Cross-validation

Looking back at evaluating model accuracy

Sonja Petrović
Created for ITMD/ITMS/STAT 514

Spring 2021.

Goals for this lecture

- Review basics of testing vs. training MSE in the regression setting
- Review why we cross-validate
- See a couple of examples

Review - estimating f

Review - estimating f

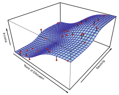
First, we recall the notation and set the context.

Let's review a couple of slides extracted from
514-3.3-StatisticalLearning-ModelAccuracyEtc.pdf

Notation and setup

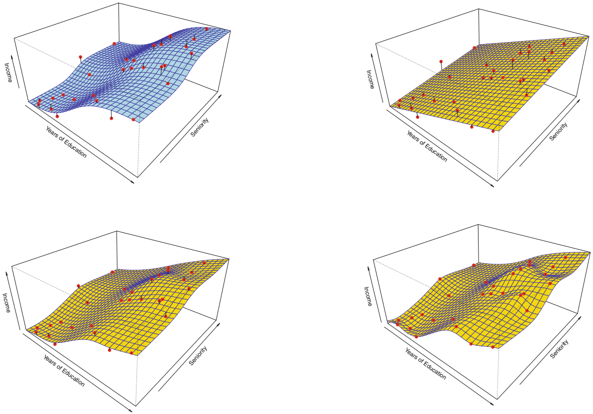
- Observe: a quantitative response Y, p different predictors, $X_1, X_2, \ldots, X_p$.
- Assume: some relationship between Y and $X = (X_1, X_2, ..., X_p)$, which can be written in the very general form

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



- f is some fixed but unknown function of $X_1, X_2, \ldots, X_p$
- ϵ is a random error term, which is independent of X and has mean zero.
- In this formulation, f represents the *systematic* information that X provides about Y.

[Regression setting][1] f, $Y = f(X) + \epsilon$, $\hat{f}$, $\hat{Y} = \hat{f}(X)$



[1] ISLR book figures.

Assessing model accuracy

- **Task:** 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.
- Need: measure how well predictions match observed data.
  - $\rightarrow$ quantify the extent to which the predicted response value for a given observation is close to the true response value for that observation.

Training Mean squared error (MSE)

$$train.MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

$\hat{f}(x_i)$ is the prediction that f gives for the ith observation.

Real question:
What is the accuracy of the predictions that we obtain when we apply our method to *previously* *unseen* test data?

Training vs. test data

Example 1

Goal: Develop an algorithm to predict a stock's price based on previous stock returns.

- We can train the method using stock returns from the past 6 months.
- But we don't really care how well our method predicts last week's stock price.
- We instead care about how well it will predict **tomorrow's price** or **next month's price**.

Example 2

Goal: predict diabetes risk for future patients based on their clinical measurements.

- Clinical measurements (e.g. weight, blood pressure, height, age, family history of disease) for a number of patients, + info whether each patient has diabetes.
- Train a statistical learning method to **predict risk of diabetes based on clinical measurements**.
- No interest: whether method accurately predicts diabetes risk for patients used to train the model, since we already know which of those patients have diabetes.

The test MSE

- (x_0, y_0) a *previously unseen test observation*
- **Goal**: $\hat{f}(x_0) \approx y_0$?

Test MSE

$$test.MSE = Ave(y_0 - \hat{f}(x_0))^2$$

average squared prediction error for test observations (x_0, y_0).

How to select a method that minimizes MSE?

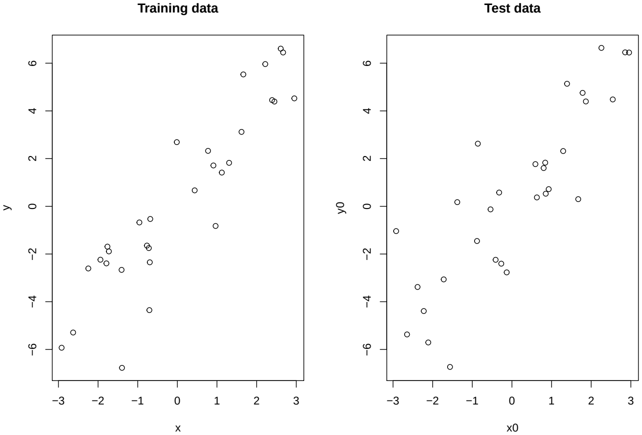
Scenario: test data available

- Set of observations not used to train the statistical model.
- Evaluate test MSE, $Ave(y_0 - \hat{f}(x_0))^2$ on that set.
  We'll partition the given sample into training & testing data sets.

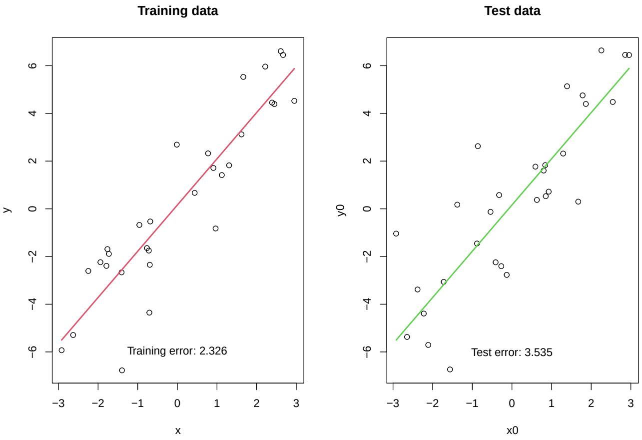
Examples

Training and testing MSE on simulated data

```r
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)

par(mfrow=c(1,2))
xlim = range(c(x,x0)); ylim = range(c(y,y0))
plot(x, y, xlim=xlim, ylim=ylim, main="Training data")
plot(x0, y0, xlim=xlim, ylim=ylim, main="Test data")
```


Training data
Test data
y
y0
x
x0
−6
−4
−2
0
2
4
6
−3
−2
−1
0
1
2
3

```r
# 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 data
Test data
Training error: 2.326
Test error: 3.535
x
y
x0
y0
−3
−2
−1
0
1
2
3
−6
−4
−2
0
2
4
6

Cross-validation

Review: idea behind cross-validation

Cross-validation is essentially one of the resampling methods. $>$ Estimate the test error rate 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.

- Remember:
    - Testing error measures average error on measurements that were not used to train the method.
    - Available test data set $\implies$ testing error easy to compute.
- 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: **sample-splitting**
    - Split the data set into two parts
    - First part: train the model/method
    - Second part: make predictions
    - Evaluate observed test error

Sample-splitting on an example

```
dat=read.table("http://www.stat.cmu.edu/~ryantibs/statcomp/data/xy.
head(dat, 3)
```

```
          x         y
1 -2.908021 -7.298187
2 -2.713143 -3.105055
3 -2.439708 -2.855283
```

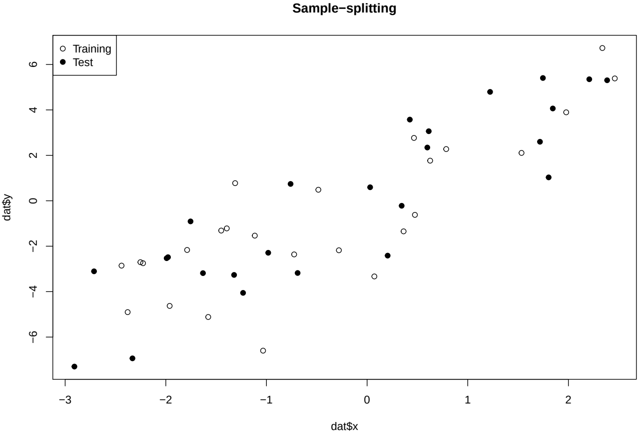
```
n = nrow(dat)
# Split data in half, randomly
set.seed(0)
inds = sample(rep(1:2, length=n))
head(inds, 10)
```

```
 [1] 2 2 1 1 2 1 1 2 2 1
```

```
table(inds)
```

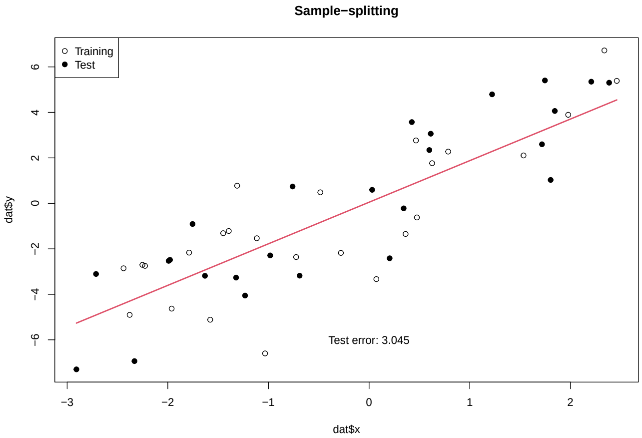
```
inds
 1  2
25 25
```

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

plot(dat$x, dat$y, pch=c(21,19)[inds], 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)

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

# evaluate test error
test.err.1 = mean((dat.te$y - pred.1)^2)
```

```
# Plot the results
xx = seq(min(dat$x), max(dat$x), length=100)
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)))
```



Why 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!)

Example

```r
# 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
```

```r
table(inds)
```

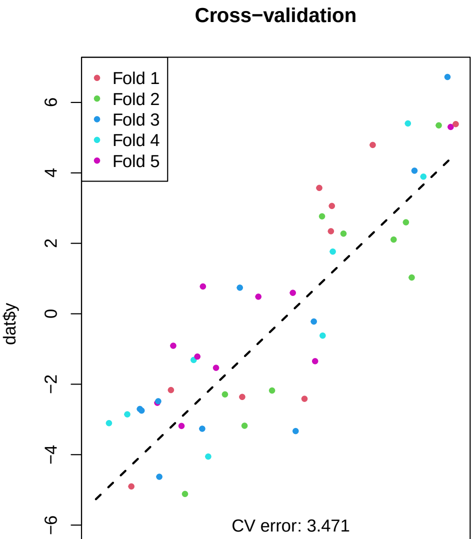
```
inds
 1  2  3  4  5
10 10 10 10 10
```

```
# 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)

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

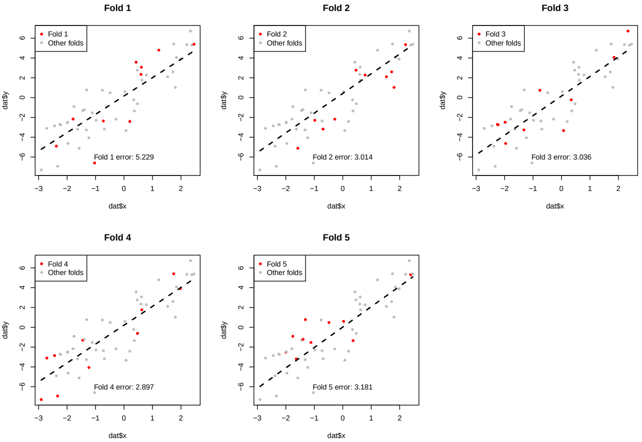
```
Fold 1 ... Fold 2 ... Fold 3 ... Fold 4 ... Fold 5 ...
```

```r
# Compute cross-validation error
cv.errs = colMeans((pred.mat - dat$y)^2)
```

```r
# Plot the results
par(mfrow=c(1,2))
xx = seq(min(dat$x), max(dat$x), length=100)
plot(dat$x, dat$y, pch=20, col=inds+1, main="Cross-validation")
lines(xx, predict(lm.1, data.frame(x=xx)), # Note: model trained on FULL data!
      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)))
```



```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)

  # Plot fitted models
  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)))
  }
```



Fold 1
Fold 1
Other folds
Fold 1 error: 5.229
Fold 2
Fold 2
Other folds
Fold 2 error: 3.014
Fold 3
Fold 3
Other folds
Fold 3 error: 3.036
Fold 4
Fold 4
Other folds
Fold 4 error: 2.897
Fold 5
Fold 5
Other folds
Fold 5 error: 3.181
dat$x
dat$y

License

Contents of this lecture is based on the chapter 3 of the textbook Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, *' An Introduction to Statistical Learning: with Applications in R'*.

The simulated test/train data example is taken from Prof. Ryan Tibshirani's statistical computing course notes.