

Lecture 5 - statistical tests

In R it is almost too easy to employ a statistical test!

One has to be careful to understand what the test tests.

What is a test?

We have a hypothesis, and want to test if the data is consistent with this hypothesis. We also have an alternative hypothesis.

Usually we will

1. Decide on a number to calculate from our data.
2. Calculate how likely is one to get this number “or worse” given our hypothesis.
(Is “or worse” one- or two-sided?)
3. This gives us a p-value, which we compare to pre-defined cutoffs.

The problem: how do we know how likely things are?

We can either use distributions that the best mathematicians in the world figured out, or we can make our own using bootstrapping.

```
> help.start()
```

Loading Libraries (packages)

In R many functions are stored in libraries. To use them, we need to load the library:

```
> library(stats)
>
```

Frequencies

If we observe a family with 1 male offspring, and 9 female offspring, is the sex ratio 50:50?

Our hypothesis is that the sex ratio is 50%.

Since we would have been as surprised to see 1:9 as 9:1, we want to know how likely is it to get a ratio of 1:9 or 9:1 or worse?

How likely are we to get 0:10, 1:9, 9:1, or 10:0 if the ratio is really 50%?

The possibilities are:

```
F F F F F F F F F F M M M M M M M M M
M F F F F F F F F F F M M M M M M M M M
F M F F F F F F F F M F M M M M M M M M
F F M F F F F F F F M M F M M M M M M
F F F M F F F F F F M M M F M M M M M
F F F F M F F F F F M M M M F M M M M
F F F F F M F F F F M M M M M F M M M
F F F F F F M F F F M M M M M M F M M
F F F F F F F M F F M M M M M M F M M
F F F F F F F F M F M M M M M M M F M
F F F F F F F F F M M M M M M M M F M
```

Each is equally likely. There are 1024 possibilities, 22 of which are as bad as our observation, so:

```
> 22/1024
[1] 0.02148438
>
```

is our p-value.

R can do this calculation thus:

```
> ?binom.test
binom.test           package:stats           R Documentation
```

Exact Binomial Test

Description:

Performs an exact test of a simple null hypothesis about the probability of success in a Bernoulli experiment.

Usage:

```
binom.test(x, n, p = 0.5,
           alternative = c("two.sided", "less", "greater"),
           conf.level = 0.95)
```

Arguments:

x: number of successes, or a vector of length 2 giving the numbers of successes and failures, respectively.

n: number of trials; ignored if 'x' has length 2.

p: hypothesized probability of success.

alternative: indicates the alternative hypothesis and must be one of "two.sided", "greater" or "less". You can specify just the initial letter.

conf.level: confidence level for the returned confidence interval.

Details:

Confidence intervals are obtained by a procedure first given in Clopper and Pearson (1934). This guarantees that the confidence level is at least 'conf.level', but in general does not give the shortest-length confidence intervals.

Value:

A list with class "htest" containing the following components:

statistic: the number of successes.

parameter: the number of trials.

p.value: the p-value of the test.

conf.int: a confidence interval for the probability of success.

estimate: the estimated probability of success.

null.value: the probability of success under the null, 'p'.

alternative: a character string describing the alternative hypothesis.

method: the character string "Exact binomial test".

data.name: a character string giving the names of the data.

References:

Clopper, C. J. & Pearson, E. S. (1934). The use of confidence or fiducial limits illustrated in the case of the binomial. *_Biometrika_*, *26*, 404-413.

William J. Conover (1971), *_Practical nonparametric statistics_*. New York: John Wiley & Sons. Pages 97-104.

Myles Hollander & Douglas A. Wolfe (1973), *_Nonparametric statistical inference_*. New York: John Wiley & Sons. Pages 15-22.

See Also:

'prop.test' for a general (approximate) test for equal or given proportions.

Examples:

```
## Conover (1971), p. 97f.  
## Under (the assumption of) simple Mendelian inheritance, a cross  
## between plants of two particular genotypes produces progeny 1/4 of  
## which are "dwarf" and 3/4 of which are "giant", respectively.  
## In an experiment to determine if this assumption is reasonable, a  
## cross results in progeny having 243 dwarf and 682 giant plants.  
## If "giant" is taken as success, the null hypothesis is that p =  
## 3/4 and the alternative that p != 3/4.  
binom.test(c(682, 243), p = 3/4)  
binom.test(682, 682 + 243, p = 3/4) # The same.  
## => Data are in agreement with the null hypothesis.
```

```
> binom.test(1,10,p=0.5)
```

```
Exact binomial test
```

```
data: 1 and 10
```

```
number of successes = 1, number of trials = 10, p-value = 0.02148
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.002528579 0.445016117
sample estimates:
probability of success
                0.1
```

>

And, we get exactly the same p-value!

In this case, we were “surprised” to see so few males. But we had no preference for males or females. Therefore our alternative hypothesis was that the sex ratio is not 0.5.

But, maybe in a previous experiment, we already saw few males. Now we just want to confirm that it isn’t an equal sex ratio, but that we have few males.

In this case we ask: what is the chance to see 1:9 or worse, i.e. 1 male, 9 females, or 0 males 10 females?

This is a one-sided test.

From above we can see that the p-value in this case is 11/1024.

```
> binom.test(1,10,alternative="less")
```

Exact binomial test

```
data: 1 and 10
number of successes = 1, number of trials = 10, p-value = 0.01074
alternative hypothesis: true probability of success is less than 0.5
95 percent confidence interval:
 0.0000000 0.3941633
sample estimates:
probability of success
                0.1
```

>

If ahead of time we want to show that there are too few males, but we get 9 males and 1 female, then we still have to test for less!

```
> binom.test(9,10,alternative="less")
```

Exact binomial test

```
data: 9 and 10
number of successes = 9, number of trials = 10, p-value = 0.999
alternative hypothesis: true probability of success is less than 0.5

95 percent confidence interval:
 0.0000000 0.9948838
sample estimates:
probability of success
                0.9
```

>

Frequency tables

Let us assume that our hypothesis is that the frequency of green, blue, and brown eyes in the population is 20%, 30%, 50%.

We see the following data:

```
> observed=c(blue=17,blue=25,brown=60)
> observed
  blue blue brown
    17  25   60
> sum(observed)
[1] 102
>
```

We would expect:

```
> expected=c(0.2,0.3,0.5)*102
> expected
[1] 20.4 30.6 51.0
>
```

We could now, for example, sum the absolute difference of observed-expected. Or $(\text{observed} - \text{expected})^2$. But for all those it will be hard to calculate to get that value.

For the following function, we do have an approximate distribution:

$$\frac{(\text{observed value} - \text{expected value})^2}{\text{expected value}}$$

```
> chisq.test(c(17,25,60),p=c(0.2,0.3,0.5))

  Chi-squared test for given probabilities

data:  c(17, 25, 60)
X-squared = 3.1797, df = 2, p-value = 0.2040
>
```

Another example:

```
> hair=sample(c("blond","black"),90,rep=T,p=c(0.3,0.7))
> eyes=sample(c("green","brown"),90,rep=T,p=c(0.8,0.2))
> T=table(hair,eyes)
> T
      eyes
hair  brown green
black    9    53
blond   3    25
>
```

We would like to know if hair color and eye color are independent. Our hypothesis is that they are independent, and therefore one can calculate the chance for each category from

```
> chisq.test(T)
```

```
      Pearson's Chi-squared test with Yates' continuity correction
```

```
data:  T
X-squared = 0.0244, df = 1, p-value = 0.8758
```

```
Warning message:
Chi-squared approximation may be incorrect in: chisq.test(T)
```

```
>
```

The reason for the warning is that the `chisq.test` is only approximate. For small values in the table, the test is not exact. Another test is exact in this case:

```
> fisher.test(T)
```

```
      Fisher's Exact Test for Count Data
```

```
data:  T

p-value = 0.7472
alternative hypothesis: true odds ratio is not equal to 1

95 percent confidence interval:
 0.3149924 8.7903421
sample estimates:
odds ratio
 1.409920
```

```
>
```

Comparing means

Often we have data from two sources, and would like to know if we have the same means, i.e. one is not bigger than the other.

As before, we can think of many functions one could compute.

Assuming that the data is normally distributed, a test called t-test exists:

The t-test calculates the (difference of the means in the two samples)/(std. error of mean)

To use the t-test we have to know if the variances are the same. To do this, there is a test called `var.test`:

```
> x=rnorm(20,mean=10)
> y=rnorm(20,mean=10.5)
> var.test(x,y)
```

```
      F test to compare two variances
```

```
data:  x and y
```

```
F = 0.411, num df = 19, denom df = 19, p-value = 0.0597
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.1626644 1.0382793
```

```
sample estimates:
ratio of variances
      0.4109636
```

>

Now we can use the t-test:

```
> t.test(x,y)
```

```
Welch Two Sample t-test
```

```
data: x and y
t = -0.4676, df = 32.36, p-value = 0.6432
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.9666928  0.6055888
sample estimates:
mean of x mean of y
 9.842836 10.023388
```

>

Notice that the test did not detect the difference in the variables.

If we know or suspect that the variances are not the same, we can use a parameter of t.test:

```
> t.test(x,y,var.eq=F)
```

```
Welch Two Sample t-test
```

```
data: x and y
t = -0.4676, df = 32.36, p-value = 0.6432
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.9666928  0.6055888
sample estimates:
mean of x mean of y
 9.842836 10.023388
```

>

A non-parametric test of means

Non-parametric tests don't assume anything about the distribution of the variable - they use only the rank of the data.

The wilcoxon test calculates the rank of all samples, and then sums the ranks of the smaller sample, and does various things with them

```
> wilcox.test(x,y)
```

Wilcoxon rank sum test

```
data: x and y
W = 184, p-value = 0.6783
alternative hypothesis: true mu is not equal to 0
```

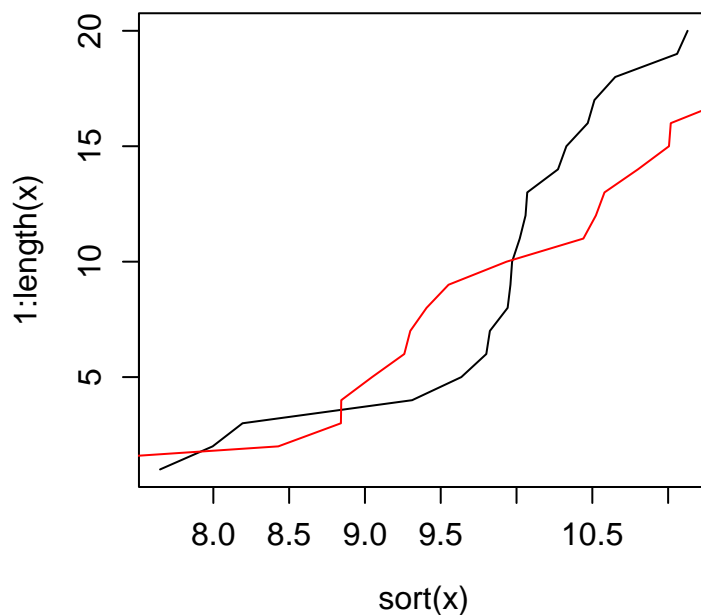
>

The wilcox test is slightly weaker than the t-test, because it doesn't take into account that the data is normal.

The Kolmogorov-Smirnov test

This is a very nice test that compares two distributions. It will differ even if the mean is the same but the variance is different.

```
> plot(sort(x),1:length(x),type="l")
> lines(sort(y),1:length(y),col=2);v()
```



>

The kolmogov-smirnov test finds the point at which the proportion of points is most different, and looks at that maximal difference.

```
> ks.test(x,y)
```

Two-sample Kolmogorov-Smirnov test

```
data: x and y

D = 0.3, p-value = 0.3356
alternative hypothesis: two.sided
```

>

Understanding magic

Let us look at the following curious thing:

```
> x=rnorm(100)
> y=rnorm(100,sd=0.1)+x
> k=ks.test(x,y)
> k
```

```
Two-sample Kolmogorov-Smirnov test
```

```
data: x and y

D = 0.04, p-value = 1
alternative hypothesis: two.sided
```

```
> print(k)
```

```
Two-sample Kolmogorov-Smirnov test
```

```
data: x and y

D = 0.04, p-value = 1
alternative hypothesis: two.sided
```

>

If everything in R is vectors or lists, why is the result of the KS test printed like this?

The magic is the following:

Even though things are always vectors or lists, they can have a "class". To see the class of an object, we can use the `class()` function:

```
> class(k)
[1] "htest"
```

>

And now the trick:

When we say `print(k)`, the actual function that is called is `print.htest()` for `k`, and `print.table()` for `s`.

```
> print.htest(k)
```

```
Two-sample Kolmogorov-Smirnov test
```

```
data: x and y
D = 0.38, p-value = 1.071e-06
```

```
alternative hypothesis: two.sided
```

```
> print.table(s)
```

```
      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.22700 -0.57190 -0.08799  0.01895  0.72390  2.04100
```

```
> print.htest(s)
```

```
Error in strsplit(x, as.character(split), as.logical(extended)) :
  non-character argument in strsplit()
```

```
> print
```

```
function (x, ...)
  UseMethod("print")
<environment: namespace:base>
```

```
>
```

All this is not very important, yet.

It only matters if we want to understand what function is called.

```
> help(print.htest)
```

```
help() for print.htest is shown in browser netscape ...
Use      help( print.htest , htmlhelp=FALSE)
or      options(htmlhelp = FALSE)
to revert.
Warning message:
Using non-linked HTML file: style sheet and hyperlinks may be incorrect in:
help(print.htest)
```

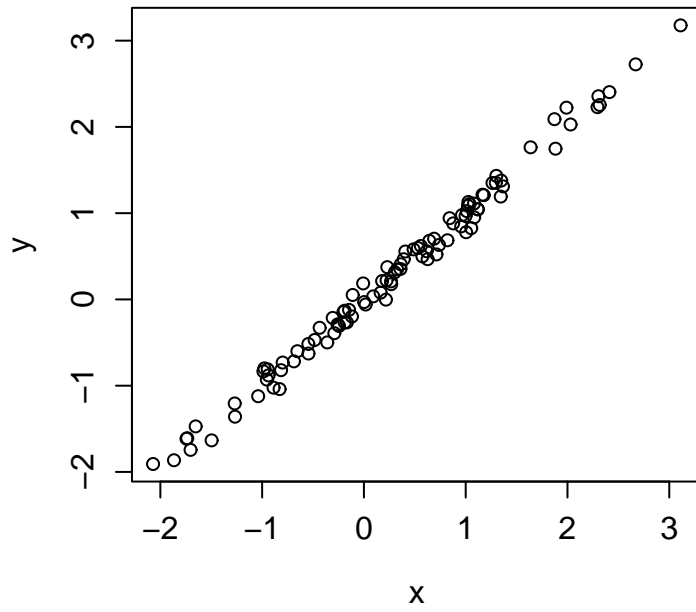
```
>
```

Another function that depends on its input is plot().

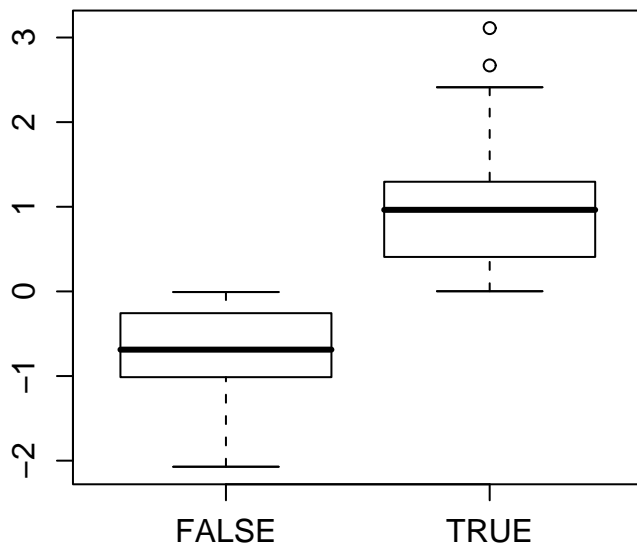
```
> plot
```

```
function (x, y, ...)
{
  if (is.null(attr(x, "class")) && is.function(x)) {
    nms <- names(list(...))
    if (missing(y))
      y <- {
        if (!"from" %in% nms)
          0
        else if (!"to" %in% nms)
          1
        else if (!"xlim" %in% nms)
          NULL
      }
    if ("ylab" %in% nms)
      plot.function(x, y, ...)
    else plot.function(x, y, ylab = paste(deparse(substitute(x)),
      "(x)"), ...)
  }
  else UseMethod("plot")
}
```

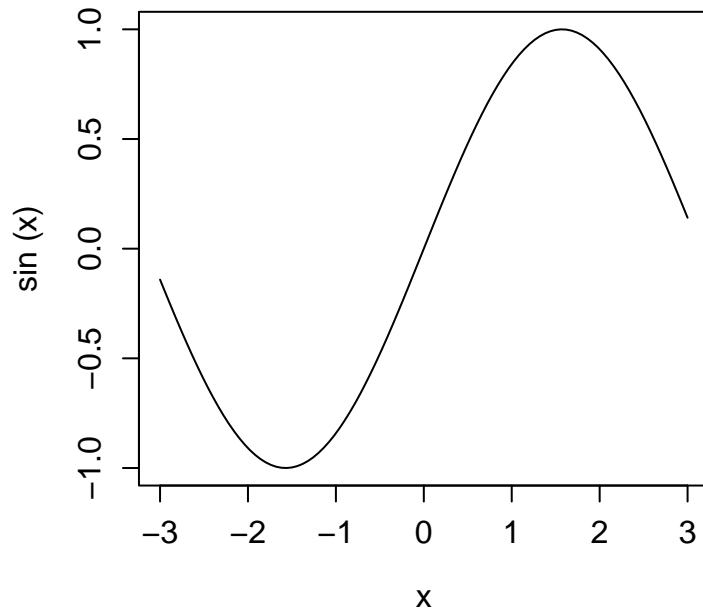
```
<environment: namespace:graphics>  
> plot(x,y);v()
```



```
> z=as.factor( x>0 )  
> plot(z,x);v()
```



```
> plot(sin,from=-3,to=3);v()
```



```
>
```

And now we can understand how to bring up help for these functions:

```
> class(sin)
```

```
[1] "function"
```

```
> help(plot.function)
```

```
curve                package:graphics                R Documentation
```

```
Draw Function Plots
```

```
Description:
```

```
Draws a curve corresponding to the given function or expression  
(in 'x') over the interval '[from,to]'.
```

```
Usage:
```

```
curve(expr, from, to, n = 101, add = FALSE, type = "l",  
       ylab = NULL, log = NULL, xlim = NULL, ...)
```

```
## S3 method for class 'function':  
plot(x, from = 0, to = 1, xlim = NULL, ...)
```

```
Arguments:
```

expr: an expression written as a function of 'x', or alternatively the name of a function which will be plotted.

x: a 'vectorizing' numeric R function.

from,to: the range over which the function will be plotted.

n: integer; the number of x values at which to evaluate.

add: logical; if 'TRUE' add to already existing plot.

xlim: numeric of length 2; if specified, it serves as default for 'c(from, to)'.

type, ylab, log, ...: graphical parameters can also be specified as arguments. 'plot.function' passes all these to 'curve'.

Details:

The evaluation of 'expr' is at 'n' points equally spaced over the range '[from, to]', possibly adapted to log scale. The points determined in this way are then joined with straight lines. 'x(t)' or 'expr' (with 'x' inside) must return a numeric of the same length as the argument 't' or 'x'.

If 'add = TRUE', 'c(from,to)' default to 'xlim' which defaults to the current x-limits. Further, 'log' is taken from the current plot when 'add' is true.

This used to be a quick hack which now seems to serve a useful purpose, but can give bad results for functions which are not smooth.

For "expensive" 'expr'essions, you should use smarter tools.

See Also:

'splinefun' for spline interpolation, 'lines'.

Examples:

```
op <- par(mfrow=c(2,2))
curve(x^3-3*x, -2, 2)
curve(x^2-2, add = TRUE, col = "violet")

plot(cos, xlim = c(-pi,3*pi), n = 1001, col = "blue")

chippy <- function(x) sin(cos(x)*exp(-x/2))
curve(chippy, -8, 7, n=2001)
curve(chippy, -8, -5)

for(ll in c("", "x", "y", "xy"))
  curve(log(1+x), 1,100, log=ll, sub=paste("log= ",ll,"'",sep=""))
par(op)
```

```

> class(z)
[1] "factor"
> help(plot.factor)
plot.factor                package:graphics                R Documentation

Plotting Factor Variables

Description:

This functions implements a "scatterplot" method for 'factor'
arguments of the _generic_ 'plot' function. Actually, 'boxplot' or
'barplot' are used when appropriate.

Usage:

## S3 method for class 'factor':
plot(x, y, legend.text = levels(y), ...)

Arguments:

x,y: numeric or factor. 'y' may be missing.

legend.text: a vector of text used to construct a legend for the plot.
Only used if 'y' is present and a factor.

...: Further arguments to 'plot', see also 'par'.

See Also:

'plot.default', 'plot.formula', 'barplot', 'boxplot'.

Examples:

plot(PlantGrowth)                # -> plot.data.frame
plot(weight ~ group, data = PlantGrowth) # numeric vector ~ factor
plot(cut(weight, 2) ~ group, data = PlantGrowth) # factor ~ factor
## passing "..." to barplot() eventually:
plot(cut(weight, 3) ~ group, data = PlantGrowth, density = 16*(1:3),col=NULL)

plot(PlantGrowth$group, axes=FALSE, main="no axes")# extremely silly
>

```