QTL-mapping with Random Forest

Jan Grossbach

April 12, 2017

1 Introduction

Inter-individual variation in a quantitative trait such as body height, crop yield or gene expression can partly be explained by the differences in genotypes. A host of machine learning approaches attempt to do this by learning which genetic information can be reliably used to predict the expression of a trait. A genetic variant that regulates such a quantitative trait is referred to as a quantitative trait locus (QTL). One of these approaches is Random Forest (RF). RF generates decision trees that group samples with a similar phenotype together by separating them based on genetic predictors that correlate with the trait. RF differs from linear methods in that it uses markers sequentially to separate increasingly smaller groups of samples which allows RF to account for complex interactions between genetic predictors. In the following the R-implementation of Random Forest randomForest is used. Other packages that provide output in the same format are suitable as well. In addition to the functions from randomForest some functions supplied with the accompanying R-package are used.

2 Preparing the data

The data used in this tutorial is the expression of SPNCRNA.1571 in 46 fission yeast strains. All data is provided in the same folder as this tutorial. Several pieces of information are required as input to determine if a trait is significantly regulated by a specific locus. Aside from the trait value for each individual we need to know its genotype. This information can be parsed to a format suitable for the mapping functions supplied here with `preMap`. The genotype should be provided as a matrix with one column per marker and one row for each unique genotype. Genotypes have to be supplied as binary information (0,1,NA). The phenotypical data is to be provided either as a vector (if only one trait should be mapped) or as a matrix with one row per trait. In addition to the genotype and phenotype, a vector specifying which sample belongs to which genotype is required. An example for this can be found below. This approach is designed to be able to handle missing allele-information. It is important that any missing genotype-data are set to NA (and not e.g. 0.5). It is possible to supply replicate measurements for the same trait and strain/genotype. We

advise against this since it increases the runtime of the method without adding new information. An alternative would be to average the available data points for the same strains to decrease noise in the data. To account for the heritability of a trait all permutations are done by systematically permuting the associations of phenotype and genotype. If this is not possible due to a different number of replicates per genotype the final p-values may not reflect the true significance of the association between a marker and a trait. `preMap` combines the genotype- and phenotype-information and handles the preprocessing of the genotype and the phenotype. In this tutorial we deal with the most basic mapping task, where a single trait with one condition and with no extra replicates is analyzed. The code used in this tutorial can also be found in tutorialCode.R to avoid pointless copying and pasting.

```
> library(RFQTL)
> genotype <- as.matrix(read.table("genotype.tsv",row.names = 1))
> phenotype <- read.table(file = "phenotype.tsv")
> strainNames <- phenotype[,1]
> phenotype <- phenotype[,2]
> sampleInfo <- sapply(strainNames,FUN=function(x){
+   which(rownames(genotype)==x)
+ })
> mappingData <- preMap(genotype=genotype,
+                       phenotype=phenotype,
+                       sampleInfo=sampleInfo,
+                       scale=T)
```

3 Mapping the trait

`rfMapper` is used to explain variance within the phenotype with differences in the genotypes. It returnes a vector of scores that indicate how close a trait and a marker are linked. All necessary data for the mapping is contained in the object generated with `preMap`. The number of trees that is generated can be controlled with the parameters `ntree` and `nforest`. Note that the final number of decision trees is the product of both values.

```
> library(randomForest)
> realScores <- rfMapper(mappingData = mappingData,
+                        permute = F,
+                        nforest = 100,
+                        ntree = 150)
```

To assess the significance of an association between a trait and a marker `rfMapper` has to run twice: once to assess how important the marker is to explain the observed trait values and once to generate a null distribution of marker importances to which the real importance can be compared. The parameter `permute` controls specifies if the function should map the real trait or

generate a null distribution. If `permute=TRUE`, i.e. if a null distribution should be generated, the `nPermutations` parameter specifies how many permutations should be performed. If more than one trait were preprocessed with `preMap` at the same time, the trait which should be mapped has to be specified with `nTrait`.

```
> permutedScores <- rfMapper(mappingData = mappingData,
+                            permute = T,
+                            nforest = 100,
+                            ntree = 150,
+                            nPermutations=10,
+                            file="perm1.RData",
+                            nCl=4,
+                            clType="SOCK")
```

Since many time-consuming permutations are necessary to accurately estimate the significance of a linkage, a folder with precomputed permutations is included in the home-directory of this tutorial. The `file`-parameter specifies a path where the output should be stored as a .RData-object. This is only true for permutations, the real scores are returned by the function. `rfMapper` can be used in parallel. The type of cluster that should be created has to be specified as `clType` and the number of cores as `nCl`. The cluster-functions from the `snow`-package are used, therefore only cluster-types implemented in `snow` can be used.

4 Estimating the significance of marker/trait-linkages

Predictor-scores can be compared to predictor-specific empirical null distributions with the `pEst`-function. The parameter `path` specifies a directory from which `pEst` loads all RData-files used to determine the empirical p-value of a specific marker. The parameter `markersPerIteration` does not change the results in any way but it specifies for how many markers the null distributions should be loaded into the working memory at a time. A higher value for this parameter reduces the computation time at the cost of higher memory-usage. If the parameter `printProg` is set to `TRUE`, the progress will be printed to the screen. Any correction for multiple testing implemented in `p.adjust` may be specified as `pCorrection`. The output will then consist of p-values that are corrected for multiple testing.

```
> pValues <- pEst(path="permutations_prepared/",
+                 scores=realScores,
+                 markersPerIteration = 350,
+                 printProg = T,
+                 pCorrection = "fdr")
```

```
[1] 350
[1] 688
```

5 Joining significant markers to QTL-regions

Markers that are significantly regulating the same trait and are in linkage disequilibrium likely belong to the same QTL. `QTLgrouper` is a function that further processes the significant marker-trait-linkages. Directly neighboring significant markers are assumed to belong to the same QTL. Markers that are separated by not more than a predefined amount of other markers and at the same time show sufficient correlation are also assumed to belong to the same QTL. If regions defined in this fashion contain markers that show strong correlation with markers in other distinct significant regions, these regions are assumed to belong to the same QTL. A QTL defined by `QTLgrouper` can therefore contain distinct regions which are separated by genomic regions not belonging to the QTL. The results are returned as a list-object where each entry corresponds to one QTL.

```
> pValuesX <- pValues[mappingData$genotype2group]
> chrVec <- read.csv("chrVec.tsv")[,1]
> QTL_list <- QTLgrouper(pmat = pValuesX,
+                        sigThreshold = 0.1,
+                        corThreshold = 0.8,
+                        distThreshold = 9,
+                        genotype = genotype,
+                        chrVec = chrVec)
```

6 Storing results in the .qtl format

The objects returned by `QTLgrouper` can be saved in a format that is easier to navigate by eye. The function `writeQTL` can be used to generate .qtl-files that contain all QTL for one trait. `writeQTL` automatically generates as many files as there are different traits with significant QTL. These files can be read with `readQTL`.

```
> markerPositions <- read.table("markerPositions.tsv",sep="\t",header=T)
> writeQTL(QTLlist = QTL_list,traitNames = "SPNCRNA.1571",markerPositions = markerPositions,
> qtl <- readQTL(path = "myResults.qtl")
```