Introduction to the *ASSIGN* Package

Ying Shen, David Jenkins, and W. Evan Johnson

April 24, 2017

Contents

1 Introduction

This vignette provides an overview of the Bioconductor package *ASSIGN* (Adaptive Signature Selection and InteGratioN) for signature-based profiling of heterogeneous biological pathways. *ASSIGN* is a computational tool used to evaluate the pathway deregulation/activation status in individual patient samples. *ASSIGN* employs a flexible Bayesian factor analysis approach that adapts predetermined pathway signatures derived either from a literature search or from perturbation experiments to create cell-/tissue-specific pathway signatures. The deregulation/activation level of each context-specific pathway is quantified to a score, which represents the extent to which a patient sample matches the pathway deregulation/activation signature.

Some distinctive features of *ASSIGN* are:

1. **Multiple Pathway Profiling**: *ASSIGN* can profile multiple pathway signatures simultaneously, accounting for 'cross-talk' between interconnected pathway components.
2. **Context specificity in baseline gene expression**: Baseline gene expression levels (i.e., the gene expression level under normal conditions) may vary widely due to differences across tissue types, disease statuses, or measurement platforms. *ASSIGN* can adaptively estimate background gene expression levels across a set of samples.
3. **Context-specific signature estimation**: *ASSIGN* provides the flexibility to use either an input gene list or magnitudes of signature genes as prior information, allowing for adaptive refinement of pathway signatures in specific cell or tissue types.
4. **Regularization of signature strength estimates**: *ASSIGN* regularizes the signature strength coefficients using a Bayesian ridge regression formulation by shrinking the strength of the irrelevant signature genes toward zero. The parameter regularization constrains the pathway signature to a small group of genes, making the results more biologically interpretable.

2 How to use the ASSIGN package

2.1 Example Data

In the following examples, we will illustrate how to run *ASSIGN* using either the easy to use `assign.wrapper` function for simple analysis or each individual *ASSIGN* step for more detailed intermediate results.

For either analysis, we will first load *ASSIGN* and create a temporary directory "tempdir" under the user's current working directory. All output generated in this vignette will be saved in "tempdir".

```
> library(ASSIGN)
> dir.create("tempdir")
> tempdir <- "tempdir"
```

Next, load the training data, test data, training labels, and test labels. The training dataset is a G (number of genomic measurements) x N (number of samples in pathway perturbation experiments) matrix, including five oncogenic pathways: B-Catenin, E2F3, MYC, RAS, and SRC pathways in this example. The training data labels denote the column indices of control and experimental samples for each perturbation experiment. For example, we specify the column indices of the 10 RAS control samples to be 1:10, and column indices of 10 RAS activated samples to be 39:48. The test dataset is a G (number of genomic measurements) x N (number of patient samples) matrix. The test data labels denote the classes of the N test samples. In our example, test samples 1-53 are adenocarcinoma and samples 54-111 are squamous cell carcinoma. We specify "Adeno" and "Squamous" in the vector of test data labels. Note that the test data labels are optional. *ASSIGN* outputs additional validation plots to evaluate classification accuracy when test data labels are provided.

```
> data(trainingData1)
> data(testData1)
> data(geneList1)
> trainingLabel1 <- list(control = list(bcat=1:10, e2f3=1:10,
+                                       myc=1:10, ras=1:10, src=1:10),
+                                       bcat = 11:19, e2f3 = 20:28, myc= 29:38,
+                                       ras = 39:48, src = 49:55)
> testLabel1 <- rep(c("Adeno", "Squamous"), c(53,58))
```

2.2 Run ASSIGN all-in-one using `assign.wrapper`

We developed an all-in-one `assign.wrapper` function to run *ASSIGN* with one command. For most users, `assign.wrapper` will be sufficient. The `assign.wrapper` function outputs the following files:

- **pathway_activity_testset.csv**: *ASSIGN* predicted pathway activity in test samples.
- **signature_heatmap_testset_prior.pdf**: heatmaps of the expression level of prior signature genes in training samples.
- **pathway_activity_scatterplot_testset.pdf**: scatterplot of pathway activity in test samples. The x-axis represents test samples ordered by pathway activity; the y-axis represents pathway activity.
- **output.rda**: The intermediate results of individual *ASSIGN* functions.
- **parameters.txt**: A log file containing the parameters used for this *ASSIGN* run.

If training data is provided, `assign.wrapper` also outputs the following files:

- **pathway_activity_trainingset.csv**: *ASSIGN* predicted pathway activity in training samples.
- **signature_heatmap_trainingset.pdf**: heatmaps of the expression level of signature genes in training samples.
- **pathway_activity_scatterplot_trainingset.pdf**: scatterplot of pathway activity in training samples.
- **signature_gene_list_prior.csv**: the gene list and prior coefficients for the pathway signature.

When `Adaptive_S` is specified TRUE, `assign.wrapper` also outputs the following files:

- **signature_heatmap_testset_posterior.pdf**: heatmaps of the expression level of posterior signature genes in training samples.
- **posterior_delta.csv**: a csv file of the prior and posterior change in expression and probability of inclusion for each gene in each signature.
- **Signature_convergence.pdf**: A plot of the MCMC convergence.

Finally, if the `testLabel` argument is not NULL, `assign.wrapper` also outputs the following files:

- **pathway_activity_boxplot_testset.pdf**: boxplot of pathway activity in every test class.

Here we illustrate how to run `assign.wrapper` function with three examples. To start, run the code in section 2.1 to create a temporary directory "tempdir" and load training and test datasets. The individual parameters are described in detail in section 2.3 and in the *ASSIGN* reference manual.

Example 1: Training data is available, but a gene list of pathway signature genes is not available:

```
> dir.create(file.path(tempdir,"wrapper_example1"))
> assign.wrapper(trainingData=trainingData1, testData=testData1,
+                trainingLabel=trainingLabel1, testLabel=testLabel1,
+                geneList=NULL, n_sigGene=rep(200,5), adaptive_B=TRUE,
+                adaptive_S=FALSE, mixture_beta=TRUE,
+                outputDir=file.path(tempdir,"wrapper_example1"),
+                iter=2000, burn_in=1000)
```

Example 2: Training data is available, and a gene list of pathway signature genes is available:

```
> dir.create(file.path(tempdir,"wrapper_example2"))
> assign.wrapper(trainingData=trainingData1, testData=testData1,
+                trainingLabel=trainingLabel1, testLabel=NULL,
+                geneList=geneList1, n_sigGene=NULL, adaptive_B=TRUE,
+                adaptive_S=FALSE, mixture_beta=TRUE,
+                outputDir=file.path(tempdir,"wrapper_example2"),
+                iter=2000, burn_in=1000)
```

Example 3: Training data is not available, but a gene list of pathway signature genes is available:

```
> dir.create(file.path(tempdir,"wrapper_example3"))
> assign.wrapper(trainingData=NULL, testData=testData1,
+                trainingLabel=NULL, testLabel=NULL,
+                geneList=geneList1, n_sigGene=NULL, adaptive_B=TRUE,
+                adaptive_S=TRUE, mixture_beta=TRUE,
+                outputDir=file.path(tempdir,"wrapper_example3"),
+                iter=2000, burn_in=1000)
```

2.3 Run ASSIGN step-by-step

We developed a series of functions: `assign.preprocess`, `assign.mcmc`, `assign.convergence`, `assign.summary`, `assign.cv.output`, and `assign.output` that work in concert to produce detailed results.

2.3.1 `assign.preprocess`

We first run the `assign.preprocess` function on the input datasets. When the genomic measurements (e.g., gene expression profiles) of training samples are provided, but predetermined pathway signature gene lists are not provided, the `assign.preprocess` function utilizes a Bayesian univariate regression module to select a gene set (usually 50-200 genes, but this can be specified by the user) based on the absolute value of the regression coefficient (fold change) and the posterior probability of the variable to be selected (statistical significance). Since we have no predetermined gene lists to provide, we leave the `geneList` option as default NULL. Here we specify 200 signature genes for each of the five pathways.

```
> # training dataset is available;
> # the gene list of pathway signature is NOT available
> processed.data <- assign.preprocess(trainingData=trainingData1,
+                                     testData=testData1,
+                                     trainingLabel=trainingLabel1,
+                                     geneList=NULL, n_sigGene=rep(200,5))
```

Alternatively, the users can have both the training data and the curated/predetermined pathway signatures. Some genes in the curated pathway signatures, although not significantly differentially expressed, need to be included for the purpose of prediction. In this case, we specify the `trainingData` and `geneList` parameters when both the training dataset and predetermined signature gene list are available.

```
> # training dataset is available;
> # the gene list of pathway signature is available
> processed.data <- assign.preprocess(trainingData=trainingData1,
+                                     testData=testData1,
+                                     trainingLabel=trainingLabel1,
+                                     geneList=geneList1)
```

In some cases, the expression profiles (training dataset) is unavailable. Only the knowledge-based gene list or gene list from the joint knowledge of some prior profiling experiments is available. In this case, we specify `geneList` and leave the `trainingData` and `trainingLabel` as default NULL.

```
> # training dataset is NOT available;
> # the gene list of pathway signature is available
> processed.data <- assign.preprocess(trainingData=NULL,
+                                     testData=testData1,
+                                     trainingLabel=NULL,
+                                     geneList=geneList1)
```

The `assign.preprocess` function returns the processed training dataset (trainingData_sub) and test dataset (testData_sub) as well as the prior parameters for the background vector (B_vector), signature matrix (S_matrix) and the probability signature matrix (Pi_matrix) and differentially expressed gene lists of each pathway (diffGeneList). The details of the `assign.preprocess` output are described in the "value" section of the manual page of `assign.preprocess` function. The output data of `assign.preprocess` function are used as the input data of the `assign.mcmc` function.

2.3.2 `assign.mcmc`

For the `assign.mcmc` function, Y, Bg, and X are specified as the output of the `assign.preprocess` function. The `adaptive_B` (adaptive background), `adaptive_S` (adaptive signature) and `mixture_beta` (regularization of signature strength) can be specified TRUE or FALSE based on the analysis context.

When training and test samples are from the different cell or tissue types, we recommend the adaptive background option to be TRUE. Notice that when the training dataset is not available, the adaptive signature option must be set TRUE, meaning that the magnitude of the signature should be estimated from the test dataset. The default `iter` (iteration) is 2000. Particularly, when training datasets are unavailable, it is better to specify the `X` option in the `assign.mcmc` using a more informative X (specify up- or down- regulated genes) to initiate the model.

```
> mcmc.chain <- assign.mcmc(Y=processed.data$testData_sub,
+                           Bg = processed.data$B_vector,
+                           X=processed.data$S_matrix,
+                           Delta_prior_p = processed.data$Pi_matrix,
+                           iter = 2000, adaptive_B=TRUE,
+                           adaptive_S=FALSE, mixture_beta=TRUE)
```

The `assign.mcmc` function returns the MCMC chain recording default 2000 iterations for each parameter. The details of `assign.mcmc` output are described in the "value" section of the manual page of `assign.mcmc` function.

2.3.3 `assign.convergence`

We can make a trace plot to check the convergence of the model parameters through `assign.convergence`. The `burn_in` default is 0, so that the trace plot starts from the first iteration. Additional iterations can be specified if the MCMC chain does not converge in 2000 iterations.

```
> trace.plot <- assign.convergence(test=mcmc.chain, burn_in=0, iter=2000,
+                                  parameter="B", whichGene=1,
+                                  whichSample=NA, whichPath=NA)
```

The `assign.convergence` function returns a vector of the estimated values from each Gibbs sampling iteration of the model parameter to be checked and a trace plot of the parameter.

2.3.4 `assign.summary`

We then apply the `assign.summary` function to compute the posterior mean of each parameter. Typically we use the second half of the MCMC chain to compute the posterior mean. We specify the default burn-in period to be the first 1000 iteration and the default total iteration to be 2000. The 1000 burn-in iterations are discarded when we compute the posterior mean. The `adaptive_B`, `adaptive_S` and `mixture_beta` options have to set the same as those in the `assign.mcmc` function.

```
> mcmc.pos.mean <- assign.summary(test=mcmc.chain, burn_in=1000,
+                                 iter=2000, adaptive_B=TRUE,
+                                 adaptive_S=FALSE, mixture_beta=TRUE)
```

The `assign.summary` function returns the posterior mean of each parameter. The details of the `assign.summary` output are described in the "value" section of the manual page of `assign.summary` function.

2.3.5 assign.cv.output

The assign.cv.output generates the cross-validation results in the training samples. Output files from assign.cv.output are:

1. **pathway_activity_trainingset.csv**: *ASSIGN* predicted pathway activity in training samples.
2. **signature_heatmap_trainingset.pdf**: heatmaps of the expression level of signature genes in training samples.
3. **pathway_activity_scatterplot_trainingset.pdf**: scatterplot of pathway activity in training samples.

```
> # For cross-validation, Y in the assign.mcmc function
> # should be specified as processed.data$trainingData_sub.
> assign.cv.output(processed.data=processed.data,
+                  mcmc.pos.mean.trainingData=mcmc.pos.mean,
+                  trainingData=trainingData1,
+                  trainingLabel=trainingLabel1, adaptive_B=FALSE,
+                  adaptive_S=FALSE, mixture_beta=TRUE,
+                  outputDir=tempdir)
```

2.3.6 assign.output

The assign.output generates the prediction results in the test samples. Output files from assign.output are:

1. **pathway_activity_testset.csv**: *ASSIGN* predicted pathway activity in test samples.
2. **signature_heatmap_testset_prior.pdf**: heatmaps of the expression level of prior signature genes in training samples.
3. **signature_heatmap_testset_posterior.pdf**: heatmaps of the expression level of posterior signature genes in training samples. This plot is only generated when Adaptive_S is specified TRUE.
4. **pathway_activity_scatterplot_testset.pdf**: scatterplot of pathway activity in test samples. The x-axis represents test samples ordered by pathway activity; the y-axis represents pathway activity.
5. **pathway_activity_boxplot_testset.pdf**: boxplot of pathway activity in every test class. This plot is only generated only the testLabel argument is not NULL.

The user needs to specify the output directory in the outputDir option, when running assign.cv.output and assign.output.

```
> assign.output(processed.data=processed.data,
+               mcmc.pos.mean.testData=mcmc.pos.mean,
+               trainingData=trainingData1, testData=testData1,
+               trainingLabel=trainingLabel1,
+               testLabel=testLabel1, geneList=NULL,
+               adaptive_B=TRUE, adaptive_S=FALSE,
+               mixture_beta=TRUE, outputDir=tempdir)
```

3 Additional Features

3.1 Anchor Gene Lists and Exclude Gene Lists

The *ASSIGN* package allows a signature to be adapted to fit other biological contexts, reducing the contribution of specific genes in the signature to better match heterogeneity observed in the test dataset. Occasionally, adapting a signature may reduce the importance of key signature genes. For example, if a signature is created by overexpressing an oncogenic gene in a cell line, but during the adaptation step, *ASSIGN* reduces the importance of that key gene, the quality of the *ASSIGN* predictions may be reduced. Alternatively, if a gene in the signature is associated with some other heterogeneity in the data, such as smoking status, *ASSIGN* may adapt to differences in that gene, rather than the actual desired signature activity predictions. To this end, we have added the ability to provide a list of key genes to anchor in the signature, and genes to exclude from the signature. *ASSIGN* accomplishes this by setting the probability of inclusion into the signature to one for anchor genes, and zero for exclude genes. The change in expression values can still adapt, increasing or reducing the fold change associated with each gene in the signature, but the anchor genes will always contribute to the final signature, and the exclude genes will not.

```
> dir.create(file.path(tempdir, "anchor_exclude_example"))
> anchorList = list(bcat="224321_at",
+                   e2f3="202589_at",
+                   myc="221891_x_at",
+                   ras="201820_at",
+                   src="224567_x_at")
> excludeList = list(bcat="1555340_x_at",
+                    e2f3="1555340_x_at",
+                    myc="1555340_x_at",
+                    ras="204748_at",
+                    src="1555339_at")
> assign.wrapper(trainingData=trainingData1, testData=testData1,
+                trainingLabel=trainingLabel1, testLabel=NULL,
+                geneList=geneList1, n_sigGene=NULL, adaptive_B=TRUE,
+                adaptive_S=TRUE, mixture_beta=TRUE,
+                outputDir=file.path(tempdir, "anchor_exclude_example"),
+                anchorGenes=anchorList, excludeGenes=excludeList,
+                iter=2000, burn_in=1000)
```

3.2 Fraction of Upregulated Genes

By default, *ASSIGN* Bayesian gene selection chooses the signature genes with an equal fraction of genes that increase with pathway activity and genes that decrease with pathway activity. Use the pctUp parameter to modify this fraction. Set pctUP to NULL to select the most significant genes, regardless of direction.

4 GFRN Optimization Procedure

When running *ASSIGN*, the number of genes in the gene list can affect the predictions that *ASSIGN* produces, but it is not always clear how long the gene list should be. Included within *ASSIGN* is the optimization procedure used in the publication “Activity of distinct growth factor receptor network components in breast tumors uncovers two biologically relevant subtypes.” The function allows you to optimize the gene list lengths for the pathways included in the paper using your own correlation data and gene list lengths. This function runs *ASSIGN* pathway prediction on various gene list lengths to find the optimum gene list length for the GFRN pathways by correlating the *ASSIGN* predictions to a matrix of correlation data that you provide. This function takes a long time to run because you are running *ASSIGN* many times on many pathways, so I recommend parallelizing by pathway or running the *ASSIGN* predictions first (long and parallelizable) and then running the correlation step (quick) separately.

4.1 Example Optimization Procedure

The following example optimizes the pathway length for the AKT pathway based on correlating *ASSIGN* predictions with proteomics data. First, read in the test data that you want to predict using *ASSIGN* and the data (e.g. proteomics data) that will be used for correlation:

```
> dir.create(file.path(tempdir, "optimization_example"))
> setwd(file.path(tempdir, "optimization_example"))
> testData <- read.table("https://dl.dropboxusercontent.com/u/62447/ASSIGN/icbp_Rsubread_
+                         row.names=1, header=1)
> corData <- read.table("https://dl.dropboxusercontent.com/u/62447/ASSIGN/proteomics.txt'
```

Next, create a list of data used for correlation. The list should contain a vector of column names from the correlation data for each of the pathways that are being optimized. The gene list length that has the largest average correlation for the columns in the correlation list will be the optimized gene list.

```
> #this is a list of pathways and columns in the correlation data that will
> #be used for correlation
> corList <- list(akt=c("Akt","PDK1","PDK1p241"))
```

Finally, run the ComBat batch correction procedure and run the `optimizeGFRN` function:

```
> #run the batch correction procedure between the test and training data
> combat.data <- ComBat.step2(testData, pcaPlots = TRUE)
> #run the default optimization procedure
> optimization_results <- optimizeGFRN(combat.data, corData, corList, run="akt")
```

ASSIGN will output the results for each gene list length in the current working directory. The `optimizeGFRN` function returns a list of optimized gene lists which can be used on other datasets and correlation results. Additional options and documentation is available in the `optimizeGFRN` function documentation.

5 Citing ASSIGN

If you use *ASSIGN* in your publication, please cite:

- Shen, Y. et al. ASSIGN: context-specific genomic profiling of multiple heterogeneous biological pathways. Bioinformatics 31 (11), 1745-1753 (2015).

6 Conclusion

Please see the *ASSIGN* reference manual for full descriptions of functions and the various options they support.

7 Session Info

```
R version 3.4.0 (2017-04-21)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.2 LTS

Matrix products: default
BLAS: /home/biocbuild/bbs-3.5-bioc/R/lib/libRblas.so
LAPACK: /home/biocbuild/bbs-3.5-bioc/R/lib/libRlapack.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=C               LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] ASSIGN_1.12.0

loaded via a namespace (and not attached):
 [1] compiler_3.4.0  backports_1.0.5 magrittr_1.5    BiocStyle_2.4.0 rprojroot_1.2
 [6] htmltools_0.3.5 tools_3.4.0     yaml_2.1.14     Rcpp_0.12.10    stringi_1.1.5
[11] rmarkdown_1.4   knitr_1.15.1    stringr_1.2.0   digest_0.6.12   evaluate_0.10
```