Multiple comparison procedures

Cavan Reilly

October 23, 2019

Table of contents

Test multiplicity

In the traditional application of statistical methods to data analysis one has a single primary outcome and the goal is to define a procedure for testing a hypothesis about that outcome.

For example, we may hypothesize that some drug lowers mortality, so we would design a study to test this hypothesis.

In practice there are also typically secondary outcomes which are also assessed but whose role is not as important as the primary outcome.

A positive finding for a secondary outcome would suggest another study whose primary outcome would be the past secondary outcome.

One could use the previous data for the secondary outcome to design a well powered study for this secondary outcome.

Power

Power is the probability that the null hypothesis will be rejected if the alternative hypothesis is true.

For example, if we simulate data so that the null hypothesis is true then we will reject the null hypothesis sometimes: if we use a significance level of 0.05 this will happen 5% of the time.

We can check this in R using a for loop.

Power

```
> pval <- rep(NA,1000)
> for(i in 1:1000){
+   y1 <- rnorm(20)
+   y2 <- rnorm(20)
+   pval[i] <- t.test(y1, y2, var.equal=TRUE)$p.value
+ }
> sum(pval<.05)/1000
[1] 0.046
```

Which is about 5%.

We can use the same approach to determine the power when the null hypothesis is not true.

Power

```
> for(i in 1:1000){
+   y1=rnorm(20)
+   y2=rnorm(20, mean=1)
+   pval[i]=t.test(y1, y2, var.equal=TRUE)$p.value
+ }
> sum(pval<.05)/1000
[1] 0.87
```

So this means that a study with 20 subjects per group and a primary outcome with mean 0 in one group and mean 1 in the other group (and a standard deviation of 1 in both groups) has about a 90% chance of finding a significant difference if one tests for a difference using a 2 sample *t*-test.

The more subjects one has in the study, or the greater difference between the 2 groups, the larger the power of the study.

Test multiplicity

If one has many outcomes and tests them all it is possible that one will mistakenly reject a null hypothesis when in fact it is true.

This is the primary reason why studies are designed with a single primary outcome.

In this context we are controlling what is called the *family-wise error rate*.

A type I error occurs when we falsely reject the null hypothesis, i.e. we say there is an effect or difference when in truth none exists.

A type II error occurs when we falsely fail to reject the null hypothesis, i.e. we fail to detect a difference.

Family-wise error rate

Suppose we are interested in testing m null hypotheses, denoted $H_0^1, \ldots, H_0^m$.

	Non-significant	Significant	
H_0	U	V	m_0
H_A	T	S	$m - m_0$
	$m - R$	R	m

Here V is the number of times a type I error is made while T is the number of times a type II error is made.

The family-wise error rate (FWER) is $P(V \geq 1)$.

Family-wise error rate

There are stricter definitions that condition on the complete set of nulls or only subsets.

In particular the *FWER under the complete null* is

$$P(V \geq 1 | H_0^1, \ldots, H_0^m)$$

whereas the *FWER under a partial null* is $P(V \geq 1)$ conditioning on some subset of the null hypotheses.

A test procedure has *strong control* of the FWER if the FWER is less than or equal to the level of the test under all partial nulls.

It is said to have *weak control* if the FWER is less than or equal to the level of the test conditioning on the complete set of nulls.

False discovery rate

The *false discovery rate* is the proportion of falsely rejected nulls among the set of nulls that are rejected.

In terms of the table we have that the FDR is just $\mathrm{E}\left(\frac{V}{R}\right)$.

Being more careful, we note that since $R = 0$ with positive probability we must define the FDR to be 0 when $R = 0$, hence we find that

$$\begin{aligned}\mathrm{FDR} &= \mathrm{E}\left(\frac{V}{R}\,\middle|\, R > 0\right)P(R > 0) + \mathrm{E}\left(\frac{V}{R}\,\middle|\, R = 0\right)P(R = 0) \\ &= \mathrm{E}\left(\frac{V}{R}\,\middle|\, R > 0\right)P(R > 0)\end{aligned}$$

False discovery rate

If we assume that all nulls are true then $V = R$ and so $\frac{V}{R}$ is 0 if $V = 0$ and it is 1 if $V > 0$, hence

$$\begin{aligned} \mathrm{E}\left(\frac{V}{R}\right) &= 0 \times P(V = 0) + 1 \times P(V \geq 1) \\ &= P(V \geq 1) \\ &= \mathrm{FWER} \end{aligned}$$

so that if all nulls are true then the FDR equals the FWER.

False discovery rate

Thus if we control the FDR (i.e. keep it less than some value) then we are controlling the FWER in the weak sense.

If not all of the null hypotheses are true, so that $V < R$, then $V/R < 1$ and so $\mathrm{E}(\frac{V}{R}|V \geq 1) < 1$ which gives us

$$\begin{aligned}
\mathrm{E}\left(\frac{V}{R}\right) &= \mathrm{E}\left(\frac{V}{R} \,\middle|\, V = 0\right)P(V = 0) + \mathrm{E}\left(\frac{V}{R} \,\middle|\, V \geq 1\right)P(V \geq 1) \\
&= 0 \times P(V = 0) + \mathrm{E}\left(\frac{V}{R} \,\middle|\, V \geq 1\right)P(V \geq 1) \\
&< \mathrm{FWER}.
\end{aligned}$$

So that the false discovery rate is less than the FWER in general.

Control of the FDR versus the FWER in the strong sense depends on the application: exploratory versus confirmatory.

Single step approaches

There are 2 types of algorithms that are used to control the FWER: single step procedures and step down procedures.

In a single step procedure the same criterion is used for all tests.

In a step down procedure the p-values are sorted and a different criterion is used for each test in the sorted arrangement.

The Bonferroni adjustment is the most widely used single step procedure.

To motivate this procedure, first assume that we control the level of each of our m tests by requiring

$$P(\text{reject } H_0^i | H_0^i \text{ true}) \leq \alpha$$

for some α, the level of the individual tests.

Single step approaches

Then we have

$$\begin{aligned}
\text{FWER} &= P(V \geq 1 | H_0^1, \ldots, H_0^m) \\
&= 1 - P(V = 0 | H_0^1, \ldots, H_0^m).
\end{aligned}$$

If we then assume the tests are independent

$$\begin{aligned}
\text{FWER} &= 1 - \prod_{i=1}^{m} P(\text{do not reject } H_0^i | H_0^i) \\
&= 1 - \prod_{i=1}^{m} [1 - P(\text{reject } H_0^i | H_0^i)] \\
&\leq 1 - \prod_{i=1}^{m} (1 - \alpha) \\
&= 1 - (1 - \alpha)^m
\end{aligned}$$

Single step approaches

Note that with $m = 2$ and $\alpha = 0.05$ we get a FWER of 0.0975, so that with just 2 independent tests each controlled at the conventional significance level we have almost doubled our chances of making a type I error!

With $m = 14$ this probability becomes over 0.50.

The Bonferroni adjustment changes the significance level for all tests.

Instead of using a significance level of α one just uses $\alpha' = \alpha/m$.

This is because $1 - (1 - \alpha)^m \approx m\alpha$ by a first order Taylor expansion for α near zero.

Single step approaches

As an example, we will test for associations between mutations in the protease gene and indinavir and nelfinavir fold resistance using the virco data set.

We will dichotomize the genotype data and only consider positions in the gene for which at least 5% of the strains have an observed mutation.

```
> attach(virco)
> PrMut <- virco[,23:121]!="-" & virco[,23:121]!="."
> dim(PrMut)
[1] 1066 99
> NObs <- dim(virco)[1]
> PrMutSub <- data.frame(PrMut[,apply(PrMut,2,sum) > NObs*0.05])
> dim(PrMutSub)
[1] 1066 47
> Trait <- IDV.Fold-NFV.Fold
```

Bonferroni adjustment

Now we will write a function to get the the *p*-value from a 2 sample *t*-test and apply this to our data set.

```
> TtestP <- function(Geno){
+ return(t.test(Trait[Geno==1],Trait[Geno==0],na.rm=T)$"p.value")
+ }
> Pvec <- apply(PrMutSub,2,TtestP)
> sort(Pvec)
         P30          P76          P88          P55          P48          P89
3.732500e-12 9.782323e-10 1.432468e-06 2.286695e-06 5.749467e-06 8.924013e-05
         P11          P82          P60          P85          P54          P43
4.171618e-04 9.500604e-04 1.115441e-03 1.219064e-03 1.489381e-03 2.025621e-03
...
         P58          P62          P41          P12          P57

5.440101e-01 6.677043e-01 6.998280e-01 8.050362e-01 9.938846e-01
```

Bonferroni adjustment

So if we just use a significance level of 0.05 for each test we get the following sets of mutations.

```
> names(PrMutSub)[Pvec < 0.05]
 [1] "P11" "P14" "P30" "P32" "P33" "P35" "P43" "P46" "P47" "P48" "P54" "P55"
[13] "P60" "P61" "P67" "P69" "P76" "P82" "P84" "P85" "P88" "P89"
```

However if we use the Bonferroni adjustment we find fewer differences.

```
> PvecAdj <- p.adjust(Pvec,method="bonferroni")
> names(PrMutSub)[PvecAdj < 0.05]
[1] "P11" "P30" "P48" "P55" "P76" "P82" "P88" "P89"
```

Multiple comparisons in ANOVA

Historically, the first investigations into multiple hypothesis testing were motivated by *post-hoc* comparisons in ANOVA.

Recall, in ANOVA one tests the null hypothesis of no difference between the groups.

So if that hypothesis is rejected the natural question is, which groups differ and how.

Tukey addressed this question by determining the sampling distribution of the largest difference between means.

Scheffe took a different approach in which he considered every possible linear combination of the means, so his approach is more general.

Multiple comparisons in ANOVA

We will examine how to use Tukey's approach in R.

We will examine the association between a SNP and the percent change in the non-dominant muscle strength.

```
> attach(fms)
> Trait <- NDRM.CH
> table(resistin_c180g)
resistin_c180g
 CC  CG  GG
330 320  89
> m1 <- lm(Trait~resistin_c180g)
```

Multiple comparisons in ANOVA

```
> summary(m1)
Call:
lm(formula = Trait ~ resistin_c180g)
Residuals:
    Min       1Q  Median      3Q     Max
-56.054 -22.754 -6.054 15.346 193.946
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)       56.054      2.004  27.973   <2e-16 ***
resistin_c180gCG  -5.918      2.864  -2.067   0.0392 *
resistin_c180gGG  -4.553      4.356  -1.045   0.2964
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error:  33.05 on 603 degrees of freedom
  (791 observations deleted due to missingness)
Multiple R-squared: 0.007296,  Adjusted
R-squared: 0.004003
F-statistic: 2.216 on 2 and 603 DF,  p-value: 0.11
```

Multiple comparisons in ANOVA

So we detect a difference between the CC and CG genotypes if we don't do any adjustment, although the *p*-value for the ANOVA test is 0.11.

We note in passing that if one looks at diagnostic plots the residuals are far from normal.

To use Tukey's method we do the following.

Multiple comparisons in ANOVA

```
> TukeyHSD(aov(Trait~resistin_c180g))
  Tukey multiple comparisons of means
    95% family-wise confidence level
Fit:  aov(formula = Trait ~ resistin_c180g)
$resistin_c180g
           diff        lwr        upr     p adj
CG-CC -5.917630 -12.645660 0.8103998 0.0977410
GG-CC -4.553042 -14.788156 5.6820721 0.5486531
GG-CG  1.364588 -8.916062 11.6452381 0.9478070
```

So after making the adjustment the difference is not significant.

Benjamini Hochberg control of FDR

The Benjamini Hochberg adjustment is a step down procedure, i.e. it uses a different criterion for each test.

The method starts with a set of p-values which are assumed to be independent and we select a threshold for the FDR, which we denote q.
The algorithm is as follows:

1. Sort the p-values in increasing order to get $p_{(1)}, p_{(2)}, \dots, p_{(m)}$
2. define $k = \max\{i : p_{(i)} \leq \frac{i}{m} q\}$
3. Reject $H_0^{(1)}, H_0^{(2)}, \dots H_0^{(k)}$.

Benjamini Hochberg control of FDR

So if we have the set of sorted p-values

```
0.001, 0.012, 0.014, 0.122, 0.245, 0.320, 0.550, 0.776, 0.840, 0.995
```

we would compare these 2 the following set of numbers

```
0.005, 0.010, 0.015, 0.020, 0.025, 0.030, 0.035, 0.040, 0.045, 0.050
```

and find that $k = 3$ so that we reject the null hypotheses associated with the first 3 p-values.

Note that we still reject hypothesis 2 even though its p-value exceeds the second cut-off value.

Benjamini Hochberg control of FDR

We can also get adjusted p-values by letting

$$p_{(i)}^{adj*} = p_{(i)}m/i.$$

However after this adjustment the adjusted p-values may no longer be strictly increasing, hence you would go though and fix that up with

$$p_{(i)}^{adj} = \min_{j \geq i} p_{(j)}^{adj*}.$$

We can obtain these adjusted p-values in R too again using the virco data.

Benjamini Hochberg in R

```
> m <- length(Pvec)
> BHp <- sort(Pvec,decreasing=T)*m/seq(m,1)
> sort(cummin(BHp))
         P30          P76          P88          P55          P48          P89
1.754275e-10 2.298846e-08 2.244200e-05 2.686866e-05 5.404499e-05 6.990477e-04
         P11          P82          P85          P60          P54          P43
2.800943e-03 5.581605e-03 5.729603e-03 5.729603e-03 6.363718e-03 7.933683e-03
...
         P20          P62          P41          P12          P57
5.946157e-01 7.132296e-01 7.309314e-01 8.225370e-01 9.938846e-01
```

Benjamini Hochberg in R

Then we can get the names of the mutations that differ, but we must first reorder the mutations.

```
> BHp[order(Pvec,decreasing=T)] <- cummin(BHp)
> names(PrMutSub)[BHp < 0.05]
 [1] "P11" "P30" "P43" "P46" "P47" "P48" "P54" "P55" "P60" "P61" "P67" "P69"
[13] "P76" "P82" "P84" "P85" "P88" "P89"
```

So we find twice as many mutations as being associated with the difference in the fold change than we found using the Bonferroni correction.

Benjamini Hochberg in R

We can also do this more directly just using the `p.adjust` function.

```
> sort(p.adjust(Pvec,method="BH"))
          P30           P76           P88           P55           P48           P89
1.754275e-10 2.298846e-08 2.244200e-05 2.686866e-05 5.404499e-05 6.990477e-04
          P11           P82           P60           P85           P54           P43
2.800943e-03 5.581605e-03 5.729603e-03 5.729603e-03 6.363718e-03 7.933683e-03
...
          P58           P62           P41           P12           P57

5.946157e-01 7.132296e-01 7.309314e-01 8.225370e-01 9.938846e-01
```

Benjamini Yakutieli adjustment

The Benjamini Hochberg adjustment assumes that the tests are independent, but this is frequently not the case.

For example, SNPs that are close on the genome are likely in linkage disequilibrium so the test statistics should be similar.

While the Benjamini Hochberg adjustment frequently works fine with some dependence among the tests, the Benjamini Yakutieli adjustment allows for dependence among the tests.

Moreover, all one needs to do is replace q with $\tilde{q} = q / \sum_{i=1}^{m} i^{-1}$.

Also easy in R via the `p.adjust` function.

Benjamini Yakutieli adjustment in R

```
> BYp <- p.adjust(Pvec, method="BY")
> names(PrMutSub)[BYp < 0.05]
 [1] "P11" "P30" "P43" "P48" "P54" "P55" "P60" "P61" "P76" "P82" "P85" "P88"
[13] "P89"
```

So we can see that we don't find as many mutations, and a comparison shows that the Benjamini Yakutieli adjusted *p*-values are larger and frequently top out at 1.

```
> cbind(BHp,BYp)[1:5,]
            BHp        BYp
P57 0.527903116 1.00000000
P12 0.002800943 0.01243048
P41 0.822537013 1.00000000
P62 0.119861101 0.53193923
P58 0.073533226 0.32633780
```

The q value

The q value is a quantity that is analogous to a p-value except it is used to control the FDR rather than the FWER.

Formally it controls what is called the positive false discovery rate, which is defined as

$$pFDR = \mathrm{E}\left(\frac{V}{R} \,\middle|\, R > 0\right).$$

There is an R package distributed through Bioconductor that computes the q value.

The q value

First we must get the package from bioconductor and load it.

```
> if (!requireNamespace("BiocManager", quietly =
TRUE))
+ install.packages("BiocManager")
> BiocManager::install("qvalue")
> library(qvalue)
```

Then we just apply the function to a set of p-values.

```
> sort(qvalue(Pvec,lambda=0)$qvalues)
         P30          P76          P88          P55          P48          P89
1.754275e-10 2.298846e-08 2.244200e-05 2.686866e-05 5.404499e-05 6.990477e-04
         P11          P82          P60          P85          P54          P43
2.800943e-03 5.581605e-03 5.729603e-03 5.729603e-03 6.363718e-03 7.933683e-03
...
         P58          P62          P41          P12          P57

5.946157e-01 7.132296e-01 7.309314e-01 8.225370e-01 9.938846e-01
```

The *q* value

and we get exactly the same mutations as from the BH adjustment.

```
> names(PrMutSub)[qvalue(Pvec,lambda=0)$qvalues < 0.05]
[1] "P11" "P30" "P43" "P46" "P47" "P48" "P54" "P55" "P60" "P61" "P67" "P69"
[13] "P76" "P82" "P84" "P85" "P88" "P89"
```

We can also use a less conservative procedure that uses the bootstrap to estimate the proportion of null mutants and obtain a larger list.

```
> names(PrMutSub)[qvalue(Pvec,pi0.method="bootstrap")$qvalues < 0.05]
[1]  "P11" "P13" "P14" "P15" "P16" "P30" "P32" "P33" "P34" "P35" "P43" "P46"
[13] "P47" "P48" "P53" "P54" "P55" "P60" "P61" "P67" "P69" "P72" "P76" "P82"
[25] "P84" "P85" "P88" "P89"
```

It also provides a method for estimating the proportion of true nulls and reports that.

```
> qvalue(Pvec,pi0.method="bootstrap")$pi0
[1] 0.1891253
```

Resampling based methods

The free step-down resampling approach of Westfall and Young (1993) uses repeated samples under the complete null hypothesis to determine adjusted *p*-values.

Recall: the complete null hypothesis is the joint hypothesis that no markers are related to the trait.

We will first describe the algorithm assuming that we have a quantitative trait.

The method relies on the *subset pivotality* assumption which states that the distribution of test statistics is the same if all null hypotheses are true or only a subset are true.

Notation: let x_{ij} represent the genotype for the j^{th} marker for individual i and let y_i represent the trait variable for this subject.

Free step-down resampling

First you fit the linear model

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_m x_{mi} + \epsilon_i,$$

and compute the test statistic T_j (which is the parameter estimate $\hat{\beta}_j$ divided by its standard error) and p-value, p_j for each marker.

Then sort the absolute value of the observed tests statistics in increasing order: $|T|_{(1)}, |T|_{(2)}, \ldots, |T|_{(m)}$.

The next step is to sample with replacement from the set of residuals

$$r_i = y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{1i} - \hat{\beta}_2 x_{2i} - \ldots - \hat{\beta}_m x_{mi}$$

and then fit a regression model with these residuals as the outcome variable and the same set of predictor variables.

Free step-down resampling

We then record the absolute value of the test statistics in the order given by the order of the observed test statistics on the b^{th} iteration:

$$|T|^{*b}_{(1)}, |T|^{*b}_{(2)}, \ldots, |T|^{*b}_{(m)}$$

We then repeat this process of sampling from the set of residuals, refitting the model and recording the test statistics many (i.e. thousands) times.

The idea here is to examine the null distribution of the ordered test statistics.

Note: these won't necessarily be ordered.

Free step-down resampling

Then, for each of the samples, we check if the j^{th} ordered statistic is less than what we obtain from our simulation under the complete null. That is, on the b^{th} iteration we compute

$$\begin{aligned}
q_1^{*b} &= |T|_{(1)}^{*b} \\
q_2^{*b} &= \max(q_1^{*b}, |T|_{(2)}^{*b}) \\
q_3^{*b} &= \max(q_2^{*b}, |T|_{(3)}^{*b}) \\
\ldots & \\
q_m^{*b} &= \max(q_{m-1}^{*b}, |T|_{(m)}^{*b})
\end{aligned}$$

and determine if $q_j^{*b} > |T|_{(j)}$.

Free step-down resampling

The proportion of times that this is true gives the initial adjusted p-value, $\tilde{p}_{(j)}$.

The adjusted p-value is obtained from these initial estimates by enforcing that they are ordered by

$$\begin{aligned}
\tilde{p}_{(m)} &= \tilde{p}_{(m)} \\
\tilde{p}_{(m-1)} &= \max(\tilde{p}_{(m)}, \tilde{p}_{(m-1)}) \\
\tilde{p}_{(m-2)} &= \max(\tilde{p}_{(m-1)}, \tilde{p}_{(m-2)}) \\
\dots & \\
\tilde{p}_{(1)} &= \max(\tilde{p}_{(2)}, \tilde{p}_{(1)})
\end{aligned}$$

Westfall and Young have shown that this controls the FWER in the strong sense.

Free step-down resampling

While there are functions in the multtest package that allow one to do these adjustments, they are developed for the gene expression data, not SNP data.

Hence we will just code this up directly in R using the FAMuSS data as an example.

We will examine if 4 SNPs in the ACTN3 gene are related to muscle strength in the non-dominant arm.

```
> attach(fms)
> Actn3Bin <- data.frame(actn3_r577x!="TT",actn3_rs540874!="AA",
+ actn3_rs1815739!="TT",actn3_1671064!="GG")
> Mod <- summary(lm(NDRM.CH~.,data=Actn3Bin))
```

Free step-down resampling in R

```
> Mod
Call:
lm(formula = NDRM.CH ~ ., data = Actn3Bin)
Residuals:
    Min      1Q  Median      3Q     Max
-55.181 -22.614 -7.414 15.486 198.786
Coefficients:
Estimate Std.  Error t value Pr(>|t|)
(Intercept)                       54.700      3.212  17.028   <2e-16 ***
actn3_r577x.....TT.TRUE     -12.891      4.596  -2.805   0.0052 **
actn3_rs540874.....AA.TRUE   10.899     11.804   0.923   0.3562
actn3_rs1815739.....TT.TRUE  27.673     17.876   1.548   0.1222
actn3_1671064.....GG.TRUE   -29.166     17.516  -1.665   0.0964 .
---
Signif.  codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'  0.1 ' ' 1
Residual standard error:  32.93 on 591 degrees of freedom
  (801 observations deleted due to missingness)
Multiple R-squared: 0.01945,    Adjusted R-squared: 0.01281

F-statistic:  2.93 on 4 and 591 DF,  p-value: 0.02037
```

Again, the residuals plots look rather poor, but let's ignore that.

Free step-down resampling in R

Next, we'll record the observed test statistics, sort them, create a complete data set (note the large amount of missing data above) and save the order of the observed statistics.

```
> TestStatObs <- Mod$coef[-1,3]
> Tobs <- as.vector(sort(abs(TestStatObs)))
> MissDat <- apply(is.na(Actn3Bin),1,any) |
+ is.na(NDRM.CH)
> Actn3BinC <- Actn3Bin[!MissDat,]
> Ord <- order(abs(TestStatObs))
```

Free step-down resampling in R

Now we are in position to do the resampling. For this we use a `for` loop as follows.

```
> M <- 1000
> NSnps <- 4
> Nobs <- sum(!MissDat)
> TestStatResamp <- matrix(nrow=M, ncol=NSnps)
> for(i in 1:M){
+   Ynew <- sample(Mod$residuals, size=Nobs, replace=T)
+   ModResamp <- summary(lm(Ynew~., data=Actn3BinC))
+   TestStatResamp[i,] <- abs(ModResamp$coef[-1,3])[Ord]
+ }
```

Free step-down resampling in R

Next we look at the cumulative maxima (the q_j^{*b} values for the j^{th} marker on the b^{th} sample), and compute the adjusted *p*-values.

```
> Qmat <- t(apply(TestStatResamp, 1, cummax))
> Padj <- apply(t(matrix(rep(Tobs,M),NSnps)) < Qmat, 2, mean)
> Padj
[1] 0.370 0.249 0.234 0.040
```

Free step-down resampling with binary traits

When our outcome is not continuous but binary, we need a slight modification of the previous procedure.

In this case we use a logistic regression model: if we let

$$P(y_i = 1 | x_{1i}, x_{2i}, \ldots, x_{mi}) = \pi_i$$

then we have the model that

$$\log\Big(\pi_i/(1 - \pi_i)\Big) = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \ldots + \beta_m x_{mi}.$$

Note that this equation has no ϵ term: that is because the randomness enters because y_i is modeled as a random binary variable with success probability π_i.

So we replace our call to `lm` with a call to `glm` with `family` set to `binomial`.

Free step-down resampling with binary traits

The other difference is that we don't sample residuals, rather we sample the binary outcome y_i so that it has success probability

$$\tilde{\pi}_i = \exp(m_i) / \Big(1 + \exp(m_i)\Big)$$

where

$$m_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1i} + \hat{\beta}_2 x_{2i} + \ldots + \hat{\beta}_m x_{mi}.$$

The null restricted bootstrap

The free step-down resampling procedure is closely related to the use of the *bootstrap*, which is a general statistical technique for computing standard errors and statistical bias.

An example of a resampling based technique that is closer to the actual bootstrap is the null restricted bootstrap.

The distinguishing feature of the bootstrap is sampling with replacement from your data.

You must use replacement to get a different sample of the same size.

The bootstrap

Although there are many applications of the bootstrap in statistics, we will consider the simplest use: generating a confidence interval for a mean.

Note: this is a large sample procedure and will give invalid results with small sample sizes.

To use the bootstrap to get a confidence interval one samples from the observed data with replacement and computes the mean many times, then use the quantiles of the sampled means to get the endpoints of the interval.

The bootstrap

This is straightforward to do in R-first we simulate some data and use known expressions for the standard error of the mean to get the endpoints of a 95% confidence interval.

```
> set.seed(1)
> y1 <- rnorm(75)
> mean(y1)-qt(.975, df=74)*sqrt(var(y1)/75)
[1] -0.0906181
> mean(y1)+qt(.975, df=74)*sqrt(var(y1)/75)
[1] 0.3348219
```

Now we run 1000 iterations and get the mean of our samples.

```
> sim <- rep(NA, 1000)
> for(i in 1:1000){
+  y2 <- sample(y1, replace=TRUE)
+  sim[i] <- mean(y2)
}
```

The bootstrap

Then examine the quantiles of the simulations.

```
> quantile(sim, c(.025, .975))
       2.5%       97.5%
-0.07578891 0.32268552
```

We can increase the number of bootstrap samples to increase the accuracy of the approximation.

```
sim <- rep(NA, 10000)
for(i in 1:10000){
+  y2 <- sample(y1, replace=TRUE)
+  sim[i] <- mean(y2)
}
```

The bootstrap

Here the new endpoints:

```
> quantile(sim, c(.025, .975))
       2.5%       97.5%
-0.08671406 0.33040957
```

For comparison, here is the original interval: -0.0906181, 0.3348219

So the approximation is pretty good and we didn't need to know the sampling distribution (i.e. the quantiles of the t distribution) or how to compute the standard error.

The null restricted bootstrap

In this technique we repeatedly sample sets of outcomes and predictors and compare the resample based estimates to the observed estimates, and then select a common threshold to apply to all predictors so that the overall level of the test of the complete null is controlled.

We will illustrate the technique using the same example we used for Westfall and Young's procedure.

The null restricted bootstrap in R

```
> CoefObs <- as.vector(Mod$coef[-1,1])
> B <- 1000
> TestStatBoot <- matrix(nrow=B, ncol=NSnps)
> for(i in 1:B){
+   SampID <- sample(1:Nobs, size=Nobs, replace=T)
+   Ynew <- NDRM.CH[!MissDat][SampID]
+   Xnew <- Actn3BinC[SampID,]
+   s1 <- summary(lm(Ynew~.,data=Xnew))
+   CoefBoot <- s1$coef[-1,1]
+   SEBoot <- s1$coef[-1,2]
+   if(length(CoefBoot)==length(CoefObs)){
+     TestStatBoot[i,] <- (CoefBoot-CoefObs)/SEBoot
+   }
+ }
```

The null restricted bootstrap in R

Then we need to look at the overall significance level as it depends on the common threshold.

```
> for(cj in seq(2.7, 2.8, .01)){
+   print(cj)
+   print(mean(apply(abs(TestStatBoot)>cj,1,sum) >= 1, na.rm=T))
+ }
[1] 2.7
[1] 0.05182927
[1] 2.71
[1] 0.05182927
...
[1] 2.76
[1] 0.05081301
[1] 2.77
[1] 0.04979675

...
```

So we find that a cut off of 2.77 gets the significance level for the complete null within our usual range of 0.05. Looking at the observed statistics we see the largest is significant.

```
> Tobs
[1] 0.923304 1.547991 1.665086 2.804549
```

Effective number of tests

When a pair of SNPs are in perfect linkage disequilibrium, the tests of association between these SNPs and a trait give exactly the same test statistic, but when we correct for multiplicity we pay the price for multiple tests anyway.

By “pay the price”, think of the Bonferroni correction: if I have more tests the cutoff is smaller, hence I need more samples to have the same power.

Collecting samples costs money, so I am not being metaphorical at all.

Hence we should clearly first examine the SNPs to see if any are in complete LD before testing for associations.

But if 2 SNPs are almost in perfect LD, shouldn’t there be some sort of middle ground?

Effective number of tests

There is: we try to estimate the effective number of tests we are computing.

The idea is that if 2 SNPs are completely dependent then we are really only conducting one test (so the Bonferroni cut off is 0.05), whereas if they are completely independent then we really are conducting 2 independent tests (so the Bonferroni cut off is 0.025).

For intermediate cases we get intermediate cutoffs.

Effective number of tests

Recall: given a square matrix R (like a correlation matrix), the eigenvalues, λ and eigenvectors x are defined by the equation

$$Rx = \lambda x.$$

If λ_{obs} are the observed eigenvalues of the correlation matrix of a set of m SNPs and $\hat{V}$ is the variance of these eigenvalues then the effective number of tests is given by the following expression:

$$m_{eff} = 1 + (m-1)\left[1 - \frac{\hat{V}}{m}\right].$$

So we use m_{eff} in place of when we conduct our Bonferroni correction.

Effective number of tests in R

We can use built in facilities for computing eigenvalues in R.

```
> corActn3 <- cor(Actn3BinC)
> eigenValActn3 <- eigen(corActn3)$values
> mEff <- 1+(4-1)*(1-var(eigenValActn3)/4)
> mEff
[1] 1.816267
> 0.05/4
[1] 0.0125
> 0.05/mEff
[1] 0.02752899
```

So the cut off is much higher giving considerably more power than the simple Bonferroni method.

Effective number of tests in R

There is some controversy over the best way to compute the correlations: some simulation studies have found that using Pearson's correlation coefficient (what we have used here) is *anti-conservative*, i.e. we are not adequately controlling the significance level.

Above we used Pearson's correlation coefficient, others suggest using r^2, the measure of LD we discussed perviously, but this has also been shown to be anti-conservative.