Data Mining and Machine Learning

Clustering I

Dr. Edgar Acuna
Departmento de Matematicas

Universidad de Puerto Rico- Mayaguez

academic.uprm.edu/eacuna

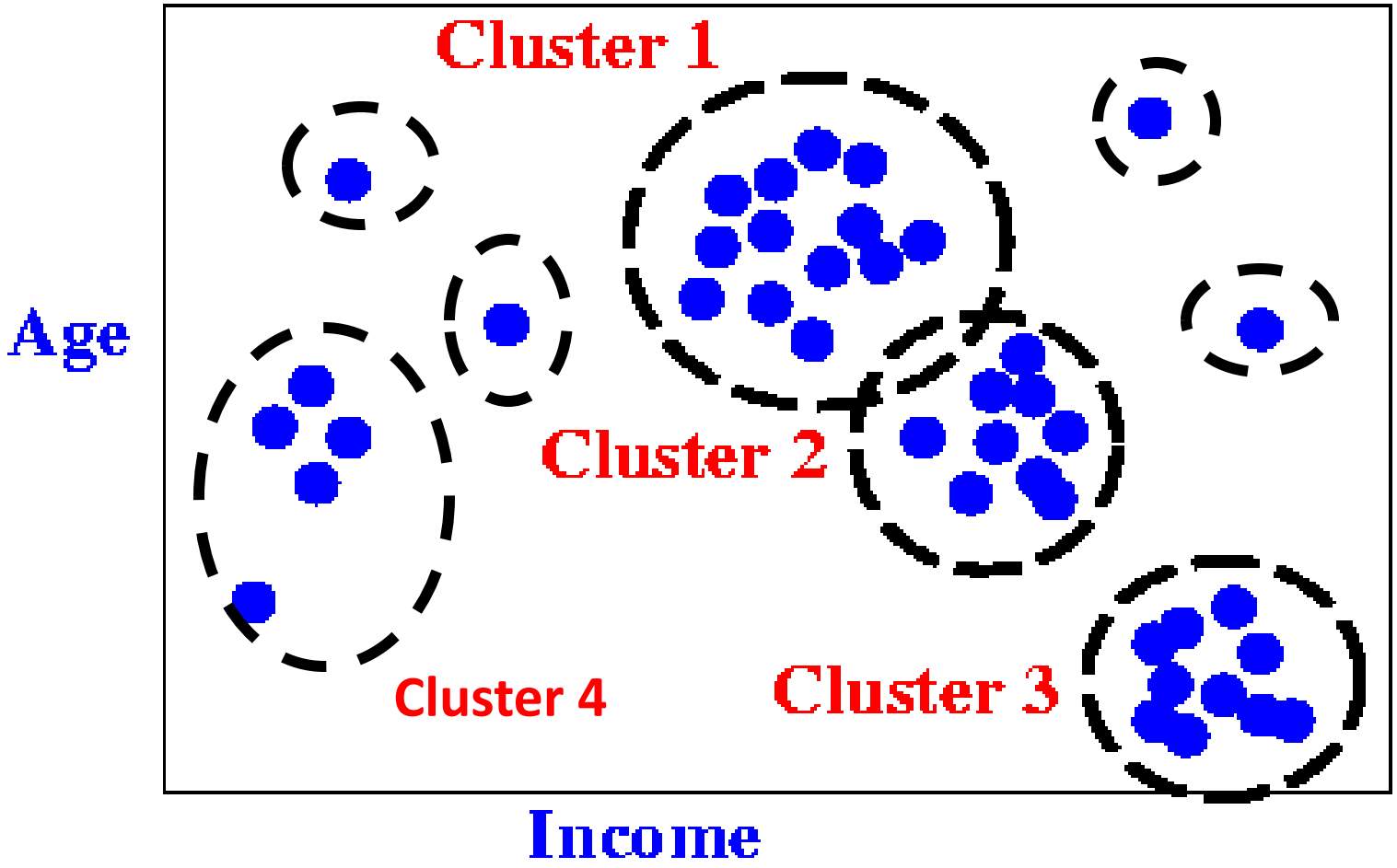
Introduction

- The main idea of clustering is to group samples (rows) or features (columns) or both at the same time, according to the separation between them. The separation is determined by a distance measure called **dissimilarity measure**. It is assumed that the classes where the instances belong are unknown.

- For each sample, there exists a measurements vector $\mathbf{X}=(X_1,\ldots X_G)$, here G is the number of features.

- The goal is to identify groups of similar samples based on n observed measurements $\mathbf{X}_1=\mathbf{x}_1,\ldots.$, $\mathbf{X}_n=\mathbf{x}_n$. For instance, if the x's represent levels of genetic expression obtained in microarrays of cáncerous tumors, one could be able to identify the genetic characteristics of persons having distinct type of cancerous tumors.

- Clustering is also known as unsupervised classification or segmentation.


Cluster 1
Age
Cluster 2
Cluster 4
Cluster 3
Income

Introduction

- When the number of columns is very large, these can be grouped according to their similarity and in this way the data dimensionality can be reduced. After that it is much easier to build a prediction model, since it is more convenient work with a small number of predictors.

- Also, clustering can be applied simultaneously to rows and columns of a dataset (Bi-clustering) (see Alon, et al, 1999, Getz et al , 2000 and, Lazaeronni and Owen, 2000)

Issues to take in account in clustering

i) Which features can be used to form the groups? ,

ii) Which dissimilarity measure should be used?.

iii) Which clustering algorithm must be used?.

iv) How many clusters should be formed?

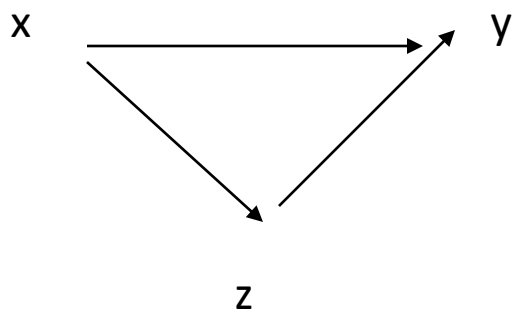
v) How to assign row (or columns) into clusters?

vi) How to validate the formed clusters?

Properties of dissimilarity measures

- Non-negativity: $d(x,y) \geq 0$
- The distance of instance to iself is 0, $d(x,x) = 0$
- Simmetry: $d(x,y) = d(y,x)$
- Triangular Inequality:

  $d(x,y) \leq d(x,z) + d(z,y)$



Dissimilarity measures (continuous variables)

a) **Minkowski Distance or Lp norm.**

$$D_p(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{M}(x_i - y_i)^p\right)^{1/p}$$

Particular cases:

- Euclidean Distance: p=2,

$$D_2 = \sqrt{\left(\sum_{i=1}^{M}(x_i - y_i)^2\right)}$$

- Manhattan or City-Block Distance:

$$D_1 = \sum_{i=1}^{M}|x_i - y_i|$$

- Chebychec Distance, p=∞,

$$D_\infty = \max_{1\le i\le M}|x_i - y_i|$$

Weighted Minkoswki Distance:

$$D_p(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{M} w_i(x_i - y_i)^p\right)^{1/p}$$

Dissimilarity measures (Cont)

b) Distances based on Quadratic forms.

Q=(Q_{ij}) is a definite positive square matriz of order MxM containing weights, then the quadratic distance between x y y is given by:

$$DQ(x,y) = [(x-y)'Q(x-y)]^{1/2} = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{M}(x_i - y_i)Q_{ij}(x_j - y_j)}$$

When $Q=V^{-1}$, where V is the covariance matrix between x and y, then DQ is called the Mahalanobis distance.

c) Camberra Distance.

$$D_{Can}(x,y) = \sum_{i=1}^{M}\frac{|x_i - y_i|}{|x_i + y_i|}$$

when x_i y y_i are both zeros then the i-th term of the sum is considered as zero.

Dissimilarity measures(nominal variables)

Hamming Distance. Let **x** an **y** be two vectors with the same dimension and taking values in Ω={0,1,2,……..k-1}, then the Hamming Distance (D_H) between them is defined as the number of different entries at the same position. Thus, if x=(1,1,0,0) and y=(0,1,0,1) then D_H=2.

- Also, it can be used for non-binary vectors. For example, if x=(0,1,3,2,1,0,1) and y=(1,1,2,2,3,0,2) then DH(**x,y**)=4 .
- In case of binary vectors , the Hamming distance, the L2 distance and the L1 distance are the same.

Similarity measures (continuous variables))

Similarity=1-dissimilarity

a) The correlación measure:

$$r(x,y)=\frac{\sum_{i=1}^{M}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{M}(x_i-\bar{x})^2\sum_{i=1}^{M}(y_i-\bar{y})^2}} \quad =\frac{(x-\bar{x})'(y-\bar{y})}{\|x-\bar{x}\|\|y-\bar{y}\|}$$

Note: 1-s(x,y) can be considered as a dissimilaty measure. The cosine distance is similar to the correlation distance but in this case the sample means for x and y are taking as zeros.

b) The Tanimoto measure. It is defined by

$$S_T(x,y)=\frac{x'y}{\|x\|^2+\|y\|^2-x'y}$$

Also, it can be used with nominal variables.

Similarity measures (nominal variables)

i) The Tanimoto measure. Let X and Y be two vectors of lenght nx and ny respectively. Let $n_{X \cap Y}$ be the cardinality of the intersection of X and Y, then the Tanimoto measure is defined by:

$$\frac{n_{X \cap Y}}{n_X + n_Y - n_{X \cap Y}}$$

This, the Tanimono measure is the ratio between the number of elements in common to the two vector, divided by the number of elments of the union. In the case of binary variables, the Tanimoto becomes

$$\frac{a+d}{a+2(c+b)+d}$$

where a represents the number of positions where both X and Y have the same value 0, d represents the number of coincidences where both X and Y take the value 1. While c and b represent the number of non-coincidences.

Similarity measures (nominal variables)

ii) **The coeficient of simple coincidences**. Defined by

$$\frac{a+d}{a+b+c+d}$$

iii) **The Jaccard-Tanimoto measure**. Defined by

$$\frac{d}{b+c+d}$$

iv) **The Russel –Rao measure**.

$$\frac{d}{a+b+c+d}$$

v) **The Dice-Czekanowski measure**. Similar to Jaccard-Tanimoto measure but assign doublé weight to the coincidences. That is,

$$\frac{2d}{b+c+2d}$$

Libraries and functions to compute distances in Python

The function pairwise_distances of sklearn.metrics.pairwise computes several distances including:
['cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan'], and from scipy.spatial.distance: ['braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule']

Example: Distances on Diabetes

```
url= "http://academic.uprm.edu/eacuna/diabetes.dat"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pd.read_table(url, names=names)
X=data.iloc[:,0:8]
#Normalizing the predictors
scaler = StandardScaler()
scaler.fit(X)
X= scaler.transform(X)
#Computing the Euclidean Distance
dist2=pairwise_distances(X)
```

Libraries and functions to compute distances in Python

```
# Manhattan distance, also called CityBlock
dist_maha=pairwise_distances(X,metric="cityblock")
dist_maha
#Mahalanobois Distance
pairwise_distances(X,metric='mahalanobis',V=np.cov(X))
#Correlation Distance  but between rows of Diabetes
distcor1=pairwise_distances(X,metric="correlation")
Distcor1
```

The class sklearn.neighbors has a function DistanceMetric for computing several distances

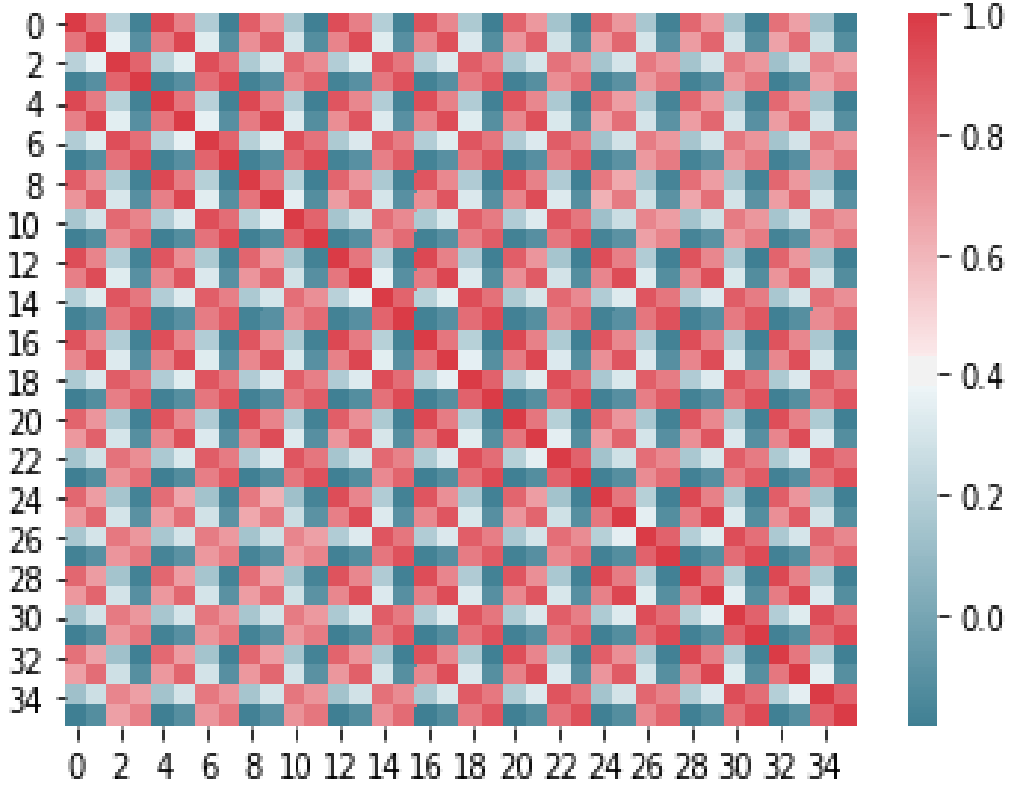
```
#Euclidean distance as a particular case of the Minkowski distance
dist_e=DistanceMetric.get_metric("minkowski",p=2)
dist_e.pairwise(X)
```

Heatmaps

Heatmaps are useful to visualize the values of a distance metric in a dataset, since it is not easier to appreciate them in the numerical matrix.

Heatmap can be obtained using either matplotlib or seaborn

Here, we show the heatmap of the correlations among the features of landsat



Types of Clustering Algorithms

I. Methods based on Partitioning (Kmeans, PAM,SOM)
II. Hierarchical Methods.
III. Methods based on density (DBSCAN): It groups in the same cluster the instances that have a lot of nearest neighbors. Outliers are identified.
IV. Methods based on Models (Gaussian Mixtures)
V. Methods based on Grids (STING, CLIQUE)

I. Partitioning Methods

The dataset is partitioned in a pre-specified number K of clusters, and then iteratively the data points are re-assigned to a cluster until a stopping criterion is satisfied. Usually until a function such as the sum of squares within each clusters is minimized.

Examples: **K-means, PAM,** CLARA, **SOM**, Clustering based on Gaussian Mixtures, Fuzzy K-means etc.

1-The k-means Algorithm (MacQueen, 1967).

The goal is to minimize the dissimilarity of the data points within each cluster and maximize the dissimilarity of the data points belonging to distinct clusters.

INPUT: A dataset S and the number of clusters k to be formed;

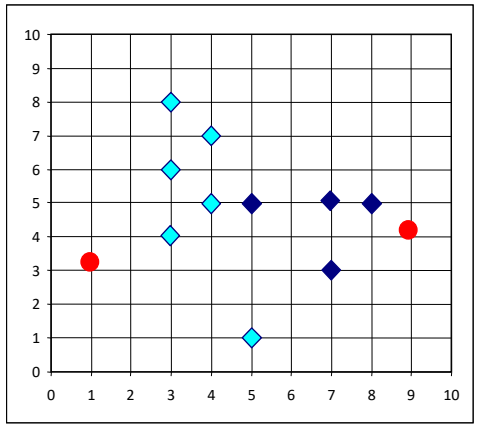
OUTPUT: A list L of the clusters where each data point of S is assigned.

1. Select the initial centroids of the K clusters: c_1, c_2, ..., c_K.
2. Assign each data point x_i of S to the cluster C_i , which centroid c_i is closer to x_i. That is, d $C_i=\text{argmin}_{1\leq k\leq K}\|\mathbf{x}_i-\mathbf{c}_k\|$
3. For each of the clusters, the centroid is recalculated based on the data points conatined in the cluster and minimizing the sum of squares within the cluster. That is,

$$WSS = \sum_{k=1}^{K} \sum_{C(i)=k} \| \mathbf{x}_i - \mathbf{c}_k \|^2$$

Go to step 2, until convergence is achieved.

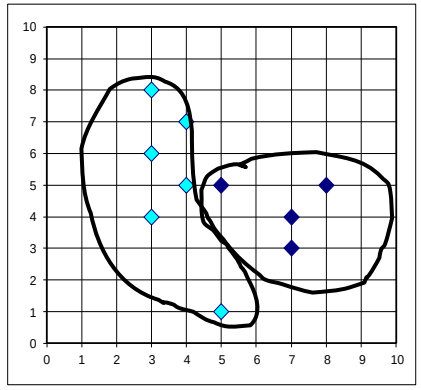
K-Means



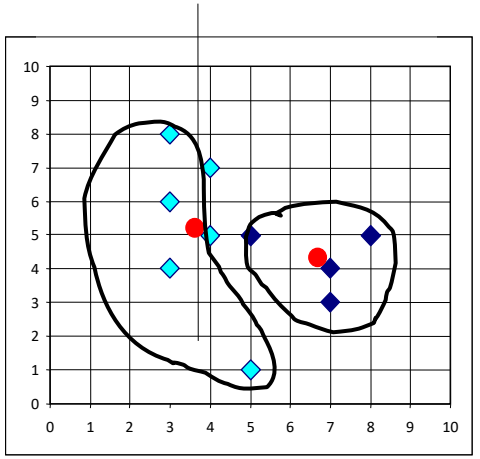
K=2

Choose arbitrarialy K clusters and identify their centroids

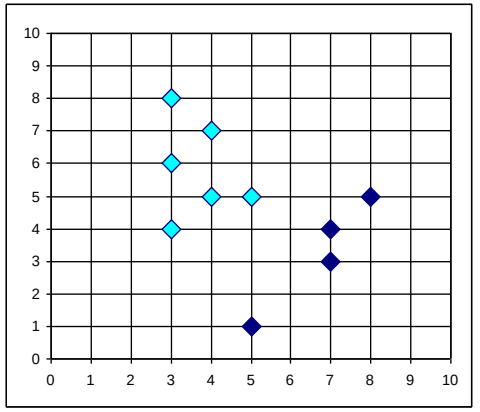
Assign each data point to the closer centroid.



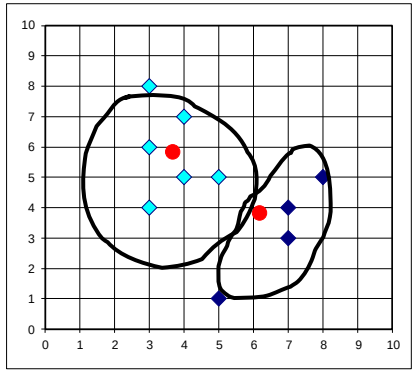
Re-calculate the centroids



Re-assigne



Re-calculate the centroids



Re-assigne

Alternatives for choosing the k initial centroids?

- Using the k first instances.
- Choosing randomly k instances.
- Taking ramdomly any partititon into k clusters and calculating theirs centroids.
- Using the k-means++ method. D. Arthur and S. Vassilvitskii (2007). k-means++: the advantages of careful seeding. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. SODA's 07. pp. 1027-1035.
- Kmeans from scikit-learn uses this last method.

The k-means algorithm : Pro and cons

- **Pros:**
- The k-means algorithm is computationlly fast.
- It works fine with data containing missing values.
- Complexity: O(nkpi), n:number of instances, k=number of clusters, p:number of predictors, i:number of iterations. But for large datasets kmeans is basically O(n).
- **Cons:**
- The optimization criterion it is not satisfied globally, only a local optimum is achieved.
- It is sensible to outliers.

kmeans in python

The class **sklearn.cluster** includes a function **KMeans** **for** performing the kmeans algorithm.

Example: K-means applied to diabetes(ignoring the class column)

```
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
clustlabels=kmeans.labels_
print clustlabels
#Choosing the two centroids at random
kmeans = KMeans(n_clusters=2, init='random', random_state=0).fit(X)
clustlabels1=kmeans.labels_
print clustlabels1
```

Partitioning around medoids (PAM)

Introduced by Kauffman and Rousseauw, 1987.

MEDOIDES, are data points representative of the clusters that are beeing formed

For a pre-specified number of clusters K, the PAM algorithm is based on the search of the K MEDOIDES,

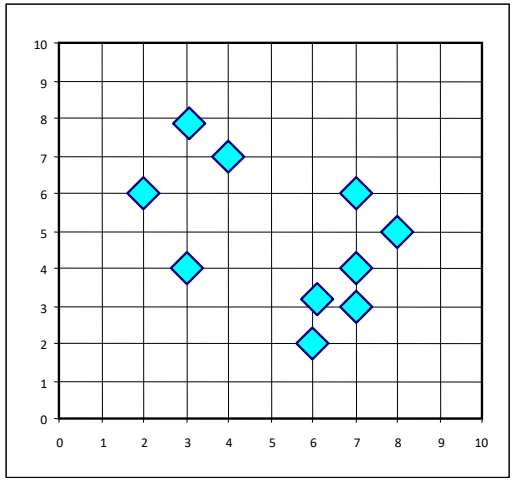
$\mathbf{M} = (\mathbf{m}_1, \ldots, \mathbf{m}_K)$ of all the instances.

In order to find M, the sum of the distances of the data points to their closest medoid must be minimized. That is,

$$M^* = \arg\min_M \sum_i \min_k d(x_i, m_k)$$

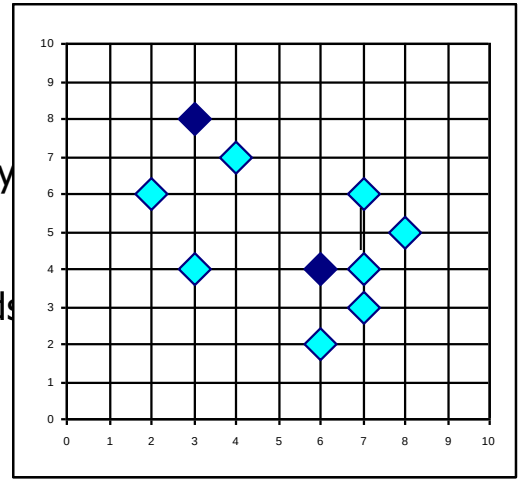
where d is a dissimilarity measure

k-Medoides



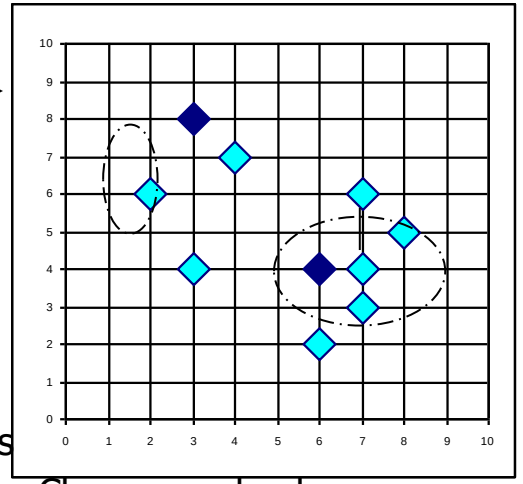
K=2

Arbitrarialy choose k instances as medoids

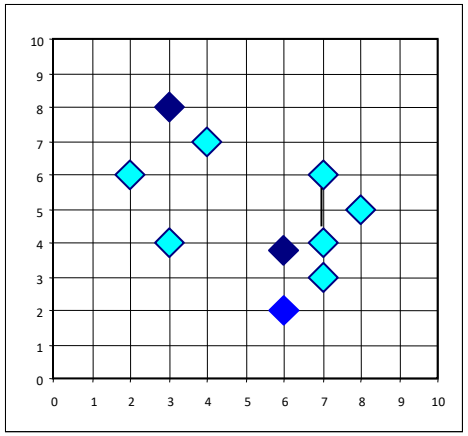


Assign the data pitns to the cluster where is located its closer medoide.

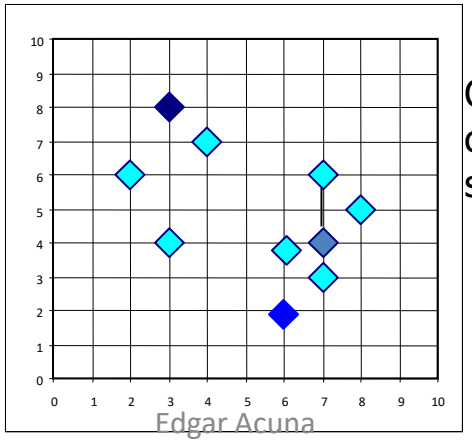
Total Cost = 20



Choose randomly a nonmedoide datapoint ,O_{ramdom}



Calculate the cost total of swapping



Swapping O an O_{ramdom}

If the Total cost decreases.

Total Cost = 26

Continue the loop until the objective function does not change lo

PAM in Python

The scikit_extra library includes a cluster class with a function kmedoids for perfoming PAM. Also, the Biopython includes a a Bio.Cluster library, where PAM is included

To apply PAM to the Diabetes dataset:

```
from Bio.Cluster import kmedoids
clusterid,error,c=kmedoids(dist2,nclusters=2)
#print(clusterid)
print('the within-cluster sum of distances for the returned PAM=',error)
unique_elements, counts_elements = np.unique(clusterid, return_counts=True)
print("Tamano de los two clusters formados")
print(np.asarray((unique_elements, counts_elements)))
```

PAM's complexity:$O(k(n-k)^2)$. It is too heavy for large datasets . There are two PAM versions CLARA and CLARANS for large datasets , but still the computation is not fast.

Self-organizing Maps, SOM (Kohonen, 1988)

SOM is a partitioning algorithm with the restriction that the clusters must be represented in a regular structure of low dimension such as a grid. SOM uses mainly one and two-dimensional grids.

The cluster that are closer among them appear in adjacent cells of the grid. This means SOM maps the sample space into a space of lower dimensión in which the similarity measure among the instances is measured by the close relationship of neighbors.

Each of the K clusters is represented by a prototype Mi , i = 1,…K .

Self-organizing Maps, (SOM)

In this method the initial position of the data point in the sample space affects somehow its assigment to a cluster.

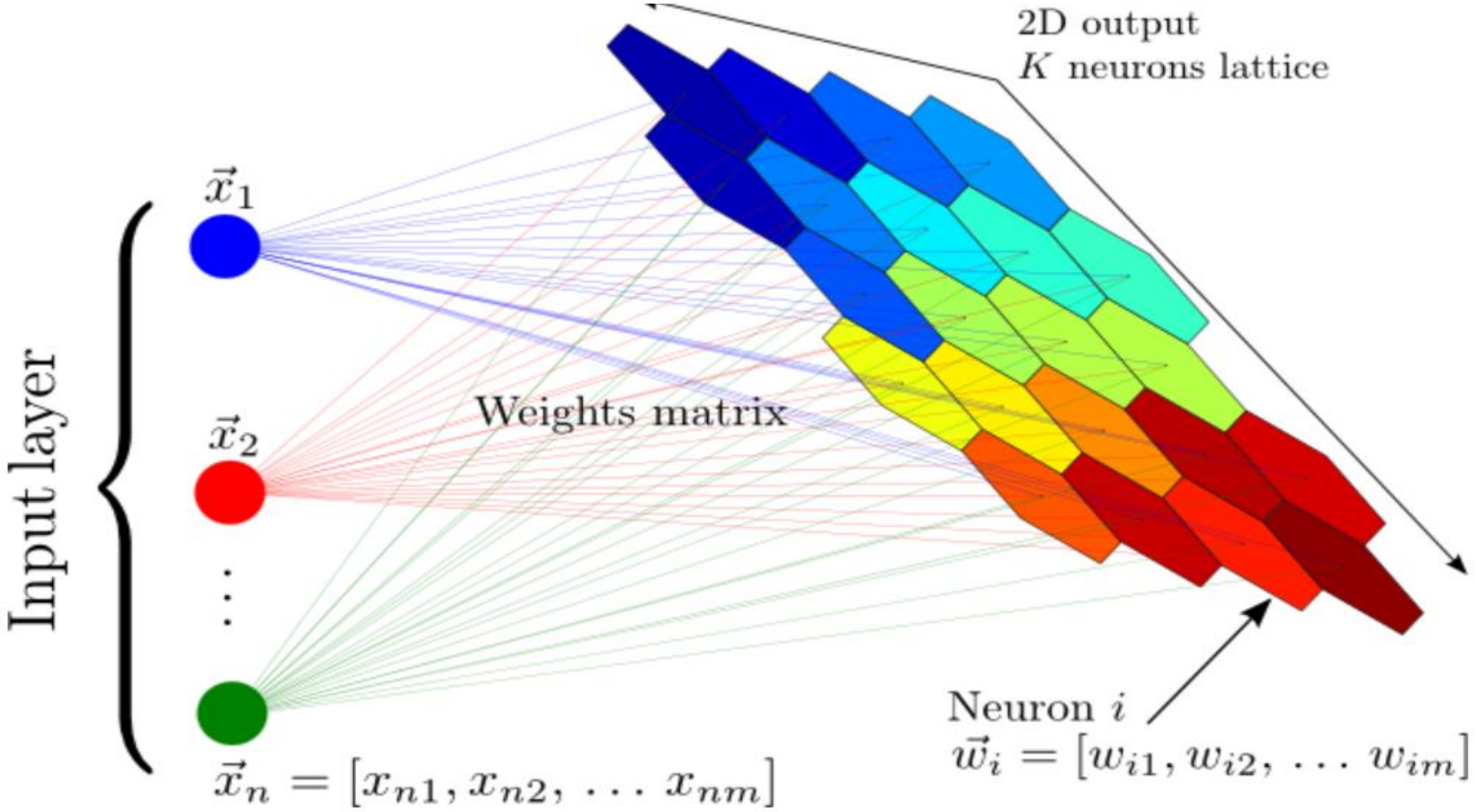
A SOM can be visualized and the datapoints that are similar among appear close in the plot.

The SOM algorithm is quite similar to k-means.

A SOM is trained as a neural network.

The SOM can also be considered as a dimensionality reduction technique. It is closed related to multidimensionalidad scaling (process to find a set of data pints in a multidimesional space that are similar among them).

Self-organizing Maps, (SOM)



SOM Algorithm considering a bidimensional rectangular grid

Step 1-Select the number of rows (q_1) and columns (q_2) to have in the grid. Therefore, the data will be clustered in $K=q_1q_2$ groups.

Step 2. Initialize the updating parameter(the learning rate in terms of neural nets) α (α=1) and the radius of the grid (r=2)

Step 3. Initialize the prototype vectors M_j, j ε (1,…,q1) x (1,….,q2) by selecting randomly K instancias.

Step 4. for each data point xi of the dataset do the following:

Identify the index j' of the prototype Mj that is closer to xi.

Identify a set S of neighbors prototyes of $M_{j'}$. That is,

$S=\{j: \text{distance } (j, j')<r\}$

The distance can be Euclidean or any other one.

Update each element of S moving the correspondimg prototype towards x_i:

$M_j \leftarrow M_j + \alpha(x_i - M_j)$ for each j ε S

Step 5. Decrease the values of α and r in a predetermined amount and continue the iteration until reach convergence.

SOM in Python

The scikit-som library includes the SOM algorithm. There is a function somclsuster in the library Bio.Cluster. Also there is a SuSi library, that includes SOM.

Other software: Somtoolbox en Matlab, Cluster by Eissen and Genecluster (MIT Center por Genome Research). R has at least 4 libraries for SOM. Rapidminer also performs som

SOM using Bio.Cluster

```
import BioCluster
clusterid,error,c=Pycluster.somcluster(Xd,nxgrid=1,nygrid=2)
clus=pd.DataFrame(clusterid)
#comparing the clusters with the actual classes
pd.crosstab(clus.sum(axis=1),yd)
```

class	1	2
row_0		
0	186	141
1	314	127

II. Hierachical algorithms for clustering

In this type of algorithms ordered sequences (Hierachy) of clusters are generated. It can be done merging smaller clusters or dividing larger clusters into smaller ones.The hierachical structure is represented as a tree plot and it is called and **Dendrogram**.

There are are two type:

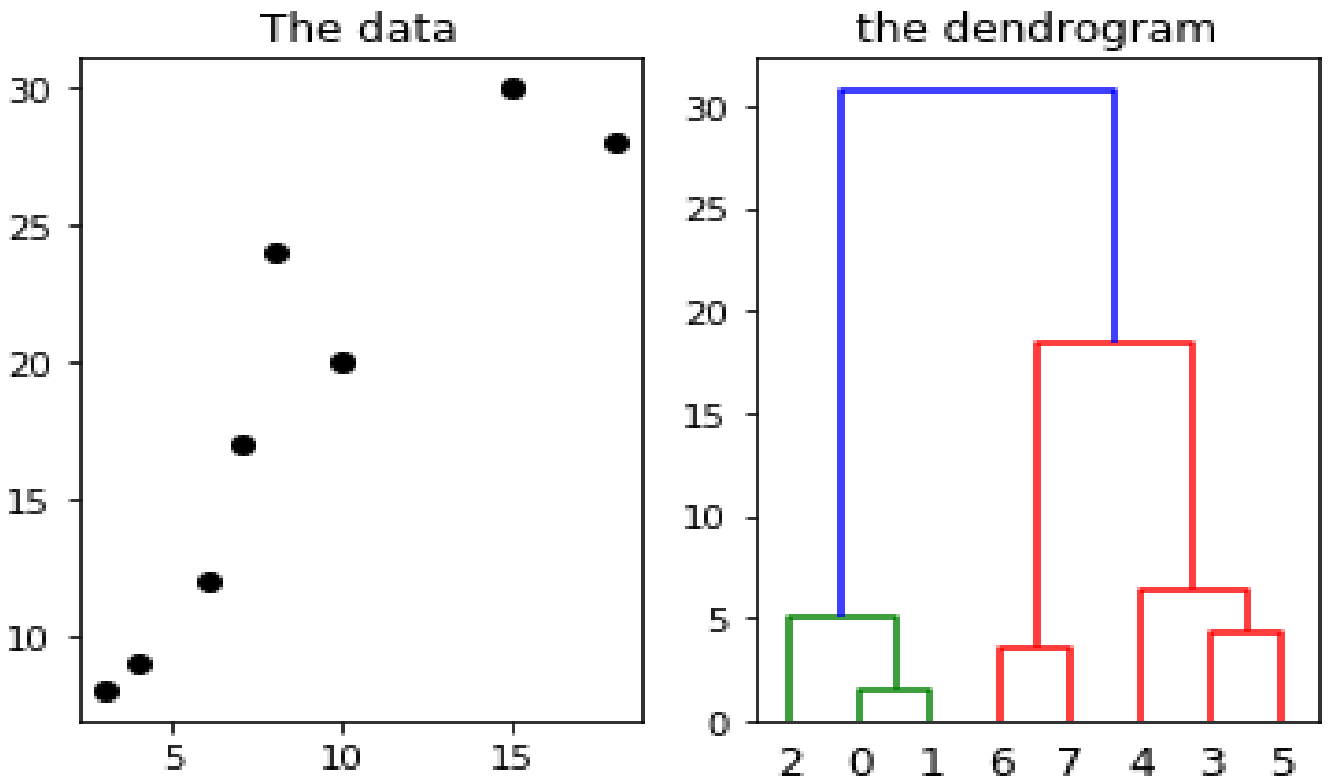
Agglomerative hierachical algorithms (bottom-up, at the beginning each data point is a single).

Divisive hierachical algorithms (top-down, at the beginning all the data point form one cluster).

The python libraries scikit-learn and scipy have function to carry out only the Agglomerative hierarchical algorithm

Basic example

The Data
x=[3, 4, 6, 7, 8, 10, 15, 18]
y=[8, 9, 12, 17, 24, 20, 30, 28]
df=list(zip(x,y))
df=pd.DataFrame(df,columns=['x','y'])
dist2=pairwise_distances(df)
Z = hierarchy.linkage(df,'ward')



Distance between clusters=Linkage functions

After choosing the dissimilarity measure, the data matrix n x p (n data points, p features) is transformed in distance matrix or dissimilarity matrix D = (d_{ij}) of order n x n for the data points to be grouped.

Based on this matrix, a distance measure between two clusters S and T must be computed. The following are some of this distance measures:

Single Linkage: $\delta(S,T)=\min_{\{x\in S, y\in T\}} d(x,y)$

Complete Linkage: $\delta(S,T)=\max_{\{x\in S, y\in T\}} d(x,y)$

Average Linkage:

$$\partial(S,T) = \frac{1}{|S||T|}\sum_{x\in S, y\in T} d(x,y)$$

where |S|, and |T| represent the cardinality of S and T, respectively.

Centroide Linkage : $\delta(S,T)= d(\bar{x},\bar{y})$

where $\bar{x}$ and $\bar{y}$ represent the centroides of S and T respectively.

Distance between clusters=Linkage functions

Median Linkage: $\delta(S,T)=\text{median}_{\{x\in S, y\in T\}} d(x,y)$

Mc Quitty Linkage:

$$\partial(S,T)=\frac{1}{|S|+|T|}[\sum_{x\in S} d(x,\bar{x})+\sum_{y\in T} d(y,\bar{y})$$

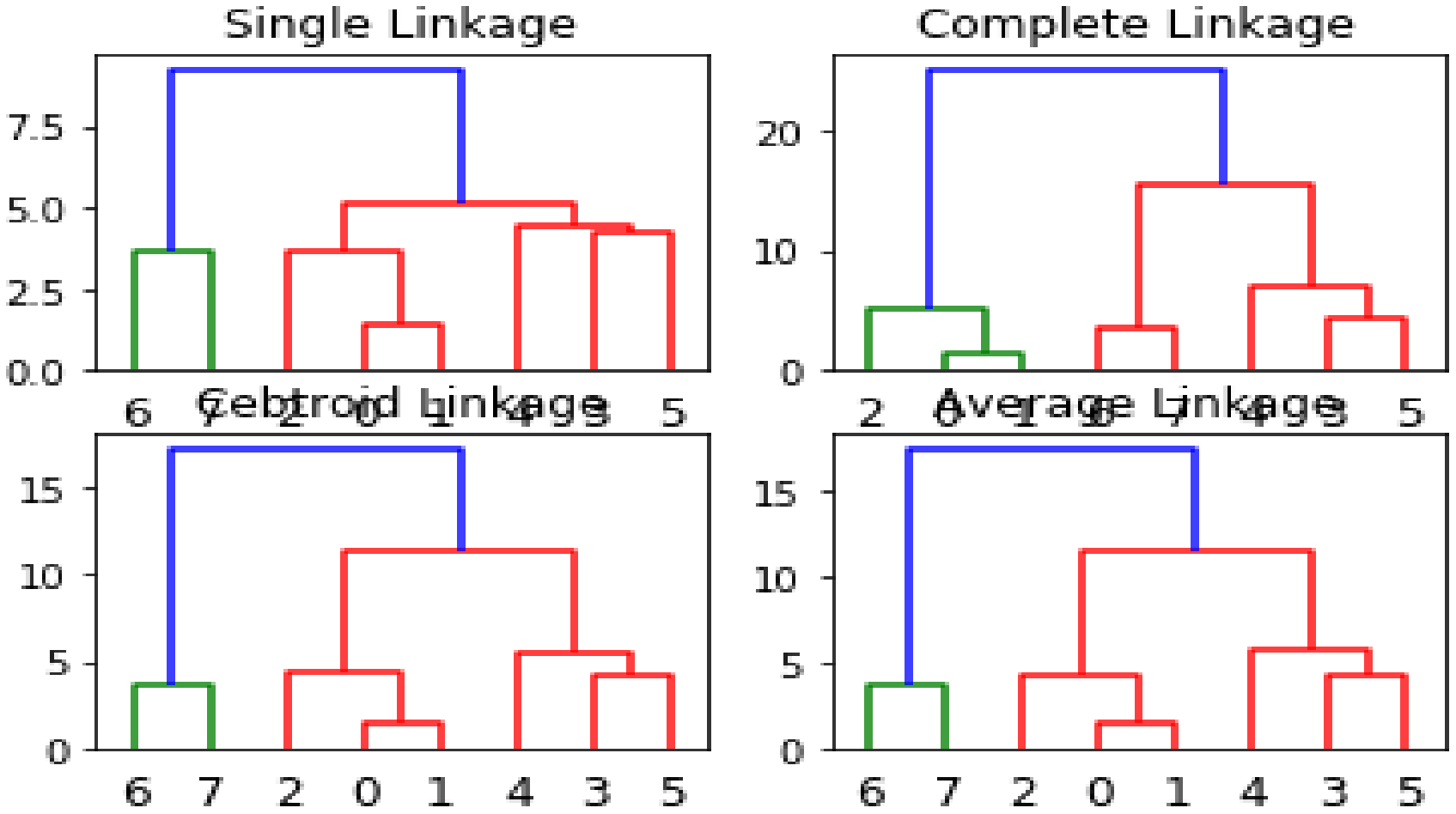
Ward Linkage:

$$\partial(S,T)=\sum_{x\in S} d^2(x,\bar{x})+\sum_{y\in T} d^2(y,\bar{y})$$

Ward linkage merges the pair of clusters the gives the smallest variance in the merged group.

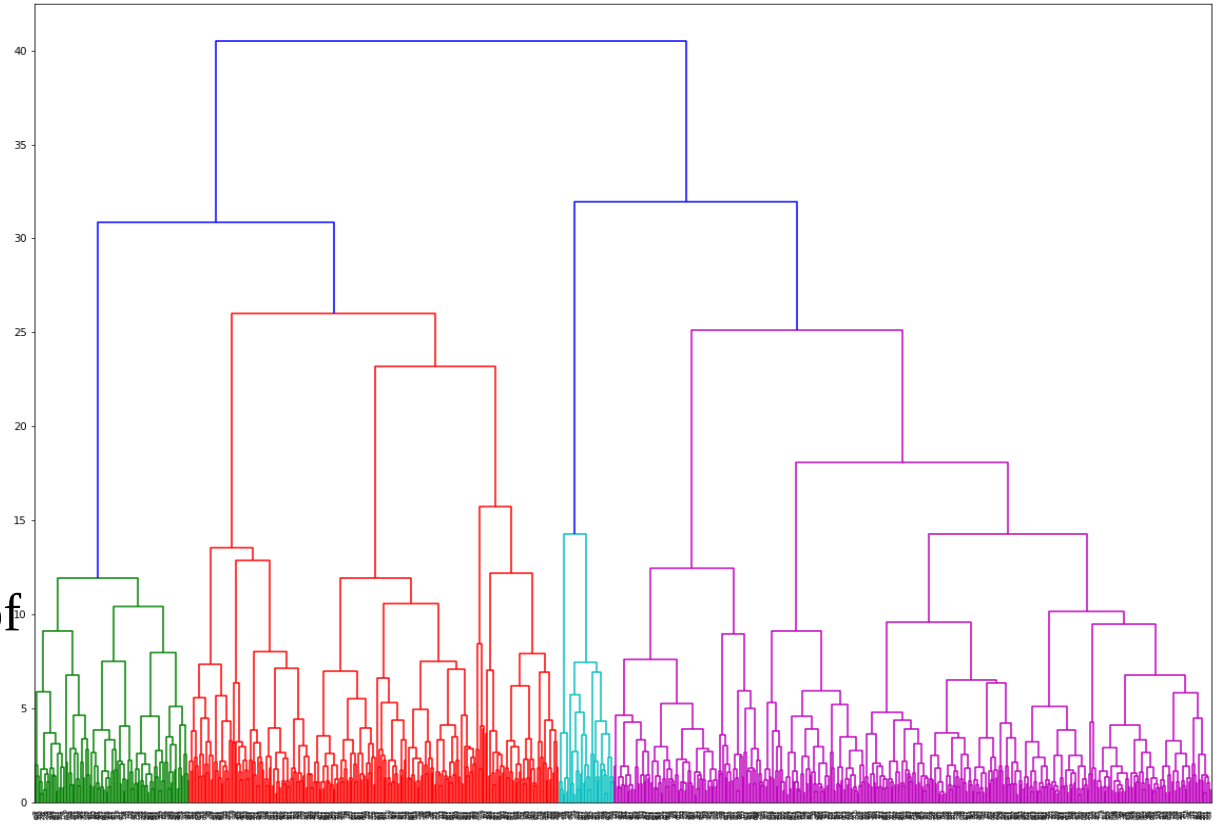
Scikit-learn considers all of the above linkage functions.

Distance between clusters=Linkage functions



Hierarchical Algorithm applied to Diabetes

Since Diabetes is supervised, we will apply clustering only to the predictor matrix, X.
First, we must apply a distance measure X(frequently the data is standarized) usually the euclidean distance.
Second, we apply the hierarchical algorithm to this distance matrix using a linkage function
Z = hierarchy.linkage(X,'ward')
The algorithm does not require the number of clusters as a parameter. The clustering assignment is done later on.



Dendrograms

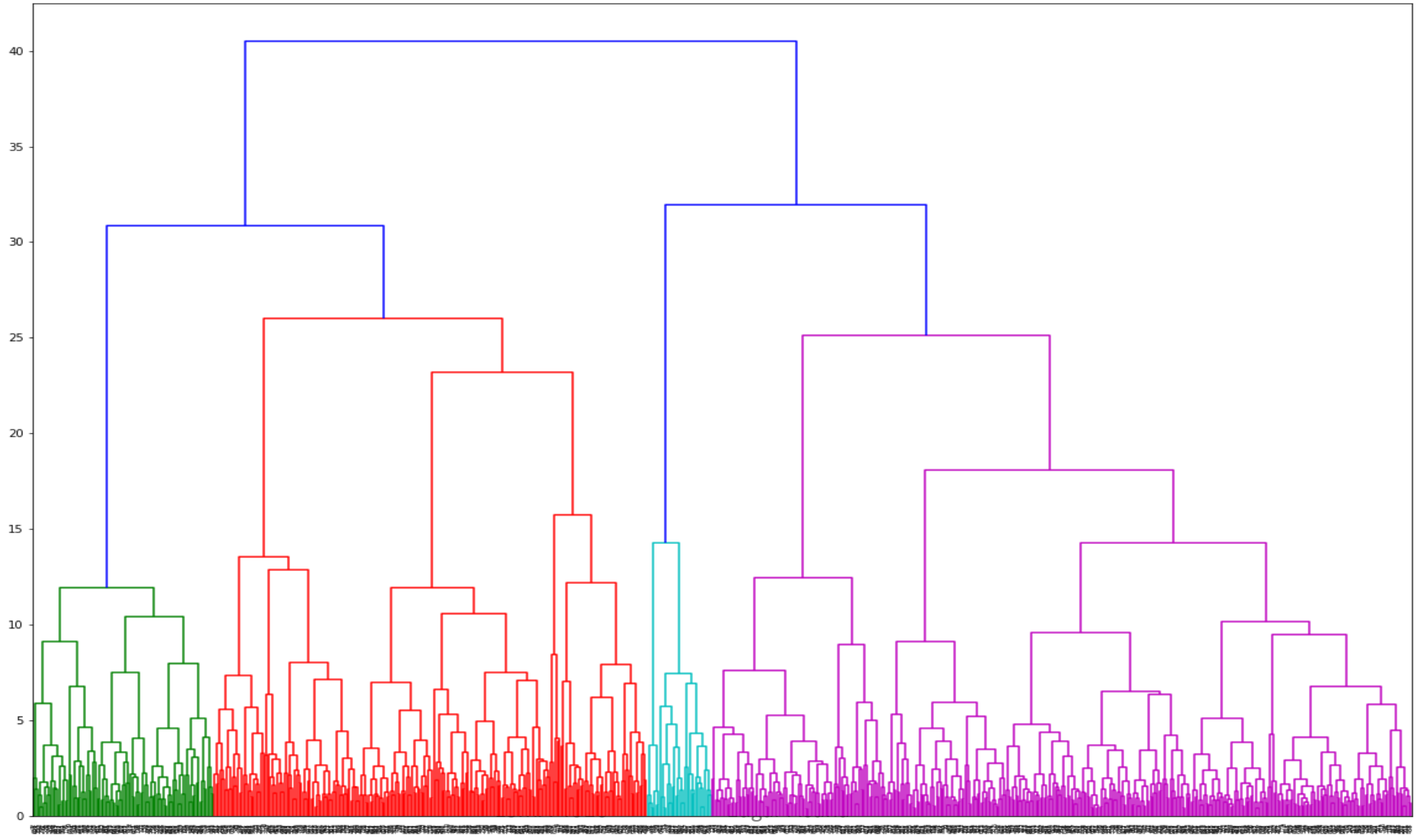
The dendrograms are easy to understand but they can lead to wrong conclusions for the following reasons:

1) The corresponding dendrogram to a hierarchical clustering is not unique, since for each merge of clusters one needs to specify which subtree goes to right and which to the left.

   Most of the programs order the trees in such way that clusters including more data points go to the right.

2) The hierarchical structure of the dendrogram does not represent with certainty the true distances between the distinct data points of the dataset.

Dendrogram for Diabetes (Ward Linkage)



Dendrogram

The cophenetic correlation coefficient can be used as a measure of how well the hierachical structure of the dendrogram represents the actual distances. It is defined as the correlation between the $n(n-1)/2$ pairs of dissimilarities and theirs cophenetic distances in the dendrogram.

The library scipy allow us to compute the cophenetic correlation coefficient.

For the example,

```
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import cophenet
from scipy.spatial.distance import pdist

Z = linkage(df, 'ward')
c, coph_dists = cophenet(Z, pdist(df))
print c
```

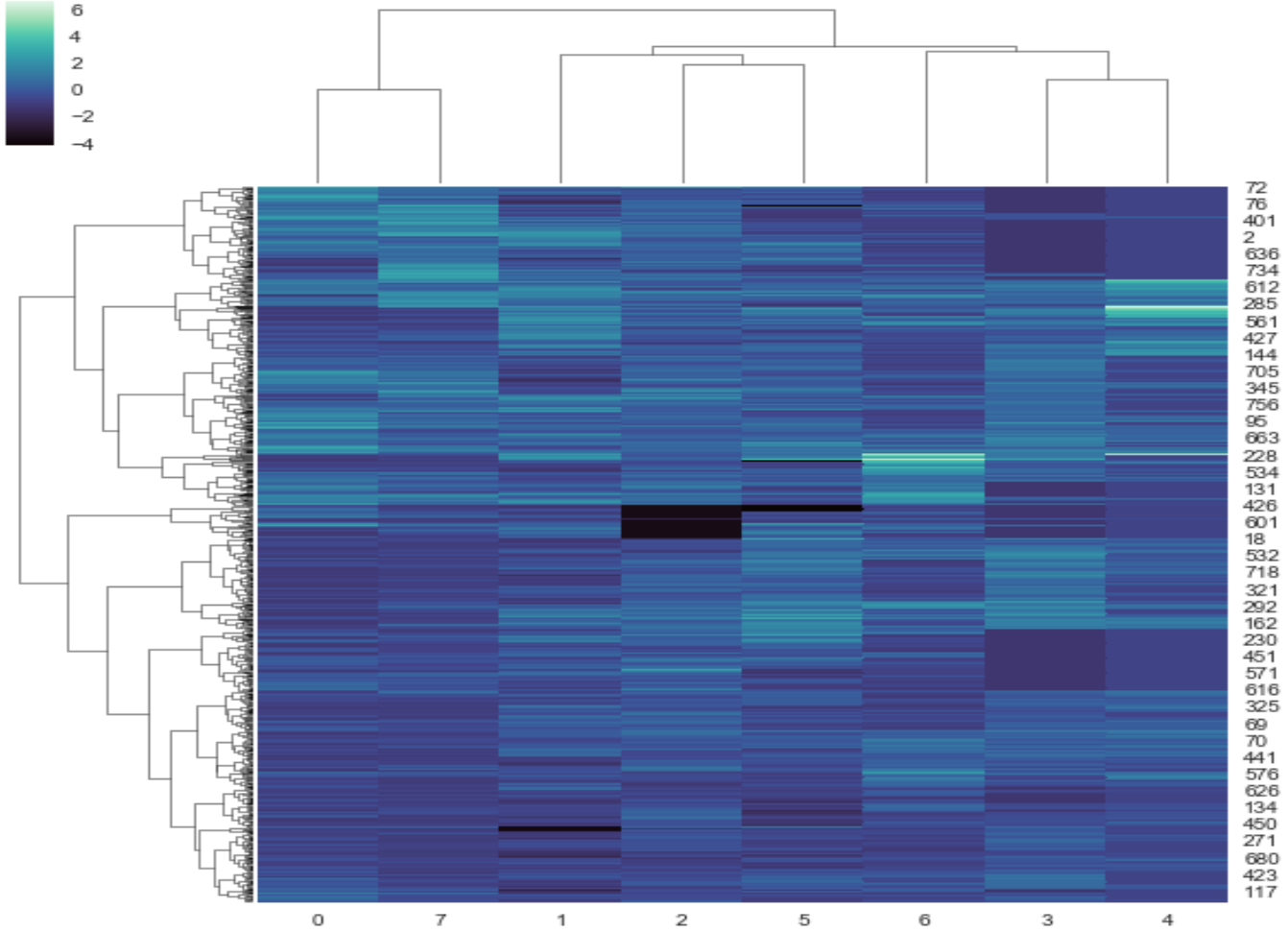
0.708252468327

Heatmaps.

Heatmaps can be used to show simultaneously the cluster of rows and columns.

The visualization library seaborn has nice heatmaps

Heatmap for Diabetes



Hierarchical Agglomerative Clustering for Diabetes

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import pairwise_distances
n_clusters = 2
model = AgglomerativeClustering(n_clusters=n_clusters,linkage="average")
model.fit(X)
clustlabels=model.fit_predict(X)
unique_elements, counts_elements = np.unique(clustlabels, return_counts=True)
print("Size of the two clusters")
print(np.asarray((unique_elements, counts_elements)))
```

```
Size of the two clusters
[[ 0 1]
 [761 7]]
```

The average linkage is very affected by outliers

Hierarchical Agglomerative Clustering for Diabetes

```
model = AgglomerativeClustering(n_clusters=n_clusters,linkage="ward")
model.fit(X)
clustlabels=model.fit_predict(X)
unique_elements, counts_elements = np.unique(clustlabels, return_counts=True)
print("Size of the two clusters")
print(np.asarray((unique_elements, counts_elements)))
```

```
Size of gthe two clusters
 [[ 0 1]
[426 342]]
```

We get much better results using the Ward Linkage

Partitioning Algorithms versus Hierarchical Algorithms

The partitioning methods have the advantage that a optimality criterion is satisfied at least approximately. They are fast and can be parallelized. The main disadavantage is that they need a number of clusters to start working.

On other hand, the hierarchical algorithms have a time complexity of at least $O(n^2)$. The agglomerative are much faster to compute than the divisive.

Other disadvantage of these methods is that theey suffer of the nesting problema. The tree structure is rigid, and it is not easy make corrections to the clustering process.