scDD: Bayesian Nonparametric Mixture Modeling of Single-Cell RNA-seq Experiments

Keegan Korthauer*

August 25, 2016

Contents

1 Introduction

The *scDD* package models single-cell gene expression data (from single-cell RNA-seq) using flexible nonparamentric Bayesian mixture models in order to explicitly handle heterogeneity within cell populations. In bulk RNA-seq data, where each measurement is an average over thousands of cells, distributions of expression over samples are most often unimodal. In single-cell RNA-seq data, however, even when cells represent genetically homogeneous populations, multimodal distributions of gene expression values over samples are common [1]. This type of heterogeneity is often treated as a nuisance factor in studies of differential expression in single-cell RNA-seq experiments. Here, we explicitly accommodate it in order to improve power to detect differences in expression distributions that are more complicated than a mean shift.

2 Background

Our aim is two-fold: (1) to detect which genes have different expression distributions across two biological conditions and (2) to classify those differences into informative patterns. Note that in (1) we explicitly say differences in 'distributions' rather than differences in 'average', which would correspond to traditional DE (differential expression) analysis in bulk

*keegan@jimmy.harvard.edu

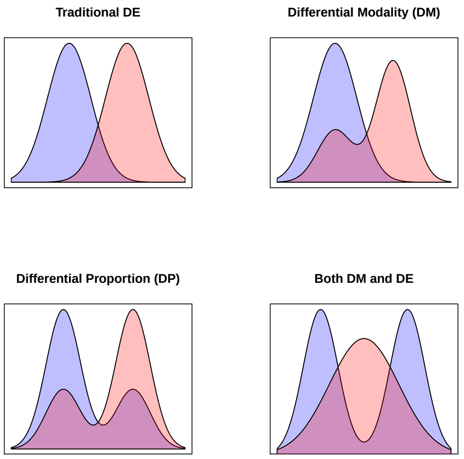

Figure 1: **Illustration of informative DD patterns**

RNA-seq. By examining the entire distribution, we are able to detect more subtle differences as well as describe complex patterns, such as the existence of subgroups of cells within and across condition that express a given gene at a different level.

We start by assuming that the log-transformed nonzero expression values arise out of a Dirichlet Process Mixture of normals model. This allows us to characterize expression distributions in terms of the number of modes (or clusters). To detect differences in these distributions across conditions, an approximate Bayes Factor score is used which compares the conditional likelihood under the hypothesis of Equivalent distributions (ED) where one clustering process governs both conditions jointly, with the hypothesis of Differential distributions (DD) where each condition is generated from its own clustering process. Significance of the scores for each gene are evaluated via empirical p-values after permutation. Zero values are considered by also implementing a χ^2 test of whether the proportion of zero values differs by condition (adjusted for overall sample detection rate). More details are provided in [1].

After the detection step is carried out, the significantly DD genes are classified into four informative patterns based on the number of clusters detected and whether they overlap. These patterns, depicted in Figure 1, include DE (differential expression of unimodal genes), DP (differential proportion for multimodal genes), DM (differential modality), and DB (both differential modality and mean expression levels). Genes where a differential proportion of zeroes were identified are classified as DZ (differential zero).

The rest of this vignette outlines the main functionality of the *scDD* package. This includes:

- Identifying genes that are expressed differently between two biological conditions and classifying them into informative patterns.
- Simulating single-cell RNA-seq data with differential expression that exhibits multimodal patterns.
- Preprocessing and formatting of single-cell RNA-seq data to facilitate analysis
- Visualizing the expression patterns using a violin plotting scheme

3 Identify and Classify DD genes

In this section, we demonstrate how to use the main function `scDD` to find genes with differential distributions and classify them into the patterns of interest described in the previous section.

First, we need to load the *scDD* package. For each of the following sections in this vignette, we assume this step has been carried out.

```
> library(scDD)
```

Next, we load the toy simulated example *ExpressionSet* object that we will use for identifying and classifying DD genes.

```
> data(scDatExSim)
```

Verify that this object is a member of the *ExpressionSet* class and that it contains 200 samples and 30 genes

```
> class(scDatExSim)
```

```
[1] "ExpressionSet"
attr(,"package")
[1] "Biobase"
```

```
> show(scDatExSim)
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 30 features, 200 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: Sample1 Sample2 ... Sample200 (200 total)
  varLabels: condition
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:
NULL
```

Specify the arguments that we'll pass to the `scDD` function. We will perform 100 permutations on each of the 30 genes.

```
> prior_param=list(alpha=0.01, mu0=0, s0=0.01, a0=0.01, b0=0.01)
> nperms <- 100
```

Finally, call the `scDD` function to perform permutations, classify DD genes, and return the results. We won't perform the test for a difference in the proportion of zeroes since none exists in this simulated toy example data. We also won't perform adjusted permutations to offset the effects of detection rate on mean expression level, since these are also absent in our toy simulated example. Note that this step will take significantly longer with more genes and/or more permutations, but multiple cores will automatically be utilized (if available) via the *BiocParallel* package. In practice, it is recommended that at least 1000 permutations are carried out.

```
> RES <- scDD(scDatExSim, prior_param=prior_param, permutations=nperms, testZeroes=FALSE,
+              adjust.perms=FALSE)
```

The `RES` object is a list with four items: the first is a data frame with nine columns:

1. `gene`: gene name (matches rownames of SCdat)
2. `nonzero.pvalue`: permutation p-value for testing of independence of condition membership with clustering
3. `nonzero.pvalue.adj`: Benjamini-Hochberg adjusted permutation p-value for testing of independence of condition membership with clustering
4. `zero.pvalue`: p-value for test of difference in dropout rate (only for non-DD genes and if `testZeroes==TRUE`)
5. `zero.pvalue.adj`: Benjamini-Hochberg adjusted p-value for test of difference in dropout rate (only for non-DD genes and if `testZeroes==TRUE`)
6. `DDcategory`: name of the DD pattern (DE, DP, DM, DB, DZ), or NC (no call), or NS (not significant).
7. `Clusters.combined`: the number of clusters identified when pooling condition 1 and 2 together
8. `Clusters.c1`: the number of clusters identified in condition 1 alone
9. `Clusters.c2`: the number of clusters identified in condition 2 alone

The remaining three elements are data frames (first for condition 1 and 2 combined, then condition 1 alone, then condition 2 alone) that contains the cluster memberships for each sample (cluster 1,2,3,...) in columns and genes in rows. Zeroes, which are not involved in the clustering, are labeled as zero.

4 Alternate test for Differential Distributions

The first step in the scDD framework that identifies Differential Distributions was designed to have optimal power to detect differences in expression distributions, but the utilization of a permutation test on the Bayes Factor can be computationally demanding. While this is not an issue when machines with multiple cores are available since the code takes advantage of parallel processing, we also provide an option to use an alternate test to detect distributional differences that avoides the use of a permutation test. This optional test is the Kolmogorov-Smirnov test, which examines the null hypothesis that two samples are generated from the same continuous distribution. While the use of this test yielded slighlty lower power in simulations than the full permutation testing framework at lower sample sizes (primarily affecting the DB pattern genes), it does not require permutations and thus is orders of magnitude faster. The overall power to detect DD genes in simulation was still comparable or favorable to exisiting methods for differential expression analysis of scRNA-seq experiments.

The remaining steps of the scDD framework remain unchanged if the alternate test is used. That is, the Dirichlet process mixture model is still fit to the observed expression measurements so that the significant DD genes can be categorized into patterns that represent the major distributional changes, and results can still be visualized with violin plots using the `sideViolin` function described in the Plotting section.

The option to use the Kolmogorov-Smirnov test instead of the full permutation testing procedure is invoked by setting the number of permutations to zero (the `permutations` argument) when calling the main `scDD` function, as follows.

```
> RES_KS <- scDD(scDatExSim, prior_param=prior_param, permutations=0, testZeroes=FALSE,
+               adjust.perms=FALSE)
```

The results object returned by `scDD` remains exactly as described in the previous section, with the exception that the `nonzero.pvalue` and `nonzero.pvalue.adj` columns of the `Genes` data frame now contain the p-values and Benjamini-Hochberg adjusted p-values of the Kolmogorov-Smirnov test rather than the permutation test.

5 Simulation

Here we show how to generate a simulated single-cell RNA-seq dataset which contains multi-modal genes. The `simulateSet` function simulates data from a two-condition experiment with a specified number of genes that fall into each of the patterns of interest. For DD genes, these include DE (differential expression of unimodal genes), DP (differential proportion for multimodal genes), DM (differential modality), and DB (both differential modality and mean expression levels), and for ED genes these include EE (equivalent expression for unimodal genes) and EP (equivalent proportion for multimodal genes). The simulation parameters are based on observed data from two conditions, so the function requires an *ExpressionSet* formatted dataset as input.

First, we load the toy example *ExpressionSet* to simulate from

```
> data(scDatEx)
```

We'll verify that this object is a member of the *ExpressionSet* class and that it contains 142 samples and 500 genes

```
> class(scDatEx)
```

```
[1] "ExpressionSet"
attr(,"package")
[1] "Biobase"
```

```
> show(scDatEx)
```

```
ExpressionSet (storageMode: lockedEnvironment)
assayData: 500 features, 142 samples
  element names: exprs
protocolData: none
phenoData
```

```
  sampleNames: C1.073 C1.074 ... C2.070 (142 total)
  varLabels: condition
  varMetadata: labelDescription
featureData: none
experimentData: use 'experimentData(object)'
Annotation:
NULL
```

Next we need to set the arguments that will be passed to the simulateSet function. In this example we will simulate 30 genes total, with 5 genes of each type and 100 samples in each of two conditions. We also set a random seed for reproducibility.

```
> nDE <- 5
> nDP <- 5
> nDM <- 5
> nDB <- 5
> nEE <- 5
> nEP <- 5
> numSamples <- 100
> seed <- 816
```

Finally, we'll create the simulated set with specified numbers of DE, DP, DM, DM, EE, and EP genes and specified number of samples, where DE gene fold changes represent 2 standard deviations of the observed fold change distribution, and multimodal genes have cluster mean distance of 4 standard deviations.

```
> SD <- simulateSet(scDatEx, numSamples=numSamples, nDE=nDE, nDP=nDP, nDM=nDM, nDB=nDB,
+                    nEE=nEE, nEP=nEP, sd.range=c(2,2), modeFC=4, plots=FALSE,
+                    random.seed=seed)
```

The matrix in the first list element of the SD object contains simulated expression values. The second element stores the fold change/modal distance values which can be useful in assessing performance of a differential expression method.

6 Formatting and Preprocessing

6.1 Constructing an ExpressionSet object

In this subsection, we provide a quick example of how to construct an object of the *ExpressionSet* class. For more detailed instructions, refer to the *Biobase* package documentation.

Here we will convert the simulated data object SD returned by simulateSet (from the previous section) into an *ExpressionSet* object. First, load the *Biobase* package:

```
> library(Biobase)
```

Next, create a vector of condition membership labels (these should be 1 or 2). In our simulated dataset, we generated numSamples samples in each of two conditions.

```
> condition <- c(rep(1, numSamples), rep(2, numSamples))
```

The rows and columns of the expression matrix should have unique names, and the names of the columns should to correspond to the names of the condition membership labels in condition.

```
> rownames(SD[[1]]) <- paste0(rownames(SD[[1]]), 1:nrow(SD[[1]]), sep="")
> colnames(SD[[1]]) <- names(condition) <- paste0("Sample", 1:ncol(SD[[1]]), sep="")
```

Once our labeling is intact, we can call the ExpressionSet function and specify the two relevant pieces of information. Optionally, additional experiment information can be stored in additional slots; see *Biobase* package.

```
> SDExpressionSet <- ExpressionSet(assayData=SD[[1]],
+                       phenoData=as(data.frame(condition), "AnnotatedDataFrame"))
```

6.2 Filtering and Normalization

In this subsection, we demonstrate the utility of the preprocess function, which can be helpful if working with raw data, or data which contains genes that are predominantly zero (common in single-cell RNA-seq experiments). This function takes as input a list of data matrices, one for each condition.

First, load the toy example data list:

```
> data(scDatExList)
```

Verify that the data is formatted as a list of 2 matrices (one for each of 2 conditions), that each matrix has 100 rows (one for each gene), and that the number of columns in each matrix corresponds to the number of samples in each condition (78 and 64, respectively):

```
> str(scDatExList)
```

```
List of 2
 $ C1: num [1:100, 1:78] 0 53.3 0 2 0 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:100] "MKL2" "CD109" "ABTB1" "MAST2" ...
  .. ..$ : chr [1:78] "C1.073" "C1.074" "C1.075" "C1.076" ...
 $ C2: num [1:100, 1:64] 6 17 0 0 0 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:100] "MKL2" "CD109" "ABTB1" "MAST2" ...
  .. ..$ : chr [1:64] "C2.001" "C2.002" "C2.003" "C2.004" ...
NULL
```

Obtain the names of the conditions to pass to the preprocess function:

```
> condition.names <- names(scDatExList)
```

Finally, apply the preprocess function to reformat the data into one data matrix with 100 rows and $78 + 64 = 142$ columns. In this example, we set the zero.thresh argument to 1 so that genes are filtered out if they are all zero, and we set the median_norm argument to FALSE to return raw data (no normalization is carried out).

```
> scDatExMat <- preprocess(scDatExList, ConditionNames=condition.names,
+                          zero.thresh=1, median_norm=FALSE)
```

Now, apply the preprocess function again, but this time use a more stringent threshold on the proportion of zeroes and apply median normalization. In this example, we set the zero.thresh argument to 0.75 so that genes with more than 75 percent zeroes are filtered out and we set the median_norm argument to TRUE to return median normalized counts.

```
> scDatExMatNormThresh <- preprocess(scDatExList, ConditionNames=condition.names,
+                                    zero.thresh=0.75, median_norm=TRUE)
```

7 Plotting

Next we demonstrate the plotting routine that is implemented in the sideViolin function. This function produces side-by-side violin plots (where the curves represent a smoothed kernel density estimate) of the log-transformed data. A count of 1 is added before log-transformation so that zeroes can be displayed, but they are not included in the density estimation. Each condition is represented by one violin plot. Individual data points are plotted (with jitter) on top.

We illustrate this function by displaying the six types of simulated genes using the toy example simulated dataset. First, load the toy simulated dataset:

```
> data(scDatExSim)
```

Next, load the Biobase package to facilitate subset operations on ExpressionSet class objects:

```
> library(Biobase)
```

The following lines will produce the figures in Figure 2.

Plot side by side violin plots for Gene 1 (DE):

```
> sideViolin(exprs(scDatExSim)[1,], scDatExSim$condition,
+            title.gene=featureNames(scDatExSim)[1])
```

Plot side by side violin plots for Gene 6 (DP):

```
> sideViolin(exprs(scDatExSim)[6,], scDatExSim$condition,
+            title.gene=featureNames(scDatExSim)[6])
```

Plot side by side violin plots for Gene 11 (DM):

```
> sideViolin(exprs(scDatExSim)[11,], scDatExSim$condition,
+            title.gene=featureNames(scDatExSim)[11])
```

Plot side by side violin plots for Gene 16 (DB):

```
> sideViolin(exprs(scDatExSim)[16,], scDatExSim$condition,
+            title.gene=featureNames(scDatExSim)[16])
```

Plot side by side violin plots for Gene 21 (EP):

```
> sideViolin(exprs(scDatExSim)[21,], scDatExSim$condition,
+            title.gene=featureNames(scDatExSim)[21])
```

Plot side by side violin plots for Gene 26 (EE):

```
> sideViolin(exprs(scDatExSim)[26,], scDatExSim$condition,
+            title.gene=featureNames(scDatExSim)[26])
```

The plot objects returned by `sideViolin` are standard *ggplot2* objects, and thus can be manipulated into multipanel figures with the help of the *gridExtra* or *cowplot* packages.

8 Session Info

Here is the output of `sessionInfo` on the system where this document was compiled:

```
> toLatex(sessionInfo())
```

- R version 3.2.2 (2015-08-14), `x86_64-apple-darwin13.4.0`
- Locale: `en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, graphics, grDevices, methods, parallel, stats, utils
- Other packages: Biobase 2.30.0, BiocGenerics 0.16.1, gridExtra 2.2.1, scDD 1.1.0
- Loaded via a namespace (and not attached): abind 1.4-3, arm 1.8-6, BiocParallel 1.4.3, BiocStyle 1.8.0, bitops 1.0-6, blockmodeling 0.1.8, caTools 1.17.1, coda 0.18-1, colorspace 1.2-6, crayon 1.3.1, digest 0.6.9, EBSeq 1.10.0, fields 8.3-6, futile.logger 1.4.1, futile.options 1.0.0, gdata 2.17.0, ggplot2 2.1.0, gplots 2.17.0, grid 3.2.2, gtable 0.2.0, gtools 3.5.0, KernSmooth 2.23-15, labeling 0.3, lambda.r 1.1.7, lattice 0.20-33, lme4 1.1-11, maps 3.1.0, MASS 7.3-45, Matrix 1.2-4, mclust 5.1, memoise 1.0.0, minqa 1.2.4, munsell 0.4.3, nlme 3.1-126, nloptr 1.0.4, outliers 0.14, plyr 1.8.3, Rcpp 0.12.4, scales 0.4.0, spam 1.3-0, splines 3.2.2, testthat 0.11.0, tools 3.2.2

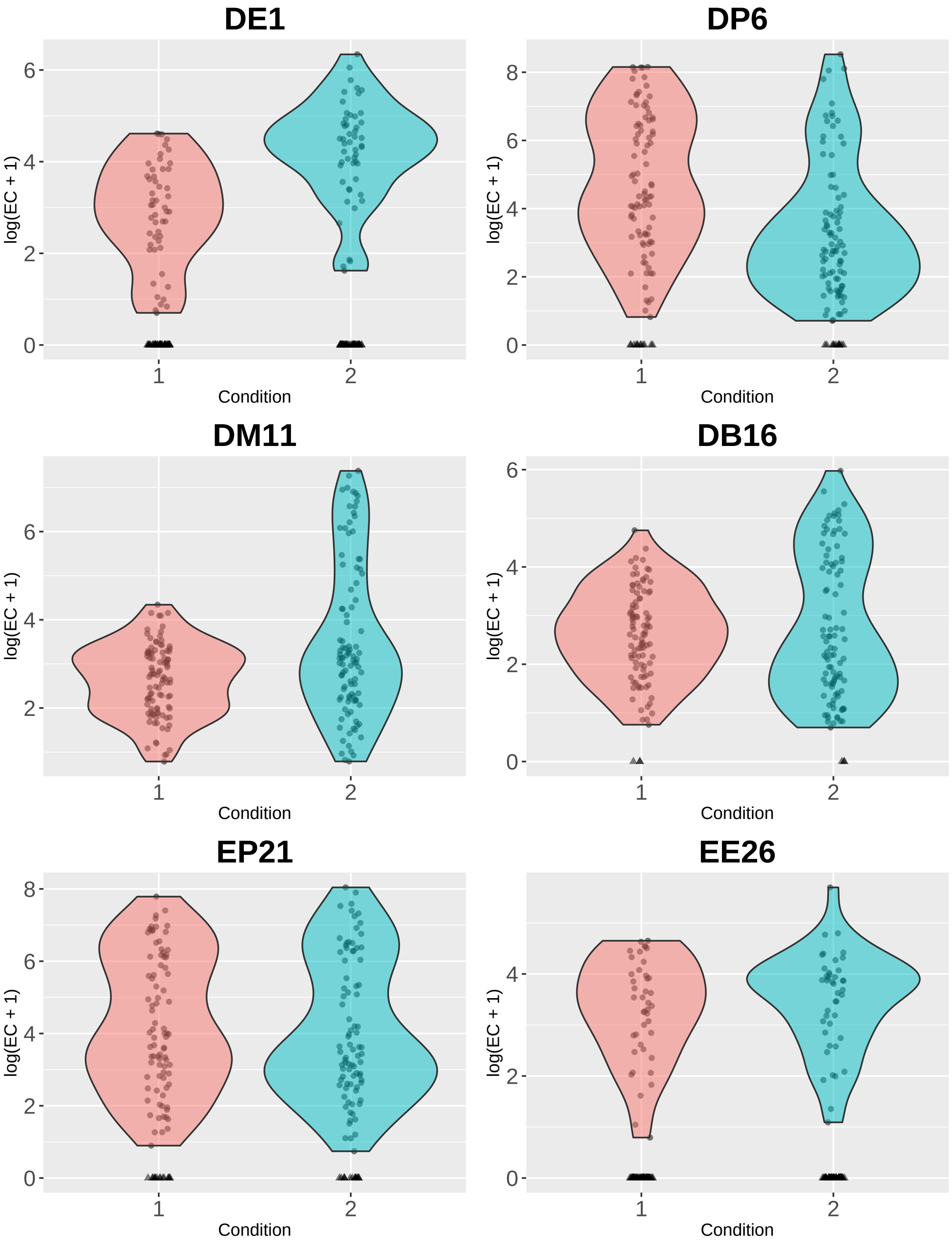

Figure 2: **Example Simulated DD genes**

References

[1] Keegan D Korthauer, Li-Fang Chu, Michael A. Newton, Yuan Li, James Thomson, Ron Stewart, and Christina Kendziorski. scDD: A statistical approach for identifying differential distributions in single-cell rna-seq experiments. *BioRxiv*, 2015.