DATA501 project - dbscanDATA501

Draft package - Test plan

Hans Eliezer - 300365985

1 Preparation

1.1 Installing the packages

It would be easiest to install my `dbscanDATA501` package directly from the GitHub repository using `devtools`'s `install_github()`. The 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 using `install.packages()`:

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

Finally, the clustering summary statistics shown in this package also requires one additional package `clValid`:

```r
install.packages('clValid')
```

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 few tests, you can try if the function works as intended when all the parameters are supplied as expected:

- data must not be NULL or have 0 rows.
- min_pts and eps should be numeric/integers.
- Default distance metric is euclidean. Another options are manhattan and precomputed: precomputed accepts 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 sliced 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 clusters. 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 Clustering summary with summary()

summary() received the dbscan object created by dbscan(), calculates and displays a few summary statistics such as how many clusters were created, as well as 'internal validation' metrics which basically tells us how "good" the created clusters are. I have included four different metrics called from three different packages: cluster, fpc and clValid. For testing, it would probably be useful to confirm that summary() successfully calls these packages and generated scores for all of them (the actual values doesn't really matter). summary()'s output should look something like this:

```
summary(dbscan_DATA501)
```

```
## DBSCAN result summary:
##
## Parameters:
## eps: 0.4
## min_pts: 5
## border_pts: TRUE
## Distance metric: euclidean
##
## Running time (s): 0.00456
## Number of generated clusters (excl. noise): 6
##
## Clustering quality metrics:
## Connectivity: 97.67222
## Silhouette width: 0.03232
## Dunn index: 0.16169
## CDbw: 4.19324
##
## NOTE: Caution must be taken when interpreting connectivity, silhouette and
## Dunn index for non-globular clusters.
```

Note that if you are testing using another dataset, there are cases where the CDbw score is NA. This *should* be okay since it is kind of a side effect on how that metric is calculated: it is a multiplication of three different values, and if one of those values is NA, then the final CDbw will be NA.

4 Breaking dbscan() and summary()

Obviously there are many, many ways you can try to break the function, the most straightforward would be to supply 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
```

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

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

```
dbscan(iris_X)
```

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

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

```
## Error in dbscan_input_checks(data, eps, min_pts, metric, normalise, border_pts): Options for c
```

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

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

It may also be possible that some datasets may generate clusters that happen to break the clustering summary statistics functions called by `summary()`, but I have yet to find such an instance.