# Statistics 452: Statistical Learning and Prediction

## Chapter 5: Resampling Methods

Brad McNeney

2017-10-05

# Resampling Methods

- Cross-validation and bootstrap.
- Grouped together because they involve random sampling of subsets of the data.
- But purpose is different: CV estimates the test error, bootstrap is used to estimate the variance of estimators.

# Test Error

▶ The test error is based on the average squared prediction error over test observations (not used to train the model):

$$\text{Ave}(y_0 - \hat{f}(x_0))^2,$$

where the average is over future $x_0$'s.

▶ We then average this average over repeated samples of training data.

▶ Picture this as repeating the following:

  1. Sample training and test data
  2. Train the model, and use on the test data to obtain the average squared error

▶ and averaging the average from step 2.

▶ With just one training and test set we can get an *estimate* of the test error.

# Validation

- This is what we have already been doing.
- We split our data into two parts, a training set and a validation, or hold-out set.
    - Use the training set for fitting and the validation set for estimating the test error.

# Validation on the `Auto` Data

▶ Split the `Auto` data in half.

```
library(tidyverse)
library(ISLR)
data(Auto)
Auto <- dplyr::select(Auto,mpg,horsepower)
n <- nrow(Auto)
set.seed(42)
# Split in half
train <- sort(sample(1:n,size=n/2)) # sorting not necessary
head(train)
```

```
## [1]  1  2  3  7 11 14
```

```
validn <- setdiff(1:n,train)
head(validn)
```

```
## [1]  4  5  6  8  9 10
```

```
Auto.train <- Auto[train,]
Auto.validn <- Auto[validn,] # same as Auto[-train,]
```

- ▶ Use the train half to train a polynomial model in horsepower and then estimate the test MSE on the validn half.
  - ▶ Software note: poly() returns polynomials and is useful in a model formula to save typing

```
afit <- lm(mpg ~ poly(horsepower,2),data=Auto.train)
yhat.v <- predict(afit,newdata=Auto.validn)
tMSE <- with(Auto.validn,mean((mpg - yhat.v)^2))
tMSE
```

```
## [1] 18.54359
```

# Validation to Select the Degree of Polynomial

```
testMSE <- function(dd,train,validn) {
  afit <- lm(mpg ~ poly(horsepower,dd),data=train)
  yhat.v <- predict(afit,newdata=validn)
  return(with(validn,mean((mpg - yhat.v)^2)))
}
nd <- 10; dd <- (1:nd); tm <- rep(NA,nd)
for(i in dd) {
  tm[i] <- testMSE(i,Auto.train,Auto.validn)
}
dMSE <- data.frame(degree=dd,testMSE= tm)
ggplot(dMSE,aes(x=degree,y=testMSE)) + geom_line()
```
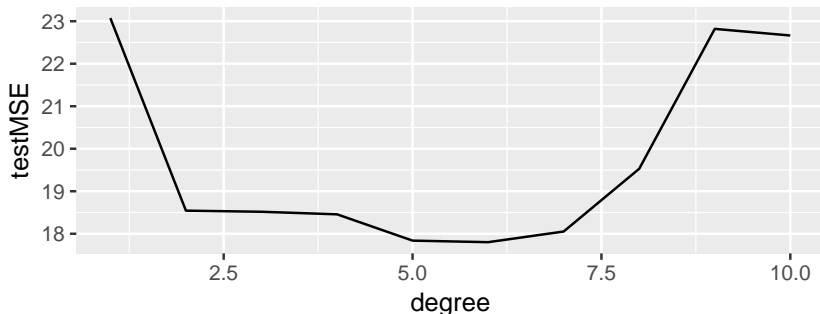
# Validation with Different Validation Sets

```
nValid <- 10
valid <- function() {
  n <- nrow(Auto)
  train <- sample(1:n,size=n/2)
  Auto.train <- Auto[train,]
  Auto.validn <- Auto[-train,]
  tm <- rep(NA,nd)
  for(i in dd) {
    tm[i] <- testMSE(i,Auto.train,Auto.validn)
  }
  tm
}
tMSE <- replicate(nValid,valid())
tMSE <- data.frame(degree=dd,tMSE)
tMSE <- gather(tMSE,validn,testMSE,X1:X10,-degree)
```

```
ggplot(tMSE,aes(x=degree,y=testMSE,color=validn))+ geom_line()
```
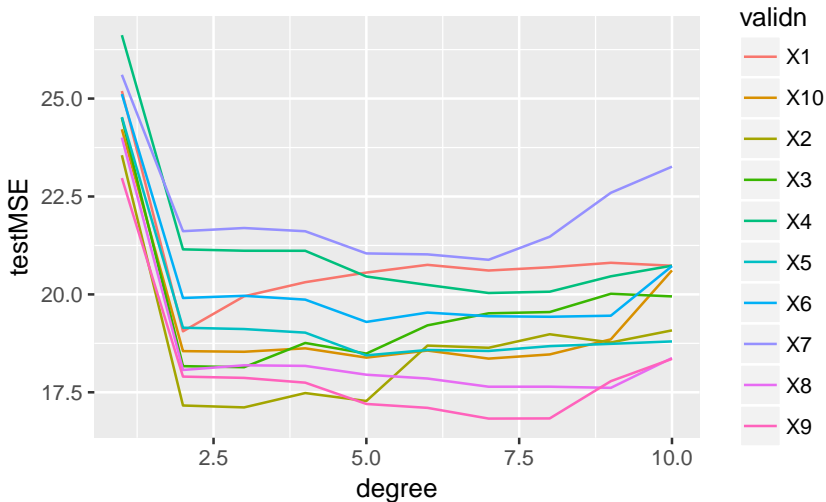


▶ Note the variability in the estimated test MSE.

# Cross-Validation (CV)

- Rather than a single data split, do multiple splits into "folds" of approximately equal size.

    - Common numbers of folds are $k = n$, 10 and 5.

- Train on all but one hold-out fold, and test on the hold-out to get $\text{MSE}_i$; $i = 1, \ldots, k$.

- Repeat for each fold and average the estimated test MSEs:

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^{k} \text{MSE}_i.$$

# Leave-Out-One CV (LOOCV)

- Break the data into *n* folds, with one observation in each fold.
- Computational trick for a linear model fit by least squares:

$$CV_{(n)} = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{y_i - \hat{y}_i}{1 - h_i} \right),$$

where $\hat{y}_i$ is the fitted value from the least squares fit and $h_i$ is the leverage of the *i*th observation.

# LOOCV on Auto Data

```
loocv <- function(dd) {
  CVn <- rep(NA,length(dd))
  for(i in dd) {
    fit <- lm(mpg ~ poly(horsepower,i),data=Auto)
    hh <- hatvalues(fit)
    ff <- fitted.values(fit)
    CVn[i] <- with(Auto,mean(((mpg-ff)/(1-hh))^2))
  }
  CVn
}
cv.err <- loocv(dd)
cv.err <- data.frame(degree=dd,CV=cv.err)
```

```
ggplot(cv.err,aes(x=degree,y=CV)) + geom_line()
```
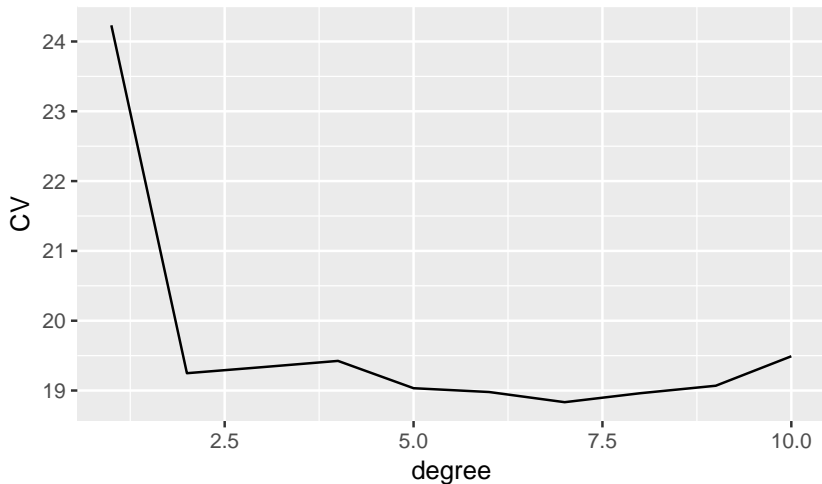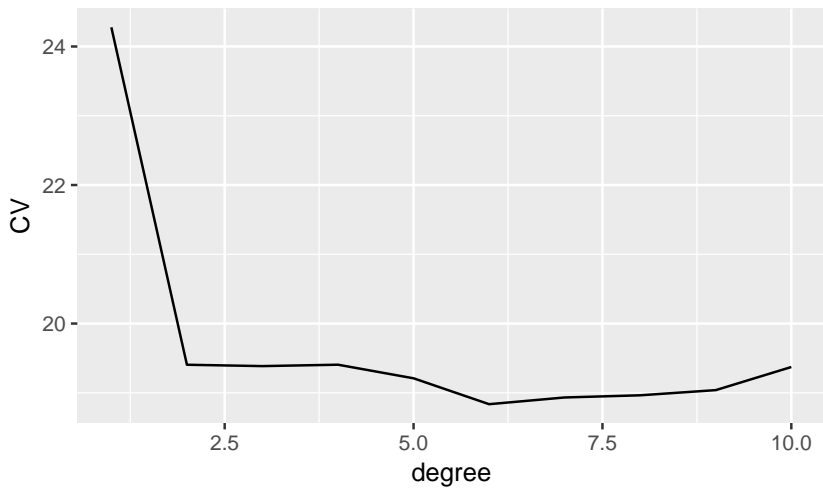
# 10-Fold CV on `Auto` Data

- ▶ Can use a function `cv.glm()` from the `boot` package.
    - ▶ Uses output from `glm()`.
    - ▶ glm default is normal errors; i.e., `lm()`.

```
library(boot)
cv.err <- rep(NA,nd)
set.seed(123)
for(i in dd) {
  fit <- glm(mpg ~ poly(horsepower,i),data=Auto)
  cc <- cv.glm(Auto,fit,K=10)
  cv.err[i] <- cc$delta[1]
}
cv.err <- data.frame(degree=dd,CV=cv.err)
```

```r
ggplot(cv.err,aes(x=degree,y=CV)) + geom_line()
```

# Bias-Variance Trade-Off for $k$-Fold CV

- In general, computation of $k$-fold CV increases with $k$.
- But more important is the accuracy of the CV estimator as a function of $k$.
- There are two components to accuracy, bias and variance.
  - It can be shown that the bias of the CV estimator of the test error *decreases* as $k$ increases.
  - It can be shown that the variance of the CV estimator *increases* with $k$.

# Bias

- If data splitting results in a training set that is small, the error of the statistical learning method will be larger than if we fit to all data.
- Implies an upward bias in the estimate of the test error.
- On the other extreme, LOOCV uses almost all the data to train, and so will have almost no bias.

# Variance

- ▶ This is harder to reason through.
- ▶ The LOOCV estimate is an average of many squared errors that are (i) highly variable, and (ii) positively correlated.
  - ▶ Averaging many things is good.
  - ▶ The positive correlation arrises from using mostly the same data to fit the model each time.
- ▶ For $k$-fold CV with smaller $k$, we average fewer MSEs that are (i) less variable and

2. less correlated.

- ▶ Which "wins"? Turns out that $k = 5$ or $10$ have been shown to work well empirically.
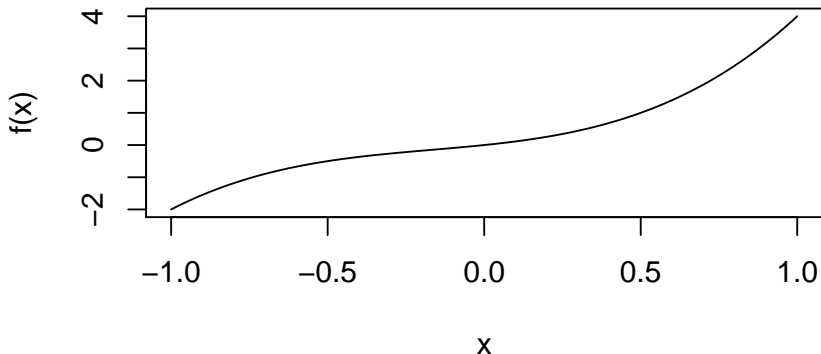
# Simulation Example

- We can illustrate the bias and variance of the CV estimator of the test error with one simlation model.
- Simulate from the model

$$Y = X + X^2 + 2X^3 + \epsilon$$

for $\epsilon \sim N(0, 1)$ and a fixed grid of $X$-values.
- Fit a linear regression.
- View the contributions to $CV_{(n)}$ and $CV_{(10)}$ and the variance of these two estimators.

```r
n <- 100
x <- seq(-1,1,length=n)
plot(x,x+x^2+2*x^3,ylab="f(x)",type="l")
```



```r
simdat <- function() {
  y<-rnorm(n,mean=x+x^2+2*x*3,sd=1)
  data.frame(y=y,x=x)
}
```

```
# LOOCV contributions for data set of size 100
loocv.contrib <- function() {
  MSE <- rep(NA,n)
  dat <- simdat()
  for(i in 1:n) {
    fit <- lm(y~x,data=dat,subset=(-i))
    newdat <- data.frame(x=dat$x[i])
    pp <- predict(fit,newdata=newdat)
    MSE[i] <- (dat$y[i] - pp)^2
  }
  MSE
}
```

- ▶ Repeat the following a few times

```
mm <- loocv.contrib()
plot(mm,ylim=c(0,10))
abline(h=mean(mm)) # LOOCV estimate of test error
```
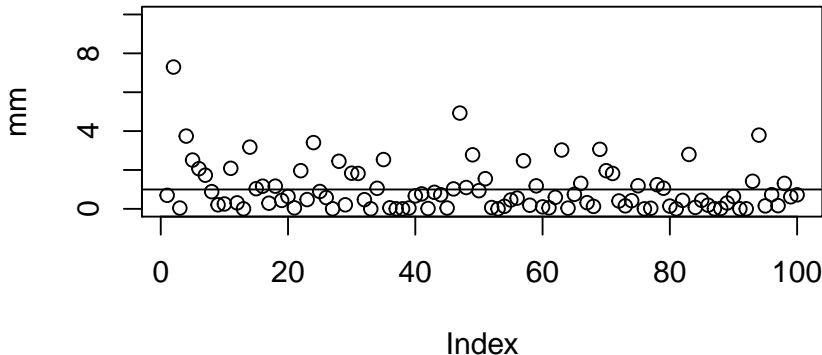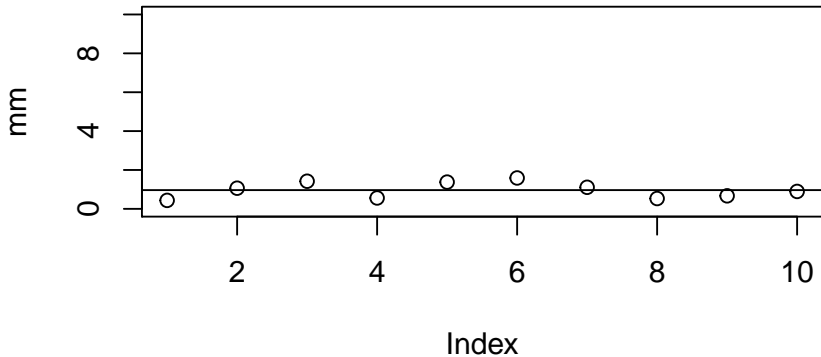
```r
# 10-fold CV contributions for data set of size 100
cv.contrib <- function(n=100,k=10) {
  MSE <- rep(NA,k)
  dat <- simdat()
  inds <- sample(1:n)
  for(i in 1:(n/k)) {
    ss <- inds[(i-1)*k + (1:(n/k))]
    fit <- lm(y~x,data=dat,subset=(-ss))
    newdat <- data.frame(x=dat$x[ss])
    pp <- predict(fit,newdata=newdat)
    MSE[i] <- mean((dat$y[ss] - pp)^2)
  }
  MSE
}
```

- ▶ Repeat the following a few times

```r
mm <- cv.contrib()
plot(mm,ylim=c(0,10))
abline(h=mean(mm)) # Estimate of test error
```

- ▶ Hard to see how the CV estimates (horzontal lines) vary over samples, so record them over many simulations.

```
set.seed(123)
NREPS <- 1000
LOOCVres <- CVres <- rep(NA,NREPS)
for(i in 1:NREPS) {
  LOOCVres[i] <- mean(loocv.contrib())
  CVres[i] <- mean(cv.contrib())
}
mean(LOOCVres); var(LOOCVres);mean(CVres); var(CVres)
```

## [1] 1.122344

## [1] 0.02634004

## [1] 1.124939

## [1] 0.02689808

- ▶ Can see suggestion of lower bias from LOOCCV, but not lower variance of 10-fold CV for this scenario.

# CV on Classification Problems

- We have illustrated the idea behind CV when the response is quantitative.
- We then use the MSE (mean squared error) to quantify test error.
- For classification problems we measure the error of a procedure by the misclassification error.
- For example,

$$\mathrm{CV}_{(n)} = \frac{1}{n} \sum_{i=1}^{n} I(y_i \neq \hat{y}_i)$$

- See the text for examples.

# The Bootstrap

- The bootstrap uses resampling to quantify uncertainty in an estimator.

# Assumptions and sampling distributions

- ▶ Under model assumptions, the sampling distribution of the statistics used for inference are known.

  - ▶ Sampling distribution: Distribution of a statistic over repeated samples of data **from the population**.
  - ▶ For regression coefficients, the sampling distribution leads to t-tests and CIs

- ▶ The bootstrap is a data-driven approach to approximating the sampling distribution of inferential statistics.

  - ▶ Find the distribution of a statistic over repeated samples of data **from the original sample**.
  - ▶ Reasonable if original sample is representative of the population.
  - ▶ Base inference on the bootstrap approximate distribution.

# Advantages

- Bootstrap may give reasonable uncertainty estimates when assumptions for traditional inference don't hold.
- We can expand the definition of the procedure used to obtain the estimates to include variable selection and/or other smoothing (more on this later).

# Resampling

- ▶ Resampling means drawing samples, with replacement, from the original sample.
  - ▶ E.G., drawing cars, with replacement.

```
set.seed(42)
n <- nrow(Auto)
Autos <- data.frame(index=1:n,Auto)
resamAuto <- sample_n(Auto,size=n,replace=TRUE)
head(resamAuto)
```

```
##        mpg horsepower
## 364 22.4        110
## 373 27.0         90
## 114 21.0        107
## 328 36.4         67
## 254 20.5         95
## 206 28.0         75
```

# Bootstrap Standard Errors

- Resample some number $B$ times.
- For each resample compute the estimates.
- Take the sample SD of the bootstrap estimates.

```
B <- 500; beta1Boot <- rep(NA,B)
for(i in 1:B) {
  rAuto <- sample_n(Auto,size=n,replace=TRUE)
  fit <- lm(mpg~poly(horsepower,2),data=rAuto)
  beta1Boot[i] <- coefficients(fit)[2]
}
beta1Boot <- data.frame(beta1=beta1Boot)
```

```r
ggplot(beta1Boot,aes(x=beta1)) + geom_histogram()
```
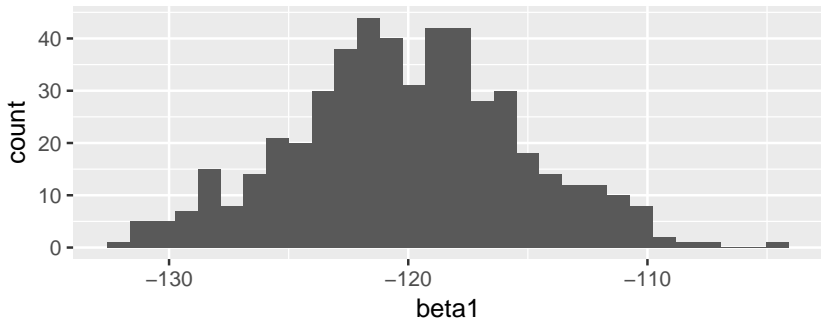


```r
with(beta1Boot,sd(beta1))
```

```
## [1] 4.72912
```

```r
fit <- lm(mpg~poly(horsepower,2),data=Auto)
round(summary(fit)$coefficients[2,],4)
```

```
##   Estimate Std. Error    t value   Pr(>|t|)
## -120.1377     4.3739   -27.4668     0.0000
```