16. CLUSTERING

- Clustering refers to a set of methods for finding natural groups or clusters in a data set. The idea is that groups should contain objects similar to each other, and the groups should be as different as possible.

  Uncovering natural groups may lead to a better understanding of a dataset, showing relationships between the observations or effectively creating a classification of the observations, or can be a pre-processing step and the groups found are further investigated as part of some other analysis.

- Clustering is usually designed to find groups of observations, but can be used to find groups of variables as well. Both clustering and PCA seek to simplify the data via a small number of summaries, but their mechanisms are different:

  - PCA looks to find a low-dimensional representation of the observations that explain a good fraction of the variance
  - clustering looks to find homogeneous groups among observations

- Clustering has a large number of applications spread across various domains:

  - *for market segmentation (or customer behavior) and recommendation engines*

    The customers are clustered based on their purchases and their activities.

    This is useful in recommendation engines to suggest content that other users in the same cluster enjoyed.

  - *for semi-supervised learning*

    If you have only partially labeled data set with only few labels, you can perform clustering and propagate the labels to all the instances in the same cluster. This can greatly increase the number of labels available for a subsequent supervised learning algorithm and improve performance.

  - *as a dimensionality reduction technique*

    When the data is clustered, we can measure each data instance's affinity to each cluster (affinity is a measure of how well the data instance fits in a cluster). We can replace feature vector for each data instance by the vector of cluster affinities. The size of this vector depends on the number of clusters and it is typically much lower than the number of original features, but it can preserve enough information for further processing.

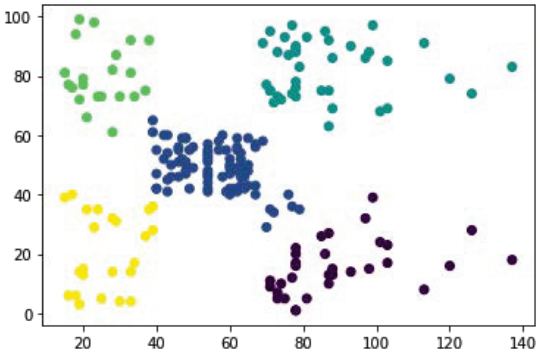
  - *for anomaly detection*

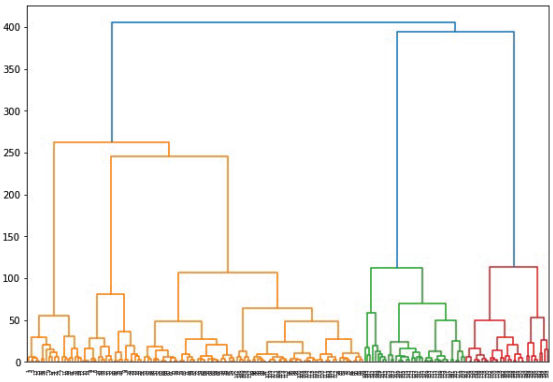
    Any instance that has a low affinity to all the clusters is likely to be an anomaly. Anomaly detection is useful in detecting defects in manufacturing or for fraud detection.

  - *for image segmentation*

    By clustering pixels according to their color and then replacing each pixel's color with the mean color of its cluster, it is possible to considerably reduce the number of different colors in the image.

- Since clustering is popular in many different fields, there exists more than 100 clustering methods. The best known methods include:
  - partitioning methods
    - *centroid methods*: These are iterative clustering algorithms in which the number of clusters is predifined and the notion of similarity is derived by the closeness of a data point to the centroid of the clusters.
      Examples: K-means, bisecting K-means, K-medoids, etc.
    - *connectivity models*: These methods are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. We do not know in advance how many clusters we want and we end up with a tree-like visual representations of the observations, called dendrogram, that allows us to view at once the clusterings obtained for each number of clusters.
      Examples: hierarchical clustering (agglomerative or divisive)
      - in the agglomerative or "bottom-up" approach, we start by considering each data point in its own separate cluster and then we iteratively aggregate clusters as the distance between them decreases
      - in the divisive or "top-down" approach, all data points are grouped in a single cluster which is then partitioned as the distance increases





  - density-based methods: These methods search the data space for areas of varied density of data points. They isolate various different density regions and assign the data points within these regions in the same cluster.
    Examples:
    - DBSCAN – Density Based Spatial Clustering of Applications with Noise
    - HDBSCAN – Hierarchical DBSCAN
    - OPTICS – Ordering Points To Identify the Clustering Structure
  - model-based methods: These methods make a strong assumption that the underlying population is actually a collection (mixture) of subpopulations (the clusters) and that they have multivariate probability density functions that enable them to be distinguishable from each other.

- Clustering quality is measured by:
  - maximizing inter-cluster distance, i.e., the space between clusters
  - minimizing intra-cluster distance, i.e., the space between members of the same cluster

  Quality of the clustering depends on the method, dissimilarity measure, and application.

I. Dissimilarities and Distances

- A dissimilarity (proximity) is any function $d : \mathbb{R}^d \times \mathbb{R}^d \to [0, \infty)$ that for every $x, y \in \mathbb{R}^d$ satisfies

$$d(x, x) = 0$$
$$d(x, y) = d(y, x)$$

  If, in addition, for every $z \in \mathbb{R}^d$, we have

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

  the dissimilarity function is called distance.

- Some common distance functions are:
  - Euclidean distance
$$d(x, y) = \left(\sum_{i=1}^{d} (x_i - y_i)^2 \right)^{1/2}$$
  - squared Euclidean distance
$$d(x, y) = \sum_{i=1}^{d} (x_i - y_i)^2$$
  - Manhattan or city block distance
$$d(x, y) = \sum_{i=1}^{d} |x_i - y_i|$$
  - Minkowski p-distance
$$d(x, y) = \left(\sum_{i=1}^{d} (x_i - y_i)^p \right)^{1/p}$$

- If features are not scaled and are on very different scales, a feature with a large range may dominate the distance measure and hence be the main driver of the clustering solution. We can resolve this by:
  - using a scaler (Min-Max Scaler or Standard Scaler)
  - giving different weights to features so that all features have equal influence in characterizing overall dissimilarity between data points

II. K-Means Clustering

- The algorithm was proposed by Lloyd at Bell Labs in 1957 (published outside of the company in 1982) and by Forgy in 1965.
- **Example:** Let's review definition of the average for a 1-dimensional data set and let's define the centroid for a 2-dimensional data set.
  - Find the average of $\{1, 4, 6, 8, 10, 12\}$.

$$\overline{X} = \frac{41}{6} = 6.83$$

  - Find the average (centroid) of $\{(1,3), (3,2), (2,4), (3,5), (4,5)\}$.
    $\overline{X}$ = average of $\{1, 3, 2, 3, 4\} = 2.6$
    $\overline{Y}$ = average of $\{3, 2, 4, 5, 5\} = 3.8$

$$\text{centroid} = (\overline{X}, \overline{Y}) = (2.6, 3.8)$$

□

- A simple approach for partitioning a dataset $\{x^{(1)}, \ldots, x^{(n)}\} \subset \mathbb{R}^d$ into K distinct non-overlapping clusters. Let $C_1, \ldots, C_K$ denote sets containing the indices of the observations in each cluster.
  - $C_1 \cup \ldots \cup C_K = \{1, \ldots, n\}$ (each observation belongs to at least one cluster)
  - $C_k \cap C_{k'} = \emptyset$, for all $k \neq k'$ (clusters are non-overlapping)

- A good clustering is one for which the within-cluster variation is as small as possible. Hence, we are looking for a partition that solves the optimization problem

$$\text{minimize}_{C_1,\ldots,C_k} \sum_{k=1}^{K} W(C_k)$$

The most common within-cluster variation involves squared Euclidean distance and is given by

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{d} \left(x_j^{(i)} - x_j^{(i')}\right)^2$$

where $|C_k|$ is the number of observation in the k-th cluster.

Therefore, the K-means clustering is the partition that solves the problem

$$\boxed{\text{minimize}_{C_1,\ldots,C_k} \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{d} \left(x_j^{(i)} - x_j^{(i')}\right)^2} \tag{1}$$

This is a very difficult optimization problem to solve since the number of possible partitions is immense and the objective function is not convex.

Number of data observations n	Number of clusters K	Number of partitions
8	2	127
10	3	9,330
15	4	42,344,950
21	5	3,791,262,568,401

- Instead of solving the above problem exactly, an iterative approach is used that provides a local optimum solution.

Steps in K-means algorithm:

1. Set a value for K and the distance measure being used.
2. Initialize randomly K starting cluster centroids; these may be actual data points or random points.
3. Assign each data observation $x^{(i)}$ to the cluster C_k that it is closest to (measured by its distance to the cluster centroid)
4. Update the cluster centroids for all K clusters based on the updated cluster assignments. The cluster centroids are found as the average of all observations in that cluster. For C_k, the cluster centroid is
$$\frac{1}{|C_k|}\sum_{i\in C_k} x^{(i)}$$
5. Repeat steps 3 and 4 until convergence, i.e., until the cluster assignments no longer change.

Remarks on K-means clustering:

- *The results depend on the initial placement of centroids*

  Because the objective function in (1) is not convex, the algorithm will find a local rather than a global minimum and the results depend on the initialization of centroids. There are several ways to deal with this problem.

  – Typically we run the algorithm several times from different random initial configurations and we select the solution for which the objective function is the lowest.

    In `sklearn` this value can be set by the hyperparameter `n_init` with the default value of 10.

  – K-means++ algorithm (2007) is an improvement that introduces a smarter initialization step that selects centroids that are distant from one another.

    In `sklearn` this initialization method is used by default. If you want to use the original method of selecting initial centroids randomly (from the data observations), set the hyperparameter `init = "random"`.

- *Within-cluster variation vs. sum of squared distances or errors (SSE) to the centroids*

  Note that the following identity holds

$$\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{d}\left(x_j^{(i)}-x_j^{(i')}\right)^2 = 2\sum_{i\in C_k}\sum_{j=1}^{d}\left(x_j^{(i)}-\bar{x}_{k,j}\right)^2$$

  where $\bar{x}_{k,j}$ is the mean for feature j in cluster C_k

$$\bar{x}_{k,j} = \frac{1}{|C_k|}\sum_{i\in C_k} x_j^{(i)}$$

  In other words, the centroid for cluster C_k is $(\bar{x}_{k,1},\ldots,\bar{x}_{k,d})$.

  The above identity means that the within-cluster variation in each cluster C_k is proportional to the sum of squared distances of data observations in that cluster to the centroid of that cluster. Hence, to measure the quality of clustering, we can compute sum of squared distances from observations to their respective centroids (`sklearn "inertia"`).

- *Computational complexity*

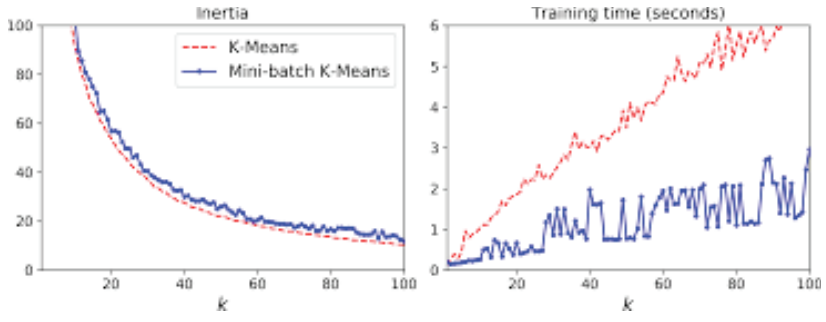
  If we have n data points in d dimensions and we want to cluster them into K clusters, the running time for the algorithm is $O(tKnd)$, where t is the number of iterations until convergence. We can see that this is true because in each iteration (t of them), we need to find a distance from each point in the data set (n of them) to each centroid (K of them), and the time to compute the distance grows with dimension d. However, compared to other clustering algorithms, K-means is typically very fast. Two recent improvements of the algorithm regarding the computational complexity are the following.

  – *Accelerated K-means* (Charles Elkan, 2003) avoids many unnecessary distance calculations by exploiting the triangle inequality and by keeping track of lower and upper bounds for distances between instances and centroids.

    In `sklearn` set the hyperparameter `algorithm = "elkan"`.

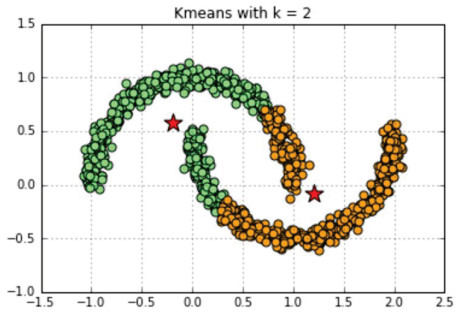
  – *Mini-batch K-means* (2010) uses small, random, fixed-size batches of data to store in memory, and then with each iteration, a random sample of the data is collected and used to update the clusters. This speeds up the algorithm typically by a factor of 3 or 4 and makes it possible to cluster huge data sets that do not fit in memory.

    However, the within-cluster variation is generally higher when using K-means.



Hands-On Machine Learning by Geron

- *Produces convex spherical-shape clusters*



https://yanpuli.github.io/posts/2017/03/blog-post-8

- *Variations on K-means algorithm*
  - *Bisecting K-means* algorithm works as follows.

    We decide how many clusters we want to have and we start with the entire data being in one large cluster. Then we use 2-means (bisection) on this cluster and obtain two clusters. If $K = 2$, we stop, and if $K > 2$, we compute SSE (the sum of squared distances from data points to their respective centroids), use 2-means on the cluster with higher SSE, and obtain three clusters (one cluster from the previous step and two cluster from this step). If $K = 3$, we stop, otherwise we use 2-means on the cluster with the highest SSE. We continue until we reach K clusters.
  - *K-medoids* algorithm works the same as K-means with two main differences.
    * In K-means, the "center" of the cluster (centroid) is found by computing the averages, while in K-medoids we use the medoid which is the most centrally located object in a cluster. In 1-dimension, the medoid is the median.
      In general, if $\mathcal{X} := \{x_1, x_2, \ldots, x_m\}$ is a set of n points in a space with a distance function d, the medoid is defined as

      $$x_{\text{medoid}} = \arg\min_{y \in \mathcal{X}} \sum_{i=1}^{m} d(y, x_i).$$

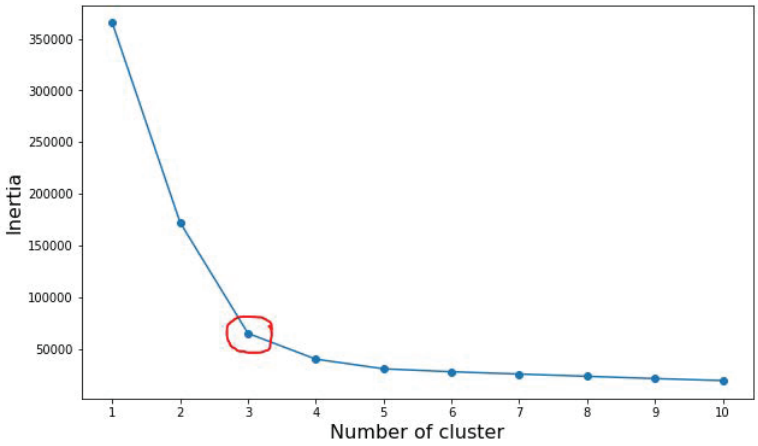
    * Secondly, the K-medoids algorithm minimizes the sum of dissimilarities between data objects whereas K-means minimizes the sum of squared Euclidean distances which places the highest influence on the largest distances.

    K-means algorithm is very sensitive to outliers and this is overcome when we use K-medoids.

How do we select the optimal value of K?

1. Note that increasing the number of clusters K reduces SSE and if each data point is its own cluster then SSE $= 0$.

   To find an optimal value of K, we vary the parameter K, plot the graph of SSE vs. K, and use the "elbow" method to find K. We select K such that the increasing the number of clusters will not significantly reduce SSE.



https://pub.towardsai.net/get-the-optimal-k-in-k-means-clustering-d45b5b8a4315

2. A rule of thumb is $K \approx \sqrt{n/2}$ where n is the number of data points in the data set.

3. Another approach to choose K (as well as to evaluate any clustering solution) is to use the silhouette score.

   The silhouette coefficient is a measure of how similar a data observation is to its own cluster (cohesion) compared to other clusters (separation). Given a data observation $x^{(i)}$ and any cluster C, the distance (or, dissimilarity) of $x^{(i)}$ to C is defined by

$$d(x^{(i)}, C) = \frac{1}{|C|} \sum_{j \in C} d(x^{(i)}, x^{(j)})$$

   If $x^{(i)}$ belongs to cluster C_k we denote

$$\boxed{a^{(i)} = d(x^{(i)}, C_k)}$$

   Then we find the next closest cluster to $x^{(i)}$ by

$$C_{k'} = \text{argmin}_{C \neq C_k} d(x^{(i)}, C)$$

   and we denote

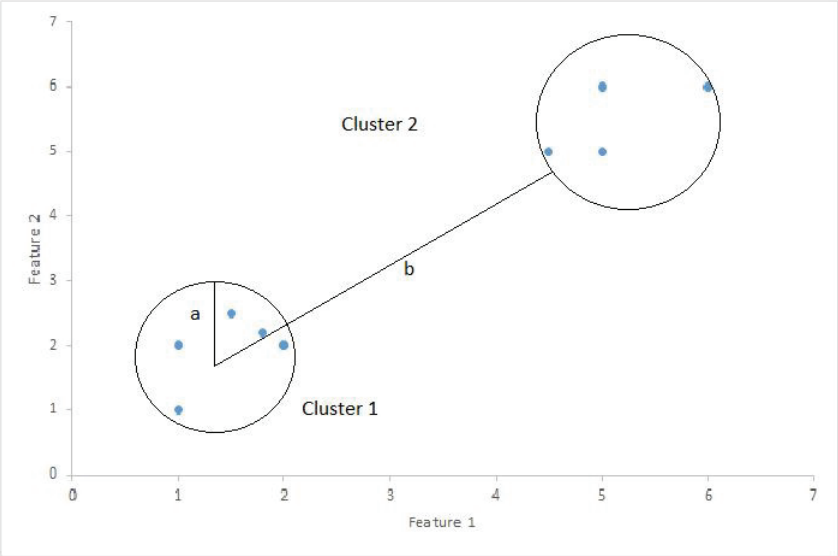
$$\boxed{b^{(i)} = d(x^{(i)}, C_{k'})}$$

In other words, we define $a^{(i)}$ as the average intra-cluster distance or average dissimilarity of $x^{(i)}$ to all points in its own cluster and we define $b^{(i)}$ as the average nearest-cluster distance. The silhouette coefficient is a comparison of $a^{(i)}$ and $b^{(i)}$, appropriately scaled.

Intuitively, for a good clustering solution, we would like $a^{(i)}$ to be smaller than $b^{(i)}$ indicating that a data point $x^{(i)}$ fits better in its assigned cluster than in a neighboring cluster.

The silhouette coefficient for $x^{(i)}$ is defined by

$$\boxed{s^{(i)} = \frac{b^{(i)} - a^{(i)}}{\max\{a^{(i)}, b^{(i)}\}}}$$



https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c

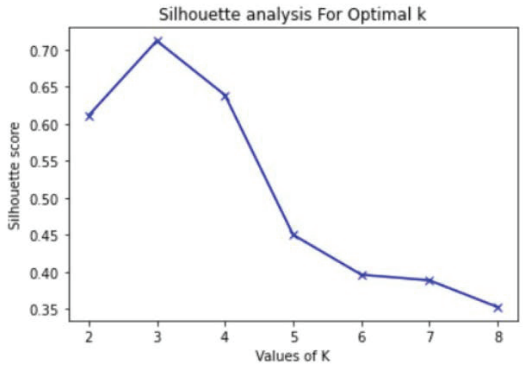
The silhouette coefficient $s^{(i)}$ can vary between -1 and $+1$.

- a value close to $+1$ indicates that the data point is well inside its own cluster and far from other clusters,
- a value of 0 means that the data is on the boundary between two clusters, and
- a value close to -1 means that the data point might have been assigned to the wrong cluster.

If most data points have a high silhouette coefficient, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

The silhouette score is the average silhouette coefficient over all data instances.

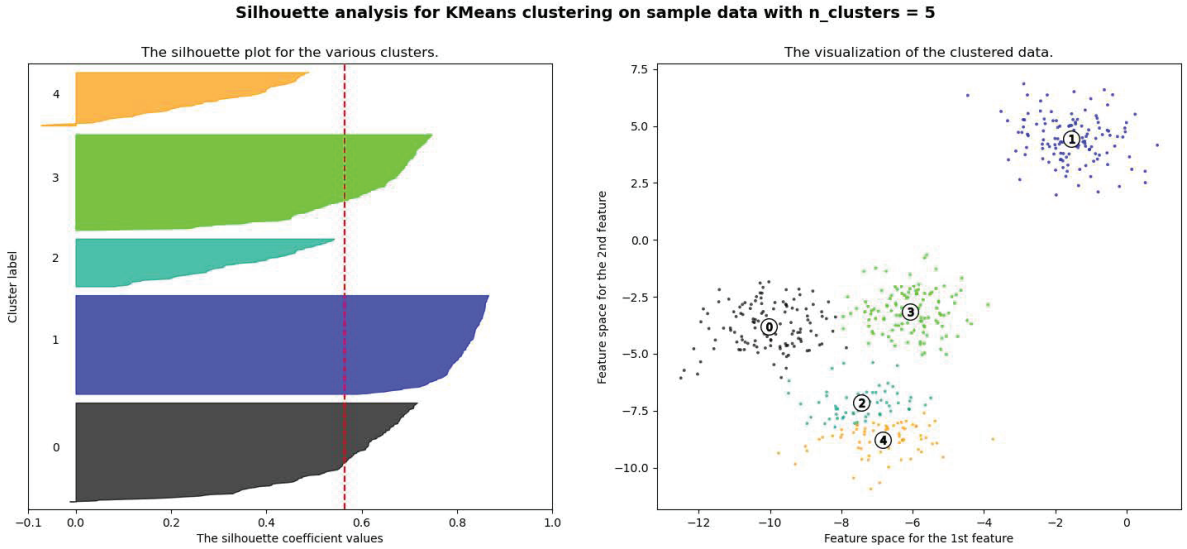
Similar to the elbow method, we pick a range of candidate values of K, train K-means clustering for each of the values of K, compute the silhouette score for each solution and observe the fluctuations.



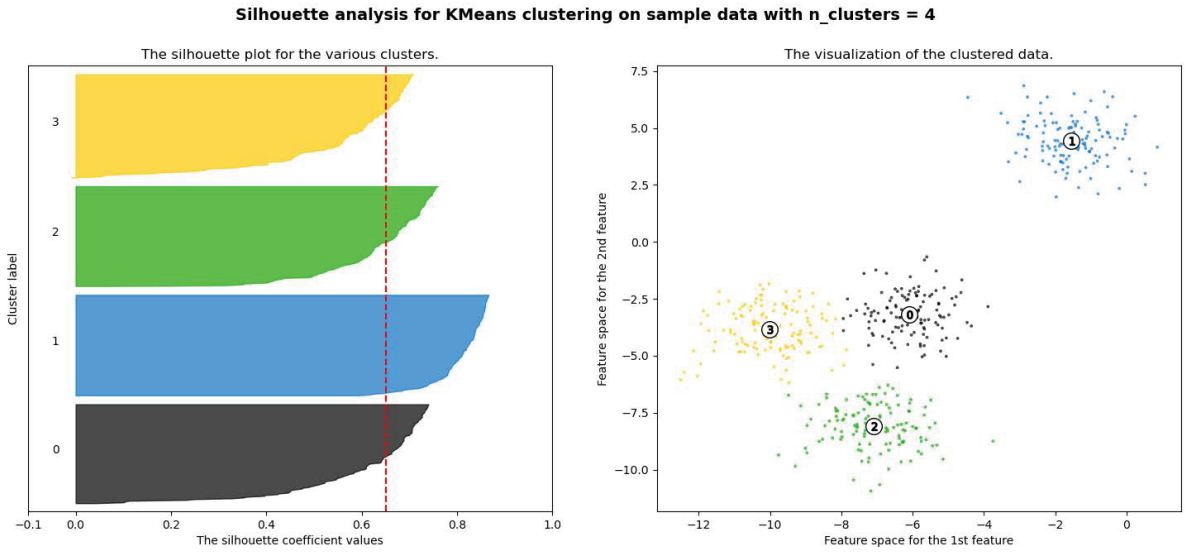
https://www.analyticsvidhya.com/blog/2021/05/k-mean-getting-the-optimal-number-of-clusters/

When dealing with higher dimensions, the silhouette score is quite useful to validate the clustering solution as we cannot use any type of visualization to validate clustering when dimension is greater than 3.

Another more informative visualization is obtained if we plot the silhouette diagram. This is a plot of the silhouette coefficients for each data instance, sorted by the cluster they are assigned to and by the value if the coefficient.



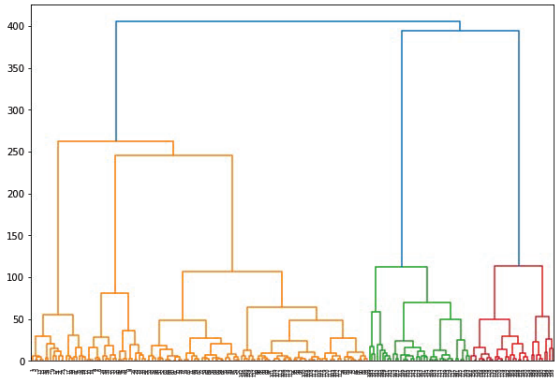
https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html



https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

III. Hierarchical Clustering

- Hierarchical clustering algorithms do not require the number of clusters K as an input like K-means clustering algorithms. They are also iterative algorithms, more computationally expensive, more informative, and they result in a tree-based representation of the observations called a *dendrogram* that allows us to view at once the clusterings obtained for each number of clusters.



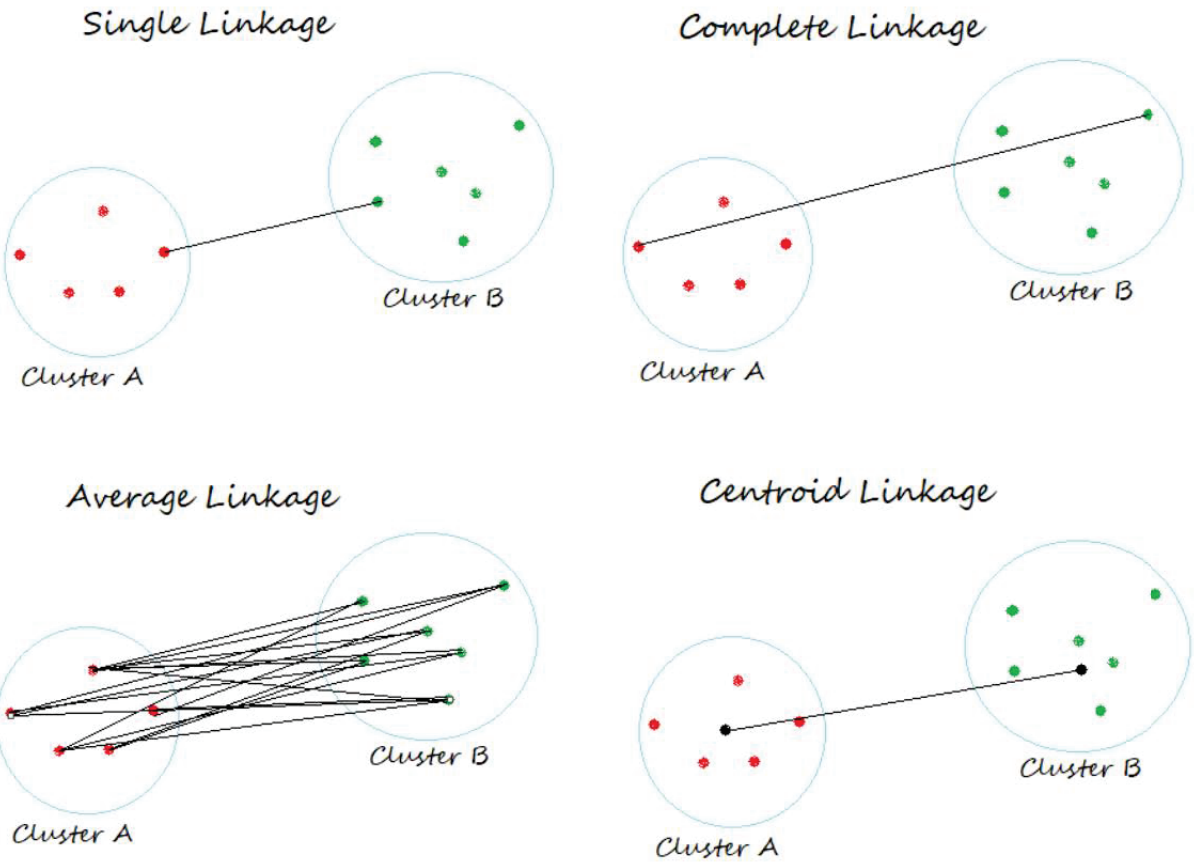
- We are essentially building a hierarchy of clusters. There are two versions:
  - agglomerative: in each iteration two most similar clusters are merged into one cluster ("bottom–up" approach)
  - divisive: in each iteration a cluster is split ("top–down" approach)

In each level of the hierarchy, a different number of clusters is present.

One of the hierarchical levels is chosen as the clustering solution according to various criteria (the number of clusters based on SSE, the silhouette score, the cophenetic coefficient, etc).

Agglomerative methods are more common and we focus on them.

- Input:
  1. data
  2. distance/dissimilarity function between two data observations
  3. distance/dissimilarity (or linkage) between two clusters $d(C, C')$



https://www.analyticsvidhya.com/blog/2021/06/single-link-hierarchical-clustering-clearly-explained/

  - *single linkage: minimal intercluster dissimilarity*

    The distance $d(C, C')$ is calculated as the minimum distance between elements of C and C'

$$d(C, C') = \min_{x \in C, x' \in C'} d(x, x')$$

– *complete linkage: maximal intercluster dissimilarity*

The distance $d(C, C')$ is calculated as the maximum distance between elements of C and C'

$$d(C, C') = \max_{x \in C, x' \in C'} d(x, x')$$

– *average linkage: mean intercluster dissimilarity*

The distance $d(C, C')$ is calculated as the average of all the pairwise distances between elements of C and C'

$$d(C, C') = \frac{1}{|C| \cdot |C'|} \sum_{x \in C, x' \in C'} d(x, x')$$

– *centroid linkage*:

The distance $d(C, C')$ is calculated as the distance between the centroids of clusters C and C'.

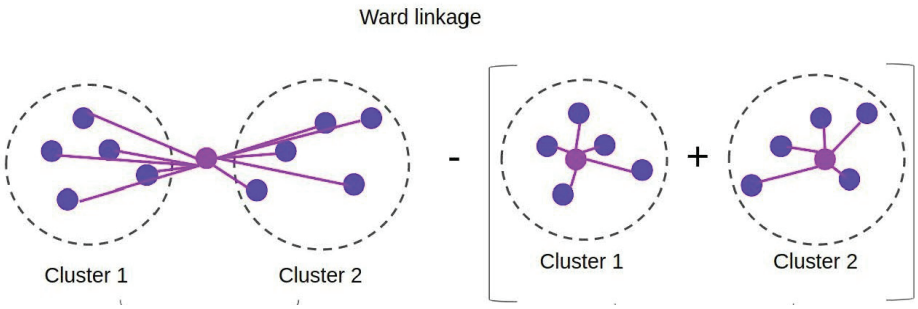
– *medoid linkage*:

The distance $d(C, C')$ is calculated as the distance between the medoids of clusters C and C'.

– *Ward linkage*:

Two clusters are merged if the sum of squared errors after merging two clusters into a single cluster is smallest among all pairs of clusters.

$$SSE_{C \cup C'} - (SSE_C + SSE_{C'})$$



https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn/

- **Algorithm:**

  1. Select a distance function and the linkage method.
  2. Compute the starting distance (or proximity) matrix for all observations.
  3. Merge the two observations (or clusters) with the smallest distance to form a new cluster. In case of ties, perform multiple merges.
  4. Update the distance matrix with the new cluster distances based on the linkage selected.
  5. Repeat steps 3 and 4 (merge and update) until all observations are merged into a single cluster.

  A common graphical representation of the solution is a dendrogram. It shows which clusters merged and at what similarities the merges occurred as well as the order in which the clusters are formed.

  The dendrogram depicts cohesion (at lower levels) and separation (at higher levels) indicating the quality of clustering solution.

  The dendrogram shows a collection of possible clustering solutions and an individual solution is obtained by cutting the dendrogram at a particular level. Typically, a judgment call must be made about where to make the cut, which is often based on the distance between the merges or the number of desired clusters or both. For example, if there is a large gap in distance before a merge perhaps the clusters existing before the merge are really separate clusters. Another approach is to define the error criterion and plot the value of the error versus the number of clusters and use the elbow method.

- The various linkages have different advantages and disadvantages related to their behavior and use and influnce the shape of clusters.

  Single linkage is also known as the nearest neighbor linkage and it can find long and thin clusters (chains) where the nearby points in the cluster are close to each other, but the points on the opposite ends might be much further apart then two elements from different clusters. With this linkage, clusters are joined if they have a single pair of close points. This linkage looks only for separation and can handle complicated shapes as long as the gap between different groups is not too small. It is very sensitive to noise and cannot separate the groups well if there is noise between them.

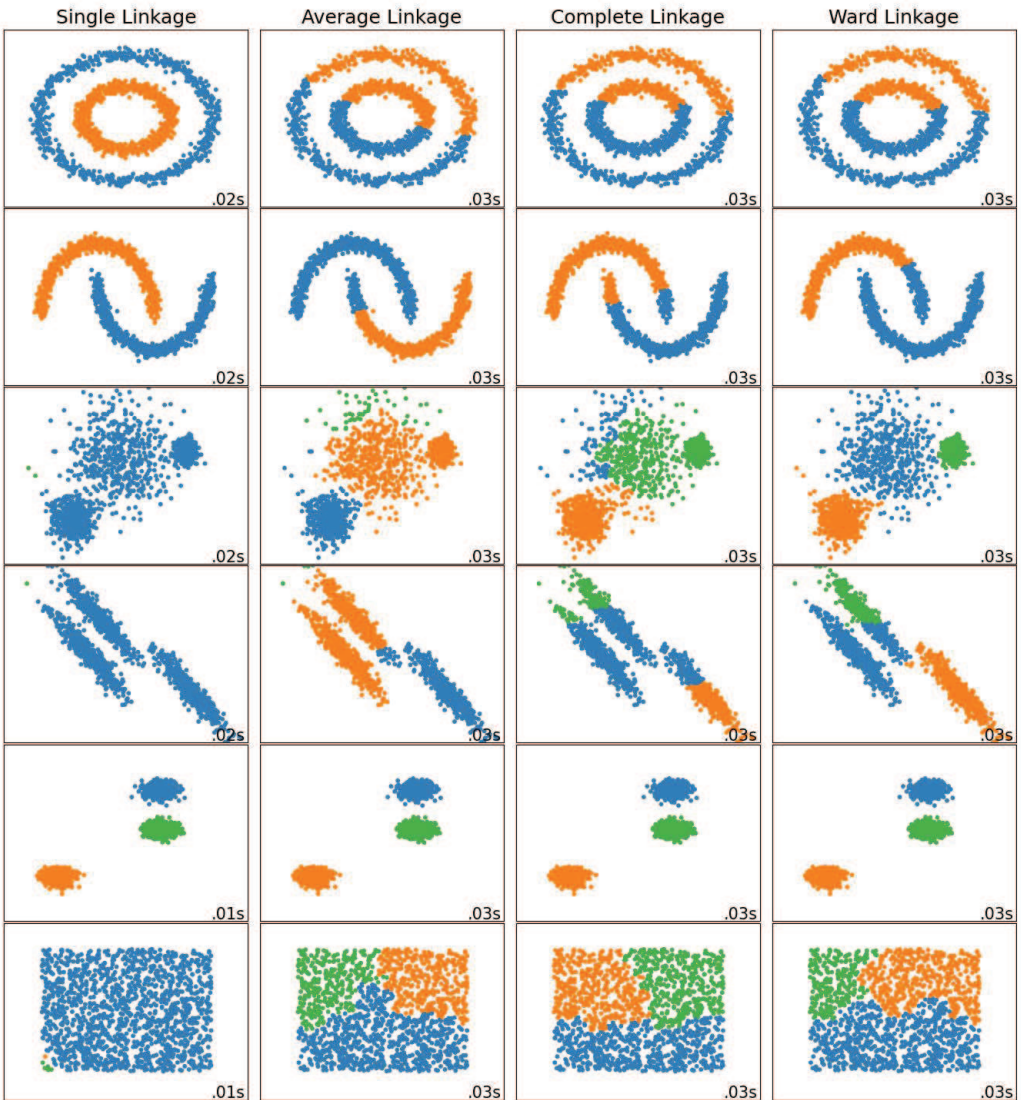
  Complete linkage is known as farthest neighbor linkage and produces clusters of spherical shapes that are of similar diameter. It may break large clusters and is less sensitive to noise.

  Average linkage is a compromise between single and complete linkages.

  Centroid linkage is less computationally expensive.

  Medoid linkage is less computationally expensive than single, complete, and average linkage, and less sensitive to outliers compared to average linkage.

  Ward linkage is also known as MISSQ (Minimal Increase in Sum or SQuares) linkage. It produces round clusters and is less sensitive to both noise and outliers. It is more preferred than other types of linkages.



https://scikit-learn.org/stable/auto_examples/cluster/plot_linkage_comparison.html

- **Example:** Perform agglomerative hierarchical clustering on two dimensional data using Euclidean distance and single linkage.

data point	x-coordinate	y-coordinate
A	1	1
B	1.5	1.5
C	5	5
D	3	4
E	4	4
F	3	3.5

Proximity matrices are

	A	B	C	D	E	F
A	0					
B	0.71	0				
C	5.66	4.95	0			
D	3.61	2.92	2.24	0		
E	4.24	3.54	1.41	1	0	
F	3.2	2.5	2.5	**0.5**	1.12	0

	A	B	C	{D, F}	E
A	0				
B	**0.71**	0			
C	5.66	4.95	0		
{D, F}	3.2	2.5	2.24	0	
E	4.24	3.54	1.41	1	0

	{A, B }	C	{D, F}	E
{A, B }	0			
C	4.95	0		
{D, F}	2.5	2.24	0	
E	3.54	1.41	**1**	0

	{A, B }	C	{{D, F}, E }
{A, B }	0		
C	4.95	0	
{{D, F}, E }	2.5	**1.41**	0

	{A, B }	{{{D, F}, E }, C}
{A, B }	0	
{{{D, F}, E }, C}	2.5	0

- **Computational Cost**

  Given n data observations, the computational cost for each iteration is at most $O(n^2)$. Since there are $n-1$ iterations until we reach a single cluster, the overall computational cost is at most $O(n^3)$, which more expensive that the cost for K-means.

- **Hierarchical Clustering Summary**
  - Advantages
    * Flexible number of clusters.
    * The resulting clusters may not be convex.
    * More informative than K-means since it produces an ordering of similarity between data observations.
  - Disadvantages
    * Hierarchical clustering cannot handle big data well. This is because the time complexity of K-means is linear $O(n)$ while that of hierarchical clustering is of oder $O(n^3)$ with n being the number of data observations.
    * We cannot un-merge clusters even if we notice at later iterations that they should have not been merged in the first place.

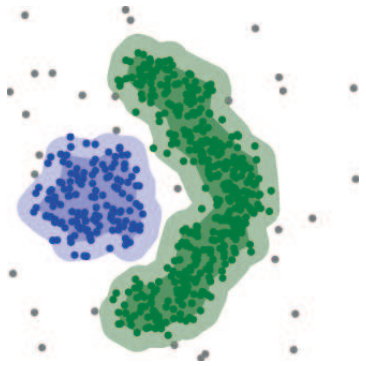
IV. DBSCAN Clustering

- This is a *Density-Based Spatial Clustering of Applications with Noise* method that looks for regions in space where data points are dense (clusters) and separated by more sparsely populated space.

  In general, clusters have good quality if they are dense (high cohesion) and separated by regions with no or few data observations (high separation).

  The density of points internal to a cluster is relatively consistent and high.

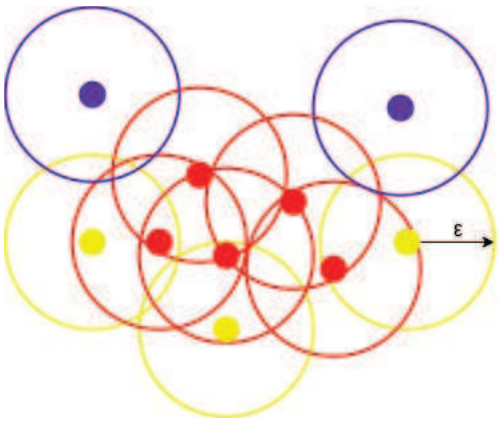
  DBSCAN defines clusters as well as noise (outliers, anomalies).



https://en.wikipedia.org/wiki/DBSCAN

- There are three key hyperparameters of DBSCAN:
  - distance/dissimilarity function between individual data observations
  - `eps` (epsilon): the value specifying the size of every point's neighborhood; in other words, how far we look around the point for its neighbors
  - `min_samples`: minimum number of data points (i.e., the amount of density) to define a cluster

- **Core points, border points, and noise**
  - For each data instance, count how many points are located within a distance `eps` from it. This region is called the point's `eps`-neighborhood.
  - If a data point has at least `min_samples` instances in its `eps`-neighborhood, including itself, then it is considered a core point. Note that core points are those points located in dense regions.
  - If a data point does not have at least `min_samples` instances in its `eps`-neighborhood, including itself, then this point is a
    * border point if it is in an `eps`-neighborhood of a core point
    * noise if it is not in an `eps`-neighborhood of a core point.



https://www.analyticsvidhya.com/blog/2020/09/how-dbscan-clustering-works/

- **Directly-density reachable and density reachable points**
  - A point p is directly-density reachable from a core point q if it is in the `eps`-neighborhood of q. Points are only said to be directly reachable from core points.
  - A point p is density reachable from q if there is a chain of points that are directly density reachable from q to p. Note that this implies that all points on the path must be core points, with the possible exception of p. A non-core point may be reachable, but nothing can be reached from it.
  - All points not reachable from any other point are outliers or noise points.
- **Algorithm:**
  - *Clusters*: If q is a core point, then it forms a cluster together with all points that are density reachable from it. Each cluster contains at least one core point. Non-core points can be part of a cluster, but they form its border, since they cannot be used to reach more points.
  - *Noise:* Points that do not belong to any cluster.

  The algorithm works well if all the clusters are dense enough and if they are well separated by low-density regions.

- **How to select hyperparameters `min_samples` and `eps`?**
  - If `min_samples=1`, then every point is a core point. If `min_samples=2`, the result will be the same as of hierarchical clustering with the single linkage, with the dendrogram cut at height `eps`. Therefore, `min_samples` must be chosen at least 3.

    Larger values are usually better for data sets with more data observations, more features, and more noise.

    As a rule of thumb, `min_samples`$= 2d$ or `min_samples`$= d + 1$ are suggested where d is the number of features (dimension of the data set).
  - The value of `eps` can be chosen by using a K-distance graph, plotting the distance to the $K =$ `min_samples`-1 nearest neighbor ordered from the largest to the smallest value. Good values of `eps` are where this plot shows an "elbow".

    If `eps` is chosen much too small, a large part of the data will not be clustered; whereas for a too high value of `eps`, clusters will merge and the majority of objects will be in the same cluster and we will not see any details.

- **DBSCAN Summary**
  - Advantages
    - does not require to specify number of clusters beforehand
    - has the ability to detect clusters of arbitrary shape, including non-spherical clusters
    - able to detect the outliers which is important in anomaly detection
    - computational complexity almost linear with respect to number of data observations $O(n \cdot \log n)$
  - Disadvantages
    - less effective in high-dimensional spaces where data points are sparse
    - determining appropriate `min_samples` and `eps` is not easy and it often requires domain knowledge
    - if clusters are very different in terms of in-cluster densities, DBSCAN is not well suited to define clusters
    - not great in separating overlapping/touching clusters

V. Clustering Validation and Evaluation

"The validation of clustering structures is the most difficult and frustrating part of cluster analysis. Without a strong effort in this direction, cluster analysis will remain a black art accessible only to those true believers who have experience and great courage."

(Algorithms for Clustering Data, textbook by Anil Kumar Jain, Richard C. Dubes)

- Why do we need to evaluate clustering solutions?
  - to determine the clustering tendency, i.e., to distinguish whether non-random structure actually exists in the data
  - to compare the results of a cluster analysis to externally known results; for example, to externally given class labels

- to compare the results of two different clustering solutions to determine which is better
- to determining the "correct" number of clusters
- we can further distinguish whether we want to evaluate the entire clustering solution or just individual clusters

- Some measures:
  - *cluster cohesion* is measured by the within cluster sum of squares, which shows how closely related are objects in a cluster
    $$SSE_{C_i} = \sum_{x \in C_i} (x - m_i)^2$$
    where C_i is the ith cluster and m_i is the centroid of C_i
  - *cluster separation* is measured by the between cluster sum of squares (BSS), which shows how distinct or well-separated a cluster is from other clusters.
    $$BSS = \sum_i |C_i|(m - m_i)^2$$
    where C_i is the ith cluster, m_i is the centroid of C_i, and m is the mean of all data points
  - *silhouette score*
  - *cophenetic coefficient* (for hierarchical clustering) is a measure of the correlation between the distance of points in the feature space and their distance on the dendrogram

Python code:

Lecture_16_Kmeans_Clustering.ipynb
Lecture_16_Kmeans_China.ipynb
Lecture_16_Hierarchical_DBSCAN.ipynb

Homework:

- Consider the `wine.csv` data set, attached on OneDrive, with 13 attributes (columns 2–14).

  The first column `"wine"` is a target variable, so this data set could also be used for classification. Ignore this column when using unsupervised learning algorithms.
- Create clusters using K-means clustering, agglomerative clustering, and DBSCAN. Investigate the values of hyperparameters for each of these algorithms and quality of clustering solutions. Choose the best clustering solution.
- Plot the best clustering solution in 2-dimensions using pairs of attributes as well as PCA.
- Compare the best clustering solution labels with the original data labels in the column `"wine"`.

References:

[1] Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow, by Geron (pp. 235-258)

[2] An Introduction to Statistical Learning, by James et al. (Section 10.3)

[3] https://en.wikipedia.org/wiki/DBSCAN