SAGA Vignette

Steven R. Talbot

2017-11-27

About SAGA package

In our paper **"Surrogate Assay for Genotoxicity Assessment (SAGA) of integrating viral vectors for gene therapy"** we report an improved in vitro test to determine the risk of insertional mutagenesis of integrating vectors for gene therapy. SAGA builds on the well accepted cell culture protocol of the in vitro immortalization assay (IVIM) but screens for the deregulation of oncogenic gene expression signatures. We demonstrate a new bioinformatic apporach to correctly classify the mutagenic potential of retroviral vectors used in previous and current clinical trials.

This vignette covers the basic functions for SAGA analysis in R.

Package Requirements

In order to use the SAGA package some Bioconductor dependencies need to be installed. First, the connection to the Bioconductor data base must be established. Then the required packages can be loaded independently. Please note that these packages are only available over Bioconductor and cannot be loaded using CRAN.

```
source("https://bioconductor.org/biocLite.R")
biocLite("Biobase")
biocLite("BiocGenerics")
biocLite("parallel")
```

From CRAN the following packages need to be loaded as well. The following links can be used directly in the R console.

```
install.packages("limma")
install.packages("sva")
install.packages("e1071")
install.packages("phenoTest")
install.packages("gridExtra")
install.packages("bapred")
```

Introduction

`SAGA` analysis starts with raw data obtained from regular gene expression microarray analysis acquired by Agilent platforms. This is important since the functions in this package will only work within the Agilent format. In the current version other platforms were not specifically included into the capacities of this package. If you wish to use other platforms you'll have to adapt the functions to your specific needs.

As described in the paper, we optimized a training set of arrays to discriminate between potentially dangerous arrays and MOCK data in order to obtain a measurement for vector safety. Vector safety is therefore described as either *transforming* or *nontransforming* in the package.

The basic SAGA classifier was obtained by varius Machine Learning algorithms as well as feature assessment techniques as described in our paper. In essence, a toplist of 12 probes were obtained that are able to predict untrained samples reliably well. The `SAGA` package uses its default data set and the probe toplist to predict user sample data.

Further, the user may specify his/her own positive and negative controls for more confidence into the prediction results.

Basic requirements

In order to use the `SAGA` package the user has to create a specific folder with sample data. The samples must follow the Agilent design and must be in *.txt format. Sample names should be numeric.

Possible sample names would be: 9910.txt, 9911.txt, 9912.txt etc.

Further, a **'SampleInformation.txt'** or targets file is needed. This file consists of a table with certain columns which are later used to merge the user data with the default data. In the shown plots user data will always be black colored for identification purposes.

It is important to know, that the targets file **has** to have the following name: **SampleInformation.txt** and also **has** to be located in the same folder as the samples.

The user may change this file manually to adjust his batch, group, vector or class settings.

`SAGA` offers a function to generate a blank SampleInformation.txt file automatically. However, this only works if there is no such file in the sample directory. Otherwise the already provided file will be used. The generated targets file may then be adjusted manually. Make sure not to hamper with the column names. Column names are case sensitive.

Data structure

Here is a short example of a minimal targets file in txt-format. The first column must contain the array names. You may specify the names by yourself or let your microarray platform choose the labelling. However, it is best that the names do not contain any special characters such as *"-"* and that they are numeric (and in an ascending order).

Make sure that the targets file has the following format.

Example of a SampleInformation.txt file.

SAMPLE_ID	Filename	Batch	Group	Vector	Class
X5741_1	5741_1.txt	1	1	MOCK	nontransforming
X5746_1	5746_1.txt	1	8	NA	NA
X5747_1	5747_1.txt	2	1	MOCK	nontransforming
X9910_1	9910_1.txt	2	6	SRSEFS	NA
X9914_1	9914_1.txt	1	5	NA	NA
X9917_1	9917_1.txt	2	5	SRSEFS	NA
...	...	...	...	...	...

In the table, the first array (X5741_1) is labelled as *nontransforming* in the **Class** column. This labelling is useful only for the `saga_optmodel` function. Using either **nontransforming** or **transforming** as class labels, the user may specify his own positive or negative controls. When left empty (NA) the arrays will be classified against the default model.

We recommend that users includes at least 2 to 3 MOCK (negative) and RV.SF (positive) controls their own models. The batch correction algorithm may need this for proper integration of the provided data into the SAGA data set.

IMPORTANT for GESEA: For the Gene Set Enrichment Analysis (GESEA) the user must provide at least 3 samples in two different batches. One of the three samples must be a MOCK control (Vector) and labelled as 1 in the Group column. Every MOCK control needs the Group label 1. All other arrays may have other numeric labels.

Using the functions of the SAGA package

The following section explains the functions of this package and how to use them in SAGA analysis.

saga_gentargets()

After pooling the samples in a specific folder (`samplepath`), a targets file called `SampleInformation.txt` must be provided for analysis. Without this file SAGA analysis will fail. Since its construction is a bit tedious we provide a function for automatic generation of an empty file. The user will then have to adjust this file manually to his experimental setup.

The only two requirements for this function are:

- there must be at least one sample file in the folder (however, there should really be more (controls!))
- there should not be a `SampleInformation.txt` in the folder when samples are new

```
# This function will generate the empty user targets file for the samples.
saga_gentargets(samplepath)
```

saga_wrapper()

Using this function enables the user to perform SAGA analysis in one single step. Only the path to the samples and the SampleInformation.txt file must be provided in the folder. Further, the option `doGESEA` (defaul setting is 0) gives control over GESEA analysis. In order to perform GESEA analysis, the `SampleInformation.txt` file has certain requirements which are shown above. One disadvantage of this wrapper function is that the user will loose some flexibility.

The results of GESEA will be generated as separate PDF and Results textfiles in the sample folder.

```
# This function performs SAGA analysis in one step.
mySAGAres <- saga_wrapper(samplepath, doGESEA=0)
```

saga_import()

The `saga_import` function imports multiple single microarray data files (*.txt format). The user must specify a sample path in the `samplepath` object to guide the function to the query data. However, the folder should not contain any other files except the `SampleInformation.txt` file wich can be generated with the `saga_gentargets` function.

The argument `showjoint` can be used for quick data visualization. A boxplot will show the joint data set after appending the user samples to the SAGA data set (default value for `showjoint` is 0).

```
# This will load the default data.
saga_import(smplpath, showjoint=0)
```

Please note: `saga_import` relies on Agilent raw data in *.txt format. There is no need to process the readout files prior to SAGA analysis.

saga_norm()

We use quantile normalization as a preprocessing step piror to analysis. The sample arrays are always normalized together with the SAGA data set. This means that training and testing data are merged during the normalization process and will again be separated later for further analysis. This ensures that data are haromonized and no artificial normalization bias enters the analysis. The `normplot` option also gives control over a boxplot showing the normalized values of the joint data set (the default argument is 0).

In case of multiple probes per array the intensities are averaged to a single value. Agilent readouts usually offer four technical probe replicates and a variety of controls. In the end, 39428 normalized and averaged probes will remain and internal controls are filtered.

```
# This function applies quantile normalization on the whole data set.
saga_norm(matrix.joint, normplot=0)
```

saga_batch()

Usually microarrays are done in batches. A plethora of external influences may (occasionally) distort otherwise equal arrays beyond recognizability. This often leads to false conclusions when not handled correctly, e.g., wrong clustering results.

In order to avoid such errors we apply batch correction on the joint data set. The user may define his own batches within the `SampleInformation.txt` file and is required to update the batch information **manually**.

The function requires the full joint targets matrix (pdata.joint) as well as the normalized and averaged data matrix from `saga_norm` (eset.rma).

```
# This function will do the batch correction.
saga_batch(pData.joint, eset.rma)
```

saga_sampling()

This function will take care of the sample preparation of the joint SAGA data set. It will take the batch corrected data from `saga_batch` and will use the top 12 list of SAGA classifier probes to further filter the whole data set. Then, the joint data set will be separated into a training (SAGA data) and a testing set (user samples).

The `showPCA` object gives control over a Principal Components Analysis (PCA) plot showing the first principal components of the joint data set. Also, the axes show the percentage of explained variance (in %) for the plotted principal components.

Red dots are potentially dangerous (transforming) samples and the rest are coding for other SAGA assays that are nontransforming.

```
# This function will do the sampling as a preparation for later classification.
saga_sampling(eset.batch, pData.joint, pData.user, filelist, pData, showPCA=0)
```

The function needs the batch corrected data (`eset.batch`), the joint targets file (`pData.joint`), the user targets file (`pData.user`), the file list (`filelist`) of user samples and the SAGA phenotype/targets data (`pData`). The `showPCA` argument will give control over the PCA plot (default object value is 0).

The function output offers a training matrix, labels for the training matrix as well as a matrix with the unknown sample data. These matrices are required for the classification and prediction step of the `saga_predict` function.

Please note: The PCA plot is not directly qualified for classification decisions. Although it will give a qualitative overview on the global "positions" of the sample data within the SAGA data context, spatial neighborhood in PCA is not a reliable classifier. There are better ways of assessing classification success than optical similarity in a PCA plot. However, rather often, ambigous data will lie in the zone between transforming and nontransforming arrays.

saga_predict()

For vector safety prediction a Support Vector Machine (SVM) with "radial" kernel is used for building the model. We optimized the cost and gamma factors using the SAGA data and generalize these settings for new and untrained samples.

In a second step, the processed user sample data are predicted using the SVM model. The output shows each sample array along with its SVM probability and a label with **transforming/nontransforming** information - whereas transforming indicates potential danger and nontransforming potential safety of the sample. The probability value can be used as a quality indicator on how clearly the decicion of the SVM was for each sample (the cutoff value is 50%, or >0.5 = transforming).

```
# This function will take care of the model building and prediction steps.
saga_predict(matrix.train, labels.train, matrix.unknown)
```

saga_optmodel()

This SAGA function lets the user specify his/her own positive or negative controls. When performing SAGA predictions, this may build additional confidence in trusting the prediction results. In order to specify arrays as transforming or nontransforming, the user has to modify the `SampleInformation.txt` file manually. Under the column `Class` the user may add either *"transforming"* or *"nontransforming"* for each array. Please be aware that this is case sensitive. When this parameter is not set (the column stays empy (NA)), all arrays will be used in the prediction step. The specified arrays, however, will be appended to the SAGA data set and are used in the model building process. Thus they will be excluded from the prediction output. Additionally, MOCK samples should always have the Group label = 1.

Further, the SVM model is optimized by an extended grid search and tuned for optimal gamma and cost values We use 10-fold cross validation in five repetitions together with a radial kernel to find the best parameters for the underlying data. So, eventually, if the user adds more data, the optimal parameters may change and adapt to the individual experimental setup.

```
# Uses grid optimization and control data for building a more user spcific and optimized model
for predictions.
saga_optmodel(pData.joint, eset.rma, eset.batch, showsummary=1)
```

The function will output a table with prediction results for each sample array as well as the probabilities from SVM classification/prediction. Again, these numbers may be used as an indicator for how close an array was predicted to either class. Ambigous arrays will have probabilities around 50% or 0.5.

Example Analysis & Array Prediction

This example assumes that 6 arrays (5741, 5746, 5747, 9910, 9914, 9917) are placed in a sample folder. The table above shows the corresponding `SampleInformation.txt`.

```
library(saga)

samplepath     <- "C:/somewhere useful/"

# saga_gentargets; automatically generates an empty (!) sample information file.
```

```
# Modify manually according to your experimental requirements
targets        <- saga_gentargets(samplepath)

################################################################################
### Wrapper function - all in one
################################################################################
mySAGAres      <- saga_wrapper(samplepath, doGESEA=1)

################################################################################
### Or use these single functions for SAGA analysis
################################################################################
# saga_import
rawdata        <- saga_import(samplepath, showjoint=1)
eset.user      <- rawdata$eset.user
pData.joint    <- rawdata$pData.joint
pData.user     <- rawdata$pData.user
pData          <- rawdata$pData
matrix.user    <- rawdata$matrix.user
SIF            <- rawdata$SIF
SAGA_RAW       <- rawdata$SAGA_RAW

# normalize saga data
normalized     <- saga_norm(SAGA_RAW, pData.joint, matrix.user, normplot=1)

# remove batch effects
batchnorm      <- saga_batch(rawdata$pData, normalized$matrix.SAGA, matrix.user, SIF)
matrix.SAGA    <- batchnorm$matrix.SAGA
matrix.user    <- batchnorm$matrix.user

# filtering & model building
NN             <- saga_sampling(matrix.SAGA, matrix.user, pData, pData.user, showPCA=1)
matrix.train   <- NN$matrix.train
labels.train   <- NN$labels.train
matrix.unknown <- NN$matrix.unknown
matrix.Top12   <- NN$matrix.Top12

# Array predictions with optimized SVM parameters (default settings)
predictions    <- saga_predict(matrix.train, labels.train, matrix.unknown, writeFile=1)
predictions

# Here are the prediction results...

        transforming nontransforming predicted.as
X5741_1    0.9832877      0.01671230 transforming
X5746_1    0.9790204      0.02097956 transforming
X5747_1    0.9786609      0.02133914 transforming
X9910_1    0.9828158      0.01718416 transforming
X9914_1    0.9814494      0.01855060 transforming
X9917_1    0.9837417      0.01625828 transforming


# Predictions with own neg/pos controls and grid optimization; may be used without controls
optPred        <- saga_optmodel(pData.joint, matrix.Top12, showbest=0, writeFile=1)
optPred

        transforming nontransforming predicted.as
X5746_1    0.8976787      0.10232133 transforming
X9910_1    0.9205793      0.07942067 transforming
X9914_1    0.9101531      0.08984687 transforming
X9917_1    0.9225601      0.07743992 transforming

# GESEA
gesea_results  <- saga_gesea(samplepath, eset.user, SIF)


############################
######## HELP Files ########
############################
# to see the help overview
help(package = "saga", help_type = "html")

# to see the vignette
vignette("saga_vignette")
```