SpatialPRo: methods for spatial proteomics data

Kieran Campbell
`kieran.campbell@dpag.ox.ac.uk`

August 19, 2014

Contents

1 Introduction

SpatialPRo provides classes and methods for handling single-cell spatially resolved proteomics data. It provides two classes: *SPExp* to represent an entire spatial proteomics experiment, and *SPData* to represent a single sample (for example a tissue section from a tumour). Also provided are methods for subsetting by tissue and channel measurement, as well as basic normalisation and clustering methods.

A major advantage of spatially-resolved proteomics data is the availability of neighbour data for cells. While many studies have looked at within-cell signalling pathways, it is now possible to examine dependencies between neighbouring cells to elucidate spatial signalling pathways. Therefore, particular emphasis is put on methods that provide easy access to the neighbour data for a given cell.

1.1 Basic concepts

In a spatial proteomics experiment, it is likely several *samples* are taken. For example, each sample could be a tissue biopsy from a given patient. The class *SPData* represents a particular sample, and holds the single-cell data along with the associated meta-data, such as cell location, sample ID and channel names.

Multiple samples (for example across different patients, or repeat experiments from a given patient) define the overall experiment, represented by the class *SPExp*. An instance of this class contains a list of *SPData* objects, along with the associated file names and experiment directory.

If the experiment is multiplexed, multiple *channels* may be measured simulatenously. These channels can be proteins and protein modifcations - the terms "channels" and "proteins" are used interchangeably.

2 SPExp

SpatialPRo can be loaded in the usual way, and comes with SPE, an example instance of *SPExp*:

```
library(devtools)
load_all("~/software/SpatialPRo")
data(SPE)
SPE

## An object of class SPExp
##  Location: /home/spatialpro/
##  With 5 samples
##
## 79
## 201
## 273
## 283
## 304
```

We can also view the sample IDs, original filenames and locations:

```
IDs(SPE)

## [1]  79 201 273 283 304

files(SPE)

## [1] "TMA_ID79.mat"  "TMA_ID201.mat" "TMA_ID273.mat" "TMA_ID283.mat" "TMA_ID304.mat"

getDir(SPE)

## [1] "/home/spatialpro/"
```

Furthermore, we can subset *SPExp* objects and extract the *SPData* object at any position:

```
SPE[2:3] # subset the second and third samples

## An object of class SPExp
##  Location: /home/spatialpro/
##  With 2 samples
##
## 201
## 273

SPE[[4]] # extract the SPData object in the fourth slot

## An object of class SPData
##  Sample ID: 283
##  601 cells with 32 channel(s)
```

3 SPData

The *SPData* object represents an individual sample and is in practice what is used in analysis. As before, we can extract an SPData instance from SPExp:

```
sp  <- SPE[[1]]
sp

## An object of class SPData
##  Sample ID: 79
##  432 cells with 32 channel(s)
```

3.1 Accessing data in a SPData object

From an instance of *SPData* we can extract all the information from a sample, including:

- Raw data matrix (cell-by-channel)
- Normalised data matrix (cell-by-channel)
- Cell locations (cell-by-2)
- Cell sizes
- Number of cells
- Number of channels
- Channel names
- Sample ID

```
rawData(sp)[1:2,1:4] # un-normalised data

##      ER(La139)D PR(Pr141)D pSHP2(Nd142)D p53(Nd143)D
## [1,]       3566      987.0         729.5         388
## [2,]       2521      722.5         374.5         189

cells(sp)[1:2,1:4] # normalised data

##      ER(La139)D PR(Pr141)D pSHP2(Nd142)D p53(Nd143)D
## [1,]    -1.5773    -1.0653       -0.4931     0.07711
## [2,]    -0.8937    -0.3664       -0.9754    -0.60380

head(xy(sp)) # cell locations

##        [,1]  [,2]
## [1,]  8.195  4.98
## [2,]  4.291 17.37
## [3,]  1.000 31.00
## [4,]  8.935 29.56
## [5,] 13.901 47.71
## [6,] 17.065 73.12

head(size(sp)) # cell sizes

## [1] 149  86  11 153 695 831

nCells(sp) # number of cells

## [1] 432

nChannel(sp) # number of channels

## [1] 32

head(channels(sp)) # channel names

## [1] "ER(La139)D"    "PR(Pr141)D"    "pSHP2(Nd142)D" "p53(Nd143)D"   "CD31(Nd144)D"
## [6] "Twist(Nd145)D"

ID(sp) # sample ID

## [1] 79
```

3.2 Neighbour information

We can access the neighbour data of the sample by calling

```
nn  <- neighbours(sp)
length(nn)

## [1] 432

dim(nn[[1]])

## [1]  3 32
```

which returns a list of length `nCells(sp)`, where each entry is an m-by-channel matrix for m nearest neighbours of a given cell. We can also retrieve the mean over the nearest neighbours for each cell:

```
nn.mean  <- neighbourMean(sp, useWeights=FALSE, normalise=TRUE)
dim(nn.mean)

## [1] 432  32

nn.mean[1:2,c(2,5,7)]

##       PR(Pr141)D CD31(Nd144)D CD68(Nd146)D
## [1,]    -0.5896      0.07947      -0.7611
## [2,]    -1.1073     -0.51612      -0.7198
```

If the class has boundary weight data, `useWeights=TRUE` will weight the average by the relative boundary size between the cells. If `normalise=TRUE` then each column is centred-scaled to mean 0 and standard deviation 1.

3.3 Subsetting SPData objects

An *SPData* object can be subsetted to include only a selection of cells and/or channels. Subsetting *SPData* objects is identical to subsetting the underlying matrix - if we wish to select out cells `i` and channels `j` then we use the `R` operation `[`. For example, if we want to extract the sample containing the first 3 cells with channels 5 and 7, we would call

```
sps  <- sp[1:3,c(5,7)]
sps

## An object of class SPData
##  Sample ID: 79
##  3 cells with 2 channel(s)
```

This will also subset the cell sizes, locations and channels:

```
size(sps)

## [1] 149  86  11

xy(sps)

##        [,1]  [,2]
## [1,] 8.195  4.98
## [2,] 4.291 17.37
## [3,] 1.000 31.00

channels(sps)

## [1] "CD31(Nd144)D" "CD68(Nd146)D"
```

When reducing the *SPData* object down to a subset of cells, it is often preferable to keep the nearest neighbour measurements for all cells in the subset, regardless of whether those neighbours themselves are actually in the subset. For example, consider cell 1, with neighbours 2, 3 and 4 (who each have some different set of neighbours). If we subset the *SPData* to cells 1 & 4, we don't wish to remove neighbours 2 and 3 from 1, since they are physically present in any interaction. Therefore, while the neighbour data for any cells not in the subset is removed, those cells aren't removed from the neighbour data for cells in the subset. We can see this by looking at the neighbour data for `sp` and `sps`:

```
length(neighbours(sp)) # list with an entry for each cell

## [1] 432

length(neighbours(sps))

## [1] 3

dim(neighbours(sp)[[1]]) # a neighbour-by-channel matrix for the first cell

## [1]  3 32

## neighbour-by-channel matrix for first cell in reduced set
## - only channels change
neighbours(sps)[[1]]

##      CD31(Nd144)D CD68(Nd146)D
## [1,]      0.20822     -0.52879
## [2,]      0.09225     -0.02768
## [3,]     -0.06451     -0.71192
```

3.4 Cell class operations

In many tissue samples there will be multiple cell types - for example, in tumour tissue there will be epithelial, stromal and immune cells present. In *SpatialPRo* the cell type is known as its *class*. Cell class can be set and retrieved using the `cellClass` method:

```
cellClass(sps)

## [1] 2 2 1

cellClass(sps) <- c(2,1,2)
cellClass(sps)

## [1] 2 1 2
```

Therefore the cell-by-channel matrix can easily be found for a particular class of cell:

```
cells(sps[cellClass(sps) == 1,])

##      CD31(Nd144)D CD68(Nd146)D
## [1,]       0.2082      -0.5288
```

3.4.1 Regression at a boundary

We can also extract the neighbour data for cells just belonging to one class. This is particularly useful for any modelling done at a boundary - if we want to see how cells of type 1 are affected by type 2, we would use the `neighbourClass` function to get a list of neighbouring cells only of a given class:

```
nn2 <- neighbourClass(sp,2) # nearest neighbour list containing only cells of type 2
```

We would then use the `findBoundary` function to find which cells lie along the boundary:

```
boundary.cells  <- findBoundary(sp)
```

This is only currently implemented for two types of cells. Then we would find the cells of type 1 that lie along the boundary:

```
type1.boundary <- intersect(which(cellClass(sp) == 1), boundary.cells)
```

We can then set up our boundary regression:

```
sp.boundary <- sp[type1.boundary,]
neighbours(sp.boundary) <- nn2[type1.boundary]

Y <- cells(sp.boundary)
X <- neighbourMean(sp.boundary, useWeights = FALSE, normalise = TRUE)

fit <- lm(Y ~ X)
```