

GCSscore: An R Package for Differential Expression Analysis of WT-Type Affymetrix Microarrays from Probe-Level Data

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

April 27, 2020

Contents

1	Introduction	2
2	What's new in this version	2
3	Reading in data and generating GCS-scores	2
4	Important Parameters for GCSscore function	4
5	Using the Batch Functionality of GCSscore	4
6	Natural Statistics of GCS-scores for Differential Gene Expression Analysis	6
7	Using GCS-Scores to produce DEGs in experimental datasets	8
8	Version history	12
9	Acknowledgements	12

1 Introduction

GCSscore is an R package for detecting differential expression on affymetrix microarrays that is available on Bioconductor and Github. It is based on the original 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 from 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 second public release of the *GCSscore* package. In this release: there are improvements to algorithm, multiple bug fixes, and major updates to the accompanying documentation.

3 Reading in data and generating GCS-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

¹<http://www.bioconductor.org/>

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. 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 examples in this vignette will illustrate the *GCSscore* package in use. We begin with an example of the most basic GCS-score analysis, which utilize the .CEL data files that are supplied with the *GCSscore* package:

```
> library(GCSscore)

> # 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 *GCSscore* function returns an object of class *ExpressionSet*, a format defined in the *Biobase* package. The GCS-score differential expression values are contained in the *assayData[["exprs"]]* section of the *ExpressionSet*. The full set of annotation information is included in the *featureData@data* section of the *ExpressionSet*. For viewing and exporting, it is often desirable to extract the relevant information from the *ExpressionSet* into an well structured object, such as a *data.table*. The class *data.table* is described in the *data.table* package, which is available on CRAN.

```
> # 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"]]))
> # show all column names included in the output:
> colnames(GCSs.single.dt)

[1] "transcriptclusterid" "symbol"          "name"
[4] "ref_id"              "db.symbol"       "db.name"
[7] "chr"                 "start"           "stop"
[10] "nProbes"             "locustype"        "category"
[13] "Sscore"
```

```
> # show simplified output of select columns and rows:
> GCSs.single.dt[10000:10005,
+               c("transcriptclusterid", "symbol",
+               "ref_id", "Sscore")]
```

	transcriptclusterid	symbol	ref_id	Sscore
1:	TC0300000948.mm.2	Hfe2	NM_027126	-1.1175008
2:	TC0300000949.mm.2	Txnip	NM_001009935	0.8790560
3:	TC0300000950.mm.2	Gm16253	ENSMUST00000148290	-0.1314328
4:	TC0300000951.mm.2	Ankrd34a	NM_001024851	-2.9600225
5:	TC0300000952.mm.2	Lix1l	NM_001163170	1.4886249
6:	TC0300000953.mm.2	Rbm8a	NM_001102407	1.2087873

4 Important Parameters for GCSscore function

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.

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: CEL-file-name1 vs CEL-file-name2.

method – This determines the method used to group and tally the probeids when calculating GCS-scores. For Whole Transcriptome 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).

5 Using the Batch Functionality of GCSscore

The **GCSscore** function is able to output mulitple GCS-score results into 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",
```

```

+                                     "GCSs_batch_ex.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

```

In these examples, the .CEL files will not be within the working directory. Therefore, the path to the .CEL files must be added to allow the `GCSscore` function to locate the files. 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 using using all info from the batch file:
> 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"]]))
> # show all column names included in the output:
> colnames(GCSs.batch.dt)

[1] "transcriptclusterid" "symbol"          "name"
[4] "ref_id"              "db.symbol"       "db.name"
[7] "chr"                 "start"           "stop"
[10] "nProbes"             "locustype"        "category"
[13] "example01"           "example02"        "example03"
[16] "example04"           "example05"        "example06"

> # show simplified output of select columns and rows:
> GCSs.batch.dt[10000:10005,
+               c("transcriptclusterid", "symbol",
+                 "example01", "example02", "example03")]
   transcriptclusterid  symbol example01 example02 example03
1:   TC0300000948.mm.2    Hfe2 -1.1175008 -0.87256331 -0.06293231
2:   TC0300000949.mm.2   Txnip  0.8790560  1.52472374  1.53629101
3:   TC0300000950.mm.2  Gm16253 -0.1314328 -0.87339382  0.52384165
4:   TC0300000951.mm.2 Ankrd34a -2.9600225 -2.36873229 -2.85869949
5:   TC0300000952.mm.2   Lix1l  1.4886249 -0.29257447  0.80216835
6:   TC0300000953.mm.2   Rbm8a  1.2087873  0.04885971  1.18225661

```

6 Natural Statistics of GCS-scores for Differential 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")])
> head(signif)
```

	transcriptclusterid	symbol	name	ref_id	db.symbol	db.name	chr	start	stop
1:	AFFX-BkGr-GC19_st	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	NA	NA
2:	AFFX-BkGr-GC23_st	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	NA	NA
3:	ERCCmix1step3	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	NA	NA
4:	ERCCmix2step7	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	NA	NA
5:	MG-1-neg-10338134_st	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	NA	NA
6:	MG-1-neg-10338223_st	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	NA	NA

	nProbes	locustype	category	Sscore
1:	849	<NA>	control->bgp->antigenomic	-4.139883
2:	407	<NA>	control->bgp->antigenomic	3.561830
3:	30	<NA>	control->affx->ercc->step	-3.319253
4:	30	<NA>	control->affx->ercc->step	-3.058580
5:	4	<NA>	normgene->intron	-3.103684
6:	4	<NA>	normgene->intron	3.035976

Similarly, the p-values can be calculated as:

```
> # Calculate p-values 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)
```

	transcriptclusterid	symbol
1:	TC1100000339.mm.2	Hba-a1
2:	TC1100000336.mm.2	Hba-a2
3:	TC0700003909.mm.2	Hbb-bs
4:	TC0700003908.mm.2	Hbb-bt
5:	TC0900000005.mm.2	Gm10719
6:	TC0Y00000223.mm.2	Erdr1

	name	ref_id
1:	hemoglobin alpha, adult chain 1	NM_008218
2:	hemoglobin alpha, adult chain 2	NM_001083955
3:	hemoglobin, beta adult s chain	NM_001201391
4:	hemoglobin, beta adult t chain	NM_008220
5:	predicted gene 10719 [Source:MGI Symbol;Acc:MGI:3641690] ENSMUST00000099047	
6:	erythroid differentiation regulator 1	NM_133362

	db.symbol	db.name	chr	start	stop
1:	<NA>	<NA>	chr11	32296489	32297310
2:	<NA>	<NA>	chr11	32283511	32284493
3:	<NA>	<NA>	chr7	103826523	103828096
4:	<NA>	<NA>	chr7	103812524	103813996
5:	<NA>	<NA>	chr9	3017408	3021593
6:	Erdrl erythroid differentiation regulator 1	chrY	90784610	90816465	

	nProbes	locustype	category	Sscore	p.values.1	p.values.2
1:	10	Multiple_Complex	main	-7.575980	1.787459e-14	3.574918e-14
2:	10	Multiple_Complex	main	-7.470252	3.996803e-14	7.993606e-14
3:	10	Multiple_Complex	main	-6.542465	3.025646e-11	6.051293e-11
4:	10	Coding	main	-6.271030	1.793340e-10	3.586680e-10
5:	10	Coding	main	-6.011108	9.212997e-10	1.842599e-09
6:	10	Coding	main	-5.902614	1.788936e-09	3.577871e-09

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 additional steps for producing multiple test corrected lists of differentially expressed genes (DEGs) for a given study are covered in the next section of the vignette.

7 Using GCS-Scores to produce DEGs in experimental datasets

For a more biologically relevant situation, the next example will cover how to use the GC-Sscore package to produce DEG lists with multiple test corrections, using the Significance Analysis of Microarrays (SAM) method. The data used in the next example is easily obtained from the GEO database. Here, we will investigate a dataset of ClariomS mouse arrays taken from GSE103380. Further information regarding the experimental design can be found at: <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE103380>.

This section of the vignette requires the following additional packages from the 'suggests' section of the package DESCRIPTION file: *siggenes*, *GEOquery*, and *R.utils*.

Begin by creating a temporary directory to store the downloaded files. For clarity, the directory will have the name of the GEO identifier.


```
> GEO <- "GSE103380"
> dir.geo <- paste(tempdir(),GEO,sep="/")
> dir.create(dir.geo, showWarnings = FALSE)
```

For this example, select files will be downloaded into the temporary directory. Here, these files are listed by the GSM ids for individual .CEL files:

- GSM2769665 (Naïve Microglia bio replicate 1)
- GSM2769666 (Naïve Microglia bio replicate 2)
- GSM2769667 (Naïve Microglia bio replicate 3)
- GSM2769668 (Naïve Microglia bio replicate 4)
- GSM2769669 (Day4 microglia bio replicate 1)
- GSM2769670 (Day4 microglia bio replicate 6)
- GSM2769671 (Day4 microglia bio replicate 7)
- GSM2769672 (Day4 microglia bio replicate 8)

The compressed CEL files are downloaded directly from GEO using the *GEOquery* package.

```
> list.cels <- c("GSM2769665","GSM2769666","GSM2769667","GSM2769668",
+               "GSM2769669","GSM2769670","GSM2769671","GSM2769672")
> # create function for pulling down the compressed .CEL files:
> cels.get <- function(x)
+   GEOquery::getGEOSuppFiles(GEO = x,
+                               makeDirectory = FALSE,
+                               baseDir = dir.geo,
+                               filter_regex = "*.CEL.gz")
> # Download the data files from GEO, into the dir.geo temp directory:
> lapply(list.cels,cels.get)
> # unzip all of the .CEL files (options set to prevent warning / errors):
> files.geo <- paste(dir.geo,list.files(path=dir.geo,
+                                       pattern = ".gz"),sep="/")
> # create function to gunzip the compressed data files:
> fun.gunzip <- function(x)
+   R.utils::gunzip(filename = x,
+                     overwrite=TRUE,
+                     remove=FALSE)
> # apply the gunzip function across the vector of compressed CEL files:
```

```

> lapply(files.geo,fun.gunzip)
> # get the path to example CSV file in the package directory:
> celtab_path <- system.file("extdata",
+                             "GSE103380_batch.csv",
+                             package = "GCSscore")
> # read in the .CSV file with fread():
> celtab <- data.table::fread(celtab_path)
> # adds path to celFile names in batch input:
> # NOTE: this is not necessary if the .CEL files
> #       are in the working directory:
> celtab$CelFile1 <- celtab[,paste(dir.geo,CelFile1,sep="/")]
> celtab$CelFile2 <- celtab[,paste(dir.geo,CelFile2,sep="/")]
> # run GCSscore using using all info from the batch file:
> GCSs.GSE103380 <- GCSscore(celTable = celtab, celTab.names = TRUE)
> # convert GCS-score output from 'ExpressionSet' to 'data.table':
> GCSs.GSE103380.dt <-
+ data.table::as.data.table(cbind(GCSs.GSE103380@featureData@data,
+                                 GCSs.GSE103380@assayData[["exprs"]]))
> # Create 4 averages of each experimental sample with each control:
> GCSs.GSE103380.dt[, day4_1_vs_naive :=
+                   rowMeans(GCSs.GSE103380.dt[
+                   ,day4_1_vs_naive_1:day4_1_vs_naive_4])]
> GCSs.GSE103380.dt[, day4_2_vs_naive :=
+                   rowMeans(GCSs.GSE103380.dt[
+                   ,day4_2_vs_naive_1:day4_1_vs_naive_4])]
> GCSs.GSE103380.dt[, day4_3_vs_naive :=
+                   rowMeans(GCSs.GSE103380.dt[
+                   ,day4_3_vs_naive_1:day4_1_vs_naive_4])]
> GCSs.GSE103380.dt[, day4_4_vs_naive :=
+                   rowMeans(GCSs.GSE103380.dt[
+                   ,day4_4_vs_naive_1:day4_1_vs_naive_4])]
> # Remove TCids without a clear symbol/name:
> GCSs.GSE103380.dt <- GCSs.GSE103380.dt[!is.na(symbol)]

> # Run SAM on 4 "averaged" Sscore columns:
> # Set the SAM 'gene.names' to be either ('TCid' or 'symbol'):
> GCSs.GSE103380.dt.SAM <-
+   siggenes::sam(GCSs.GSE103380.dt[
+   ,day4_1_vs_naive:day4_4_vs_naive],
+   cl = rep(1,4),rand=123,
+   gene.names = GCSs.GSE103380.dt$symbol)

```

We're doing 16 complete permutations

```
> # View details of SAM analysis:
> GCSs.GSE103380.dt.SAM
```

SAM Analysis for the One-Class Case

	Delta	p0	False	Called	FDR
1	0.1	0.12	8092.688	18995	0.05119
2	8.4	0.12	201.375	1673	0.01446
3	16.6	0.12	51.750	502	0.01239
4	24.9	0.12	13.000	205	0.00762
5	33.2	0.12	4.625	74	0.00751
6	41.4	0.12	1.688	27	0.00751
7	49.7	0.12	0.312	5	0.00751
8	58.0	0.12	0.125	2	0.00751
9	66.2	0.12	0.062	1	0.00751
10	74.5	0.12	0.062	1	0.00751

```
> # create name and path of SAM results:
> sam.path <- paste(dir.geo,
+                   "GSE103380_ex_SAM_16_6.csv",
+                   sep = "/")
> # Save TCids with delta >= 16.6:
> siggenes::sam2excel(GCSs.GSE103380.dt.SAM,
+                     delta = 16.6,
+                     file = sam.path)
```

Output is stored in /tmp/RtmpdIsjFC/GSE103380/GSE103380_ex_SAM_16_6.csv

```
> # Read the SAM output back into R:
> sam.results <- data.table::fread(sam.path)
> # View the top 10 DEGs output by SAM:
> head(sam.results,10)
```

	Row	d.value	stdev	rawp	q.value	R.fold	Name
1:	20111	85.7	0.00485	5.63e-06	0.015	NA	Clic1
2:	7609	69.7	0.03658	1.13e-05	0.015	NA	Apobec1
3:	6270	65.1	0.03884	1.69e-05	0.015	NA	Naaa
4:	1086	63.4	0.06162	2.25e-05	0.015	NA	AI607873
5:	926	60.4	0.01498	2.81e-05	0.015	NA	Rnpep
6:	14938	57.0	0.04817	3.38e-05	0.015	NA	Tmem106a
7:	4500	56.4	0.05877	3.94e-05	0.015	NA	Stmn1
8:	14989	56.2	0.03549	4.50e-05	0.015	NA	Milr1
9:	11251	55.7	0.03863	5.07e-05	0.015	NA	Rps271
10:	9699	54.6	0.08193	5.63e-05	0.015	NA	Mki67

The list of DEG gene symbols produced in the SAM analysis can be used for functional enrichment, using tools such as ToppFun. Likewise, the list of DEG transcription cluster ids can be input directly into software, such as Ingenuity Pathway Analysis, to get information on pathway enrichment. Further exploration of these results will be investigated in an accompanying publication.

8 Version history

1.2.0 second public release (BioC 3.11)

1.0.0 first public release (BioC 3.10)

0.99.1 initial development version

9 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. 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 $SDT = 4 * RawQ * SF$). 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. SSscore: 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.