Stat 432 Homework 3

Assigned: Sep 9, 2024; Due: 11:59 PM CT, Sep 19, 2024

- Question 1: Another Simulation Study
- Question 2: Modeling High-Dimensional Data
- Question 3: Linear Regression with Coordinate Descent

Question 1: Another Simulation Study

Here is a GPT-4 prompt I used:

> **What is the advantage of a Ridge regression? Can you suggest a simulation study to confirm that? Please write your code in R.**

The following is the answer produced by GPT-4

Ridge regression is a regularization technique that helps to solve the problem of multicollinearity in linear regression. It can handle high-dimensional data where the number of predictors may exceed the number of observations. The advantages include:

1. **Bias-Variance Tradeoff**: By adding a penalty term, ridge regression can reduce variance, thus preventing overfitting. While it does introduce bias, this tradeoff often leads to a better overall predictive performance.
2. **Stability**: It can handle ill-posed problems where multicollinearity exists, providing stable solutions.
3. **Including All Predictors**: Unlike Lasso regression, ridge regression includes all predictors in the model, shrinking their coefficients but not setting them to zero.

You can simulate the advantage of Ridge regression by comparing its performance with ordinary linear regression on a dataset with multicollinearity. Here’s an R script to conduct this study:

```
# Load required library
library(MASS)
library(glmnet)

# Simulate data with multicollinearity
set.seed(42)
n <- 100
p <- 10
X <- matrix(rnorm(n * p), n, p)
X[,2] <- X[,1] + rnorm(n) * 0.01 # Introducing multicollinearity
beta <- runif(p)
y <- X %*% beta + rnorm(n)

# Ordinary Linear Regression
lm_model <- lm(y ~ X)
lm_pred <- predict(lm_model, newdata=data.frame(X=X))

# Ridge Regression
ridge_model <- glmnet(X, y, alpha=0)
ridge_pred <- predict(ridge_model, s=0.01, newx=X)

# Compare Performance
lm_mse <- mean((y - lm_pred)^2)
ridge_mse <- mean((y - ridge_pred)^2)

cat("MSE of Ordinary Linear Regression:", lm_mse, "\n")
cat("MSE of Ridge Regression:", ridge_mse, "\n")
```

The script generates a dataset with multicollinearity and then compares the mean squared error (MSE) of ordinary linear regression and ridge regression. Typically, you'll observe that ridge regression provides a better performance in this scenario.

---

a. [15 points] Based on what we have learned, do you find this answer correct? does this code confirm the mentioned three properties of Ridge regressions? Please explain your judgement for each of them. Comment on how you would do to demonstrate these properties. You **do not need to** write the code.

Answer:

To judge the Bias-Variance Trade-off property we need to simulate train/test datasets multiple times, and each time estimate the coefficients via linear and ridge regression. Then, we could have a prediction for the test dataset for the linear and ridge regression setting. Across all simulations, we could compute the variance of our predictions for both ridge and linear regression, and compare the two. This is not happening in this code.

To judge the Stability property, we need to follow the same multiple simulation procedure as we would to judge the Bias-Variance Trade-off property, but this time we could compute the variance of the estimators for both linear and ridge regression and compare the two. This is also not happening in the code.

To judge the Including All Predictors property, we need to print the coefficients and ensure they are not zero. If they were zero, that would mean the predictor is not included. In the code above, none of the coefficients are printed, so we cannot judge this property either, although it is always theoretically true for ridge regression.

b. [25 points] To properly demonstrate the bias-variance trade-off, we could consider using a (correct) simulation. Adapt this existing code into a simulation study to show this properties. While you are doing this, please consider the following:

  - You can borrow similar ideas of simulation we used in previous lecture notes

- Modify the GPT-4 code with the following settings to generate the data:
    - trainning sample size $trainn = 50$
    - Testing sample size $testn = 200$
    - $p = 200$
    - Fix $b = rep(0.1, p)$ for all simulation runs
- Since linear regression doesn't work in this setting, you only need to consider `glmnet()`
- Use a set of λ values `exp(seq(log(0.5), log(0.01), out.length = 100))*trainn`
- Instead of evaluating the bias and variance separately (we will do that in the future), we will **use the testing error as the metric**.
- Demonstrate your result using plots and give a clear explanation of your findings. Particularly, which side of the result displays a large bias, and which side corresponds to a large variance?

Answer:

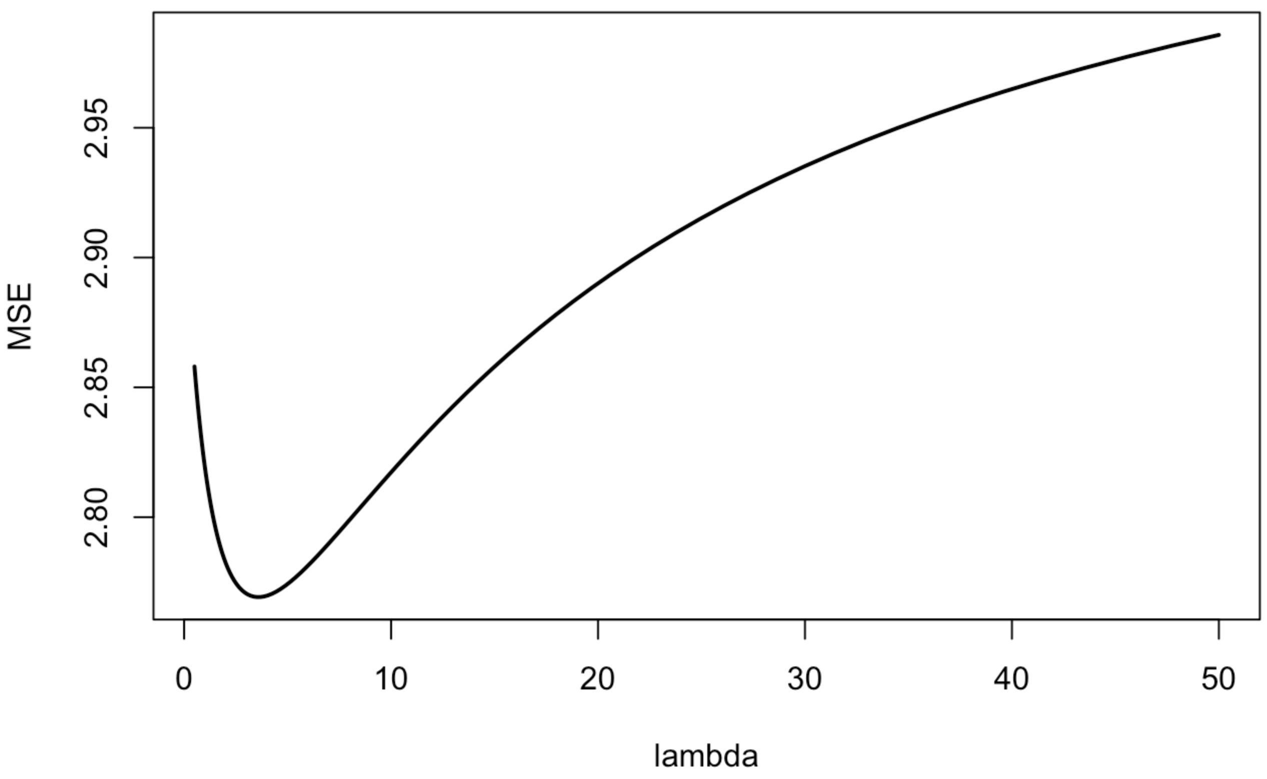
Based on the requirements, we can modify the code as follows. Specifically, we simulate the data 200 times, and calculate the prediction error of a Ridge regression at each λ value. We then average the results over all simulation runs and plot the results.

```
  # Load required library
  library(MASS)
  library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```r
# parameters
set.seed(42)
trainn = 50
testn = 200
n = trainn + testn
nsim = 200
p = 200

# set a fixed beta
b = rep(0.1, p)

# set a set of lambda
lambda = exp(seq(log(1), log(0.01), length.out = 100))*trainn

# object to record results
ridge_mse = matrix(NA, nsim, length(lambda))

# start simulation
for (k in 1:nsim)
{
  # generate data
  X <- matrix(rnorm(n * p), n, p)
  X[,2] <- X[,1] + rnorm(n) * 0.01 # Introducing multicollinearity
  y <- X %*% b + rnorm(n)

  alldata = data.frame(X, y)
  traindata = alldata[1:trainn, ]
  testdata = alldata[-(1:trainn), ]

  # Ridge Regression
  ridge_model <- glmnet(data.matrix(traindata[, 1:p]), traindata$y, alpha=0, lambda = lambda)
  ridge_pred <- predict(ridge_model, newx=data.matrix(testdata[, 1:p]))

  # Compare Performance
  ridge_mse[k, ] <- colMeans((sweep(ridge_pred, 1, testdata$y, "-"))^2)
}

# plot the results
plot(lambda, colMeans(ridge_mse), type = "l", col = c("black", "red"), lwd = 2,
     xlab = "lambda", ylab = "MSE")
```



This shows that as we changing the λ value, the testing error first decreases and then increases. This is consistent with the bias-variance trade-off. When λ is small, the model is more flexible and has a small bias but large variance. When λ is large, the model is less flexible and has a large bias but small variance.

Question 2: Modeling High-Dimensional Data

We will use the `golub` dataset from the `multtest` package. This dataset contains 3051 genes from 38 tumor mRNA samples from the leukemia microarray study Golub et al. (1999). This package is not included in `R`, but on `bioconductor`. Install the latest version of this package from `bioconductor`, and read the documentation of this dataset to understand the data structure of `golub` and `golub.cl`.

a. [25 points] We will not use this data for classification (the original problem). Instead, we will do a toy regression example to show how genes are highly correlated and could be used to predict each. Carry out the following tasks:
    - Perform marginal association test for each gene with the response `golub.cl` using `mt.teststat()`. Use `t.equalvar` (two sample t test with equal variance) as the test statistic.
    - Sort the genes by their p-values and select the top 100 genes
    - Construct a dataset with the top 10 genes and another one (call it X) with the remaining genes
    - Perform principal component analysis (PCA) on the top 100 genes and extract the first principal component, **use this as the outcome** y. Becareful about the oriantation of the data matrix.
    - Perform ridge regression with 19-fold cross-validation on X and the outcome y. Does your model fit well? Can you provide detailed model fitting results to support your claim?
    - Fit ridge regression but use GCV as the criterion. Does your model fit well?

Answer:

The following code shows how to install the `multtest` package from `bioconductor` and load the data.

We can then perform the required tasks. First, using `cv.glmnet` to perform ridge regression with 19-fold cross-validation.

```r
library(glmnet)
library(multtest)
data(golub)
```

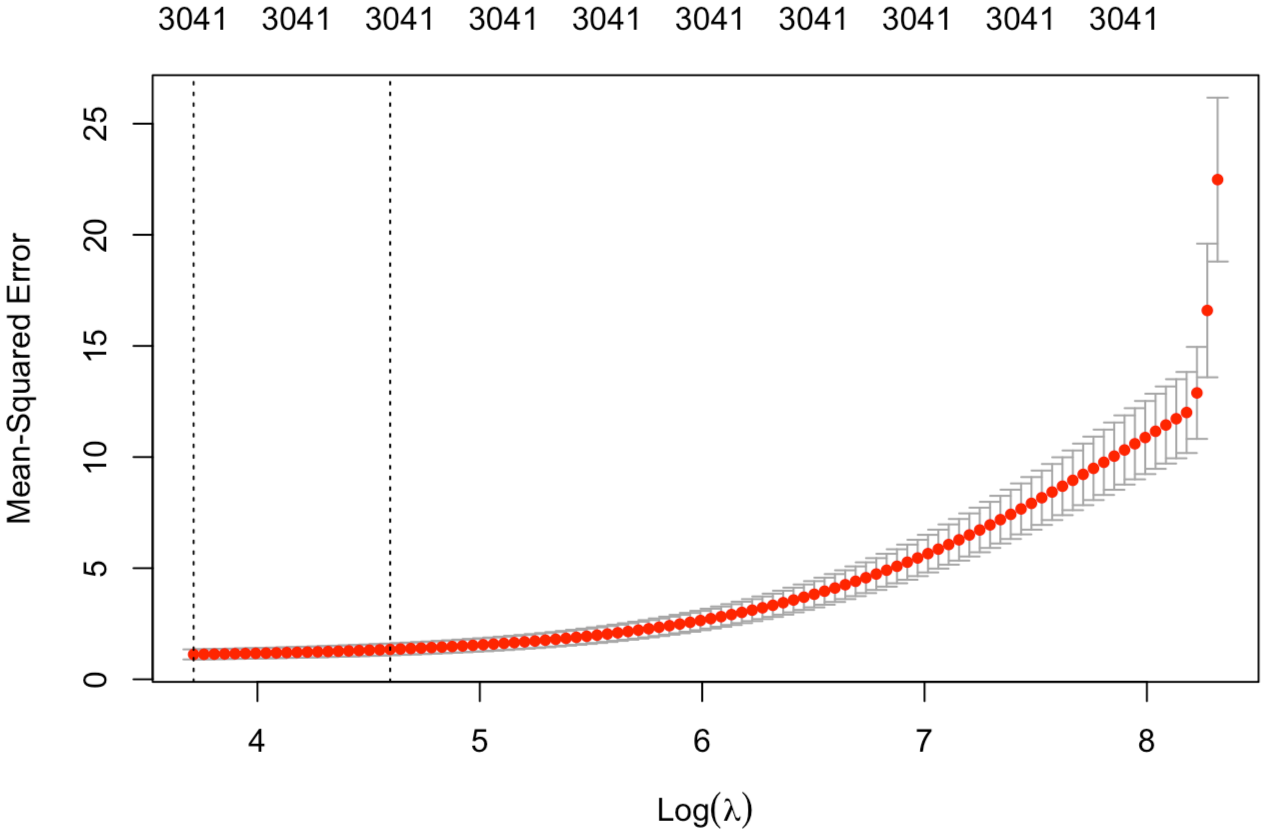
```r
# perform marginal association test for each gene with the response golub.cl
pvals <- mt.teststat(golub, golub.cl, test="t.equalvar")

# sort the names of genes by p-values
pvals <- sort(pvals, index.return=TRUE)

# select the top 100 genes
topgenes <- pvals$ix[1:100]

# top 10 genens
top10genes <- pvals$ix[1:10]

# construct the dataset with the top 10 genes and the remaining genes
y0 <- t(golub[top10genes, ])
X <- t(golub[-top10genes, ])

# perform PCA on the top 100 genes and extract the first principal component
y <- prcomp(t(golub[topgenes, ]))$x[, 1]

# perform ridge regression with 19-fold cross-validation
cvfit <- cv.glmnet(X, y, alpha=0, nfolds=19)
```

```
## Warning: Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per
## fold
```

```r
cvfit$lambda.min
```

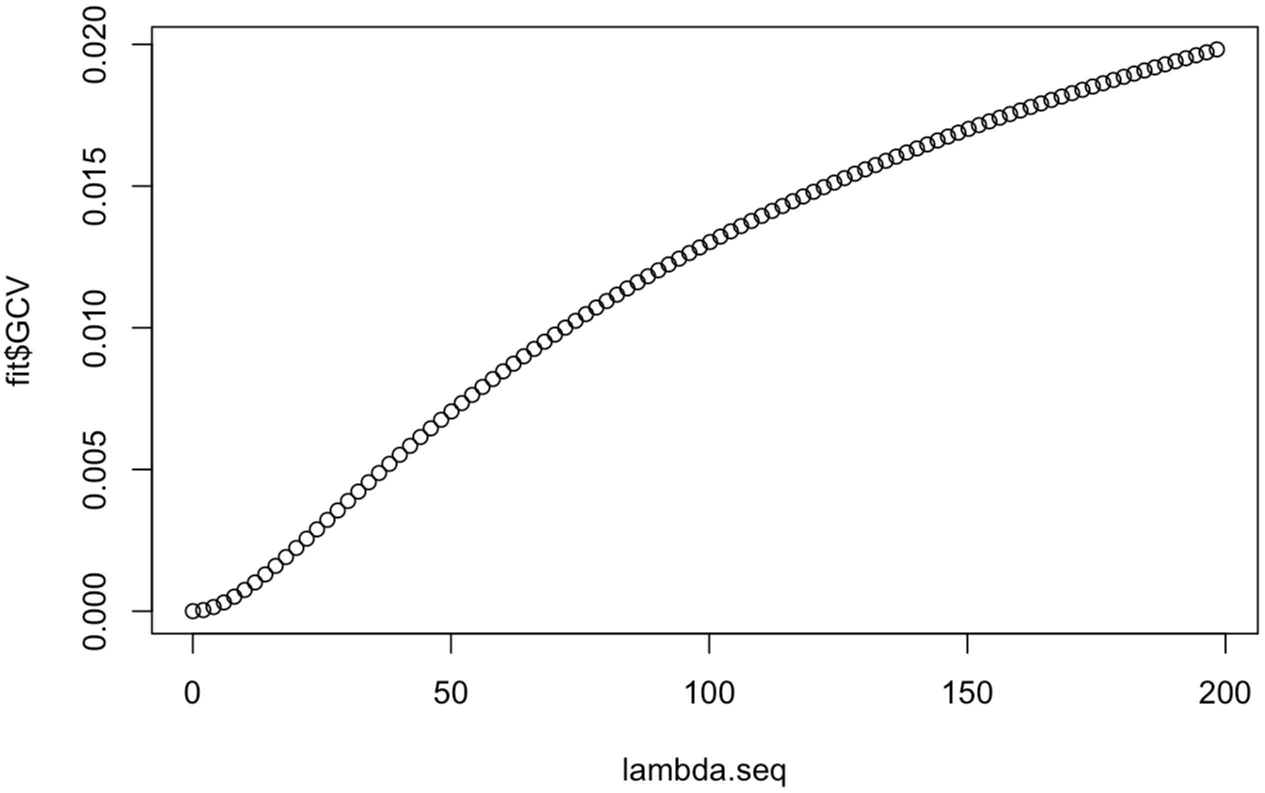
```
## [1] 40.97329
```

```r
plot(cvfit)
```



We print the lambda.min value above, this is the value of lambda that gives the smallest cross-validation error. We could have also chosen the lambda.1se if we wanted to introduce more bias. The plot above shows the mean-squared error vs lambda. Next, we fit ridge regression using the `lm.ridge()\` function and use GCV as the selection criterion.

```
# fit ridge regression using lm.ridge() and use GCV as the selection criterion
library(MASS)
lambda.seq <- seq(0.01, 2*cvfit$lambda.1se, length.out = 100)
fit <- lm.ridge(y ~ X, lambda = lambda.seq)
plot(lambda.seq, fit$GCV)
```



GCV increases as λ increases, indicating that the introduction of bias is hurting the model fit. This is because X includes the 11th-100th most important genes, which are used to come up with the y.

b. [5 points] Based on your results, do you observe any bias-variance trade-off? If not, can you explain why?

In this situation we do not see any bias-variance trade-off. This is because the linear combination we used in PCA to generate y uses the 11th-100th most important genes. Thus, it is reasonable to expect that a model that includes all those genes would have the lowest MSE. A model that includes all the genes, or predictors, would be one that has minimal bias, corresponding to a small λ being optimal. We can see from our plots in part a that MSE and GCV are increasing as λ increases.

Question 3: Linear Regression with Coordinate Descent

Recall the previous homework, we have a quadratic function for minimization. We know that analytical solution exist. However, in this example, let's use coordinate descent to solve the problem. To demonstrate this, let's consider the following simulated dataset, with design matrix x (without intercept) and response vector y:

```
set.seed(432)
n <- 100
x <- matrix(rnorm(n*2), n, 2)
y <- 0.7 * x[, 1] + 0.5 * x[, 2] + rnorm(n)
```

We will consider a model without the intercept term. In this case, our objective function (of β_1 and β_2 for linear regression is to minimize the sum of squared residuals:

$$f(\beta_1, \beta_2) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta_1 x_{i1} - \beta_2 x_{i2})^2$$

where x_{ij} represents the jth variable of the ith observation.

a. [10 points] Write down the objective function in the form of

$$f(x, y) = a\beta_1^2 + b\beta_2^2 + c\beta_1\beta_2 + d\beta_1 + e\beta_2 + f$$

by specifying what are coefficients a, b, c, d, e, and f, using the simulated data. Calculate them in R, **using vector operations rather than for-loops.**

Answer:

The objective function can be written as:

$$f(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_1 x_{i1} - \beta_2 x_{i2})^2$$

Expanding the squared term, we get:

$$f(\beta_1, \beta_2) = \frac{1}{n}\left(\sum_{i=1}^{n} y_i^2 - 2\beta_1 \sum_{i=1}^{n} x_{i1} y_i - 2\beta_2 \sum_{i=1}^{n} x_{i2} y_i + \beta_1^2 \sum_{i=1}^{n} x_{i1}^2 + 2\beta_1\beta_2 \sum_{i=1}^{n} x_{i1} x_{i2} + \beta_2^2 \sum_{i=1}^{n} x_{i2}^2\right)$$

Let $a = \sum_{i=1}^{n} x_{i1}^2$, $b = \sum_{i=1}^{n} x_{i2}^2$, $c = \sum_{i=1}^{n} x_{i1} x_{i2}$, $d = \sum_{i=1}^{n} x_{i1} y_i$, $e = \sum_{i=1}^{n} x_{i2} y_i$, and $f = \sum_{i=1}^{n} y_i^2$. Then, the objective function can be written as:

$$f(\beta_1, \beta_2) = \frac{1}{n}(a\beta_1^2 + b\beta_2^2 + 2c\beta_1\beta_2 - 2d\beta_1 - 2e\beta_2 + f)$$

To calculate these coefficients, we can use the simulated data:

```
a = sum(x[, 1]^2)
b = sum(x[, 2]^2)
c = sum(x[, 1] * x[, 2])
d = sum(x[, 1] * y)
e = sum(x[, 2] * y)
f = sum(y^2)

cat("a =", a, "\n")
```

```
## a = 92.41812
```

```
cat("b =", b, "\n")
```

```
## b = 86.25308
```

```
cat("c =", c, "\n")
```

```
## c = -13.47017
```

```
cat("d =", d, "\n")
```

```
## d = 56.1096
```

```
cat("e =", e, "\n")
```

```
## e = 25.80776
```

```
cat("f =", f, "\n")
```

```
## f = 125.586
```

b. [10 points] A coordinate descent algorithm essentially does two steps: i. Update β_1 to its optimal value while keeping β_2 fixed ii. Update β_2 to its optimal value while keeping β_1 fixed

   Write down the updating rules for β_1 and β_2 using the coordinate descent algorithm. Use those previously defined coefficients in your fomula and write them in Latex. Implement them in a for-loop algorithm in R that iterates at most 100 times. Use the initial values $\beta_1 = 0$ and $\beta_2 = 0$. Decide your stopping criterion based on the change in β_1 and β_2. Validate your solution using the lm() function.

Answer:

The updating rules for β_1 and β_2 using the coordinate descent algorithm are as follows:

i. Update β_1 to its optimal value while keeping β_2 fixed:

$$\beta_1 = \frac{d - c\beta_2}{a}$$

ii. Update β_2 to its optimal value while keeping β_1 fixed:

$$\beta_2 = \frac{e - c\beta_1}{b}$$

The following R code implements the coordinate descent algorithm:

```
beta1 = 0
beta2 = 0
max_iter = 100
tol = 1e-6

for (i in 1:max_iter)
{
  beta1_new = (d - c * beta2) / a
  beta2_new = (e - c * beta1) / b

  if (abs(beta1_new - beta1) < tol & abs(beta2_new - beta2) < tol)
  {
    break
  }

  beta1 = beta1_new
  beta2 = beta2_new
}

cat("beta1 =", beta1, "\n")
```

```
## beta1 = 0.6658953
```

```
cat("beta2 =", beta2, "\n")
```

```
## beta2 = 0.4032027
```

```
  summary(lm(y ~ x - 1))
```

```
##
## Call:
## lm(formula = y ~ x - 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.44478 -0.60651 -0.01577  0.53748  2.26058
##
## Coefficients:
##    Estimate Std. Error t value Pr(>|t|)
## x1  0.66590    0.09377   7.102 1.98e-10 ***
## x2  0.40320    0.09706   4.154 6.98e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8911 on 98 degrees of freedom
## Multiple R-squared:  0.3804, Adjusted R-squared:  0.3677
## F-statistic: 30.08 on 2 and 98 DF,  p-value: 6.525e-11
```