5. Permutation tests, and debugging

Ken Rice
Thomas Lumley

Universities of Washington and Auckland

Seattle, June 2011

Overview

- Permutation tests
- A mean
- Smallest p-value across multiple models
- Cautionary notes

Testing

When testing a null hypothesis we use a test statistic – one which we expect will have different values under the null hypothesis and the alternatives we care about (e.g. a relative risk of diabetes)

To do the test, we need to compute the **sampling distribution** of the test statistic when the null hypothesis is true. (For some test statistics and some null hypotheses this can be done analytically.) The **p-value** is the probability that the test statistic would be *at least as extreme* as the data we observed, if the null hypothesis is true.

Permutation is a simple way to compute the sampling distribution for any test statistic, under the 'strong null hypothesis' that a set of genetic variants has absolutely no effect on the outcome.

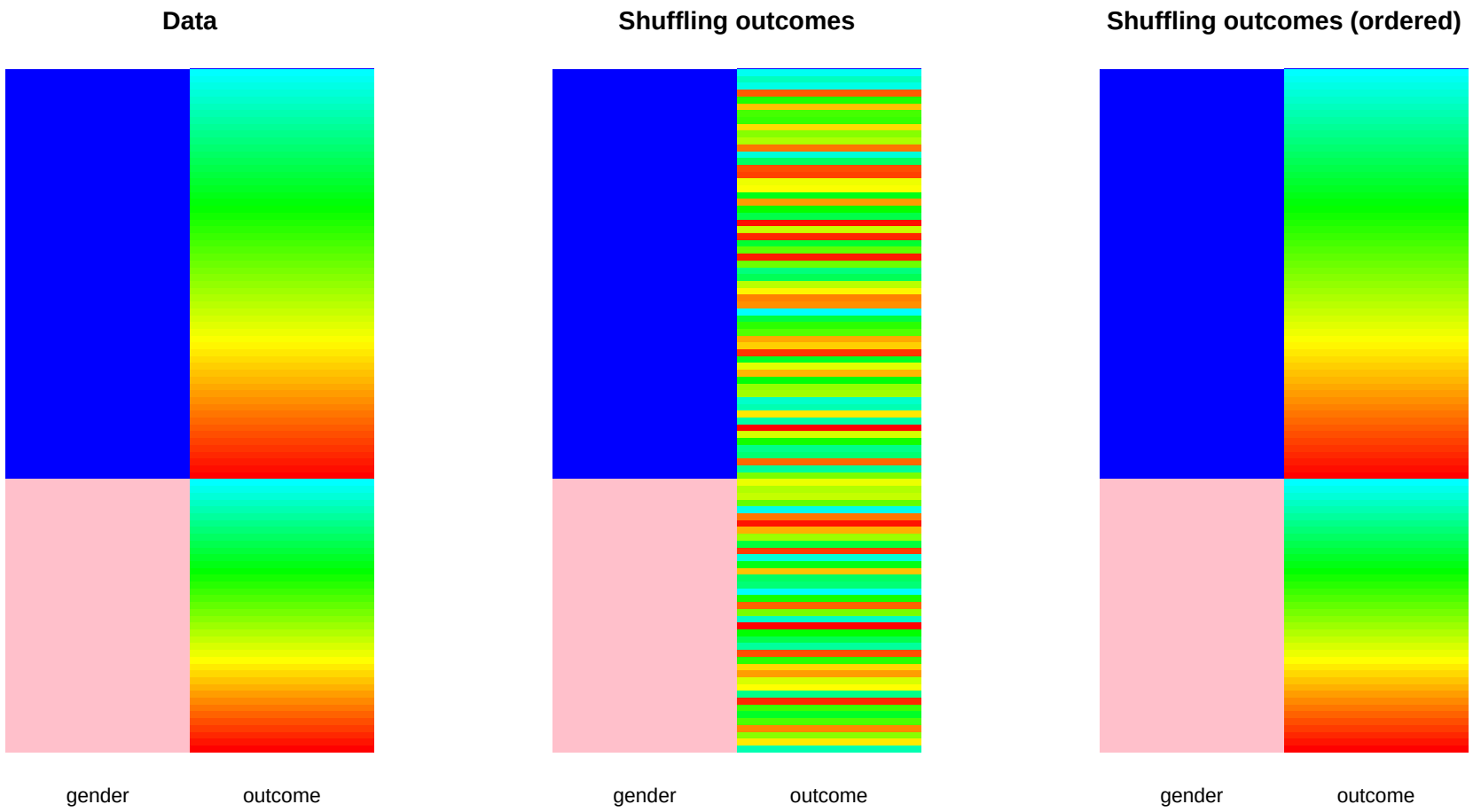
Permutations

The sampling distribution describes how frequently each test statistic value occurs.

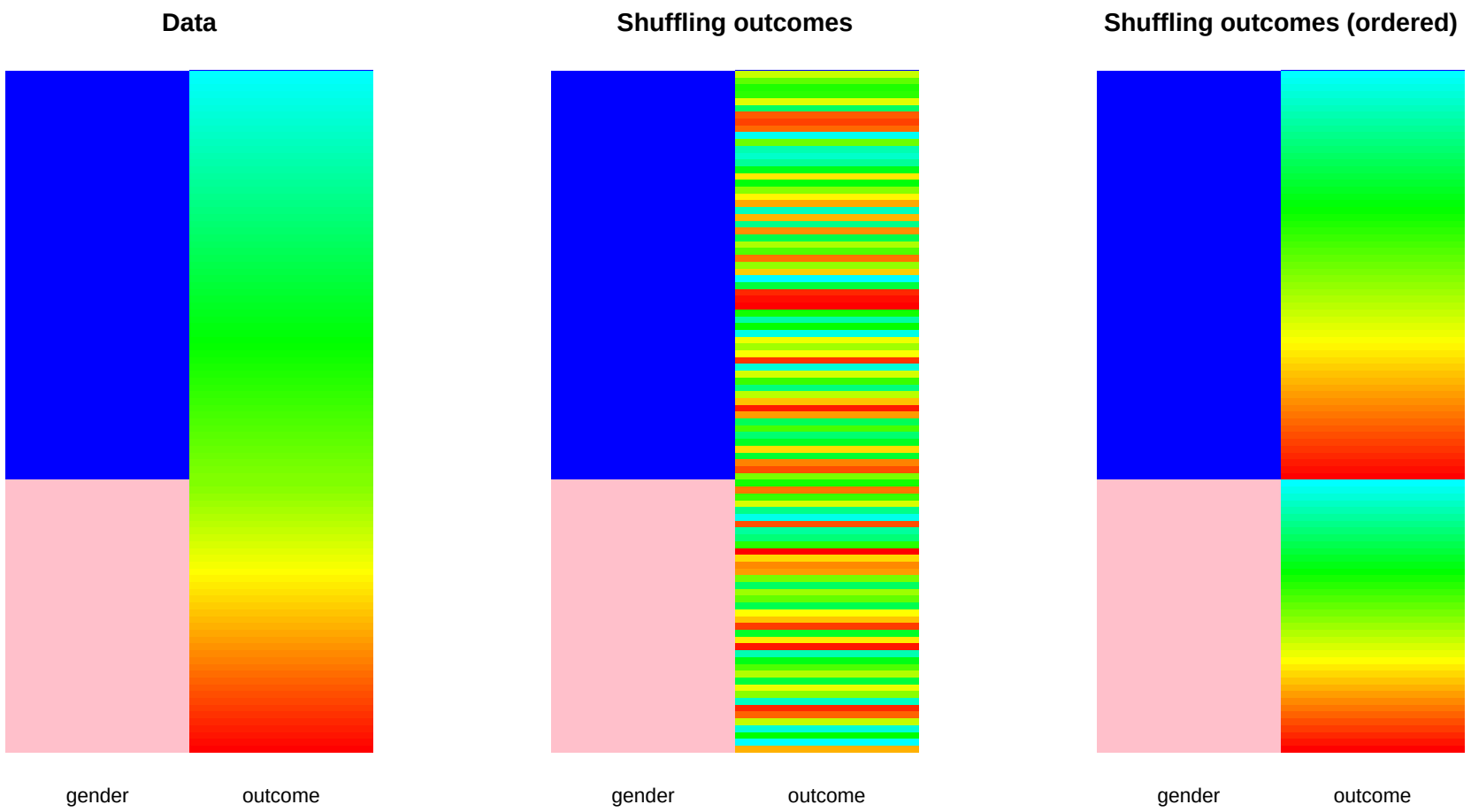
If the null hypothesis is true, changing the exposure would have no effect on the outcome. So, by randomly shuffling (permuting) the exposures we can make up as many 'null' data sets as we like.

Comparing the real data's test statistic to the shuffled data's test statistics gives a p-value

Example: shuffing data, null is true



Example: shuffling data, null is false



Testing: Mean difference (v1)

Our first example is a difference in mean outcome in a dominant model for a single SNP

```
## make up some 'true' data
carrier<-rep(c(0,1), c(100,200))
alt.y<-rnorm(300, mean=carrier/2)
```

There is a real effect; carriers have higher y-values, on average.

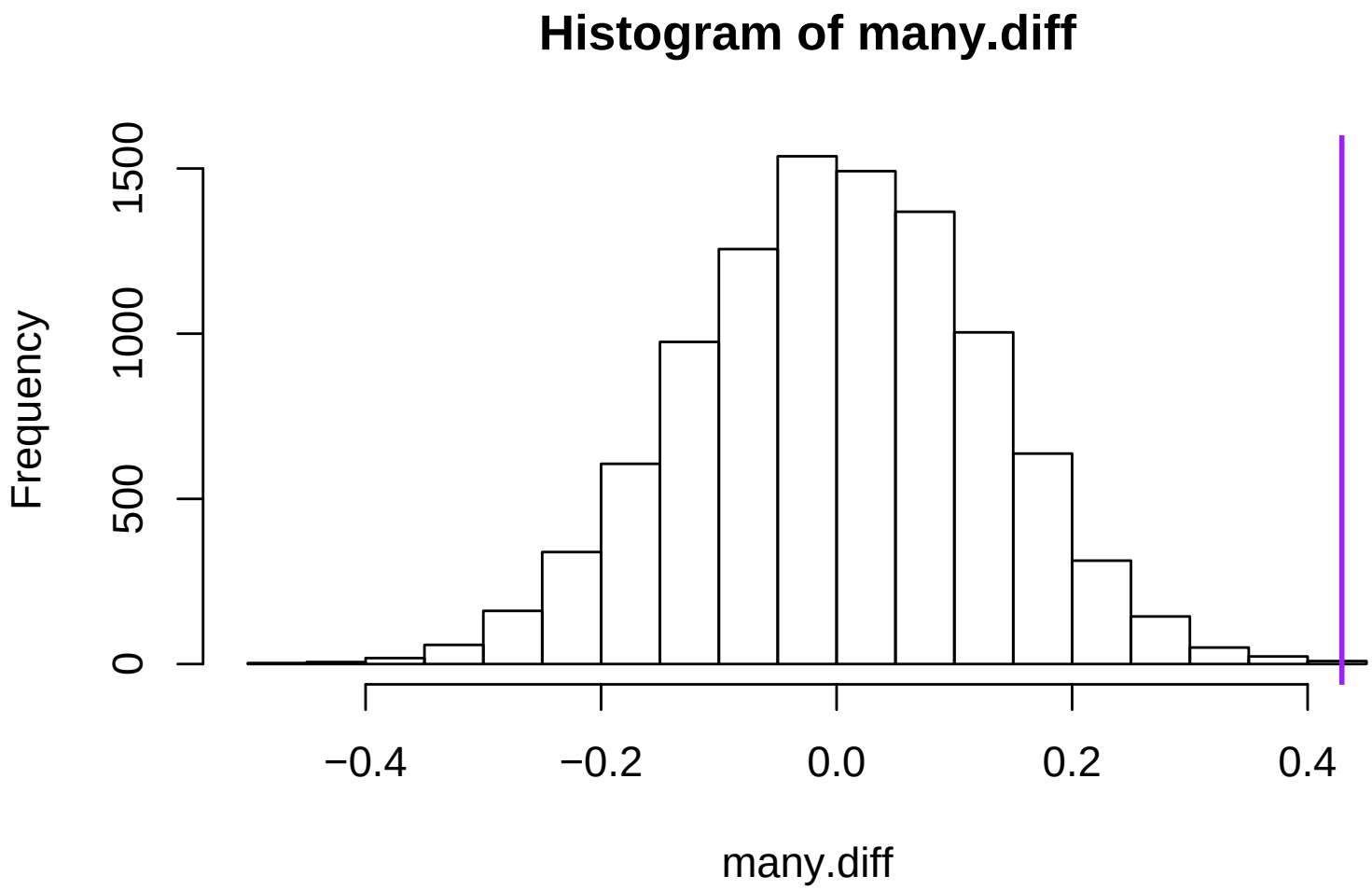
(In fact, for this situation theory tells us the distribution of a difference in means, and that we could just do a t-test)

Means: permutation test

To perform the permutation test...

```
alt.diff<-mean(alt.y[carrier==1])-mean(alt.y[carrier==0])

one.diff <- function(x,y) {
            xstar<-sample(x)
            mean(y[xstar==1])-mean(y[xstar==0])
            }

#do all the permutations;
many.diff <- replicate(10000, one.diff(carrier, alt.y))

#draw a helpful histgram
hist(many.diff)
abline(v=alt.diff, lwd=2, col="purple")

#calculate the p-value
mean(abs(many.diff) > abs(alt.diff))
```

Example: null is false



9 out of 10,000 shuffled difference exceeds observed difference: $p \approx 0.0009$.

Testing: Mean difference (v2)

Now we generate some y where the null holds, i.e. mean is unrelated to being a carrier;

```
carrier<-rep(c(0,1), c(100,200))
null.y<-rnorm(300, mean=0)
```

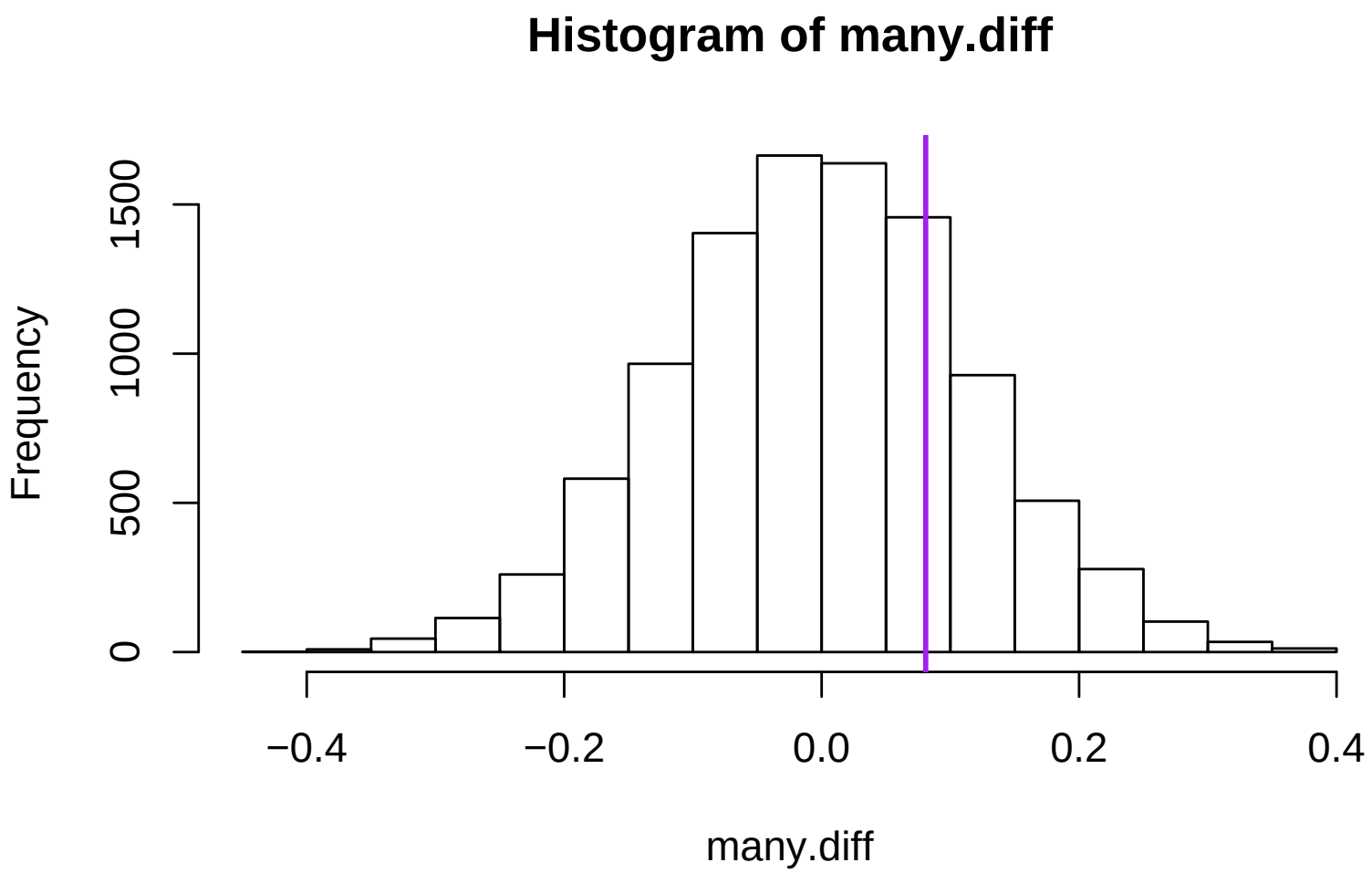
(what should happen?)

Testing: Mean difference (v2)

The rest of the commands work exactly as before...

```
null.diff<-mean(null.y[carrier==1])-mean(null.y[carrier==0])

one.diff <- function(x,y) {
            xstar<-sample(x)
            mean(y[xstar==1])-mean(y[xstar==0])
            } # same function as before

#do all the permutations;
many.diff <- replicate(10000, one.diff(carrier, null.y))

#draw a helpful histgram
hist(many.diff)
abline(v=null.diff, lwd=2, col="purple")

#calculate the p-value
mean(abs(many.diff) > abs(null.diff))
```

Example: null is false



4815/10000 difference exceeds observed difference: $p \approx 0.48$.

Means: t-test

How do our permutations compare to classical t-tests?

```
> t.test(alt.y~carrier, var.equal=T)
        Two Sample t-test
data:  alt.y by carrier
t = -3.4696, df = 298, p-value = 0.0005983 # we got 0.0009
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.6723963 -0.1856865
sample estimates:
mean in group 0 mean in group 1
      0.1416750       0.5707164
> t.test(null.y~carrier, var.equal=T)
        Two Sample t-test
data:  null.y by carrier
t = -0.6945, df = 298, p-value = 0.4879 # we get 0.48
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.3099976  0.1482715
sample estimates:
mean in group 0 mean in group 1
    -0.09224281     -0.01137975
```

How many permutations?

With 10000 permutations the smallest possible p-value is 0.0001, and the uncertainty near $p = 0.05$ is about $\pm 0.4\%$

If we need to know whether $p < 1 \times 10^{-5}$, we'll need much more precision.

Using, say, 100,000 permutations reduces the uncertainty near $p = 0.05$ to $\pm 0.1\%$ and allows accurate p-values as small as 0.0001 (i.e. 10^{-4}).

A useful strategy is to start with 1000 permutations and continue to larger numbers only if p is small enough to be interesting, e.g. $p < 0.1$. Also, parallel computing of permutations is easy: just run `R` on multiple computers.

Debugging

R error messages are sometimes opaque, because the error occurs in a low-level function.

`traceback()` reports the entire call stack, which is useful for seeing where the error really happened.

Suppose our outcome variable were actually a data frame with one column rather than a vector:

```
>  one.diff(carrier,ywrong)
Error in `[.data.frame`(y, xstar == 1) : undefined columns selected
```

We didn't know we were calling `[.data.frame`, so we don't understand the message

Debugging

```
> traceback()
5: stop("undefined columns selected")
4: `[.data.frame`(y, xstar == 1)
3: y[xstar == 1]
2: mean(y[xstar == 1])
1: one.diff(carrier, ywrong)
```

so the problem happens in our line of code `mean(y[xstar==1])` and is a problem with computing `y[xstar==1]`.

We might want to have a look at `y` and `xstar`

Debugging

The post-mortem debugger lets you look inside the code where the error occurred.

```
> options(error=recover)
>  one.diff(carrier,ywrong)
Error in `[.data.frame`(y, xstar == 1) : undefined columns selected

Enter a frame number, or 0 to exit

1: one.diff(carrier, ywrong)
2: mean(y[xstar == 1])
3: y[xstar == 1]
4: `[.data.frame`(y, xstar == 1)

Selection: 1
Called from: eval(expr, envir, enclos)
```

Debugging

```
Browse[1]> str(xstar)
 num [1:300] 1 1 1 1 1 1 1 0 1 1 ...
Browse[1]> str(y)
'data.frame': 300 obs. of  1 variable:
 $ null.y: num   1.265  0.590 -0.722  0.676 -0.431 ...
```

Turn the post-mortem debugger off with

```
options(error=NULL)
```

Minimum p-value

As we saw, for a permutation test for the mean, you would do just as well with a t-test – see theory, by Fisher.

Permutation tests are useful when we *don't* know how to compute the distribution of a test statistic.

An example: suppose we test additive effects of 8 SNPs, one at a time, and we want to know if the most significant association is real.

For any one SNP the z-statistic from a logistic regression model has a Normal distribution. But we need to know the distribution of the most extreme of eight z-statistics. This is not a standard distribution, but a permutation test is still straightforward.

Minimum p-value

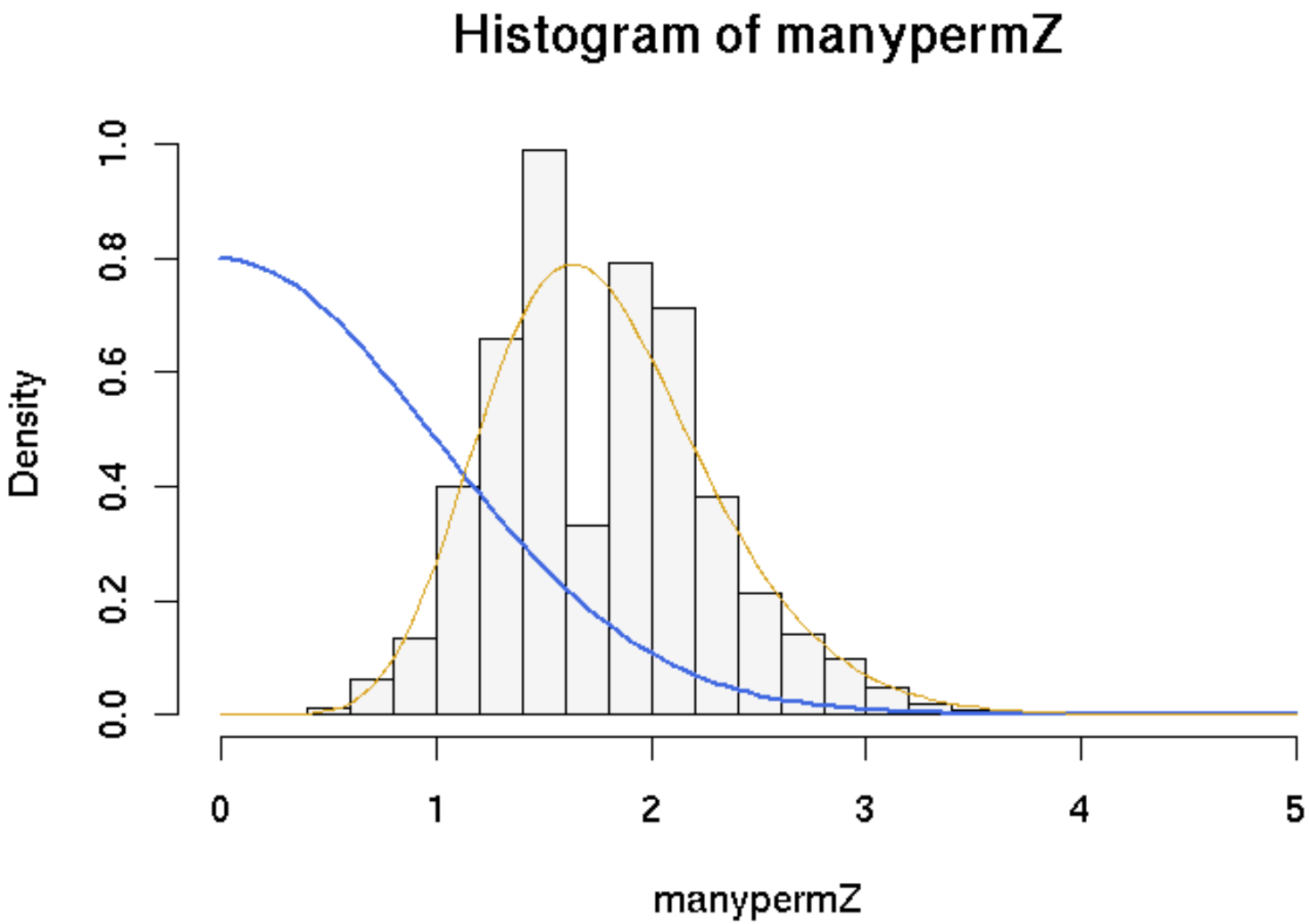
```
dat <- data.frame(y=rep(0:1,each=100), SNP1=rbinom(200,2,.1),
SNP2=rbinom(200,2,.2),SNP3=rbinom(200,2,.2),
SNP4=rbinom(200,2,.4),SNP5=rbinom(200,2,.1),
SNP6=rbinom(200,2,.2),SNP7=rbinom(200,2,.2),
SNP8=rbinom(200,2,.4))
```

```
> head(dat)
  y SNP1 SNP2 SNP3 SNP4 SNP5 SNP6 SNP7 SNP8
1 0    0    0    0    0    0    1    0    0
2 0    0    1    0    1    0    1    0    2
3 0    0    1    0    1    1    0    0    0
4 0    0    0    1    1    0    0    0    0
5 0    0    1    0    1    1    0    0    0
6 0    0    0    0    1    0    1    0    1
```

Minimum p-value

```
oneZ<-function(outcome, snp){
    model <- glm(outcome~snp, family=binomial())
    coef(summary(model))["snp","z value"]
}
maxZ<-function(outcome, snps){
    allZs <- sapply(snps,
         function(snp) oneZ(outcome, snp))
    max(abs(allZs))
}


true.maxZ<-maxZ(dat$y, dat[,-1])

manypermZ<-replicate(10000, maxZ(sample(dat$y), dat[,-1]))
```

Minimum p-value



Minimum p-value

The histogram shows the permutation distribution for the maximum Z-statistic.

The blue curve is the theoretical distribution for one Z-statistic

The yellow curve is the theoretical distribution for the maximum of eight independent Z-statistics.

Clearly the multiple testing is important: a Z of 2.5 gives $p = 0.012$ for a single test but $p = 0.075$ for the permutation test.

The theoretical distribution for the maximum has the right range but the permutation distribution is quite discrete. The discreteness is more serious with small sample size and rare SNPs.

[The theoretical distribution is not easy to compute except when the tests are independent.]

More debugging

Permutation tests on other people's code might reveal a lack of robustness.

For example, a very few permutations might result in all controls being homozygous for one of the SNPs; this could (quite reasonably) give an error message

We can work around this with `tryCatch()`

```
oneZ<-function(outcome, snp){
    tryCatch({model <- glm(outcome~snp, family=binomial())
              coef(summary(model))["snp","z value"]},
       error=function(e) rep(NA, 8)
       )
}
```

Now `oneZ()` will return a vector of 8 `NA`s if there is an error in the model fitting.

Caution: wrong null

Permutation tests cannot solve all problems: they are valid only when the null hypothesis is 'no association'

Suppose we are studying a set of SNPs that each have some effect on outcome and we want to test for interactions (epistasis).

Permuting the genotype data would break the links between genotype and outcome and created shuffled data with no main effects of SNPs.

Even if there are no interactions the shuffled data will look different from the real data.

Caution: weak null hypothesis

A polymorphism could increase the variability of an outcome but not change the mean.

In this case the strong null hypothesis is false, but the hypothesis of equal means is still true.

- If we want to detect this difference the permutation test is unsuitable because it has low power
- If we do not want to detect this difference the permutation test is invalid, because it does not have the correct Type I error rate.

When groups are the same size the Type I error rate is typically close to the nominal level, otherwise it can be too high or too low.

To illustrate this we need many replications of a permutation test. We will do 1000 permutation tests for a mean, each with 1000 permutations.

```
meandiff<-function(x,trt){
mean(x[trt==1])-mean(x[trt==2])
}
meanpermtest<-function(x,trt,n=1000){
observed<-meandiff(x,trt)
perms<-replicate(n, meandiff(x, sample(trt)))
mean(abs(observed)>abs(perms))
}

trt1<-rep(c(1,2),c(10,90))

perm.p<-replicate(1000, {
          x1<-rnorm(100, 0, s=trt1)
          meanpermtest(x1,trt1)})

table(cut(perm.p,c(0,.05,.1,.5,.9,.95,1)))/1000
```

```
(0,0.05] (0.05,0.1]  (0.1,0.5]  (0.5,0.9] (0.9,0.95]   (0.95,1]
      86         99        564        244           6          0
```

The p-values are too small, relative to a uniform distribution. If we reverse the standard errors we get

```
(0,0.05] (0.05,0.1]  (0.1,0.5]  (0.5,0.9] (0.9,0.95]   (0.95,1]
      27         28        275        354          67        249
```

If the two groups each have 50 observations we get

```
(0,0.05] (0.05,0.1]  (0.1,0.5]  (0.5,0.9] (0.9,0.95]   (0.95,1]
      50         45        403        407          52         43
```

which is much better.