Description of GCS-score: Expression Analysis of WT-Type Affymetrix GeneChips from Probe-Level Data

Guy M. Harris, Shahroze Abbas,
and Michael F. Miles

October 30, 2019

Contents

1 Introduction **2**

2 What's new in this version **2**

3 Reading in data and generating S-Scores **2**

4 Submitting a batch job **5**

5 Using GCS-Scores in gene expression analysis **7**

6 Version history **8**

7 Acknowledgements **8**

1 Introduction

The S-Score algorithm, described by Zhang et al. (2002), Kerns et al. (2003), and Kennedy et al. (2006b) is a novel comparative method for gene expression data analysis that performs tests of hypotheses directly from probe level data. It is based on an error model in which the detected signal is assumed to be proportional to the probe signal for highly expressed genes, but assumed to approach a background level (rather than 0) for genes with low levels of expression. This error model is used to calculate relative changes in probe intensities that converts probe signals into multiple measurements with equalized errors, which are summed over a probe set to form the significance score (S-score). The original S-score method required the mismatch (MM) probes to estimate non-specific binding (NSB) for each perfect-match (PM) probes, and the MM probes were removed the arrays beginning with the Affymetrix Whole Transcriptome (WT) style arrays. This new algorithm uses a gc-content based NSB, thus eliminating the original algorithm's dependence on MM probes. The GCS-score algorithm is capable of working on all modern Affymetrix array types (3' IVT and up). Assuming no expression differences between chips, the GCS-score output follows a standard normal distribution. Thus, a separate step estimating the probe set expression summary values is not needed and p-values can be easily calculated from the GCS-score output. Furthermore, in previous comparisons of dilution and spike-in microarray datasets, the original S-Score demonstrated greater sensitivity than many existing methods, without sacrificing specificity (Kennedy et al., 2006a). The *GCSscore* package (Harris et al., 2019) implements the GCS-score algorithm in the R programming environment, making it available to users of the Bioconductor [1] project.

2 What's new in this version

This is the initial release.

3 Reading in data and generating S-Scores

Affymetrix data are generated from microarrays by analyzing the scanned image of the chip (stored in a *.DAT file) to produce a *.CEL file. The *.CEL file contains, among other information, a decimal number for each probe on the chip that corresponds to its intensity. The GCS-score algorithm compares two microarrays by combining all of the probe intensities from a probesetID / transcriptionclusterID into a single summary statistic for each annotated gene or exon. The *GCSscore* package processes the data obtained from .CEL files, which must be loaded into R prior to calling the `GCSscore` function. Thus, the typical sequence of steps to accomplish this is as follows:

[1] `http://www.bioconductor.org/`

1. Create a directory containing all *.CEL files relevant to the planned analysis.

2. Load the library.

```
> # load GCSscore package:
> library(GCSscore)
>
> # load data.table package for parsing data files
> # outside of the GCSscore package:
> # library(data.table)
```

The `GCSscore` function utilizes the `readCel` function to directly access the individual .CEL files. Additional information regarding the `readCel` function and detailed description of the structure of .CEL files can be found in the *affxparser* vignette. The `readCel` function allows the *GCSscore* package to access additional variables that are necessary for the noise and probe error estimations. Almost all other reader packages only read the probe intensities values from the .CEL files

The `GCSscore` function returns an object of class `data.table`. The class `data.table` is described in the *data.table* package, which is available on CRAN. The GCS-score values are returned in a `data.table` object, along with relevant annotation information. The following examples illustrate the *GCSscore* package. These examples utilize the .CEL data files that are supplied with the *GCSscore* package.

A basic GCS-score analysis is generated using the `GCSscore` function:

```
> # get the path to example CEL files in the package directory:
> celpath1 <- system.file("extdata/","MN_2_3.CEL", package = "GCSscore")
> celpath2 <- system.file("extdata/","MN_4_1.CEL", package = "GCSscore")
> # run GCSscore() function directly on the two .CEL files above:
> GCSs.single <- GCSscore(celFile1 = celpath1, celFile2 = celpath2)
```

The returned object object uses the Biobase data structure: ExpressionSet. The GCS-score differential expression is contained in the assayData. For viewing and exporting, it is often desirable to convert the ExpressionSet back to a data.table/data.frame object:

```
> # view class of output:
> class(GCSs.single)[1]

[1] "ExpressionSet"

> # convert GCSscore single-run from ExpressionSet to data.table:
> GCSs.single.dt <-
+   data.table::as.data.table(cbind(GCSs.single@featureData@data,
+                                   GCSs.single@assayData[["exprs"]]))
```

```
> # preview the beginning and end of the output.
> # *remove 'gene_name' column for printing to PDF:
> head(GCSs.single.dt[,-c("gene_name","nProbes")])

   transcriptclusterid  symbol     Sscore
1:   TC0100000014.mm.2 Atp6v1h  0.4376491
2:   TC0100000018.mm.2   Oprk1 -0.4870829
3:   TC0100000021.mm.2  Rb1cc1 -2.2629908
4:   TC0100000022.mm.2  Alkal1  1.2842710
5:   TC0100000023.mm.2    St18 -0.3077626
6:   TC0100000027.mm.2  Pcmtd1  0.1158114
```

Parameters for `GCSscore` function include:

celFile1 – character string giving the .CEL file name the directory in which the *.CEL files are stored. If a directory is not specified, the current working directory is used.

celTable – A CSV file containing batch submission information.

fileout – Determines if the resulting GCS-score coutput is written to disk in a CSV format following the completion of the function. By default, this is set to FALSE so unnecessary GCS-score outputs are not saved to disk after each run. Each output that is written to file includes a timestap for later reference.

celTab.names – If set to TRUE, then the GCS-score batch output is assigned the user-designated name, as specified in the first column of the batch input .CSV file. If set to FALSE, when the user submits a batch job, the column name of the run in the the batch output `data.table` will be: CELfilename1 vs CELfilename2.

typeFilter – If set to 0, all available probe types are included in the calculation and normalization of the GCS-score values. If set to 1, only probes well-annotated probeids (from BioConductor .db packages) are included in the calculation and normalization of the GCS-score output.

method – This determines the method used to group and tally the probeids when calculating GCS-scores. For Whole Transcriptome (WT) arrays, for gene-level (transcriptclusterid-based) analysis, set method = 1. For exon-level (probesetid-based) analysis, set method = 2. For the older generation arrays (3 IVT-style), if a GC-content based background correction is desired on the 3 IVT arrays, set method = 1, if a PM-MM based background correction is desired, set method = 2 (PM-MM gives identical results to the original S-score package).

rm.outmask – If set to TRUE, then probes that are flagged as MASKED or OUTLIER in either celFile1 or celFile2 will be removed from the analysis. If set to FALSE, these probes are not filtered out and will be used in the GCS-score calculation.

SF1 and SF2 – the Scaling Factors (SF1 and SF2). The Scaling Factors are used to scale the median raw intensities of the probe grouping method on both chips to a target value, in this case that value is 500. The Standard Difference Threshold (SDTs) is used as an estimate of background noise on each chip, and is equal to the standard deviation for the lowest 2% of probe intensities from 16 differnet zones on each chip. These values are calculated internally by the `GCSscore` function.

fileout – Determines if the resulting GCS-score output is written to disk in a .CSV format following the completion of the function. By default, this is set to FALSE so unnecessary GCS-score outputs are not saved to disk after each run.

verbose – a logical value indicating whether internally calculated values (SF, RawQ, SDT) are returned to the console during the analysis.

4 Submitting a batch job

The `GCSscore` function is able to output mulitple GCS-score runs to a single file. This is done by leaving `celFile1` and `celFile2` variables empty, and using the `celTable` argument instead. The `celTable` argument accepts a three column `data.table` object, that is read into R from a .CSV file via the `fread` function from the *data.table* package.

```
> # get the path to example CSV file in the package directory:
> celtab_path <- system.file("extdata",
+                            "Ss2_BATCH_example.csv",
+                            package = "GCSscore")
> # read in the .CSV file with fread():
> celtab <- data.table::fread(celtab_path)
> # view structure of 'celTable' input:
> celtab

    run_name   CelFile1   CelFile2
1: example01 MN_2_3.CEL MN_4_1.CEL
2: example02 MN_2_3.CEL MN_4_2.CEL
3: example03 MN_2_3.CEL MN_4_3.CEL
4: example04 MN_4_1.CEL MN_4_2.CEL
5: example05 MN_4_1.CEL MN_4_3.CEL
6: example06 MN_4_2.CEL MN_4_3.CEL

> # For the following example, the .CEL files are not in the working
> # directory.  The path to the .CEL files must be added to allow
> # the GCSscore() function to find them:
>
> # adds path to celFile names in batch input:
```

```
> #   NOTE: this is not necessary if the .CEL files
> #         are in the working directory:
> path <- system.file("extdata", package = "GCSscore")
> celtab$CelFile1 <- celtab[,paste(path,CelFile1,sep="/")]
> celtab$CelFile2 <- celtab[,paste(path,CelFile2,sep="/")]
```

The GCSscore function will loop through all the runs listed in .CSV file before all GCS-score values are assigned to the end of the same annotation file (each GCS-score run is contained within one column). If the celTab.names is set to TRUE, the column names of each run will correspond to the run name assigned in the first column of the .CSV batch input file. In this example, all four .CEL files included with the package are run in pairwise fashion.

```
> # run GCSscore() function with batch input:
> GCSs.batch <- GCSscore(celTable = celtab, celTab.names = TRUE)
```

The ExpressionSet returned from the GCSscore package can easily be converted back to a data.table structure. This matches the structure of the .CSV file that is created if the fileout option is set to TRUE. The conversion of the ExpressionSet object to data.table is as follows:

```
> # view class of output:
> class(GCSs.batch)[1]

[1] "ExpressionSet"

> # converting GCS-score output from'ExpressionSet' to 'data.table':
> GCSs.batch.dt <-
+   data.table::as.data.table(cbind(GCSs.batch@featureData@data,
+                                   GCSs.batch@assayData[["exprs"]]))
> # preview the beginning and output of the batch output:
> # *remove 'gene_name' and 'nProbes' columns for printing to PDF:
> head(GCSs.batch.dt[,-c("gene_name","nProbes")])

   transcriptclusterid  symbol  example01    example02   example03  example04
1:   TC0100000014.mm.2 Atp6v1h  0.4376491  0.177223857 -1.36062277 -0.1261603
2:   TC0100000018.mm.2   Oprk1 -0.4870829  1.690161845  1.13432022  2.1157907
3:   TC0100000021.mm.2  Rb1cc1 -2.2629908  0.410159306 -2.95364801  2.2224996
4:   TC0100000022.mm.2  Alkal1  1.2842710 -0.006707483  0.43544482 -1.0419209
5:   TC0100000023.mm.2    St18 -0.3077626  1.269376283  0.15349238  1.6471287
6:   TC0100000027.mm.2  Pcmtd1  0.1158114  1.791275559  0.06975856  1.7688242
     example05  example06
1: -1.96527771 -1.4024547
```

```
2:  1.72750417 -0.6592905
3: -1.26128536 -2.9990005
4: -0.83925001  0.4437061
5:  0.62318825 -1.1199760
6: -0.02660543 -1.6857510
```

5 Using GCS-Scores in gene expression analysis

Under conditions of no differential expression, the GCS-Score output follows a standard normal (Gaussian) distribution with a mean of 0 and standard deviation of 1. This makes it straightforward to calculate p-values corresponding to rejection of the null hypothesis and acceptance of the alternative hypothesis of differential gene expression. Cutoff values for the GCS-scores can be set to achieve the desired level of significance. As an example, an absolute GCS-score value of 3 (signifying 3 standard deviations from the mean, a typical cutoff value) would correspond to a p-value of 0.003. Under this scenario, the significant genes can be found as:

```
> ## find scores greater than 3 SD:
> signif <- GCSs.single.dt[abs(Sscore) >= 3]
> # View the resulting table:
> # removing 'gene_name' and 'nProbes' columns for PDF printing:
> head(signif[,-c("gene_name","nProbes")])
```

```
   transcriptclusterid  symbol    Sscore
1:   TC0100000302.mm.2  Slc9a2 -4.508737
2:   TC0100000660.mm.2   Asic4 -3.753547
3:   TC0100002769.mm.2   Usp40  3.332291
4:   TC0100002844.mm.2 St8sia4  3.379366
5:   TC0100003586.mm.2   Ackr1 -3.312196
6:   TC0200000309.mm.2   Armc3  3.294991
```

Similarly, the p-values can be calculated as:

```
> # Calculate p-valus significant
> ## find the corresponding one-sided p-values:
> signif[,p.values.1 := (1 - pnorm(abs(signif[,Sscore])))]
> ## find the corresponding two-sided p-values
> signif[,p.values.2 := 2*(1 - pnorm(abs(signif[,Sscore])))]
> # sort the probe_ids by the absolute value of the Sscore:
> signif <- signif[order(abs(Sscore),decreasing = TRUE)]

> # View the top of the most differentially expressed genes
> # from the GCSs.single output:
```

```
>
> # removing 'gene_name' and 'nProbes' columns for PDF printing:
> head(signif[,-c("gene_name","nProbes")])
```

```
   transcriptclusterid        symbol    Sscore   p.values.1   p.values.2
1:   TC0Y00000223.mm.2         Erdr1 -5.825356 2.849552e-09 5.699104e-09
2:   TC0900001771.mm.2        Amotl1 -5.289357 6.137359e-08 1.227472e-07
3:   TC0900001632.mm.2 1700048O20Rik  4.607763 2.035124e-06 4.070247e-06
4:   TC0100000302.mm.2        Slc9a2 -4.508737 3.260727e-06 6.521455e-06
5:   TC1000000171.mm.2          Sgk1 -4.334561 7.302579e-06 1.460516e-05
6:   TC0200003560.mm.2         Nr4a2 -4.155619 1.622039e-05 3.244079e-05
```

While the GCS-score algorithm does account for the correlations among probes within a two-chip comparison, it does not adjust p-values for multiple comparisons when comparing more than one pair of chips. The calculations for the SF and SDT are performed as originally described in the Affymetrix Statistical Algorithms Description Document (Affymetrix, 2002) and implemented in Affymetrix software (using SDT = 4 * RawQ * SF). The calculations for each of the *.CEL files are independent.

6 Version history

1.0.0 first public release

0.0.1 initial development version

7 Acknowledgements

The development of the original S-Score algorithm and its original implementation in C++ is the work of Dr. Li Zhang. The Delphi implementation of the S-Score algorithm is the work of Dr. Robnet Kerns. The original S-score R package was work of Dr. Robert Kennedy. This work was partly supported by F30 training grant (F30AA025535) to Guy M. Harris and NIAAA research grant AA13678 to Michael F. Miles.

References

Affymetrix. Statistical Algorithms Description Document. Technical report, Affymetrix, 2002.

Guy M. Harris, Shahroze Abbas, and Michael F. Miles. GCSscore: an R package for differential gene expression detection in affymetrix/thermo-fisher whole transcriptome microarrays. *Bioinformatics*, page TBD, 2019.

Richard E. Kennedy, Kellie J. Archer, and Michael F. Miles. Empirical validation of the S-Score algorithm in the analysis of gene expression data. *BMC Bioinformatics*, 7: 154, 2006a.

Richard E. Kennedy, Robnet T. Kerns, Xiangrong Kong, Kellie J. Archer, and Michael F. Miles. SScore: An R package for detecting differential gene expression without gene expression summaries. *Bioinformatics*, 22(10):1272–1274, 2006b.

Robnet T. Kerns, Li Zhang, and Michael F. Miles. Application of the S-Score algorithm for analysis of oligonucleotide microarrays. *Methods*, 31(3):274–281, 2003.

Li Zhang, Long Wang, Ajay Ravindranathan, and Michael F. Miles. A new algorithm for analysis of oligonucleotide arrays: Application to expression profiling in mouse brain regions. *Journal of Molecular Biology*, 317(3):225–235, 2002.