Brainconductor demo

Setting up

We first install the relevant packages, `brcbase` (which contains the core code needed for this demo) and `brcdata` (which contains the example datasets).

```
library(devtools)
devtools::install_github("cdgreenidge/brcbase", ref = "kevin", subdir = "brcbase")
```

```
## Skipping install of 'brcbase' from a github remote, the SHA1 (b0d1ed54) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
devtools::install_github("linnylin92/brcdata", ref = "kevin")
```

```
## Skipping install of 'brcdata' from a github remote, the SHA1 (bd32bc5e) has not changed since last i
##   Use `force = TRUE` to force installation
```

Basics of `brcbase`

The `BrcFmri` class, through a functional MRI

Let us investigate first `COBRE_0040071_funcSeg`, the functional MRI scan of subject 0040071 in the COBRE repository. We can look up more information on this dataset using `?COBRE_0040071_funcSeg`.

```
library(brcbase)
library(brcdata)
dat <- brcdata::COBRE_0040071_funcSeg
class(dat)
```

```
## [1] "BrcFmri"
```

```
summary(dat)
```

```
## Summary of BrcFmri object
## ----------
## Id:                        COBRE_0040071
## Volume resolution:         61 x 73 x 61 voxels
## Number of parcellations: 82470 parcels
## Scan length:               5 volumes
## Estimate size:             5.22 Mb
```

We see that from the summary that `dat` represents an fMRI that has dimension $61 \times 73 \times 61$, has 82470 parcels in its parcellation, and has a scan length of 5, meaning there are 5 time indices. Let's dig into how this data is stored. We have coded an `print` function for the `BrcFmri` class to give you an overview of the contents of `dat`.

```
dat
```

```
## BrcFmri object of dimension 61 x 73 x 61
##  with 82470 parcels and 5 length
## ----------
##
## $data2d (Abridged)
##           [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,] 30.02991 174.8699 356.7232 511.1431 576.2834 685.0399 769.8184
```

```
## [2,] 27.63441 160.8071 332.9987 471.9348 510.8638 609.2991 666.3053
## [3,] 32.30598 188.1584 356.6695 494.9095 552.8895 656.2498 736.8334
## [4,] 29.51525 171.8208 340.1711 482.5676 545.7252 678.8959 761.8602
## [5,] 28.45482 165.5816 329.9427 463.3896 508.2646 623.3168 684.4691
##          [,8]     [,9]    [,10]
## [1,] 812.0490 725.7983 373.0438
## [2,] 673.5076 615.9930 319.4124
## [3,] 782.1230 738.2408 387.2846
## [4,] 781.6453 720.3726 374.6099
## [5,] 680.3885 594.5089 302.8029
##
## $id
## [1] "COBRE_0040071"
##
## $parcellation (Object of class BrcParcellation)
## $$dim3d
## [1] 61 73 61
##
## $$partition (Abridged)
##  [1] 0 0 0 0 0 0 0 0 0 0
```

We see some distinctive components of `dat`, an object of the `BrcFmri` class. These are `data2d`, `id`, and `parcellation`.

```
names(dat)
```

```
## [1] "data2d"       "id"           "parcellation"
```

Let's first start with the last component, `parcellation`. It is an object of class `BrcParcellation`.

```
dat$parcellation
```

```
## BrcParcellation object of dimension 61 x 73 x 61
##  with 82470 parcels
## ----------
##
## $dim3d
## [1] 61 73 61
##
## $partition (Abridged)
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```
class(dat$parcellation)
```

```
## [1] "BrcParcellation"
```

```
length(dat$parcellation$partition)
```

```
## [1] 271633
```

We see that it has two elements, `dim3d` and `partition`. Here, `dim3d` represents the size of the fMRI scan, which is $61 \times 73 \times 61$ voxels in our case. Next, `partition` is a vector of length 271633 (which is equal to $61 * 73 * 61$), one element for each voxel. Each element in this vector corresponds to a specific voxel (which is not made explicit to the user), and the values of each element corresponds to which parcel the corresponding voxel belongs in. Elements with 0 are have a special meaning to denote that its corresponding voxel are empty (i.e., there is no time-series information for these voxels). For example, these empty voxels could represent voxels outside the skull.

In our case, we see that there are 82470 unique parcellations (excluding the voxels with the 0 value). We also see that each parcel has only one voxel each. This means that this is the singleton parcellation. In the later sections, we will discuss how to incorporate parcellations where each parcel contains many voxels.

```
length(unique(dat$parcellation$partition))
```

```
## [1] 82471
```

```
table(table(dat$parcellation$partition))
```

```
##
##      1 189163
##  82470      1
```

The next component we will discuss is `data2d`. In our case, this is a 5×82470 matrix, where each column represents a different time index and each row represents a different voxel. (Recall that there were 82470 parcels based on `parcellation`.) We call this the “2d” representation since it ignores spatial information, as it is unclear a priori which voxels neighbor which voxels.

The last component is `id`, which allows the researcher (you) to add some unique identifier to this fMRI data. This will have utility later on when we link fMRI data to phenotype data.

Manipulating the data representation

Our data `dat` was in the 2d representation. We discuss how to make it transform it into the “4d” representation, which will explicitly encode which voxels are spatially adjacent to which voxels.

```
dat4d <- data2dTo4d(dat$data2d, dat$parcellation)
dim(dat4d)
```

```
## [1] 61 73 61  5
```

```
class(dat4d)
```

```
## [1] "array"
```

We see that `dat4d` is an `array` object with dimensions $61 \times 73 \times 61 \times 5$, exactly what we would’ve expected had we worked with a `nifti` object (using previously established NifTI standards). Here, the first three dimensions represent the (x, y, z) spatial coordinates, so we can easily discern which voxels are spatially adjacent to which voxels. The last dimension encode the time series.

We can do a simple checks to ensure our understanding. First, we can count how many non-zero voxels there are in any of the 5 time intervals.

```
apply(dat4d, 4, function(x){length(which(x != 0))})
```

```
## [1] 82470 82470 82470 82470 82470
```

In each of the 5 time intervals, there are 82470 non-zero voxels, which is what we expected.

Applying a parcellation

Suppose we wanted to apply a (non-trivial) parcellation to `dat`. That is, we wanted to reduce the dimensionality of data from 82470 dimensions (one dimension for each voxel, since each voxel represents a different time series) to a substantially smaller dimensionality. To do this, we will need a parcellation encoded by the `BrcParcellation` class. We have an example of one in the `brcData` package.

```
aal <- brcdata::AAL_3mm
summary(aal)
```

```
## Summary of BrcParcellation object
## ----------
## Volume resolution:       61 x 73 x 61 voxels
## Number of parcellations: 90 parcels
## Estimate size:           2.07 Mb
```

```r
class(aal)
```

```
## [1] "BrcParcellation"
```

```r
aal
```

```
## BrcParcellation object of dimension 61 x 73 x 61
##  with 90 parcels
## ----------
##
## $dim3d
## [1] 61 73 61
##
## $partition (Abridged)
##  [1] 0 0 0 0 0 0 0 0 0 0
```

This is the AAL (Automated Anatomical Labeling) parcellation for 3 millimeter voxels. More information on this parcellation can be found with the `?AAL_3mm` command. Notice that `aal` has the same dimension (based on `aal$dim3d`) as `dat`. This is important, as we will not be able to apply the `aal` parcellation onto `dat` if their 3d dimensions differed. Unsurprisingly, we see that the representation of `aal` is similar to that of `dat$parcellation` as both are objects of the `BrcParcellation` class.

However, we now will show that unlike `dat$parcellation`, `aal` does not include singleton parcels. To do this, we plot how many voxels are in each parcel, sorted from fewest to most. We exclude the "0" parcel (again, which represents empty space).

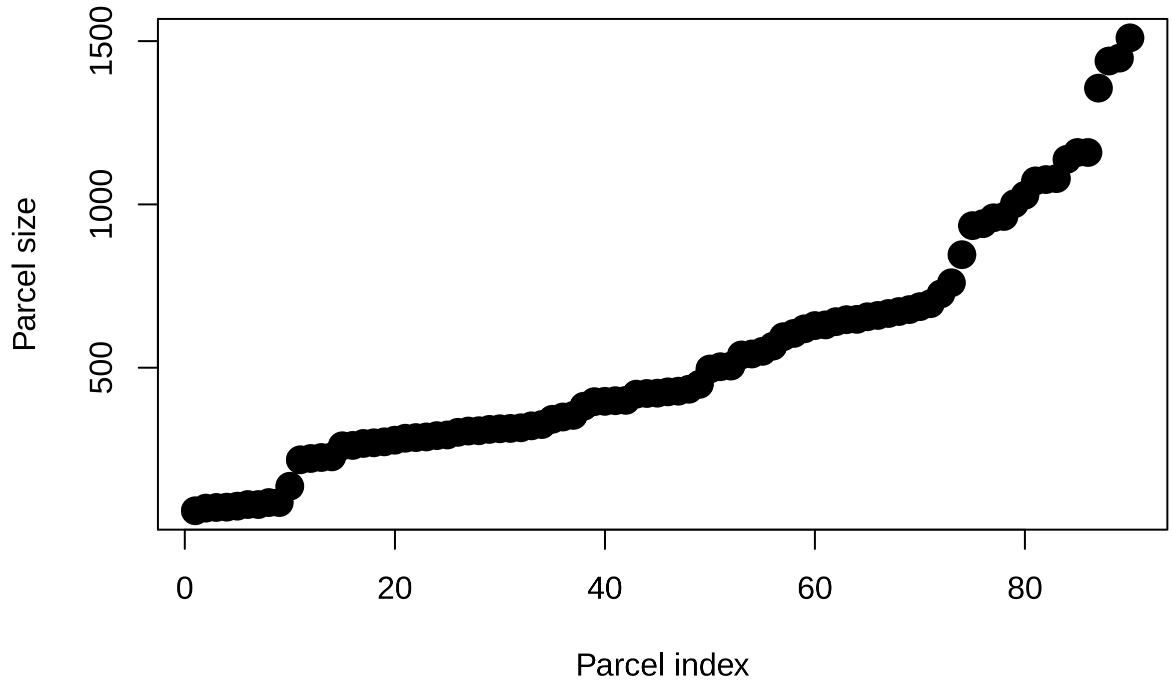
```r
vec <- aal$partition[aal$partition != 0]
length(unique(vec))
```

```
## [1] 90
```

```r
quantile(table(vec))
```

```
##     0%    25%    50%    75%   100%
##   62.0  289.0  424.0  670.5 1510.0
```

```r
plot(as.numeric(sort(table(vec))), xlab = "Parcel index", ylab = "Parcel size",
     pch = 16, cex = 2)
```



We see that more than half the parcels contain more than 400 voxels, and there are 90 parcels total (again, excluding the "0" parcel).

Now we want to apply `aal` to `dat`, returning a new dataset where there are only 90 dimensions (i.e., 90 time series) instead of 82470 dimensions.

```
dat_aal <- reduce(dat, aal, func = reduce_mean)
summary(dat_aal)
```

```
## Summary of BrcFmri object
## ----------
## Id:                         COBRE_0040071
## Volume resolution:          61 x 73 x 61 voxels
## Number of parcellations: 90 parcels
## Scan length:                5 volumes
## Estimate size:              2.08 Mb
```

```
dim(dat_aal$data2d)
```

```
## [1]  5 90
```

We see that there are only 90 parcels in our new dataset, `dat_aal`, and the 2d representation of `dat_aal` is now a 5×90 matrix. We used a function called `reduce_mean` to summarize the time series of all the voxels in the same parcellation (according to `aal`) into one time series using the mean. There are other functions we could have used such as `reduce_pca`. We encourage the reader to read about these functions using the commands `?reduce`, `?reduce_mean`, and `?reduce_pca`.

If we count how many voxels are included in our new dataset `dat_aal`, we will notice something.

```
length(which(dat_aal$parcellation$partition != 0))
```

```
## [1] 47636
```

```
length(which(dat$parcellation$partition != 0))
```

```
## [1] 82470
```

This means only 47636 voxels are accounted for in `dat_aal`, down from the original number of 82470 voxels in `dat`. (Remember, the number of parcels in `dat` is equal to the number of voxels since `dat$parcellation` is the singleton parcellation.) This is because not all the voxels represented in the `dat_aal` parcellation are represented in the `dat$parcellation` parcellation, and vice-versa. In neuroscience analyses, this can thought of as a region-ofinterest (RoI) analysis, as not every voxel is scientifically interesting to study. If the researcher wanted to "fill" the `aal` parcellation so as many of the original 82470 original voxels are accounted for in the new, reduced dataset, we discuss functions to achieve this in later sections.

Building a `BrcFmri` object

Visualization

Statistics

Example custom packages: Parcellation

Summary