dbscanDATA501

Draft package - Test plan

Hans Eliezer - 300365985

1 Preparation

1.1 Installing the package

It would be easiest to install directly from the GitHub repository using `devtools`'s `install_github()`. The repository link is pasted below:

```r
devtools::install_github('https://github.com/hanseliezer/dbscanDATA501')
```

There are two other well-known implementations of DBSCAN available in R in packages `fpc` and `dbscan`. These will be useful to compare results.You can install these two packages using `install.packages()`:

```r
install.packages(c('fpc' 'dbscan'))
```

1.2 Loading package and dataset

The package can now be loaded as usual:

```r
library(dbscanDATA501)
```

For this test, you can just use the `iris` dataset. It has the original class labels, which is not needed for a clustering task, so it can be excluded:

```r
data(iris)
iris_X <- iris[, 1:4]
```

Though feel free to try out other datasets. The only requirement is that every column must be of numeric class.

2 Using `dbscan()` properly

2.1 Right parameters

The primary functionality for this package is the `dbscan()` function. For the first tests, you can see if the function works as intended when all the parameters are supplied as expected:

- `min_pts` and `eps` should be numeric/integers.

- Default distance `metric` is `euclidean`. Another options are `manhattan` and `precomputed`: `precomputed` accepts the data in the form of a pre-calculated distance matrix.

- Default `normalise` is `TRUE`. This will normalise the dataset prior to distance matrix calculation and algorithm fitting (skipped if `metric` is `precomputed`).

- Default `border_points` is `TRUE`. This will include "border points" as part of a cluster; this is equivalent to the original described DBSCAN algorithm. Whereas `border_points = FALSE` excludes them, which is equivalent to a later proposed 'hierarchical DBSCAN'.

The following are some example tests where everything should work as intended/no errors generated (remember to use the scaled `iris` rather than the original one):

```
test_1 <- dbscan(iris_X, 0.2, 5)
test_2 <- dbscan(iris_X, 10, 4, metric='manhattan')
test_3 <- dbscan(iris_X, 9L, 2L, normalise=FALSE)
test_4 <- dbscan(iris_X, 0.1, 10, metric='euclidean', border_pts=FALSE)
```

2.2 Right results

The DBSCAN algorithm should be fully deterministic when given exactly the same dataset, which means the generated cluster/cluster labels should be exactly equal between different implementations given the same parameters. You can use `fpc` and `dbscan`'s `dbscan()` functions to cluster the `iris_X` dataset to get their clusters, and compare them with this package's result. Note that `fpc` and `dbscan`'s `dbscan()` does *not* normalise the dataset prior to fitting, so you should do that beforehand:

```
iris_X_scl <- scale(iris_X)
```

Also note that only `dbscan`'s `dbscan()` has the `borderPoints` parameter which is equal to this package's `border_pts`, while `fpc`'s does not. `fpc` will always include border points.

Some example tests are as follows. You might want to explicitly include the package's name when calling `dbscan()` so you don't confuse yourself when recalling which result came from which package.

```
dbscan_DATA501 <- dbscanDATA501::dbscan(iris_X, 0.4, 5)
dbscan_dbscan <- dbscan::dbscan(iris_X_scl, 0.4, 5)
dbscan_fpc <- fpc::dbscan(iris_X_scl, 0.4, 5)

all(dbscan_DATA501$cluster_labs == dbscan_dbscan$cluster)
```

```
## [1] TRUE
```

```
all(dbscan_DATA501$cluster_labs == dbscan_fpc$cluster)
```

```
## [1] TRUE
```

3 Breaking it

Obviously there are so many ways you can try to break the function, the most obvious of which would be supplying the wrong type of parameters:

```
dbscan(iris_X, "9", 5)
```

```
## Error in dbscan_input_checks(data, eps, min_pts, metric, normalise, border_pts): Both min_pts
```

```r
dbscan(NULL, 0.3, 9)
```

Error in dbscan_input_checks(data, eps, min_pts, metric, normalise, border_pts): Please supply

```r
dbscan(iris_X)
```

Error in dbscan(iris_X): argument "eps" is missing, with no default

```r
dbscan(iris_X, 9, 10, metric="nonsense")
```

Error in dbscan_input_checks(data, eps, min_pts, metric, normalise, border_pts): Options for o

```r
dbscan(iris_X, 1.1, 2, normalise=iris_X)
```

Error in dbscan_input_checks(data, eps, min_pts, metric, normalise, border_pts): Both normalis

As a draft package, obviously there could be some things I have missed in terms of input checking, so by all means try as hard as you can to come up with ways to get errors out of the function.