genBART package Vignette

Jacob Cardenas

2017-06-13

BART is a user friendly, point and click, R shiny application that was developed as a biostatistical reporting tool for a broad range of high throughput experiments such as RNA sequencing, Microarray, Flow Cytometry, and Metabolomics. BART provides users with the ability to interactively view and download (all tables and figures generated by the app) the entire analysis workflow:

- Design file and summary statistics
- Quality control metrics
- Unsupervised analysis (heatmaps and clustering)
- Differential gene expression (easily sortable genelists, venn diagrams)
- Gene set analysis
- Correlation analysis

To make the tools that BART offers more readily available, we introduce the genBART package, a series of functions that transforms R objects generated from modeling packages limma, DESeq2, and edgeR into files that can be uploaded into BART. In the following sections, we illustrate how to use the functions available in genBART by walking through the analysis of a longitudinal microarray study tracking gene expression changes in cynomolgus macaques infected with *M. tuberculosis* (Skinner et al.). In addition to microarray data, the study also has flow data that will be incorporated in to demonstrate the flexibility of the BART app.

Summary of genBART package

Table 1: Functions in the genBART package ordered to demonstrate a typical analysis workflow.

Function	Purpose
`desInfo()`	Match design and expression. Define elements of design.
`genDendrograms()`	normalize and cluster expression.
`genModules()`	Generate module (gene set) maps for plotting.
`genModelResults()`	Process and format modeling results.
`qBart()|rBart()`	Run gene set analysis using QuSAGE or ROAST algorthims.
`crossCorr()`	Run pairwise correlations between two data sets.
`genFile()`	Generate and save BART ready .rda file.
`updateFile()`	Update/add to existing BART file.
`runBart()`	Run BART shiny app.

The genBART package contains 10 functions, each of which is listed in the table above. Greater detail on each of these functions and their parameters is available via the help commands in R. Below is a summary of how they are used and integrated into a typical workflow:

1. `desInfo` - Check that the samples in expression match the samples in design. Define elements of experimental design/sample information. This initial step is important as the information provided here is used to determine the various ways the data is normalized, annotate the heat maps produced in BART, and generate module maps for plotting.
2. `genDendrograms` - Use the output from `desInfo` to perform various normalizations (e.g. to healthy controls, baseline samples, etc.) and hiearchical clustering of the genes.

3. `genModules` - Use the output from `desInfo` to generate module maps that show, for each sample, the percentage of probes within a module that are above or below a certain threshold (e.g. mean of healthy controls +- 2 sd).
4. `genModuleResults` - Use result objects produced by limma, DESeq2, or edgeR to process and format for BART.
5. `qBart|rBart` - Given a list of gene sets, run either QuSAGE (by also including `genModuleResults` output as input in `qBart`) or ROAST, a rotation based approach to gene set testing proposed by Wu et al (2010).
6. `crossCorr` - Calculate pairwise correlations and p-values between the variables in two data frames or matrices, with the option to correlate by a grouping factor (e.g. time in a longitudinal setting). Flexible to correlate data from different platforms (e.g. microarray vs flow).
7. `genFile` - Generate and save a BART ready .rda file based on the output objects from the functions above. It is not necessary to run all of the functions in order to generate a BART file, making genBART flexible to the project at hand and the stage of analysis a user is at.
8. `updateFile` - Allows the user to easily update and/or add on to an exisiting BART file without having to re-run all of the functions.
9. `runBart` - Run the BART shiny app.

Design Information

Before running any analysis, the first step in generating a BART ready file is to define the elements of the design file and sample information. Keep in mind that not every one of these elements is necessary for the rest of the genBART functions to run, so in practice, much of this section is left to user discretion and the particular project at hand.

```
library(genBART)
library(limma)
```

Below are the first six lines of the design file from the TB microarray study described in the introduction.

```
head(tb.design)
#>                 columnname monkey_id timepoint sample_id clinical_status
#> 1  AVG_Signal__7196763044_K        M1         0   M1-Pre1          Active
#> 5  AVG_Signal__6303256034_B        M1        20    M1-D20          Active
#> 7  AVG_Signal__6303256020_I        M1        42    M1-D42          Active
#> 22 AVG_Signal__7196763078_I       M11         0  M11-Pre1          Latent
#> 26 AVG_Signal__7196771011_D       M11        20   M11-D20          Latent
#> 28 AVG_Signal__7196763087_B       M11        42   M11-D42          Latent
#>    timerange
#> 1        Pre
#> 5      Early
#> 7     Middle
#> 22       Pre
#> 26     Early
#> 28    Middle
```

For this study, the function call would be:

```
des.info <- desInfo(y = tb.expr, design = tb.design, data_type = "micro",
                    columnname = "columnname", long = TRUE,
                    sample_id = "sample_id", patient_id = "monkey_id",
                    time_var = "timepoint", baseline_var = "timepoint", baseline_val = 0,
                    control_var = "clinical_status", control_val = "Latent",
                    summary_var = NULL, responder_var = "clinical_status",
                    project_name = "TB")
```

We proceed similarly for the flow data:

```
des.info.flow <- desInfo(y = tb.flow, design = tb.flow.des, data_type = "flow",
                         columnname = "columnname", long = TRUE, patient_id = "monkey_id",
                         baseline_var = "timepoint", baseline_val = 0,
                         control_var = "clinical_status", control_val = "Latent",
                         time_var = "timepoint", summary_var = NULL,
                         responder_var = "clinical_status", sample_id = "sample_id",
                         project_name = "TB")
```

Please refer to the R help for `desInfo` for more details on function parameters.

Normalization and Clustering

Next we generating dendrograms through hierarchical clustering. This step is necessary to produce the heat maps that can be viewed and downloaded in BART. This can be done through the `genDendrograms` function, which takes as input the output from `desInfo`.

```
dendros <- genDendrograms(design_info = des.info)
#> [1] "Normalizing Expression and Clustering...."
#> [1] "Clustering Baseline Median Normalized Heatmap"
#> [1] "Clustering Baseline Healthy Normalized Heatmap"
#> [1] "Clustering All Samples Median Normalized Heatmap"
#> [1] "Clustering All Samples Healthy Normalized Heatmap"
#> [1] "Clustering All Samples Baseline Normalized Heatmap"
```

From the output we can see that the data was normalized in three different ways, and this is determined by the study design and the information given to `desInfo`. For example, if the study included healthy controls then additional normalizations to healthies would have been included.

Generate Module Maps

By default, BART automatically generates module maps using a set of 260 biologically similar and generally co-regulated Baylor modules. The user is also able to provide his own list of gene sets to plot. As an example, we formed gene sets by choosing 10 clusters determined through hierarchical clustering. The function call to generate the gene set maps is below.

```
mods <- genModules(design_info = des.info, gene_sets = clusters)
```

Sample limma Analysis

Now we demonstrate a quick walkthrough of limma analysis for microarray and flow data. The example we are using comes from a study in which 38 healthy macaques were infected with *M. tuberculosis*. Data measurements were taken at baseline (before infection) and at 11 additional unequally spaced timepoints up to 6 months post infection. Each macaque was identified as developing symptomatic (active) TB or asymptomatic (latent) infection. For the purposes of illustration and time, a subset of the microarray data was chosen for analysis. We randomly selected 4000 probes from those contained within the first seven rounds of the Baylor modules, 10 macaques, and 3 time points (days 0, 20, and 42). For the flow, data was collected on 19 of the 38 macaques and 13 variables are analyzed. For more details about modeling in limma, refer to the limma User's Guide.

Microarray

Create a grouping factor by combining clinical status and time point.

```
tb.design$Group <- paste(tb.design$clinical_status, tb.design$timepoint, sep = "")
grp <- factor(tb.design$Group)
```

Create a design matrix that includes separate coefficients for each level within the grouping factor so that the desired differences can be extracted as contrasts.

```
design2 <- model.matrix(~0+grp)
colnames(design2) <- levels(grp)
```

Since there are repeated measurements on each monkey, we treat monkey_id as a random effect and estimate the correlation between measurements on the same subject. In limma, this is done using duplicateCorrelation.

```
dupcor <- duplicateCorrelation(tb.expr, design2, block = tb.design$monkey_id)
```

Fit model and extract contrasts of interest.

```
fit <- lmFit(tb.expr, design2, block = tb.design$monkey_id, correlation = dupcor$consensus.correlation)
contrasts <- makeContrasts(A_20vsPre = Active20-Active0, A_42vsPre = Active42-Active0,
                           L_20vsPre = Latent20-Latent0, L_42vsPre = Latent42-Latent0,
                           levels=design2)

fit2 <- contrasts.fit(fit, contrasts)
fit2 <- eBayes(fit2, trend = FALSE)
```

Flow

Analysis of flow data in limma proceeds similarly.

```
tb.flow.des$Group <- paste(tb.flow.des$clinical_status, tb.flow.des$timepoint, sep = "")
grp <- factor(tb.flow.des$Group)

design2 <- model.matrix(~0+grp)
colnames(design2) <- levels(grp)
dupcor <- duplicateCorrelation(tb.flow, design2, block = tb.flow.des$monkey_id)

fit.flow <- lmFit(tb.flow, design2, block = tb.flow.des$monkey_id, correlation = dupcor$consensus.correl

# Add more contrasts since we have all timepoints available. Additional contrasts could be added as wel
contrasts <- makeContrasts(A_20vsPre = Active20-Active0, A_30vsPre = Active30-Active0,
                           A_42vsPre = Active42-Active0, A_56vsPre = Active56-Active0,
                           L_20vsPre = Latent20-Latent0, L_30vsPre = Latent30-Latent0,
                           L_42vsPre = Latent42-Latent0, L_46vsPre = Latent56-Latent0,
                           levels=design2)

fit2.flow <- contrasts.fit(fit.flow, contrasts)
fit2.flow <- eBayes(fit2.flow, trend = FALSE)
```

Generate Model Results File

The next step in the process is to generate a model results file that is formatted specifically for BART. This can be done by using the `genModelResults` function, which takes as input the results output from `desInfo`, as well as models run in limma, DESeq2, or edgeR. In our example, we are ran limma models on both microarray and flow data.

```
mod_results <- genModelResults(design_info = des.info, object = fit2, lm_Fit = fit, method = "limma")
mod_results.flow <- genModelResults(design_info = des.info.flow, object = fit2.flow, lm_Fit = fit.flow,
                                    method = "limma")
```

Gene Set Analysis

Next we run gene set analysis by incorporating functions available in the `qusage` package, which tests whether the average log2 fold change of the genes within a gene set are different than 0. The The function `qBart` simplifies the process by requiring only two input parameters, the model results object produced by `genModelResults` and a list of gene sets.

```
qus <- qBart(object = mod_results, gene_sets = modules)
#> Aggregating gene data for gene sets.Done.
#> Calculating VIF's on residual matrix.
#> Q-Gen analysis complete.Aggregating gene data for gene sets.Done.
#> Calculating VIF's on residual matrix.
#> Q-Gen analysis complete.Aggregating gene data for gene sets.Done.
#> Calculating VIF's on residual matrix.
#> Q-Gen analysis complete.Aggregating gene data for gene sets.Done.
#> Calculating VIF's on residual matrix.
#> Q-Gen analysis complete.
```

Correlations

In addition to providing tools for the standard analysis workflow, BART also offers a tool that allows users to visualize and sort through potentially hundreds of thousands pairwise correlations between clinical outcomes. The `crossCorr` function takes two data frames of outcomes and outputs a long format file of all pairwise correlations. We walk through an example in which we wish to correlate module activity scores with flow variables. Since this is a longitudinal setting, we correlate by time.

```
# Create time variable
time <- module.as$time
module.as$time <- NULL

# Format flow data to run correlations and match flow samples with module.as
flow <- data.frame(t(tb.flow))
flow <- flow[match(rownames(module.as), rownames(flow), nomatch = 0), ]

# Run correlations formatted for BART
corrs <- crossCorr(x = module.as, y = flow, by = time, by_name = "days",
                   description = "Mod.Act.Score vs Flow", x_var = "Mod.Act.Score",
                   y_var = "Flow", method = "spearman")
```

Generate BART file

The last step in the process is to generate the final file that can be uploaded into BART. This is done through the function `genFile`, which takes as its arguments the objects generated from the previous functions. Not every object is required to generate a BART file. For example, if gene set analysis and correlations are not run, `desInfo`, `genDendrograms`, `genModules`, and `genModelResults` objects can still be run through `genFile`. BART will only show the results that are uploaded in.

```
genFile(design_info = list(des.info, des.info.flow), module_maps = mods, dendros = dendros,
        model_results = list(mod_results, mod_results.flow))
```

In our example, we did run gene set analysis and correlations. Adding these results into BART is made simple through the `updateFile` function. The user must simply provide the path to the BART file that needs to be updated and add the new additions in the same way that `genFile` takes them.

```
path <- paste(getwd(), "/", des.info$project_name, " Pipeline", sep = "")
updateFile(load.path = path, qusage_results = qus, corr_results = list(corrs))
```

Run BART App

Now that a BART file has been created, it is now time to begin using the app by running the function `runBart`. An in-app vignette is provided that walks through each of the tools BART provides and how to use them.

```
runBart()
```