CSI 2300: Lecture 21 – Variable Selection

Outline

In this lecture, we'll look at how to select the independent variables that are best for modeling. We'll look in particular at:

- How to think about relevant and irrelevant variables
- Stepwise method of variable selection (forward and backward)
- Penalized regression methods (LASSO and Ridge regression)

Motivation – Which Variables should I use?

Suppose that I would like to go for a run (outdoors), and I want to predict whether today is a good day for doing that. I could use data to help me make that decision. If I gathered observations about all of the following, which of them do you think would be relevant to my decision?

- Weather-related:
  - temperature
  - wind speed
  - humidity
- Time-related:
  - the day of the year
  - the time I woke up today
- Other:
  - the amount of memory in my computer
  - the futures price of rice

Your experience might tell you that the weather-related variables seem very relevant. The two time-related variables are possibly relevant (but not clearly so). The last two are probably completely irrelevant.

The point is: just because we *can* measure something does not mean that we *should*. We should not include all possible independent variables hoping for a predictive model.

We need to be careful to select good independent variables, and avoid irrelevant variables.

Why Are Irrelevant Variables Bad?

Or: **What's the Worst that Could Happen?**

Quick answer: irrelevant variables can harm our model.

```r
# load some data, and add some irrelevant variables
suppressMessages(library(mowateR))
data(eml)

# select the dependent variable (DO) and two relevant independent variables
relevant <- subset(eml,select=c(DO, Depth, pH))
num_relevant <- ncol(relevant) - 1 # DO is not an independent variable

num_irrelevant <- 5
noise_multiplier <- 0.10
irrelevant <- as.data.frame(matrix(rnorm(nrow(eml) * num_irrelevant) * noise_multiplier,
                                   nrow(eml), num_irrelevant))

# join together the columns of the relevant and irrelevant variables
all_data <- cbind(relevant, irrelevant)

# for this example, limit the amount of data we're using to increase the effect
n <- 30 # use only a small amount of data
selected <- sample(1:nrow(eml), n)
# create a subset of the data with the selected sample
y <- eml[selected,"DO"]
small_data <- all_data[selected,]

# create a place to store the coefficients and r2 values
all_coefficients <- matrix(0, num_irrelevant + 1, num_relevant + num_irrelevant)
colnames(all_coefficients) <- colnames(all_data)[-1]
r2 <- array(0, num_irrelevant + 1)
adjusted_r2 <- array(0, num_irrelevant + 1)

# add in irrelevant variables, one by one
for (i in 0:num_irrelevant) {
    # create a matrix with all relevant variables and i irrelevant variables
    X <- small_data[,1:(1+num_relevant+i)] # 1+ means include DO
    # "DO ~ ." means "use DO as the dependent variable, and all other variables
    # as independent"
    m <- lm(DO ~ ., data=X)

    # store all the coefficients (except the intercept)
    all_coefficients[1+i,1:(num_relevant + i)] = m$coefficients[-1]
    r2[1+i] <- summary(m)$r.squared
    adjusted_r2[1+i] <- summary(m)$adj.r.squared
}

# inspect the results
round(all_coefficients, 2)
```

```
##      Depth   pH    V1    V2   V3    V4  V5
## [1,] -0.10 6.01  0.00  0.00 0.00  0.00 0.0
## [2,] -0.08 6.05 -2.09  0.00 0.00  0.00 0.0
## [3,] -0.08 6.05 -2.11 -0.12 0.00  0.00 0.0
## [4,] -0.09 6.05 -2.15  0.06 0.58  0.00 0.0
## [5,] -0.01 6.22 -3.61 -0.11 0.26 -5.81 0.0
```

```
## [6,]  0.00 6.34 -3.95  0.88 1.83 -6.91 3.4
```

```
round(r2, 2)
```

```
## [1] 0.82 0.82 0.82 0.82 0.85 0.85
```

```
round(adjusted_r2, 2)
```

```
## [1] 0.81 0.80 0.80 0.79 0.80 0.80
```

We notice several things from this example. As the number of irrelevant variables increases:

- The **coefficients for the relevant variables are not reliable**; they fluctuate.
- The **coefficients for all variables fluctuate** – and can even change sign!
- The **un-adjusted R^2 increases** – indicating that we are somehow getting a better fit with irrelevant variables. This doesn't make sense.
- The **adjusted R^2 does not increase as quickly as R^2** – which is a warning to us that these extra variables are not actually helping our model fit. This tells you to pay more attention to adjusted R^2 than un-adjusted R^2.

The point is: irrelevant independent variables are worse than irrelevant; **irrelevant variables actually harm our model's fit**. Including them causes the model to try to find a fit to irrelevant information – often called **noise**. We do **not** want to do that! We can be misled if we just pay attention to R^2 fit values.

How Can We Identify Good Independent Variables?

We mentioned a couple of ideas in the previous lecture. A good independent variable should be **correlated** with the dependent variable, and **uncorrelated** with other independent variables.

- All of the independent variables we just used were uncorrelated with each other (they were just random values)
- Only the relevant variables were correlated with the dependent variable.

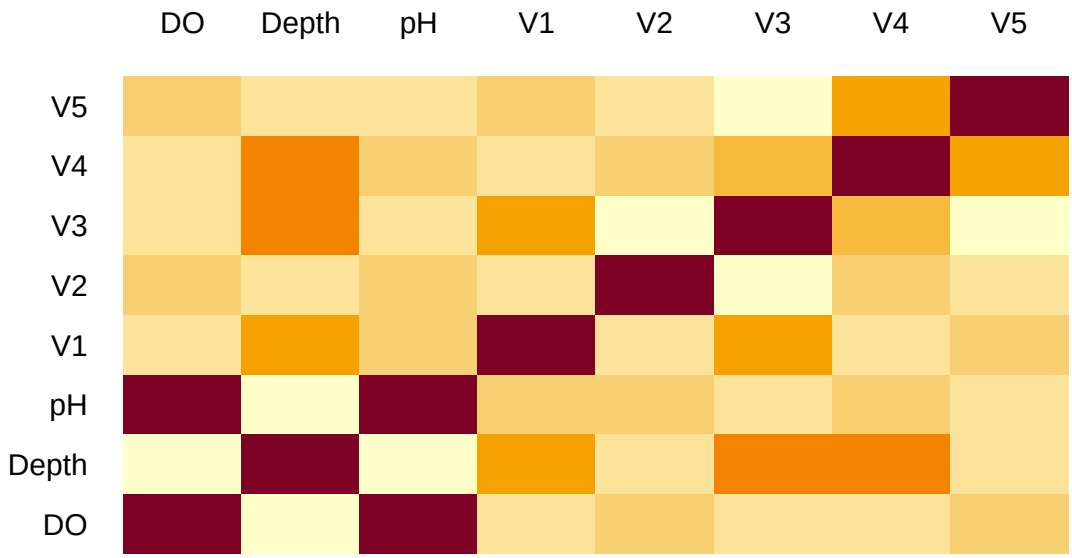
```
## correlation between y and all independent variables
round(cor(y, relevant[selected,]), 2)
```

```
##      DO Depth  pH
## [1,]  1 -0.41 0.9
```

```
round(cor(y, irrelevant[selected,]), 2)
```

```
##         V1 V2    V3    V4    V5
## [1,] -0.09  0 -0.09 -0.15 -0.06
```

```
indexes <- 1:ncol(all_data)
image(indexes, indexes, cor(small_data), axes=F, xlab="", ylab="")
axis(2, at=indexes, labels=colnames(small_data), las=1, tick=F)
axis(3, at=indexes, labels=colnames(small_data), las=1, tick=F)
```



As we can see, the relevant variables are correlated with y, but the irrelevant variables are not. And no pair of different independent variables are strongly correlated with each other.

Stepwise Variable Selection

A simple technique for choosing variables is to use either **forward** or **backward stepwise selection**. The ideas behind these two methods are similar: keep adding (for forward) or removing (for backward) variables until the model's fit no longer significantly improves (for forward) or gets worse (for backward).

Important Questions

How can we measure whether adding a variable helps?

- We don't compare R^2 values – we saw earlier how the non-adjusted R^2 tends to go up (improve) with additional independent variables.
- Instead, we use something called a *penalized likelihood* – an approximate measure of how well the model fits the data (the negative log likelihood), penalized by a measure of how many coefficients there are to estimate.
  - The negative log likelihood is a measure of fit (in some sense like R^2), but where lower is better.
  - The penalty is adding the number of coefficients in the model.
- Two popular penalized likelihoods are:
  - The AIC (Akaike Information Criterion)[1], and
  - The BIC (Bayesian Information Criterion)[2]

[1] https://en.wikipedia.org/wiki/Akaike_information_criterion
[2] https://en.wikipedia.org/wiki/Bayesian_information_criterion

How do we know which variable to add or remove next?

- Try adding (or removing) each variable to (or from) the current model.
- Measure the penalized likelihood.
- Choose the variable that improves the penalized likelihood the most.

How do we know when to stop?

Either we run out of variables to add/remove, or adding/removing a variable does not improve the penalized likelihood score.

The Stepwise Algorithms

Let's assume we have a dependent variable y and a set of independent variables $x_1, x_2, \ldots, x_k$. We want to select a subset of the x_i variables that predicts y well.

Forward Stepwise Selection:

- Start with a very simple model, e.g. `y ~ 1` (i.e., y has only an intercept).
- Let X initially contain all of the k independent variables
- While we have not converged and X is not empty:
  - Rank all $x_i \in X$ for how well adding x_i would improve the model
  - Let x^* be the best independent variable to add.
  - If adding x^* would improve the current model:
    - Add x^* to the model
    - Remove x^* from X
  - Otherwise, stop.

Backward Stepwise Elimination:

- Start with a very complex model, e.g. `y ~ x1 + x2 + ... + xk` (e.g. all independent variables).
- While we have not converged:
  - Rank all x_i in the model for how well removing x_i would improve the model
  - Let x^* be the best independent variable to remove
  - If removing x^* would improve the current model:
    - Remove x^* from the model
  - Otherwise, stop.

Example of Stepwise Variable Selection

In R, we use the `step()` command to do stepwise variable selection. Let's return to the EML example again. Here we'll use a bit more of the data.

```r
# backward selection -- start with the full model ("DO ~ ." means "use all
# independent variables in the data frame")
full_model <- lm(DO ~ ., data=all_data)

# step backward; the "k=log(n)" means "use BIC instead of AIC"; the BIC
# penalizes complex models (irrelevant variables) more than the AIC.
step(full_model, direction="backward", k=log(n))
```

```
## Start:  AIC=30047.11
## DO ~ Depth + pH + V1 + V2 + V3 + V4 + V5
##
##         Df Sum of Sq    RSS   AIC
## - V3     1         0  82707 30044
## - V1     1         0  82707 30044
## - V2     1         1  82708 30044
## - V5     1         2  82709 30044
## - V4     1         6  82713 30046
## <none>                82707 30047
## - Depth  1     11398  94105 34631
## - pH     1    172460 255167 70075
##
## Step:  AIC=30043.79
## DO ~ Depth + pH + V1 + V2 + V4 + V5
##
##         Df Sum of Sq    RSS   AIC
## - V1     1         0  82707 30041
## - V2     1         1  82708 30041
## - V5     1         2  82709 30041
## - V4     1         6  82713 30043
## <none>                82707 30044
## - Depth  1     11398  94105 34628
## - pH     1    172466 255173 70072
##
## Step:  AIC=30040.54
## DO ~ Depth + pH + V2 + V4 + V5
##
##         Df Sum of Sq    RSS   AIC
## - V2     1         1  82708 30037
## - V5     1         2  82709 30038
## - V4     1         6  82714 30040
## <none>                82707 30041
## - Depth  1     11400  94108 34625
## - pH     1    172467 255175 70069
##
## Step:  AIC=30037.41
## DO ~ Depth + pH + V4 + V5
##
##         Df Sum of Sq    RSS   AIC
## - V5     1         2  82710 30035
## - V4     1         6  82714 30037
## <none>                82708 30037
## - Depth  1     11400  94108 34622
```

```
## - pH     1    172467 255175 70065
##
## Step:  AIC=30034.74
## DO ~ Depth + pH + V4
##
##         Df Sum of Sq    RSS   AIC
## - V4     1         6  82716 30034
## <none>                82710 30035
## - Depth  1     11402  94112 34620
## - pH     1    172466 255176 70062
##
## Step:  AIC=30034.03
## DO ~ Depth + pH
##
##         Df Sum of Sq    RSS   AIC
## <none>                82716 30034
## - Depth  1     11398  94114 34618
## - pH     1    172498 255214 70064


##
## Call:
## lm(formula = DO ~ Depth + pH, data = all_data)
##
## Coefficients:
## (Intercept)        Depth           pH
##    -36.0219      -0.2115       5.2454
```

```
# forward selection -- start with the smallest model (intercept only prediction)
small_model <- lm(DO ~ 1, data=all_data)
# step forward, again with the BIC ("k=log(n)"). Note the
# "scope=formula(full_model)" which tells step() which additional variables to
# consider.
step(small_model, direction="forward", scope=formula(full_model), k=log(n))
```

```
## Start:  AIC=83316.28
## DO ~ 1
##
##         Df Sum of Sq    RSS   AIC
## + pH     1    276500  94114 34618
## + Depth  1    115401 255214 70064
## <none>                370614 83316
## + V4     1        20 370595 83318
## + V3     1        14 370600 83318
## + V1     1         2 370612 83319
## + V2     1         2 370613 83320
## + V5     1         0 370614 83320
##
## Step:  AIC=34617.74
## DO ~ pH
##
##         Df Sum of Sq   RSS   AIC
## + Depth  1   11398.4 82716 30034
```

```
## <none>                94114 34618
## + V5     1       3.9 94111 34620
## + V4     1       2.5 94112 34620
## + V1     1       2.3 94112 34620
## + V3     1       0.3 94114 34621
## + V2     1       0.2 94114 34621
##
## Step:  AIC=30034.03
## DO ~ pH + Depth
##
##        Df Sum of Sq   RSS   AIC
## <none>              82716 30034
## + V4    1    6.2633 82710 30035
## + V5    1    1.6582 82714 30037
## + V2    1    0.5915 82715 30037
## + V1    1    0.3638 82716 30037
## + V3    1    0.2198 82716 30037


##
## Call:
## lm(formula = DO ~ pH + Depth, data = all_data)
##
## Coefficients:
## (Intercept)           pH        Depth
##    -36.0219       5.2454      -0.2115
```

Note that in this example, both forward and backward selection chose the same model. But this is a coincidence; it's not always going to happen! Both methods are "greedy" – they are building models one variable at a time, without reconsidering past choices.

There is also the option of doing "all subsets" variable selection – given k independent variables, try all 2^k subsets of them (and choose the best model according to a penalized likelihood like AIC or BIC). But this can be costly when k is large.

Penalized Regression: LASSO and Ridge

Another method of choosing variables is to use a **penalized** regression model. We used a "penalized" method earlier (with AIC and BIC), where the penalty was based on the *number of coefficients*. In LASSO and Ridge regression, we use a penalty on the *values of the coefficients*.

- A coefficient with value 0 is like eliminating the corresponding variable from the model.
- Large coefficient values need to be justified by giving better model fit.

The Point Is: both LASSO and Ridge regression use *all* the variables, but *penalize large coefficient values*. Hopefully this requires some of the coefficients to be (effectively) zero. *A coefficient of zero implies we might be able to eliminate the variable.*

The Math behind Penalized Regression

Suppose that we have a model with k independent variables $x_1, \ldots, x_k$ and $k+1$ coefficients $w_0, w_1, \ldots, w_k$ (w_0 is for the intercept). Recall that the standard linear regression model chooses the w_i values to minimize the following ("least-squares"):

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where $\hat{y}_i = w_0 + w_1 x_1 + \ldots + w_k x_k$ is the model's prediction.

LASSO and Ridge regression both minimize the *same* least-squares function, but also add a constraint:

- LASSO: minimize $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ subject to this constraint: $\sum_{j=1}^{k} |w_i| \leq t$.
- Ridge: minimize $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ subject to this constraint: $\sum_{j=1}^{k} w_i^2 \leq t$.

Note the difference: a sum of absolute values, versus a sum of squared values. Also, note that we have to pick the limiting value t. Here's a picture of a way to think about the constraints in the *coefficient space*.

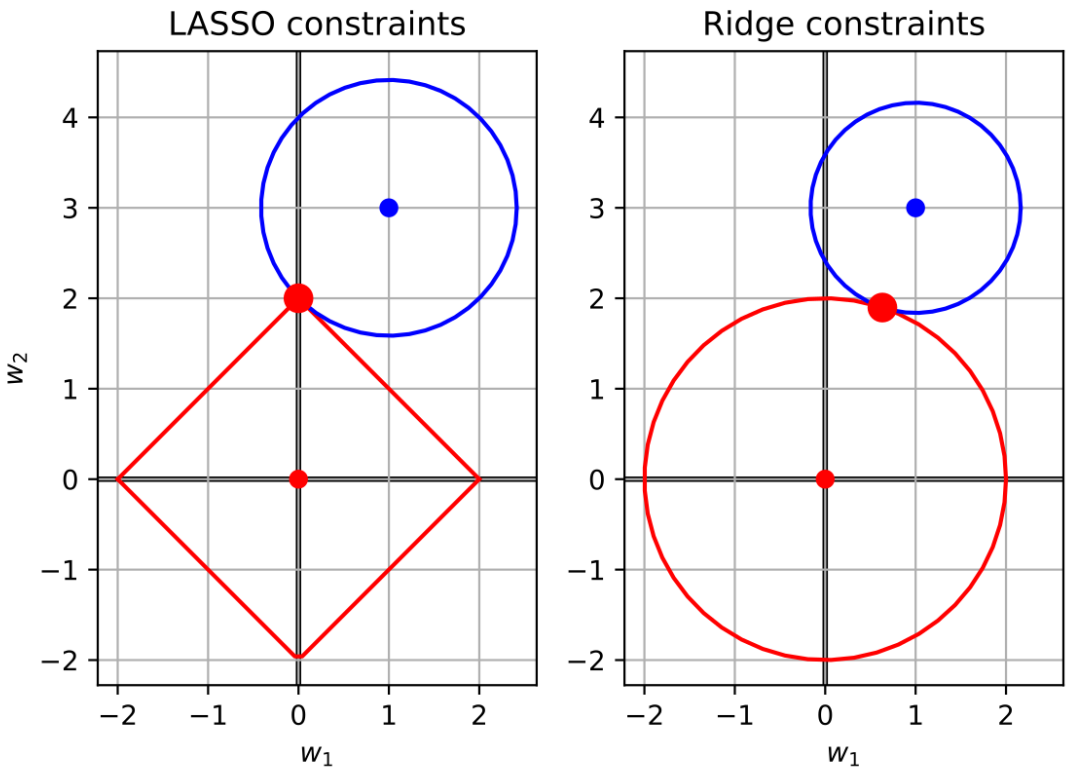


Figure 1: LASSO and Ridge regression constraints in coefficient space

We can think of the LASSO as using a diamond-shaped constraint space, and Ridge as using a sphere-shaped constraint space, as seen above. On each plot:

- the red diamond and circle indicate the constraints;
- the blue dot indicates the un-penalized least-squares fit (what `lm()` would give);
- the blue circles indicate a constant value of R^2;
- the large red dots indicate the coefficient values that each penalty would choose.

Note that the LASSO penalty chooses $w_1 = 0$, while Ridge has non-zero values for both w_1 and w_2. The difference in these shapes (diamond vs. sphere) causes different penalization behavior. You can reason geometrically that the diamond shape of LASSO tends to cause it to choose more coefficient values as exactly 0 (compared to Ridge regression), as we'll see again later.

Enough Math, Let's LASSO that Ridge

In R, we can use the `glmnet` and `cv.glmnet` functions for both LASSO and Ridge regression. Fortunately, it will search over many values of t mentioned above, but for (*complicated reasons about mathematical duality*), we will refer to a complementary value called λ. All you need to know is:

- lower t implies higher λ, and vice-versa;
- lower t (higher λ) is more constrained (pushes the coefficient values closer to 0)

We'll use the `cv.glmnet` method, which searches over many λ values, and chooses what it thinks is the best, using a technique known as *cross-validation*. We'll learn more about cross-validation later in the semester.

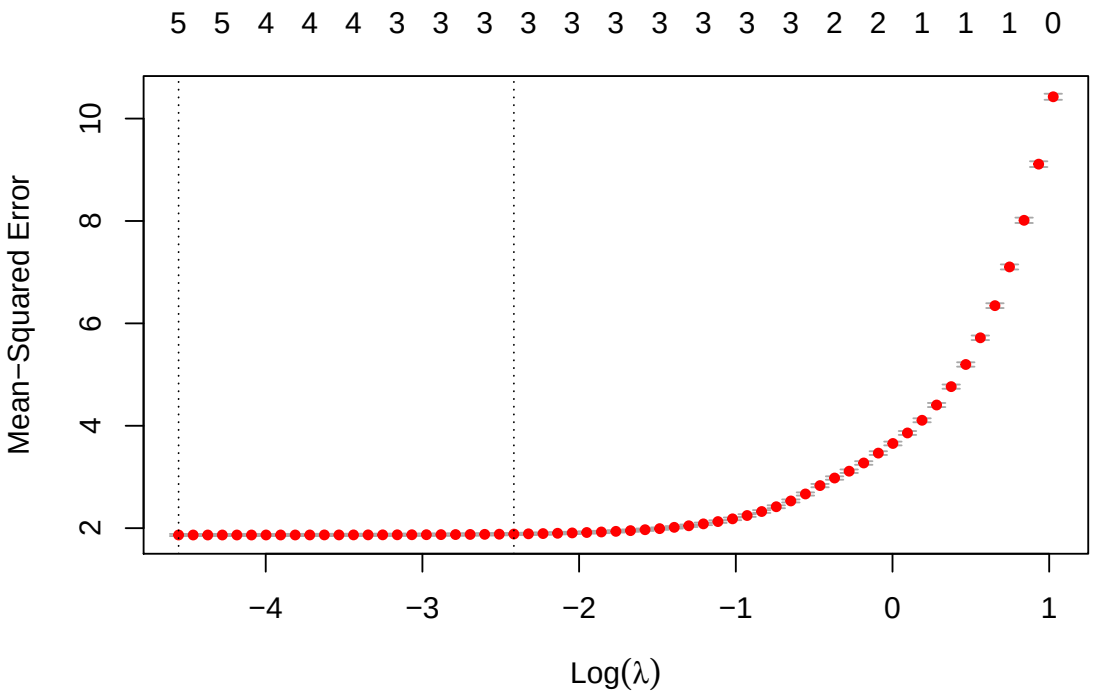
```
suppressMessages(library(glmnet))

# don't make predictions based on these variables
leave_out <- which((colnames(eml) == "DO") | (colnames(eml) == "DOsat") | (colnames(eml) == "Date.Time"))

# create our independent variables (x) and dependent variable (y)
irrelevant <- matrix(rnorm(nrow(eml) * 5), nrow(eml), 5)
colnames(irrelevant) <- paste("i", 1:5, sep="")
x <- cbind(as.matrix(eml[,-leave_out]), irrelevant)
y <- eml$DO

lasso_fit <- cv.glmnet(x, y)
coef(lasso_fit)
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) -41.87367940
## Depth        -0.21503079
## Temp          .
## pH            5.00162252
## Cond          0.02154442
## i1            .
## i2            .
## i3            .
## i4            .
## i5            .
```

The `lasso_fit` object is complicated. The `cv.glmnet` method tries many values of λ using cross-validation ("cv"). But if we call `coef(lasso_fit)`, it selects the one for `lasso_fit$lambda.1se`, (1se means one standard error) which is the largest value of λ which achieves a close-enough error to the model that used almost all coefficients. Put another way, it is the simplest model that is still getting low error.

Note that the LASSO fit eliminated all five irrelevant variables, and also `Temp`.

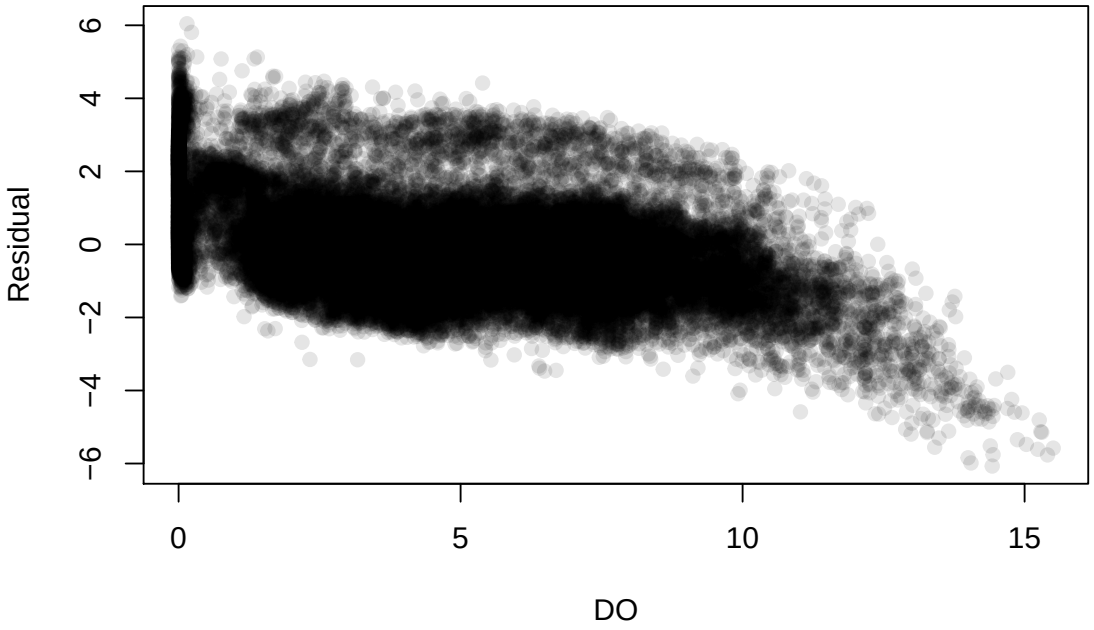
```
plot(lasso_fit)
```



This shows how the mean squared error (MSE), which is an average of the SSR, changes as we increase λ. The minimal λ and the 1-standard error λ are plotted with vertical dotted lines. Above the plot shows how many coefficients are non-zero.

Here's an example of how to make predictions and compute the residuals for the selected LASSO model.

```
lasso_predictions <- predict(lasso_fit, x)
lasso_residuals <- lasso_predictions - y
plot(y, lasso_residuals, pch=19, col=rgb(0, 0, 0, 0.1), xlab="DO", ylab="Residual",
     main="Residuals of LASSO fit")
```



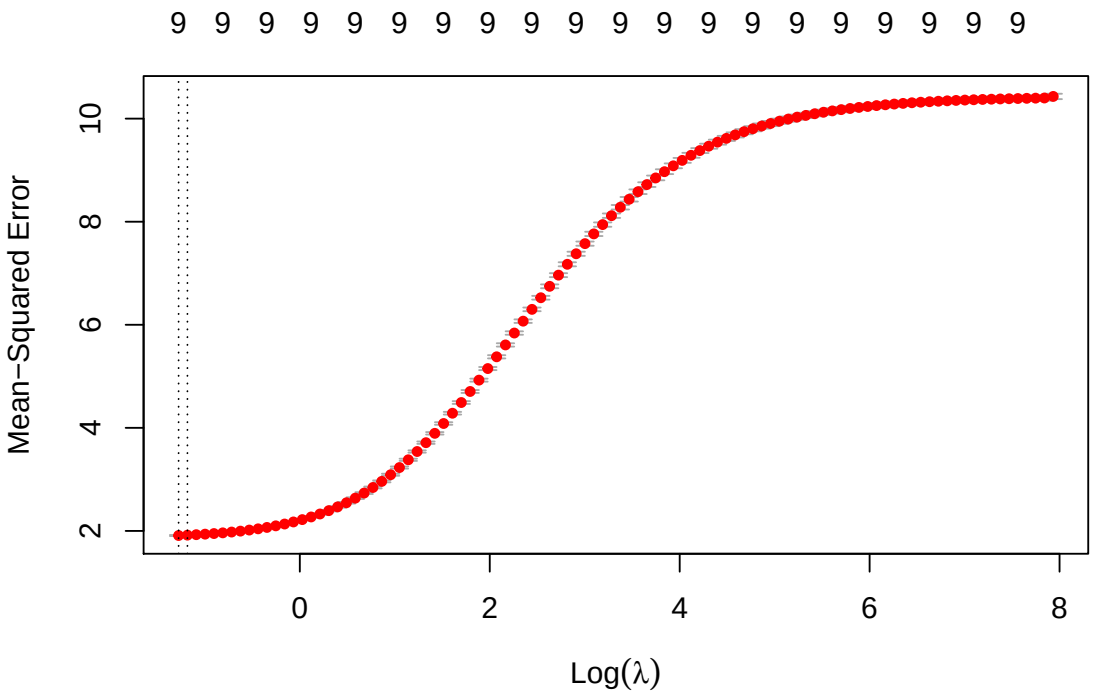
Lastly, if you want to extract the coefficients and/or predictions for a *different* lambda, you can add the argument `s=lambda`, where `lambda` is the value you want to the calls `coef` and `predict`.

Here is the same sort of information for Ridge regression.

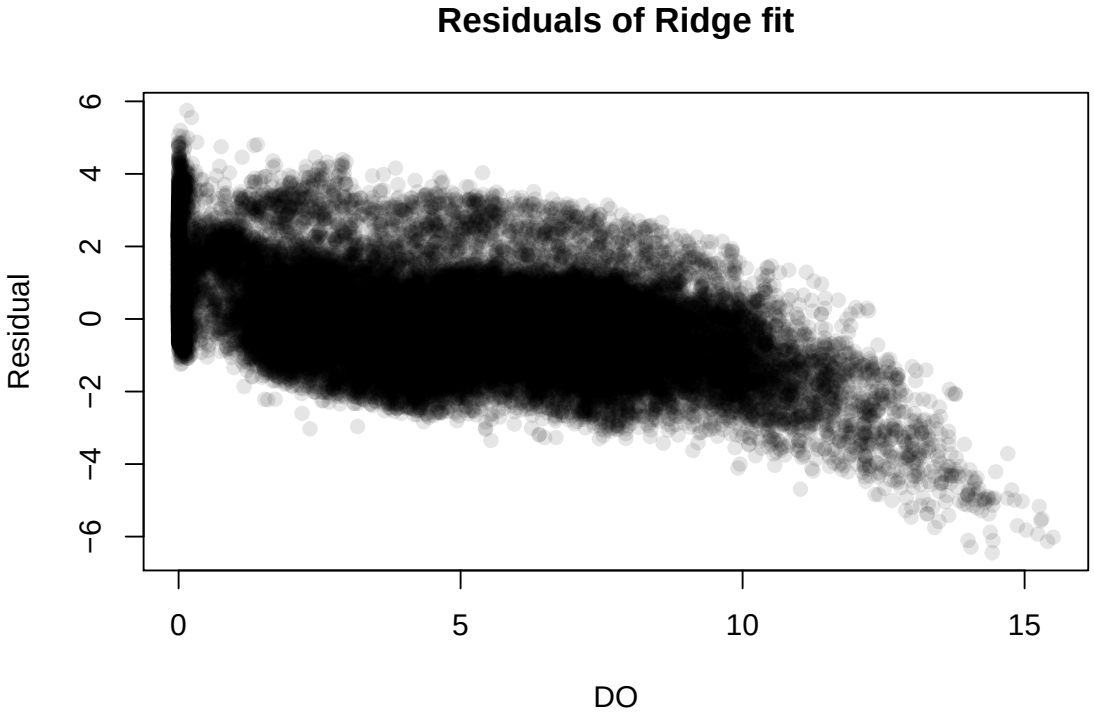
```
ridge_fit <- cv.glmnet(x, y, alpha=0) # alpha=0 implies Ridge regression
coef(ridge_fit)
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept) -35.404273863
## Depth        -0.258508677
## Temp         -0.058522845
## pH            4.580481192
## Cond          0.017810921
## i1           -0.001328896
## i2           -0.015325817
## i3            0.006384721
## i4            0.003309571
## i5            0.008634728
```

```
plot(ridge_fit)
```




```
ridge_predictions <- predict(ridge_fit, x)
ridge_residuals <- ridge_predictions - y
plot(y, ridge_residuals, pch=19, col=rgb(0, 0, 0, 0.1), xlab="DO", ylab="Residual",
     main="Residuals of Ridge fit")
```



There are several things to note here:

- LASSO was able to completely eliminate the fake irrelevant variables.
  - Ridge gave the irrelevant variables coefficient values close to 0.
- LASSO completely eliminated `Temp`, while Ridge again gave it a fairly small coefficient value.
- As $\log(\lambda)$ increases, that increases the penalty for model complexity (large coefficient values), which forces the models to reduce their coefficient values, which makes for slightly worse fits in terms of Mean-Squared Error (which is an average of SSR, which we've seen before).