README.md

Data mining Clustering algorithms implementation and comparison: K-MEANS and HAC – Paweł Paczuski

Clustering

Clustering is a name of a task of arranging similar objects into groups called `clusters`. Objects in one cluster are closer to each other than objects in another cluster. The objective of this project is to implement and compare two algorithms that were designed to perform clustering on data: `K-MEANS` and `HAC`

Unsupervised learning

Clustering is a task that belongs to a class of problems called *unsupervised learning*. There is no source of objective truth how the clusters should be organized, so there can be multiple correct outputs for each algorithm, depending on how *measure of closeness* is defined. In my implementations simple Euclidean distance is used to calculate distances between points.

Centroid-based clustering – K-MEANS

K-MEANS algorithm focuses on finding K points in multidimentional space that will be centers of K clusters to which belong points lying closest to a given point. This way a flat partitioning is obtained. The problem itself is NP-hard in terms of computational complexity, however, the most popular interpretation of the concept behind this algorithm (referred to simply as `K-MEANS`) does not find the optimal solution, but a local maximum. This algorithm places the k points randomly in space and then finds points that are closest to it. Then, a center of a newly formed cluster is calculated – so called `centroid` which is a mean of all points belonging to a cluster. In the next iteration points that are closest to the centroids are found and this operation is repeated until no points change cluster after an iteration. This way a set of clusters is obtained, very often not the best one. In practice, this algorithm is ran several times and the best solution among the ones obtained is selected. This makes the algorithm much more scalable in comparison to the NP-hard optimal algorithm.

Hierarchical agglomerative clustering HAC

Hierarchical agglomerative clustering does not create flat partitioning of data but a hierarchy of clusters. Initially all data-points are treated as singleton clusters. Two points closest to each other are merged into one cluster and a point being a mean of them is used instead of them in the clusters set. Then, merging is repeated until a threshold of "closeness" is crossed or all data-points are connected. Conceptually, a binary tree of connections between data-points is created – so called `dendrogram`. This may be useful for any tasks that require finding hierarchical structure in flat representation of data-points. HAC is also computationally expensive – `O(n^3)` and also requires `O(n^2)` memory which makes it impractical for larger data-sets.

Assumptions and testing data

The two algorithms were implemented in R language. This is a dynamic scripting language used mainly for data-science applications. It's strengths lie in wide range of libraries of utilities for data manipulation, data mining and statistics. R built-in data-structures are flat and there is no simple notion of a `reference` so in case of HAC implementation, it was necessary to come up with flat representation of a hierarchical results of the algorithm.

For testing purposes, builtin R dataset `Iris` was used. Data set contains 4 dimensional samples describing flowers' Sepals and Petals with label to which species it belongs.

Implementation

K-MEANS

Algorithm

Centroid-based variant of k-means algorithm was implemented. Distance between points is calculated using simple Euclidean distance.

Initialization

k-randomly chosen points are used as centers of initial clusters.

Input

Unlabelled data-frame containing data-points in each row.

Output

The algorithm adds new column to the data-frame: `cluster` which stores results of clustering operation.

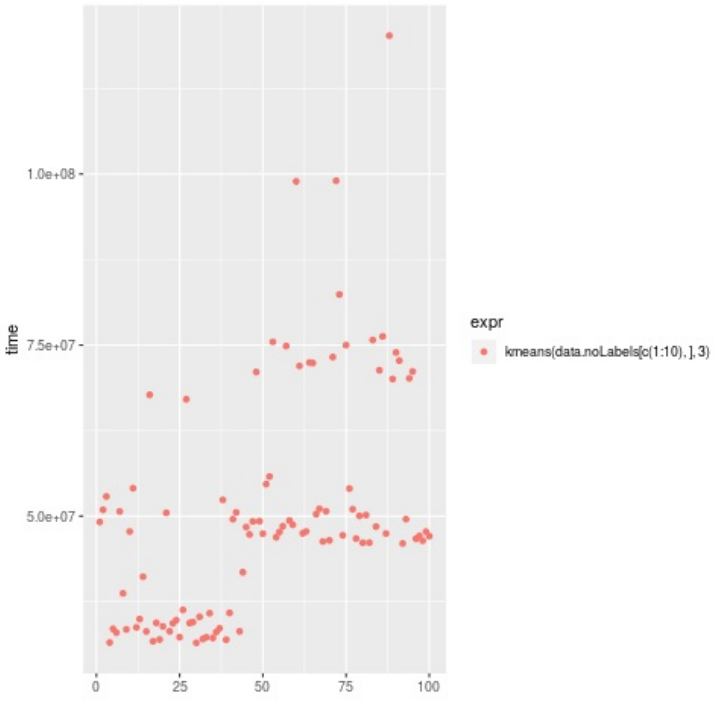
```
kmeans(data.noLabels[c(1 : 10),], 3)
```

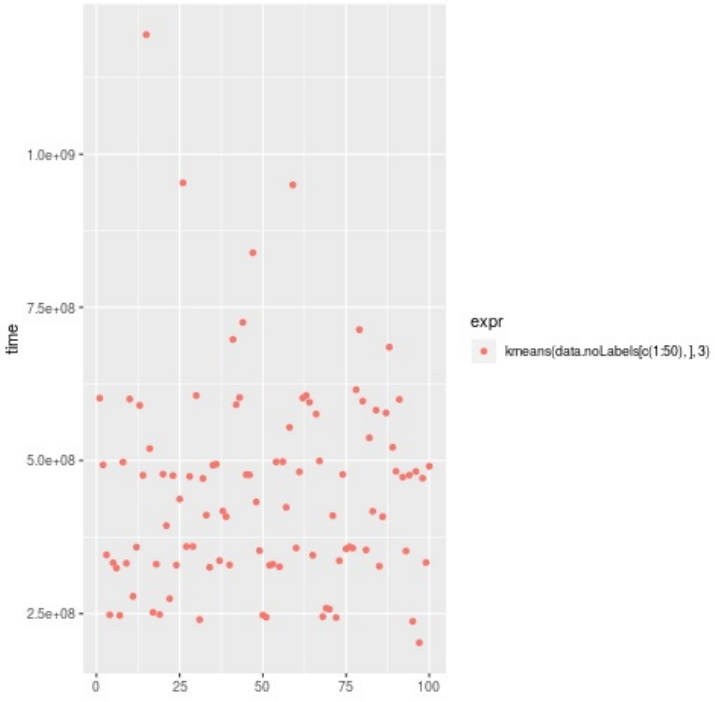
```
   Sepal.Length Sepal.Width Petal.Length Petal.Width clusterId
1           5.1         3.5          1.4         0.2         3
2           4.9         3.0          1.4         0.2         1
3           4.7         3.2          1.3         0.2         2
4           4.6         3.1          1.5         0.2         2
5           5.0         3.6          1.4         0.2         3
6           5.4         3.9          1.7         0.4         3
7           4.6         3.4          1.4         0.3         2
8           5.0         3.4          1.5         0.2         3
9           4.4         2.9          1.4         0.2         2
10          4.9         3.1          1.5         0.1         1
```

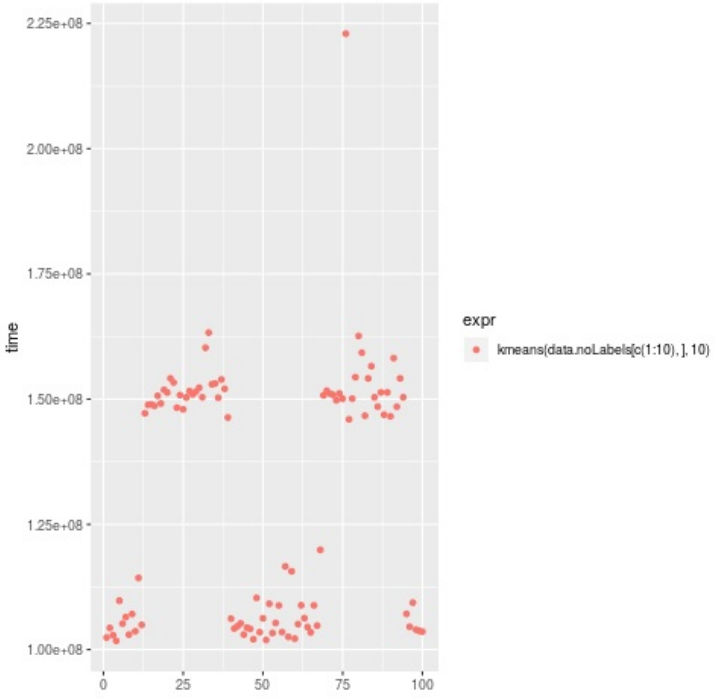
Performance

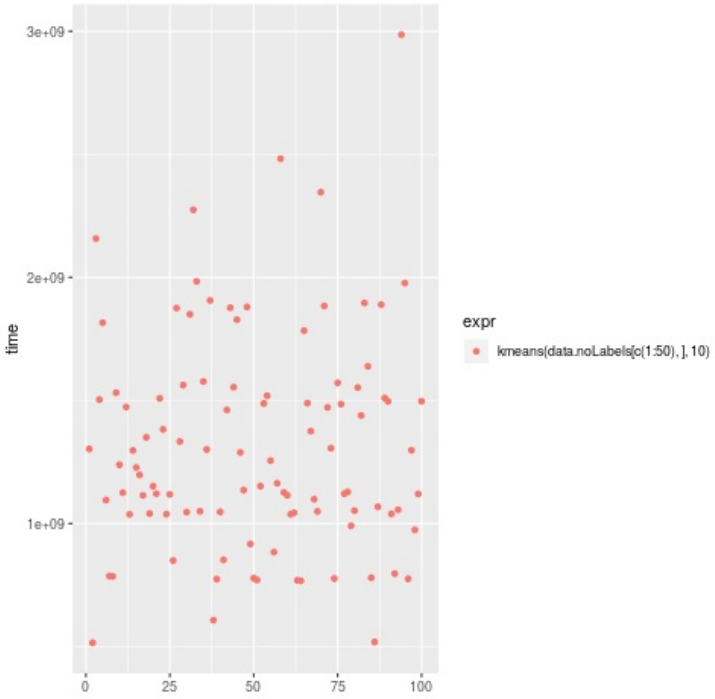
The algorithm was ran 100 times using two different sizes of input data frame and different numbers of clusters. In the table there are measurements of execution time in ms.

expr	min	lq	mean	median	uq	max	neval
kmeans(data.noLabels[c(1:10),], 3)	31.47939	35.11494	50.56061	47.73692	54.05549	120.2285	100
kmeans(data.noLabels[c(1:50),], 3)	202.6218	332.7086	452.3831	434.7092	529.3968	1195.064	100
kmeans(data.noLabels[c(1:10),], 10)	101.7517	104.8609	130.8737	146.6242	151.1198	222.9607	100
kmeans(data.noLabels[c(1:50),], 10)	515.5413	1041.901	1313.249	1232.979	1525.464	2985.969	100


time
1.0e+08
7.5e+07
5.0e+07
0
25
50
75
100
expr
kmeans(data.noLabels[c(1:10),], 3)


time
1.0e+09
7.5e+08
5.0e+08
2.5e+08
0
25
50
75
100
expr
kmeans(data.noLabels[c(1:50),], 3)





HAC

Algorithm

Algorithm creating complete `dendrogram` of clusters was implemented. Each cluster candidate is represented as a centroid of all points belonging to it.

The `agglomerative` adjective in the context of hierarchical clustering means that the clustering is performed in a bottom-up way. First we create smaller clusters and then decide how they should be merged to create higher-order clusters containg them.

HAC is very unfriendly for the R language. Everything in R is passed as a value, so even pushing to a data-frame requires copying lots of data. No straightforward concept of a reference made it harder to reason about hierarchical dependencies between clusters.

Initialization

Each data-point is treated as a separate cluster candidate.

Input

Unlabelled data-frame containing data-points in each row.

Output

The algorithm returns a data-frame containing flat representation of a dendrogram. This allows to interpret how the merging of clusters was performed. Each entry in `data` is smallest index number of a row in original data representing cluster.

```
hac(data.noLabels[c(1 : 10),])
```

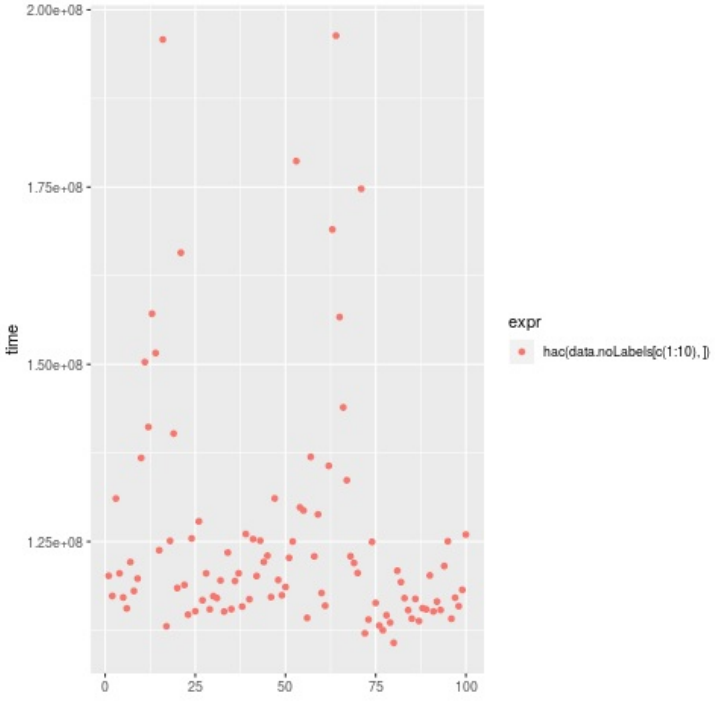
```
    data cluster
1     1       1
2     5       1
3     2       2
4    10       2
5     8       3
6     1       3
7     3       4
8     4       4
9     7       5
10    3       5
11    2       6
12    3       6
13    9       7
14    2       7
15    1       8
16    2       8
17    6       9
18    1       9
```

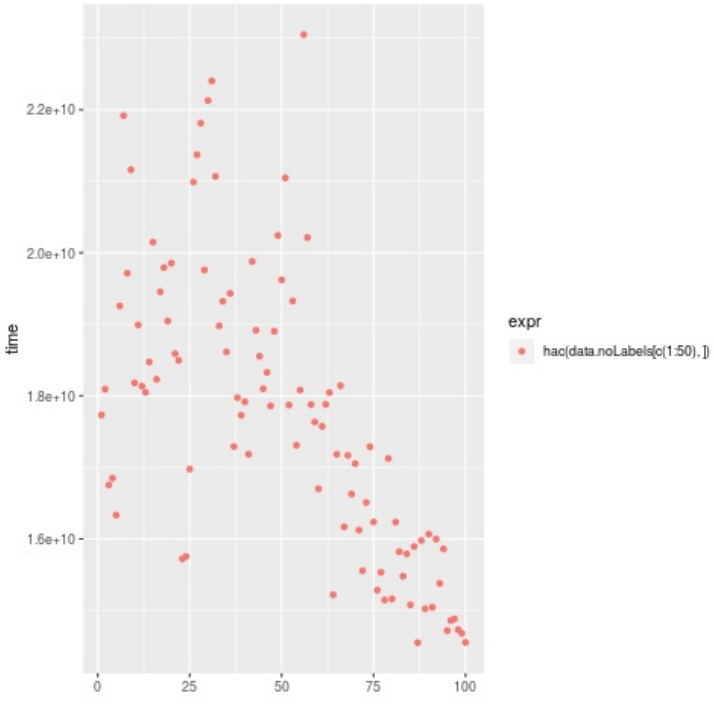
It would be possible to retrieve the full tree structure from this form in `O(n)` time.

Performance

The algorithm was ran 100 times using two different sizes of input data frame. In the table there are measurements of execution time in ms.

expr	min	lq	mean	median	uq	max	neval
hac(data.noLabels[c(1:10),])	114.3293	165.0287	173.7432	169.8413	179.6501	280.5231	100
hac(data.noLabels[c(1:50),])	14548.5	15986.98	17727.52	17795.09	19017.74	23047.62	100





Comparison and summary

Clustering quality evaluation for different distance functions

When one has labelled data, they can mark how accurately clustering was performed. Evaluation was performed on two datasets: the first one was very simple and checked whether implementations are correct, the second one was iris dataset. Two indexes were calculated for each clustering. For the first datasets, both algorithms were 100% correct.

10 - element ds (kmeans):

```
$rand
[1] 0.3555556

$folkes_mallows
[1] 0.5962848
```

full iris ds (kmeans):

```
$rand
[1] 0.873736

$folkes_mallows
[1] 0.8112428
```

10-element ds (hac):

```
[1] 0.4

$folkes_mallows
[1] 0.6324555
```

words better than kmeans for this ds

Manhattan distance

10 - element ds kmeans:

```
$rand
[1] 0.3111111

$folkes_mallows
[1] 0.5577734
```

Similar to Euclid

Full iris ds (kmeans)

```
$rand
[1] 0.8679194

$folkes_mallows
[1] 0.8031907
```

This time it was a little bit worse.

10-element ds hac:

```
$rand
[1] 0.1555556

$folkes_mallows
[1] 0.3944053
```

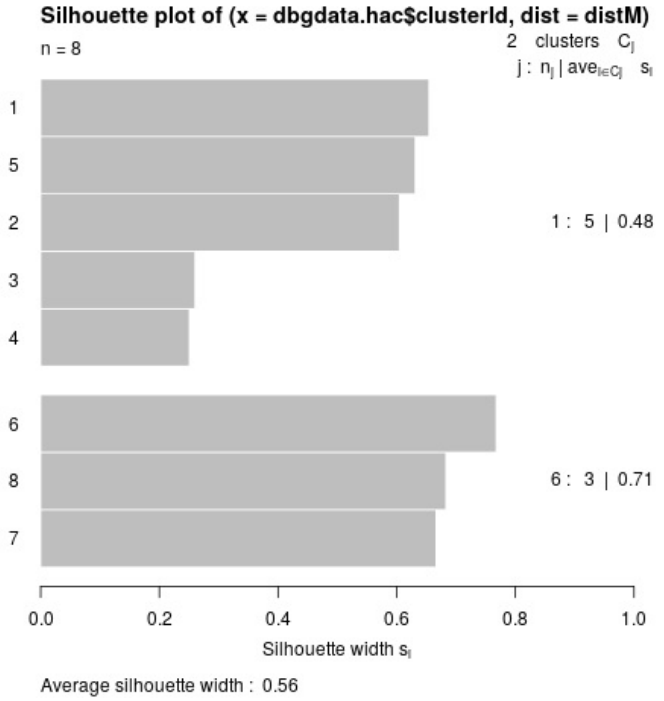
Manhattan distance for this ds performs very poorly.

This shows that clustering data when there are very few samples can be very inaccurate due to small differences between data-points. Euclidean distance should be more sensitive to outliers in data as opposed to the Manhattan. We have very few dimensions, so the differences between two distance methods are very small. HAC was, however, much worse for 10-element ds when Manhattan was used instead of Euclidean.

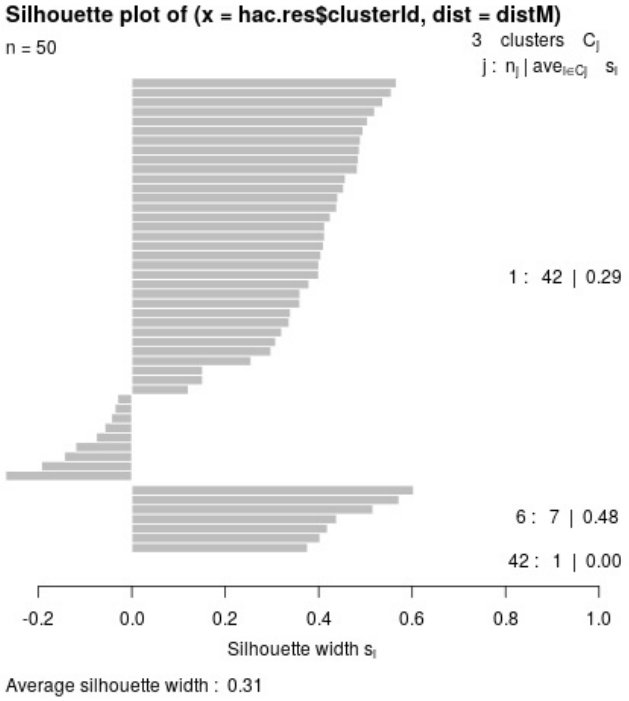
Kmeans uses randomized starting point so it is fragile to finding local minimum whereas this implementation of HAC calculates the best distance without any shortcuts.

Silohuette plots

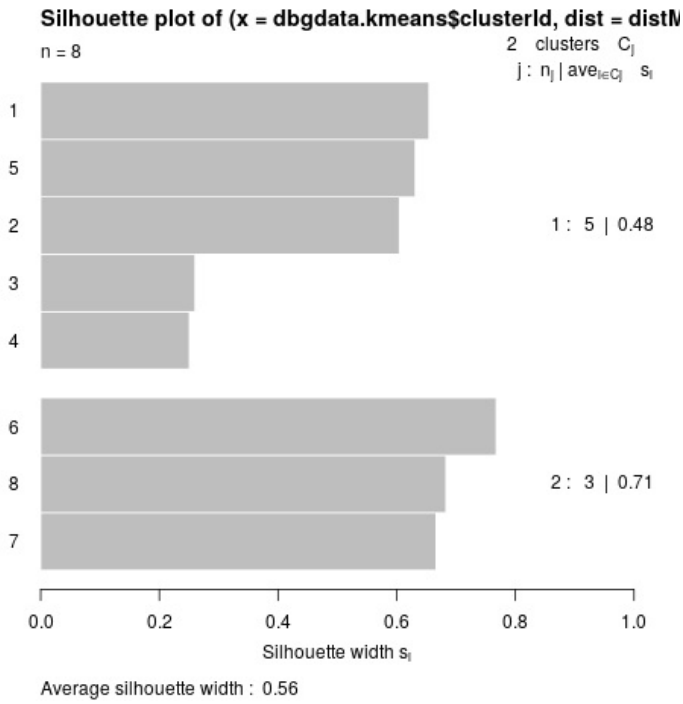
hac dbgdata



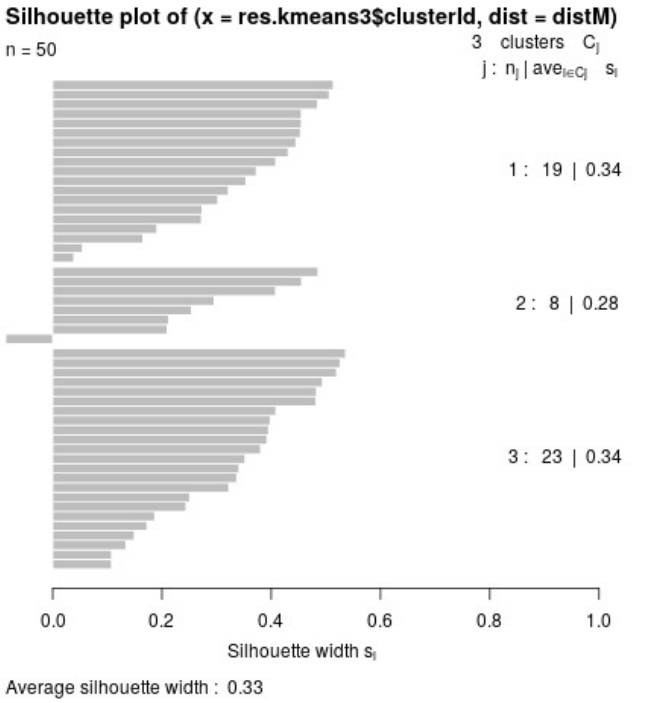
hac iris



kmeans dbgdata



kmeans iris



Observations

Although both K-MEANS and HAC split data into clusters, they have different understanding of the shapes of their results. K-means assigns each data-point to a single cluster, whereas HAC creates hierarchy of nested sets.

Simple randomized heuristic (randomized placement of centroids and testing convergence by counting how many points moved between clusters) allowed for implementation of k-means that finds reasonably good clustering quickly.

That was not the case with HAC. In spite of outputting correct and complete results, it was painfully slow. Its base time complexity was `O(n^3)` but conceptual limitations of R made it drastically slower compared to theoretical speed. If one has to really use HAC in R language, then it is much more convenient to implement this algorithm in C++ and use the builtin R interfacing mechanisms for calling external procedures written in a different language.

Which of these two algorithms one should choose depends on the task being solved. When one needs a flat partition of data-points into groups that are similar, then k-means is the way to go. On the other hand, if one is interested in finding structure in the data, then HAC should be used.

This project allowed me to get familiar with limitations of R language and to get used to the tabular way of thinking about data in this language.