Bootstrapping for prediction

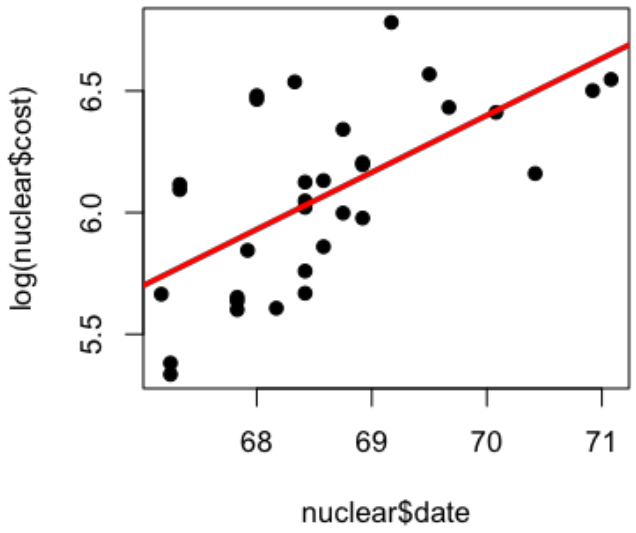
Jeffrey Leek, Assistant Professor of Biostatistics
Johns Hopkins Bloomberg School of Public Health

Key ideas

- Bootstrapping can be used for
  - Cross-validation type error rates
  - Prediction errors in regression models
  - Improving prediction

Bootstrapping prediction errors

```
library(boot); data(nuclear)
nuke.lm <- lm(log(cost) ~ date,data=nuclear)
plot(nuclear$date,log(nuclear$cost),pch=19)
abline(nuke.lm,col="red",lwd=3)
```



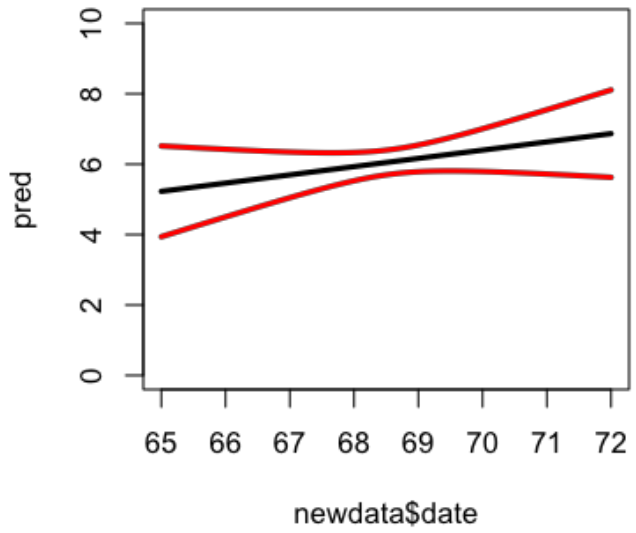
Bootstrapping prediction errors

```
newdata <- data.frame(date = seq(65,72,length=100))
nuclear <- cbind(nuclear,resid=rstudent(nuke.lm),fit=fitted(nuke.lm))
nuke.fun <- function(data,inds,newdata){
  lm.b <- lm(fit + resid[inds] ~ date,data=data)
  pred.b <- predict(lm.b,newdata)
  return(pred.b)
}
nuke.boot <- boot(nuclear,nuke.fun,R=1000,newdata=newdata)
head(nuke.boot$t)
```

```
     [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12] [,13] [,14] [,15]
[1,] 5.475 5.490 5.506 5.521 5.536 5.551 5.566 5.581 5.597 5.612 5.627 5.642 5.657 5.672 5.688
[2,] 6.370 6.365 6.359 6.354 6.348 6.343 6.337 6.332 6.326 6.321 6.315 6.310 6.304 6.299 6.293
[3,] 5.353 5.366 5.380 5.393 5.406 5.420 5.433 5.446 5.459 5.473 5.486 5.499 5.513 5.526 5.539
[4,] 4.476 4.506 4.537 4.567 4.597 4.627 4.658 4.688 4.718 4.748 4.779 4.809 4.839 4.869 4.900
[5,] 4.980 5.002 5.024 5.046 5.067 5.089 5.111 5.133 5.155 5.177 5.199 5.220 5.242 5.264 5.286
[6,] 2.674 2.739 2.804 2.869 2.935 3.000 3.065 3.130 3.195 3.260 3.325 3.390 3.456 3.521 3.586
     [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30]
[1,] 5.703 5.718 5.733 5.748 5.763 5.779 5.794 5.809 5.824 5.839 5.854 5.870 5.885 5.900 5.915
[2,] 6.288 6.282 6.277 6.271 6.266 6.260 6.255 6.249 6.244 6.238 6.233 6.227 6.222 6.216 6.211
[3,] 5.553 5.566 5.579 5.593 5.606 5.619 5.633 5.646 5.659 5.673 5.686 5.699 5.713 5.726 5.739
[4,] 4.930 4.960 4.991 5.021 5.051 5.081 5.112 5.142 5.172 5.202 5.233 5.263 5.293 5.323 5.354
[5,] 5.308 5.330 5.352 5.373 5.395 5.417 5.439 5.461 5.483 5.505 5.526 5.548 5.570 5.592 5.614
```

Bootstrapping prediction errors

```
pred <- predict(nuke.lm,newdata)
predSds <- apply(nuke.boot$t,2,sd)
plot(newdata$date,pred,col="black",type="l",lwd=3,ylim=c(0,10))
lines(newdata$date,pred + 1.96*predSds,col="red",lwd=3)
lines(newdata$date,pred - 1.96*predSds,col="red",lwd=3)
```



Bootstrap aggregating (bagging)

Basic idea:

1. Resample cases and recalculate predictions
2. Average or majority vote

Notes:

- Similar bias
- Reduced variance
- More useful for non-linear functions

Bagged loess

```
library(ElemStatLearn); data(ozone,package="ElemStatLearn")
ozone <- ozone[order(ozone$ozone),]
head(ozone)
```

```
    ozone radiation temperature wind
17      1         8          59  9.7
19      4        25          61  9.7
14      6        78          57 18.4
45      7        48          80 14.3
106     7        49          69 10.3
7       8        19          61 20.1
```

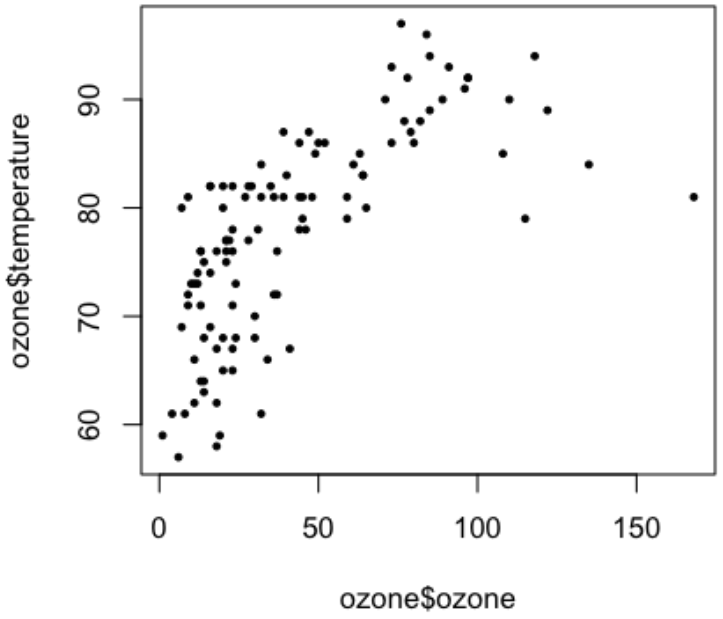
http://en.wikipedia.org/wiki/Bootstrap_aggregating

Bagged loess

```
ll <- matrix(NA,nrow=10,ncol=155)
for(i in 1:10){
  ss <- sample(1:dim(ozone)[1],replace=T)
  ozone0 <- ozone[ss,]; ozone0 <- ozone0[order(ozone0$ozone),]
  loess0 <- loess(temperature ~ ozone,data=ozone0,span=0.2)
  ll[i,] <- predict(loess0,newdata=data.frame(ozone=1:155))
}
```

Bagged loess

```
plot(ozone$ozone,ozone$temperature,pch=19,cex=0.5)
```




```
for(i in 1:10){lines(1:155,ll[i,],col="grey",lwd=2)}
lines(1:155,apply(ll,2,mean),col="red",lwd=2)
```

Bagged trees

Basic idea:

1. Resample data
2. Recalculate tree
3. Average/mode) of predictors

Notes:

1. More stable
2. May not be as good as random forests

Iris data

```
data(iris)
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Bagging a tree

```
library(ipred)
bagTree <- bagging(Species ~.,data=iris,coob=TRUE)
print(bagTree)
```

```

Bagging classification trees with 25 bootstrap replications

Call: bagging.data.frame(formula = Species ~ ., data = iris, coob = TRUE)

Out-of-bag estimate of misclassification error:  0.0667
```

Looking at bagged tree one

```
bagTree$mtrees[[1]]$btree
```

```
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 150 98 virginica (0.32667 0.32667 0.34667)
    2) Petal.Length< 2.6 49  0 setosa (1.00000 0.00000 0.00000) *
    3) Petal.Length>=2.6 101 49 virginica (0.00000 0.48515 0.51485)
      6) Petal.Length< 4.85 48  2 versicolor (0.00000 0.95833 0.04167)
       12) Sepal.Length>=4.95 46  0 versicolor (0.00000 1.00000 0.00000) *
       13) Sepal.Length< 4.95 2  0 virginica (0.00000 0.00000 1.00000) *
      7) Petal.Length>=4.85 53  3 virginica (0.00000 0.05660 0.94340)
       14) Petal.Width< 1.7 10  3 virginica (0.00000 0.30000 0.70000)
         28) Sepal.Width>=3.05 2  0 versicolor (0.00000 1.00000 0.00000) *
         29) Sepal.Width< 3.05 8  1 virginica (0.00000 0.12500 0.87500)
           58) Sepal.Length< 6.05 3  1 virginica (0.00000 0.33333 0.66667)
            116) Sepal.Width>=2.45 1  0 versicolor (0.00000 1.00000 0.00000) *
            117) Sepal.Width< 2.45 2  0 virginica (0.00000 0.00000 1.00000) *
           59) Sepal.Length>=6.05 5  0 virginica (0.00000 0.00000 1.00000) *
       15) Petal.Width>=1.7 43  0 virginica (0.00000 0.00000 1.00000) *
```

Looking at bagged tree two

```
bagTree$mtrees[[2]]$btree
```

```
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 150 92 versicolor (0.28667 0.38667 0.32667)
    2) Petal.Length< 2.7 43  0 setosa (1.00000 0.00000 0.00000) *
    3) Petal.Length>=2.7 107 49 versicolor (0.00000 0.54206 0.45794)
      6) Petal.Width< 1.65 59  3 versicolor (0.00000 0.94915 0.05085)
       12) Petal.Length< 5.35 57  1 versicolor (0.00000 0.98246 0.01754)
         24) Petal.Length< 4.85 54  0 versicolor (0.00000 1.00000 0.00000) *
         25) Petal.Length>=4.85 3  1 versicolor (0.00000 0.66667 0.33333)
           50) Sepal.Width>=2.45 2  0 versicolor (0.00000 1.00000 0.00000) *
           51) Sepal.Width< 2.45 1  0 virginica (0.00000 0.00000 1.00000) *
       13) Petal.Length>=5.35 2  0 virginica (0.00000 0.00000 1.00000) *
      7) Petal.Width>=1.65 48  2 virginica (0.00000 0.04167 0.95833)
       14) Petal.Width< 1.85 11  2 virginica (0.00000 0.18182 0.81818)
         28) Sepal.Width>=2.9 6  2 virginica (0.00000 0.33333 0.66667)
           56) Petal.Length< 5.05 3  1 versicolor (0.00000 0.66667 0.33333)
            112) Sepal.Length< 5.95 1  0 versicolor (0.00000 1.00000 0.00000) *
            113) Sepal.Length>=5.95 2  1 versicolor (0.00000 0.50000 0.50000)
```

Random forests

1. Bootstrap samples
2. At each split, bootstrap variables
3. Grow multiple trees and vote

Pros:

1. Accuracy

Cons:

1. Speed
2. Interpretability
3. Overfitting

Random forests

```
library(randomForest)
forestIris <- randomForest(Species~ Petal.Width + Petal.Length,data=iris,prox=TRUE)
forestIris
```

```
Call:
 randomForest(formula = Species ~ Petal.Width + Petal.Length,      data = iris, prox = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 3.33%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         50          0         0        0.00
versicolor      0         47         3        0.06
virginica       0          2        48        0.04
```

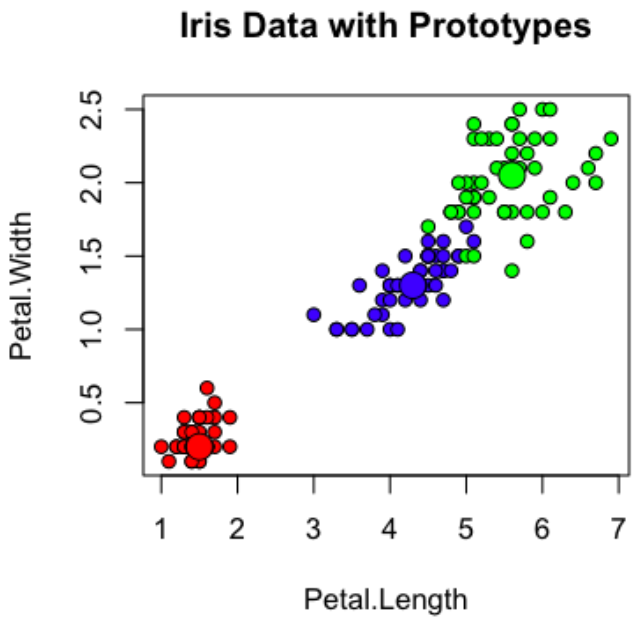
Getting a single tree

```
getTree(forestIris,k=2)
```

```
  left daughter right daughter split var split point status prediction
1              2               3         1        0.80      1          0
2              0               0         0        0.00     -1          1
3              4               5         1        1.75      1          0
4              6               7         2        5.45      1          0
5              8               9         1        1.85      1          0
6              0               0         0        0.00     -1          2
7              0               0         0        0.00     -1          3
8              0               0         0        0.00     -1          3
9              0               0         0        0.00     -1          3
```

Class "centers"

```
iris.p <- classCenter(iris[,c(3,4)], iris$Species, forestIris$prox)
plot(iris[,3], iris[,4], pch=21, xlab=names(iris)[3], ylab=names(iris)[4],
bg=c("red", "blue", "green")[as.numeric(factor(iris$Species))],
main="Iris Data with Prototypes")
points(iris.p[,1], iris.p[,2], pch=21, cex=2, bg=c("red", "blue", "green"))
```



Combining random forests

```
forestIris1 <- randomForest(Species~Petal.Width + Petal.Length,data=iris,prox=TRUE,ntree=50)
forestIris2 <- randomForest(Species~Petal.Width + Petal.Length,data=iris,prox=TRUE,ntree=50)
forestIris3 <- randomForest(Species~Petal.Width + Petal.Length,data=iris,prox=TRUE,nrtee=50)
combine(forestIris1,forestIris2,forestIris3)
```

```
Call:
 randomForest(formula = Species ~ Petal.Width + Petal.Length,      data = iris, prox = TRUE, ntree = 50
               Type of random forest: classification
                     Number of trees: 600
No. of variables tried at each split: 1
```

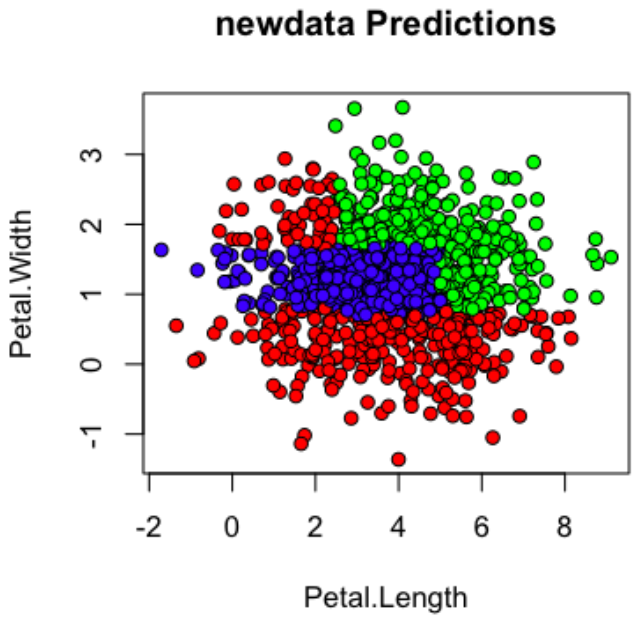
Predicting new values

```
newdata <- data.frame(Sepal.Length<- rnorm(1000,mean(iris$Sepal.Length),
                                           sd(iris$Sepal.Length)),
                      Sepal.Width <- rnorm(1000,mean(iris$Sepal.Width),
                                           sd(iris$Sepal.Width)),
                      Petal.Width <- rnorm(1000,mean(iris$Petal.Width),
                                           sd(iris$Petal.Width)),
                      Petal.Length <- rnorm(1000,mean(iris$Petal.Length),
                                            sd(iris$Petal.Length)))

pred <- predict(forestIris,newdata)
```

Predicting new values

```
plot(newdata[,4], newdata[,3], pch=21, xlab="Petal.Length",ylab="Petal.Width",
bg=c("red", "blue", "green")[as.numeric(pred)],main="newdata Predictions")
```



Notes and further resources

Notes:

- Bootstrapping is useful for nonlinear models
- Care should be taken to avoid overfitting (see rfcv funtion)
- Out of bag estimates are efficient estimates of test error

Further resources:

- Random forests
- Random forest Wikipedia
- Bagging
- Bagging and boosting