Computational Statistics / Resampling Inference

Eli Gurarie

StatR 301 - Lecture 1
University of Washington - Seattle

April 2, 2013

Outline

- Review of inference and hypothesis testing
- Randomization and Permutation Tests
- The bootstrap
- Timing and profiling your code

The Question:

WHICH IS BIGGER?

Seed ant (*Pogonomyrmex salinus*)

Thatch ant (*Formica planipilis*)

Data

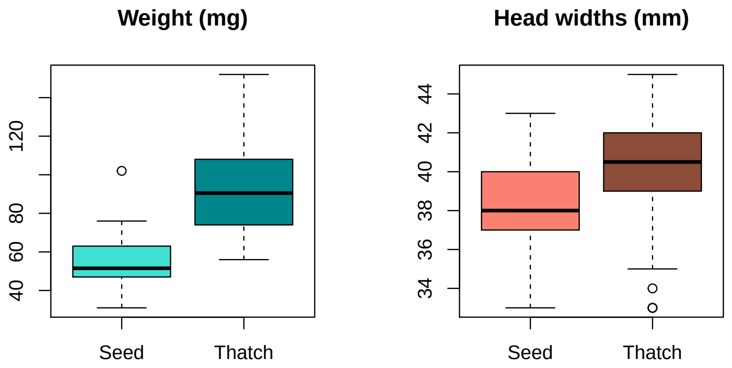
```
Ant <- read.csv("./data/AntSample.csv")
data.frame(subset(Ant[,c(1,4,5)], Species=="Seed"),
           subset(Ant[,c(1,4,5)], Species=="Thatch"))[1:20,]

##    Species Weight Headwidth1 Species.1 Weight.1 Headwidth1.1
## 1     Seed     51         38    Thatch       90           39
## 2     Seed     55         41    Thatch      104           45
## 3     Seed     53         37    Thatch      106           40
## 4     Seed     48         35    Thatch       57           34
## 5     Seed     31         33    Thatch       90           43
## 6     Seed     72         39    Thatch      132           40
## 7     Seed     45         37    Thatch       91           42
## 8     Seed     65         40    Thatch      110           42
## 9     Seed     50         38    Thatch       86           41
## 10    Seed    102         43    Thatch      152           45
## 11    Seed     57         40    Thatch       74           38
## 12    Seed     38         39    Thatch       58           33
## 13    Seed     67         38    Thatch       71           33
## 14    Seed     57         37    Thatch       79           39
## 15    Seed     76         43    Thatch       67           35
## 16    Seed     67         40    Thatch      112           44
## 17    Seed     43         37    Thatch      103           41
## 18    Seed     50         38    Thatch       61           41
## 19    Seed     35         36    Thatch      141           42
## 20    Seed     65         39    Thatch       81           39
```

note: data taken from http://www.stat.ucla.edu/datasets/

Step 2: Visualize Data

```r
col <- c("turquoise", "turquoise4", "salmon", "salmon4")
par(mfrow = c(1, 2))
boxplot(Weight ~ Species, main = "Weight (mg)", col = col[1:2], data = Ant)
boxplot(Headwidth1 ~ Species, main = "Head widths (mm)", col = col[3:4], data = Ant)
```



Step 3: Summary Statistics

Weight:

```
tapply(Ant$Weight, Ant$Species, function(x) c(mean = mean(x), sd = sd(x)))

## $Seed
##  mean    sd
## 54.67 13.98
##
## $Thatch
##  mean    sd
## 92.80 25.96
```

Head width:

```
tapply(Ant$Headwidth1, Ant$Species, function(x) c(mean = mean(x), sd = sd(x)))

## $Seed
##   mean     sd
## 38.500  2.271
##
## $Thatch
##   mean     sd
## 39.700  3.495
```

A tough question

A tough question

- It certainly *seems* like Thatch ants *might* to be bigger than Seed ants.

A tough question

- It certainly *seems* like Thatch ants *might* to be bigger than Seed ants.
- But there are obviously *some* Seed ants that are bigger than *some* Thatch ants!

A tough question

- It certainly *seems* like Thatch ants *might* to be bigger than Seed ants.
- But there are obviously *some* Seed ants that are bigger than *some* Thatch ants!
- What does the question "Which is Bigger?" actually mean?

A tough question

- It certainly *seems* like Thatch ants *might* to be bigger than Seed ants.
- But there are obviously *some* Seed ants that are bigger than *some* Thatch ants!
- What does the question "Which is Bigger?" actually mean?

The short answer

- It doesn't mean anything stated simply... We need to refine the question!

Refining the question...

E.g: what is the probability that *any given* Thatch Ant is bigger than *any given* Seed Ant?

Refining the question...

E.g: what is the probability that *any given* Thatch Ant is bigger than *any given* Seed Ant?

Introduce some crazy comprehensive comparison statistic:

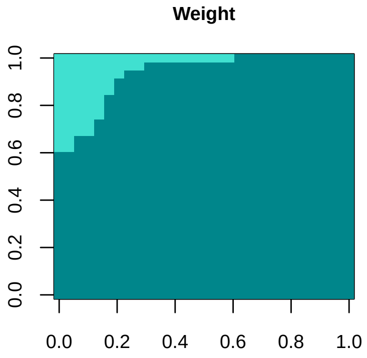
$$C_w = \frac{1}{N_t N_s} \sum_{i=1}^{N_t} \sum_{j=1}^{N_s} I(Wt_i > Ws_j) = \frac{872}{900} = 0.92$$

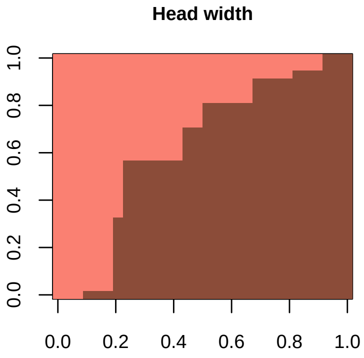
$$C_h = \frac{1}{N_t N_s} \sum_{i=1}^{N_t} \sum_{i=1}^{N_s} I(Ht_i > Hs_j) = \frac{559}{900} = 0.62$$

Visualize!

```r
W.s <- subset(Ant, Species=="Seed")$Weight
W.t <- subset(Ant, Species=="Thatch")$Weight
W.M <- outer(sort(W.t), sort(W.s), ">")
image(W.M, main="Weight", xlab="Seed Ants",
      ylab="Thatch Ants", col=col[1:2])
```

```r
H.s <- subset(Ant, Species=="Seed")$Headwidth2
H.t <- subset(Ant, Species=="Thatch")$Headwidth2
H.M <- outer(sort(H.t), sort(H.s), ">")
image(H.M, main="Head width", xlab="Seed Ants",
      ylab="Thatch Ants", col=col[3:4])
```





```r
sum(W.M > 0)/prod(table(Ant$Species))

## [1] 0.9139
```

```r
sum(H.M > 0)/prod(table(Ant$Species))

## [1] 0.6811
```

Getting there, but more questions!

The statement that **A** is bigger than **B** *about* **X** % of the time is an improvement ... But how do we know that this comparison isn't an artifact of random sampling?

The short answer

Again, there is no short answer. We need to rely on the framework of *hypothesis testing* - in which we assume a simpler version of reality, and see what the probability of our data is if the added *structure* (model) were not there.

Review of Hypothesis Testing.

1. Since it can be tricky to even define what it is we want to know, we define *it's opposite*, which is often simpler. This is the **null hypothesis**(H_0).

Review of Hypothesis Testing.

1. Since it can be tricky to even define what it is we want to know, we define *it's opposite*, which is often simpler. This is the **null hypothesis**(H_0).
2. What the **null hypothesis** *isn't* we call the **alternative hypothesis** (H_1 or H_A).

Review of Hypothesis Testing.

1. Since it can be tricky to even define what it is we want to know, we define *it's opposite*, which is often simpler. This is the **null hypothesis**(H_0).
2. What the **null hypothesis** *isn't* we call the **alternative hypothesis** (H_1 or H_A).
3. We choose some summary of the data called the **test statistic** ($\hat{\theta} \sim f(t)$).

Review of Hypothesis Testing.

1. Since it can be tricky to even define what it is we want to know, we define *it's opposite*, which is often simpler. This is the **null hypothesis**(H_0).
2. What the **null hypothesis** *isn't* we call the **alternative hypothesis** (H_1 or H_A).
3. We choose some summary of the data called the **test statistic** ($\hat{\theta} \sim f(t)$).
4. We create/compute/generate a **null distibution** of the test statistic... i.e. the distribution we would expect of the test statistic if the null hypothesis were true.

Review of Hypothesis Testing.

1. Since it can be tricky to even define what it is we want to know, we define *it's opposite*, which is often simpler. This is the **null hypothesis**(H_0).
2. What the **null hypothesis** *isn't* we call the **alternative hypothesis** (H_1 or H_A).
3. We choose some summary of the data called the **test statistic** ($\hat{\theta} \sim f(t)$).
4. We create/compute/generate a **null distibution** of the test statistic... i.e. the distribution we would expect of the test statistic if the null hypothesis were true.
5. We calculate the observed value of the **test statistic**, $\hat{\theta}^*$, and compare it to our distribution.

Review of Hypothesis Testing.

1. Since it can be tricky to even define what it is we want to know, we define *it's opposite*, which is often simpler. This is the **null hypothesis**(H_0).
2. What the **null hypothesis** *isn't* we call the **alternative hypothesis** (H_1 or H_A).
3. We choose some summary of the data called the **test statistic** ($\hat{\theta} \sim f(t)$).
4. We create/compute/generate a **null distibution** of the test statistic... i.e. the distribution we would expect of the test statistic if the null hypothesis were true.
5. We calculate the observed value of the **test statistic**, $\hat{\theta}^*$, and compare it to our distribution.
6. We set some criterion, often called the **critical region**, within which we would *fail to reject* (not quite the same as "accept") the **null hypothesis**. Here, two things can happen:

Review of Hypothesis Testing.

1. Since it can be tricky to even define what it is we want to know, we define *it's opposite*, which is often simpler. This is the **null hypothesis**(H_0).
2. What the **null hypothesis** *isn't* we call the **alternative hypothesis** (H_1 or H_A).
3. We choose some summary of the data called the **test statistic** ($\hat{\theta} \sim f(t)$).
4. We create/compute/generate a **null distibution** of the test statistic... i.e. the distribution we would expect of the test statistic if the null hypothesis were true.
5. We calculate the observed value of the **test statistic**, $\hat{\theta}^*$, and compare it to our distribution.
6. We set some criterion, often called the **critical region**, within which we would *fail to reject* (not quite the same as "accept") the **null hypothesis**. Here, two things can happen:
   1. If $\hat{\theta}^*$ is "extreme" (lies outside our critical region), we reject the null hypothesis, accept the alternative hypothesis, humbly acknowledging that we *might* be wrong, and call the probability that we might be wrong the **Type I error**.

Review of Hypothesis Testing.

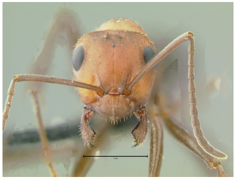
1. Since it can be tricky to even define what it is we want to know, we define *it's opposite*, which is often simpler. This is the **null hypothesis**(H_0).
2. What the **null hypothesis** *isn't* we call the **alternative hypothesis** (H_1 or H_A).
3. We choose some summary of the data called the **test statistic** ($\hat{\theta} \sim f(t)$).
4. We create/compute/generate a **null distibution** of the test statistic... i.e. the distribution we would expect of the test statistic if the null hypothesis were true.
5. We calculate the observed value of the **test statistic**, $\hat{\theta}^*$, and compare it to our distribution.
6. We set some criterion, often called the **critical region**, within which we would *fail to reject* (not quite the same as "accept") the **null hypothesis**. Here, two things can happen:
    1. If $\hat{\theta}^*$ is "extreme" (lies outside our critical region), we reject the null hypothesis, accept the alternative hypothesis, humbly acknowledging that we *might* be wrong, and call the probability that we might be wrong the **Type I error**.
    2. If $\hat{\theta}^*$ is not "extreme", we *fail to reject* the null hypothesis, calling the probability that we *might* be wrong the **Type II error**.

Example: Step 1-2

Null and Alternative Hypotheses

$=$

H_0: Seed and Thatch ants can be considered to come from the "same" population.

$\neq$ 

H_A: Seed and Thatch ants come from different sized populations.

Example: Step 3

Choose some test statistic

We *could* do something crazy, like the count statistic:

$$C_w = \frac{1}{N_t N_s} \sum_{i=1}^{N_t} \sum_{j=1}^{N_s} I(W_{ti} > W_{sj})$$
$$C_h = \frac{1}{N_t N_s} \sum_{i=1}^{N_t} \sum_{i=1}^{N_s} I(H_{ti} > H_{sj})$$

Example: Step 3

Choose some test statistic

We *could* do something crazy, like the count statistic:

$$C_w = \frac{1}{N_t N_s} \sum_{i=1}^{N_t} \sum_{j=1}^{N_s} I(W_{ti} > W_{sj})$$
$$C_h = \frac{1}{N_t N_s} \sum_{i=1}^{N_t} \sum_{i=1}^{N_s} I(H_{ti} > H_{sj})$$

Or something simpler... like the difference between the sample means:

$$D_W = \overline{W_t} - \overline{W_s}$$
$$D_H = \overline{H_t} - \overline{H_s}$$

Example: Step 4

Obtain null-distribution of the test statistic

- If the null hypothesis is true, then there is **no** difference between the two groups means we can resample them in any which way
- Randomization Test:
  1. shuffle all weights W

Example: Step 4

Obtain null-distribution of the test statistic

- If the null hypothesis is true, then there is **no** difference between the two groups means we can resample them in any which way
- Randomization Test:
  1. shuffle all weights W
  2. split randomly into two new vectors: $W_{S.sim}$ and $W_{T.sim}$

Example: Step 4

Obtain null-distribution of the test statistic

- If the null hypothesis is true, then there is **no** difference between the two groups means we can resample them in any which way
- Randomization Test:
    1. shuffle all weights W
    2. split randomly into two new vectors: $W_{S.sim}$ and $W_{T.sim}$
    3. obtain and store the statistic $D_{W.sim} = \overline{W_{T.sim}} - \overline{W_{T.sim}}$

Example: Step 4

Obtain null-distribution of the test statistic

- If the null hypothesis is true, then there is **no** difference between the two groups means we can resample them in any which way
- Randomization Test:
  1. shuffle all weights W
  2. split randomly into two new vectors: $W_{S.sim}$ and $W_{T.sim}$
  3. obtain and store the statistic $D_{W.sim} = \overline{W_{T.sim}} - \overline{W_{T.sim}}$
  4. Repeat steps 1-3 a bunch of times.

Example: Step 4

Obtain null-distribution of the test statistic

- If the null hypothesis is true, then there is **no** difference between the two groups means we can resample them in any which way
- Randomization Test:
  1. shuffle all weights W
  2. split randomly into two new vectors: $W_{S.sim}$ and $W_{T.sim}$
  3. obtain and store the statistic $D_{W.sim} = \overline{W_{T.sim}} - \overline{W_{T.sim}}$
  4. Repeat steps 1-3 a bunch of times.
- Repeat for Heights

Randomization in R

Obtain observed statistic:

```
getD.means <- function(Y, X) mean(Y[X == levels(X)[1]] - Y[X == levels(X)[2]])
(D.weight.obs <- getD.means(Ant$Weight, Ant$Species))

## [1] -38.13

(D.head.obs <- getD.means(Ant$Headwidth2, Ant$Species))

## [1] -0.05047
```

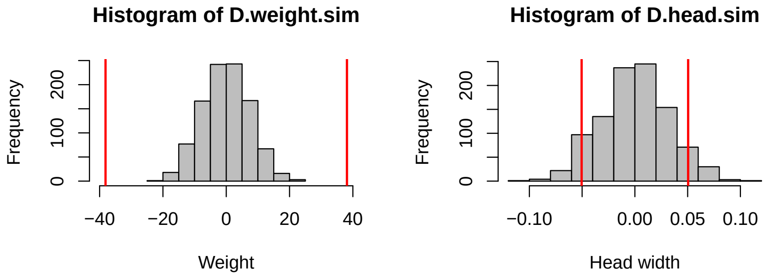
Repeat many times, resampling the response (or the covariate) each time:

```
nreps <- 1000
D.weight.sim <- rep(0, nreps)
for (i in 1:nreps) D.weight.sim[i] <- getD.means(Ant$Weight, sample(Ant$Species))
D.head.sim <- rep(0, nreps)
for (i in 1:nreps) D.head.sim[i] <- getD.means(Ant$Headwidth2, sample(Ant$Species))
```

The above is the most basic template for a randomization test ... we will talk about some other ways to code this later.

Visualize null-distribution and compute p-value

```r
hist(D.weight.sim, col = "grey", xlim = c(-40, 40), xlab = "Weight")
abline(v = c(-1, 1) * D.weight.obs, col = "red", lwd = 2)
hist(D.head.sim, col = "grey", xlab = "Head width")
abline(v = c(-1, 1) * D.head.obs, col = "red", lwd = 2)
```



Obtain randomization p-values $= 2\Pr(\theta > |\widehat{\theta*}|)$

```r
c(p.weight = 2 * sum(D.weight.sim > abs(D.weight.obs))/nreps, p.height = 2 *
    sum(D.head.sim > abs(D.head.obs))/nreps)

## p.weight p.height
##    0.000    0.134
```

What conclusions do we make based on these p-values?

R-Programming: Passing a function as a variable

Remember, you can always pass a function into a function. This makes it very easy to perform randomization tests on ANY statistic. Don't forget the powerful ellipsis either: "..."

```
getD.fun <- function(Y, X, fun, ...) fun(Y[X == levels(X)[1]] - Y[X == levels(X)[2]],
    ...)

getD.fun(Ant$Weight, Ant$Species, sd)

## [1] 27.14

getD.fun(Ant$Weight, Ant$Species, quantile)

##      0%     25%     50%     75%    100%
## -106.00  -55.25  -41.00  -17.75    9.00

getD.fun(Ant$Weight, Ant$Species, quantile, 0.75)

##    75%
## -17.75
```

R-Programming: the mosaic package and do()

The mosaic package - which is designed to provide some easy-to-use tools for randomization - makes naive loops more efficient with a funny little function called do():

```r
require(mosaic)
do(5) * getD.fun(Ant$Weight, sample(Ant$Species), quantile)

##      0%    25%  50%   75% 100%
## 1  -60 -21.50 -6.0  8.50   50
## 2  -38 -18.50 -2.5 15.50   52
## 3  -74 -13.50  4.0 18.50   60
## 4  -74 -11.50  3.0 33.50   62
## 5 -101 -25.75 -1.5 22.25   66
```

PRACTICE

Using the functions above, test to see whether the variance of the Weights are significantly different between the two species of ant.

Randomization vs. Permutation

Note that every time we perform the operation above, we get a slightly different result and our test is therefore provides only an approximate *p*-value. With 30 of each kind of ant, it is prohibitive to try EVERY possible permutation (there are: $\binom{60}{30} = 1.18 \times 10^{17}$ possible ways to subsample these data for this test). With much smaller samples we can compute an *exact* permutation test *p*-value.

Permutation test

Here are some data on mandible lengths of male and female jackals (*Canis Aureus*) from the British museum in London:

```
Y.m <- c(120, 107, 110, 116, 114, 111, 113)
Y.f <- c(110, 111, 107, 108, 110, 105)
```

There are 7 males and 6 females, so only 1716 ways to break up the data into those groups. The sample means are 113 and 108.5 mm, respectively, for males and females ... so are males larger? Or, more precisely, what is the probability that $D = 0$ given that $D_{obs} = 4.5$?

Permutation test: code

```
require(gtools)  # For the combinations function

Z <- c(Y.m, Y.f)
C <- combinations(length(Z), length(Y.m))

getD.means <- function(index) mean(Z[index]) - mean(Z[-index])
```

Obtain observed statistics, and permutation distribution:

```
D.observed <- mean(Y.m) - mean(Y.f)
D.permuted <- apply(C, 1, getD.means)

sum(D.permuted > D.observed)/length(D.permuted)

## [1] 0.01282

sum(D.permuted >= D.observed)/length(D.permuted)

## [1] 0.02448
```

Of ALL the possible combinations of 6 and 7 males and females, in only 22 was the null-male subset still larger than the observed difference of 4.5 mm. [Note, that another 20 had a difference of EXACTLY 4.5! This frequently happens with small datasets].

Flashback: How did we do this before?

Without recourse to pretty rapid computation, statisticians had to pull all kinds of tricky and elegant contortions to obtain null distributions based not on *resampling* but on the central limit theorem and asymptotic results.

Recall that to compare two sample means, we used the T statistic:

$$t^* = \frac{\bar{X}_1 - \bar{X}_2}{s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

where

$$s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2},$$

and that under the null hypothesis the statistic has a known distribution:

$$t^* \sim \mathcal{T}_{n_1 + n_2 - 2}$$

allowing us to compute p-values by merely looking up values of the cumulative probability in a table. Mathematically and manually - this is a much more complicated procedure, but computationally trivial (easy to do even by hand).

Some T-tests

```
t.test(Weight ~ Species, data = Ant)

##
##  Welch Two Sample t-test
##
## data:  Weight by Species
## t = -7.084, df = 44.52, p-value = 8.093e-09
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -48.98 -27.29
## sample estimates:
##   mean in group Seed mean in group Thatch
##                54.67                92.80
```

```
t.test(Headwidth2 ~ Species, data = Ant)

##
##  Welch Two Sample t-test
##
## data:  Headwidth2 by Species
## t = -1.575, df = 49.75, p-value = 0.1217
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.11485  0.01392
## sample estimates:
##   mean in group Seed mean in group Thatch
##                1.621                1.671
```

How do these p-values compare to ours?

Some comments on classical inference

All of the so-called "pivot-quantities" - z-scores, t, Chi-squared and F statistics - all exploit the central limit theorem to fold up on themselves and reduce data to scale-free quantities with asymptotically known distributions with minimal free parameters. This contortion is very elegant (one Shakespeare versus a billion monkeys!), and a large amount of 20th century statistics was devoted to obtaining these quantities and understanding their properties and applicability. But their use comes with certain constraints:

- Variances between groups must be equal,
- Residuals must be normally distributed,
- A heavy reliance on the mean as a measure of comparison between groups,
- A strong assumption of independence in the sampling,
- An assumption of an observation arising from an infinite population.

Some comments on classical inference

All of the so-called "pivot-quantities" - z-scores, t, Chi-squared and F statistics - all exploit the central limit theorem to fold up on themselves and reduce data to scale-free quantities with asymptotically known distributions with minimal free parameters. This contortion is very elegant (one Shakespeare versus a billion monkeys!), and a large amount of 20th century statistics was devoted to obtaining these quantities and understanding their properties and applicability. But their use comes with certain constraints:

- Variances between groups must be equal,
- Residuals must be normally distributed,
- A heavy reliance on the mean as a measure of comparison between groups,
- A strong assumption of independence in the sampling,
- An assumption of an observation arising from an infinite population.

Open Question: how much longer will we be teaching these tools as a core component of statistics? Are they the equivalent of Newton's Mechanics 101 or Daguerrotyping 101?

Some comments on resampling inference

The development of **computational statistics** allows for relaxations of all of these constraints in different ways to different extents. In particular: permutation tests exist for *any* statistic, regardless of whether its distribution is known. There is absolute freedom in choosing a statistic which best discriminates between null and alternative hypotheses. Quick example **medians** of regional GDP:

```r
GDP <- c(49683, 3163, 7983, 3228, 2962, 1950, 15741, 4171, 10373, 14621, 8866)
Region <- factor(c(rep("SEAsia", 6), rep("SAmerica", 5)))
tapply(GDP, Region, function(x) c(mean(x), median(x)))

## $SAmerica
## [1] 10754 10373
##
## $SEAsia
## [1] 11495  3196
```

T-test of means:

```r
t.test(GDP ~ Region)$p.value

## [1] 0.9291
```

Permutation test of medians:

```r
C <- combinations(length(GDP),table(Region)[1])
getD.median <- function(index, Y)
  median(Y[index]) - median(Y[-index])
D.observed <- diff(tapply(GDP, Region, median))
D.permuted <- apply(C, 1, getD.median, Y=GDP)
sum(D.permuted >= abs(D.observed)) / length(D.permuted)

## [1] 0.08009
```

R tool: `lmPerm`

A very powerful package called `lmPerm` boasts the use of permutation tests to do anything `lm()`. Let's see how it works. Consider the following data on lettuce growth as a function of nitrogen (N) and potash (P) treatments:

	y	P	N
1	449	1	1
2	413	1	2
3	326	1	3
4	409	2	1
5	358	2	2
6	291	2	3
7	341	3	1
8	278	3	2
9	312	3	3

```
summary(lmp(y ~ P * N, data = CC164, perm = "Exact"))
```

```
## Call:
## lmp(formula = y ~ P * N, data = CC164, perm = "Exact")
##
## Residuals:
## ALL 9 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##          Estimate Pr(Exact)
## P.L        -60.58     0.079 .
## P.Q          0.41     1.000
## N.L        -63.64     0.064 .
## N.Q          4.08     0.893
## P.L:N.L    47.00     0.466
## P.Q:N.L    24.25     0.708
## P.L:N.Q    42.72     0.505
## P.Q:N.Q    13.00     0.852
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This is "Exact", because it computes all the p-values for all 9!=362,880 permutations. With 9, this is pretty instant. Accprding to authors: "12 needs 3 minutes, while 14 is an overnighter. No wonder permutations are seldomly used!"

R tool: `lmPerm`

Easy enough to obtain approximtae *p*-values, which are computed efficiently because of a stoppage rule, e.g. when standard deviation of p-value falls below some fraction of the estimated p-value

```
summary(lmp(y ~ P * N, data = CC164, perm = "Prob"))

## [1] "Settings:  unique SS "

##
## Call:
## lmp(formula = y ~ P * N, data = CC164, perm = "Prob")
##
## Residuals:
## ALL 9 residuals are 0: no residual degrees of freedom!
##
## Coefficients:
##         Estimate Iter Pr(Prob)
## P.L      -60.575 1371    0.069 .
## P.Q        0.408   51    1.000
## N.L      -63.640 1581    0.060 .
## N.Q        4.082   51    0.863
## P.L:N.L   47.000  190    0.347
## P.Q:N.L   24.249   51    0.824
## P.L:N.Q   42.724   56    0.643
## P.Q:N.Q   13.000   51    0.843
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: NaN on 0 degrees of freedom
## Multiple R-Squared:    1, Adjusted R-squared:  NaN
## F-statistic:  NaN on 8 and 0 DF,  p-value: NA
```

Also: ANOVA

```
anova(lmp(y ~ P * N, data = CC164, perm = "Prob"))

## [1] "Settings:  unique SS "
## Analysis of Variance Table
##
## Response: y
##           Df R Sum Sq R Mean Sq Iter Pr(Prob)
## P          2    11009      5504  913     0.26
## N          2    12200      6100 1263     0.19
## P:N        4     4791      1198  142     0.83
## Residuals  0        0
```

R tool: `lmPerm`

Also:

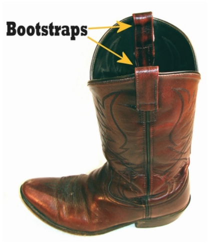
- P-values for saturated models
- Polynomial surfaces
- Multiple responses
- Nice vignette!

BUT:

- You are limited to the typical regression type statistics (means, sums of squares, etc.)

So: if you can have complex data structure, multiple covariates, multiple responses, and you are interested in comparing **means**, but are (understandably) uncomfortable with the normality assumptions of ANOVA, are nervous about outliers, there is no reason not to use `lmPerm`, which is (by definition) more robust to assumption violations than `lm()`. As an added bonus if the number of *rows* in your data is small (or you have a megacomputer), you can always brag that your *p*-value is EXACT, and not dependent on some hundred-year-old asymptotics.

The Bootstrap



The idea is simple: Assume your sample represents the population, and that any time you resample from your sample, you are effectively sampling from the population, and the distribution of any statistic that you observe by resampling from your own sample reflects the distribution of that statistic in the population.

Because you are relying only on your own data - without any of the heavy-handed innovation-killing tax-happy bureaucracy of asymptotics - it is as if you are raising yourself up by your bootstraps (or pony-tail).

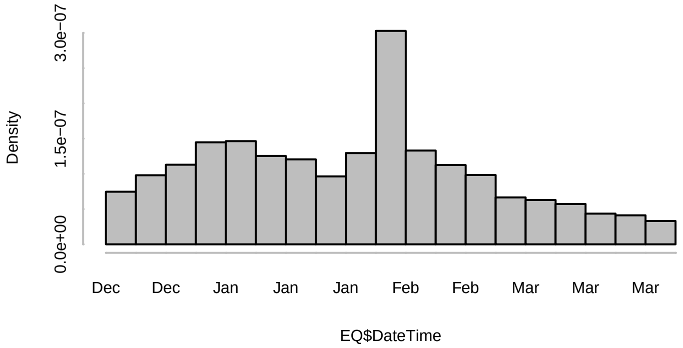
The Bootstrap: Confidence Intervals

Simplest example: Confidence intervals. Let's find a highly non-normal data set (our old - yet every renewing friend - the earthquakes of the world):

```
EQ <- read.csv("http://neic.usgs.gov/neis/gis/qed.asc")
```

And convert the times to POSIX.lt object:

```
EQ$DateTime <- strptime(paste(EQ$Date, EQ$Time), format = "%Y/%m/%d %H:%M:%S")
hist(EQ$DateTime, col = "grey", breaks = 20, main = "")
```



Classical Confidence Intervals

Let's pick out the waiting times just in April:

```
EQ.apr <- EQ$DateTime[EQ$DateTime$mon == 3]
dEQ.apr <- na.omit(difftime(EQ.apr[-length(EQ.apr)], EQ.apr[-1], units = "min"))
sort(dEQ.apr)

## Time differences in mins
##  [1]   0.8833   2.2167  14.2000  14.8333  27.8500  38.2000  69.5167
##  [8]  71.2167  73.9333  74.2500 102.8500 111.1667 117.5667 132.4667
## [15] 193.3833 197.3167 283.6167 516.2000
```

And obtain the pivoted 95% Confidence Interval around the mean:

$$\widehat{\mu} = \overline{X} \pm t_{0.975,n-1} \frac{s_x}{\sqrt{n}}$$

```
n <- length(dEQ.apr)
mean(dEQ.apr) + c(-1, 1) * qt(0.975, df = n - 1) * sd(dEQ.apr)/sqrt(n)

## [1]  50.81 176.04
```

Or just:

```
t.test(dEQ.apr)$conf.int

## Time differences in mins
## [1]  50.81 176.04
## attr(,"conf.level")
## [1] 0.95
```

The Bootstrapped Confidence Intervals

Bootstrapped confidence interval, by hand:

```r
nreps <- 1000
mean.BS <- rep(0, nreps)
for (i in 1:nreps) mean.BS[i] <- mean(sample(dEQ.apr, replace = TRUE))
quantile(mean.BS, c(0.025, 0.975))

##   2.5%  97.5%
##  63.25 170.92
```

OR, with `mosaic`'s `do()`:

```r
mean.BS <- do(nreps) * mean(sample(dEQ.apr, replace = TRUE))
quantile(mean.BS[, 1], c(0.025, 0.975))

##   2.5%  97.5%
##  62.87 177.08
```

The Bootstrapped Confidence Intervals

Bootstrapped confidence interval, by hand:

```
nreps <- 1000
mean.BS <- rep(0, nreps)
for (i in 1:nreps) mean.BS[i] <- mean(sample(dEQ.apr, replace = TRUE))
quantile(mean.BS, c(0.025, 0.975))

##   2.5%  97.5%
##  63.25 170.92
```

OR, with `mosaic`'s `do()`:

```
mean.BS <- do(nreps) * mean(sample(dEQ.apr, replace = TRUE))
quantile(mean.BS[, 1], c(0.025, 0.975))

##   2.5%  97.5%
##  62.87 177.08
```

Just as easy to do for the median (of course):

```
median.BS <- do(nreps) * median(sample(dEQ.apr, replace = TRUE))
quantile(median.BS[, 1], c(0.025, 0.5, 0.975))

##   2.5%    50%  97.5%
##  38.07  74.09 121.82
```

The boot package

This provides the powerful `boot()` function, which acquires the bootstrapped distribution of any statistic applied to any data or model object. Here, we obtain a bootstrap on the correlation between magnitude and depth of earthquakes in March:

```r
require(boot)
EQ.mar <- EQ[EQ$DateTime$mon == 3, ]
cor(EQ.mar$Magnitude, EQ.mar$Depth, use = "complete.obs")

## [1] -0.09485
```

It is negative. But is it *significantly* negative? The correlation coefficient, even under null-assumptions of normality, does not have a trivial distribution. But the bootstrapped version is quick and easy to obtain:

```r
cor.bs <- boot(EQ[EQ$DateTime$mon == 3,],
  function(x, i)
    cor(x$Magnitude[i], x$Depth[i],
        use= "complete.obs"),
  R = 1000)

## Error:  could not find function "boot"

(CI <- quantile(cor.bs$t, c(0.025, 0.975)))

## Error:  object 'cor.bs' not found
```

```r
hist(cor.bs$t, col = "grey", main = "")

## Error:  object 'cor.bs' not found

abline(v = CI, col = 2, lwd = 2)

## Error:  object 'CI' not found
```

So, at least so far in April (by mid-afternoon on the 2nd), deeper earthquakes have not been weaker than shallower earthquakes.

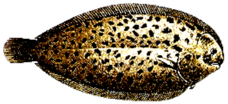
Practice:

1. What about for a larger sample, e.g. March?
2. How would we adapt this code to obtain a bootstrapped confidence interval around the regression slope of Magnitude against Depth?

Bootstrapped Prediction Intervals: on a glm

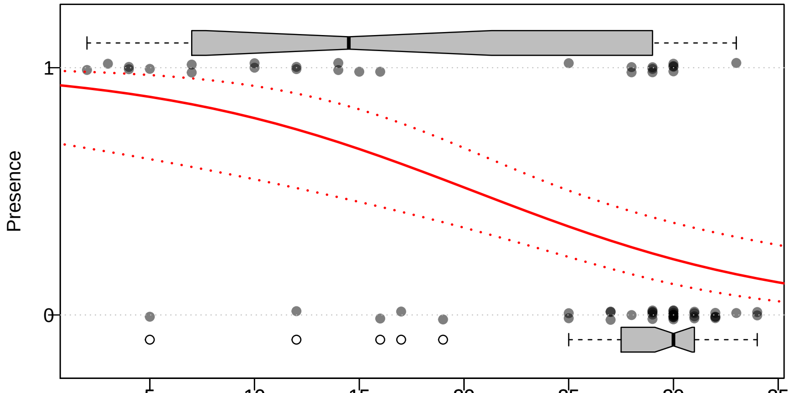
Using the example with the Portuguese sole posted here: http://faculty.washington.edu/eliezg/StatR201/VisualizingPredictions.html
Loading the data and fitting a binomial glm:

```
Solea <- read.csv("./data/Solea.csv")
Y <- Solea$Solea_solea
X <- Solea$salinity
Sole.glm <- glm(Y ~ X, family = "binomial")
expit <- function(x) exp(x)/(1 + exp(x))
Sole.predict <- predict(Sole.glm, newdata = data.frame(X = 0:40), se.fit = TRUE)
```



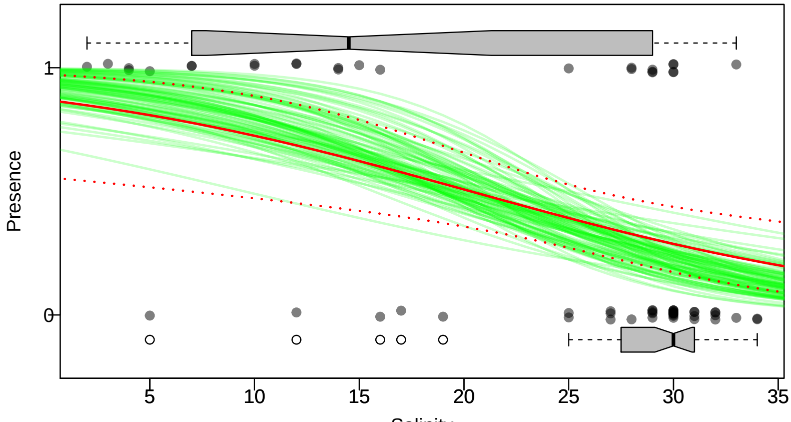
Parametric prediction intervals

```
plot(X, jitter(Y, factor = 0.1), col = rgb(0, 0, 0, 0.5), pch = 19, ylim = c(-0.2,
    1.2), xlab = "Salinity", ylab = "Presence", yaxt = "n")
boxplot(X ~ Y, horizontal = TRUE, notch = TRUE, add = TRUE, at = c(-0.1, 1.1),
    width = c(0.1, 0.1), col = "grey", boxwex = 0.1, yaxt = "n")
axis(2, at = 0:1, las = 1)
lines(0:40, expit(Sole.predict$fit), col = 2, lwd = 2)
lines(0:40, expit(Sole.predict$fit + 2 * Sole.predict$se), col = 2, lty = 3,
    lwd = 2)
lines(0:40, expit(Sole.predict$fit - 2 * Sole.predict$se), col = 2, lty = 3,
    lwd = 2)
abline(h = c(0, 1), col = "grey", lty = 3)
```



The boot package: Bootstrapped Prediction Intervals

```
Sole.bs <- data.frame(X, Y)
for (i in 1:100) {
    Sole.glm.bs <- glm(Y ~ X, data = Sole.bs[sample(1:nrow(Sole.bs), replace = TRUE),
        ], family = "binomial")
    Sole.predict.bs <- predict(Sole.glm.bs, newdata = data.frame(X = 0:40),
        se.fit = TRUE)
    lines(0:40, expit(Sole.predict.bs$fit), col = rgb(0, 1, 0, 0.2), lwd = 2)
}
```



Exercise (to think about):

How would you use the `boot()` function to obtain a 95% bootstrapped envelope around those prediction intervals?

Some comments on resampling inference

Resampling places just as much (or more!) emphasis on *well-formulated hypotheses* and awareness of the *assumptions* when designing tests and making inference on them!

Some comments on resampling inference

Resampling places just as much (or more!) emphasis on *well-formulated hypotheses* and awareness of the *assumptions* when designing tests and making inference on them!

For example, can you design a permulation/randomization test that compares the means of two samples with unequal variance? It's not easy!

Some comments on resampling inference

Resampling places just as much (or more!) emphasis on *well-formulated hypotheses* and awareness of the *assumptions* when designing tests and making inference on them!

For example, can you design a permulation/randomization test that compares the means of two samples with unequal variance? It's not easy!

In general, permutation/randomization-type tests test equality of distibutions: $F_1 = F_2$, whereas BOOTSTRAP methods (coming after the break) are better at testing parameters $\theta_1 = \theta_2$.

R Programming: Some tools for speeding up your code

Now that we are in randomization / resampling territory, it is becoming impossible to avoid loops. Because loops multiply any little inefficiency many times, it becomes worth our while to try to figure out how to speed some operations up. Here, the `system.time()` function is a useful initial diagnostic.
Lets compare two versions of my `getD.mean` function, for finding the difference in the means of Y sorted by a categorical vector X with two levels.

```
getD.meanA <- function(Y, X) as.numeric(diff(tapply(Y, X, mean)))
system.time(a <- do(1000) * getD.meanA(Ant$Weight, Ant$Species))

##    user  system elapsed
##    2.67    0.00    2.84


getD.meanB <- function(Y, X) mean(subset(Y, X == unique(X)[1])) - mean(subset(Y,
    X == unique(X)[2]))

system.time(a <- do(1000) * getD.meanB(Ant$Weight, Ant$Species))

##    user  system elapsed
##    2.85    0.02    2.95


getD.meanC <- function(Y, X) sum(Y[X == levels(X)[1]])/sum(Y == levels(X)[1]) -
    sum(Y[X == levels(X)[2]])/sum(Y == levels(X)[2])
system.time(a <- do(1000) * getD.meanC(Ant$Weight, Ant$Species))

##    user  system elapsed
##    0.62    0.00    0.64
```

Why is the last one SO much faster? ...
because it uses more basic functions! `sum(x)/length(x)` always beats `mean(x)`

R Programming: Profiling your code

The `Rprof()` function allows for detailed profiling of the time it takes to perform certain tasks.

```
Rprof("Rprof.out")
a <- do(1000) * getD.meanB(Ant$Weight, Ant$Species)
Rprof(NULL)
```

This opens a connection to a file called: `Rprof.out`, writes a lot of information into it, which you can then summarize using:

```
summaryRprof("Rprof.out")
```

Because the output is complex, we're better off looking at "live" in R proper.

Trade-offs

Remember that there is often a trade-off between faster code and more lucid and understandable code

We are often told to “Vectorize! Vectorize! Vectorize!” in R. It is often assumed that is for speed. But in fact, it is not always faster, but often more legible. I, for one, find:

```
diff(tapply(Y,X,mean))
```

to be far more understandable than the champion:

```
sum(Y[X==levels(X)[1]])/sum(Y == levels(X)[1]) -
 sum(Y[X==levels(X)[2]])/sum(Y == levels(X)[2])
```