Categorical Permulation Walkthrough

Ruby Redlich

10 January, 2023

Introduction

This walkthrough explains how to perform permulation analysis to calculate empirical p-values for genes and pathways for Categorical traits. For a description of what permulations are and why they are important, refer to the Permulations Walkthrough.

Categorical Permulations Overview

Categorical permulations are accomplished slightly differently than binary and categorical permulations because the simulation step is not based on a Brownian motion model. Instead, a simulated phenotype is generated according to a continuous time Markov Model, the same model that was used to reconstruct the ancestral history of the trait. As with binary and categorical permulations, the simulation is based on a phylogeny with branch lengths representing the average genome-wide evolutionary rate along that branch. Next, 3 steps are taken to ensure that the permulated phenotype contains the same number of species with each trait value as the original phenotype:

1) Rejection: any simulated phenotype in which there are not the same number of **extant** species with each trait value as the original phenotype is rejected.
2) Permutation of internal traits: the simulated values for **internal** species are ignored. Instead, the original trait values for the internal species in the phylogeny are permuted and assigned to internal species in the permulated phenotype.
3) Re-organize internal traits: a search technique similar to simulated annealing is used to re-organize the internal traits relative to the traits of the extant species to improve the likelihood of the permulated phenotype. This generates a plausible trait history that exactly matches trait category counts and has a comparable probabilistic likelihood to the original simulation.

Note that the permulation functions can take a long time to run on large data sets and for large numbers of permulations.

Categorical Permulations

This vignette will use the basal metabolic rate (BMR) categorical phenotype to demonstrate how to run a categorical permulation analysis. This vignette will briefly walk through the steps for ancestral state reconstruction (ASR) and calculating correlation statistics that are required for this analysis, but for more details regarding these steps please refer to the Categorical Walkthrough and the ASR Walkthrough.

Getting Started With a Categorical Trait Analysis in RERconverge

Start by loading the RERconverge library. For more detailed instructions on getting started with RERconverge, refer to the RERconverge Analysis Walkthrough vignette.

```
if (!require("RERconverge", character.only = T, quietly = T)) {
  require(devtools)
  install_github("nclark-lab/RERconverge", ref = "master")
  # ref refers to the branch of RERconverge being installed
}
library(RERconverge)
```

Next read in the phenotype data and the gene trees. Additionally, calculate the relative evolutionary rates. For more details on using `readTrees` and `getAllResiduals` refer to the RERconverge Analysis Walkthrough vignette.

```
# find where the package is located on your machine
rerpath = find.package('RERconverge')

# read in the trees with the given file name
toytreefile = "subsetMammalGeneTrees.txt"
toyTrees=readTrees(paste(rerpath,"/extdata/",toytreefile,sep=""), max.read = 200)

# load the phenotype data into your workspace
# This will create a named vector with the name basalRate
data("basalRate")

# calculate the relative evolutionary rates with getAllResiduals
RERmat = getAllResiduals(toyTrees, useSpecies = names(basalRate))
```

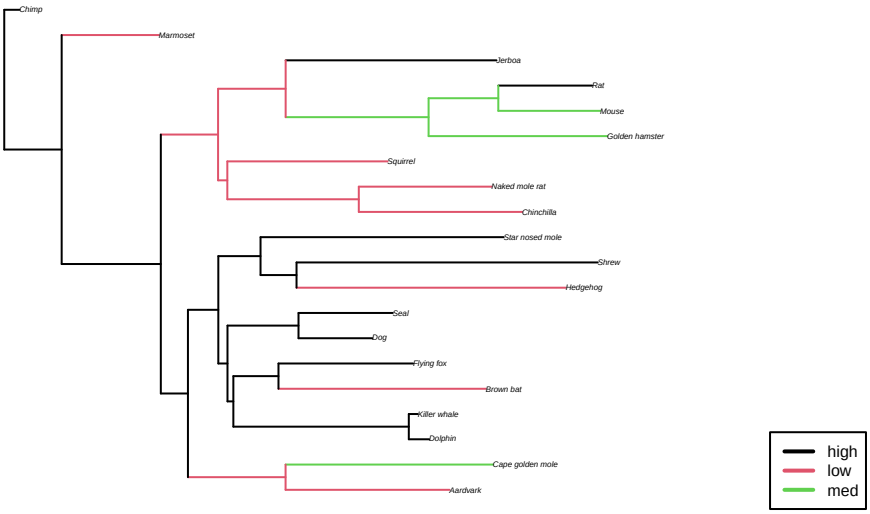
The next steps is to infer the phenotypes of the ancestral species and calculate paths using the function `char2PathsCategorical`. For the purpose of this walkthrough, we will just use `"ARD"` (all rates different) as the model of evolution. (Note that the choice of rate model impacts the ancestral reconstruction and the analysis. Refer to the ASR Walkthrough for more details).

The `toyTrees` object contains a separate gene tree for each gene in the analysis with branch lengths representing the evolutionary rates of that gene. All of the gene trees have the same overall topology as the master tree, but some of them are missing certain species. To handle missing species, RERconverge generates something called paths. For a more detailed discussion of paths see the "RERconverge Analysis Walkthrough" vignette.

```
# get the names of all the species for which there is phenotype data
allspecs = names(basalRate)

# infer ancestral states and calculate paths
charP = char2PathsCategorical(basalRate, toyTrees, useSpecies = allspecs, model = "ARD",
                              plot = TRUE)
```

```
## Species from master tree not present in useSpecies: Platypus,Opossum,Tasmanian_devil,Wallaby,Armadil
```

```
## [1] "The integer labels corresponding to each category are:"
## high  low  med
##    1    2    3
```



Next calculate the association statistics relating the relative evolutionary rates to basal metabolic rate phenotype. For more details on how the output of `correlateWithCategoricalPhenotype` is organized, refer to the Categorical Walkthrough.

```
# Kruskal Wallis/Dunn posthoc testing (default)
cors = correlateWithCategoricalPhenotype(RERmat, charP)

# view the first few results
head(cors[[1]][order(cors[[1]]$P),])
```

```
##             Rho  N           P     p.adj
## ADAD1 0.3229020 35 0.004130593 0.2876621
## ARSA  0.3764943 29 0.005138883 0.2876621
## AP5M1 0.3056010 35 0.005543016 0.2876621
## BRAF  0.3700350 28 0.006768521 0.2876621
## BRSK2 0.3024616 31 0.010706289 0.2907447
## BDH1  0.2717130 34 0.011296477 0.2907447
```

Performing Permulations

The goal of running the permulation analysis is to generate many permulated phenotypes then calculate correlation statistics relating evolutionary rates of genes to the trait for each permulated phenotype. This generates a set of null correlation statistics – the statistics we would expect by chance given the same phylogeny and same numbers of species with each trait value. Permulation p-values are thus the fraction of correlation statistics among the permulated phenotypes that are more extreme than the correlation statistic calculated for the original trait data.

Generate Permulated Phenotypes

To run a permulation analysis, start by generating a set of permulated phenotypes using the function `categoricalPermulations`. In this example we generate 100 permulated phenotypes. For a more rigorous analysis we recommend using 1000 or more permulations (though this may be time consuming). This is because the permulation p-values can only be as precise as one over the number of permulations performed. `categoricalPermulations` takes the following as input:

- `treesObj` : The trees object containing every gene tree returned by `readTrees`
- `phenvals` : The named vector of phenotype data (should be a categorical phenotype)

- `rm` : The rate model. **This should be the same rate model as passed to `char2PathsCategorical` to perform the ancestral reconstruction.**
- `rp` : The root probabilities to use when simulating the phenotype. This gives the probability of each state at the root. The default value is `"auto"`. It can also be a numeric vector with length equal to the number of phenotype categories. Other options are `"stationary"` and `"flat"`. `"flat"` sets the probabilities of all categories at the root equal. `"stationary"` uses the stationary distribution of the transition matrix. For more details refer to the documentation for the castor function `simulate_mk_model` (Louca and Doebeli 2017).
- `ntrees` : the number of permulated phenotypes to generate

The following code generates 100 permulated phenotypes. This step may take a few minutes.

```
perms <- categoricalPermulations(toyTrees, phenvals = basalRate, rm = "ARD",
                                 rp = "auto", ntrees = 100)
```

```
## Fitting transition matrix

## Simulating trees

## Shuffling internal states

## Improving tree likelihoods

## Done
```

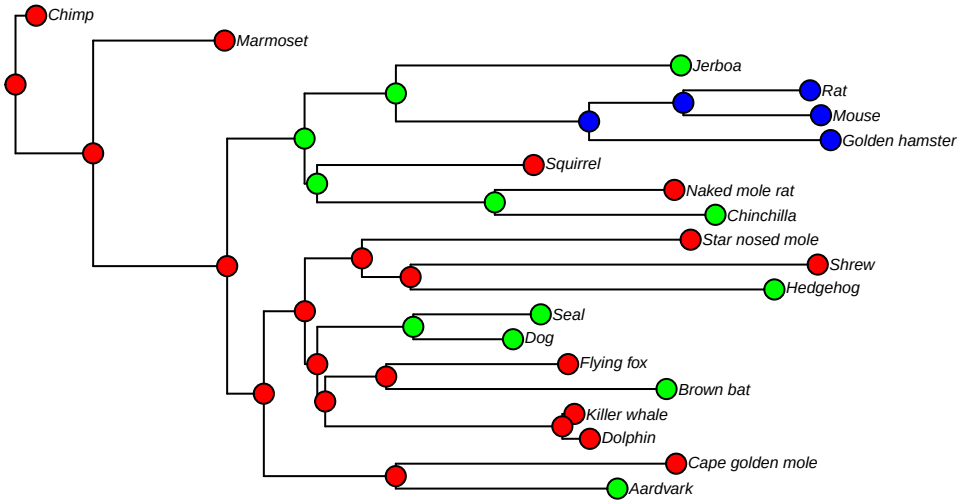
`perms`, the output of `categoricalPermulations` is a 3-element list. The first element `sims` contains the original simulated trees. `sims` is a list of two matrices, `tips` and `nodes`. The matrices have `ntrees` rows corresponding to the `ntrees` simulations. Columns of the `tips` and `nodes` matrices correspond to the extant or internal species respectively. The second element is `trees`. These are the permulated phenotypes. `trees` is an `ntrees`-element list. Each element of `trees` is itself a list containing a `tips` vector and a `nodes` vector corresponding to the states of the extant and internal species. The third element of `perms` is `startingTrees`. `startingTrees` has the same structure as `trees` and corresponds to the permulated phenotypes before step 3, re-organize internal traits (see Categorical Permulations Overview).

Visualize Permulated Phenotypes

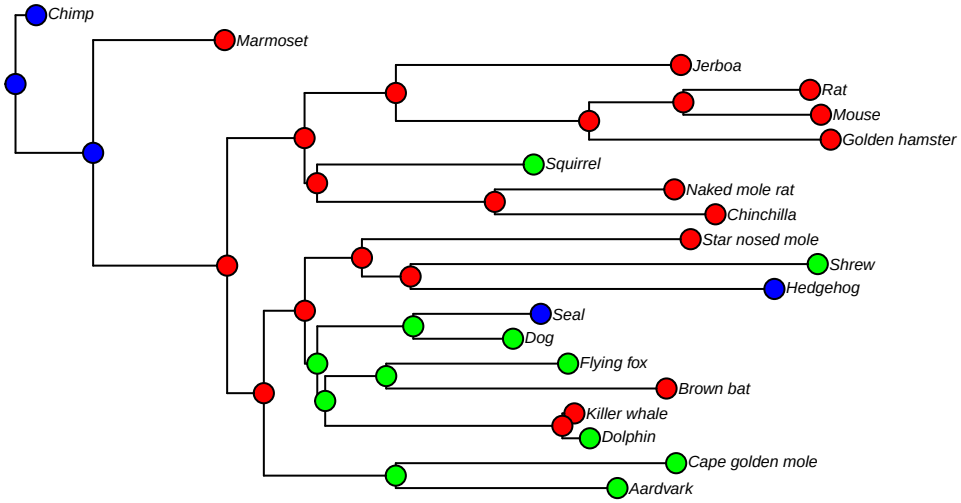
We can easily visualize some of the permulated phenotypes. For convenience we define a function that will plot the states as colored circles on the tree. Note that your trees will look slightly different from the ones shown here.

```
# define a function to plot the permulated phenotypes on the tree
library(RERconverge)
```

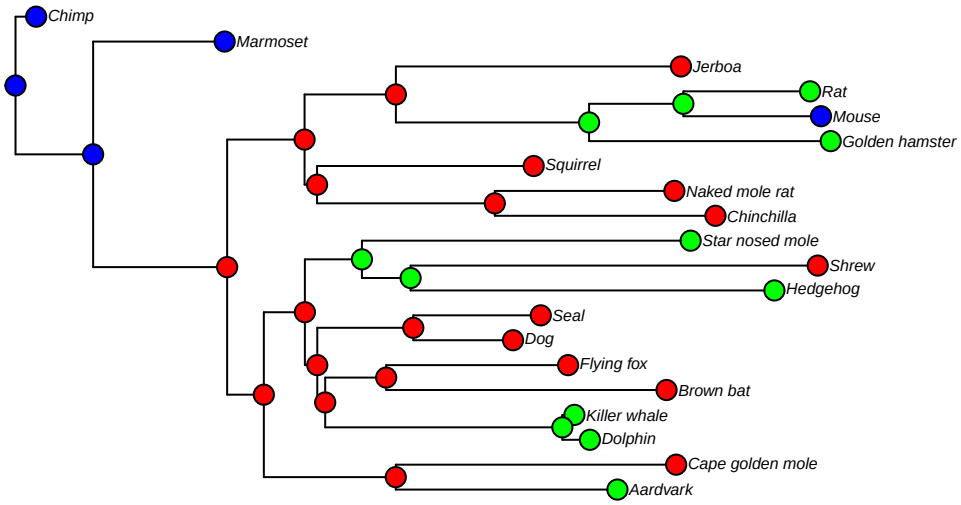
```
plotPermPhen <- function(tree, tips, internal_states) {
  plot(tree, label.offset = 0.005, cex = 0.5)
  tiplabels(pie = to.matrix(tips, sort(unique(tips))),cex = 0.5)
  nodelabels(pie = to.matrix(internal_states, sort(unique(internal_states))), cex = 0.5)
}

# prune the master tree to only contain species for which there are phenotype values
tree = toyTrees$masterTree
tree = pruneTree(tree, names(basalRate))

# plot some of the permulated trees
plotPermPhen(tree, perms$trees[[1]]$tips, perms$trees[[1]]$nodes)
```



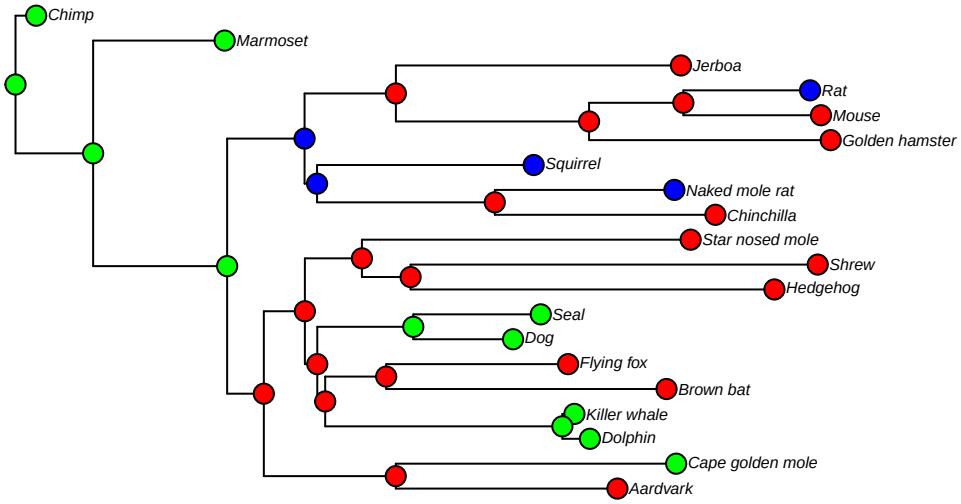

```
plotPermPhen(tree, perms$trees[[25]]$tips, perms$trees[[25]]$nodes)
```



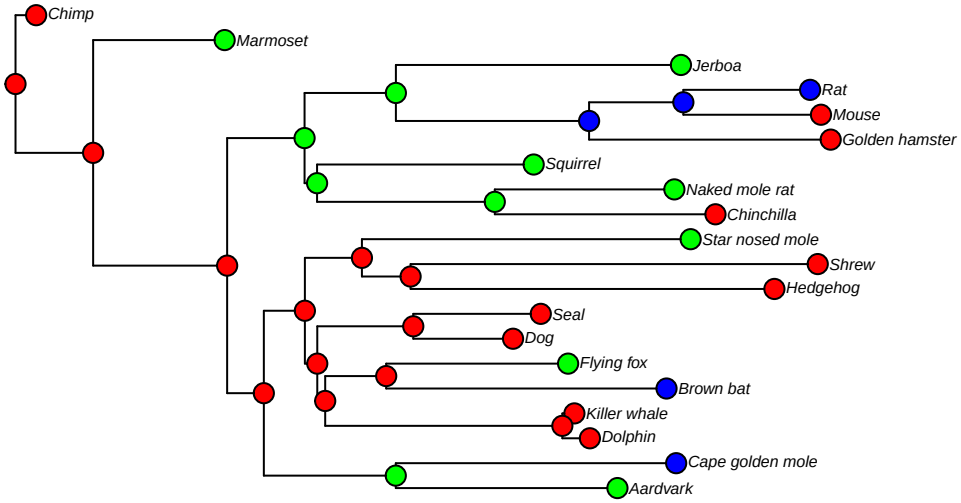

```
plotPermPhen(tree, perms$trees[[50]]$tips, perms$trees[[50]]$nodes)
```




```
plotPermPhen(tree, perms$trees[[75]]$tips, perms$trees[[75]]$nodes)
```




```
plotPermPhen(tree, perms$trees[[100]]$tips, perms$trees[[100]]$nodes)
```



Obtain Permulation P-values

We can obtain permulation p-values using the function `getPermPvalsCategorical`. This function takes as input:

- `realCors` : the correlation statistics object returned by `correlateWithCategoricalPhenotype`.
- `nullPhens` : the permulated phenotypes. This should be the `trees` element in the list returned by `categoricalPermulations`.
- `phenvals` : The named phenotype vector.
- `treesObj` : The trees object returned by `readTrees`.
- `RERmat` : The matrix of relative evolutionary rates returned by `getAllResiduals`.
- `method` : either `"kw"` for Kruskal Wallis or `"aov"` for ANOVA; this should be whichever method was used to calculate the correlation statistics using `correlateWithCategoricalPhenotype`, the default method of which is `"kw"`. If another method is provided that is not `"kw"` or `"aov"`, then the trait will be treated as a binary trait.

```
pres <- getPermPvalsCategorical(realCors = cors, nullPhens = perms$trees,
                                phenvals = basalRate, treesObj = toyTrees,
                                RERmat = RERmat, method = "kw")
```

```
## Generating null paths

## Calculating correlation statistics

## Obtaining permulations p-values

## Done
```

The output of `getPermPvalsCategorical` is a 3-element list. The first element, `res`, has the same format as `cors`, the output of `correlateWithCategoricalPhenotype`, except each data frame has an additional column called `permP` containing the permulation p-values. The second and third elements are `pvals` and `effsize`. Each of these is a 2-element list. The first element contains a (number of genes) x (number of permulations) table containing the p-value or effect size of each gene for each permulation. The second element contains a list of tables for each pairwise test. Each table has dimensions (number of genes) x (number of permulations) and contains the p-value or effect size of each gene for each permulation.

If the trait is binary (has only 2 categories) then the output will look very similar except it will not contain lists of tables for the pairwise tests.

The pairwise tables are named with the integer mappings to the categories e.g. `"1 - 2"`. Recall that the integer mapping is printed by `char2PathsCategorical` and can also be obtained by running the following code:

```
# view the names of the pairwise tables
names(pres$res[[2]])
```

```
## [1] "1 - 2" "1 - 3" "2 - 3"
```

```
# get the category to integer mapping
intlabels = map_to_state_space(basalRate)
print(intlabels$name2index)
```

```
## high  low  med
##    1    2    3
```

```
# view the results ordered by permulation p-value
head(pres$res[[1]][order(pres$res[[1]]$permP),])
```

```
##                      Rho  N           P     p.adj permP
## ADAD1          0.3229020 35 0.004130593 0.2876621  0.00
## AP5M1          0.3056010 35 0.005543016 0.2876621  0.00
## ARSA           0.3764943 29 0.005138883 0.2876621  0.00
## BIRC5          0.3047823 27 0.019021805 0.3216999  0.01
## BRAF           0.3700350 28 0.006768521 0.2876621  0.01
## Em:AC008101.5 0.1797311 33 0.056376766 0.3986397  0.02
```

```
# view the results of the pairwise tests ordered by permulation p-value
head(pres$res[[2]][[1]][order(pres$res[[2]][[1]]$permP),]) # high - low
```

```
##              Rho          P p.adj permP
## ARSA   -2.708968 0.02024788     1  0.00
## BIRC5   1.775278 0.22755608     1  0.01
## BTBD18 -2.502992 0.03694448     1  0.01
## APOH    2.144787 0.09590962     1  0.03
## BRAF   -2.389181 0.05065792     1  0.03
## ABHD5  -1.859582 0.18883420     1  0.04
```

```
head(pres$res[[2]][[2]][order(pres$res[[2]][[2]]$permP),]) # high - med
```

```
##                Rho          P     p.adj      permP
## ACOT13    2.658824 0.02352420 0.7058718 0.00000000
## ADAM1A    2.498884 0.03737552 0.7802140 0.01000000
## ANO2      2.927877 0.01023855 0.7058718 0.01000000
## ADM      -2.376555 0.05242547 0.8348500 0.01000000
## BRAF     -2.726717 0.01919035 0.7058718 0.01000000
## BC118554 -1.767767 0.23129962 1.0000000 0.01388889
```

```
head(pres$res[[2]][[3]][order(pres$res[[2]][[3]]$permP),]) # low - med
```

```
##                     Rho           P     p.adj permP
## Em:AC008101.5  2.397934 0.049463461 0.6740709  0.00
## ADAD1          3.281030 0.003102857 0.3328440  0.00
## AP5M1         -3.209704 0.003986156 0.3328440  0.00
## ARSA           2.691795 0.021320581 0.4705343  0.00
## BIRC5         -2.682493 0.021922723 0.4705343  0.00
## ACTL7B         2.710343 0.020164073 0.4705343  0.02
```

Categorical Permulations for Pathway Enrichment Statistics

For details on how pathway enrichment statistics are calculated, refer to the RERconverge Analysis Walkthrough vignette. Essentially a pathway enrichment analysis identifies groups of genes that are evolving faster or slower with the phenotype of interest. We recommend calculating permulation p-values for the pathway enrichment statistics in addition to the gene-evolutionary rate association statistics due to non-independence between genes in pathways.

Getting Started with Pathway Enrichment

You will need to download the gene sets and gene symbols from GSEA-MSigDB as gmtfile.gmt. Follow the instructions in the "RERconverge Analysis Walkthrough" vignette in order to properly download and save the gmt file in your current working directory. The "RERconverge Analysis Walkthrough" may say to download the file named c2.all.v6.2.symbols.gmt, however if that is not available, c2.all.v7.5.1.symbols.gmt will work. Ensure that the name of the gmt file in your working directory is "gmtfile.gmt".

```
# read in the annotations
annots = read.gmt("gmtfile.gmt")

# format in a list
annotlist=list(annots)
names(annotlist)="MSigDBpathways"
```

Obtain Permulation P-values for Pathway Enrichment Statistics

Calculate Enrichment Statistics for the Original Gene Association Results The first step is the calculate the pathway enrichment statistics for the original gene association results before permulations (the output of `correlateWithCategoricalPhenotype`). This can be done using the function `getRealEnrichments` which calls the RERconverge function, `fastwilcoxGMTall`, to calculate enrichment statistics for the categorical results and the results of each posthoc pairwise test.

Note that running `getRealEnrichments` can be time consuming especially if the number of pairwise tests is large.

`getRealEnrichments` takes the following as input:

- `cors` : the output of `correlateWithCategoricalPhenotype`

- `annotlist` : the pathway annotations formatted as a list
- `outputGeneVals` : the default value is `FALSE`. If set to `TRUE`, the genes in each pathway will be included in the output.

```
# run enrichments
realenrich <- getRealEnrichments(cors, annotlist)
```

The output of `getRealEnrichments` is a 2-element list. The first element contains the enrichment statistics for the categorical correlations results. The second element contains a list of enrichment statistics for each posthoc pairwise test. For more information on interpreting pathway enrichment results, refer to the Enrichment Walkthrough section in the RERconverge Analysis Walkthrough vignette.

Calculate Permulation P-values Recall that `getPermPvalsCategorical` returns a list of p-value matrices and effect size matrices. Each column in these matrices corresponds to the parametric p-values or effect size statistics returned by `correlateWithCategoricalPhenotype` (or `getAllCors`) for one permulated phenotype. To calculate permulation p-values for the enrichment statistics, null enrichment statistics are calculated for each permulated phenotype using a ranked gene list based on the p-values and effect size statistics for that permulated phenotype. This is handled by the functions, `getEnrichPermsCategorical`. Then the permulation p-value is determined by the proportion of times the null enrichment statistics are more extreme than the real enrichment statistics returned by `getRealEnrichments`. This is handled by the function `getEnrichPermPvals`.

During a call to `getEnrichPermsCategorical`, `fastwilcoxGMTall` (the RERconverge function that calculates enrichmentis statistics) is called many times (once per permulation for the categorical results and once per permulations for EACH pairwise test). As a result `getEnrichPermsCategorical` can take a long time to run.

`getEnrichPermsCategorical` takes the following as input:

- `perms`: The output of `getPermPvalsCategorical`; the object containing the null p-values and null enrichment statistics for each permulated phenotype.
- `realenrich`: The output of `getRealEnrichments`; the pathway enrichment statistics on the original gene association results.
- `annotlist`: the list of pathway annotations formatted as a list as shown above

```
# run enrichments permulations
permenrich = getEnrichPermsCategorical(perms = pres, realenrich = realenrich,
                                       annotlist = annotlist)
```

`permenrich`, the output of `getEnrichPermsCategorical`, is a 2-element list. The first element contains a list of tables of P-values and a list of tables of enrichment statistics. There is one table of p-values or enrichment statistics for each annotation pathway set. For the annotation list in this walkthrough these sets are: mgi, canonical, GO, hairfollicle, and tissueannots. The second element contains a list of such lists, one for each posthoc pairwise test.

To calculate permulation p-values, call the function `getEnrichPermPvals` which takes the following as input:

- `permenrich`: the output of `getEnrichPermsCategorical` containing the enrichment statistics and p-values for each permulated phenotype
- `realenrich`: the output of `getRealEnrichments` containing the enrichment statistics and p-values for the original gene association results for the original phenotype.

```
pvals = getEnrichPermPvals(permenrich, realenrich)
```

The output of `getEnrichPermPvals` is also a 2-element list of very similar format to the output of of `getEnrichPermsCategorical` except that instead of tables of p-values and enrichment statistics, there is a list of named numeric vectors of permulation p-values for each pathway in each annotation pathway set.

The code below demonstrates how to view the permulation p-values for the enrichment pathways.

```
# convert the mgi annotations ordered by p-value to a dataframe
df = as.data.frame(pvals[[1]]$MSigDBpathways[order(pvals[[1]]$MSigDBpathways)])
colnames(df) = c("permulation p-values")
head(df)
```

```
##                                                 permulation p-values
## FLECHNER_BIOPSY_KIDNEY_TRANSPLANT_REJECTED_VS_OK_DN           0.0500000
## REACTOME_TRANSPORT_OF_SMALL_MOLECULES                         0.1000000
## IVANOVA_HEMATOPOIESIS_STEM_CELL_AND_PROGENITOR                0.1688312
## REACTOME_METABOLISM_OF_LIPIDS                                 0.2100000
## PEREZ_TP53_TARGETS                                            0.2300000
## BRUINS_UVC_RESPONSE_VIA_TP53_GROUP_A                          0.2700000
```

```
# do the same for the first posthoc pairwise test
# change the number 1 in the second set of brackets (to 2 or 3) to view the other posthoc tests
df = as.data.frame(pvals[[2]][[1]]$MSigDBpathways[order(pvals[[2]][[1]]$MSigDBpathways)])
colnames(df) = c("permulation p-values")
head(df)
```

```
##                                              permulation p-values
## GOBERT_OLIGODENDROCYTE_DIFFERENTIATION_DN          0.06000000
## ZWANG_TRANSIENTLY_UP_BY_2ND_EGF_PULSE_ONLY         0.07000000
## MIKKELSEN_ES_ICP_WITH_H3K4ME3                      0.08695652
## NUYTTEN_EZH2_TARGETS_DN                            0.14583333
## CHEN_METABOLIC_SYNDROM_NETWORK                     0.18000000
## BRUINS_UVC_RESPONSE_VIA_TP53_GROUP_A               0.36000000
```

We are often interested not only in the permulation p-values of the pathways, but the direction and magnitude of the association given by the enrichment statistic. The following code demonstrates how to add the permulation p-values to the original enrichment results.

```
# make a copy of the real enrichment results
enrichWithPvals = realenrich

# add p-values for each annotation set in the first element of enrichWithPvals
for(cnt in 1:length(enrichWithPvals[[1]])) {
  indices = match(rownames(enrichWithPvals[[1]][[cnt]]), names(pvals[[1]][[cnt]]))
  enrichWithPvals[[1]][[cnt]]$permpvals = pvals[[1]][[cnt]][indices]
}

# add p-values for each annotation set in the second element of enrichWithPvals
# (the list of posthoc pairwise tests)
for(j in 1:length(enrichWithPvals[[2]])){
  name = names(enrichWithPvals[[2]])[j] # the name of the pairwise test
  for(cnt in 1:length(enrichWithPvals[[2]][[j]])){
    indices = match(rownames(enrichWithPvals[[2]][[j]][[cnt]]),
                    names(pvals[[2]][[name]][[cnt]]))
    enrichWithPvals[[2]][[j]][[cnt]]$permpvals = pvals[[2]][[name]][[cnt]][indices]
  }
}

# view some of the results
head(enrichWithPvals[[1]]$MSigDBpathways[order(enrichWithPvals[[1]]$MSigDBpathways$permpvals),])
```

```
##                                                        stat       pval
```

```
## FLECHNER_BIOPSY_KIDNEY_TRANSPLANT_REJECTED_VS_OK_DN  0.14572193 0.10851379
## REACTOME_TRANSPORT_OF_SMALL_MOLECULES                0.15520362 0.03774806
## IVANOVA_HEMATOPOIESIS_STEM_CELL_AND_PROGENITOR      -0.14963504 0.11477744
## REACTOME_METABOLISM_OF_LIPIDS                        0.09898990 0.20975206
## PEREZ_TP53_TARGETS                                  -0.08610792 0.30613932
## BRUINS_UVC_RESPONSE_VIA_TP53_GROUP_A                 0.07962963 0.36144024
##                                                         p.adj num.genes
## FLECHNER_BIOPSY_KIDNEY_TRANSPLANT_REJECTED_VS_OK_DN 0.7078327        11
## REACTOME_TRANSPORT_OF_SMALL_MOLECULES               0.7078327        17
## IVANOVA_HEMATOPOIESIS_STEM_CELL_AND_PROGENITOR      0.7078327        10
## REACTOME_METABOLISM_OF_LIPIDS                       0.7078327        15
## PEREZ_TP53_TARGETS                                  0.7078327        13
## BRUINS_UVC_RESPONSE_VIA_TP53_GROUP_A                0.7078327        12
##                                                     gene.vals permpvals
## FLECHNER_BIOPSY_KIDNEY_TRANSPLANT_REJECTED_VS_OK_DN        NA 0.0500000
## REACTOME_TRANSPORT_OF_SMALL_MOLECULES                      NA 0.1000000
## IVANOVA_HEMATOPOIESIS_STEM_CELL_AND_PROGENITOR             NA 0.1688312
## REACTOME_METABOLISM_OF_LIPIDS                              NA 0.2100000
## PEREZ_TP53_TARGETS                                         NA 0.2300000
## BRUINS_UVC_RESPONSE_VIA_TP53_GROUP_A                       NA 0.2700000
```

```
# view some of the results for the first pairwise test
head(enrichWithPvals[[2]][[1]]$MSigDBpathways[order(enrichWithPvals[[2]][[1]]$MSigDBpathways$permpvals)
```

```
##                                              stat      pval     p.adj
## GOBERT_OLIGODENDROCYTE_DIFFERENTIATION_DN   0.13502674 0.1369837 0.9386787
## ZWANG_TRANSIENTLY_UP_BY_2ND_EGF_PULSE_ONLY -0.11600430 0.1540540 0.9386787
## MIKKELSEN_ES_ICP_WITH_H3K4ME3               0.15364964 0.1053633 0.9386787
## NUYTTEN_EZH2_TARGETS_DN                     0.11094891 0.2422642 0.9386787
## CHEN_METABOLIC_SYNDROM_NETWORK              0.08799342 0.2165926 0.9386787
## BRUINS_UVC_RESPONSE_VIA_TP53_GROUP_A        0.09444444 0.2790683 0.9386787
##                                             num.genes gene.vals  permpvals
## GOBERT_OLIGODENDROCYTE_DIFFERENTIATION_DN          11        NA 0.06000000
## ZWANG_TRANSIENTLY_UP_BY_2ND_EGF_PULSE_ONLY         14        NA 0.07000000
## MIKKELSEN_ES_ICP_WITH_H3K4ME3                      10        NA 0.08695652
## NUYTTEN_EZH2_TARGETS_DN                            10        NA 0.14583333
## CHEN_METABOLIC_SYNDROM_NETWORK                     19        NA 0.18000000
## BRUINS_UVC_RESPONSE_VIA_TP53_GROUP_A               12        NA 0.36000000
```

Conclusion

This concludes the walkthrough of how to use the functions for permulations for categorical traits in RERconverge. Thank you!

Louca, Stilianos, and Michael Doebeli. 2017. "Efficient Comparative Phylogenetics on Large Trees." https://doi.org/10.1093/bioinformatics/btx701.