Colocalisation Analysis

Chris Wallace

<2013-05-20 Mon>

A brief outline of colocalisation testing

The coloc package can be used to perform genetic colocalisation analysis of two potentially related phenotypes, to ask whether they share common genetic causal variant(s) in a given region. There are a few key references which this vignette will not duplicate (see below), but, in brief, two approaches are implemented.

First, the proportional approach uses the fact that for two traits sharing causal variants, regression coefficients for either trait against any set of SNPs in the neighbourhood of those variants must be proportional. This test was first proposed by Plagnol et al. [1] in the context of evaulating whether expression of the gene *RPS26* mediated the association of type 1 diabetes to a region on chromosome 12q13 as had recently been proposed. The test addressed a common issue in genetics, and meant researchers could avoid the need to squint at parallel manhattan plots to decide whether two traits share causal variants. The function `coloc.test()` in this package evolved from code released by Vincent, but no longer available.

However, choosing **which** SNPs to use for the test is a problem. The obvious choice is to use those most strongly associated with one or other trait to maximise information, but doing so induces bias in the regression coefficients, which in turn leads to increased likelihood of spuriously rejecting the null of colocalisation, ie a quite substantially increased type 1 error rate [2]. I proposed two alternatives to address this problem, eithir using a principal component summary of genetic variation in the region to overcome the need to select a small set of test SNPs, implemented in `coloc.pcs()` and associated functions, or to use the ideas of Bayesian model averaging

[1] `http://www.ncbi.nlm.nih.gov/pubmed/19039033`

[2] `http://arxiv.org/abs/1301.5510`

to average p values over SNP selections, generating posterior predictive p values, implemented in `coloc.bma()`.

Proportional testing, however, requires individual level genotype data, which are not always available. Claudia Giambartolomei and Vincent Plagnol have proposed an alternative method, which makes use of Jon Wakefields work on determining approximate Bayes Factors from p values [3] to generate a Bayesian colocalisation analysis [4], implemented in the function `coloc.summaries()`. Note that it is possible to use Bayesian analysis for proportional testing too, in determining the posterior distribution of the proportionality constant η.

Usage

Let's simulate a small dataset, and compare the three methods.

```
> setClass("simdata",
+          representation(df1="data.frame",df2="data.frame"))
> setValidity("simdata", function(object) {
+   n <- nrow(object@df1)
+   if(nrow(object@df2)!=n)
+     return("nrow of '@df1' should equal nrow of '@df2'")
+ })

Class "simdata" [in ".GlobalEnv"]

Slots:

Name:         df1        df2
Class: data.frame data.frame

> setMethod("show", signature="simdata", function(object) {
+   cat("pair of simulated datasets, with",ncol(object@df1)-1,"SNPs and",nrow(object@
+ })

[1] "show"

> sim.data <- function(nsnps=10,nsamples=200,causals=1:2,nsim=1) {
+   cat("Generate",nsim,"small sets of data\n")
```

[3] http://www.ncbi.nlm.nih.gov/pubmed/18642345
[4] http://arxiv.org/abs/1305.4022

```
+   ntotal <- nsnps * nsamples * nsim
+   X1 <- matrix(rbinom(ntotal,1,0.4)+rbinom(ntotal,1,0.4),ncol=nsnps)
+   Y1 <- rnorm(nsamples,rowSums(X1[,causals]),2)
+   X2 <- matrix(rbinom(ntotal,1,0.4)+rbinom(ntotal,1,0.4),ncol=nsnps)
+   Y2 <- rnorm(nsamples,rowSums(X2[,causals]),2)
+   colnames(X1) <- colnames(X2) <- paste("s",1:nsnps,sep="")
+   df1 <- cbind(Y=Y1,X1)
+   df2 <- cbind(Y=Y2,X2)
+   if(nsim==1) {
+     return(new("simdata",
+                df1=as.data.frame(df1),
+                df2=as.data.frame(df2)))
+   } else {
+     index <- split(1:(nsamples * nsim), rep(1:nsim, nsamples))
+     objects <- lapply(index, function(i) new("simdata", df1=as.data.frame(df1[i,]),
+                                              df2=as.data.frame(df2[i,])))
+     return(objects)
+   }
+ }
> ## simulate some data
> set.seed(12345)
> data <- sim.data(nsim=1)

Generate 1 small sets of data
```

Proportional testing

Principal components

The code below first prepares a principal component object by combining the genotypes in the two dataset, then models the most informative components (the minimum set required to capture 80% of the genetic variation) in each dataset, before finally testing whether there is colocalisation between these models.

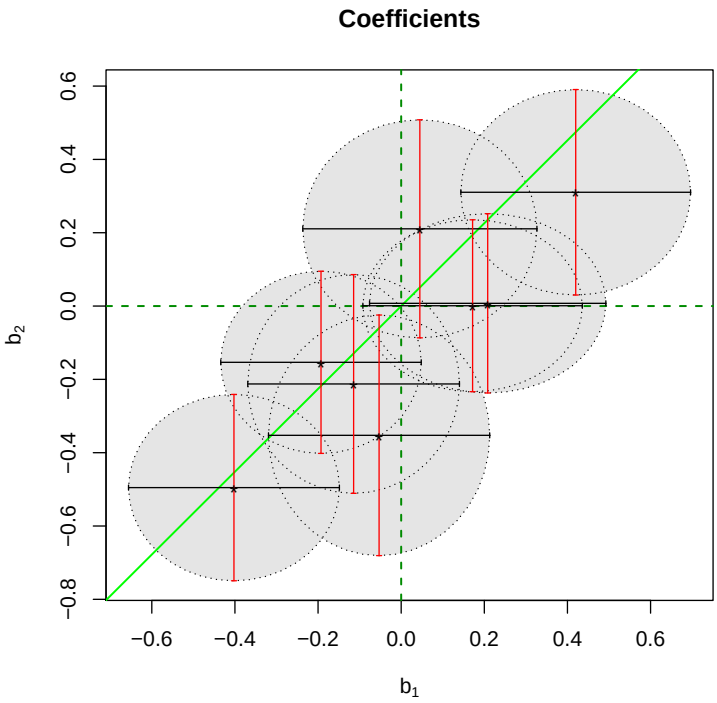
```
> ## run a coloc with pcs
> pcs <- pcs.prepare(data@df1[,-1], data@df2[,-1])
> pcs.1 <- pcs.model(pcs, group=1, Y=data@df1[,1], threshold=0.8)

selecting 8 components out of 10 to capture 0.8451804 of total variance.
```

```
> pcs.2 <- pcs.model(pcs, group=2, Y=data@df2[,1], threshold=0.8)

selecting 8 components out of 10 to capture 0.8451804 of total variance.

> ct.pcs <- coloc.test(pcs.1,pcs.2)
```



The plot shows the estimated coefficients for each principal component modeled for traits 1 and 2 on the x and y axes, with circles showing the 95% confidence region. The points lie close to the line through the origin, which supports a hypothesis of colocalisation.

A little more information is stored in the `ct.pcs` object:

```
> ct.pcs

          eta.hat         chisquare                 n p.value.chisquare
        1.1297564         5.2672429         8.0000000         0.6273863

> str(summary(ct.pcs))
```

```
         eta.hat         chisquare                 n p.value.chisquare
       1.1297564         5.2672429         8.0000000         0.6273863
Named num [1:4] 1.13 5.267 8 0.627
- attr(*, "names")= chr [1:4] "eta.hat" "chisquare" "n" "p.value.chisquare"
```

The best estimate for the coefficient of proportionality, $\hat{\eta}$, is 1.13, and the null hypothesis of colocalisation is not rejected with a chisquare statistic of 5.27 based on 7 degrees of freedom ($n-1$ where the n is the number of components tested, and one degree of freedom was used in estimating η), giving a p value of 0.63. The `summary()` method returns a named vector of length 4 containing this information.

If more information is needed about η, then this is available if the `bayes` argument is supplied:

```
> ct.pcs.bayes <- coloc.test(pcs.1,pcs.2, bayes=TRUE)

........

> ci(ct.pcs.bayes)

$eta.mode
[1] 1.119514

$lower
[1] 0.5911005

$upper
[1] 2.141321

$level.observed
[1] 0.948087

$interior
[1] TRUE
```

Bayesian model averaging

This approach appears simpler. There is no need to do any preparatory work, you require only a single function:

```
> ct.bma <- coloc.bma(data@df1, data@df2, family1="gaussian", family2="gaussian")
```

```
Dropped 0 of 10 SNPs due to LD: r2 > 0.95
 10 SNPs remain.
Restricting model space.
 8 SNPs have single SNP posterior probabilities < 0.01
Models containing only these SNPs will not be explored.
Fitting 17 multi SNP models to dataset 1
Fitting 17 multi SNP models to dataset 2
Averaging coloc testing over 3 models with posterior probabilities >= 2.3e-17
...
```

```
> ct.bma.bayes <- coloc.bma(data@df1, data@df2, family1="gaussian", family2="gaussian
```

```
Dropped 0 of 10 SNPs due to LD: r2 > 0.95
 10 SNPs remain.
Restricting model space.
 8 SNPs have single SNP posterior probabilities < 0.01
Models containing only these SNPs will not be explored.
Fitting 17 multi SNP models to dataset 1
Fitting 17 multi SNP models to dataset 2
Averaging coloc testing over 3 models with posterior probabilities >= 2.3e-17
...................
```

However, `coloc.bma()` is doing quite some work to cover the model space efficiently, and it is important to understand how it does this. First, the `r2.trim` parameter is used to "tag" the SNPs - a subset of SNPs are selected so that no pair have $r^2 >$ `r2.trim`. The default value is 0.95 and the idea is that models containing SNPs with very similar genotypes provide little additional information, so the p value need be averaged over only one of each such group. Lower values of `r2.trim` will produce a sparser model space and so decrease computation. Second, the `thr` parameter is used to discard SNPs which are uninformative with regards the phenotype, that is, if pp_{ij} is the posterior probability of inclusion in single SNP models for SNP i, trait j, the set of discarded SNPs is formed by those for which $pp_{i1} <$ `thr` and $pp_{i2} <$ `thr`. Models containing **only** SNPs from this set will be ignored. Note that models containing one SNP from this set and one SNP *not* in the set **will** be evaluated.

Finally, you should tell `coloc.bma()` how many SNPs should be included in each model. The default is `nsnps=2`, 3 appears slightly more powerful but will geneally require considerably more computation, whilst values of 4 and above are both unlikely to provide more information and very unlikely to be computed in any reasonable time for interactive work.

Bayes Factors

TODO

Configuration analyses

The idea behind the configuration analysis is that the association of each trait with SNPs in a region may be summarised by a vector of 0s and at most a single 1, with the 1 indicating the causal SNP (so, assuming a single causal SNP for each trait). The posterior probability of each possible configuration can be calculated and so, crucially, can the posterior probabilities that the traits share their configurations. This allows us to estimate the support for the following cases:

- H_0: neither trait has a genetic association in the region
- H_1: only trait 1 has a genetic association in the region
- H_2: only trait 2 has a genetic association in the region
- H_3: both traits are associated, but with different causal variants
- H_4: both traits are associated and share a single causal variant

The function `coloc.config` is ideally suited to the case when only summary data are available, when a `data.frame` containing three columns giving the p values for each SNP and the SNP's minor allele frequency should be supplied. Here, we have simulated full genotype data, so I will use the wrapper function `coloc.config.datasets()` for the analysis. This makes use of functions in the snpStats package to calculate single SNP p values quickly.

```
> library(snpStats)
> ct.conf <- coloc.config.datasets(data@df1, data@df2, response1="Y", response2="Y")

coercing object of mode  numeric  to SnpMatrix
coercing object of mode  numeric  to SnpMatrix
PP.H0.abf PP.H1.abf PP.H2.abf PP.H3.abf PP.H4.abf
  0.99400   0.00120   0.00181   0.00251   0.00034
[1] "PP abf for shared variant: 0.034%"
```