STAT 621 Lecture Notes
Some Numerical Methods

Before we continue our discussion of classical distribution-free approachs to statistical inference, let's take a look at some interesting and useful numerical methods. These share some similarities with classical techniques, but their practical use has developed largely due to increases in computing power. They are also generalizable to many situations and as we will see, are used in modern approaches as well. Specifically, we'll look at randomization and permutation tests, the bootstrap, the jackknife and cross-validation. These are often refered to as **resampling methods**.

Permutation and Randomization Tests

Permutation and randomization tests are related methods for testing hypotheses. Here the distribution of a test statistic under the null hypothesis is determined without making any distributional assumptions about the population of interest. The two terms are often used interchangeably, but a general difference is

Example: The Lady Tasting Tea. This famous example by Fisher was presented in his 1935 book *The Design of Experiments* and was one of the first examples of hypothesis testing. The resulting test has become known as **Fisher's Exact Test**. Apparently a British lady had boasted of her ability to detect whether a cup of tea had been prepared by pouring tea before milk, or by pouring milk before tea. Fisher described an experiment where he presented the lady with 8 cups of tea in random order. He told her that 4 of these were poured milk-first and 4 were poured tea-first. After tasting each cup the lady made her determination.

(1) What are the hypotheses of interest?

(2) What would the data look like if the lady were able to discriminate perfectly?

(3) Suppose the test statistic is the number of milk-first cups she identifies correctly. Derive the null distribution.

(4) Suppose the lady guessed four correctly. What would you decide? Three correctly?

The R function `fisher.test` can be used to test the null of independence of rows and columns in a contingency table with *fixed marginals*. Supposedly the lady had correctly guessed three of the four cups that had been poured milk-first.

```
> TeaTasting <-matrix(c(3, 1, 1, 3),nrow = 2,
+              dimnames = list(Guess = c("Milk", "Tea"),Truth = c("Milk", "Tea")))

> TeaTasting
      Truth
Guess  Milk Tea
  Milk    3   1
  Tea     1   3

> fisher.test(TeaTasting, alternative = "greater")

Fisher's Exact Test for Count Data
p-value = 0.2429
alternative hypothesis: true odds ratio is greater than 1
95 percent confidence interval:
 0.3135693       Inf
sample estimates: odds ratio
  6.408309
```

Example: (Keuhl, *Design of Experiments: Statistical Research Design and Analysis*, 2nd ed., pp. 22–23). Here is a permutation in a slightly different context. An experiment has two treatments randomly assigned to seven subjects. Four subjects receive treatment A and three receive treatment B. The following data are collected.

Subject	1	2	3	4	5	6	7
Treatment	B	B	A	A	B	A	A
Response	14	16	19	17	15	13	17

Interest is in determining if there is a difference in treatment means. That is, we test $H_0 : \mu_A = \mu_B$. Under the null the response does not depend on the treatment so that, for example, the response for Subject 1 would be 14 regardless of the treatment received.

(1) What would a reasonable test statistic be here? Describe the null distribution given the observed responses.

(2) The null distribution of the test statistic $T = \overline{Y}_A - \overline{Y}_B$ is given below. Show how this was computed.

t	-3.17	-2.58	-2.0	-1.42	-0.83	-0.25	0.33	0.92	1.50	2.08	2.67	3.25
$P(T = t)$	.029	.057	.057	.114	.114	.143	.114	.143	.089	.089	.029	.029

(3) Test for a difference in treatment means. What is the p-value?

The previous example shows that determining the exact null distribution for a permutation test can be quite tedious, especially if the sample size is large. When that's the case, we can approximate this distribution by sampling from the possible randomizations of the data, rather than enumerating them all. This leads the the version of the procedure known as the *randomization test*.

```
> response=c(14, 16, 19, 17, 15, 13, 17)
> treatment=c("B","B","A","A","B","A","A")
#------------------------------------------------
# Now randomly assign 4-As and 3-Bs to responses
# calculate mean difference
# repeat many times and save all mean diffs
# this approximates the null distribution
#---------------------------------------------------
# here's the procedure illustrated once (note sample and tapply functions)

> newtreatment=sample(treatment, size=7, replace=F)
> newtreatment
[1] "A" "B" "A" "A" "B" "A" "B"

> a=tapply(response, newtreatment, mean)
> a
    A     B
15.75 16.00

> a[1]-a[2]
    A
-0.25
# ---------------------------------------------------
# Here's a loop to repeat the procedure many times
#  each time save the difference in means
#---------------------------------------------------
> nreps=100
> diffs=rep(NA, nreps)

> for (i in 1:nreps)
+ {
+ newtreatment=sample(treatment, size=7, replace=F)
+ a=tapply(response, newtreatment, mean)
+ diffs[i]=a[1]-a[2]
+ diffs
+ }
#-----------------------------------------------------
# Here's the approximate distribution!

> table(round(sort(diffs),2))/nreps

-3.17  -2.58    -2  -1.42  -0.83  -0.25   0.33   0.92   1.5   2.08   2.67   3.25
 0.03   0.07  0.04   0.11   0.12   0.14   0.11   0.13   0.12  0.10   0.02   0.01
```

What's the approximate p-value?

The Bootstrap

Like the randomization test, the bootstrap is a resampling method. Here, however, the objective is not to test a hypothesis, but rather to approximate the sampling distribution of a statistic. This allows one to estimate the standard error of a statistic or compute confidence intervals for a parameter without making assumptions about the true distribution of the data. One can also use the bootstrap to approximate the sampling distribution of a test statistic under a null hypothesis, and use that to perform a test, similar to the randomization test.

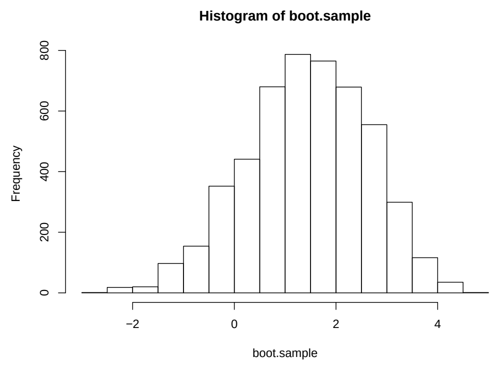
Here's the basic idea: Recall that the sampling distribution of a statistic describes its properties and behavior under repeated sampling. In reality we never actually repeat the sampling to find this distribution, we usually just work from assumptions made on the distribution of the population we sampled from. Instead of making assumptions, the bootstrap actually does create new samples. But these samples are generated by resampling the values in the single observed data set. That is, we sample from the *empirical distribution* of the data.

Let's write out the steps for creating a boostrap sample of a statistic:

How is this bootstrap sample used?

Example: In this example we will approximate the sampling distribution of the difference in means for the experiment described earlier (page 3). An important thing to keep in mind with the bootstrap is that the resampling should mimic the way the data were sampled originally. Here subjects were separated into two treatment groups. So we will resample separately from the two groups.

```
# separate data into two vectors
A=response[treatment=="A"]
B=response[treatment=="B"]

# The bootstrap loop
nreps=5000
boot.sample <- rep(0,nreps)
    for(i in 1:nreps)
    {
    newA <- sample(A,replace=T)
    newB <- sample(B,replace=T)
    boot.sample[i] <- mean(newA)-mean(newB)
    }

# look at some summaries
hist(boot.sample)

mean(boot.sample)
 [1] 1.516433

sd(boot.sample)
 [1] 1.192112

quantile(boot.sample ,c(.025,.975))
      2.5%     97.5%
   -1.000000  3.666667
```



Some discussion: Bias correction, complicated resampling, parametric bootstrap....

The Jackknife

The jackknife is another type of resampling method. It's mainly used to estimate the standard error and bias of an estimator. Here we don't randomly generate new samples. Instead we compute the value of an estimator many times by sequentially leaving observations out of the data set.

Let's write out the steps for the jackknife procedure.

How is this used to estimate variance and bias?

A little intuition about why the bias estimator works.

Example: We will use the jackknife to estimate the bias and standard deviaion of the *coefficient of variation*, defined as

$$\theta = \frac{\sigma}{\mu}$$

Consider a sample $X_1, \dots, X_{12}$ from a Chisquare(8) distribution. Consider estimating θ using $\widehat{\theta} = s/\overline{x}$. What is the true value of θ in this case? Could we figure out the actual bias and standard deviation of $\widehat{\theta}$?

Below I simulate a random sample and compute $\widehat{\theta}$.

```
> n=12
> df=8
> x=rchisq(n, df)
> theta.hat=sd(x)/mean(x)
> theta.hat
     [1] 0.4120635
```

This next chunk of code has a loop that sequentially removes each data point and computes the leave-one-out estimators $\widehat{\theta}_{-i}$. This jackknife sample is used to compute the jackknife estimator, estimate variance and estimate bias.

```
> Jsample=rep(NA, n)
> for (i in 1:n)
   {
   temp=x[-i]
   Jsample[i]=sd(temp)/mean(temp)
   }

> sort(Jsample)
 [1] 0.3110576 0.3762330 0.3774318 0.4065952 0.4193006 0.4216740 0.4318004
 [8] 0.4331242 0.4345554 0.4345685 0.4352425 0.4352442

> theta.jack=mean(Jsample)
> se.jack=sqrt((n-1)*sum((Jsample-theta.jack)^2)/n)
> bias.jack=(n-1)*(theta.jack-theta.hat)
> theta.corrected=theta.jack-bias.jack
> cbind(theta.hat, theta.jack, se.jack, bias.jack, theta.corrected)

     theta.hat theta.jack   se.jack  bias.jack theta.corrected
[1,] 0.4120635  0.4097356 0.1200227 -0.0256064        0.435342
```

Comments?

The R package `bootstrap` has a jackknife function that will do calculations for you. The syntax is `jackknife(x, fun)` where x is the data used to compute an estimator defined in the function `fun`. This can be a standard R function or one that you write yourself. R doesn't have a CV function, so I create one below. Let's check our work above:

```
> library(bootstrap)
> cv.fun=function(x)
+ {
+   cv=sd(x)/mean(x)
+   cv
+ }
>
> jackknife(x,cv.fun)
$jack.se
[1] 0.1200227

$jack.bias
[1] -0.0256064

$jack.values
 [1] 0.4193006 0.3110576 0.4331242 0.3774318 0.4352425 0.3762330 0.4345685
 [8] 0.4318004 0.4345554 0.4352442 0.4216740 0.4065952
```

Cross-Validation

Cross-validation is used to evaluate prediction error in statistical modeling. Here the objective is to build a model that relates a response Y to one or more predictors $X_1, \dots, X_p$.

As a motivating example, consider a simple linear regression model,

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

The least squares estimators of β_0 and β_1 are selected to mininize what quantity?

Suppose however that the primary interest is in predicting a new value of the response, Y_{n+1} from a new observation X_{n+1}. Is there a better quantity we might think about minimizing?

Cross-validation attempts to minimize this *prediction error*. There are a number of variations to the basic idea. Here are the basic steps.

(1) Divide the data into the **Training Set** and the **Validation Set**.

(2) Fit the model on the Training Set.

(3) Measure prediction error on the Validation Set.

(4) Select model with lowest prediction error.

Some different ways to do this include the following. Note that this makes the most sense when we are trying to select a best model to predict new observations. Competing models need to be organized in some logical way. For example we might be looking to determine the best degree polynomial, or looking for the optimal value of a "tuning" parameter.

- Split data in half:

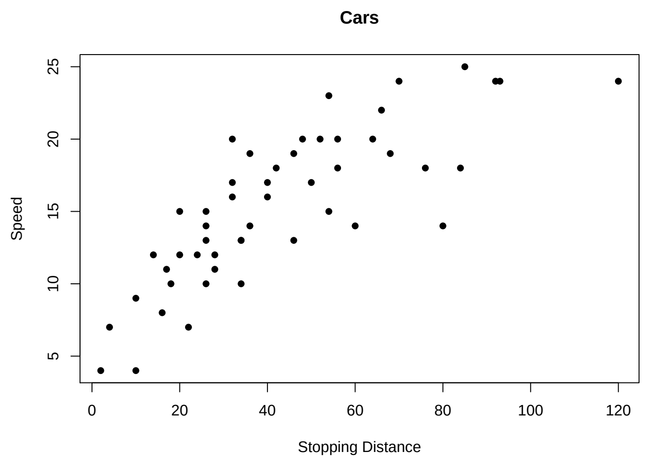
- Leave-one-out cross-validation:

- K-fold cross-validation:

<u>**Example:**</u> (From the R-Bloggers website). This example illustrates K-fold cross validation using the R package `boot` and the R data set `cars`. The function `cv.glm` computes the K-fold prediction error for models that are fit by `glm`. **Note:** It's not hard to write your own code to perform cross-validation – give it a try!

First load packages and make a plot of the variables Y=speed and X=stopping dist. This seems a little backwards, but this way makes a better example.

```
require(boot)     #loads package, same a library command
library(MASS)
plot(speed~dist, data=cars, main = "Cars", xlab = "Stopping Distance", ylab = "Speed", pch=16)
```

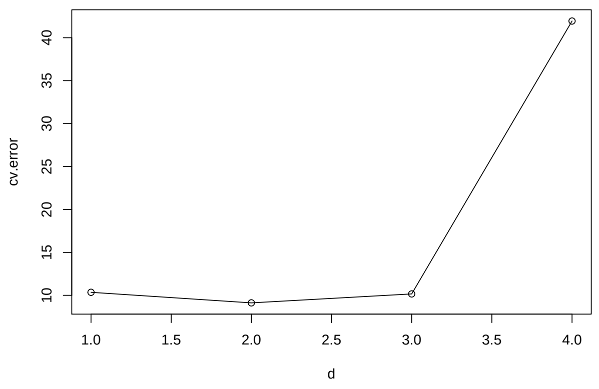


First we fit the SLR model and compute the 5-fold CV measure.

```
glm.fit1 = glm(speed~dist, data=cars)
cv.error1=cv.glm(cars, glm.fit1, K=5)
cv.error1$delta
  [1] 10.35175  10.26411                 # first one is usual pred error
```

A SLR model is probably not the best fit here. Let's use 5-fold cross-validation to pick the best degree polynomial:

```
glm.fit2 = glm(speed~poly(dist,2), data=cars)    # quadratic model
cv.error2=cv.glm(cars, glm.fit2, K=5)

glm.fit3 = glm(speed~poly(dist,3), data=cars)    # cubic model
cv.error3=cv.glm(cars, glm.fit3, K=5)

glm.fit4 = glm(speed~poly(dist,4), data=cars)    # quartic model
cv.error4=cv.glm(cars, glm.fit4, K=5)

cv.error=c(cv.error1$delta[1],cv.error2$delta[1],cv.error3$delta[1],cv.error4$delta[1])
d=c(1,2,3,4)
plot(d,cv.error)
lines(d,cv.error)
```



What model would you pick? Why the big jump at $d = 4$?