Introduction to the ASSIGN Package

Ying Shen and W. Evan Johnson

April 12, 2014

Contents

1 Introduction **1**

2 How to use the ASSIGN package **2**

2.1 Run ASSIGN in an all-in-one way 2

2.2 Run ASSIGN in a step-by-step way 4

3 Conclusion **6**

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.

As input, ASSIGN requires a gene expression dataset from the samples to be profiled (test dataset), and a pathway profiling dataset from perturbation experiments (training dataset), or predetermined signature gene lists based on public databases (usually 50 – 200 genes). Besides the training and test datasets, ASSIGN requires training data labels specifying the control and experimental groups each training samples associated with to generate differentially expressed genes as pathway signatures. The user can specify adaptive background, adaptive signature and signature strength regularization options based on the analysis context. ASSIGN outputs a matrix of signature coefficients (strength of each signature for each sample) and the prior/posterior signature gene lists and magnitude changes. In addition, ASSIGN also provides the user with an internal cross-validation on the perturbation data, MCMC posterior convergence diagnostics, and an evaluation of classification accuracy if clinical labels are available on the profiling dataset.

2 How to use the ASSIGN package

2.1 Run ASSIGN in an all-in-one way

We created 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 returns the pathway signature strength, validation plots, and signature heatmaps as output. The intermediate results are stored in the output.rda file.

To run `assign.wrapper`, we first create a temporary directory "tempdir" under the user's current directory. The output generated in this vignette will be saved in the "tempdir".

Let's first load the training dataset, test sets and the training and test data labels.

Notice that the test data labels are optional. ASSIGN outputs the 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("subtypeA","subtypeB"),c(53,58))
```

Here we illustrate how to run `assign.wrapper` function by three examples. For details of parameter settings, see next 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=20, burn_in=10)
```

```
> # 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=20, burn_in=10)
```

```
> #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=20, burn_in=10)
```

2.2 Run ASSIGN in a step-by-step way

Although `assign.wrapper` function generates basic results that may be sufficient for most users, we created a series of functions: `assign.preprocess`, `assign.mcmc`, `assign.convergence`, `assign.summary`, `assign.cv.output`, and `assign.output` that work together to produce more detailed results for advanced users.

In the following example, we will illustrate how to run these functions in the ASSIGN package in a step-by-step way.

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 and test dataset as well as the prior parameters for the background vector (B_vector), signature matrix (S_matrix) and the probability signature matrix (Pi_matrix). These parameters are the input of the `assing.mcmc` function.

For the `assign.mcmc` 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, rather than directly using the output from the `assign.preprocess` function.

```
> 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 = 20, adaptive_B=TRUE,
+                           adaptive_S=FALSE, mixture_beta=TRUE)
```

The `assign.mcmc` function returns the MCMC chain recording default 2000 iterations for each parameters. We can make a trace plot to check the convergence of the model parameters. 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=20,
+                                  parameter="B", whichGene=1,
+                                  whichSample=NA, whichPath=NA)
```

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. Those 1000 burn-in iterations are discarded when we compute the posterior mean. The `adaptive_B`, `adaptive_S` and `mixture_beta` options should be set the same as those in the `assign.mcmc` function.

```
> mcmc.pos.mean <- assign.summary(test=mcmc.chain, burn_in=10,
+                                 iter=20, adaptive_B=TRUE,
+                                 adaptive_S=FALSE,mixture_beta=TRUE)
```

The `assign.cv.output` and `assigm.output` function outputs the cross-validation results in the training samples and the prediction results in the test samples, respectively. The user needs to specify the output directory in the `outputDir` option.

```
> 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)
```

3 Conclusion

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