MATH 567: Mathematical Techniques in Data Science
Lab 2

Dominique Guillot

Departments of Mathematical Sciences
University of Delaware

February 15, 2016

Exercise

1. Load the `Auto` dataset.
2. Use the `lm` function to fit a linear model $\texttt{mpg} = \beta_0 + \beta_1 \cdot \texttt{horsepower} + \beta_2 \cdot \texttt{weight}$.
3. Compute the coefficients directly by solving the normal equations. Do you get the same results?

Note: You may need to convert the data frame to a matrix using `as.matrix(X)`.

Exercise

1. Load the `Auto` dataset.
2. Use the `lm` function to fit a linear model $\texttt{mpg} = \beta_0 + \beta_1 \cdot \texttt{horsepower} + \beta_2 \cdot \texttt{weight}$.
3. Compute the coefficients directly by solving the normal equations. Do you get the same results?

Note: You may need to convert the data frame to a matrix using `as.matrix(X)`.

If you do not get the same results: did you include an intercept in the normal equations?

```
X = as.matrix(Auto[,c(4,5)])
Xp = cbind(matrix(1,392,1), X)
```

Training error and test error

Complexity of the model:

- A complex model that fits data very well will often make poor predictions. **Overfitting**.

Training error and test error

Complexity of the model:

- A complex model that fits data very well will often make poor predictions. **Overfitting**.
- On the other hand, a very simple model may not capture the complexity of the data. **Underfitting**.

Training error and test error

Complexity of the model:

- A complex model that fits data very well will often make poor predictions. **Overfitting**.
- On the other hand, a very simple model may not capture the complexity of the data. **Underfitting**.

To test the ability of a model to predict new values:

1. We split our data into 2 parts (training data and test data) as uniformly as possible. People often use 75% training, 25% test.

Training error and test error

Complexity of the model:

- A complex model that fits data very well will often make poor predictions. **Overfitting**.
- On the other hand, a very simple model may not capture the complexity of the data. **Underfitting**.

To test the ability of a model to predict new values:

1. We split our data into 2 parts (training data and test data) as uniformly as possible. People often use 75% training, 25% test.
2. We fit our model using the training data only. (This minimizes the **training error**).

Training error and test error

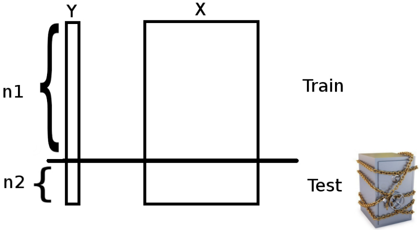
Complexity of the model:

- A complex model that fits data very well will often make poor predictions. **Overfitting**.
- On the other hand, a very simple model may not capture the complexity of the data. **Underfitting**.

To test the ability of a model to predict new values:

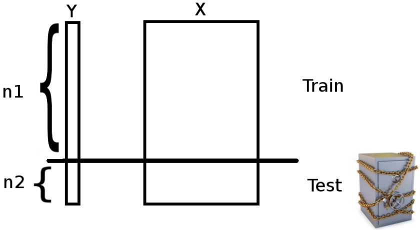
1. We split our data into 2 parts (training data and test data) as uniformly as possible. People often use 75% training, 25% test.
2. We fit our model using the training data only. (This minimizes the **training error**).
3. We use the fitted model to predict values of the test data and compute the **test error**.

Splitting data into training/test data:



In the case of least squares:

Splitting data into training/test data:

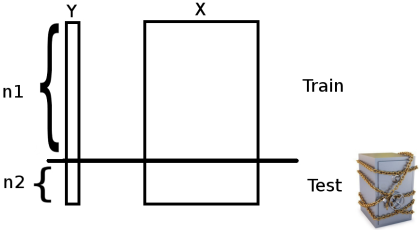


In the case of least squares:

1. $\hat{\beta} = (X_{\text{train}}^T X_{\text{train}})^{-1} X_{\text{train}}^T Y_{\text{train}}$.

Training error and test error (cont.)

Splitting data into training/test data:

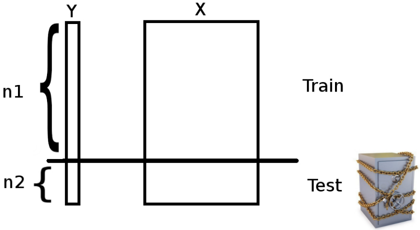


In the case of least squares:

1. $\hat{\beta} = (X_{\text{train}}^T X_{\text{train}})^{-1} X_{\text{train}}^T Y_{\text{train}}$.
2. $\widehat{Y}_{\text{test}} = X_{\text{test}} \hat{\beta}$.

Training error and test error (cont.)

Splitting data into training/test data:



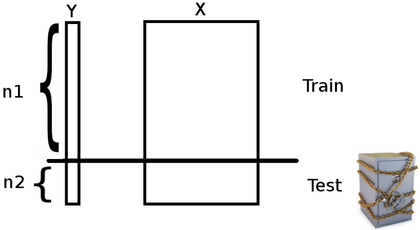
In the case of least squares:

1. $\hat{\beta} = (X_{\text{train}}^T X_{\text{train}})^{-1} X_{\text{train}}^T Y_{\text{train}}$.
2. $\widehat{Y}_{\text{test}} = X_{\text{test}} \hat{\beta}$.
3. Test error:

$$\text{MSE}_{\text{test}} = \frac{1}{n_2} \sum_{i=1}^{n_2} (\widehat{Y}_{\text{test},i} - Y_{\text{test},i})^2.$$

Training error and test error (cont.)

Splitting data into training/test data:



In the case of least squares:

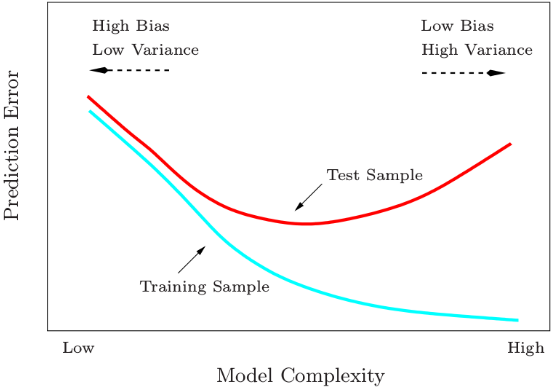
1. $\hat{\beta} = (X_{\text{train}}^T X_{\text{train}})^{-1} X_{\text{train}}^T Y_{\text{train}}$.
2. $\widehat{Y}_{\text{test}} = X_{\text{test}} \hat{\beta}$.
3. Test error:

$$\text{MSE}_{\text{test}} = \frac{1}{n_2} \sum_{i=1}^{n_2} (\widehat{Y}_{\text{test},i} - Y_{\text{test},i})^2.$$

We choose a model that minimizes the test error.

Training error and test error (cont.)

Typical behavior of the test and training error, as model complexity is varied.



ESL, Fig 2.11.

Train/test sets in R

```
library(ISLR)
data(Auto)

Auto <- Auto[,-9]  # Remove the "names" column

n <- dim(Auto)[1]

ntrain <- floor(0.75*n)
ntest <- n - ntrain

train_ind <-  sample(1:n, ntrain)

train <-  Auto[train_ind,]
test <-  Auto[-train_ind,]
```

Train/test sets in R

```
library(ISLR)
data(Auto)

Auto <- Auto[,-9]  # Remove the "names" column

n <- dim(Auto)[1]

ntrain <- floor(0.75*n)
ntest <- n - ntrain

train_ind <-  sample(1:n, ntrain)

train <-  Auto[train_ind,]
test <-  Auto[-train_ind,]
```

Compute the test error:

```
model_full <- lm(mpg ~ ., data=train)
mean((predict(model_full, test[,-1]) - test[,1])**2)
```

Using a subset of variables

Fit a model using only the last 3 variables:

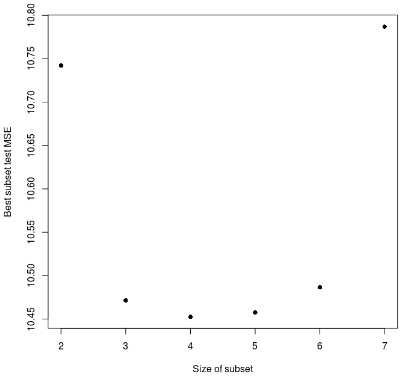
```
model <- lm(mpg ~ ., data=train[,append(c(5,7,8),1)])
mean((predict(model, test[,c(5,7,8)]) - test[,1])**2)
```

Using a subset of variables

Fit a model using only the last 3 variables:

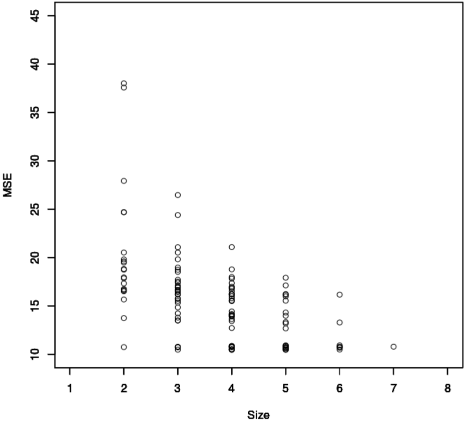
```
model <- lm(mpg ~ ., data=train[,append(c(5,7,8),1)])
mean((predict(model, test[,c(5,7,8)]) - test[,1])**2)
```

Minimal test error for subsets of a given size:



Examining all subsets

For this dataset, we can examine all the possible subsets (usually impossible):



Best subset selection

Best subset selection: Given $k \in \{1, \ldots, p\}$, we find the subset of size k of $\{1, \ldots, p\}$ that minimizes the prediction error.

Best subset selection

Best subset selection: Given $k \in \{1, \ldots, p\}$, we find the subset of size k of $\{1, \ldots, p\}$ that minimizes the prediction error.

- Note: there are $\binom{p}{k}$ subsets of size k and 2^k possible subsets. So the procedure is only computationally feasible for small values of p.

Best subset selection

Best subset selection: Given $k \in \{1, \ldots, p\}$, we find the subset of size k of $\{1, \ldots, p\}$ that minimizes the prediction error.

- Note: there are $\binom{p}{k}$ subsets of size k and 2^k possible subsets. So the procedure is only computationally feasible for small values of p.
- The leaps and bounds procedure (Furnival and Wilson, 1974) makes this feasible for p as large as 30 or 40.

Forward- and Backward- stepwise regression

- Best subset selection performs well, but is too computationally intensive to be useful in practice.

Forward- and Backward- stepwise regression

- Best subset selection performs well, but is too computationally intensive to be useful in practice.

Two natural "greedy" variants of the best subset selection technique:

- **Forward stepwise regression:** starts with the intercept, and then sequentially adds into the model the predictor that most improves the fit.

Forward- and Backward- stepwise regression

- Best subset selection performs well, but is too computationally intensive to be useful in practice.

Two natural "greedy" variants of the best subset selection technique:

- **Forward stepwise regression:** starts with the intercept, and then sequentially adds into the model the predictor that most improves the fit.
- **Backward stepwise regression:** starts with the full model, and sequentially deletes the predictor that has the least impact on the fit.

Forward- and Backward- stepwise regression

- Best subset selection performs well, but is too computationally intensive to be useful in practice.

Two natural "greedy" variants of the best subset selection technique:

- **Forward stepwise regression:** starts with the intercept, and then sequentially adds into the model the predictor that most improves the fit.
- **Backward stepwise regression:** starts with the full model, and sequentially deletes the predictor that has the least impact on the fit.

Can be used even when the number of variables is very large.
However,

Forward- and Backward- stepwise regression

- Best subset selection performs well, but is too computationally intensive to be useful in practice.

Two natural "greedy" variants of the best subset selection technique:

- **Forward stepwise regression:** starts with the intercept, and then sequentially adds into the model the predictor that most improves the fit.
- **Backward stepwise regression:** starts with the full model, and sequentially deletes the predictor that has the least impact on the fit.

Can be used even when the number of variables is very large. However,

- Greedy approach: doesn't guarantee a global optimum.
- Less rigorous than other methods, less supporting theory.

Forward- and Backward- stepwise regression

- Best subset selection performs well, but is too computationally intensive to be useful in practice.

Two natural "greedy" variants of the best subset selection technique:

- **Forward stepwise regression:** starts with the intercept, and then sequentially adds into the model the predictor that most improves the fit.
- **Backward stepwise regression:** starts with the full model, and sequentially deletes the predictor that has the least impact on the fit.

Can be used even when the number of variables is very large. However,

- Greedy approach: doesn't guarantee a global optimum.
- Less rigorous than other methods, less supporting theory.

Nevertheless, the stepwise approaches often return predictors similar to the predictors obtained from more complex methods with better theory.

The R package leaps

1. Install and load the `leaps` package.
2. Use the `regsubsets` function to perform forward and backward stepwise regressions.

```
library(leaps)

regfit.fwd = regsubsets(mpg ~ ., data=Auto[,-9],
   method="forward")

regfit.bwd = regsubsets(mpg ~ ., data=Auto[,-9],
   method="backward")
```

3. Examine the output of `summary(regfit.fwd)` and `plot(regfit.fwd)`.