Introduction to the ASSIGN Package

Ying Shen and W. Evan Johnson

October 17, 2016

Contents

1 Introduction

This vignette provides an overview of the Bioconductor package ASSIGN for signature-based profiling of heterogeneous biological pathways. ASSIGN (Adaptive Signature Selection and InteGratioN) is a computational tool 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 knowledge-based literatures or from perturbation experiments to the 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 encompasses the pathway deregulation/activation signature.

Some distinguishable features of ASSIGN are described as follows: 1). multiple pathways profiling: ASSIGN profiles pathway signatures simultaneously, accounting for 'cross-talks' between interconnected pathway components. 2). Context specificity in baseline gene expression: Baseline gene expression levels (i.e., the gene expression level under normal status) may vary widely due to the differences in tissue types, disease status, or across different 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 the input gene list or the magnitudes of signature genes as prior information, allowing for the adaptive refinement of pathway signatures in specific cell or tissue samples. 4). Regularization of signature strength estimates: ASSIGN regularizes the signature strength coefficients using a Bayesian ridge regression formulation by shrinking strength of the irrelevant signature genes toward zero. The parameter regularization constrains the pathway signature to a small group of genes, thus, making the results more biologically interpretable.

2 How to use the ASSIGN package

2.1 Run ASSIGN in a step-by-step way

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. For quick use, we also developed an all-in-one `assign.wrapper` to integrated single step functions. The details of `assign.wrapper` is discribed in the next section.

In the following example, we will illustrate how to run the functions in the ASSIGN package in a step-by-step way. The input and output of each function are described here.

We first load ASSIGN, and create a temporary directory "tempdir" under the user's current directory. All output generated in this vignette will be saved in the "tempdir".

```
> ## create a temporary directory
> library(ASSIGN)
> dir.create("tempdir")
> tempdir <- "tempdir"
```

Next thing is to load the training and test datasets and the labels of training and test datasets. The training dataset is a G (Number of genomic measures) 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 index of control and experimental samples of each purturbation experiment. For example, we specify the column index of 10 RAS control samples to be 1:10, and column index of 10 RAS activated samples to be 39:48. The test dataset is a G (Number of genomic measures) x N (number of patient samples) matrix. The test data labels denote the classes of the N test samples. In our example, test sample 1-53 are adenocarcinoma and 54-111 are squamous cell carcinoma. We specify "Adeno" and "Squamous" in the vector of test data labels. Notice that the test data labels are optional. ASSIGN outputs additional validation plots to evaluate classification accuracy when the 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))
```

We first run `assign.preprocess` function on the input datasets. When the genomic measures (i.e., gene expression profiles) of training samples are provided, but predetermined pathway signature gene lists are not provided, `assign.preprocess` function utilizes a Bayesian univariate regression module to select a gene set (usually 50-200 but 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 differently expressed, need to be included for the prediction purpose. In this case we specify `trainingData` and `geneList` when BOTH of 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 differential expressed gene lists of each pathways (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.

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 parameters. The details of `assign.mcmc` output are described in the Value section of the manual page of `assign.mcmc` function. We can make a trace plot to check the convergence of the model parameters through `assign.convergence`. The `burn_in` is set default 0, so that the trace plot starts from the first iteration. The additional iteration can be specified if the MCMC chain is not converged 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 the 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.

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 parameters. The details of `assign.summary` output are described in the Value section of the manual page of `assign.summary` function.

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 acitivity in training samples.

2). signature_heatmap_trainingset.pdf: heatmaps of expression level of signature genes in training samples.

3). pathway_activity_scatterplot_trainingset.pdf: scatterplot of pathway activity in training samples.

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

1). pathway_activity_testset.csv: ASSIGN predicted pathway acitivity in test samples.

2). signature_heatmap_testset_prior.pdf: heatmaps of expression level of prior signature genes in training samples.

3). signature_heatmap_testset_posterior.pdf: heatmaps of 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 `assigm.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)

> # 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)
```

2.2 Run ASSIGN in an all-in-one way

We developed an all-in-one `assign.wrapper` function to run ASSIGN in a simple and fast way. For the purpose of fast run and basic results, the user will ONLY need to run this `assign.wrapper` function. The `assign.wrapper` function output the following files:

1). pathway_activity_testset.csv: ASSIGN predicted pathway acitivity in test samples.

2). signature_heatmap_testset_prior.pdf: heatmaps of expression level of prior signature genes in training samples.

3). signature_heatmap_testset_posterior.pdf: heatmaps of 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.

6). output.rda: The intermediate results of individual ASSIGN functions.

Here we illustrate how to run `assign.wrapper` function by three examples. To start, first follow the code chuncks 1 and 2 above to create a temporary dirctory "tempdir" and load training and test datasets. Parameter settings have been described in details in the previous section.

```
> # Example 1: training dataset is available;
> # the gene list of pathway signature is NOT available
> 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= tempdir,
+                iter=2000, burn_in=1000)
```

```
> # Example 2: training dataset is available;
> # the gene list of pathway signature is available
> 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=tempdir, iter=2000, burn_in=1000)
```

```
> #Example 3: training dataset is NOT available;
> #the gene list of pathway signature is available
> 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= tempdir, iter=2000, burn_in=1000)
```

3 Conclusion

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

4 Session Info

```
R version 3.3.1 (2016-06-21)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

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

other attached packages:
[1] ASSIGN_1.10.0 gplots_3.0.1  msm_1.6.4     Rlab_2.15.1

loaded via a namespace (and not attached):
 [1] Matrix_1.2-7.1     tools_3.3.1        expm_0.999-0       survival_2.39-5
 [5] KernSmooth_2.23-15 mvtnorm_1.0-5      splines_3.3.1      gdata_2.17.0
 [9] grid_3.3.1         caTools_1.17.1     bitops_1.0-6       gtools_3.5.0
[13] lattice_0.20-34
```