

01. Statistical Learning and Cross-validation

Table of Contents

- ① Introduction to statistical learning
 - Supervised learning problem
 - Prediction and inference
 - Parametric and non-parametric methods
 - Flexibility and interpretability
- ② Assessing model accuracy
 - Cubic model
 - Linear model
 - Nonlinear model
- ③ Bias-variance trade-off
- ④ Training errors and test errors
- ⑤ Cross-validation
 - Validation set approach
 - K-fold cross-validation
 - Leave-one-out cross-validation

Statistical Learning

- **Statistical learning** is a set of tools for modeling and understanding complex datasets.
- It is recently developed area in statistics along with **machine learning** in computer science.
- With the explosion of “**Big Data**” problems, statistical learning has become a very hot field in many scientific areas.
- The tools of statistical learning can be classified as **supervised** or **unsupervised** learning.
- **Supervised statistical learning** builds a statistical model for predicting or estimating for data with an output based on one or more inputs.
- **Unsupervised statistical learning** learns relationships and structure from data that has inputs but no supervising output.

Supervised Learning Problem

- Outcome measurement Y (also called dependent variable, response, target).
- Vector of p predictor measurements X (also called independent variables, inputs, regressors, features).
- In the **regression problem**, Y is quantitative (e.g numerical values such as blood pressure and weight).
- In the **classification problem**, Y takes values in a finite, unordered set (survived/died, cancer class of tissue sample, case/control).
- We have n observations consisting of $(x_1, y_1), \dots, (x_n, y_n)$, where

$$x_i = (x_{i1}, x_{i2}, \dots, x_{ip})^T$$

is the p -dimensional vector.

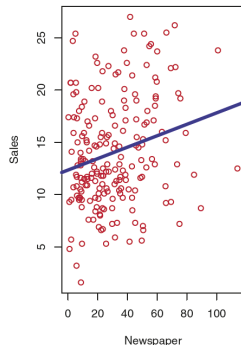
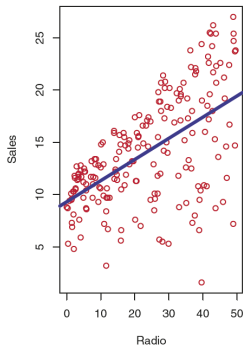
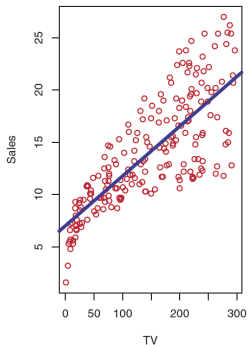
Objectives and Philosophy

- On the basis of the training data we would like to:
 - Accurately predict unseen **test cases**.
 - Understand which inputs affect the outcome, and how.
 - Assess the quality of our predictions and inferences.
- Philosophy
 - It is important to understand the ideas behind the various techniques, in order to know how and when to use them.
 - One has to understand the simpler methods first, in order to grasp the more sophisticated ones.
 - It is important to accurately assess the performance of a method, to know how well or how badly it is working (simpler methods often perform as well as fancier ones!)
 - This is an exciting research area, having important applications in **high-dimensional data analysis**.

Statistical Learning vs. Machine Learning

- Machine learning arose as a subfield of Artificial Intelligence.
- Statistical learning arose as a subfield of Statistics.
- There is much overlap — both fields focus on supervised and unsupervised problems:
 - Machine learning has a greater emphasis on large scale applications and prediction accuracy.
 - Statistical learning emphasizes models, interpretability, precision and uncertainty.
- But, the distinction has become more and more blurred, and there is a great deal of “cross-fertilization”.

Advertising Data



- Shown are Sales vs TV, Radio and Newspaper, with a blue linear-regression line fit separately to each.
- Can we predict Sales using these three?
- Perhaps we can do better using a model

$$\text{Sales} \approx f(\text{TV}, \text{Radio}, \text{Newspaper})$$

Advertising Data

```
## Open the dataset linked to the book website
url.ad <- "https://www.statlearning.com/s/Advertising.csv"
Advertising <- read.csv(url.ad, h=T)
attach(Advertising)
```

```
## Least square fit for simple linear regression
par(mfrow = c(1,3))
plot(sales~TV, col=2, xlab="TV", ylab="Sales")
abline(lm(sales~TV)$coef, lwd=3, col="darkblue")
```

```
plot(sales~radio, col=2, xlab="Radio", ylab="Sales")
abline(lm(sales~radio)$coef, lwd=3, col="darkblue")
```

```
plot(sales~newspaper, col=2, xlab="Newspaper", ylab="Sales")
abline(lm(sales~newspaper)$coef, lwd=3, col="darkblue")
```


Notation

- Here **Sales** is a **response** or target that we wish to predict. We generically refer to the response as Y .
- **TV** is a **feature**, or input, or predictor; we name it X_1 . Likewise name **Radio** as X_2 , and so on.
- We can refer to the **input vector** collectively as

$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix}$$

- We assume that there is some **relationship** between Y and $X = (X_1, X_2, X_3)^T$

Supervised Learning: Model

- Now we write our model as

$$Y = f(X) + \epsilon$$

where ϵ captures measurement errors and other discrepancies.

- With a good $f(\cdot)$ we can make predictions of Y at new points $X = x$.
- Depending on the complexity of $f(\cdot)$, we may be able to understand how each component X_j of X affects Y .
- The function f that connects the input variable to the output variable is in general **unknown**.
- **Statistical learning** refers to a set of approaches for estimating the function f .

Advertising Data

```
AD <- Advertising[ , -1]
```

```
## Multiple linear regression  
lm.fit <- lm(sales ~ ., AD)  
summary(lm.fit)  
names(lm.fit)  
coef(lm.fit)  
confint(lm.fit)
```

```
par(mfrow=c(2,2))  
plot(lm.fit)
```

```
dev.off()  
plot(predict(lm.fit), residuals(lm.fit))  
plot(predict(lm.fit), rstudent(lm.fit))  
plot(hatvalues(lm.fit))  
which.max(hatvalues(lm.fit))
```

Estimation of f for Prediction

- There are two main reasons that we may wish to estimate f :
 - prediction
 - inference

- We can predict

$$\hat{Y} = \hat{f}(X)$$

where \hat{f} represents our estimate for f , and \hat{Y} represents the resulting prediction for Y .

- In this setting, \hat{f} is often treated as a **black box**, in the sense that one is not typically concerned with the exact form of \hat{f} , provided that it yields **accurate predictions** for Y .
- Statistically, an ideal function $f(\cdot)$ is

$$f(x) = E(Y|X = x)$$

which is called the **regression function**.

How to find $f(\cdot)$

- For any estimate $\hat{f}(x)$ of $f(x)$, we have

$$\begin{aligned} E[(Y - \hat{f}(X))^2 | X = x] &= E[f(x) + \epsilon - \hat{f}(x)]^2 \\ &= \underbrace{E[(f(x) - \hat{f}(x))^2]}_{\text{Reducible}} + \underbrace{\text{Var}(\epsilon)}_{\text{Irreducible}} \end{aligned}$$

- **Irreducible** error: even if we knew $f(x)$, we would still make errors in prediction, since at each $X = x$ there is typically a distribution of possible Y values.

$$\epsilon = Y - f(x)$$

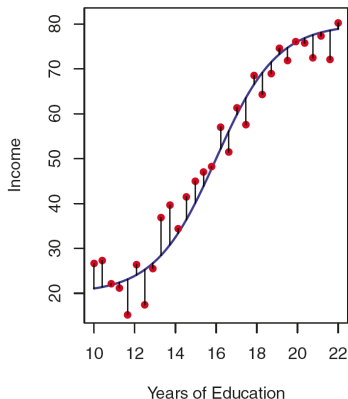
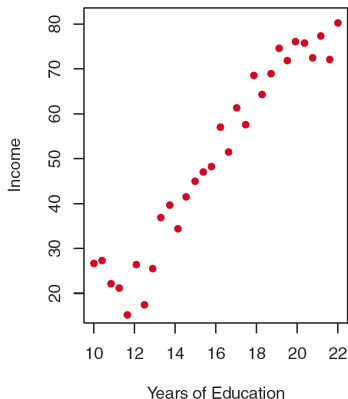
- The focus is on statistical learning techniques for estimating f with the aim of minimizing the **reducible error**.

Estimation of f for Inference

- We are often interested in understanding the way that Y is affected as X_1, \dots, X_p change.
- In this situation we wish to estimate f , but our goal is not necessarily to make predictions for Y .
- Now \hat{f} cannot be treated as a **black box**, because we need to know its **exact form**.
- In this setting, one may be interested in answering the following questions:
 - Which predictors are associated with the response?
 - What is the relationship between the response and each predictor?
 - Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

Income Data

- As another example, a plot of **income** versus **years of education** for 30 individuals in the **Income** data set.
- The plot suggests that one might be able to predict **income** using **years of education**.



Income Data

```
url.in <- "https://www.statlearning.com/s/Income1.csv"
Income <- read.csv(url.in, h=T)
```

```
## Polynomial regression fit
par(mfrow = c(1,2))
plot(Income~Education, col=2, pch=19, xlab="Years of Education",
      ylab="Income", data=Income)
g <- lm(Income ~ poly(Education, 3), data=Income)
```

```
plot(Income~Education, col=2, pch=19, xlab="Years of Education",
      ylab="Income", data=Income)
lines(Income$Education, g$fit, col="darkblue", lwd=4,
      ylab="Income", xlab="Years of Education")
```

```
y <- Income$Income
mean((predict(g) - y)^2)
mean(residuals(g)^2)
```


Income Data

```
dist <- NULL
```

```
par(mfrow=c(3,4))
for (k in 1:12) {
  g <- lm(Income ~ poly(Education, k), data=Income)
  dist[k] <- mean(residuals(g)^2)
  plot(Income~Education, col=2, pch=19,
        xlab="Years of Education", ylab="Income",
        data=Income, main=paste("k =", k))
  lines(Income$Education,g$fit,col="darkblue",lwd=3,
        ylab="Income", xlab="Years of Education")
}
```

```
x11()
plot(dist, type="b", xlab="Degree of Polynomial",
      ylab="Mean squared distance")
dist
```

Parametric and Non-parametric Methods

- Observations we have are called the **training data** because we use these observations to train or teach our method.
- Our goal is to apply a **statistical learning** method to the **training data** in order to estimate the unknown function f such that

$$Y \approx \hat{f}(X)$$

for any observation (X, Y) .

- Most statistical learning methods for this task can be characterized as either **parametric** or **non-parametric**.
 - **Parametric methods**: make an assumption about the functional form or shape of f .
 - **Non-parametric methods**: do not make explicit assumptions about the functional form of f .

- For example, f is assumed to be linear so,

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon,$$

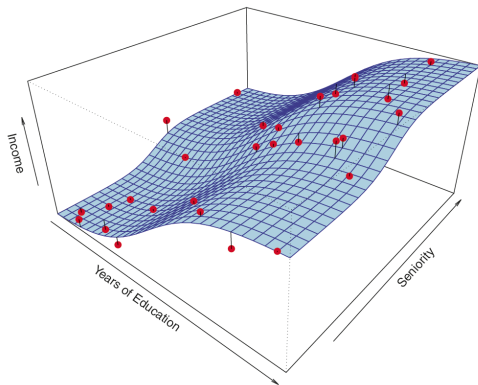
where we only need to estimate $p + 1$ coefficients.

- **Parametric methods** reduce the problem of estimating f down to one of estimating a set of parameters.
- The **disadvantage** is that the model we choose will usually not match the true unknown form of f .
- If the chosen model is too far from the true f , then our estimate will be poor.
- **Flexible** models can fit many different possible functional forms for $f \rightarrow$ **overfitting** problem.

Non-parametric Methods

- **Non-parametric methods** estimates f that gets as close to data points as possible without being too rough or wiggly.
- They have the potential to **accurately fit** a wider range of possible shapes for f since they do not assume a particular functional form for f .
- **No assumption** about the form of f is made, so there is no danger that estimate f is very different from the true f .
- Since the number of parameters is not small, a **very large number of observations** is required in order to obtain an accurate estimate for f .
- There are advantages and disadvantages to **parametric** and **non-parametric** methods for statistical learning.

Example

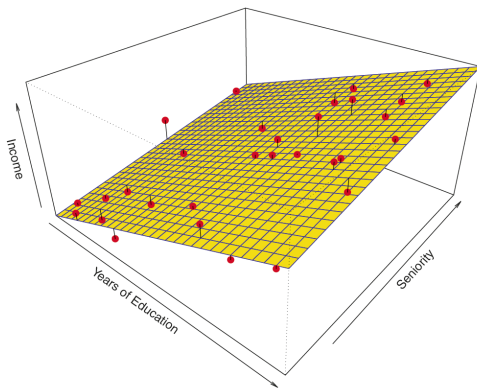


- Red points are simulated values for **income** from the model

$$\text{income} = f(\text{education}, \text{seniority}) + \epsilon$$

f is the blue surface.

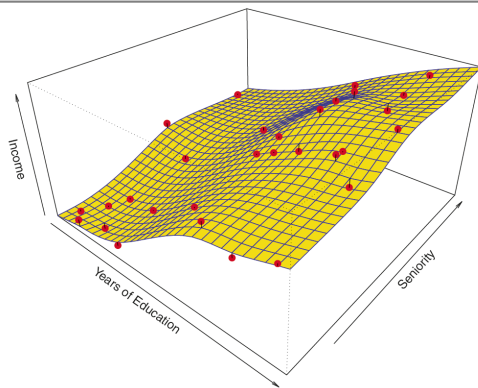
Linear Regression



- Linear regression model fit to the simulated data.

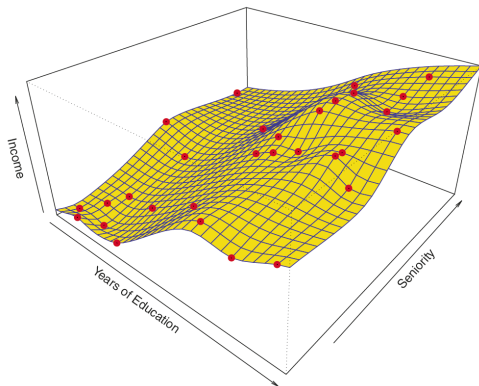
$$\hat{f}^L(\text{education, seniority}) = \hat{\beta}_0 + \hat{\beta}_1 \times \text{education} + \hat{\beta}_2 \times \text{seniority}$$

Thin-plate Spline



- More flexible regression model $\hat{f}^S(\text{education, seniority})$ fit to the simulated data.
- Here we use a technique called a **thin-plate spline** to fit a flexible surface.
- We can control the roughness of the fit.

Overfitting Spline



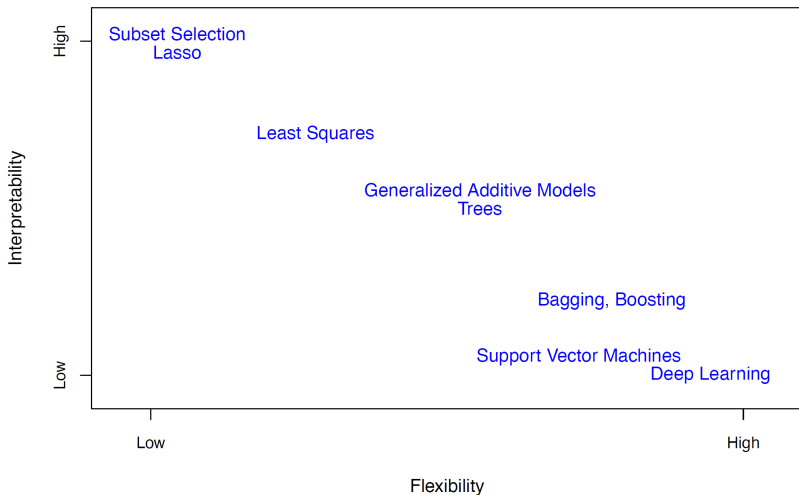
- Even more flexible spline regression model $\hat{f}^S(\text{education}, \text{seniority})$ fit to the simulated data.
- Here the fitted model makes **no errors** on the training data!
- This is known as a **overfitting** problem.

Flexibility and Interpretability

- Among many statistical learning methods, some are less flexible, or more restrictive.
- Linear regression is a relatively inflexible approach, because it can only generate linear functions.
- Thin plate splines are more flexible because they can generate a wider range of possible shapes to estimate f .
- If we are mainly interested in inference, then restrictive models are much more interpretable.
- Flexible approaches are difficult to understand how any individual predictor is associated with the response.
- If we are only interested in prediction, the most flexible model is preferable — often we can obtain more accurate predictions using a less flexible method.

Flexibility and Interpretability

- Trade-off between flexibility and interpretability.



Assessing Model Accuracy

- Why is it necessary to study so many different **statistical learning** approaches, rather than just a single **best** method?
- **No one method** dominates all others over all possible data sets

“There is no free lunch in statistics.”

- On a particular data set, one specific method may work best, but some other method may work better on a similar but different data set.
- It is an important task to decide for any given set of data **which method** produces the **best** results.
- Selecting **the best approach** can be one of the most challenging parts of performing statistical learning in practice.

Assessing Model Accuracy

- In order to evaluate the performance of a statistical learning method, we need some way to measure how well its **predictions** actually match the observed data.
 - Quantitative response: **mean squared error (MSE)**, ...
 - Qualitative response: **classification error rate**, ...
- Type of data sets
 - **Training set**: to fit statistical learning models
 - **Validation set**: to select the optimal tuning parameter
 - **Test set**: to select the best model
- We are interested in the accuracy of the predictions that we obtain when we apply our method to previously **unseen test data**.

Assessing Model Accuracy

- Suppose we fit a model $\hat{f}(x)$ to some **training data**

$$\text{tran} = \{x_i, y_i\} \quad \text{for } i \in \{1, \dots, n_1\}.$$

- We could compute the average squared prediction error over training data:

$$MSE_{\text{tran}} = \frac{1}{n_1} \sum_{i \in \text{tran}} \left[y_i - \hat{f}(x_i) \right]^2$$

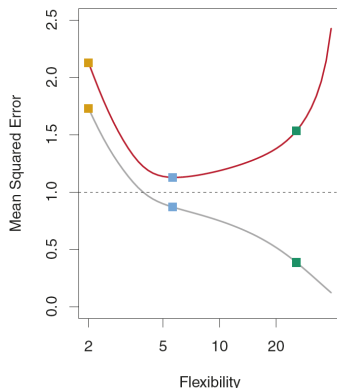
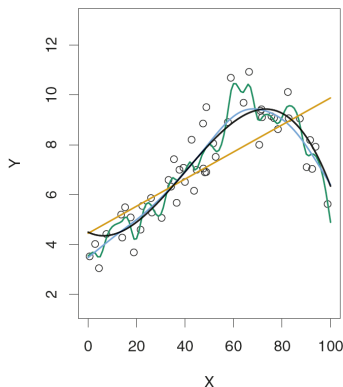
This may be biased toward more **overfit** models.

- Instead we should, if possible, compute it using fresh **test** data:

$$MSE_{\text{test}} = \frac{1}{n_2} \sum_{i \in \text{test}} \left[y_i - \hat{f}(x_i) \right]^2,$$

where $\text{test} = \{x_i, y_i\}$ for $i \in \{1, \dots, n_2\}$.

Simulation Example: Cubic Model



- In the left, black curve is truth. Orange, blue and green curves/squares correspond to fits of different flexibility.
- Red curve on right is MSE_{test} , grey curve is MSE_{tran} .

Cubic Model

```
set.seed(12345)
```

```
## Simulate x and y based on a known function
fun1 <- function(x) -(x-100)*(x-30)*(x+15)/13^4+6
x <- runif(50,0,100)
y <- fun1(x) + rnorm(50)
```

```
## Plot linear regression and splines
par(mfrow=c(1,2))
plot(x, y, xlab="X", ylab="Y", ylim=c(1,13))
```

```
plot(x, y, xlab="X", ylab="Y", ylim=c(1,13))
lines(sort(x), fun1(sort(x)), col=1, lwd=2)
abline(lm(y~x)$coef, col="orange", lwd=2)
lines(smooth.spline(x,y, df=5), col="blue", lwd=2)
lines(smooth.spline(x,y, df=23), col="green", lwd=2)
legend("topleft", lty=1, col=c(1, "orange", "blue", "green"),
      legend=c("True", "df = 1", "df = 5", "df =23"),lwd=2)
```

Cubic Model

```
set.seed(45678)
```

```
## Simulate training and test data (x, y)
tran.x <- runif(50,0,100)
test.x <- runif(50,0,100)
tran.y <- fun1(tran.x) + rnorm(50)
test.y <- fun1(test.x) + rnorm(50)
```

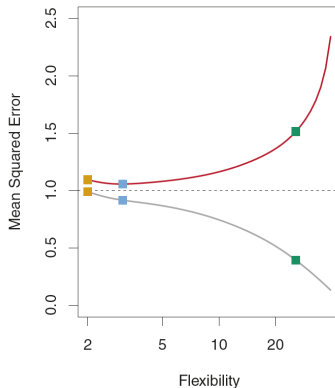
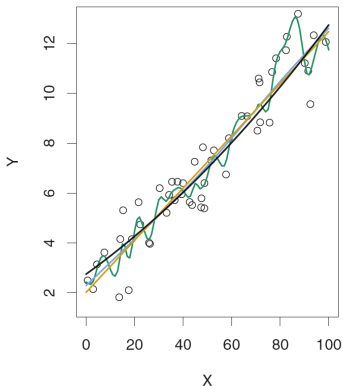
```
## Compute MSE along with different df
df <- 2:40
MSE <- matrix(0, length(df), 2)
for (i in 1:length(df)) {
  tran.fit <- smooth.spline(tran.x, tran.y, df=df[i])
  MSE[i,1] <- mean((tran.y - predict(tran.fit, tran.x)$y)^2)
  MSE[i,2] <- mean((test.y - predict(tran.fit, test.x)$y)^2)
}
```


Cubic Model

```
## Plot both test and training errors
matplot(df, MSE, type="l", col=c("gray", "red"),
        xlab="Flexibility", ylab="Mean Squared Error",
        lwd=2, lty=1, ylim=c(0,4))
abline(h=1, lty=2)
legend("top", lty=1, col=c("red", "gray"),lwd=2,
      legend=c("Test MSE", "Training MSE"))
abline(v=df[which.min(MSE[,1])], lty=3, col="gray")
abline(v=df[which.min(MSE[,2])], lty=3, col="red")
```

```
matplot(df, MSE, type="l", col=c("gray", "red"),
        xlab="Flexibility", ylab="Mean Squared Error",
        lwd=2, lty=1, ylim=c(0.5,2), xlim=c(1,10))
abline(h=1, lty=2)
abline(v=df[which.min(MSE[,2])], lty=3, col="red")
```

Simulation Example: Linear Model



- Here the truth is a smoother, so the smoother fit and linear model do really well.

Linear Model

```
set.seed(12345)
```

```
## Simulate x and y based on a known function  
fun2 <- function(x) x/10 +2  
x <- runif(50,0,100)  
y <- fun2(x) + rnorm(50)
```

```
## Plot linear regression and splines  
par(mfrow=c(1,2))  
plot(x, y, xlab="X", ylab="Y", ylim=c(1,13))
```

```
plot(x, y, xlab="X", ylab="Y", ylim=c(1,13))  
lines(sort(x), fun2(sort(x)), col=1, lwd=2)  
abline(lm(y~x)$coef, col="orange", lwd=2)  
lines(smooth.spline(x,y, df=5), col="blue", lwd=2)  
lines(smooth.spline(x,y, df=23), col="green", lwd=2)  
legend("topleft", lty=1, col=c(1, "orange", "blue", "green"),  
      legend=c("True", "df = 1", "df = 5", "df =23"),lwd=2)
```

Linear Model

```
set.seed(45678)
```

```
## Simulate training and test data (x, y)
tran.x <- runif(50,0,100)
test.x <- runif(50,0,100)
tran.y <- fun2(tran.x) + rnorm(50)
test.y <- fun2(test.x) + rnorm(50)
```

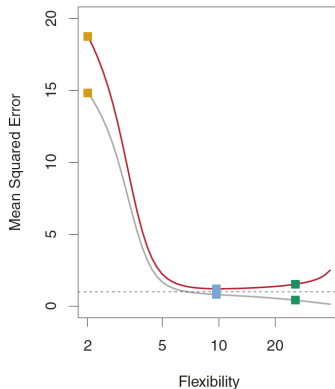
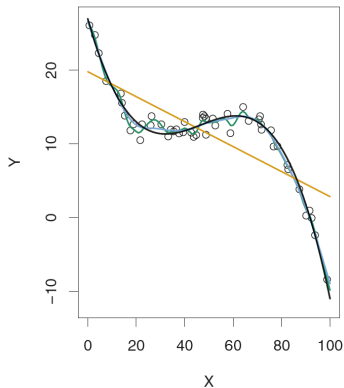
```
## Compute MSE along with different df
df <- 2:40
MSE <- matrix(0, length(df), 2)
for (i in 1:length(df)) {
  tran.fit <- smooth.spline(tran.x, tran.y, df=df[i])
  MSE[i,1] <- mean((tran.y - predict(tran.fit, tran.x)$y)^2)
  MSE[i,2] <- mean((test.y - predict(tran.fit, test.x)$y)^2)
}
```

Linear Model

```
## Plot both test and training errors
matplot(df, MSE, type="l", col=c("gray", "red"),
        xlab="Flexibility", ylab="Mean Squared Error",
        lwd=2, lty=1)
abline(h=1, lty=2)
legend("top", lty=1, col=c("red", "gray"), lwd=2,
      legend=c("Test MSE", "Training MSE"))
abline(v=df[which.min(MSE[,1])], lty=3, col="gray")
abline(v=df[which.min(MSE[,2])], lty=3, col="red")
```

```
matplot(df, MSE, type="l", col=c("gray", "red"),
        xlab="Flexibility", ylab="Mean Squared Error",
        lwd=2, lty=1, ylim=c(0,1.5), xlim=c(1,10))
abline(h=1, lty=2)
abline(v=df[which.min(MSE[,2])], lty=3, col="red")
```

Simulation Example: Nonlinear Model



- Here the truth is wiggly and the noise is low, so the more flexible fits do the best.

Nonlinear Model

```
set.seed(12345)
```

```
## Simulate x and y based on a known function  
fun3 <- function(x) -(x-80)*(x-45)*(x-25)/15^3+10  
x <- runif(50,0,100)  
y <- fun3(x) + rnorm(50)
```

```
## Plot linear regression and splines  
par(mfrow=c(1,2))  
plot(x, y, xlab="X", ylab="Y")
```

```
plot(x, y, xlab="X", ylab="Y")  
lines(sort(x), fun3(sort(x)), col=1, lwd=2)  
abline(lm(y~x)$coef, col="orange", lwd=2)  
lines(smooth.spline(x,y, df=5), col="blue", lwd=2)  
lines(smooth.spline(x,y, df=23), col="green", lwd=2)  
legend("topright", lty=1, col=c(1, "orange", "blue", "green"),  
      legend=c("True", "df = 1", "df = 5", "df =23"),lwd=2)
```

Nonlinear Model

```
set.seed(45678)
```

```
## Simulate training and test data (x, y)
tran.x <- runif(50,0,100)
test.x <- runif(50,0,100)
tran.y <- fun3(tran.x) + rnorm(50)
test.y <- fun3(test.x) + rnorm(50)
```

```
## Compute MSE along with different df
df <- 2:40
MSE <- matrix(0, length(df), 2)
for (i in 1:length(df)) {
  tran.fit <- smooth.spline(tran.x, tran.y, df=df[i])
  MSE[i,1] <- mean((tran.y - predict(tran.fit, tran.x)$y)^2)
  MSE[i,2] <- mean((test.y - predict(tran.fit, test.x)$y)^2)
}
```


Nonlinear Model

```
## Plot both test and training errors
matplot(df, MSE, type="l", col=c("gray", "red"),
        xlab="Flexibility", ylab="Mean Squared Error",
        lwd=2, lty=1)
abline(h=1, lty=2)
legend("top", lty=1, col=c("red", "gray"), lwd=2,
      legend=c("Test MSE", "Training MSE"))
abline(v=df[which.min(MSE[,1])], lty=3, col="gray")
abline(v=df[which.min(MSE[,2])], lty=3, col="red")
```

```
matplot(df, MSE, type="l", col=c("gray", "red"),
        xlab="Flexibility", ylab="Mean Squared Error",
        lwd=2, lty=1, ylim=c(0,3), xlim=c(4,20))
abline(h=1, lty=2)
abline(v=df[which.min(MSE[,2])], lty=3, col="red")
```

Bias-Variance Trade-off

- Suppose that we fit a model $\hat{f}(X)$ to **training** data, and let (x_0, y_0) be a **test** observation drawn from the population.
- If the true model is

$$Y = f(X) + \epsilon$$

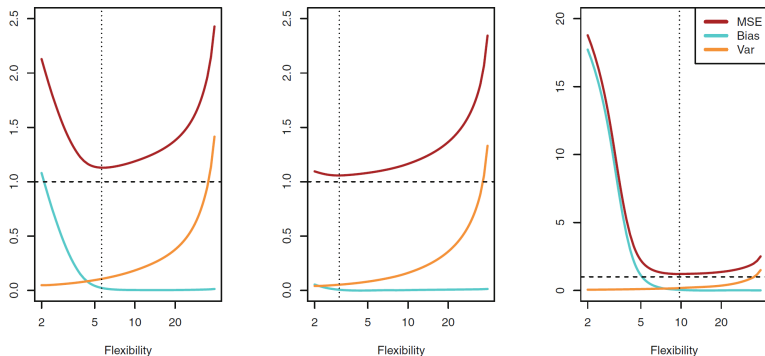
with $f(x) = E(Y|X = x)$, then

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \text{Var}\left(\hat{f}(x_0)\right) + \left[\text{Bias}(\hat{f}(x_0))\right]^2 + \text{Var}(\epsilon)$$

where $\text{Bias}(\hat{f}(x_0)) = E[\hat{f}(x_0)] - f(x_0)$.

- Typically as the **flexibility** of \hat{f} increases, its **variance** increases, and its **bias** decreases. So choosing the flexibility based on average test error amounts to a **bias-variance trade-off**.

Bias-Variance Trade-off



- Squared bias (blue curve), variance (orange curve), $\text{Var}(\epsilon)$ (dashed line), and test MSE (red curve).

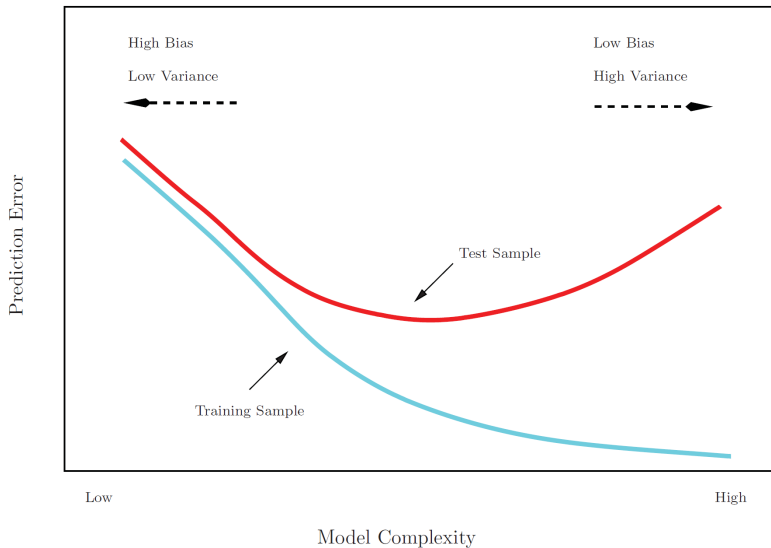
Bias-Variance Trade-off

- Good test set performance of a statistical learning method requires **low variance** as well as **low squared bias**.
- **Trade-off**: it is easy to obtain a method with extremely low bias but high variance or a method with very low variance but high bias.
- In a real-life situation in which f is unobserved, it is generally **not possible** to explicitly compute the test MSE, bias, or variance for a statistical learning method.
- We should always keep the **bias-variance trade-off** in mind to select the best statistical learning method.

Training Error versus Test error

- The **test error** is the average error that results from using a statistical learning method to predict the response on a new observation, one that was not used in training the method.
- In contrast, the **training error** can be easily calculated by applying the statistical learning method to the observations used in its training data.
- But the training error rate often is quite different from the test error rate, and in particular the former can **dramatically underestimate** the latter.
- In absence of a large designated test set to directly estimate the **test error rate**, a number of techniques can be used to estimate this quantity using available training data.

Training-set versus Test-set Performance



More on Prediction-error Estimates

- **Best solution**: a large designated test set. Often not available.
- Some methods make a **mathematical adjustment** to the training error rate in order to estimate the test error rate. These include the C_p statistic, *AIC* and *BIC*.
- Here we instead consider a class of methods that estimate the test error by **holding out** a subset of the training observations from the fitting process, and then applying the statistical learning method to those **held out** observations.
 - Validation set approach
 - K -fold cross-validation
 - Leave-one-out cross-validation (LOOCV)

Validation Set Approach

- Here we randomly divide the available set of samples into two parts: a **training set** and a **validation** or **hold-out set**.
- The model is fit on the **training set**, and the fitted model is used to predict the responses for the observations in the **validation set**.
- The resulting **validation set error** provides an estimate of the **test error**. This is typically assessed using Mean Squared Error (MSE) in the case of a **quantitative** response and misclassification rate in the case of a **qualitative** (or **binary**) response.

Validation Set Approach

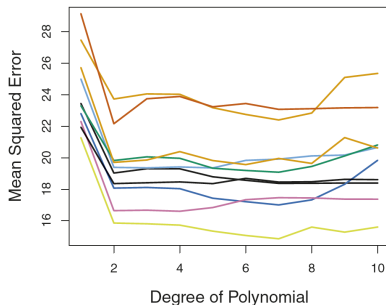
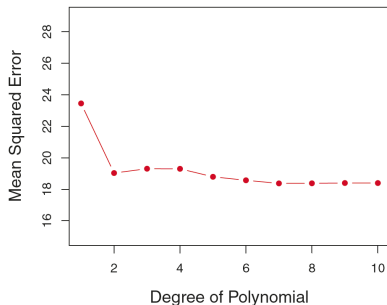
- A random splitting into two halves: left part is training set, right part is validation set.



- In next example, we want to compare linear vs. higher-order polynomial terms in a linear regression, using the validation set approach.

Example: Auto Data

- We randomly split the 392 observations into two sets, a **training set** containing 196 of the data points, and a **validation set** containing the remaining 196 observations.



Left panel shows single split and right panel shows multiple splits

Auto Data

```
library(ISLR)
data(Auto)
str(Auto)
summary(Auto)
```

```
mpg <- Auto$mpg
horsepower <- Auto$horsepower
```

```
dg <- 1:9
u <- order(horsepower)
```

```
par(mfrow=c(3,3))
for (k in 1:length(dg)) {
  g <- lm(mpg ~ poly(horsepower, dg[k]))
  plot(mpg~horsepower, col=2, pch=20, xlab="Horsepower",
       ylab="mpg", main=paste("dg =", dg[k]))
  lines(horsepower[u], g$fit[u], col="darkblue", lwd=3)
}
```

Auto Data: Single Split

```
set.seed(1)
n <- nrow(Auto)
```

```
## training set
tran <- sample(n, n/2)
```

```
MSE <- NULL
for (k in 1:length(dg)) {
  g <- lm(mpg ~ poly(horsepower, dg[k]), subset=tran)
  MSE[k] <- mean((mpg - predict(g, Auto))[-tran]^2)
}
```

```
par(mfrow=c(1,3))
plot(dg, MSE, type="b", col=2, xlab="Degree of Polynomial",
     ylab="Mean Squared Error", ylim=c(15,30), lwd=2, pch=19)
abline(v=which.min(MSE), lty=2)
```

Auto Data: Multiple Splits

```
K <- 10
MSE <- matrix(0, length(dg), K)
```

```
for (i in 1:K) {
  tran <- sample(392, 196)
  for (k in 1:length(dg)) {
    g <- lm(mpg ~ poly(horsepower, dg[k]), subset=tran)
    MSE[k, i] <- mean((mpg - predict(g, Auto))[-tran]^2)
  }
}
matplot(dg, MSE, type="l", xlab="Degree of Polynomial", lty=1,
        ylab="Mean Squared Error", col=1:10, ylim=c(15,30))
```

```
avg <- apply(MSE, 1, mean)
plot(dg, avg, type="b", col=2, xlab="Degree of Polynomial",
     ylab="Mean Squared Error", ylim=c(15,30), lwd=2, pch=19)
abline(v=which.min(avg), lty=2)
```

Drawbacks of Validation Set Approach

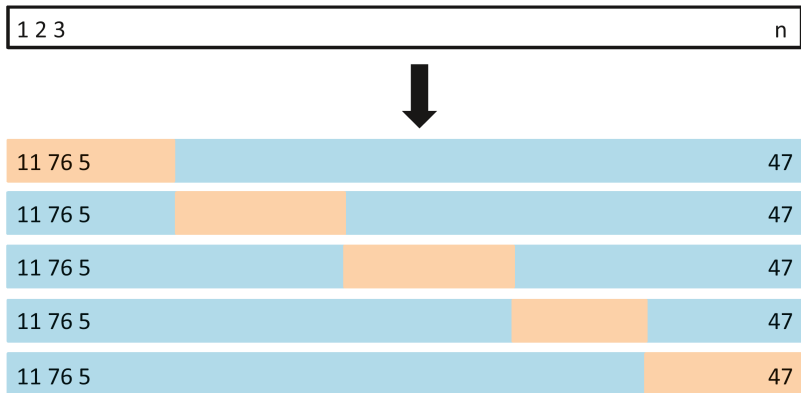
- The validation estimate of the **test error** can be highly variable, depending on precisely which observations are included in the **training set** and which observations are included in the **validation set**.
- In the validation approach, only a subset of the observations — those that are included in the **training set** rather than in the **validation set** are used to fit the model.
- This suggests that the **validation set error** may tend to incorrectly estimate the test error for the model fit on the entire data set.

K -fold Cross-validation

- K -fold Cross-validation is widely used approach for estimating test error.
- Estimates can be used to select the best model, and to give an idea of the test error of the final chosen model.
- Idea is to randomly divide the data into K equal-sized parts. We leave out part k , fit the model to the other $K - 1$ parts (combined training set), and then obtain predictions for the left-out k th part (test set).
- This is done in turn for each part $k = 1, 2, \dots, K$, and then the results are combined.

The Validation Process

- Divide data into K roughly equal-sized parts ($K = 5$ here)



Cross-validation Error for Quantitative Outcomes

- Suppose that y_i is a quantitative value.
- Let the K parts be C_1, C_2, \dots, C_K , where C_k denotes the indices of the observations in part k . There are n_k observations in part k .
- Compute Cross-validation Error (CVE)

$$\text{CVE} = \frac{1}{n} \sum_{k=1}^K n_k \text{MSE}_k$$

where

$$\text{MSE}_k = \frac{1}{n_k} \sum_{i \in C_k} (y_i - \hat{y}_i^{[-k]})^2$$

and $\hat{y}_i^{[-k]}$ is the fit for the observation i , obtained from the data with part k removed.

Cross-validation Error for Binary Outcomes

- Suppose that $y_i = 1$ for cases and $y_i = 0$ for controls.
- Cross-validation Error (CVE) can be computed from
 - Deviance

$$\text{CVE} = \frac{1}{n} \sum_{k=1}^K \sum_{i \in C_k} -2 \left(y_i \log \hat{p}_i^{[-k]} + (1 - y_i) \log(1 - \hat{p}_i^{[-k]}) \right),$$

where $\hat{p}_i^{[-k]}$ is the probability of $y_i = 1$ for the observation i , obtained from the data with part k removed.

- Classification error

$$\text{CVE} = \frac{1}{n} \sum_{k=1}^K \sum_{i \in C_k} I \left(y_i - \hat{y}_i^{[-k]} \right),$$

where $\hat{y}_i^{[-k]}$ is the fit for the observation i , obtained from the data with part k removed.

Special Case of LOOCV

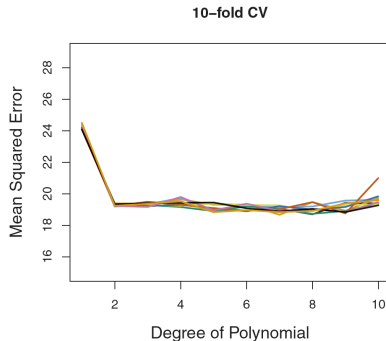
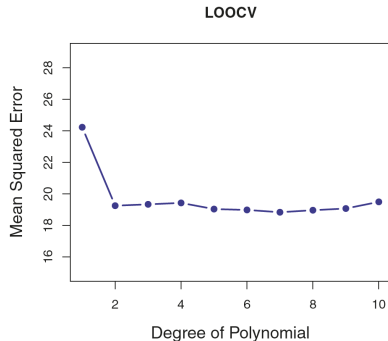
- Setting $K = n$ yields **leave-one out cross-validation (LOOCV)**.
- With **least-squares** of linear or polynomial regression, an amazing shortcut makes the cost of LOOCV the same as that of a single model fit. The following formula holds:

$$\text{CV error} = \frac{1}{n} \sum_{i=1}^n \left(\frac{y_i - \hat{y}_i}{1 - h_i} \right)^2$$

where \hat{y}_i is the i th fitted value from the original least squares fit, and h_i is the leverage. This is like the ordinary MSE, except the i th residual is divided by $1 - h_i$.

- LOOCV sometimes useful, but typically doesn't **shake up** the data enough. The estimates from each fold are highly correlated and hence their average can have high variance. A better choice is $K = 5$ or 10 .

Example: Auto Data



Left panel shows LOOCV and right panel shows 10-fold CV

Auto Data : LOOCV

```
n <- nrow(Auto)
dg <- 1:9
MSE <- matrix(0, n, length(dg))
```

```
for (i in 1:n) {
  for (k in 1:length(dg)) {
    g <- lm(mpg ~ poly(horsepower, k), subset=(1:n)[-i])
    MSE[i, k] <- mean((mpg - predict(g, Auto))[i]^2)
  }
}
aMSE <- apply(MSE, 2, mean)
```

```
par(mfrow=c(1, 2))
plot(dg, aMSE, type="b", col="darkblue",
     xlab="Degree of Polynomial", ylab="Mean Squared Error",
     ylim=c(18,25), lwd=2, pch=19)
abline(v=which.min(aMSE), lty=2)
```

Auto Data : LOOCV

```
ncv <- NULL
for (k in 1:length(dg)) {
  g <- lm(mpg ~ poly(horsepower, k))
  ncv[k] <- mean((g$res/(1-influence(g)$hat))^2)
}
lines(dg, ncv, col=2, lty=2, lwd=2)
```

```
K <- 10      ## 10-fold cross validation
MSE <- matrix(0, n, length(dg))
```

```
set.seed(54321)
u <- sample(rep(seq(K), length=n))
table(u)
```

Auto Data: K -fold CV

```
for (k in 1:K) {  
  tran <- which(u!=k)  
  test <- which(u==k)  
  for (i in 1:length(dg)) {  
    g <- lm(mpg ~ poly(horsepower, i), subset=tran)  
    MSE[test, i] <- (mpg - predict(g, Auto))[test]^2  
  }  
}  
CVE <- apply(MSE, 2, mean)
```

```
plot(dg, CVE, type="b", col="darkblue",  
     xlab="Degree of Polynomial", ylab="Mean Squared Error",  
     ylim=c(18,25), lwd=2, pch=19)  
abline(v=which.min(CVE), lty=2)
```

```
N <- 9      ## Number of K-fold CV replications  
KCV <- matrix(0, length(dg), N)
```

Auto Data: K -fold CV

```
set.seed(1234)
for (j in 1:N) {
  MSE <- matrix(0, n, length(dg))
  u <- sample(rep(seq(K), length=n))
  for (k in 1:K) {
    tran <- which(u!=k)
    test <- which(u==k)
    for (i in 1:length(dg)) {
      g <- lm(mpg ~ poly(horsepower, i), subset=tran)
      MSE[test, i] <- (mpg - predict(g, Auto))[test]^2
    }
  }
  KCV[,j] <- apply(MSE, 2, mean)
}
```

```
dev.off()
matplot(dg, KCV, type="l", xlab="Degree of Polynomial", lty=1,
        ylab="MSE", ylim=c(18,25), main="10-fold CV")
apply(KCV, 2, which.min)
```


R Package 'boot': Cross Validation

```
library(boot)
set.seed(101010)
```

```
## Leave-one-out CV
MSE <- NULL
for (i in 1:length(dg)) {
  glm.fit <- glm(mpg ~ poly(horsepower ,i))
  MSE[i] <- cv.glm(Auto, glm.fit)$delta[1]
}
plot(dg, MSE, type="b", col="darkblue", ylim=c(15,29),
      xlab="Degree of Polynomial", ylab="MSE", lwd=2, pch=19)
```

```
## K-fold cross validation
K <- 10
KCV <- NULL
for (i in 1:length(dg)) {
  glm.fit <- glm(mpg ~ poly(horsepower ,i))
  KCV[i] <- cv.glm(Auto, glm.fit, K=K)$delta[1]
}
lines(dg, KCV, col=2, lwd=2, type="b", pch=19, lty=2)
```

Cubic Model

```
set.seed(12345)
```

```
## Simulate x and y based on a known function
fun1 <- function(x) -(x-100)*(x-30)*(x+15)/13^4+6
x <- runif(50,0,100)
y <- fun1(x) + rnorm(50)
```

```
## Plot linear regression and splines
par(mfrow=c(1,2))
plot(x, y, xlab="X", ylab="Y", ylim=c(1,13))
```

```
plot(x, y, xlab="X", ylab="Y", ylim=c(1,13))
lines(sort(x), fun1(sort(x)), col=1, lwd=2)
abline(lm(y~x)$coef, col="orange", lwd=2)
lines(smooth.spline(x,y, df=5), col="blue", lwd=2)
lines(smooth.spline(x,y, df=23), col="green", lwd=2)
legend("topleft", lty=1, col=c(1, "orange", "blue", "green"),
      legend=c("True", "df = 1", "df = 5", "df =23"),lwd=2)
```

Cross-validation for Cubic Model

```
set.seed(45678)
x <- runif(50,0,100)
y <- fun1(x) + rnorm(50)
```

```
K <- 5
df <- 2:40
MSE <- matrix(0, length(x), length(df))
u <- sample(rep(seq(K), length=length(x)))
for (k in 1:K) {
  tr <- which(u!=k)
  te <- which(u==k)
  for (j in 1:length(df)) {
    fit <- smooth.spline(x[tr], y[tr], df=df[j])
    MSE[te, j] <- y[te] - predict(fit, x[te])$y
  }
}
CVE <- apply(MSE^2, 2, mean)
data.frame(Df=df, CVE=CVE)
```

Linear Model

```
set.seed(12345)
```

```
## Simulate x and y based on a known function  
fun2 <- function(x) x/10 +2  
x <- runif(50,0,100)  
y <- fun2(x) + rnorm(50)
```

```
## Plot linear regression and splines  
par(mfrow=c(1,2))  
plot(x, y, xlab="X", ylab="Y", ylim=c(1,13))
```

```
plot(x, y, xlab="X", ylab="Y", ylim=c(1,13))  
lines(sort(x), fun2(sort(x)), col=1, lwd=2)  
abline(lm(y~x)$coef, col="orange", lwd=2)  
lines(smooth.spline(x,y, df=5), col="blue", lwd=2)  
lines(smooth.spline(x,y, df=23), col="green", lwd=2)  
legend("topleft", lty=1, col=c(1, "orange", "blue", "green"),  
      legend=c("True", "df = 1", "df = 5", "df =23"),lwd=2)
```

Cross-validation for Linear Model

```
set.seed(45678)
x <- runif(50,0,100)
y <- fun2(x) + rnorm(50)
```

```
K <- 5
df <- 2:40
MSE <- matrix(0, length(x), length(df))
u <- sample(rep(seq(K), length=length(x)))
for (k in 1:K) {
  tr <- which(u!=k)
  te <- which(u==k)
  for (j in 1:length(df)) {
    fit <- smooth.spline(x[tr], y[tr], df=df[j])
    MSE[te, j] <- y[te] - predict(fit, x[te])$y
  }
}
CVE <- apply(MSE^2, 2, mean)
data.frame(Df=df, CVE=CVE)
```

Nonlinear Model

```
set.seed(12345)
```

```
## Simulate x and y based on a known function  
fun3 <- function(x) -(x-80)*(x-45)*(x-25)/15^3+10  
x <- runif(50,0,100)  
y <- fun3(x) + rnorm(50)
```

```
## Plot linear regression and splines  
par(mfrow=c(1,2))  
plot(x, y, xlab="X", ylab="Y")
```

```
plot(x, y, xlab="X", ylab="Y")  
lines(sort(x), fun3(sort(x)), col=1, lwd=2)  
abline(lm(y~x)$coef, col="orange", lwd=2)  
lines(smooth.spline(x,y, df=5), col="blue", lwd=2)  
lines(smooth.spline(x,y, df=23), col="green", lwd=2)  
legend("topright", lty=1, col=c(1, "orange", "blue", "green"),  
      legend=c("True", "df = 1", "df = 5", "df =23"),lwd=2)
```

Cross-validation for Nonlinear Model

```
set.seed(45678)
x <- runif(50,0,100)
y <- fun3(x) + rnorm(50)
```

```
K <- 5
df <- 2:40
MSE <- matrix(0, length(x), length(df))
u <- sample(rep(seq(K), length=length(x)))
for (k in 1:K) {
  tr <- which(u!=k)
  te <- which(u==k)
  for (j in 1:length(df)) {
    fit <- smooth.spline(x[tr], y[tr], df=df[j])
    MSE[te, j] <- y[te] - predict(fit, x[te])$y
  }
}
CVE <- apply(MSE^2, 2, mean)
data.frame(Df=df, CVE=CVE)
```