Description of S-Score: Expression Analysis of Affymetrix GeneChips from Probe-Level Data

Richard E. Kennedy, Kellie J. Archer,
Robnet T. Kerns, and Michael F. Miles

October 17, 2016

Contents

1 Introduction

The S-Score algorithm described by **?** and **?** is a novel comparative method for gene expression data analysis that performs tests of hypotheses directly from probe level data. It is based on a new error model in which the detected signal is assumed to be proportional to the probe pair signal for highly expressed genes, but assumed to approach a background level (rather than 0) for genes with low levels of expression. This model is used to calculate relative change in probe pair 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). Assuming no expression differences between chips, the S-Score follows a standard normal distribution. Thus, p-values can be easily calculated from the S-Score, and a separate step estimating the probe set expression summary values is not needed. Furthermore, in comparisons of dilution and spike-in microarray datasets, the S-Score demonstrated greater sensitivity than many existing methods, without sacrificing specificity (**?**). The *sscore* package (**?**) implements the S-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 release has minor changes for compatibility with the `ExpressionSet` data class, as well as minor bug fixes.

3 Reading in data and generating S-Scores

Affymetrix data are generated from GeneChips® 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 S-Score algorithm compares two GeneChips by combining all of the probe intensities from a probeset (typically 11 to 20) into a single summary statistic for each gene. The *sscore* package processes the data obtained from *.CEL files, which must be loaded into R prior to calling the `SScore` function. Thus, the typical sequence of steps to accomplish this is as follows:

1. Create a directory containing all *.CEL files relevant to the planned analysis.
2. If using Linux / Unix, start R in that directory.
3. If using the Rgui for Microsoft Windows, make sure your working directory contains the *.CEL files (use "File -> Change Dir" menu item).

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

4. Load the library.

```
> library(sscore)
> options(width=60)
> library(affydata)
```

5. Read in the data and create an expression set.

Both of the functions `SScore` and `SScoreBatch` operate on an `AffyBatch` object containing all of the relevant information from the *.CEL files. Additional information regarding the `ReadAffy` function and detailed description of the structure of *.CEL files can be found in the *affy* vignette. Note that, even though the intensities have been loaded into R, `SScore` will still need direct access to the *.CEL files later to obtain the information about outliers. **If a copy of the *.CEL files is not available when `SScore` is called, an error may result.**

The `SScore` and `SScoreBatch` functions return an object of class `ExpressionSet`. (The class `ExpressionSet` is described in the *Biobase* vignette.) The S-Score values are returned in the `exprs` slot. The following examples illustrate the *sscore* package with the results of the S-Score analysis for the `Dilution` data set included with the *affydata* package. Due to the nature of this dataset, *.CEL files are not included and computation fo the SF and SDT data (as described below) cannot be performed.

A basic S-Score analysis is generated using the `SScore` function:

```
> data(Dilution)  ## get the example data
> ## get the path to the package directory
> pathname <- system.file("doc",package="sscore")
> cel <- Dilution[,c(1,3)]
> ## only need the first 2 samples per condition
> SScore.basic <- SScore(cel,celfile.path=pathname,
+ SF=c(4.46,5.72),SDT=c(57.241,63.581),rm.extra=FALSE)
```

and the first few S-Score values are

```
> exprs(SScore.basic)[1:20]
```

```
 [1] -0.456881232  1.201785056  1.167657431 -0.644052314
 [5] -0.918102279 -0.151458708  0.004924887  1.397271888
 [9] -2.135317520 -0.621832098 -0.142922922  0.771029168
[13]  0.743589518  1.680697421  0.591378144 -1.488923283
[17] -1.723861195 -1.064468884  0.323086069 -0.921048258
```

Optional parameters for `SScore` include:

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

celfile.names – character vector giving the filenames of the *.CEL files corresponding to the columns of the `AffyBatch` object. If filenames are not specified, the sample names of the `AffyBatch` object are used.

SF, SDT – the Scale Factor and Standard Difference Threshold. Each is a vector with length equal to the number of columns in the AffyBatch object, and contains a numeric value for each chip. The Scale Factor is used to scale each intensity to a target background value, with the default of 500 (as used by the Affymetrix GeneChip Operating Software [GCOS]). The Standard Difference Threshold is used as an estimate of background noise, and is equal to the standard deviation for the lowest 2% of intensities on a chip. These values are available from the Affymetrix GCOS output, or may be calculated by the `SScore` function.

rm.outliers, rm.mask, rm.extra – These are logical values used to exclude certain probes from the S-Score calculations. These options perform the same as they do in the `ReadAffy` function, which it calls. `rm.outliers` excludes all probes designated as outliers in the *.CEL file. `rm.mask` excludes all probes designated as masked in the *.CEL file. `rm.extra` removes both outlier and mask probes, and overrides `rm.outliers` and `rm.mask` if these are specified.

digits – a numeric value that specifies the number of significant decimal places for the S-Score and CorrDiff values, which are rounded as needed. The default uses full precision with no rounding. The output from the stand-alone version of the S-Score uses `digits=3`.

verbose – a logical value indicating whether additional information on the analyses is printed. This includes the chip type, sample names, values of alpha and gamma, and the SF and SDT values.

4 Multichip comparisons

Beginning with release 1.7.0, the `SScore` function is capable of comparing two classes where each class includes replicates. As with previous releases, only two class comparisons are available. The multichip comparisons are performed by adding a `classlabel` vector which distinguishes classes, similar to that of the *multtest* package. The vector `classlabel` describes to which class each GeneChip belongs. Its length is equal to the number of chips being compared, with each element containing either a 0 or a 1, indicating class assignment. Thus, the assignment

```
> labels <- c(0,0,0,1,1,1)
```

would compare the first three chips to the last three chips. (Note that the number of chips in the two classes being compared do not have to be equal.) If the `classlabel`

parameter is not specified, it defaults to a two-chip comparison for compatibility with previous versions of `SScore`.

An example of a multichip S-Score comparison would be

```
> data(Dilution)
> pathname <- system.file("doc",package="sscore")
> cel <- Dilution
> SScore.multi <- SScore(cel,classlabel=c(0,0,1,1),
+ SF=c(4.46,6.32,5.72,9.22),SDT=c(57.241,53.995,63.581,
+ 69.636),celfile.path=pathname,rm.extra=FALSE)
```

and the first few S-Score values are

```
> exprs(SScore.multi)[1:20]
```

```
 [1]  0.6243332  1.7195570  0.9667666 -0.2993475 -0.9249448
 [6]  1.5786197 -0.1658512  2.1431512 -2.7207101 -0.4010540
[11]  0.5236953  0.6283698  1.1199451  1.3237946  0.7159979
[16] -1.7620347 -1.7987401 -0.3079212  0.5429998 -0.4086409
```

The other parameters of `SScore` remain unchanged. The output data from the multichip comparison are still standard S-Scores, i.e., they still follow a Normal(0,1) distribution and may be converted to p-values as described below.

5 Multiple pairwise comparisons

Previous versions of the `SScore` function calculated the S-Score values for one pair of chips (i.e. a single two-chip comparison). However, for many experiments, several chips need to be compared. This can be done using the `SScoreBatch` function, which automates the process of making several two-chip comparisons. The setup and options for `SScoreBatch` are very similar to `SScore`.

The `SScoreBatch` function has an additional parameter, the `compare` matrix, which specifies the pairs of chips to compare. It is an N x 2 matrix, where N is the number of comparisons being made. Each row contains the column number of the chips in the `AffyBatch` object that are being compared. For example, if the `compare` matrix is set up as

```
      [1,]  [2,]
[,1]     2     5
[,2]     2     6
[,3]     5     9
[,4]    10     2
[,5]     5     7
```

```
[,6]    10     8
[,7]     9     4
[,8]     1     2
[,9]     3    10
```

The first comparison made is between the chips in columns 2 and 5 of the `AffyBatch` object; the second comparison made is between the chips in columns 2 and 6; the third comparison made is between the chips in columns 5 and 9; and so forth. If the `compare` matrix has more than two columns, only the first two columns will be used for identifying the GeneChips in the `AffyBatch` object to be compared.

Each column of `eset` will contain the results of a single two-chip comparison. The first column of `eset` will contain the comparison corresponding to the first row of the `compare` matrix, the second column of `eset` will contain the comparison corresponding to the second row of the `compare` matrix, and so forth.

A basic S-Score analysis using `SScoreBatch` is generated using the commands:

```
> data(Dilution)
> pathname <- system.file("doc",package="sscore")
> compare <- matrix(c(1,2,1,3,1,4),ncol=2,byrow=TRUE)
> SScoreBatch.basic <- SScoreBatch(Dilution,compare=compare,
+ SF=c(4.46,6.32,5.72,9.22),SDT=c(57.241,53.995,63.58,169.636),
+ celfile.path=pathname,rm.extra=FALSE)
```

and the first few S-Score values are

```
> exprs(SScoreBatch.basic)[1:10,]
```

```
          Chip 1 vs 2  Chip 1 vs 3  Chip 1 vs 4
1000_at    0.02470422 -0.457308566  2.069884875
1001_at   -0.35329047  1.201978543  1.029368680
1002_f_at  0.93159010  1.167838856  1.037607850
1003_s_at  0.11736867 -0.644545046  0.009928982
1004_at    0.68014346 -0.918701637  0.088110534
1005_at   -2.08584651 -0.151773936  0.883104329
1006_at    0.54733837  0.004665505  0.333656840
1007_s_at  0.16752329  1.397538814  2.106257809
1008_f_at  0.70457816 -2.136371809 -1.456303050
1009_at    0.60503351 -0.622318234  0.786350475
```

Other parameters for `SScoreBatch` are identical to `SScore`.

6 Using S-Scores in gene expression analysis

Under conditions of no differential expression, the S-Scores follow 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 S-Scores can be set to achieve the desired level of significance. As an example, an absolute S-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:

```
> sscores <- exprs(SScore.basic) ## extract the S-Score values
> ## find those greater than 3 SD
> signif <- geneNames(Dilution)[abs(sscores) >= 3]
```

Similarly, the p-values can be calculated as:

```
> sscores <- exprs(SScore.basic) ## extract the S-Score values
> p.values.1 <- 1 - pnorm(abs(sscores)) ## find the corresponding
>                         ## one-sided p-values
> p.values.2 <- 2*(1 - pnorm(abs(sscores))) ## find the corresponding
>                         ## two-sided p-values
```

The S-Score algorithm does account for the correlations among probes within a two-chip comparison. However, it does not adjust p-values for multiple comparisons when comparing more than one pair of chips.

7 Computing scale factor and statistical difference threshold

The `SScore` and `SScoreBatch` functions call the function `computeSFandSDT` to compute the values for the Scale Factor (SF) and Statistical Difference Threshold (SDT) if these are not supplied by the user. `computeSFandSDT` is an internal function that generally will not be called or modified.

The calculations for the SF and SDT are performed as described in the Affymetrix Statistical Algorithms Description Document (**?**) and implemented in the Affymetrix (using SDT = 4 * RawQ * SF). The calculation of these values can be both time- and memory-intensive; it is recommended that the user supply these values from the Affymetrix MAS5 or GCOS Metrics table whenever possible. Alternatively, `computeSFandSDT` may be called directly to obtain the SF and SDT values for each *.CEL file, which are then supplied by the user in subsequent calls to `SScore`. The calculations for each *.CEL file are independent. If memory is not sufficient to allow computation of all SF and SDT

values simultaneously, the *.CEL files may be broken into smaller batches; identical results will be obtained either way.

In addition to computing the specified values, `computeSFandSDT` may be used to generate histograms of the log intensities for the chips being compared. Such plots are useful for identifying potentially problematic chips prior to analysis. It may also be used to display additional information about the *.CEL file parameters. The options for `computeSFandSDT` are

TGT – a numeric value for the target intensity to which the arrays should be scaled.

verbose – a logical value indicating whether additional information on the calculations is printed. This includes the SF, SDT, and RawQ values, as well as descriptive statistics on the background and noise. This is similar to the information provided by the Affymetrix GCOS Metrics table for the *.CEL file.

plot.histogram – a logical value indicating whether a histogram should be plotted. Both the PM and MM log intensities will be shown in a single graphics window. Separate plots will be generated for each chip being analyzed.

digits – a numeric value that specifies the number of significant decimal places for the SF and SDT values, which are rounded as needed. Using `digits=3` rounds to the same number of digits as the stand-alone version of the S-Score.

celfile.path – character string specifying the directory for *.CEL files

`computeSFandSDT` requires that the *.CEL files be in text format. The alternate function `computeAffxSFandSDT` expects information obtained from the *affxparser* routines, so that either text or binary files may be used. In addition to the options for `computeSFandSDT`, `computeAffxSFandSDT` has the following required parameters:

stdvs – a vector of standard deviations of the probe intensities (which can be read using the `readStdvs=TRUE` option in the *affxparser* function `readCel`).

pixels – a vector of the number of pixels used in calculating the probe intensity (which can be read using the `readStdvs=TRUE` option in the *affxparser* function `readCel`).

8 Identifying outliers

The current version of the `SScore` and `SScoreBatch` functions use the information contained in the *.CEL files to flag probes as outliers that should be excluded from the S-Score calculation. In previous versions, this was accomplished using the `computeOutlier` function, which is retained for compatibility. This is an internal function that generally will not be called or modified. The `computeOutlier` function was called if the `rm.outliers`, `rm.mask`, or `rm.extra` parameters of `SScore` or `SScoreBatch` are set to

`TRUE`. These parameters work as described in the *affy* documentation since they are passed to the `ReadAffy` function to identify outlier and mask probes. The return value from `computeOutlier` is a logical matrix the same size and order as the intensity matrix for the `AffyBatch` object. Each cell of the logical matrix contains a `TRUE` value if the corresponding intensity is identified as an outlier and excluded from the S-Score calculation; otherwise it contains `FALSE`.

9 Changes from the stand-alone version

The S-Score algorithm has been previously implemented as a stand-alone executable for the Windows operating system, using Borland Delphi. This version has been available from the Miles Laboratory at `http://www.brainchip.vcu.edu/expressionda.htm`. Users of the stand-alone version will notice small differences in results compared to the *sscore* package as it is implemented in R, though these should not significantly affect inferences regarding gene expression. The following lists identifies differences between the two implementations:

1. The stand-alone version excludes outlier, masked, and modified intensities from calculations when using *.CEL files. When using *.CSV files, the stand-alone program also excludes outlier, masked, and modified intensities *if the corresponding *.CEL file is present for obtaining this information.* (The *.CSV file does not contain any information about which intensities are outlier, masked, or modified.) The default for the R package is *not* to exclude outlier, masked, or modified intensities, though this may be changed using various options. Note that, due to the way the *affy* package is implemented, it is not possible to exclude modified intensities using the *sscore* package.

2. The rounding methods are not identical for Borland Delphi and R, which can lead to slight differences in calculations. The difference is negligible for most of the S-Score calculations, and should be less than or equal to 0.001.

3. The SF and SDT calculations in the stand-alone version are performed using an independently developed algorithm. The original C++ version uses natural logarithms, while the Delphi version uses base 10 logarithm. The *sscore* package uses a ported version of the Affymetrix algorithms described on the Affymetrix website `http://www.affymetrix.com` under Support -> Developer's Network -> Open Source -> MAS5 Stat SDK. Base 2 logarithms are used for these calculations.

A Java version of the S-Score algorithm is also under development. Differences between the Java version and the *sscore* package will be included after the Java version is released.

10 Version history

1.7.0 added routines to compute S-Scores for replicate chips within a 2-class comparison. Also updated functions to operate on the new `ExpressionSet` class, and changed routines for reading of binary *.CEL files from *affxparser* to *affyio* due to stability problems with the former on the Macintosh PowerPC platform.

1.5.4 incorporated routines from the *affxparser* package for reading of binary *.CEL files. Added option to specify *.CEL file names in the `SScore` and `SScoreBatch` functions.

1.4.2 corrected a bug resulting in too many open file handles for large `AffyBatch` objects.

1.4.1 corrected a bug in assigning column names to `exprSet` object

1.4.0 first public release

1.0.0 initial development version

11 Acknowledgements

The development of the 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. This work was partly supported by NLM F37 training grant LM008728 to Richard E. Kennedy and NIAAA research grant AA13678 to Michael F. Miles.