The BIRCH API – An interface between R and C

In this document we briefly document the BIRCH API interface for programmers who wish to further develop this package and its capabilities.

The perspective here is from within R, and we document the `.Call` functions used and the values returned. If a more in-depth look is required, then the reader is encouraged to look at the source files, which are well documented. The source files also have dOxygen tags incorporated for the class descriptions, which may be of use as a quick overview.

LL_main

Syntax:

```
.Call("LL_main",  x, radius, compact, keeptree)
```

Arguments:

- `x`: a matrix of data of type double
- `radius`: a scalar of type double
- `compact`: a scalar of type double
- `keeptree`: an integer either 0 or 1

Returns:

A pointer to the parent node

Example:

```
.Call("LL_main",  matrix(as.double(x), ncol = ncol(x)),
                         as.double(radius), as.double(compact),
                         as.integer(1))
```

Description:

This function passes a matrix to the C code, along with specification of the radius and compactness criteria, for building the tree from nothing. The `keeptree` argument (effectively a Boolean) tells the C code whether or not to stay resident in memory.

LL_getdata

Syntax:

```
.Call("LL_getdata", pointer)
```

Arguments:

- `pointer`: a pointer

Returns:

A list containing (they are not named, so order is important)

- [[1]]: a vector with the number of observations in each subcluster
- [[2]]: a matrix of the sumXi values for each subcluster
- [[3]]: an array of the sumXisq values for each subcluster
- [[4]]: a list, each member of which contains an index of the observations in that subcluster

Example:

```
.Call("LL_getdata", attr(birchObject, "internal"))
```

Description:

Generally this command is used after either “LL_main” or “LL_adddata” to retrieve the data, as these return pointers and not the actual data. It doesn't alter the C object in any way. In the implementation in R, one needs to name these objects before returning (in the future, this should be done in C).

LL_getdim

Syntax:

```
.Call("LL_getdim", pointer)
```

Arguments:

- pointer: a pointer

Returns:

- A vector of length 2 (total number of observations; number of subclusters)

Example:

```
.Call("LL_getdim", attr(birchObject, "internal"))
```

Description:

Is used after updating a tree (e.g. adding data to it) in order to retrieve the number of subclusters, and total number of observations in the object.

LL_adddata

Syntax:

```
.Call("LL_adddata", pointer, x)
```

Arguments:

- pointer: a pointer to the parent node
- x: matrix of type double

Returns:

NULL

Example:

```
.Call("LL_adddata", attr(birchObject,"internal"),
     matrix(as.double(x), ncol = ncol(x)))
```

Description:

This is used for adding further data to the tree. Doesn't return the new tree (instead use “LL_getdata” and “LL_getdim”). Uses the same radius and compactness values that were established when the original tree was built.

LL_killtree

Syntax:

```
.Call("LL_killtree", pointer)
```

Arguments:

- `pointer`: a pointer to the parent node

Returns:

NULL

Example:

```
.Call("LL_killtree", attr(birchObject,"internal"))
```

Description:

This removes the tree from memory (by calling the destructor for the parent node). If this is not done before closing, then a “finalizor” code is registered with R (this is established when the first `keeptree=TRUE` argument is set). Still, good practice to set this off oneself if possible!