The genetics package

Utilities for handling genetic data

by Gregory R. Warnes

Purpose

The genetics package provides classes and methods for handling genetic data. Friedrich ('Fritz') Leisch and I collaborated on the design of the package. I was motivated by the desire to provide a natural way to include single-locus genetic variables in statistical models. Fritz also wanted to support multiple genetic changes spread across one or more genes. While my goal has largely been realized, more work is necessary to fully support Fritz's goal.

Current Status

As of version 0.5.0 the library includes classes and methods for creating, representing, and manipulating genotypes (unordered allele pairs) and haplotypes (ordered allele pairs). Genotypes and haplotypes can be annotated with chromosome, locus, gene, and marker information. Utility functions compute genotype and allele frequencies, flag homozygotes or heterozygotes, flag allele carriers of certain alleles, count the number of a specific allele carried by an individual, extract one or both alleles, and test Hardy-Weinberg equilibrium. These functions make it easy to create and use single-locus genetic information.

Creating variables representing genotype in a statistical package often requires considerable string manipulation. The code for the `genotype` function has been designed to remove this requirement. It allows alleles pairs to be specified in four ways:

- A single vector with a character separator:

```
g1 <- genotype( c('A/A','A/C','C/C','C/A',
                  NA,'A/A','A/C','A/C') )
g3 <- genotype( c('A A','A C','C C','C A',
                  '','A A','A C','A C'),
                sep=' ', remove.spaces=F)
```

- A single vector with a positional separator

```
g2 <- genotype( c('AA','AC','CC','CA','',
                  'AA','AC','AC'), sep=1 )
```

- Two separate vectors

```
g4 <- genotype(
        c('A','A','C','C','','A','A','A'),
        c('A','C','C','A','','A','C','C')
        )
```

- A dataframe or matrix with two columns

```
gm <- cbind(
        c('A','A','C','C','','A','A','A'),
        c('A','C','C','A','','A','C','C') )
g5 <- genotype( gm )
```

A second difficulty with variables representing genotypes is the need to extract different information at different times. Each of the three basic ways of modeling the effect the allele combinations is supported by the `genetics` package:

categorical Each allele combination acts differently.

This situation is handled by entering the `genotype` variable without modification into a model. In this case, it will be treated as a factor:

```
lm( outcome ~ genotype.var + confounder )
```

additive The effect depends on the number of copies of a specific allele (0, 1, or 2).

The function `allele.count( gene, allele )` returns the number of copies of the specified allele.

```
lm( outcome ~ allele.count(genotype.var,'A')
              + confounder )
```

dominant/recessive The effect depends only on the presence or absence of a specific allele.

The function `carrier( gene, allele )` returns a boolean flag if the specified allele is present:

```
lm( outcome ~ carrier(genotype.var,'A')
              + confounder )
```

Implementation

The basic functionality of the `genetics` package is provided by the `genotype` class and the `haplotype` class, which is a simple extension of the former. In designing the `genotype` class, we had several goals. First, we wanted to be able to manipulate both alleles as a single variable. Second, we needed a clean way of accessing the individual alleles. Third, a genotype variable should be able to be stored in dataframes as they are currently implemented in R. Fourth, storage should be space-efficient.

After considering several potential implementations, we chose to implement the genotype class as an extension to the in-built factor variable type with additional information stored in attributes. Genotype objects are stored as factors

and have the class list `c("genotype","factor")`. The names of the factor levels are constructed as `paste(allele1,"/",allele2,sep="")`. Since most genotyping methods do not indicate which allele comes from which member of a chromosome pair, the alleles for each individual are placed in a consistent order controlled by the `reorder` argument. In cases when the allele order is informative, the `haplotype` class, which preserves the allele order, should be used instead.

The set of allele names is stored in the attribute `allele.names`. A translation table from the factor levels to the names of each of the two alleles is stored in the attribute `allele.map`. This map is a two column character matrix with one row per factor level. The columns provide the individual alleles for each factor level. Accesing the individual alleles, as performed by the `allele` function, is accomplished by simply indexing into this table,

```
allele.x <- attrib(x,"allele.map")
alleles.x[genotype.var,which]
```

where `which` is `1`, `2`, or `c(1,2)` as appropriate.

Finally, there is often additional meta-information associated with a genotype. The functions `locus`, `gene`, and `marker` create objects to store information, respectively, about genetic loci, genes, and markers. Any of these objects can be included as part of a genotype object using the `locus` argument, which creates a `locus` attribute in the genotype object. The print and summary functions for genotype objects properly display this information when it is present.

This implementation of the genotype class met our four design goals and offered an additional benefit: the default behavior for factors is similar to the desired behavior for genotypes. Consequently, relatively few additional methods needed to written. Further, in the absence of the genetics package, the information stored in genotype objects is still accessible in a reasonable way.

The `genotype` class is accompanied by a full complement of helper methods for standard R operators (`[]`, `[<-`, `==`, etc.) and object methods (`summary`, `print`, `is.genotype`, `as.genotype`, etc.). The `genetics` package provides the additional functions:

HWE.test Estimates disequilibrium parameter and test the null hypothesis that Hardy-Weinberg equilibrium holds.

allele Extracts individual alleles. matrix.

allele.names Extracts the set of allele names.

homozygote Creates a logical vector inidicating whether both alleles of each observation are the same.

heterozygote Creates a logical vector indicating whether the alleles of each observation differ.

carrier Creates a logical vector indicating whether the specified alleles are present.

allele.count Returns the number of copies of the specified alleles carried by each observation.

getlocus Extracts locus, gene, or marker information.

For complete details on the objects and functions provided by the `genetics` package, please see the help pages `?genotype`, `?HWE.test`, `?homozygote`, `?locus`, and `?HWE.test` or the corresponding auto-generated documentation.

Example

Here is a partial session using tools from the genotype package to examine the features of 3 simulated markers and thier relationships with a continuous outcome:

```
> library(genetics)

Attaching package 'genetics':


        The following object(s) are masked from package:base :

         as.factor

> ## Create a sample dataset with 3 SNP markers
>
> g1 <- sample( x=c('C/C', 'C/T', 'T/T'),
+               prob=c(.6,.2,.2), 20, replace=T)
> g2 <- sample( x=c('A/A', 'A/G', 'G/G'),
+               prob=c(.6,.1,.5), 20, replace=T)
> g3 <- sample( x=c('C/C', 'C/T', 'T/T'),
+               prob=c(.2,.4, 4), 20, replace=T)
>
> y <- rnorm(20) + (g1=='C/C') +
+      0.25 * (g2=='A/A' | g2=='A/G')
>
> ## Form into a data frame
> data <- data.frame( y, g1, g2, g3)
>
> # Create marker labels for the data

[...]

> a1691g  <- marker(name="A1691G",
+                   type="SNP",
+                   locus.name="MBP2",
+                   chromosome=9,
+                   arm="q",
+                   index.start=35,
+                   bp.start=1691,
+                   relative.to="intron 1")
>
>
```

```
[...]

>
> data$g1 <- genotype(data$g1, locus=c104t)
> data$g2 <- genotype(data$g2, locus=a1691g)
> data$g3 <- genotype(data$g3, locus=c2249t)
>
> data
              y  g1  g2  g3
1  -0.084796634 T/T G/G T/C
2   1.454537575 C/C G/G T/T
3  -0.899625344 T/T G/G T/T
4  -1.980679630 C/T A/A T/T
5   0.231087028 C/T A/A T/T
6   2.588083646 C/C A/A T/C
7   0.209338731 C/C A/A T/T
8   1.435823157 C/T G/G T/T
9  -0.078796949 C/C G/G T/T
10 -2.091110058 C/T A/A T/T
11 -0.842655686 C/T G/G T/T
12  1.316828279 C/C G/G T/T
13  0.470126626 C/T A/A T/T
14 -0.364828611 T/T G/A T/T
15 -0.002438264 C/T A/A T/C
16  0.949432430 C/C G/G T/T
17 -0.096626850 C/T G/A T/T
18  1.065637984 T/T A/A T/T
19  0.817213289 C/C A/A T/T
20  0.644714638 C/T G/G T/T
>
> summary(data$g2)

Marker: MBP2:A1691G (9q35:1691) Type: SNP

Allele Frequency:
  Count Proportion
A    20        0.5
G    20        0.5

Genotype Frequency:
    Count Proportion
A/A     9       0.45
G/A     2       0.10
G/G     9       0.45

> HWE.test(data$g2)

Test for Hardy-Wienburg-Equilibrium

Call:
HWE.test.genotype(x = data$g2)

Disequlibrium Estimate (D-hat):
        Observed Expected Obs-Exp D-hat
G/G            9        5       4   0.2
G/A            2       10      -8  -0.2
A/A            9        5       4   0.2
Overall       20       NA      NA   0.2

Significance Test:

        Pearson's Chi-squared test with simulated
        p-value (based on 10000 replicates)

data:  data$g2
X-squared = 12.8, df = NA, p-value = 0.0011



>
> summary(lm( y ~ homozygote(g1,'C') +
                allele.count(g2, 'G') +
+                 g3, data=data))
+
Call:
lm(formula = y ~ homozygote(g1, "C") + allele.count(g2, "G") +
    g3, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-1.6686 -0.6625 -0.0172  0.6973  1.6196

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                 0.3499     0.6229   0.562   0.5821
homozygote(g1, "C")TRUE     1.2124     0.4778   2.537   0.0220 *
allele.count(g2, "G")       0.1193     0.2429   0.491   0.6298
g3T/T                      -0.7724     0.6414  -1.204   0.2460
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.013 on 16 degrees of freedom
Multiple R-Squared: 0.3405,     Adjusted R-squared: 0.2169
F-statistic: 2.754 on 3 and 16 DF,  p-value: 0.07661
```

Conclusion

The genetics package has already proven useful in my work here at Pfizer. I hope that it will also be of user to others who need to analyze genetic data.

In the future I expect to add functions to compute pairwise disequilibrium, perform haplotype imputation, and generate standard genetics plots. I welcome comments and contributions.

Gregory R. Warnes
Pfizer Global Research and Development
gregory_r_warnes@groton.pfizer.com