Efficient manipulation of sparse data

Michael Love
love@molgen.mpg.de

June 26, 2013

Abstract

This package allows for efficient manipulation of experiment data using sparse matrix representations. The sparse matrix representation used is the *dgCMatrix* class from the Matrix package. The SparseData package allows users to quickly calculate t-statistics across conditions, in order to provide a ranking of features by their specificity for a given condition. Other functions are provided for efficient calculation of similarity/distance measures.

Contents

1 Quick start **1**
1.1 Point to count files and feature files 1
1.2 Make pheno and feature data 2
1.3 Build a SparseDataSet 2
1.4 Normalize 2
1.5 Calculate means and t-statistics 2
1.6 Get regions by decreasing t-statistic 3

2 Input data **3**

3 Creating a new *SparseDataSet* **4**

4 Calculating statistics **5**

5 Calculating correlation and distance matrices **6**

6 Details on t-statistics **7**

7 Combining *SparseDataSet* objects **8**

8 Session info **10**

1 Quick start

1.1 Point to count files and feature files

```
> library(SparseData)
> sparse.data.files <- list.files(system.file("extdata",package="SparseData"),
+                                 "counts", full=TRUE)
> sparse.data.names <- list.files(system.file("extdata",package="SparseData"),
+                                 "counts")
> feature.file <- list.files(system.file("extdata",package="SparseData"),
+                            "ranges", full=TRUE)
> feature.file.name <- list.files(system.file("extdata",package="SparseData"),
+                                 "ranges")
```

```
> # these filenames look like:
> sparse.data.names[1]

[1] "GSM493385_UW.Fetal_Kidney.counts"

> feature.file.name

[1] "ranges_hg19_200bp_masked_sorted_subset.bed"
```

Note: the `sparse.data.files` are counts of reads in genomic ranges, generated using the BED-Tools suite. The output is sorted to correspond with a sorted BED file.

```
bedtools coverage -abam filename.bam -b sorted_regions.bed -counts |
   sort -k 1,1 -k 2,2n | cut -f 4 > filename.counts
```

Warning: this sorts the count files alphabetically by chromosome and then numerically by the starting base pair. The ranges must also be sorted this way, or else the counts will not correspond.

1.2 Make pheno and feature data

```
> sparse.data.conditions <- sub(".+(Fetal.+)\\.counts","\\1",sparse.data.names)
> phenoData <- AnnotatedDataFrame(data.frame(filename=sparse.data.names,
+                                            conditions=sparse.data.conditions))
> feature.data.frame <- read.delim(feature.file, header=FALSE)
> featureData <- AnnotatedDataFrame(data.frame(chr=feature.data.frame[,1],
+                                              start=feature.data.frame[,2],
+                                              end=feature.data.frame[,3]))
```

1.3 Build a SparseDataSet

```
> # threshold incoming count files at 50%
> sparse.data.list <- lapply(sparse.data.files, function(filename) {
+   sparseThreshold(Matrix(scan(filename,quiet=TRUE), sparse=TRUE), nzr=.5)
+ })
> # bind the sparse columns together as a sparse matrix
> sparse.data <- do.call(cBind, sparse.data.list)
> sds <- newSparseDataSet(sparseData = sparse.data,
+                         conditions=phenoData$conditions,
+                         featureData=featureData,
+                         phenoData=phenoData)
```

1.4 Normalize

```
> logPlusOne <- function(x) log(x + 1)
> sparseData(sds) <- applyFunctionSparsely(sparseData(sds), logPlusOne)
> norm.mat <- Matrix(diag(1/colMeans(sparseData(sds))),sparse=TRUE)
> colnames(norm.mat) <- rownames(pData(sds))
> sparseData(sds) <- sparseData(sds) %*% norm.mat
```

1.5 Calculate means and t-statistics

```
> options(mc.cores=1)
> sds <- calculateMeans(sds)
> sds <- calculateTStats(sds)
```

1.6 Get regions by decreasing t-statistic

```
> # the top five features specific to Fetal_Brain
> fData(sds)[head(order(-tStats(sds)[["Fetal_Brain"]]),5),]

      chr    start      end
103 chr11 32221801 32222000
102 chr11 32221601 32221800
15  chr11 32198401 32198600
14  chr11 32198201 32198400
188 chr11 32247201 32247400

> # the top five global features
> fData(sds)[head(order(-means(sds)[["global"]]),5),]

       chr    start      end
1258 chr11 32605201 32605400
12   chr11 32197801 32198000
1957 chr11 32915801 32916000
1257 chr11 32605001 32605200
1259 chr11 32605401 32605600
```

2 Input data

We start by reading in some example data in order to create a *SparseDataSet* object. The example data are counts of DNase-seq reads, generated by the Roadmap Epigenome Mapping Consortium, in 2000 genomic ranges of 200 bp. The sample data is listed in `phenoData` and the ranges are listed in `featureData`. The ranges are a subset of nonoverlapping ranges covering the genome, after removing ranges which had more than 25% overlap with a RepeatMasker region of score greater than 1000. The data file contains the name including GSM number from the GEO series GSE18927 which is available for download at `http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE18927`.

```
> library(SparseData)
> sparse.data.files <- list.files(system.file("extdata",package="SparseData"),
+                                 "counts", full=TRUE)
> sparse.data.names <- list.files(system.file("extdata",package="SparseData"),
+                                 "counts")
> sparse.data.conditions <- sub(".+(Fetal.+)\\.counts","\\1",sparse.data.names)
> phenoData <- AnnotatedDataFrame(data.frame(filename=sparse.data.names,
+                                            conditions=sparse.data.conditions))
> feature.file <- list.files(system.file("extdata",package="SparseData"),
+                            "ranges", full=TRUE)
> feature.data.frame <- read.delim(feature.file, header=FALSE)
> featureData <- AnnotatedDataFrame(data.frame(chr=feature.data.frame[,1],
+                                              start=feature.data.frame[,2],
+                                              end=feature.data.frame[,3]))
```

The `sparse.data.files` are counts of reads in genomic ranges, generated using the BEDTools suite, with calls:

```
bedtools coverage -abam filename.bam -b sorted_regions.bed -counts |
sort -k 1,1 -k 2,2n | cut -f 4 > filename.counts
```

Alternatively, counts of reads in genomic regions can be found using the python package HTSeq `http://www-huber.embl.de/users/anders/HTSeq/doc/overview.html` or the `summarizeOverlaps` function of the GenomicRanges package.

Here we show one method for constructing a sparse matrix from individual files containing a single column of counts defined over the same ranges. However, the sparse matrix data can be created in any way described by the Matrix package.

We read in a single column of counts using `scan`. These numeric vectors are converted to sparse matrices using `Matrix` from the Matrix package with argument `sparse=TRUE`. A function of this package, `sparseThreshold`, is called, which pushes small values (in absolute value) to zero to achieve a desired nonzero ratio. A reasonable threshold will vary for different datasets and depending on the memory and time savings desired. Finally the list of sparse single-column matrices are bound together using `cBind`, the equivalent function to `cbind`, defined in the Matrix package.

```
> quantile(scan(sparse.data.files[1],quiet=TRUE),0:10/10)

  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
   0    0    0    0    1    1    1    2    3    4  358

> sparse.data.list <- lapply(sparse.data.files, function(filename) {
+   sparseThreshold(Matrix(scan(filename,quiet=TRUE), sparse=TRUE), nzr=.5)
+ })
> sparse.data <- do.call(cBind, sparse.data.list)
```

3 Creating a new *SparseDataSet*

Now we create a *SparseDataSet* object:

```
> sds <- newSparseDataSet(sparseData = sparse.data,
+                         conditions=phenoData$conditions,
+                         featureData=featureData,
+                         phenoData=phenoData)
```

We also include the experiment data:

```
> expData <- new("MIAME",
+ name="2000 ranges of DNase-seq from Roadmap Epigenome Mapping Consortium",
+ lab="University of Washington",
+ contact="rharris1@bcm.tmc.edu",
+ title="Human Reference Epigenome Mapping Project",
+ url="http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE18927")
> pubMedIds(expData) <- "20944595"
> experimentData(sds) <- expData
```

Now we have a *SparseDataSet* object with the sparse matrix accessible via `sparseData`, and the standard `phenoData` and `featureData` functions.

```
> head(sparseData(sds),10)

10 x 15 sparse Matrix of class "dgCMatrix"

1   4  9 10  5 12  8  6 28 17  3  4  7  .  9  4
2   5 41  5  8 19 30 52 60 53  5  3  9  6 16 13
3   4 16  8 11  5 16 20 38 31  3  6 22  6 12 12
4   .  9  3 10  3  9 15 31 20 15  7 13  7 24 12
5  12 24 18 24 34 19 46 42 57 15 10 40 17 41 47
6   3  5  8  5  6  6 12 18 25  8  7 38  4 24 19
7   3  3  .  .  3  .  .  3  2  3  2  .  .  .  .
8   4  4  2  .  2  2  3  9 13  2  .  4  .  .  3
9   .  .  2  3  4  5  7  8 13  3  .  4  .  .  3
10  3  .  2  5  .  4  7  9 12  4  . 11  3  8  5
```

```
> head(pData(sds),3)
```

```
                          filename   conditions    condition
1 GSM493385_UW.Fetal_Kidney.counts Fetal_Kidney Fetal_Kidney
2  GSM530651_UW.Fetal_Brain.counts  Fetal_Brain  Fetal_Brain
3  GSM530654_UW.Fetal_Heart.counts  Fetal_Heart  Fetal_Heart
```

```
> head(fData(sds),3)
```

```
    chr    start      end
1 chr11 32194201 32194400
2 chr11 32194401 32194600
3 chr11 32194601 32194800
```

The sparsity of the matrix can be calculated using the `nnzero` function from the `Matrix` package.

```
> nnzero(sparseData(sds))/prod(dim(sds))
```

```
[1] 0.3984333
```

While multiplication on sparse matrices works as expected, we implement a function `applyFunctionSparsely` which allows other operations to be called only on the nonzero elements of the matrix. Here we demonstrate taking $log(x+1)$ on the nonzero elements. Also demonstrated is an example of normalization by dividing each column by its mean (multiplying on the right by a diagonal matrix with elements 1/column-mean).

```
> logPlusOne <- function(x) log(x + 1)
> sparseData(sds) <- applyFunctionSparsely(sparseData(sds), logPlusOne)
> norm.mat <- Matrix(diag(1/colMeans(sparseData(sds))),sparse=TRUE)
> colnames(norm.mat) <- rownames(pData(sds))
> sparseData(sds) <- sparseData(sds) %*% norm.mat
```

4 Calculating statistics

`SparseDataSet` methods allow for calculation of means, sum of squares from the means, and t-statistics for all conditions. The `calculateMeans` function calculates means and sum of squares for each condition, as well as a "global" mean of condition means and a "global" sum of condition sum of squares. The `calculateMeans` function makes use of the `mclapply` function in the parallel package, allowing the user to distribute means and sum of squares calculations across multiple cores. Extra arguments to `calculateMeans` are passed to `mclapply`. The t-statistics which are calculated compare each condition to the mean of all condition means, dividing by a pooled within-condition standard deviation. More details on the t-statistic calculation is provided later in this vignette and in the man page.

```
> options(mc.cores=1)
> sds <- calculateMeans(sds)
> sds <- calculateTStats(sds)
```

The `calculateMeans` function avoids recalculating means and sum of squares unless the user sets `recalc=TRUE`. This allows for the use of `combine` to add new samples without having to recalculate means and sum of squares for conditions that do not gain samples, as shown later in the vignette. Here we see from the order of the list that the recalculated condition mean was added to the end of the list.

```
> names(means(sds))
```

```
[1] "Fetal_Brain"  "Fetal_Heart"  "Fetal_Kidney" "global"
```

```
> means(sds)[["Fetal_Brain"]] <- NULL
> sds <- calculateMeans(sds)
> names(means(sds))
```

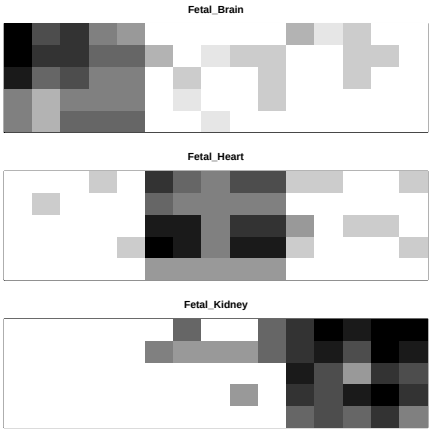
```
[1] "Fetal_Heart"  "Fetal_Kidney" "global"       "Fetal_Brain"
```

We can find the features for each condition with the largest t-statistic:

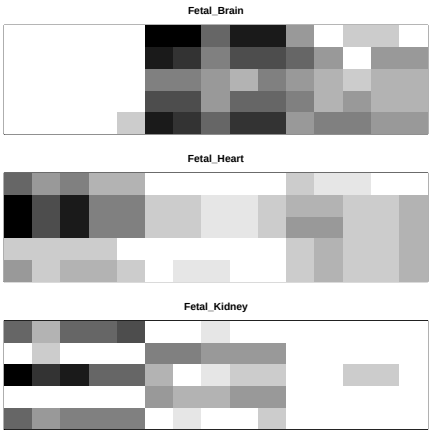
```
> fData(sds)[head(order(-tStats(sds)[["Fetal_Brain"]]),5),]
```

```
      chr    start      end
103 chr11 32221801 32222000
102 chr11 32221601 32221800
15  chr11 32198401 32198600
14  chr11 32198201 32198400
188 chr11 32247201 32247400
```

Here we plot the features with largest t-statistic, using the function `image` defined in the Matrix package for sparse matrices. These are features where the condition of interest has higher values than the mean of all conditions.



Plotting the features with smallest t-statistic, where the condition of interest has smaller values than the mean of all conditions.

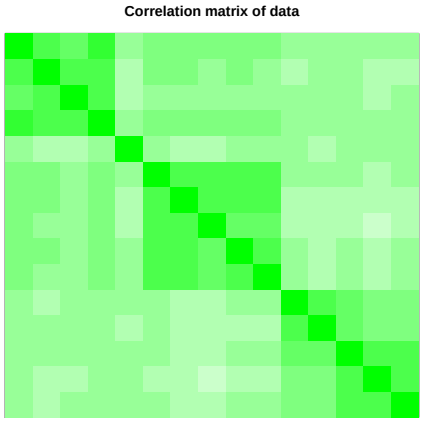


5 Calculating correlation and distance matrices

The package provides a function `sparseCov` for calculating covariance and correlation matrices without losing the sparsity of the data. The function calculates the covariance and correlation

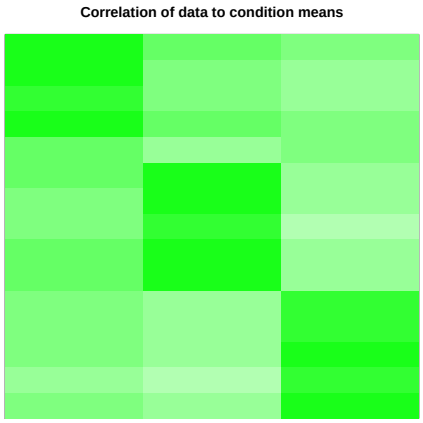
matrix simultaneously, and returns a list with names `cov` and `cor`. Other functions provided in this package include `sparseCosine` for the cosine similarity, and `sparseEuclid` for Euclidean distance. See the timing vignette for comparisons with dense calculations.

```
> cormat <- sparseCov(sparseData(sds))$cor
```



We can calculate the correlation of the data to the matrix of means.

```
> means.matrix <- do.call(cBind, means(sds)[match(
+   levels(pData(sds)$condition), names(means(sds)))])
> match.cormat <- sparseCov(sparseData(sds), means.matrix)$cor
```



6 Details on t-statistics

The t-statistics are calculated comparing a single condition against all samples (including that condition). Some differences between the t-statistic provided here and the "typical" equal-variance t-statistic:

- The global mean used is weighted (the mean of condition means), rather than a simple mean of all samples.
- The denominator of our t-statistic includes the sum of squared distances to the condition means, rather than to the mean of all samples.
- An offset is included in the denominator (the mean of the pooled standard deviation over all features) to avoid division by zero.

The resulting t-statistics are closely related in rank to the typical equal-variance t-statistics. Below we provide a formula, using the notation of Tibshirani, R., Hastie, T., Narasimhan, B. & Chu, G. "Diagnosis of multiple cancer types by shrunken centroids of gene expression". Proceedings of the National Academy of Sciences 99, 6567-6572 (2002). For n_k samples in condition k, n total samples, K conditions, weighted mean μ_w and sum of squares of samples to their condition means SSE^*, the t-statistic provided by `calculateTStats` is defined by:

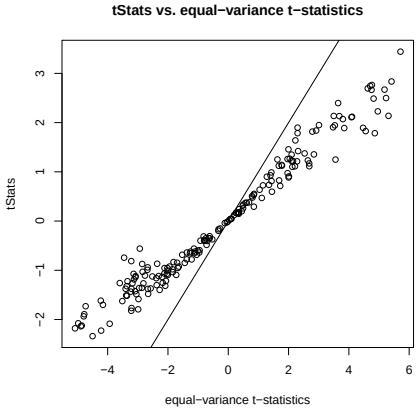
$$s = \sqrt{SSE^*/(n-K)}$$
$$t_k = \frac{1}{\sqrt{(1/n_k + 1/n)}} \left(\frac{\mu_k - \mu_w}{s + \text{offset}} \right)$$

For sum of squares for condition k SSE_k, the mean of all samples μ, and sum of squares of samples to the mean of all samples SSE, the equal-variance t-statistic is:

$$s = \sqrt{(SSE_k + SSE)/(n_k + n - 2)}$$
$$t_k = \frac{1}{\sqrt{(1/n_k + 1/n)}} \left(\frac{\mu_k - \mu}{s} \right)$$

A scatter plot of `tStats` against the equal-variance t-statistic calculated by `t.test`.

```
> sim.sds <- simulateSparseDataSet(200, c(50, 50, 50), nzg = 0.5, nzs = 0.5)
> sim.sds <- calculateMeans(sim.sds, quiet=TRUE)
> sim.sds <- calculateTStats(sim.sds, quiet=TRUE)
> equalvar.t.stats <- sapply(1:nrow(sim.sds), function(i) t.test(
+   x=sparseData(sim.sds)[i,pData(sim.sds)$condition == "c1"],
+   y=sparseData(sim.sds)[i,], var.equal=TRUE)$statistic)
> plot(equalvar.t.stats, tStats(sim.sds)[["c1"]],
+      xlab="equal-variance t-statistics",ylab="tStats",
+      main="tStats vs. equal-variance t-statistics")
> abline(0,1)
```



7 Combining *SparseDataSet* objects

Methods inherited from *eSet* should work as expected, including indexing (because the default for *eSet* is to set `drop = FALSE`, preserving the sparsity of the matrix stored in assayData). We define a method `combine` for the class `SparseDataSet` and the class `dgCMatrix`. Row slicing of column sparse matrices can be very slow if the matrix is large, therefore we only allow combination of objects which have the same feature names in the same order, and we also require that the sample names of the two objects have no intersection.

Here we demonstrate combination of two `SparseDataSet` objects. We add unique sample names to the object `y`.

```
> x <- simulateSparseDataSet(10,c(2,2,2))
> y <- simulateSparseDataSet(10,c(2,2))
> sampleNames(y) <- paste("sample",(ncol(x) + 1:ncol(y)),sep="")
> pData(y)$sampleID <- sampleNames(y)
> z <- combine(x,y)
> pData(z)
```

```
         sampleID condition
sample1   sample1        c1
sample2   sample2        c1
sample3   sample3        c2
sample4   sample4        c2
sample5   sample5        c3
sample6   sample6        c3
sample7   sample7        c1
sample8   sample8        c1
sample9   sample9        c2
sample10 sample10        c2
```

```
> sparseData(z)
```

```
10 x 10 sparse Matrix of class "dgCMatrix"

feat1   .  .  .  .  .  .  .  .  .  .
feat2   .  .  .  .  .  .  .  .  .  .
feat3   .  .  .  .  .  .  .  .  .  .
feat4   .  .  .  .  .  .  .  .  .  .
feat5  23 41  1 54 23 2 .  .  .  .
feat6   . 16 13  . 31 5 .  .  .  .
feat7   .  .  .  .  .  .  .  .  8  1
feat8   2  4  .  .  .  .  .  .  .  .
feat9   .  .  .  .  .  . 8  . 10  4
feat10  .  .  4  2  .  .  .  .  .  .
```

Furthermore, the combine method will check to see if there are shared conditions between the two *SparseDataSet* objects. If so, the means and sum of squares for these will be removed, as they need to be recalculated. The t-statistics are also removed as the global mean and global sum of squares have changed.

```
> x <- calculateMeans(x)
> x <- calculateTStats(x)
> y <- calculateMeans(y)
> y <- calculateTStats(y)
> z <- combine(x,y)
> names(means(z))
```

```
[1] "c3"
```

```
> names(tStats(z))
```

```
NULL
```

```
> z <- calculateMeans(z)
> z <- calculateTStats(z)
> names(means(z))
```

```
[1] "c3"     "c1"     "c2"     "global"
```

```
> names(tStats(z))
```

```
[1] "c1" "c2" "c3"
```

8 Session info

```
> sessionInfo()

R version 3.0.1 (2013-05-16)
Platform: x86_64-apple-darwin10.8.0 (64-bit)

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

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

other attached packages:
[1] SparseData_1.0.0   Biobase_2.20.0     BiocGenerics_0.6.0 Matrix_1.0-12
[5] lattice_0.20-15

loaded via a namespace (and not attached):
[1] grid_3.0.1  tools_3.0.1
```