BEARscc: Using spike-ins to assess single cell cluster robustness

David T. Severson

2017-11-28

Contents

1 Summary

Single-cell transcriptome sequencing data are subject to substantial technical variation and batch effects that can confound the classification of cellular sub-types. Unfortunately, current clustering algorithms don't account for this uncertainty. To address this shortcoming, we have developed a noise perturbation algorithm called **BEARscc** that is designed to determine the extent to which classifications by existing clustering algorithms are robust to observed technical variation.

BEARscc makes use of ERCC spike-in measurements to model technical variance as a function of gene expression and technical dropout effects on lowly expressed genes. In our benchmarks, we found that BEARscc accurately models read count fluctuations and drop-out effects across transcripts with diverse expression levels. Applying our approach to publicly available single-cell transcriptome data of mouse brain and intestine, we have demonstrated that BEARscc identified cells that cluster consistently, irrespective of technical variation. For more details, see the manuscript that is now available on bioRxiv.

2 Installation

Installing BEARscc is easy. You can download a source package here. You can then use `install.packages`, but give it the location of the downloaded file:

```r
install.packages('inst/BEARscc_0.99.0.tar.gz', repos = NULL, type="source")
#> Warning in install.packages("inst/BEARscc_0.99.0.tar.gz", repos = NULL, :
#> installation of package 'inst/BEARscc_0.99.0.tar.gz' had non-zero exit
#> status
```

3 Usage

Here we provide a limited illustrative example of BEARscc on example data. A comprehensive vignette is being drafted and will be available in the near future. We are using the excellent data.table library here, which can be easily installed by typing install.packages('data.table').

In R, load a single cell data table and ERCC known concentrations:

```
library('data.table')
data.counts.dt <- fread("example/brain_control_example.tsv")
ERCC.meta.dt <- fread("example/ERCC.meta.tsv")
```

Separate ERCC observations into a data.frame and transfrom counts and ERCC known concentration data.table into data.frame.

```
ERCC.counts.df <- data.frame(data.counts.dt[GENE_ID%like%"ERCC-",], row.names="GENE_ID")
data.counts.df <- data.frame(data.counts.dt, row.names = "GENE_ID")
ERCC.meta.df <- data.frame(ERCC.meta.dt, row.names="ERCC_ID")
```

Estimate noise inputting ERCC known concentrations, and both endogenous and spike-in counts matrices into the estimate_noiseparameters() function.

```
results <- estimate_noiseparameters(ERCC.counts.df,
                                    data.counts.df,
                                    ERCC.meta.df,
                                    granularity=30,
                                    write.noise.model=TRUE,
                                    file="noise_estimation",
                                    model_view=c("Observed","Optimized"))
```

4 Options

Several options exist:

- granularity determines the number of bins for comparison of the quality of fit between the mixed-model and observed data for each alpha. This should be set lower for small datasets and higher for datasets with more observations
- write.noise.model=TRUE outputs two tab-delimited files containing the dropout effects and noise model parameters; this allows users to apply the noise generation on a seperate high compute node.
- plot==TRUE will plot all linear fits and individual ERCCs distributions across samples, where model_view=c("Observed", "Optimized", "Poisson", "Neg. Binomial" determines the statistical distributions that should be plotted for the ERCC plots.
- file="./Rplot" determines the root name for all plots, which write to the current working directory unless a path is contained in the root name.

5 Details

Following estimation of noise, the parameters are used to generate a noise-injected counts matrix.

```
noisy_counts.list <- create_noiseinjected_counts(results, n=10)
```

results is the list object generated by the function estimate_noiseparameters() and the variable n is the desired number of clusters. The resulting object is a list, where each element is a noise-injected counts matrix, and one element is the original counts matrix.

For larger datasets it is recommended that the user set write.noise.model=TRUE and copy the written bayesian drop-out and noise estimate files with the observed counts table to a high powered computing environment. The script HPC_generate_noise_matrices contains create_noiseinjected_counts() functions that are adapted to a parallel environment along with suggested code, which is commented out for user-modification. The script generates seperate noise-injected counts files, which can be loaded into R as a list or re-clustered seperately in a high powered compute environment.

After generating noise-injected counts tables, these should be re-clustered using the clustering method applied to the original dataset. For simplicity, here we use hierarchical clustering on a euclidean distance metric to identify two clusters. In our experience, some published clustering algorithms are sensitive to cell order, so we suggest scrambling the order of cells for each noise iteration as we do below in the function, recluster().

To quickly recluster a list, we define a reclustering function:

```
recluster <- function(x) {
  x <- data.frame(x, row.names = "GENE_ID")
  scramble <- sample(colnames(x), size=length(colnames(x)), replace=FALSE)
  x <- x[,scramble]
  clust <- hclust(dist(t(x),method="euclidean"),method="complete")
  clust <- cutree(clust,2)
  data.frame(clust)
}
```

We then apply the function recluster() to all noise-injected counts matrices and the original counts matrix and manipulate the list into a data.frame.

```
cluster.list<-lapply(noisy_counts.list, `recluster`)
clusters.df<-do.call("cbind", cluster.list)
colnames(clusters.df)<-names(cluster.list)
```

If running clustering algorithms on a seperate high power cluster, the user should retrieve labels and format as a data.frame of cluster labels, where the last column must be the original cluster labels derived from the observed count data. As an example, examine the file, example/example_clusters.tsv.

Using the cluster labels file as described above, we can generate a noise consensus matrix using:

```
noise_consensus <- compute_consensus(clusters.df)
```

Using the aheatmap() function in the NMF library, the consensus matrix result of 30 iterations of BEARscc on the provided example data will look this:

To reproduce the plot run:

```
library("NMF")
aheatmap(noise_consensus, breaks=0.5)
```

In order to interpret the noise consensus, we have defined three cluster (and analagous cell) metrics. Stabilty indicates the propensity for a putative cluster to contain the same cells across noise-injected counts matrices. Promiscuity indicates a tendency for cells in a putative cluster to associate with other clusters across noise-injected counts matrices. Score represents the promiscuity substracted from the stability.

We have found it useful to identify the optimal number of clusters in terms of resiliance to noise by examining these metrics by cutting hierarchical clustering dendograms of the noise consensus and comparing the results to the original clustering labels. To do this create a vector containing each number of clusters one wishes to examine (the function automatically determines the results for the dataset as a single cluster) and then cluster the consensus with cluster_consensus():

```
vector <- seq(from=2, to=5, by=1)
BEARscc_clusts.df <- cluster_consensus(noise_consensus,vector)
```

We add the original clustering to the data.frame:

```
BEARscc_clusts.df <- cbind(BEARscc_clusts.df, Original=clusters.df$Original_counts)
```

Compute cluster metrics by running the command:

```
cluster_scores.dt <- report_cluster_metrics(BEARscc_clusts.df,noise_consensus, plot=TRUE, file="example")
```

The output is a melted data.table that displays the name of each cluster, the size of each cluster, the metric (score, Promiscuity, Stability), the value of each metric for the respective cluster and clustering, the clustering in question (1,2,. . . ,Original), whether the cluster consists of only one cell, and finally the mean of each metric across all clusters in a clustering.

An example of the resulting plot for 3 noise-injected perturbations is provided for the user's reference: example/example_cluster_scores.pdf. It is evident from the plot that one cluster is optimal and outperforms the original clustering which bifurcated this set of purely technical data into 2 clusters.

Likewise, the cell metrics may be computed using:

```
cell_scores.dt <- report_cell_metrics(BEARscc_clusts.df, noise_consensus)
```

The output is a melted data.table that displays the name of each cluster to which the cell belongs, the cell label, the size of each cluster, the metric (score, Promiscuity, Stability), the value for each metric, and finally the clustering in question (1,2,. . . ,Original).

These results can be plotted to visualize cells in the context of different clusterings using ggplot2.

6 License

This software is made available under the terms of the GNU General Public License v3

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Vignettes are long form documentation commonly included in packages. Because they are part of the distribution of the package, they need to be as compact as possible. The html_vignette output type provides a custom style sheet (and tweaks some options) to ensure that the resulting html is as small as possible. The html_vignette format:

- Never uses retina figures
- Has a smaller default figure size
- Uses a custom CSS stylesheet instead of the default Twitter Bootstrap style