Week 7

Smoothing

for observing non-linear trends - and for being able to predict those trends in other observations

it tries to fit a curvy smooth line -

but there is a risk of overfitting and it's hard to interpret.

in R -

filter - smooths data

as.vector=(filter(cd4$time , filter = rep(1,200))/200)

averaging - is smoothing (or weighted sum)

"moving average"

can use weight

sum of weight should be 1

the closer to the middle, the more weight

Lowess (loess) ← locally-weighted scatter-plot smoothing

lw1 = loess(cd4 ~ time, data = c4)

lines(time, lw1$fitted, ...)

span - number of points used when smoothing

span = 0.1 - 0.1 of data is used for calculating

0.25 - a quarted of the date is used

as the span increases, the line becomes more smooth

by default, loess calculates the span automatically

predicting:

pred1 = predict(lw1, new.data = ..., se = T)

↑ calculates the standart error

Splines

$$Y_i = b_0 + \sum_{k=1}^{k} b_k s_k(x_i) + e_i$$

Y_i - outcome of the observation

b_0 - intercept term

b_k - coefficient for k^{th} spline function

x_i - covariate for i^{th} observation

e_i - error

```
library(splines)
ns1 = ns(cd4$time, df=3)
```

natural cubic splines

degrees of freedom (number of functions to be applied to the variable)

```
lm1 = lm(cd4 ~ ns1)
summary(lm1)
```

splain matrix

Notes:

cross-validation is important

The bootstrap

for estimating standart errors
for improving predictions

key idea:

- treat the sample as population – and then perform analysis as if you taken a sample from population

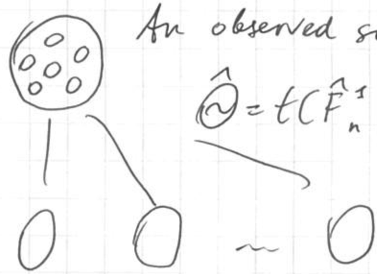
good for

- calculating standart errors
- forming confidence intervals
- performing hypothesis testing
- improving predictors

The „central dogma" of statistics

$$\underbrace{\hat\theta^1 = t(\hat F^1) \quad \hat\theta^2 = t(\hat F^2) \quad \hat\theta^\infty = t(\hat F^\infty)}_{\text{estimates of } \theta = t(F)}$$

bootstrap



An observed sample

$\hat{\Theta} = t(\hat{F}_n^1)$

take repeated samples from

and treat each sample as if it was a sample from the population

$\hat{\Theta}^1 = t(\hat{F}^1)$ $\hat{\Theta}^2 = t(\hat{F}^2)$ ~ $\hat{\Theta}^\infty = t(\hat{F}^\infty)$

thos idea is quite powerful and works wery well

```
bootMean = boot( x, meanFunc, 1000 )
```

original data — number of iterations

```
meanFunc = function (x, i) {
    mean( x[i])
}
```

dataset — set of indexes

```
bootMean$t
```

for calculating confidence intervals

library (boot)   data (nuclear)

nuke. lm = lm (log (cost) ~ date, data = nuclear)

without making assumptions about normal distribution of the data set or any other assumptions

results = boot (data = nuclear, statistics = bs, R = 1000, formula = log(cost) ~ date)

number of iterations

bs = function (data, indices, formula)

d = data [indices,]
fit = lm (formula, data = data
return (coef (fit))
}

Combination of boot and bs function generates 1000 linear fits where we resample the data set with replacements boot.ci(results)

↑

bootstrap confidence intervals

Bootstrapping from a model

resid = rstudent(nuke.lm) .. calculate
Student's residuals

fit = fitted(lm(log(cost) ~ 1,
data = nuclear)) ← fitted value if
we only included
the intercept term
(when you fit a straight
line to the data)

newNuc = cbind(nuclear, resid = resid,
fit0 = fit0)

bs = function(data, indices) {

coef(glm($\overbrace{\text{data\$fit0 + data\$resid[indices]}}^{\text{outcome}}$ ~
data$date, data = data)

}

results = boot(data = newNuc, statistic = bs,
R = 1000)

Things you can't bootstrap:

- max

notes

- useful for complicated statistics
- careful near boundaries
- careful with non-linear functions

resources:

- ~~An introduction to the bootstrap, from the slide 2~~

Bootstrapping for predictions

dates between 65 and 72

newdata = data.frame(date = seq(65, 72, length = 100))
nuclear = cbind(nuclear, resid = rstudent(nuke.lm), fit = fitted(nuke.lm))

residual value from nuke.lm

fitted value from nuke.lm

nuke.boot = boot(nuclear, nuke.fun, R = 1000, newdata = newdata)

```
nuke.fun = function (data, inds, newdata) {
```
$\ll$ randomised bootstrap indices

```
lm.b = lm(fit + resid[inds] ~ date, data = data)
```
$\uparrow$ bootstrap-sampled noise

it tries to generate the data that has the same trend observed in the original dataset

```
pred.b = predict(lm.b, newdata)
```
$\uparrow$ predicting values for new data

```
return (pred.b)
}
```

nuke.boot$t – 100 replications for each of values from 65 to 72 (in columns)

```
pred = predict(nuke.lm, newdata)
```

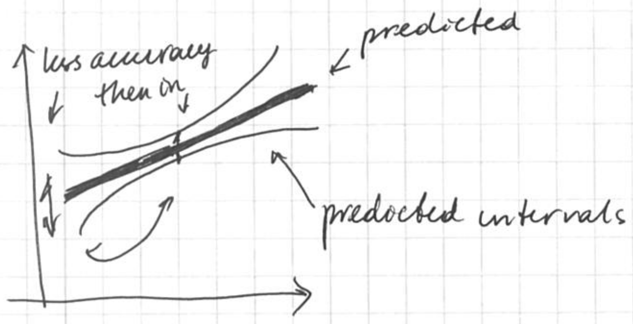
our predicted value for new data set

```
predStds = apply(nuke.boot$t, 2, sd)
```

standart deviations for each of those 100 values

```
plot(newdata$date, col = "black", ylim = c(0,10))

lines(newdate$date, pred + 1.96 * predStds, col = "red")
lines(newdata$date, pred - 1.96 * predStds, col = "red")
```



Bootstrap aggregating

for improving prediction accuracy

basic ideas

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

(have a look at wiki)

it reduces variances
and more useful for non-linear functions

```
library(ElemStatLearn)
data(ozone, package="ElemStatLearn")

ozone = ozone[order(ozone$ozone),]

ll = matrix(NA, nrow=10, ncol=155)        Bagged loess
for (i in 1:10) {

    ss = sample(1:dim(ozone)[1], replace=T)

    ozone0 = ozone[ss,]  ← subsample
    loess0 = loess(temperature ~ ozone,
                   data=ozone, span=0.2)

    ll[i,] = predict(loess0, newdata=
                data.frame(ozone=1:155))

}
```

```
lines(1:155, apply(ll, 2, mean), col = "red")
```

↑ bagged loess line

Bagged trees

basic idea

- resample data
- recalculate tree
- average/mode of predictors

- more stable
- may not be as good as random forests

```
library(ipred)
bagTree = bagging(Species ~., coob = T)
```

↑ some values are left out

↓ returns a bunch of trees with different classifications

Random forests

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

pros

- accuracy

cons

- speed
- interpretability
- overfitting

```
library(randomForest)

forestIris = randomForest(Species ~ Petal.Width, +
             Petal.Length, data=iris, prox=T )
```

getTree(forestIris, k=2) – returns a tree

```
iris.p = classCenter(iris[, c(3,4)],
         iris$Species, forestIris$prox)
```

↓

draws centers of clusters

combine (...)

↑

combines random forests into one

prediction: the same

predict(forestIris, newdata)

Notes:

- bootstrapping is useful for non-linear models
- care should be taken to avoid overfitting
- out of bag estimates are efficient estimates of test error

Combining Predictors

key ideas

- you can combine classifiers by averaging/voting

$\Downarrow$

improves accuracy
but reduces interpretability

Basic intuition

Suppose we have 5 completely independend classifiers

if accuracy is 70% for each

$$10 \times (0.7)^3 (0.3)^2 + 5 \cdot (0.7)^4 (0.3)^1 + (0.7)^5 =$$

10 ways when 3 classifiers are right and 2 wrong

5 ways when 4 right and 1 wrong

= 83.7% accuracy

which is higher than accuracy of any of the individuals

with 101 independent => 99.9%

Approaches for combining:

- bagging
- boosting
- combining (in weighted / unweighted fasion) different classifiers — avg or majority

combine1 <- predict(lm1, data = testData)/2 +
+ predict(tree1, data = testData)/2