Using the STROMA4 package

Sadiq MI Saleh, Michael T Hallett

May 1, 2024

1 Introduction

Triple-negative breast cancer is a molecularly heterogeneous cancer that is difficult to treat. TNBC microenvironmental (stromal) heterogeneity has not been well characterized despite the key role that it may play in tumor progression. This package assigns stroma (tumor microenvironment) and Lehmann properties to a set of samples. These properties were identified in Triple negative breast cancer patients, but the stromal properties in particular have shown applicability in other subtypes as well.

assign.properties is a function to assign stromal property and TNBCType generative property levels to a TNBC dataset. Given an expressionset with a gene expression matrix, and a a list of HGNC IDs, this function will estimate the status of samples for either the TNBCType generative properties, the stromal properties, or both at the same time. (Saleh et al. , under review at Cancer Research). While they were identified in TNBC stromal samples, these properties are applicable outside of this context.

2 Description of the package

We defined a method for estimating stromal and Lehmann generative properties in a gene expression dataset (either LCM stroma or whole tumor breast cancer data). The **assign.properties** function extracts the expression data from the expression set using the exprs() function. The HGNC ID annotations for the dataset are required as this is what the function uses to identify genes to estimate property levels. The HGNC IDs should be in the featureData of the expressionset, and the corresponding column ID should be passed to the geneID.column argument. The function can be used to estimate the stromal properties, the Lehmann generative properties or both simultaneously depending on whether 'STROMA4' (stromal properties), 'TNBCTYPE' (Lehmann generative properties), or both are passed to the genelists argument. This function will then return property assignments.

3 Analysis of sample dataset

To apply the stromal properties, simply use pass an 'ExpressionSet' to the function "assign.properties"

First we will load the STROMA4 package.

```
> if (!requireNamespace("BiocManager", quietly=TRUE))
+     install.packages("BiocManager")
> #BiocManager::install("STROMA4")
> library("STROMA4")
```

Next we load a test dataset to serve as an example. We use the gene expression dataset published by Schmidt et al. [2008] provided in the breastCancerMAINZ package.

```
> library(breastCancerMAINZ)
> data(mainz, package='breastCancerMAINZ')
```

Note that this package has HGNC ID annotations in the featureData in the 'Gene.symbol' column.

```
> head(fData(mainz)[, "Gene.symbol", drop=FALSE])
```

```
          Gene.symbol
1007_s_at        DDR1
1053_at          RFC2
117_at          HSPA6
121_at           PAX8
1255_g_at      GUCA1A
1294_at          UBA7
```

We can therefore pass this column to the function. If we did not have these annotations, we would be required to first obtain these annotations (e.g. from AnnotationDbi) and make sure that these annotations were added to the featureData of the expressionset.

3 types of analyses can then be run:

3.1 Assignment of stromal properties alone

```
> just.stromal.properties <- assign.properties(ESet=mainz, geneID.column="Gene.symbol",
+                           genelists="Stroma4", n=10, mc.cores=1)
```

3.2 Assignment of Lehmann properties alone

```
> just.lehmann.properties <- assign.properties(ESet=mainz, geneID.column="Gene.symbol",
+                           genelists="TNBCType", n=10, mc.cores=1)
```

3.3 Assignment of both stromal and Lehmann properties simultaneously

```
> all.properties <- assign.properties(ESet=mainz, geneID.column="Gene.symbol",
+                  genelists=c("Stroma4", "TNBCType"), n=10, mc.cores=1)
```

3.4 Expected result from assigning properties

The function returns a list of assignments for each property.

```
> property.cols <- grepl('properties', pData(all.properties))
> print(apply(pData(all.properties)[, property.cols], 2, table))
```

```
named integer(0)
```

The properties can then analyzed independently (e.g. associated with patient outcome, etc.).

3.5 Combining stromal properties into a subtyping scheme

The stromal properties can also be combined to generate a novel stromal subtyping scheme.

```
> property.cols <- paste0(c('T', 'B', 'D', 'E'), '.stroma.property')
> patient.subtypes <- pData(just.stromal.properties)[, property.cols]
> for(i in c('T', 'B', 'D', 'E'))
+   patient.subtypes[, paste0(i, '.stroma.property')] <-
+     paste0(i, '-', patient.subtypes[, paste0(i, '.stroma.property')])
> patient.subtypes <- apply(patient.subtypes, 1, paste, collapse='/')
> print(head(patient.subtypes))
```

```
                                          MAINZ_BC6001
                 "T-low/B-high/D-low/E-intermediate"
                                          MAINZ_BC6002
         "T-low/B-intermediate/D-low/E-intermediate"
                                          MAINZ_BC6003
                       "T-low/B-high/D-high/E-high"
                                          MAINZ_BC6004
"T-intermediate/B-high/D-intermediate/E-intermediate"
                                          MAINZ_BC6005
 "T-low/B-intermediate/D-intermediate/E-intermediate"
                                          MAINZ_BC6006
                 "T-low/B-low/D-high/E-intermediate"
```

This represents a novel method to generate subtypes based on individually assigned properties.

4 Notes:

4.1 Size of intermediate region and non-informative signature

We note that the number of samples assigned to the low, intermediate, and high classes varies between properties. The 'B.stroma.property' splits samples roughly into thirds, while the D.stroma.property has an enrichment for low and high assignments.
The number of samples assigned to the intermediate category gives us insight into the applicability of the signature to the samples. Signatures which assign a high proportion of samples to the intermediate category (e.g. $>$ 80 percent) indicates that the signature is non-informative for the dataset. All of the signatures appear to be informative for this dataset.

4.2 Confidence of intermediate region

By increasing the number or random samples (n), the confidence in the intermediate region is increased at the expense of increased computing time

4.3 Using multiple cores

2. Using multiple cores (mc.cores) on machines with multiple cores available can reduce runtime. The function detectCores() can be used to determine how many cores you have available.