

Permutation tests

Permutation tests are very similar to bootstraps. They test if two distributions are the same:

```
> x=rnorm(10)
> y=rnorm(20)+1
> mean(x)-mean(y)
```

```
[1] -0.7245577
```

```
>
```

In a permutation test, we permute the data, and ask how often it has such a difference between the samples:

```
> perm=t( sapply( 1:1000, function(i) sample( c(x,y) ) ) )
> perm.diff = apply(perm, 1, function(xy){ mean(xy[1:10])-mean(xy[11:30]) } )
> sum( abs(perm.diff) > abs((mean(x)-mean(y))) )
```

```
[1] 36
```

```
> 36/1000
```

```
[1] 0.036
```

```
>
```

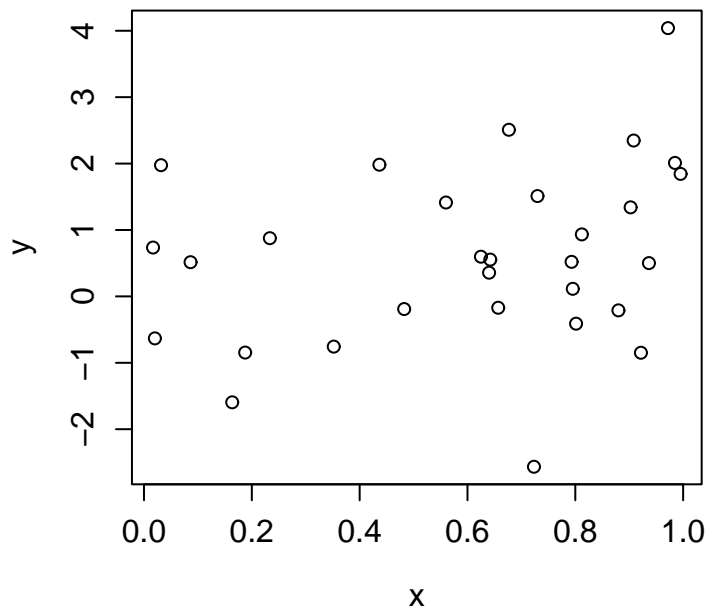
So this permutation test sees the data as significantly different

```
> t.test(x,y)$p.value  
[1] 0.07868077  
>
```

The permutation test tests the hypothesis that the two distributions are the same.

With a bootstrap test, we could do a very similar test, except that we sample **with replacement**.

```
r]  
> x=runif(30)  
> y=x+rnorm(30)  
> plot(x,y);v()
```



```
> l=lm(y~x)  
> anova(l)
```

Analysis of Variance Table

Response: y

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
x	1	3.970	3.970	2.2348	0.1461

```

Residuals 28 49.742 1.776
> names(anova(l))
[1] "Df" "Sum Sq" "Mean Sq" "F value" "Pr(>F)"
> anova(l)["Sum Sq"]
      Sum Sq
x          3.970
Residuals 49.742
> anova(l)["x","Sum Sq"]
[1] 3.970052
> our.test=function(y) {l=lm(y~x);anova(l)["x","Sum Sq"]}
> our.test(y)
[1] 3.970052
> perm.y=sapply(1:1000,function(i) sample(y))
> dim(perm.y)
[1] 30 1000
> res=apply(perm.y,2,our.test)
> range(res)
[1] 1.235280e-05 1.821772e+01
> sum(res <= 3.970052)
[1] 834
> sum(res > 3.970052)/1000
[1] 0.166
>

```

Plugging some holes

Files

```

> str=read.table("strange.txt",head=T,sep=",")
> str
      x y      z
1    1 1 3.8449461
2    2 1 2.9999548
3    3 1 2.2000090
4    4 1 -0.1760195
5    5 1 0.8905869
6    3 2 5.7175075
7    4 2 6.2294734
8    5 2 6.8006919
9    6 2 3.3612630

```

```
10 7 2 3.3244980
11 5 3 8.6946998
12 6 3 9.0335607
13 7 3 8.4194550
14 8 3 6.4132143
15 9 3 5.9953834
16 7 4 12.4814162
17 8 4 12.4018421
18 9 4 11.2890104
19 10 4 8.9024239
20 11 4 9.3622899
21 9 5 16.3395314
22 10 5 16.4536945
23 11 5 14.5808492
24 12 5 11.9888531
25 13 5 11.3470732
```

```
>
```

How to write it back?

```
> setwd("~/R-course-2006/lecture10")
> write.table(str,file="strange2.txt",col.names=T,row.names=F,sep="\t")
> ?write.table
```

```
write.table          package:base          R Documentation
```

```
Data Output
```

```
Description:
```

```
'write.table' prints its required argument 'x' (after converting
it to a data frame if it is not one nor a matrix) to a file or
connection.
```

```
Usage:
```

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"))

write.csv(..., col.names = NA, sep = ",", qmethod = "double")
write.csv2(..., col.names = NA, dec = ".", sep = ";", qmethod = "double")
```

```
Arguments:
```

```
x: the object to be written, preferably a matrix or data frame.
    If not, it is attempted to coerce 'x' to a data frame.
```

```
file: either a character string naming a file or a connection. ''''
       indicates output to the console.
```

`append`: logical. If `'TRUE'`, the output is appended to the file. If `'FALSE'`, any existing file of the name is destroyed.

`quote`: a logical value or a numeric vector. If `'TRUE'`, any character or factor columns will be surrounded by double quotes. If a numeric vector, its elements are taken as the indices of the columns to quote. In both cases, row and column names are quoted if they are written. If `'FALSE'`, nothing is quoted.

`sep`: the field separator string. Values within each row of `'x'` are separated by this string.

`eol`: the character(s) to print at the end of each line (row).

`na`: the string to use for missing values in the data.

`dec`: the string to use for decimal points in numeric or complex columns: must be a single character.

`row.names`: either a logical value indicating whether the row names of `'x'` are to be written along with `'x'`, or a character vector of row names to be written.

`col.names`: either a logical value indicating whether the column names of `'x'` are to be written along with `'x'`, or a character vector of column names to be written.

`qmethod`: a character string specifying how to deal with embedded double quote characters when quoting strings. Must be one of `"escape"` (default), in which case the quote character is escaped in C style by a backslash, or `"double"`, in which case it is doubled. You can specify just the initial letter.

`...`: further arguments to `'write.table'`.

Details:

By default there is no column name for a column of row names. If `'col.names = NA'` a blank column name is added. This can be used to write CSV files for input to spreadsheets. `'write.csv'` and `'write.csv2'` provide convenience wrappers for doing so.

If the table has no columns the rownames will be written only if `'row.names=TRUE'`, and *_vice versa_*.

Real and complex numbers are written to the maximal possible precision.

If a data frame has matrix-like columns these will be converted to multiple columns in the result (*_via_* `'as.matrix'`) and so a character `'col.names'` or a numeric `'quote'` should refer to the

columns in the result, not the input. Such matrix-like columns are unquoted by default.

Any columns in a data frame which are lists or have a class (e.g. dates) will be converted by the appropriate 'as.character' method: such columns are unquoted by default. On the other hand, any class information for a matrix is discarded.

The 'dec' argument only applies to columns that are not subject to conversion to character because they have a class or are part of a matrix-like column, in particular to columns protected by 'I()'.

Note:

'write.table' can be slow for data frames with large numbers (hundreds or more) of columns: this is inevitable as each column could be of a different class and so must be handled separately. If they are all of the same class, consider using a matrix instead.

See Also:

The 'R Data Import/Export' manual.

'read.table', 'write'.

'write.matrix' in package 'MASS'.

Examples:

```
## Not run:
## To write a CSV file for input to Excel one might use
x <- data.frame(a = I("a \" quote"), b = pi)
write.table(x, file = "foo.csv", sep = ",", col.names = NA,
            qmethod = "double")
## and to read this file back into R one needs
read.table("foo.csv", header = TRUE, sep = ",", row.names = 1)
## NB: you do need to specify a separator if qmethod = "double".

### Alternatively
write.csv(x, file = "foo.csv")
read.csv("foo.csv", row.names = 1)
## End(Not run)
```

>

But, we can also save variables in R (this we did see before)

```
> save(str,file="strange.Rdata")
> save(str,file="strange.Rdata",ascii=T)
> save(str,x,y,file="strange.Rdata",ascii=T)
> rm(list=ls())
> load("strange.Rdata")
> str
```

```
      x y      z
1    1 1  3.8449461
2    2 1  2.9999548
3    3 1  2.2000090
4    4 1 -0.1760195
5    5 1  0.8905869
6    3 2  5.7175075
7    4 2  6.2294734
8    5 2  6.8006919
9    6 2  3.3612630
10   7 2  3.3244980
11   5 3  8.6946998
12   6 3  9.0335607
13   7 3  8.4194550
14   8 3  6.4132143
15   9 3  5.9953834
16   7 4 12.4814162
17   8 4 12.4018421
18   9 4 11.2890104
19  10 4  8.9024239
20  11 4  9.3622899
21   9 5 16.3395314
22  10 5 16.4536945
23  11 5 14.5808492
24  12 5 11.9888531
25  13 5 11.3470732
```

>

Finally, we can also read command from a file

```
> source("program.R")
```

```
This program will simply print the numbers from 1 to 10
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

```
Then it will do the same, again
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

>