Cross-Validation

Canhong Wen

Agenda

Suppose we have several different models for a particular data set. How should we choose the best one? Naturally, we would want to select the best performing model in order to choose the best. What is performance? How to estimate performance? Thinking about these ideas, we'll consider the following:

- Model assessment and selection
- Prediction error
- Cross-validation
- Smoothing example

Part I

Training and testing errors

Statistical (regression) models

- You have some data $X_1, \ldots, X_p, Y$:
  - the variables $X_1, \ldots, X_p$ are called predictors,
  - Y is called a response.
- You're interested in the relationship that governs them. So you assume that $Y|X_1, \ldots, X_p \sim P_\theta$, where θ represents some unknown parameters.
- This is called **regression model** for Y given $X_1, \ldots, X_p$.
- **Goal** is to estimate parameters. Why?
  - To assess model validity, predictor importance (**inference**)
  - To predict future Y's from future $X_1, \ldots, X_p$'s (**prediction**)

Linear regression models

The linear model is arguably the **most widely used** statistical model, has a place in nearly every application domain of statistics

Given response Y and predictors $X_1, \ldots, X_p$, in a **linear regression model**, we assume:

$$Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_p X_p + \epsilon, \quad \text{where } \epsilon \sim N(0, \sigma^2)$$

Goal is to estimate parameters $\beta_0, \beta_1, \ldots, \beta_p$. Why?

- To assess whether the linear model is true, which predictors are important (**inference**)
- To simply predict future Y's from future $X_1, \ldots, X_p$'s (**prediction**)

Test error

Suppose we have **training data** $X_{i1}, \ldots, X_{ip}, Y_i$, $i = 1, \ldots, n$ used to estimate regression coefficients $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_p$

Given new $X_1^*, \ldots, X_p^*$ and asked to predict the associated Y^*. From the estimated linear model, prediction is: $\hat{Y}^* = \hat{\beta}_0 + \hat{\beta}_1 X_1^* + \ldots + \hat{\beta}_p X_p^*$. We define the **test error**, also called **prediction error**, by

$$\mathbb{E}(Y^* - \hat{Y}^*)^2$$

where the expectation is over every random: training data, $X_{i1}, \ldots, X_{ip}, Y_i$, $i = 1, \ldots, n$ and test data, $X_1^*, \ldots, X_p^*, Y^*$

This was explained for a linear model, but the same definition of test error **holds in general**

Estimating test error

Often, we want an accurate **estimate of the test error** of our method (e.g., linear regression). Why? Two main purposes:

- **Predictive assessment:** get an absolute understanding of the magnitude of errors we should expect in making future predictions
- **Model/method selection:** choose among different models/methods, attempting to minimize test error. Model selection is ubiquitous. Where do we use model selection?
  - Linear regression (variable selection)
  - Smoothing (smoothing parameter selection)
  - Kernel density estimation (bandwidth selection)
  - ...

Training error

Suppose, as an estimate the test error of our method, we take the observed **training error**

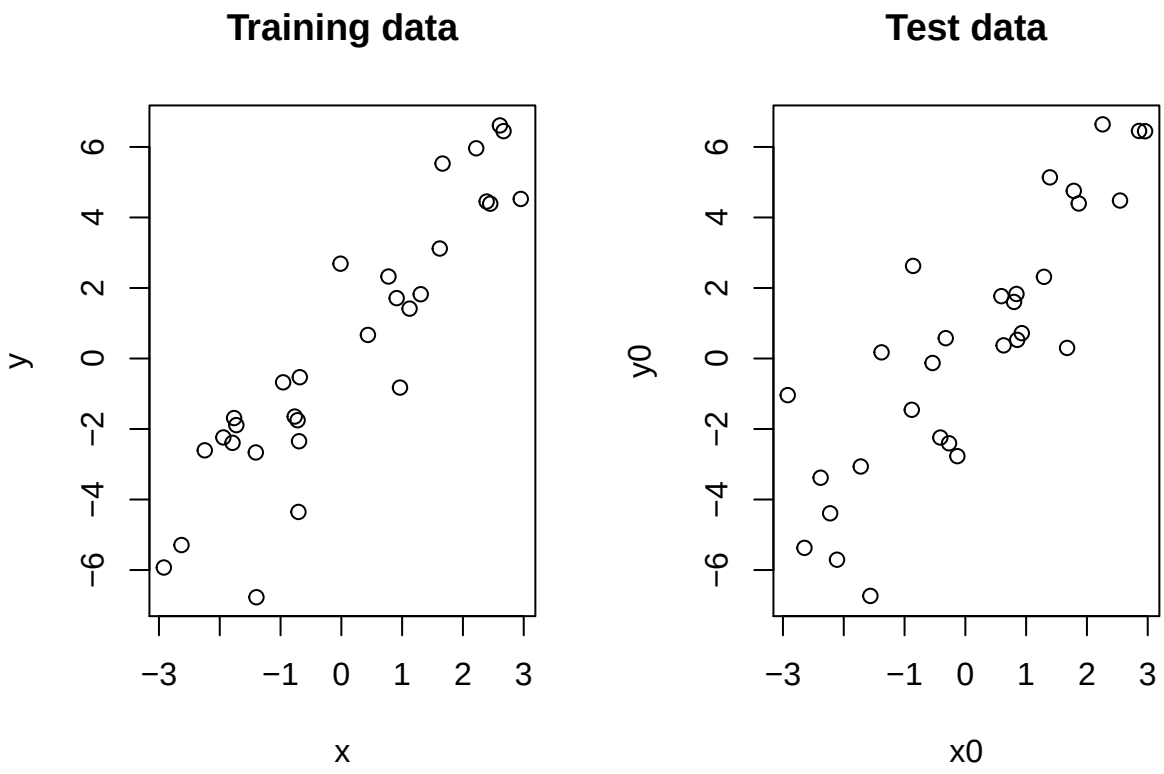
$$\frac{1}{n} \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

What's wrong with this? Generally **too optimistic** as an estimate of the test error—after all, the parameters $\hat{\beta}_0, \hat{\beta}_1, , \ldots, \hat{\beta}_p$ were estimated to make $\hat{Y}_i$ close to Y_i, $i = 1, \ldots, n$, in the first place!

Also, importantly, the more **complex/adaptive** the method, the more optimistic its training error is as an estimate of test error

Examples

```
set.seed(1)
n = 30
x = sort(runif(n, -3, 3))
y = 2*x + 2*rnorm(n)
x0 = sort(runif(n, -3, 3))
y0 = 2*x0 + 2*rnorm(n)
```




```
# Training and test errors for a simple linear model
lm.1 = lm(y ~ x)
yhat.1 = predict(lm.1, data.frame(x=x))
train.err.1 = mean((y-yhat.1)^2)
y0hat.1 = predict(lm.1, data.frame(x=x0))
test.err.1 = mean((y0-y0hat.1)^2)
```

```
par(mfrow=c(1,2))
plot(x, y, xlim=xlim, ylim=ylim, main="Training data")
lines(x, yhat.1, col=2, lwd=2)
text(0, -6, label=paste("Training error:", round(train.err.1,3)))

plot(x0, y0, xlim=xlim, ylim=ylim, main="Test data")
lines(x0, y0hat.1, col=3, lwd=2)
text(0, -6, label=paste("Test error:", round(test.err.1,3)))
```


```
# Training and test errors for a 10th order polynomial regression
# (The problem is only exacerbated!)
lm.10 = lm(y ~ poly(x,10))
yhat.10 = predict(lm.10, data.frame(x=x))
train.err.10 = mean((y-yhat.10)^2)
y0hat.10 = predict(lm.10, data.frame(x=x0))
test.err.10 = mean((y0-y0hat.10)^2)
```

```
par(mfrow=c(1,2))
xx = seq(min(xlim), max(xlim), length=100)

plot(x, y, xlim=xlim, ylim=ylim, main="Training data")
lines(xx, predict(lm.10, data.frame(x=xx)), col=2, lwd=2)
text(0, -6, label=paste("Training error:", round(train.err.10,3)))

plot(x0, y0, xlim=xlim, ylim=ylim, main="Test data")
lines(xx, predict(lm.10, data.frame(x=xx)), col=3, lwd=2)
text(0, -6, label=paste("Test error:", round(test.err.10,3)))
```


```
fit <- loess(y ~ x, span = 0.1, control = loess.control(surface = 'direct'))
yhat = predict(fit, data.frame(x=x))
train.err = mean((y-yhat)^2)
y0hat = predict(fit, data.frame(x=x0))
test.err = mean((y0-y0hat)^2)
```

```
par(mfrow=c(1,2))

plot(x, y, xlim=xlim, ylim=ylim, main="Training data")
lines(fit, col=2, lwd=2)
text(0, -6, label=paste("Training error:", round(train.err,3)))

plot(x0, y0, xlim=xlim, ylim=ylim, main="Test data")
lines(fit, col=3, lwd=2)
text(0, -6, label=paste("Test error:", round(test.err,3)))
```

Part II

Sample-splitting and cross-validation

Sample-splitting

- Given a data set, how can we estimate test error? (Can't simply simulate more data for testing.) We know training error won't work
- A tried-and-true technique with an old history in statistics: **sample-splitting**
  - Split the data set into two parts
  - First part: train the model/method
  - Second part: make predictions
  - Evaluate observed test error

Examples

```
dat = read.table("data/xy.dat")
head(dat)
```

```
##           x         y
## 1 -2.908021 -7.298187
```

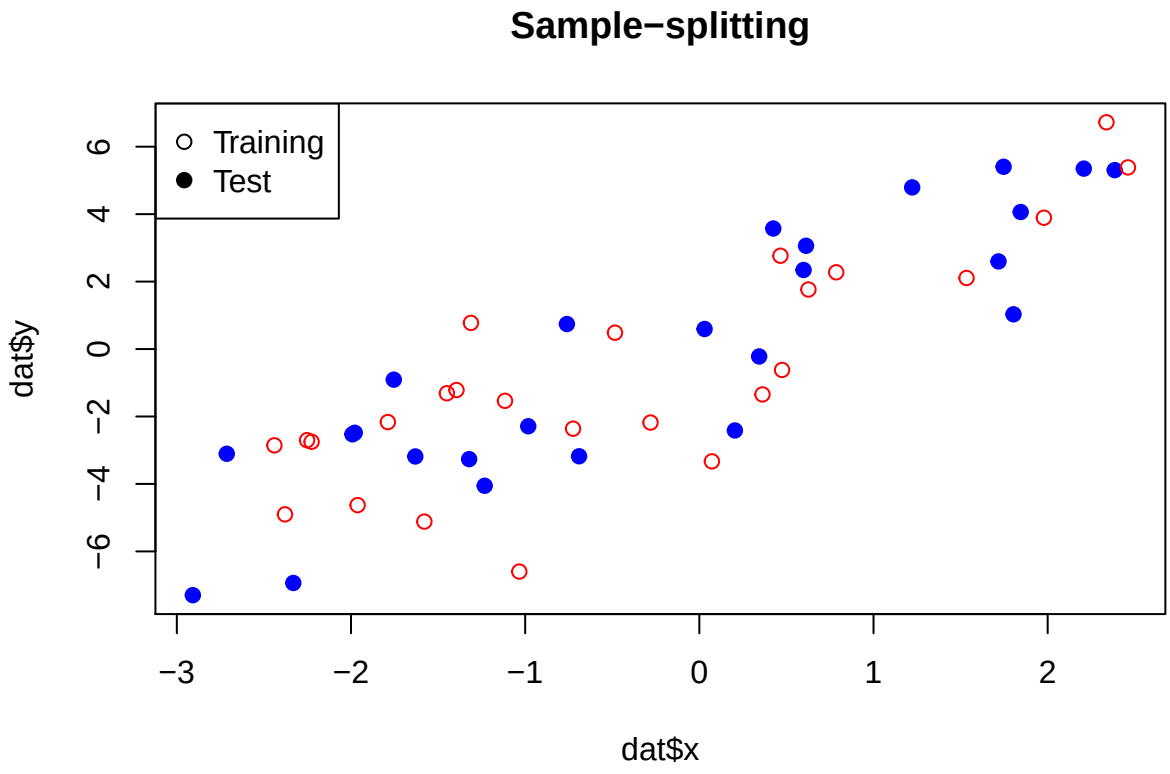
```
## 2 -2.713143 -3.105055
## 3 -2.439708 -2.855283
## 4 -2.379042 -4.902240
## 5 -2.331305 -6.936175
## 6 -2.252199 -2.703149
```

```
n = nrow(dat)
set.seed(0)
inds = sample(rep(1:2, length=n)) # Split data in half, randomly
table(inds)
```

```
## inds
##  1  2
## 25 25
```

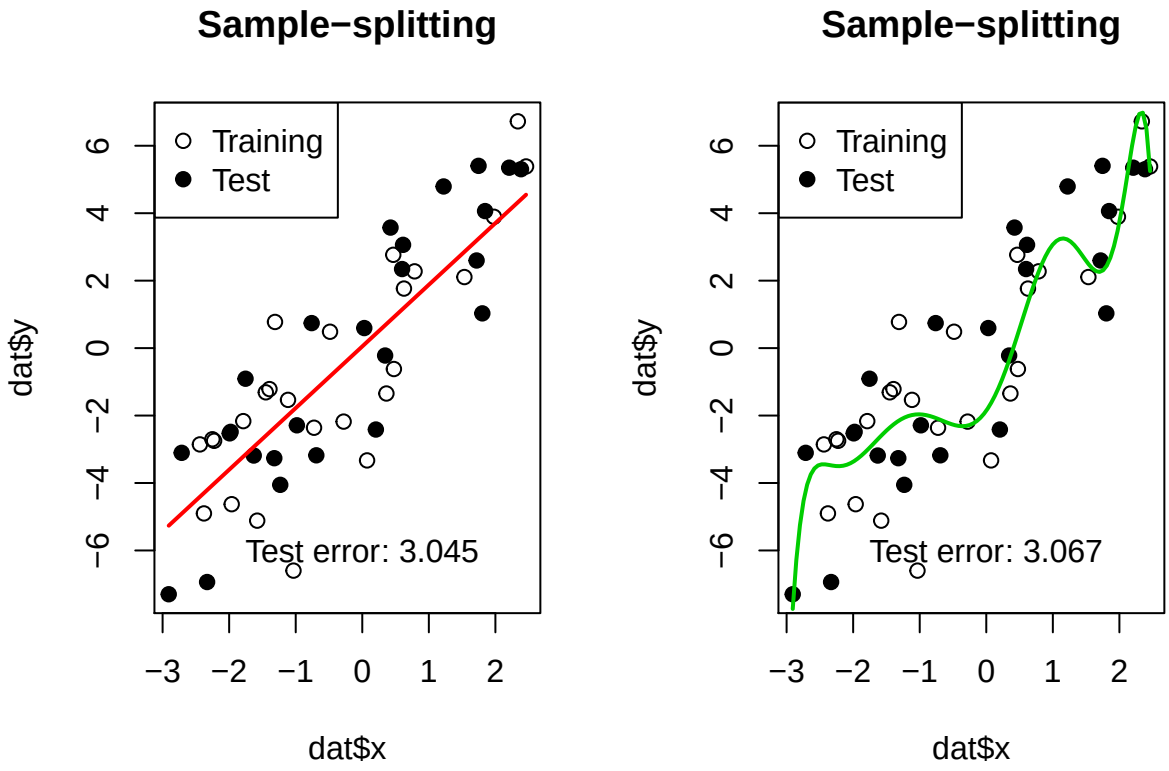
```
dat.tr = dat[inds==1,] # Training data
dat.te = dat[inds==2,] # Test data
```

```
plot(dat$x, dat$y, pch=c(21,19)[inds], col=c("red", "blue")[inds ], main="Sample-splitting")
legend("topleft", legend=c("Training","Test"), pch=c(21,19))
```



```r
# Train on the first half
lm.1 = lm(y ~ x, data=dat.tr)
lm.10 = lm(y ~ poly(x,10), data=dat.tr)

# Predict on the second half, evaluate test error
pred.1 = predict(lm.1, data.frame(x=dat.te$x))
pred.10 = predict(lm.10, data.frame(x=dat.te$x))

test.err.1 = mean((dat.te$y - pred.1)^2)
test.err.10 = mean((dat.te$y - pred.10)^2)

xx = seq(min(dat$x), max(dat$x), length=100)
```

```r
par(mfrow=c(1,2))

plot(dat$x, dat$y, pch=c(21,19)[inds], main="Sample-splitting")
lines(xx, predict(lm.1, data.frame(x=xx)), col=2, lwd=2)
legend("topleft", legend=c("Training","Test"), pch=c(21,19))
text(0, -6, label=paste("Test error:", round(test.err.1,3)))

plot(dat$x, dat$y, pch=c(21,19)[inds], main="Sample-splitting")
lines(xx, predict(lm.10, data.frame(x=xx)), col=3, lwd=2)
legend("topleft", legend=c("Training","Test"), pch=c(21,19))
text(0, -6, label=paste("Test error:", round(test.err.10,3)))
```



Cross-validation

Sample-splitting is simple, effective. But its it estimates the test error when the model/method is trained on **less data** (say, roughly half as much)

An improvement over sample splitting: **k-fold cross-validation**

- Split data into k parts or folds
- Use all but one fold to train your model/method
- Use the left out folds to make predictions
- Rotate around the roles of folds, k rounds total
- Compute squared error of all predictions, in the end

A common choice is $k = 5$ or $k = 10$ (sometimes $k = n$, called leave-one-out!)

For demonstration purposes, suppose $n = 6$ and we choose $k = 3$ parts

Data point	Part	Trained on	Prediction
Y_1	1	2,3	$\hat{Y}_1^{-(1)}$
Y_2	1	2,3	$\hat{Y}_2^{-(1)}$
Y_3	2	1,3	$\hat{Y}_3^{-(2)}$
Y_4	2	1,3	$\hat{Y}_4^{-(2)}$

Data point	Part	Trained on	Prediction
Y_5	3	1,2	$\hat{Y}_5^{-(3)}$
Y_6	3	1,2	$\hat{Y}_6^{-(3)}$

Notation: model trained on parts 2 and 3 in order to make predictions for part 1. So prediction $\hat{Y}_1^{-(1)}$ for Y_1 comes from model trained on all data except that in part 1. And so on

The **cross-validation estimate** of test error (also called the cross-validation error) is

$$\frac{1}{6}\Big((Y_1 - \hat{Y}_1^{-(1)})^2 + (Y_1 - \hat{Y}_2^{-(1)})^2 + (Y_1 - \hat{Y}_3^{-(2)})^2 + (Y_1 - \hat{Y}_4^{-(2)})^2 + (Y_1 - \hat{Y}_5^{-(3)})^2 + (Y_1 - \hat{Y}_6^{-(3)})^2\Big)$$

Leave-one-out cross validation

- A special case of cross-validation is known as **leave-one-out cross validation**
- Simply K-fold cross-validation where $K = n$

1. For the kth observation
   - Fit the model using the remaining $k - 1$ observations
   - Calculate the prediction error of the fitted model when predicting for the kth observation

Pseudo-code

The following is some pseudo-code for K-fold cross-validation.

```
K <- 5
n <- nrow(mydata)
cv.error <- vector(length = K)

# Randomly split data into K subsets
# For each observation gets a foldid  between 1 and K
foldid <-sample(rep(1:K, length = n))

# Repeat K times
for(i in 1:K) {
  # Fit using training set
  f.hat <- estimator(mydata[foldid != i, ])

  # Calculate prediction error on validation set
  cv.error[i] <- calc_error(f.hat, mydata[foldid == i, ])
}

cv.error.estimate <- mean(cv.error)
```

Examples

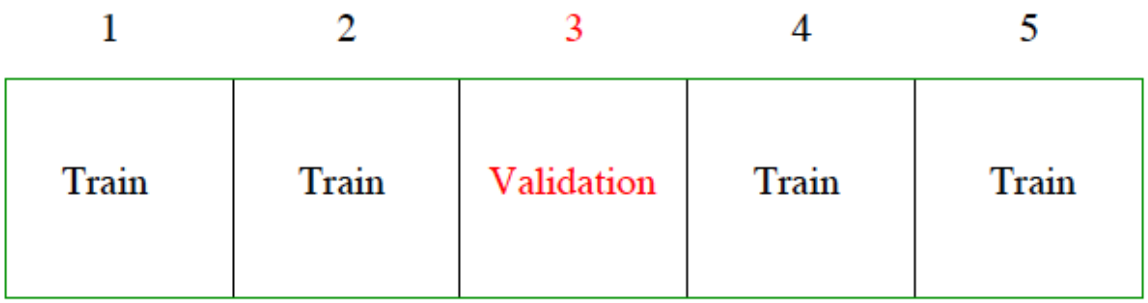


Figure 1:

```
# Split data in 5 parts, randomly
k = 5
set.seed(0)
inds = sample(rep(1:k, length=n))
head(inds, 10)
```

```
##  [1] 4 4 4 1 4 3 3 5 3 3
```

```
table(inds)
```

```
## inds
##  1  2  3  4  5
## 10 10 10 10 10
```

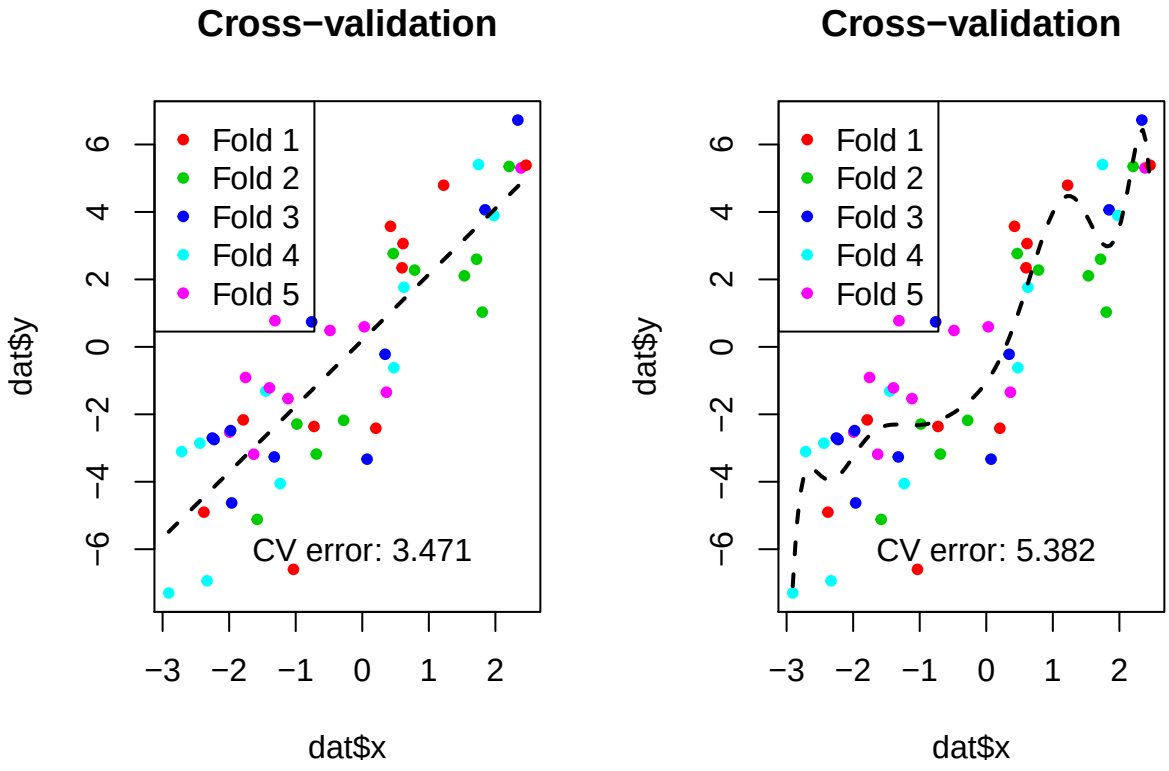
```
# Now run cross-validation: easiest with for loop, running over
# which part to leave out
pred.mat = matrix(0, n, 2) # Empty matrix to store predictions
for (i in 1:k) {
  cat(paste("Fold",i,"... "))

  dat.tr = dat[inds!=i,] # Training data
  dat.te = dat[inds==i,] # Test data

  # Train our models
  lm.1.minus.i = lm(y ~ x, data=dat.tr)
  lm.10.minus.i = lm(y ~ poly(x,10), data=dat.tr)

  # Record predictions
  pred.mat[inds==i,1] = predict(lm.1.minus.i, data.frame(x=dat.te$x))
  pred.mat[inds==i,2] = predict(lm.10.minus.i, data.frame(x=dat.te$x))
}
```

```
## Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
```

```
# Compute cross-validation error, one for each model
cv.errs = colMeans((pred.mat - dat$y)^2)

# Model trained on FULL data!
```

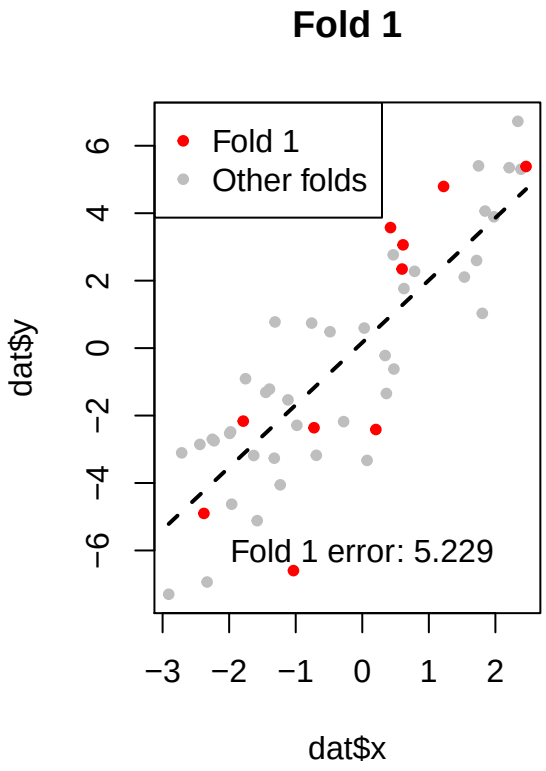
```r
lm.1 = lm(y ~ x, data=dat)
lm.10 = lm(y ~ poly(x,10), data=dat)
```

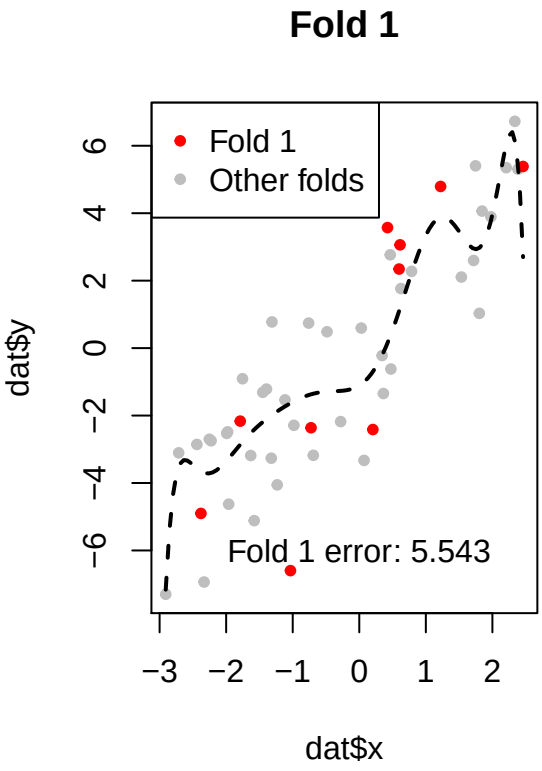
```r
par(mfrow=c(1,2))

plot(dat$x, dat$y, pch=20, col=inds+1, main="Cross-validation")
lines(xx, predict(lm.1, data.frame(x=xx)), lwd=2, lty=2)
legend("topleft", legend=paste("Fold",1:k), pch=20, col=2:(k+1))
text(0, -6, label=paste("CV error:", round(cv.errs[1],3)))

plot(dat$x, dat$y, pch=20, col=inds+1, main="Cross-validation")
lines(xx, predict(lm.10, data.frame(x=xx)), lwd=2, lty=2)
legend("topleft", legend=paste("Fold",1:k), pch=20, col=2:(k+1))
text(0, -6, label=paste("CV error:", round(cv.errs[2],3)))
```

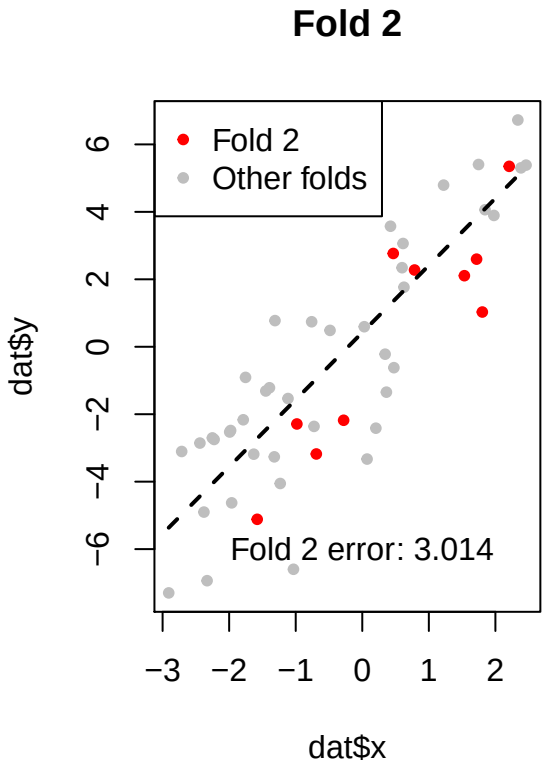


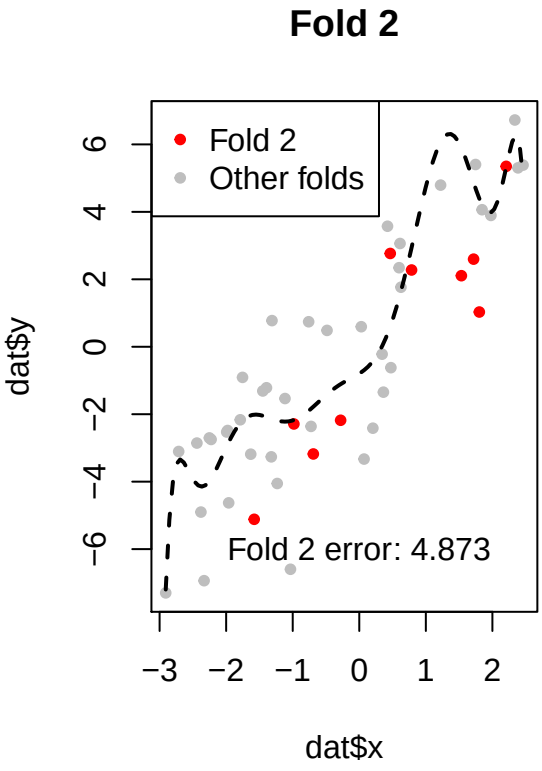

```r
# Now we visualize the different models trained, one for each CV fold
for (i in 1:k) {
  dat.tr = dat[inds!=i,] # Training data
  dat.te = dat[inds==i,] # Test data
```

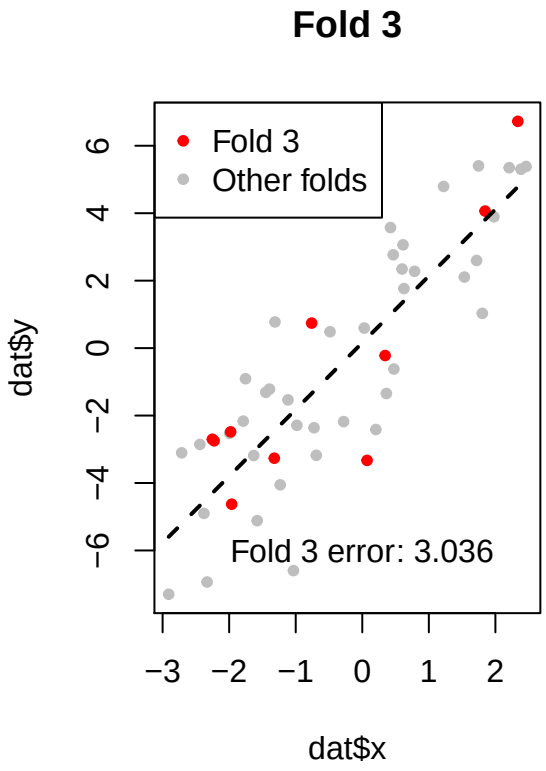
```
  # Train our models
  lm.1.minus.i = lm(y ~ x, data=dat.tr)
  lm.10.minus.i = lm(y ~ poly(x,10), data=dat.tr)

  # Plot fitted models
  par(mfrow=c(1,2)); cols = c("red","gray")
  plot(dat$x, dat$y, pch=20, col=cols[(inds!=i)+1], main=paste("Fold",i))
  lines(xx, predict(lm.1.minus.i, data.frame(x=xx)), lwd=2, lty=2)
  legend("topleft", legend=c(paste("Fold",i),"Other folds"), pch=20, col=cols)
  text(0, -6, label=paste("Fold",i,"error:",
       round(mean((dat.te$y - pred.mat[inds==i,1])^2),3)))

  plot(dat$x, dat$y, pch=20, col=cols[(inds!=i)+1], main=paste("Fold",i))
  lines(xx, predict(lm.10.minus.i, data.frame(x=xx)), lwd=2, lty=2)
  legend("topleft", legend=c(paste("Fold",i),"Other folds"), pch=20, col=cols)
  text(0, -6, label=paste("Fold",i,"error:",
       round(mean((dat.te$y - pred.mat[inds==i,2])^2),3)))
}
```

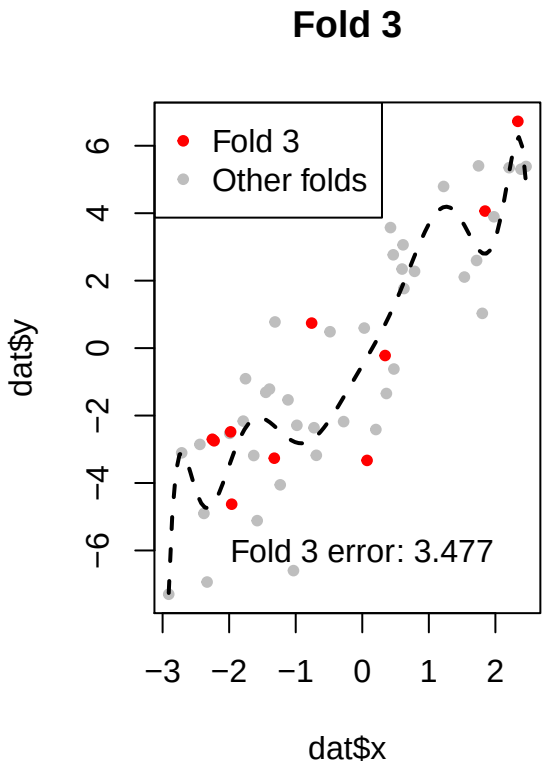

Fold 1
Fold 1
Other folds
Fold 1 error: 5.229
dat$y
dat$x
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2

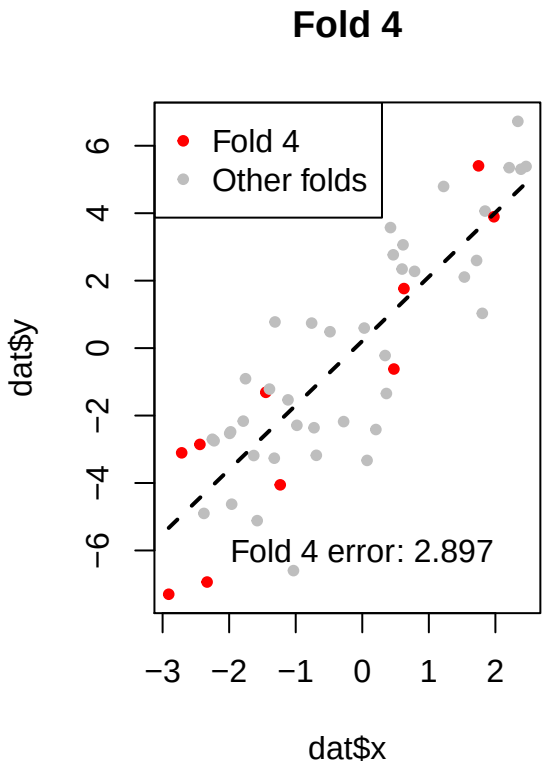

Fold 1
Fold 1
Other folds
Fold 1 error: 5.543
dat$y
dat$x
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2

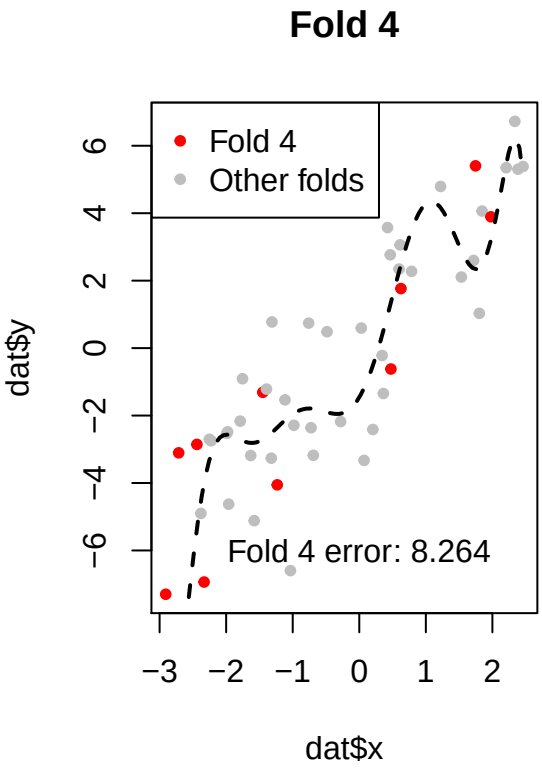

Fold 2
Fold 2
Other folds
Fold 2 error: 3.014
dat$y
dat$x
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2

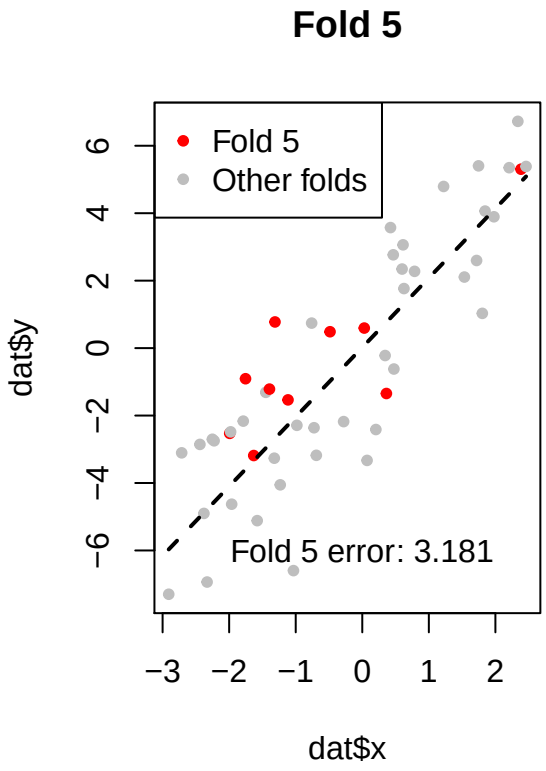

Fold 2
Fold 2
Other folds
Fold 2 error: 4.873
dat$y
dat$x
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2

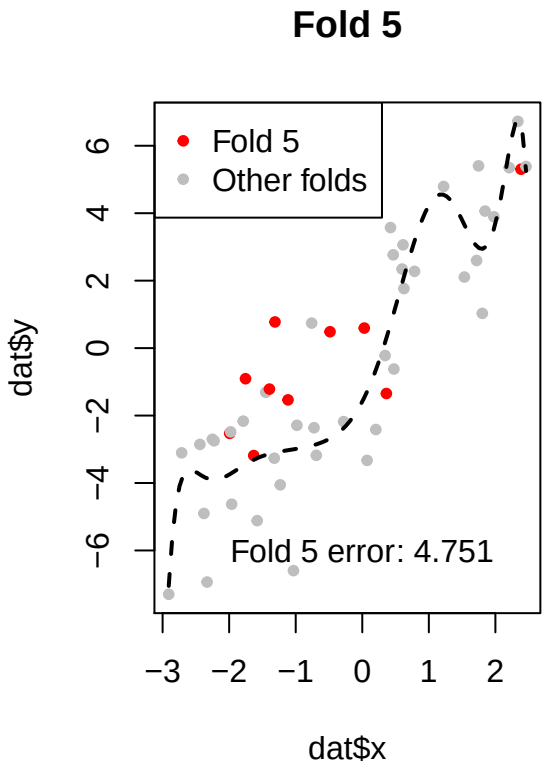

Fold 3
Fold 3
Other folds
Fold 3 error: 3.036
dat$y
dat$x
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2


Fold 3
Fold 3
Other folds
Fold 3 error: 3.477
dat$y
dat$x
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2


Fold 4
Fold 4
Other folds
Fold 4 error: 2.897
dat$y
dat$x
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2


Fold 4
Fold 4
Other folds
Fold 4 error: 8.264
dat$y
dat$x
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2





Part III

Smoothing example

Example: Lidar data

- Consider the `lidar` data from `SemiPar` package in R

```
library(SemiPar)
data(lidar)
```

- Randomly split data into K subsets
- Each observation gets a `foldid` between `1` and `K`

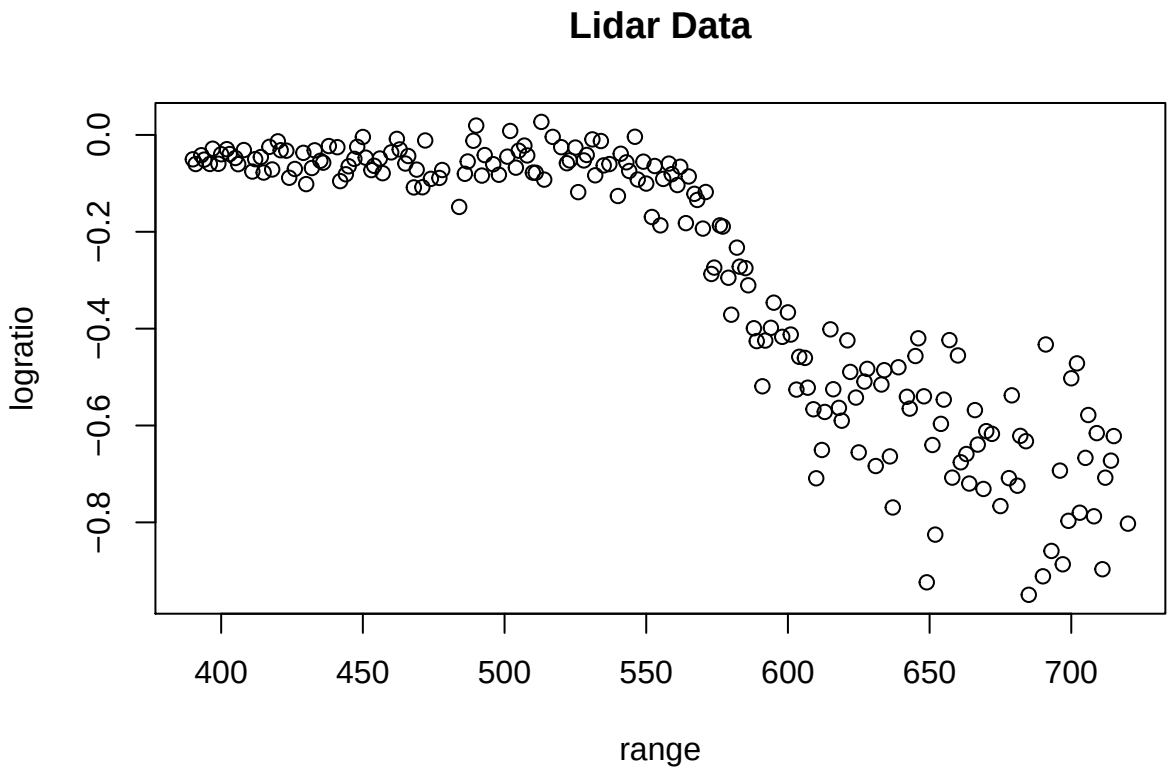
```
K <- 10
n <- nrow(lidar)
foldid <-sample(rep(1:K, length = n))
```

- Split data into training and test data

```
lidar.train <- subset(lidar, foldid != 1)
lidar.test <-  subset(lidar, foldid == 1)
attach(lidar.train)
```

Example: Lidar data

```
plot(range,logratio, main="Lidar Data")
```



loess (locally weighted smoothing)

- loess (locally weighted smoothing) is a popular tool that creates a smooth line through a timeplot or scatter plot to help you to see relationship between variables and foresee trends
- loess is typically to
  - Fitting a line to a scatter plot or time plot where noisy data values, sparse data points or weak interrelationships interfere with your ability to see a line of best fit
  - Linear regression where least squares fitting doesn't create a line of good fit or is too labor-intensive to use
  - Data exploration and analysis

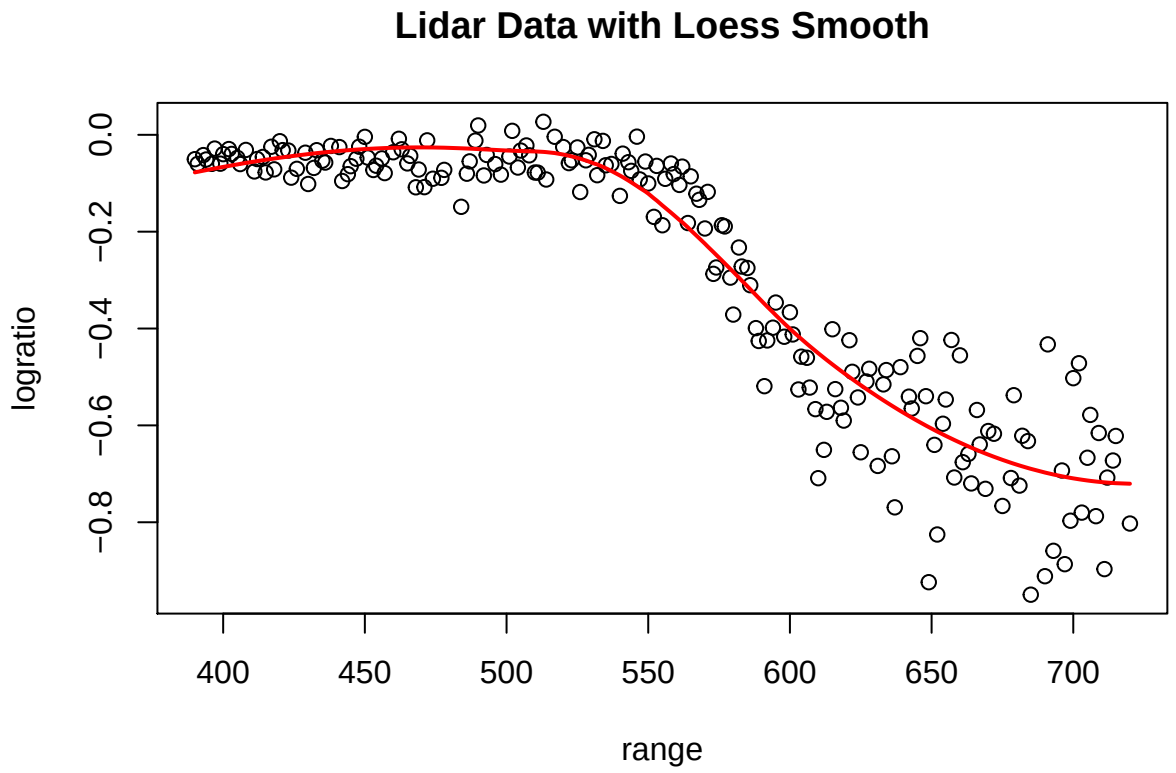
Example: Lidar data

- Can consider fitting a `loess` smooth to the data

```
obj0 <- loess(logratio ~ range, data = lidar.train,
              control = loess.control(surface = 'direct'))
```

Example: Lidar data

```
plot(obj0, xlab="range", ylab="logratio", main="Lidar Data with Loess Smooth")
points(obj0$x , obj0$fitted, type="l", col="red", lwd=2)
```

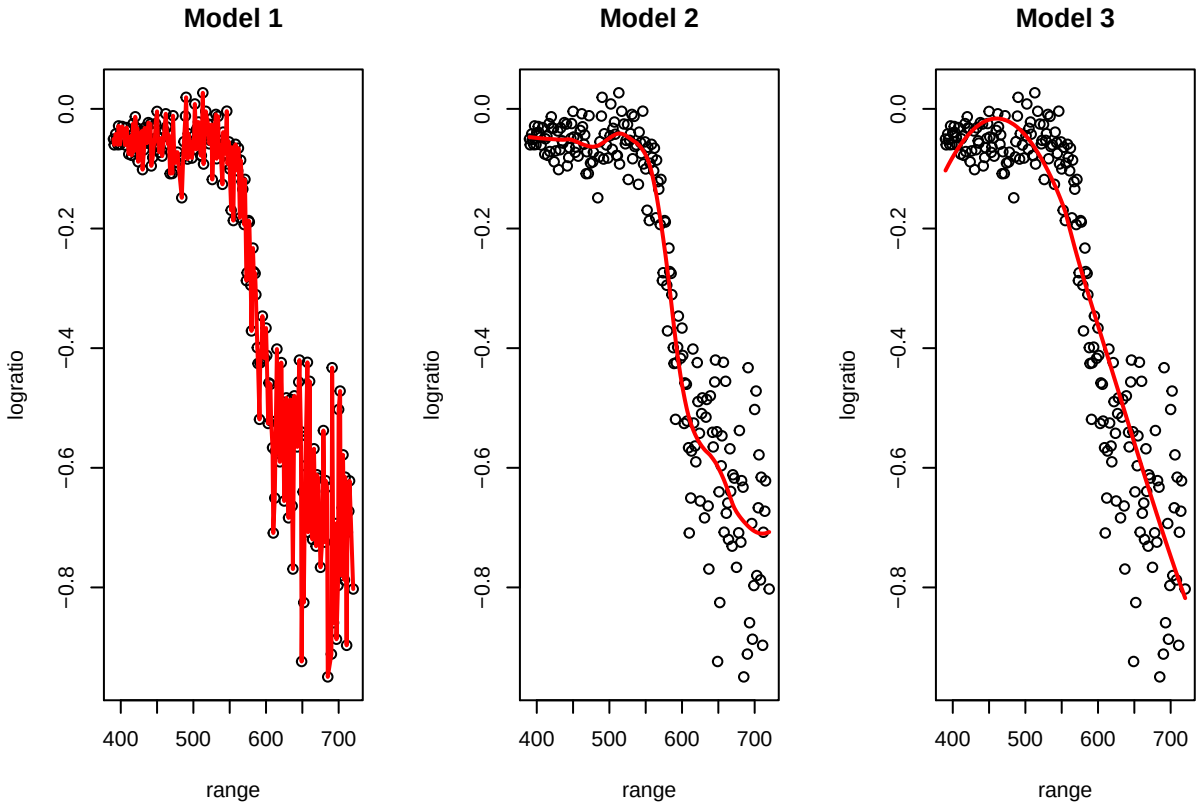


Example: Lidar data

- How to choose the **span** parameter?
- Which model is the best? Why?

```
obj3 <- loess(logratio ~ range, data = lidar.train, span = 1,
              control = loess.control(surface = 'direct'))
obj2 <- loess(logratio ~ range, data = lidar.train, span = .3,
              control = loess.control(surface = 'direct'))
obj1 <- loess(logratio ~ range, data = lidar.train, span = .02,
              control = loess.control(surface = 'direct'))
```

Example: Lidar data



Example: Lidar data

- Model 1 has the smallest mean squared error over the training data

```
c(mean((logratio - obj1$fitted)^2), mean((logratio - obj2$fitted)^2),
  mean((logratio - obj3$fitted)^2))
```

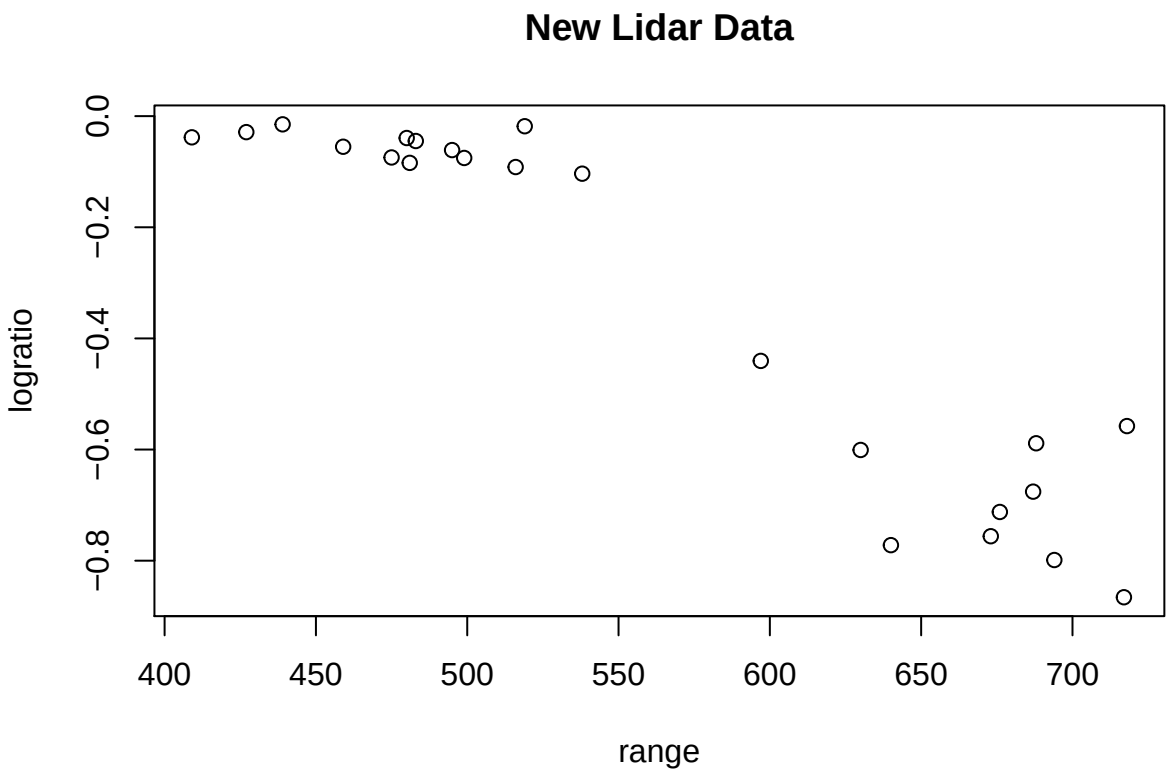
```
## [1] 7.286257e-33 6.033527e-03 9.306627e-03
```

Example: Lidar data

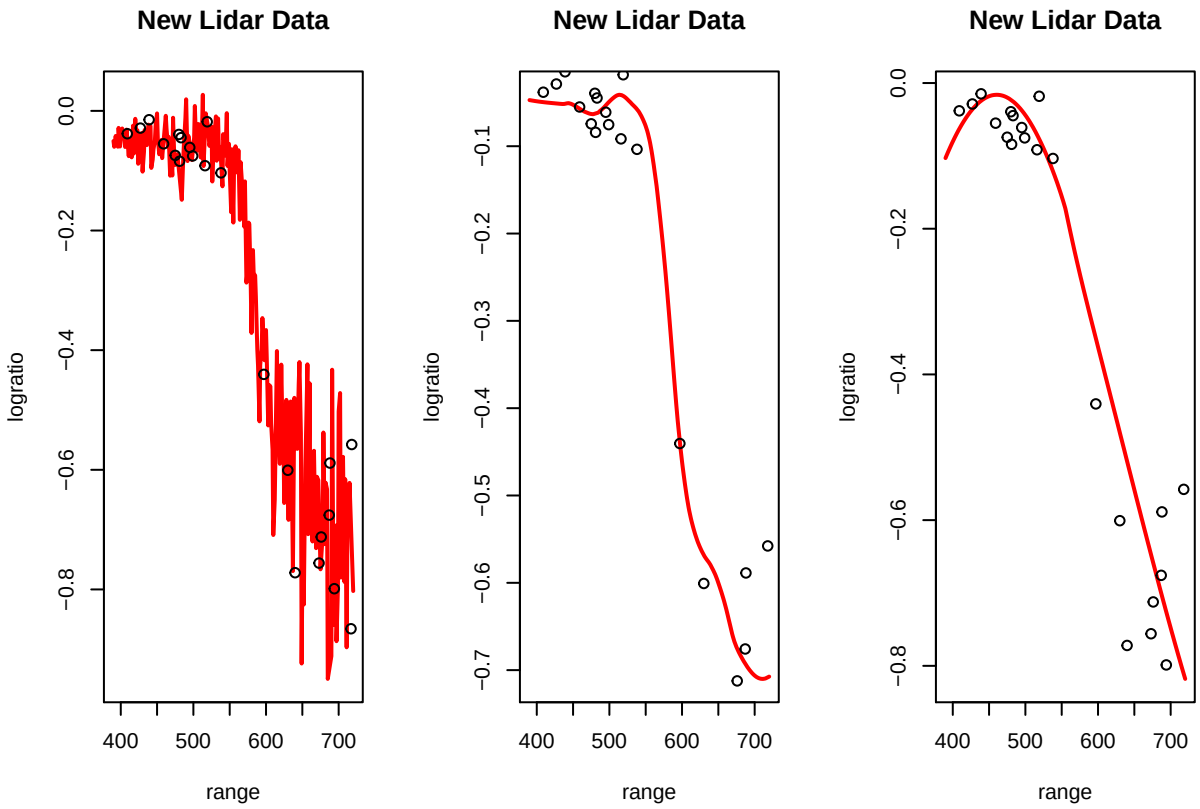
- Suppose now that we have new data from a second experiment, `lidar.test`
- How well do my three models predict the new data? Consider again $1/n_{new} \sum_i \left| Y_i^{(new)} - \hat{f}(X_i^{(new)}) \right|^2$

Example: Lidar data

```
plot(lidar.test$range,lidar.test$logratio, xlab="range",
     ylab="logratio", main="New Lidar Data")
```


New Lidar Data
logratio
0.0
−0.2
−0.4
−0.6
−0.8
400
450
500
550
600
650
700
range

Example: Lidar data



Example: Lidar data

```
y.hat <- predict(obj1, newdata=lidar.test)
mean((lidar.test$logratio - y.hat)^2)
```

```
## [1] 0.05309698
```

```
y.hat <- predict(obj2, newdata=lidar.test)
mean((lidar.test$logratio - y.hat)^2)
```

```
## [1] 0.005388645
```

```
y.hat <- predict(obj3, newdata=lidar.test)
mean((lidar.test$logratio - y.hat)^2)
```

```
## [1] 0.008844213
```

Now notice Model 2 has the smallest squared error over the **new** data

Example: Lidar Data

Using the `lidar` data again, we will

1. Fit several different `loess` smooths corresponding to different choices of bandwidth (span)

2. Estimate the mean squared prediction error of each smooth
3. Choose the smooth with the smallest estimated MSE on the test data

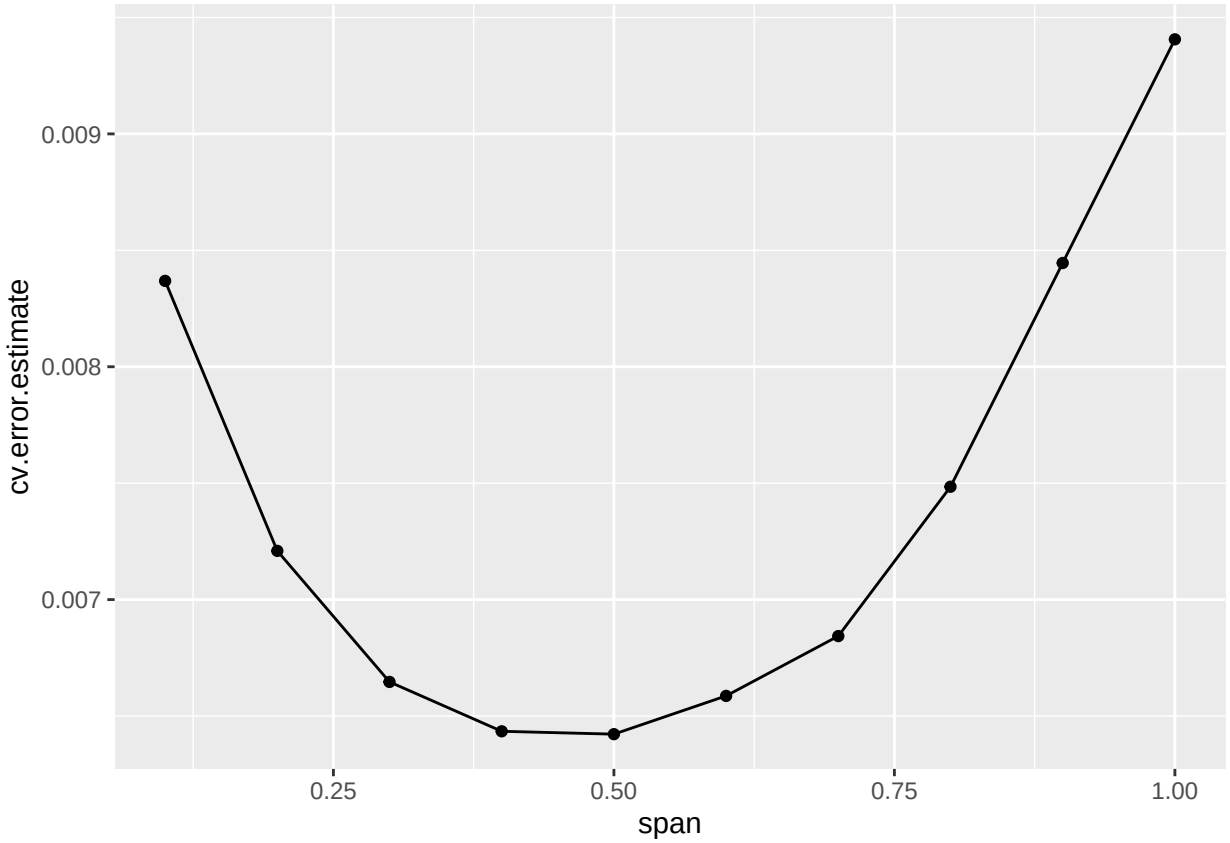
```
library(ggplot2)
library(SemiPar)
data(lidar)
s = seq(from = 0.1, to = 1.0, by = 0.1)
K <- 10
n <- nrow(lidar)
```

Example: Lidar Data

```
cv.error <- matrix(nrow = K, ncol = length(s))
foldid <-sample(rep(1:K, length = n))

for(i in 1:K) {
  # Fit, predict, and calculate error for each bandwidth
  cv.error[i, ] <- sapply(s, function(span) {
    # Fit LOESS model to training set
    obj <- loess(logratio ~ range,
                 data = subset(lidar, foldid != i),
                 span = span,
                 control = loess.control(surface = 'direct'))
    # Predict and calculate error on the validation set
    y.hat <- predict(obj, newdata = subset(lidar, foldid == i))
    pse <- mean((subset(lidar, foldid == i)$logratio - y.hat)^2)
    return(pse)
})
}

# Columns of cv.error correspond to different bandwidths
cv.error.estimate <- colMeans(cv.error)
```

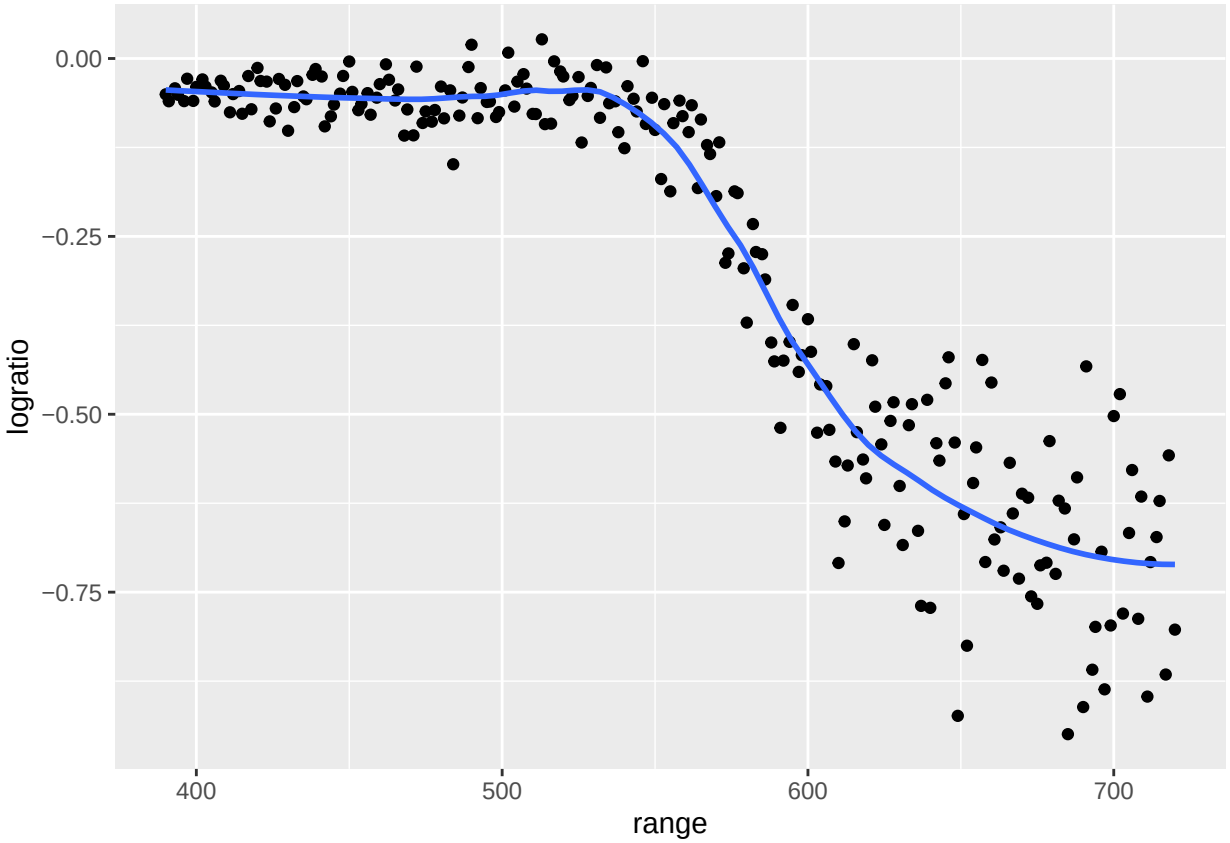
Cross-validation estimates of MSPE

```
qplot(s, cv.error.estimate, geom=c('line', 'point'), xlab='span')
```



Smoother selected by cross-validation

```
s.best <- s[which.min(cv.error.estimate)]
qplot(range, logratio, data=lidar) + geom_smooth(method = 'loess', span = s.best, se = FALSE)
```



Summary

- Can assess models by prediction error
- Select model with smallest prediction error
- Using same data for fitting and estimating prediction error leads to overfitting
- Cross-validation is a method for estimating prediction error that avoids overfitting

Bonus

```
library(glmnet)
```

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

```
## Loading required package: foreach
```

```
## Warning: package 'foreach' was built under R version 3.6.3
```

```
## Loaded glmnet 2.0-18
```

```
set.seed(1010)
n=1000;p=100
x=matrix(rnorm(n*p),n,p)
beta=rnorm(10)
fx= x[,seq(10)] %*% beta
eps=rnorm(n)*5
y=drop(fx+eps)
```

```
set.seed(1011)
cvob1=cv.glmnet(x,y)
```

```
plot(cvob1)
```