastic Net
set.seed(1234)
en <- train(medv~.,
            train,
            method = 'glmnet',
            tuneGrid = expand.grid(alpha = seq(0,1,length=10),
                                   lambda = seq(0,0.2,length=10)),
            trControl = custom)

en$results
en_table <- data.frame(en$results)
en_table$lambda <- factor(round(en_table[,2],3))
ggplot(en_table, aes(x=alpha, y=RMSE,color=lambda)) +
  geom_point() +
  geom_line(size=0.5) +
  scale_colour_manual(values=c("red","blue","darkgreen","gray","orange","purple"
  ,"yellow","green", "darkblue","darkred"))+
  labs(title="Regularization Parameter",
       x = "Mixing Percentage", y = "RMSE (Cross-Validation)") +
  theme_classic()+
  theme(legend.position = 'top',plot.title = element_text(hjust=0.5))
windows(width = 10, height = 6)
plot(en$finalModel, xvar = 'lambda', label=T)
plot(en$finalModel, xvar = 'dev', label=T)
plot(varImp(en))
```










```
# Save Models
saveRDS(lm, "linear_regression.rds")
mlin <- readRDS("linear_regression.rds")
saveRDS(ridge, "ridge.rds")
mrid <- readRDS("ridge.rds")
saveRDS(lasso, "lasso.rds")
mlas <- readRDS("lasso.rds")
saveRDS(en, "en.rds")
men <- readRDS("en.rds")

# Compare models using MSE
RMSE_train <- data.frame(matrix(nrow=1000, ncol=4))
mlist <- list(mlin, mrid, mlas, men)
for (i in 1:1000){
  set.seed(i)
  boost_train <- sample(train,nrow(train), replace = TRUE)
  for (j in 1:4) {
    p <- predict(mlist[j], boost_train)
    RMSE_train[i,j]<-sqrt(mean((boost_train$medv-p[[1]])^2))
  }
}

Mean_RMSE_train <- c(mean(RMSE_train[,1]),mean(RMSE_train[,2]),
                     mean(RMSE_train[,3]),mean(RMSE_train[,4]))


RMSE_test <- data.frame(matrix(nrow=1000, ncol=4))
mlist <- list(mlin, mrid, mlas, men)
for (i in 1:1000){
  set.seed(i)
  boost_test <- sample(test,nrow(test), replace = TRUE)
```

```
  for (j in 1:4) {
    p <- predict(mlist[j], boost_test)
    RMSE_test[i,j]<-sqrt(mean((boost_test$medv-p[[1]])^2))
    }
}

Mean_RMSE_test <- c(mean(RMSE_test[,1]),mean(RMSE_test[,2]),
                    mean(RMSE_test[,3]),mean(RMSE_test[,4]))
```