

8CC00 2021 / 2022

Week 4: Clustering

Peter Hilbers

March 2022

1 Introduction on machine learning techniques

In weeks 4 through 7 the emphasis is on a second biomedical data analysis method, viz. machine learning. Roughly stated machine learning techniques can be split into 2 parts:

- Supervised learning: As the name already suggests learning takes place in the presence of a supervisor(teacher) meaning that some data is already tagged with the correct answer. The labeled data, helps you to predict outcomes for unforeseen data.
- Unsupervised learning: In unsupervised learning the data is unlabeled.

A simple example to explain the difference is: If you see a chair in a furniture shop you have never seen before, you immediately know that it is a chair. No supervisor has to assist you in giving it that label, so you 'learned' it unsupervised.

In another case when you are not sure what object it is, you may ask for help to a supervisor to tell you what kind of object it is, so you ask for the label, and then it becomes supervised.

The most used supervised method is classification. In classification, we have data points for which the groups are already known, and we analyze what differentiates these groups (i.e., a classification function) to properly classify future data.

A popular unsupervised method is clustering. In clustering, we consider data points for which groups are unknown and undefined, and we somehow divide them into groups as well as determine what differentiates the groups from each other.

In both methods objects(data points) are grouped such that objects in the same group are more similar to one another than they are to objects in other groups. Both in clustering and classification we have to define what it means that 2 data points (objects) are looking similar, hence when do they belong to the same group. To that end usually a distance is defined. In this notes therefore the topic of distances will be discussed (see section 4), but we start with two topics from computing science that are often used in machine learning, viz. **recursion** (see section 2) and **graphs** (see section 3).

A fundamental notion in programming is repetition. In previous courses and lectures

iterative methods such as `for` and `while`-constructs have been introduced. Here we discuss another repetition technique, viz. recursion. Informally speaking recursion is a repetition method in which the object to be defined is used itself. That sounds maybe somewhat confusing so an example may help.

2 Recursion

A fundamental notion in programming is repetition. In previous courses and lectures iterative methods such as `for` and `while`-constructs have been introduced. Here we discuss another repetition technique, viz. recursion. Informally speaking recursion is a repetition method in which the object to be defined is used itself. That sounds maybe somewhat confusing so an example may help.

2.1 Recursion example

Assume we have a sequence $a(n), n = 0, 1, \dots$, of integer values starting with 1, and each subsequent element is the sum of twice its predecessor and 1. Mathematically we could define this sequence by

$$a_n = \begin{cases} 1 & \text{if } n = 0 \\ 2 * a_{n-1} + 1 & \text{otherwise} \end{cases}$$

This is an example of a recursive definition, since in the definition of the sequence a the sequence a is used. Also notice that there is also a base case in which we know the value immediately (without recursion). Here the base case is $n == 0$ since then we know that $a(n) = 1$. If we would have to implement this definition in Python, we could have:

```
def a(n):
    if n==0: return 1
    else:
        h=a(n-1)
        return 2*h+1
```

For educational purposes we introduced here a local auxiliary variable h . The reason will soon become clear. Note also that we, against our philosophy, did not give a specification of the method. Why will also below become clear.

Next assume that we would have to calculate the value of $a(2)$. So we extend our program with:

```
if __name__=="__main__":
    print(a(2))
```

Simple, isn't it?

For once and only for once let us describe what happens if that piece of code is run:

call $a(2)$: The interpreter inspects its object space and finds a method called a . The method has one parameter, called n . So the method matches the call $a(2)$ and the parameter n gets the value 2 and it starts evaluating the method. The first statement is the question whether $n == 0$ so here $2 == 0$. The answer is no, so it starts evaluating the `else` part. The first statement of the `else`-block is `h=a(n-1)`, so it introduces within this method evaluation an auxiliary variable h . Because

in the current evaluation of the method $n = 2$ and to discriminate the different variables we will introduce, we will call, this specific h : h_2 . So we could read the statement $h=a(n-1)$ as “Please assistant h_2 get me the value of $a(1)$ ” and the assistant goes on its way to get $a(1)$, and the execution of call $a(2)$ is temporarily stopped.

call $a(1)$: The method a is found and a new parameter n is introduced having the value 1. Please note that this n inside $a(1)$ is another parameter, and is different from the one inside call $a(1)$. The method starts evaluating, the if-part is passed, since n differs from 0, and a new assistant h_1 is introduced to get the value of $a(0)$. The call $a(1)$ is paused...

call $a(0)$: The method a is found and a new parameter n is introduced having the value 0, the method is evaluated and now the if-part is executed, since $n == 0$. So the value 1 is returned to ... assistant h_1 .

return to call $a(1)$: Assistant h_1 has the value 1, so the call $a(1)$ can be continued. The next statement is `return 2*h+1`, so the value 3 is returned to, ..., assistant h_2 , and the call $a(1)$ ends.

return to call $a(2)$: Assistant h_2 has the value 3, so the call $a(2)$ can be continued. The next statement is `return 2*h+1`, so the value 7 is returned, to the print-statement and on the screen the number 7 is shown.

This is a reasonable description of what happens in reality. We have shown it to demonstrate the repetitive scheme, and for educational purposes, but note that in the design we never should do it this way.

2.2 The solution

What we should do is give a specification of the method:

```
{parameter n is a non-negative integer}
a(n)
{a returns 1 if n=0, and for n>0: a(n)=2*a(n-1)+1}
```

If we have this specification then we do not have to bother about what happens behind the scenes for $h = a(n - 1)$. We simply may assume that h has the correct value and then we can calculate $a(n)$ by $2 * h + 1$. That is all!

This specification scheme should always be given for any method, so also for recursive methods. The special thing in dealing with a recursion is that there should always be at least one base case in which there is no call to the recursive method. Otherwise the method will be calling itself for ever....

In the clustering and classification assignment a more complex recursive method is to be designed. In order to practice the programming of a recursive method, the following exercise should be made. (These exercises are just meant for practicing, the solutions do not have to be submitted.)

2.3 Recursion exercise 1

Implement the above recursive method a and include print statements that clarify the different values the parameters h and n have during the evaluation of the method.

Demonstrate the correctness of your solution by calling `a(4)`.

2.4 Recursion exercise 2

Given is a positive integer n . Design a recursive method to print the digits n consist of in reverse order.

Example: if $n=123$, then on the screen 321 should be printed.

In your solution you should not first determine the number of decimals n has.

Hint: for integer n , $n\%10$ gives the last decimal...

3 Graphs

A graph is a notion from mathematics that is frequently used in biomedical data analysis. Next to creating meaningful visualisations, graphs are attractive to infer patterns in the data. The fundamental element of graphs is that some kind of relation is used between data points. Although you may have not realised it in daily life you are frequently confronted with graphs such as the internet, in family trees, in the railway system, and molecules.

A graph consists of 2 collections:

1. a set(collection) of nodes. Each of our data points forms a node.
2. a set of edges. Two nodes share an edge when the two nodes are related. Instead of sharing we will also use the term 'connected', so two nodes are connected by an edge.

The collection of nodes, also called vertices, are usually denoted in mathematics by V , in object oriented terms by $G.V$. The collection of edges are usually denoted in mathematics by E , in object oriented terms by $G.E$.

Although edges may have a direction as in a one way street, we assume that they are undirected. That means that the relations we are considering are symmetric: if A is related to B, then B is also related to A. The motivation for it, is that in clustering and classification we use similarity between objects, and that relation has no direction. We therefore are considering undirected graphs.

A simple example of a graph is the molecule phenol. It consists of 6 carbon atoms, 1 oxygen atom and 6 hydrogen atoms. So the 13 atoms form our collection of nodes. In this phase we neglect the double bonds between two nodes(atoms) and consider the atoms simply to be connected if they either have a single bond or a double bond. A pictorial presentation of this molecule is shown in Figure 1.

3.1 Degree

The number of nodes a node is connected to, may depend on the the node. In the phenol example node 9, has only node 6 as neighbour, while node 1 share edges with 3 other nodes, viz. nodes 6, 2, and 8. The number of neighbours a node has is called the degree.

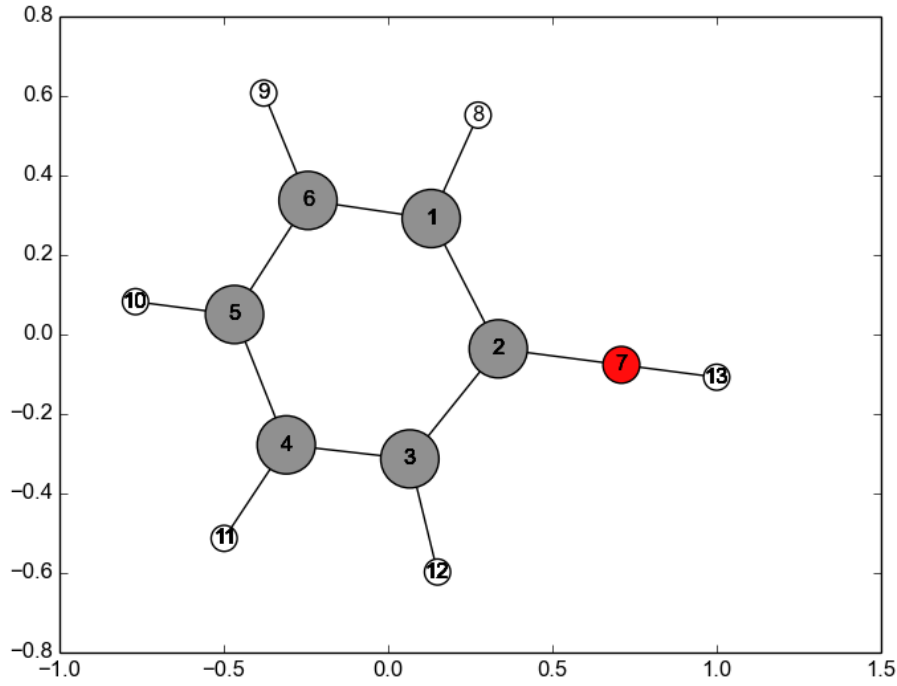


Figure 1: A drawing of the graph phenol. The oxygen atom is colored red, the carbons grey and the hydrogen white.

In mathematical notation:

$$d(u) = (\underline{N}v : v \in G.V : \{u, v\} \in G.E),$$

where \underline{N} is the notation for 'the number of'.

3.2 Path

In most cases not every 2 nodes are connected, but by following a sequence of nodes one can go from one node to the other. Consider for instance in the phenol example nodes 3 and 6. They do not share an edge, but we can go from 3 to 6, by starting in 3, going to 2, then to 1, and finally to 6. Such a sequence of nodes in which each 2 consecutive nodes share an edge is called a path and the length of the path is defined as the number of edges on the path.

A *path* of length n of G is a sequence $v(i : 0 \leq i \leq n)$ of nodes such that

$$(\forall i : 0 \leq i < n : \{v(i), v(i+1)\} \in G.E).$$

We define the distance between 2 nodes as the length of a shortest path between the two nodes. As an example in the phenol graph the distance between nodes 6 and 7 is 3.

A graph is called *connected* if every pair of vertices is joined by a path. Most graphs are connected, but for instance if our nodes would consists of the European countries and

we have as relation “share a border”, then we can go from the Netherlands to Poland via Germany, but we cannot reach England from the Netherlands.

3.3 Subgraph

Given a graph G , consider a subset V_0 of $G.V$. That means we select a collection of nodes from G . We then can make a new graph by considering the edges from G that have both endpoints of the edge in V_0 . If we call this graph H , then we have $H.V \subseteq G.V$ and $H.E \subseteq G.E$ and $(\forall u, v : \{u, v\} \in H.E : u \in H.V \text{ and } v \in H.V)$.

Given a set V . A *partition* of V is a grouping of its elements into non-empty subsets, in such a way that every element is included in exactly one subset. When considering graphs, partitioning $G.V$ into 2 subsets has a special name. It is called a *cut*. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition.

In mathematical notation: A cut $C = (S, T)$ is a partition of $G.V$ of a graph $G = (G.V, G.E)$ into two subsets S and T . The cut-set of a cut $C = (S, T)$ is the set $\{\{u, v\} \in G.E \mid u \in S, v \in T\}$ of edges that have one endpoint in S and the other endpoint in T .

If we consider our phenol example again, then an example of a cut is: $S = \{10\}$ and $T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12\}$ with cut-set $\{\{5, 10\}\}$, but of course many other cuts are possible.

In particular when dealing with cluster analysis we are interested in so-called *minimum cuts*, the smallest number of edges that has to be removed such that G becomes disconnected. The example cut-set given above is a minimum cut since after removing the edge $\{5, 10\}$, node 10 can no longer be reached from for instance node 1 and removing less edges does not lead to a disconnected graph.

3.4 Highly connected subgraph

For instance in clustering the task is to group similar objects together. Hence we want within a cluster some kind of homogeneity. On the other hand between clusters there should be some difference, hence a kind of heterogeneity. There are many possibilities for defining a homogeneity characteristic, here we choose one that is also to be used in the assignment: *highly connectedness*.

A graph with n nodes is called highly connected when its minimum cut has more than $n/2$ edges or when $n = 1$. Its applicability in cluster homogeneity comes from a famous theorem in graph theory that says that in a highly connected graph where each node has degree at least $n/2$ then every two nodes are connected by a path of length at most 2. So in our cluster analysis clusters considered as graph having a distance of at most 2, is interpreted as being homogeneous.

3.5 Multigraph

Above we have assumed that between any two nodes there is at most a single edge. In general that is too severe a restriction and we should allow for graphs having more than one edge between 2 nodes. Such a graph is called a *multigraph*. All notions introduced for a graph also hold for multigraphs, only when considering the degree one has to consider the number of edges a node is part of and not the number of neighbours.

3.6 NetworkX

For the implementation of graphs and multigraphs we will make use of *NetworkX*, see <https://networkx.org/>. There are 2 ways to install networkx:

- In the anaconda prompt type `conda install networkx`
- When that fails, an alternative is to type in a command window `pip install networkx`

On that site you will also find a [tutorial](#) and you are advised to follow that tutorial first.

Next we give our code to generate the phenol graph.

```
import networkx as NX
import pylab as P

def atomcolorsandsizes():
    """construct a dictionary to give different
        atoms a different color and size
        returns 2 dictionaries
    """

    atoms = {}

    atomcolors={}
    atomcolors['C']='#909090'
    atomcolors['O']='#FF0D0D'
    atomcolors['H']='#DDDDDD'

    atomsizes={}
    atomsizes['C']=1000
    atomsizes['O']=400
    atomsizes['H']=300
    return atomcolors, atomsizes

def readnodesfromfile(nodesfilename="", G=NX.Graph()):
    """ read the nodes from a file
        each line consists of a number and the atomtype
        separated by blank space.

        nodes are added to the graph G
        atomtypes to the list of atoms

        returns G and a dictionary of atoms
    """

    f=open(nodesfilename)
    lines = f.readlines()
    atoms={}
    f.close()
```

```

for line in lines:
    [nr, atomtype] = line.split()
    if not atomtype in atoms.keys():
        atoms[atomtype] = []
    G.add_node(nr)
    atoms[atomtype].append(nr)
return G, atoms

def readedgesfromfile(edgesfilename="", G=NX.Graph()):
    """ read the edges from a file
        each line consists of the 2 numbers of the nodes of the edge

        returns updated G
    """

    f=open(edgesfilename)
    lines = f.readlines()
    f.close()
    for line in lines:
        inds = line.split()
        G.add_edge(inds[0],inds[1])
    return G

def makegraphfromfile(nodesfilename="", edgesfilename=""):

    atomcolors, atomsizes=atomcolorsandsizes()
    G=NX.Graph()
    G, atoms=readnodesfromfile(nodesfilename=nodesfilename, G=G)
    G=readedgesfromfile(edgesfilename=edgesfilename, G=G)
    print("The nodes of G are:", list(G.nodes))
    print("The edges of G are:", list(G.edges))
    return G, atoms, atomcolors, atomsizes

def drawGraph(G=NX.Graph(), atoms={}, atomsizes={}, atomcolors={},
              nritations=100):
    """ - G is a graph,
        - atoms a dictionary with as keys the atomtypes
          and as values a list of atom numbers of that atomtype
        - atomcolors is a dictionary of colors per atomtype
        - atomsizes is a dictionary of colors per atomtype
        - nritations the number of iterations the layout
          algorithm should do
    """

    fig=P.figure()

```



```

pos=NX.spring_layout(G, iterations=nriterations)
for atomtype in atoms.keys():
    NX.draw_networkx_nodes(G, pos, nodelist=atoms[atomtype],
                           node_color=atomcolors[atomtype],
                           node_size=atomsizes[atomtype])
NX.draw_networkx_edges(G,pos)

labels = {}
for node in G.nodes():
    labels[node] = node
NX.draw_networkx_labels(G,pos,labels,font_color='black',
                        font_family='sans-serif')

P.show()

if __name__=="__main__":
    G, atoms, atomcolors, atomsizes=
        makegraphfromfile('fenolnodes.txt', 'fenoledges.txt')
    drawGraph(G=G, atoms=atoms, atomcolors=atomcolors,
              atomsizes=atomsizes, nriterations=500)

```

where contents of the files 'fenolnodes.txt' and 'fenoledges.txt' are:

```

1 C
2 C
3 C
4 C
5 C
6 C
7 O
8 H
9 H
10 H
11 H
12 H
13 H

and
1 2
1 6
1 8
2 7
2 3
3 12
3 4
4 11
4 5
5 6
5 10

```

6 9
7 13

respectively.

Exercise phenol: Study the above Python code and improve it by designing a class Atom and by showing its usage. An object of this class should have at least as attributes the size of an atom and its color.

4 Distances

As stated in the introduction section in machine learning methods an important concept is [similarity](#). If 2 data objects look similar, then when considering grouping of objects they should become part of the same group. On the other hand two objects that are completely different should not be put in the same groups. This leads to 2 important notions:

Homogeneity: Objects that are in the same group should be highly similar to each other.

Separation: Objects from different groups should have low similarity to each other.

For these notions we need a (*dis*)similarity measure, such as a distance.

Assume that $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ is the data set (objects) of n vectors from a D -dimensional space.

- The most well-known distance is the *Euclidean distance*:

$$d_E(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{p=1}^D (u_p - v_p)^2} = \sqrt{(\mathbf{u} - \mathbf{v})^T (\mathbf{u} - \mathbf{v})}.$$

Notice that this is dissimilarity measure: the larger the value the more distinct the objects are.

- Instead of the Euclidean distance its squared version is also frequently used:

$$d_E^2(\mathbf{u}, \mathbf{v}) = \sum_{p=1}^D (u_p - v_p)^2 = (\mathbf{u} - \mathbf{v})^T (\mathbf{u} - \mathbf{v}).$$

- A third distance measure is the Manhattan distance

$$d_{MH}(\mathbf{u}, \mathbf{v}) = \sum_{p=1}^D |u_p - v_p|.$$

- Correlation based dissimilarity: If \mathbf{u} and \mathbf{v} are 2 vectors from a D -dimensional space we can define their correlation by:

$$Cor(\mathbf{u}, \mathbf{v}) = \frac{\sum_{p=1}^D (u_p - \bar{\mathbf{u}})(v_p - \bar{\mathbf{v}})}{\sqrt{\sum_{p=1}^D (u_p - \bar{\mathbf{u}})^2 \sum_{p=1}^D (v_p - \bar{\mathbf{v}})^2}},$$

where

$$\bar{\mathbf{u}} = \frac{1}{D} \sum_{p=1}^D u_p$$

is the mean of \mathbf{u} , and similarly for \mathbf{v} . Notice that $-1 \leq \text{Cor}(\mathbf{u}, \mathbf{v}) \leq 1$.

It is not difficult to make from a correlation coefficient a similarity measure, simply use:

$$d_C(\mathbf{u}, \mathbf{v}) = 1 - |\text{Cor}(\mathbf{u}, \mathbf{v})|.$$

So when \mathbf{u} and \mathbf{v} are highly (negatively or positively) correlated they have a small $d_C(\mathbf{u}, \mathbf{v})$ -value.

So when either performing a clustering or a classification we should always first determine which similarity measure to choose and to note that results may depend on that choice.

5 Clustering

The goal of cluster analysis is to group elements into disjoint subsets, or clusters, based on similarity between elements, so that elements in the same cluster are highly similar to each other (homogeneity), while elements from different clusters have low similarity to each other (separation).

Hence, we have two tasks to achieve:

- Maximal homogeneity within each cluster
- Maximal heterogeneity between clusters: separation

To discuss the clustering we need some definitions:

- $C = \{C_1, \dots, C_k\}$ is a clustering of X , a partitioning of X into k clusters.
- $n_i = |C_i|$ the number of elements of cluster C_i .
- $C(\mathbf{x}_j)$ is the cluster where \mathbf{x}_j is assigned to.

In general we do not know a priori how many clusters our data set should have. In some methods (such as k -means) one has on beforehand to choose the number of clusters and then typically a series of small values for k are chosen, such as $1 \leq k \leq 10$, but there is no guarantee that not a better clustering could be obtained by a larger value of k . So we may obtain several clusterings? How should we compare them?

5.1 Cluster weights

How to determine the quality of a clustering? If we have a distance measure, then we may consider how similar all objects in a cluster are by considering their distances and take the sum over all clusters to obtain the clustering weight W_C .

$$W_i = \sum_{\mathbf{x}, \mathbf{y} \in C_i} d(\mathbf{x}, \mathbf{y})$$

$$W_C = \sum_{i=1}^k W_i = \sum_{i=1}^k \sum_{\mathbf{x}, \mathbf{y} \in C_i} d(\mathbf{x}, \mathbf{y})$$

So we are opting for getting W_C as small as possible.

Another approach, that is used in the calculation of the silhouette score (see 5.4), is to consider not only the distances within a cluster, but also the distances of a data point to other clusters:

Let \mathbf{x} be a data point and C_j be a cluster, then we can define the mean distance of \mathbf{x} to cluster C_j by

$$MC(\mathbf{x}, C_j) = \frac{1}{|C_j|} \sum_{\mathbf{y} \in C_j} d(\mathbf{x}, \mathbf{y})$$

For the cluster $C(\mathbf{x})$ to which data point \mathbf{x} belongs, $W(\mathbf{x}) = MC(\mathbf{x}, C(\mathbf{x}))$ denotes how well \mathbf{x} is assigned to its cluster, for all other clusters it denotes the mean dissimilarity of point \mathbf{x} to that cluster. The cluster with the smallest mean dissimilarity is said to be the “neighbouring cluster” of \mathbf{x} because it is the next best fit cluster for that point:

$$NC(\mathbf{x}) = \min_{C_j \neq C(\mathbf{x})} MC(\mathbf{x}, C_j).$$

For data point \mathbf{x} , the *silhouette* of \mathbf{x} is defined by

$$s(\mathbf{x}) = \begin{cases} 0 & \text{if } |C(\mathbf{x})| = 1 \\ \frac{NC(\mathbf{x}) - W(\mathbf{x})}{\max(NC(\mathbf{x}), W(\mathbf{x}))} & \text{otherwise} \end{cases}$$

This can also be written as:

$$s(\mathbf{x}) = \begin{cases} 1 - \frac{W(\mathbf{x})}{NC(\mathbf{x})} & \text{if } W(\mathbf{x}) < NC(\mathbf{x}) \\ 0 & \text{if } W(\mathbf{x}) = NC(\mathbf{x}) \\ \frac{NC(\mathbf{x})}{W(\mathbf{x})} - 1 & \text{if } W(\mathbf{x}) > NC(\mathbf{x}) \end{cases}$$

As $W(\mathbf{x})$ is a measure of how dissimilar \mathbf{x} is to its own cluster, a small value means it is well matched. Furthermore, a large $NC(\mathbf{x})$ means that \mathbf{x} is badly matched to its neighbouring cluster. Thus an $s(\mathbf{x})$ close to one means that the data point is appropriately clustered. If $s(\mathbf{x})$ is close to negative one, then it would be more appropriate if it was clustered in its neighbouring cluster.

Since calculating all pair distances is costly, for large n an alternative is more appropriate.

5.2 Reference vector based weights

Instead of calculating the distances between all pairs in a cluster, we could also choose some kind of reference vector \mathbf{r}_i for cluster C_i and calculate the distance from each of the cluster points to that reference vector:

$$R_i = \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{r}_i)$$

Summing this over all clusters gives then:

$$R = \sum_{i=1}^k R_i = \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} d(\mathbf{x}, \mathbf{r}_i)$$

How should we choose \mathbf{r}_i the reference vector of cluster C_i ? A frequently used one and the one we choose for the assignment is the so-called *centroid* or *mean vector*:

$$\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}.$$

So the quality of the clustering is determined by the distance of all data points to this mean point. As we will see in the next subsection it is also the main actor in one of the most used clustering methods: *k*-means.

5.3 *k*-means clustering

k-means is one of the most well-known clustering methods. On beforehand the number of clusters has to be chosen. This is the number *k* in *k*-means. The aim of the method is to partition the *n* data points into *k* clusters in which each data point belongs to the cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. The algorithm can informally be described by:

Assign randomly each data point \mathbf{x}_i to a cluster

repeat the following steps

calculate the location of each of the cluster centroids

(re)assign each data point to a cluster with a nearest centroid

until the clustering does not change any more

An obvious question is whether this program ends. That is not trivial, since it depends on the distance chosen. One can prove that

$$E = \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} d_E^2(\mathbf{x}, \mathbf{m}_i),$$

(the sum of the squares of the Euclidean distance of a data point to its cluster centroid) is minimized. Since this sum is finite, the algorithm indeed ends in a minimum.

Notice that *k*-means might not always terminate in an optimal clustering.

A very simple 1D example is: $X = \{0, 8, 14\}$, so $n = 3$, $x_1 = 0, x_2 = 8, x_3 = 14$ and $k = 2$. Let $C = \{C_1, C_2\}$ be given by $C_1 = \{x_1, x_2\}$ and $C_2 = \{x_3\}$, then C is stable under *k*-means, since each data point is nearest to its centroid ($m_1 = 4, m_2 = 14$) and $E = 4^2 + 4^2 + 0^2 = 32$. For the optimal clustering $C = \{\{x_1\}, \{x_2, x_3\}\}$ we have $m_1 = 0, m_2 = 11$ and $E = 0^2 + 3^2 + 3^2 = 18$.

Some other remarks with respect to *k*-means algorithm:

- Since in the initial steps data points are randomly assigned to clusters, this step influences the resulting *E*-value. *k*-means should therefore be run several times (e.g. 1000) to obtain a good result.

- k -means is non-deterministic.

In the iteration step, a data point is (re)assigned to a cluster with a nearest centroid. Although unlikely there might be more than one nearest centroid. Consequently the selection of which of those nearest centroids is arbitrary and may lead to different results in different runs on the same data set.

- Clusters may become empty.

Although we select k , it might happen that we end with less than k non-empty clusters after running the algorithm. A simple example is:

$X = \{(0, -1), (0, 1), (1, 0), (5, 0), (6, -1), (6, 1)\}$ and $k = 3$.

Consider the initial clustering $C = \{\{(0, -1), (0, 1)\}, \{(1, 0), (5, 0)\}, \{(6, -1), (6, 1)\}\}$. Data point $(1, 0)$ of the second cluster is closer to the centroid of the first cluster ($m_1 = (0, 0)$), while data point $(5, 0)$ of the second cluster is closer to the centroid of the third cluster ($m_3 = (6, 0)$), so these points are reassigned to other clusters. The other data points remain in the cluster they were in, so we end with 2 non-empty clusters of each 3 data points.

- The complexity of k -means is low.

In one iteration the search for the nearest centroid for each data point is in total of the order $O(n)$. In most cases the number of iterations before a stable clustering is obtained is small, making the algorithm fast and attractive.

5.4 The choice of k

Above we have already introduced some weights measures. To determine which choice of k is the most appropriate the so-called *Silhouette* method is often used. As the name already suggests it is using the silhouette scores of the data points:

The *Silhouette coefficient* of a clustering C of data set $X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ is defined by

$$S(C, X) = \frac{1}{n} \sum_{i=1}^n s(\mathbf{x}_i).$$

So to determine the best choice of k , for different values of k the Silhouette coefficient is calculated and the one with the highest Silhouette coefficient is chosen to be the best.

5.5 Highly connected subgraph clustering

In 2000 the article "A clustering algorithm based on graph connectivity", appeared in Information Processing Letters (IPL 76 (4–6): 175–181, doi:10.1016/S0020-0190(00)00142-3). In it a method is described that does not make any prior assumptions on the number of the clusters and instead of Euclidean distances the similarity of data is represented in a similarity graph. On https://en.wikipedia.org/wiki/HCS_clustering_algorithm a nice example is shown on how the Highly Connected Subgraph(HCS) clustering algorithm partitions an example similarity graph into three clusters.

An obvious question is what to choose as similarity, or in graph terms to determine when do 2 nodes (data points) share an edge. Since a high absolute value of the cor-

relation coefficient of 2 data points(vectors) means that the vectors are similar, one possibility is to choose the absolute value of the correlation coefficient of 2 data points as a similarity measure and to select a threshold value. An edge between 2 different data points is then included in the graph as the absolute value of their correlation coefficient is above that threshold.

In such a similarity graph, the more edges exist for a given number of vertices, the more similar such a set of vertices are between each other. In other words, if we try to disconnect a similarity graph by removing edges, the more edges we need to remove before the graph becomes disconnected, the more similar this set of vertices in this graph. In the short note about graphs we already have introduced the notion *cut* and the coupled *cut-set*. Instead of the algorithm of the article where minimum cuts are employed, we take a somewhat simpler approach by using *Karger* cut-sets.

A pseudocode of the adapted HCS clustering algorithm is (Note that it is a pseudocode and need to be worked out):

```
def adaptedHCS(G=nx.Graph()):
    if highly_connected(G):
        return G
    else:
        H1, H2, C=KargerCut(G)
        p1=adaptedHCS(H1)
        p2=adaptedHCS(H2)
```

The KargerCut-method in the above code refers to Karger's algorithm. On https://en.wikipedia.org/wiki/Karger%27s_algorithm it is described, but a nice explanation is also given on <https://medium.com/@dev.elect.iitd/kargers-algorithm-d8067eb1b790>. From that last source we cite:

```
``Karger's algorithm is a randomized
algorithm to compute a minimum cut of a connected graph. The
fundamental operation of Karger's algorithm is a form of edge
contraction. The algorithm iteratively contracts randomly chosen edges
until only two nodes remain; those nodes represent a cut in the
original graph. By iterating this basic algorithm a sufficient number
of times, a minimum cut can be found with high probability.
```

Edge Contraction:

```
The edge-contraction is a simple process, whereby two nodes are merged
to form a super-node. The edges between two super nodes are called
super edges which are the set of all the edges between those two set
of super nodes. The algorithm will randomly find two nodes and merge
them to create a super-node and perform this step until there are
only two super nodes left. ``
```

So Karger's algorithm is not necessarily computing a minimum cut, and that is why have called it a KargerCut.

If the adapted HCS algorithm is applied as clustering method the result is a collection of highly connected subgraphs that forms the set of clusters.

6 Assignment week4

In this individual assignment several Python programs have to be designed to cluster data points.

The data that is to be used is the same as for your individual PCA assignment, so it consists of 100 data points(molecules), where each data point is a 30-dimensional vector. Each data point has also a label, its pharmaceutically relevant target.

In your solution you may use pandas to read the data, random for generating random numbers, and numpy for numerical calculations, but it is not allowed to use the kmeans method from modules such as scipy and sklearn. In this case the task is to program it yourself. As stated in the individual assignment of the PCA case your report should include documentation, including and with emphasis on specifications, the model, complexity analysis, and last but not least the interpretation of the results.

The assignment about clustering consists of several subtasks that have to be performed:

1. Implement the k -means algorithm where k and the distance method should be parameters of your method.
2. Use as distance the squared Euclidean distance and run your code for different values of k and apply a silhouette analysis. Note that the results of the k -means algorithm strongly depend on the initial assignment, so it is advised to run the algorithm a several (e.g. 1000) times and select the best solutions.
3. Choose for k the value with the highest silhouette score. Discuss the k -means clusters that are generated and give possible interpretations.
4. In order to generate a graph in which edges connect nodes having a sufficiently high absolute value of their correlation coefficient, we need a threshold value. If we have n nodes, there are $n(n-1)/2$ pairs of nodes. For $0 \leq c \leq 1$ let $f(c)$ denote the fraction of those $n(n-1)/2$ pairs of nodes that have an absolute value of their correlation coefficient at least c . So if $c = 0$ all pairs satisfy this criterion and hence $f(0) = 1$ while for $c = 1$ only the pairs of nodes with a perfect correlation would rest. Construct a plot with on the x-axis a value c ranging from 0 to 1, and $f(c)$ as y-value.
5. Implement the Highly Connected Subgraph(HCS) algorithm including the KargerCut method.
6. Choose a value c such that $f(c)$ is approximately 0.1, and construct the corresponding graph with approximately $0.1 * n(n-1)/2$ edges. Apply the HCS algorithm on this graph.
7. Discuss the differences of the results of the HCS algorithm with those of the k -means algorithm with for k the value with the highest silhouette score.

So apart from the programming aspects, the interpretation of the results are also important in the reporting.

Success and enjoy!!!