Fourth Industrial Summer School

Module 4: ML

Unsupervised Learning: Clustering Algorithms

Outlines

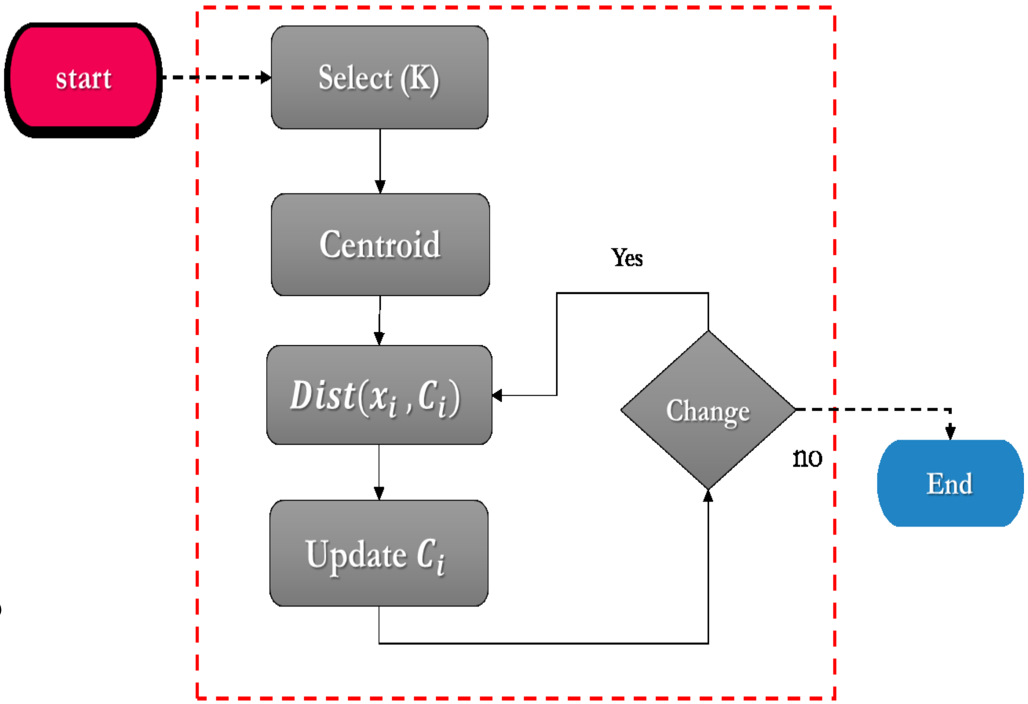
- ✓ Clustering algorithms
  - ✓ Centroid Based (K-means)
  - ✓ Clustering Validation
  - ✓ How to choose K for K-means
  - ✓ Sklearn implementation

K-means Clustering

- The K-means algorithm by far is the most popular and widely used algorithm.
- K-means clustering algorithm is a machine learning technique used to identify distinct patterns or structures within unlabeled data.
- It is centroid-based (partition) algorithm
- The variable ***K*** represents the number of groups in the data.

K-means clustering algorithm

1. Choose (guess) the number of clusters ***K***
2. Specify the cluster seeds (Initial Centroids)
3. Assign each point to a centroid based on distance
4. Adjust the centroids coordinates
5. Repeat the last two steps until no centroid updates.



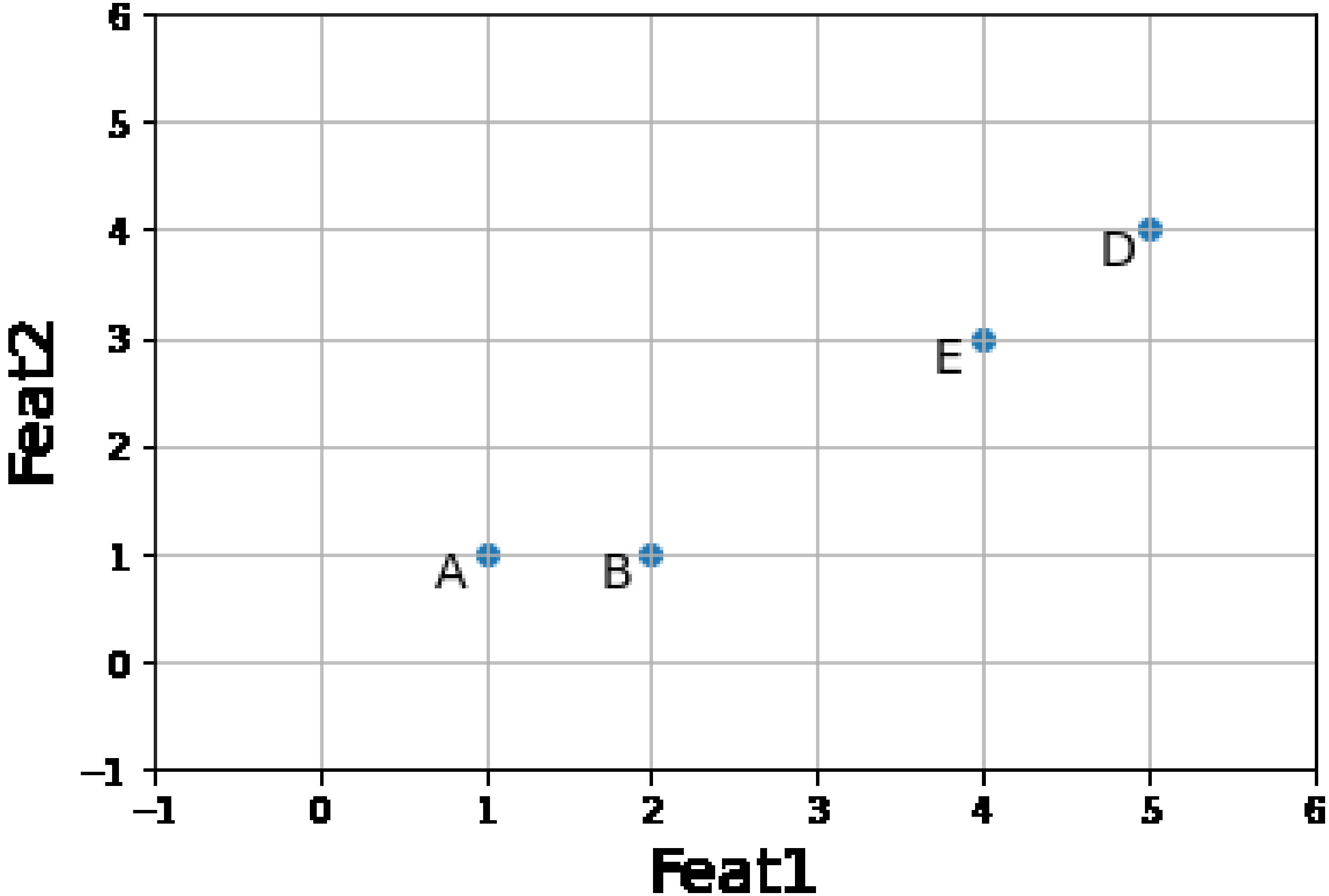
Quick numerical example

Suppose we have four samples, and each has two features (Feat1, and Feat2). Our goal is to gather them into groups (*K = 2*).

Points	Feat1 (x1)	Feat2 (x2)
A	1	1
B	2	1
E	4	3
D	5	4

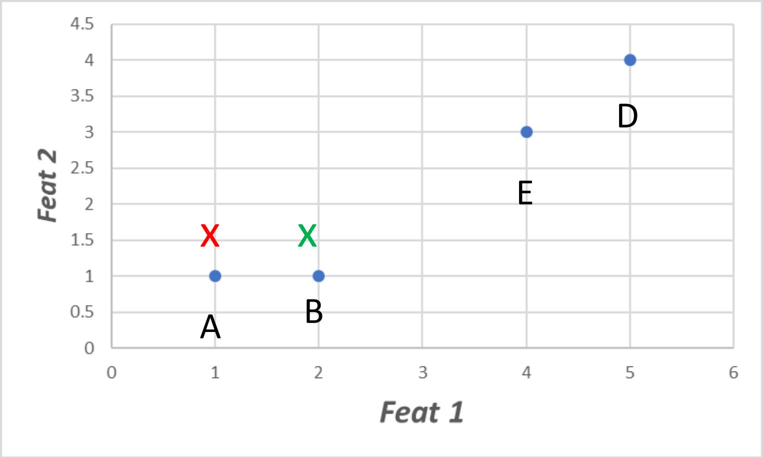
Quick numerical example

Plotting the data



Example

- Step 1: Select number of clusters: 2
- Step 2: Initial seed points, let us select A, and B points



$C_1 = (1,1), \qquad C_2 = (2,1)$

$$d(A, C_1) = \sqrt{(1-1)^2 + (1-1)^2} = 0,$$
$$d(B, C_1) = \sqrt{(2-1)^2 + (1-1)^2} = 1,$$
$$d(E, C_1) = \sqrt{(4-1)^2 + (3-1)^2} = 3.61,$$
$$d(D, C_1) = \sqrt{(5-1)^2 + (4-1)^2} = 5,$$

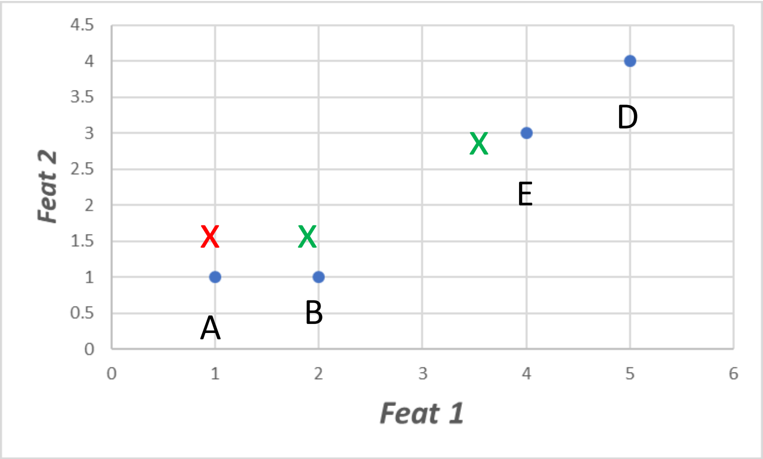
$$d(A, C_2) = \sqrt{(1-2)^2 + (1-1)^2} = 1,$$
$$d(B, C_2) = \sqrt{(2-2)^2 + (1-1)^2} = 0,$$
$$d(E, C_2) = \sqrt{(4-2)^2 + (3-1)^2} = 2.83,$$
$$d(D, C_2) = \sqrt{(5-2)^2 + (4-1)^2} = 4.24,$$

- Step 3: Assign each point to a cluster with the nearest seed point

$$C_1 \Rightarrow \{A\}, C_2 \Rightarrow \{B, E, D\}$$

Example

- Step 4: Adjust the centroids coordinates



Centroid: $C_1 = (1{,}1)$, No change!

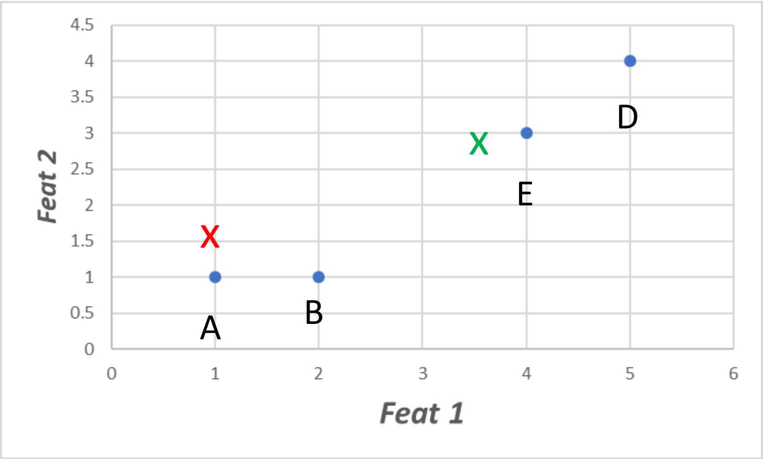
Centroid: $C_2 = \dfrac{2+4+5}{3}, \dfrac{1+3+4}{3}$

$C_2 = (3.66, 2.66)$

End of Iteration 1, repeat! (we have centroid change coordinates)

Example (Iteration 2)

- Step 3: Assign each point to a cluster with the nearest seed point



$C_1 = (1,1), \qquad C_2 = (3.66, 2.66)$

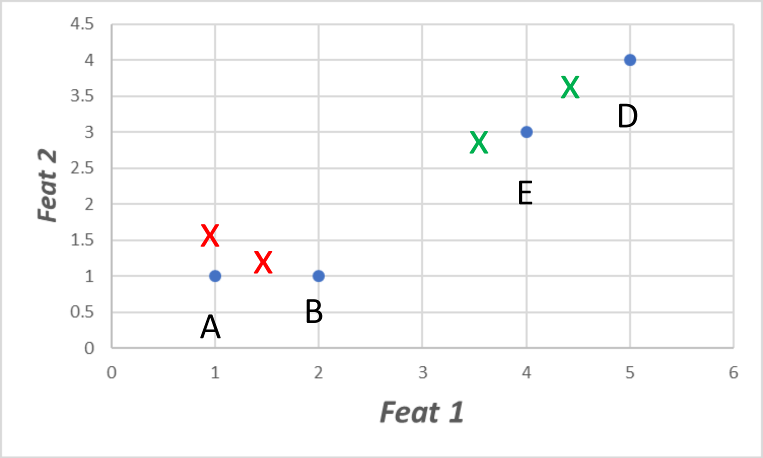
$$d(A, C_1) = \sqrt{(1-1)^2 + (1-1)^2} = 0,$$
$$d(B, C_1) = \sqrt{(2-1)^2 + (1-1)^2} = 1,$$
$$d(E, C_1) = \sqrt{(4-1)^2 + (3-1)^2} = 3.61,$$
$$d(D, C_1) = \sqrt{(5-1)^2 + (4-1)^2} = 5,$$

$$d(A, C_2) = \sqrt{(1-3.66)^2 + (1-2.66)^2} = 3.13,$$
$$d(B, C_2) = \sqrt{(2-3.66)^2 + (1-2.66)^2} = 2.35,$$
$$d(E, C_2) = \sqrt{(4-3.66)^2 + (3-2.66)^2} = 0.48,$$
$$d(D, C_2) = \sqrt{(5-3.66)^2 + (4-2.66)^2} = 1.89,$$

$$C_1 \Rightarrow \{A, B\}, C_2 \Rightarrow \{C, D\}$$

Example

- Step 4: Adjust the centroids coordinates



Centroid: $C_1 = \frac{1+2}{2}, \frac{1+1}{2}$

$C_1 = (1.5, 1)$

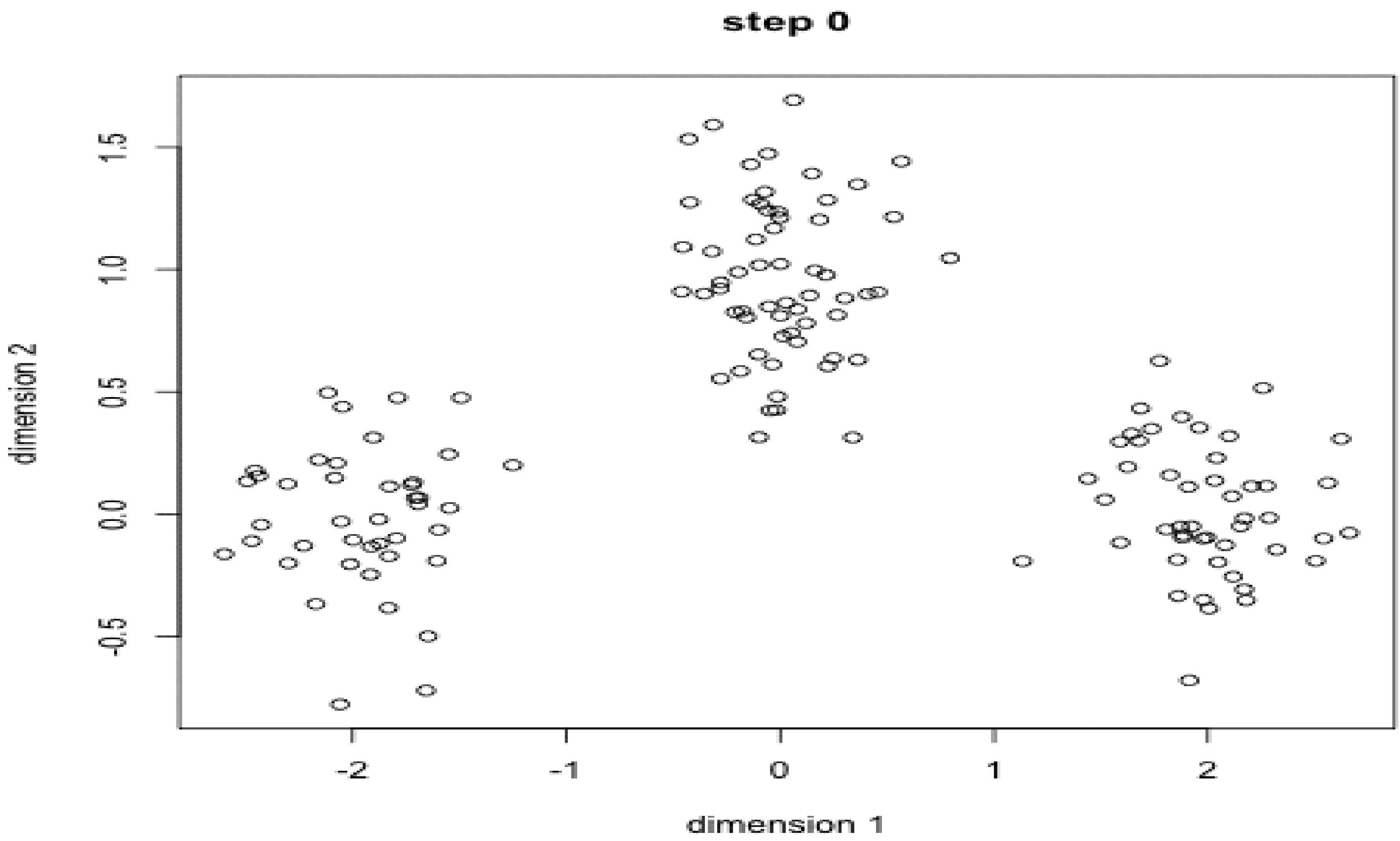
Centroid: $C_2 = \frac{4+5}{2}, \frac{3+4}{2}$

$C_2 = (4.5, 3.5)$

End of Iteration 2, repeat! (we have centroid change coordinates)

Iteration 3, Will introduce no change to clusters centroids, hence Convergence.

K-means Illustration (animi)



Source: https://miro.medium.com/max/960/1*KrcZK0xYgTa4qFrVr0fO2w.gif

Scikit Learn - KMeans

- Import KMeans

```
from sklearn.cluster  import KMeans
```

- Initialize KMeans and specify required parameters as needed

```
#generate an instance of KMeans: initial it
k = 2
kmeans=KMeans(n_clusters=k,
              init="k-means++",
              max_itr=300,
              random_state=None)
```

- Fit it to a training data

```
# Incorrect number of clusters
kmeans.fit(X)
y_pred = kmeans.predict(X)
```

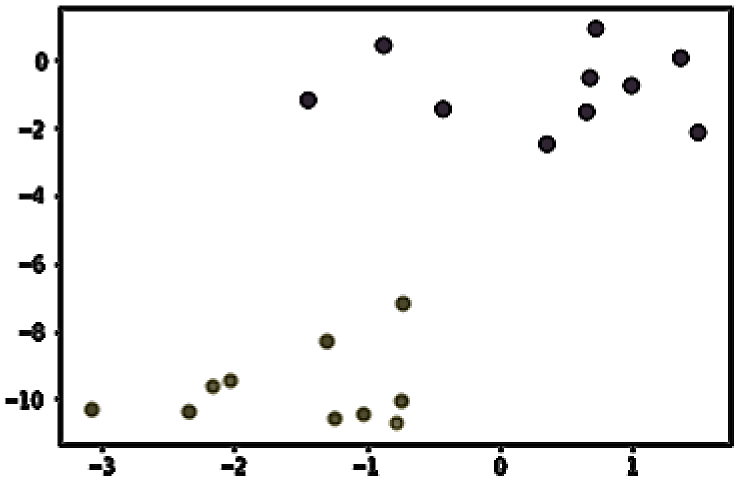
Kmeans Parameters and Attributes

Paramter	Description	Value
n_clusters	The number of clusters to form	K
init	Method for initialization	auto, random, manual
n_init	Number of time the k-means algorithm will be run with different centroid seeds	10
max_iter	Maximum number of iterations	300
tol	Relative tolerance with regards to inertia to declare convergence	1.00E-04

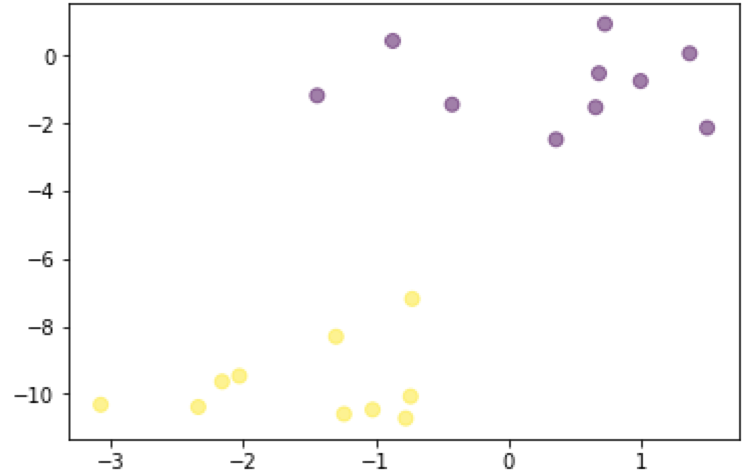
Attribute	Description
cluster_centers_	Coordinates of cluster centers
labels_	Labels of each point
inertia_	WSS
n_iter_	Number of iterations run.

Results (using different K value)

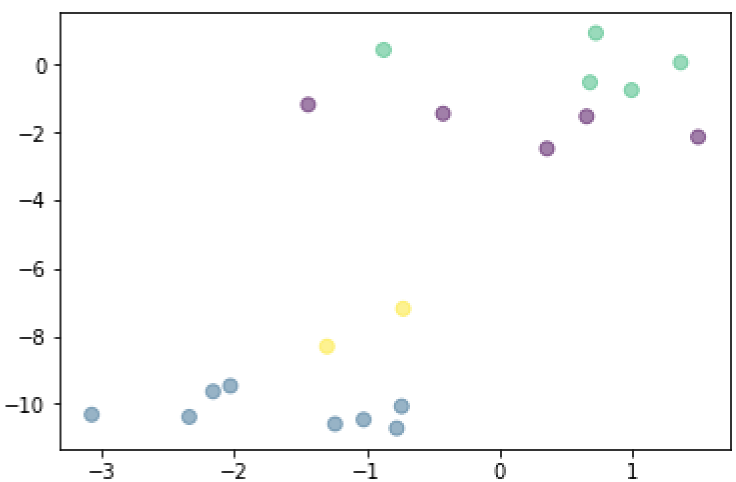
Data



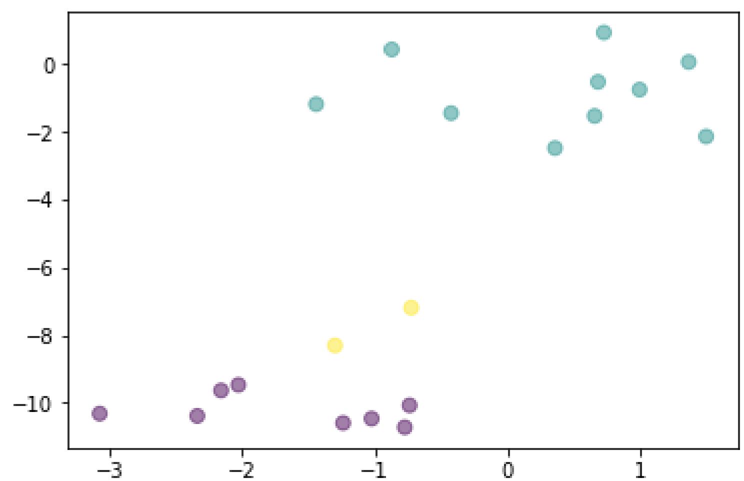
2 clusters



4 clusters



3 clusters




< / >
® Creative Commons licenses

Similarity:

- Given a structured data as shown below without labels, our task is to discover patterns (i.e., clusters)

variables

Samples

$$\begin{array}{cc} & \begin{array}{ccc} v_1 & \quad\quad & v_m \end{array} \\ \begin{array}{c} s_1 \\ s_2 \\ \\ s_n \end{array} & \begin{bmatrix} x_{01} & \cdots & x_{0,m} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,m} \end{bmatrix} \end{array}$$

If we pick two points (s_1, s_2) from the dataset, Then

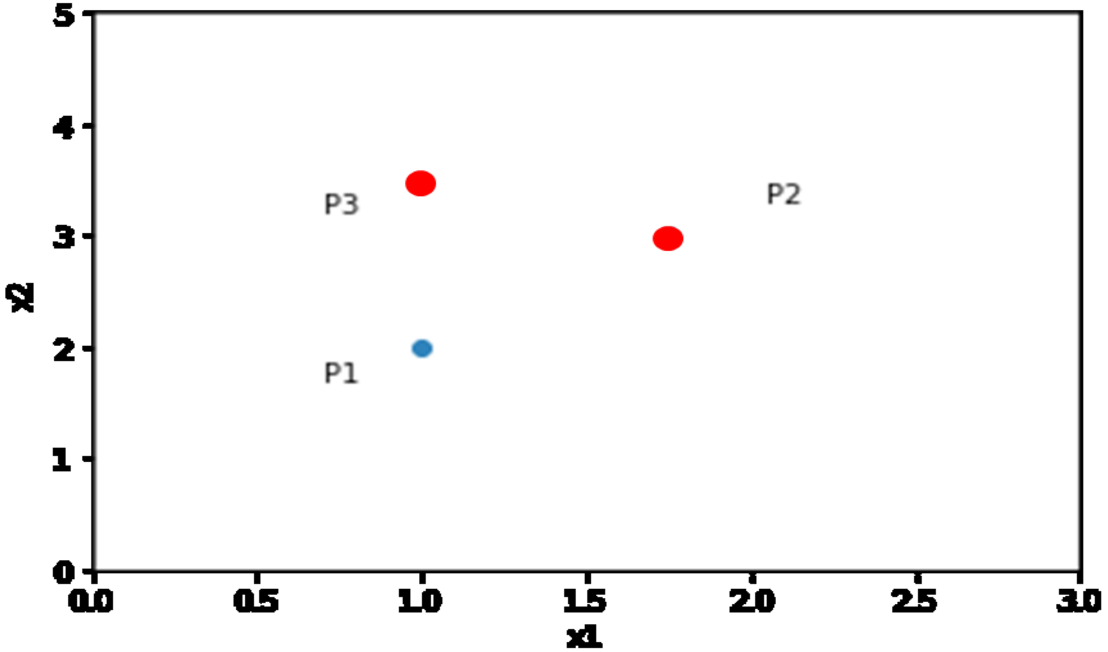
how can we decide if these two data points are similar?

Measurements: Dissimilarity

- One way to decide if the two points are similar is to measure the ***distance*** between them!
- **Distance Metric**
  - Similar points tend to be close to each other, while dis-similar points are far from each other.
    - Value ≈ 0 means the datapoints are similar
    - The distance value ranges [0, ∞)
- **Distance Metric examples:**
  - Minkowski distance [redacted]
  - Euclidean distance: is equivalent Mink with $q = 2$
  - Manhattan/Cityblock distance: is equivalent Mink with $q = 1$

Distance Metric effect

- Does it have any effects on the results?
  - Suppose we have three points



P_1=[1 , 2], P_2=[1.75, 3], P_3=[1, 3.5]

Which of the two points *P2 or P3* is closer *P1*?

Distance Metric effect

- Given three points P_1, P_2 and P_3 as:

$$P_1=[1, 2], \quad P_2=[1.75, 3], \quad P_3=[1, 3.5]$$

  - Compute *Euclidean* distances between P_1 to P_2, and between P1 to P_3?
  - Compute *Manhattan* (i.e. *Cityblock*) distances between P_1 to P_2, and between P1 to P_3?

```
import sklearn.metrics.pairwise as pw
```

  - use ***euclidean_distances( )*** and ***manhattan_distances( )***

Contd.

```
d1 = pw.euclidean_distances(p1.reshape(-1,2),  p2.reshape(-1,2) )[0]
d2 = pw.euclidean_distances(p1.reshape(-1,2),  p3.reshape(-1,2))[0]
if d1< d2:
    print('P2 is closer to P1: {}'.format(d1))
else:
    print('P3 is closer to P1: {}'.format(d2))
```

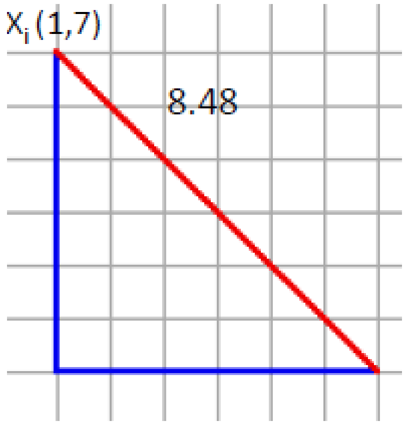
```
P2 is closer to P1: [1.25]
```

```
d1 = pw.manhattan_distances(p1.reshape(-1,2),  p2.reshape(-1,2) )[0]
d2 = pw.manhattan_distances(p1.reshape(-1,2),  p3.reshape(-1,2))[0]
if d1< d2:
    print('P2 is closer to P1: {}'.format(d1))
else:
    print('P3 is closer to P1: {}'.format(d2))
```

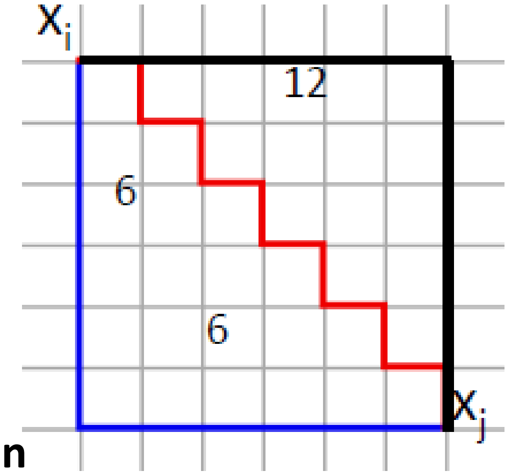
```
P3 is closer to P1: [1.5]
```

Differences in outcomes

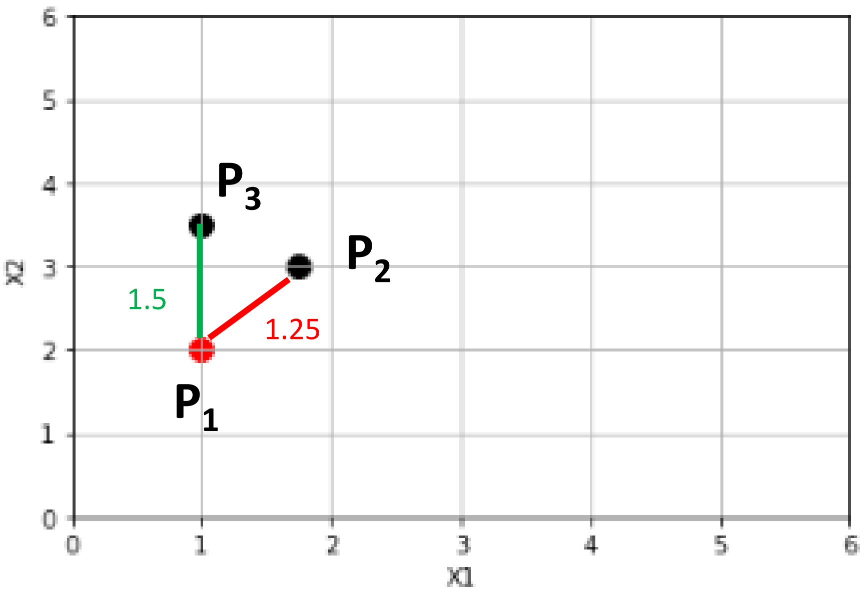
Euclidean



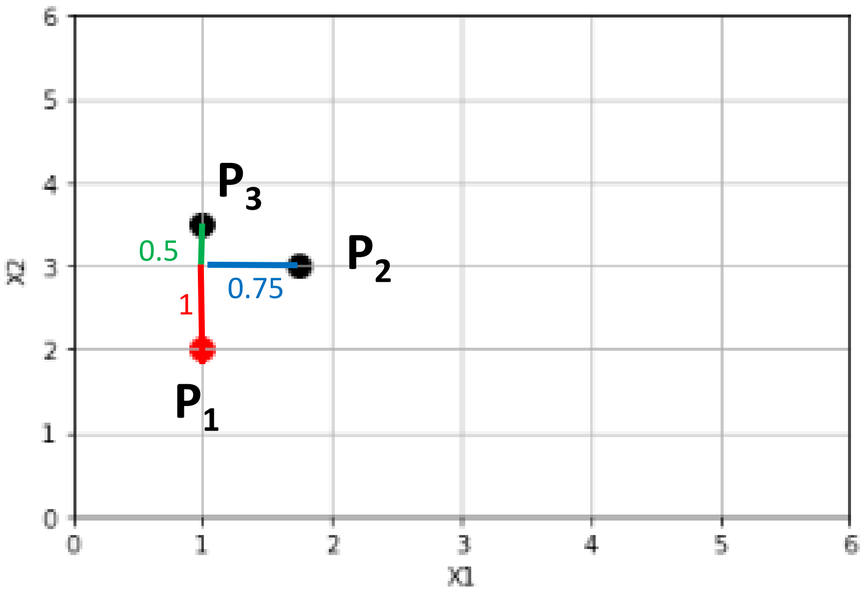
Manhattan

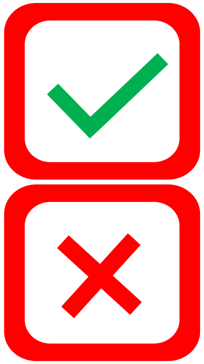


Euclidean



Manhattan





Clustering Validation

- The Effect of similarity metrics on results
- Clustering Validation (How good are the results?)
  - External Index
  - Internal Index

Clustering Validation

- A good clustering will produce high quality clusters
  - The **intra-class similarity** is high
  - The **inter-class similarity** is low

- Remember the clustering results, also, depends on the metrics used

- Why we need to evaluate?
  1. Compare clustering algorithms (to select the one suitable for the problem)
  2. Improving our confidence on the clustering results

- Mainly, there are two types of clustering validation:

  - **External Index validation:**
    - Comparing the clustering results to a **ground truth**

  - **Internal Index validation :**
    - Assesses the clustering quality, while **no ground truth** is available.

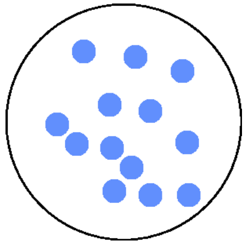
External Index: Entropy

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_b P(x_i)$$

- 2-Class Case:
  - What is the entropy of a group in which all examples belong to the same class?

$$H(1) = -\frac{13}{13}\log_2\left(\frac{13}{13}\right) = 0$$

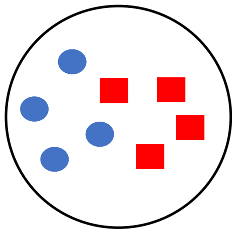
Minimum impurity



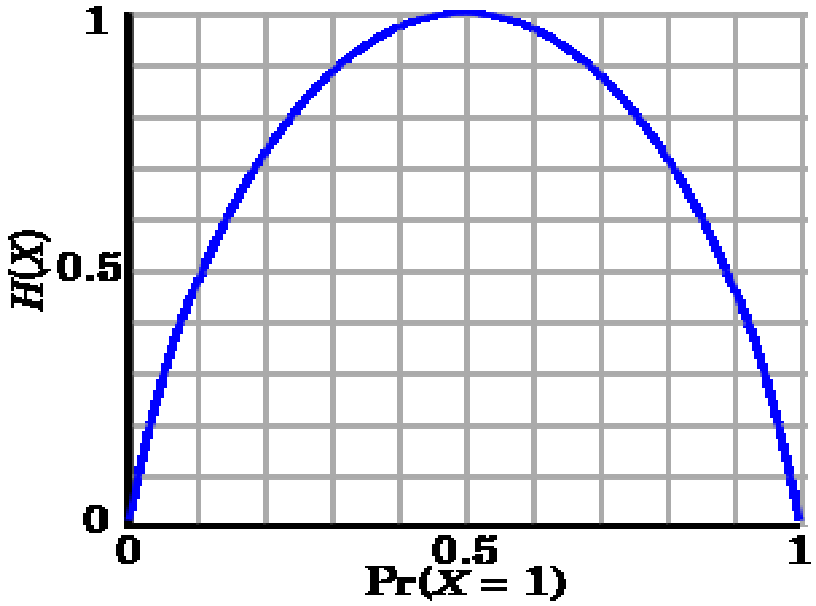
  - What is the entropy of a group with 50% in either class?

$$H(1) = -\frac{4}{8}\log_2\left(\frac{4}{8}\right) - \frac{4}{8}\log_2\left(\frac{4}{8}\right) = 1$$

Maximum impurity



External Index: Entropy-based



Impurity levels, max is
happened when 50% mixed
From both clusters

External Index: Entropy-based

- **Homogeneity score:** A clustering algorithm must assign only those datapoints that are similar to a cluster, **zero entropy**

- The homogeneity score is set to **1.0** for a perfectly homogenous solution, to adhere to the convention of 1.0 being desirable and 0.0 being undesirable

$$h = \begin{cases} 1 & \text{if } H(C,K) = 0 \\ 1 - \frac{H(C|K)}{H(C)} & \text{else} \end{cases}$$

where

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|} \frac{a_{ck}}{N} \log \frac{a_{ck}}{\sum_{c=1}^{|C|} a_{ck}}$$

$$H(C) = -\sum_{c=1}^{|C|} \frac{\sum_{k=1}^{|K|} a_{ck}}{n} \log \frac{\sum_{k=1}^{|K|} a_{ck}}{n}$$

For a set of classes, $C = \{c_i \,|\, i = 1,\ldots,n\}$ and a set of clusters, $K = \{k_i \mid 1,\ldots,m\}$.

Source: https://aclanthology.org/D07-1043.pdf

External Index: Homogeneity score

```python
# sklearn.metrics.homogeneity_score

from sklearn.metrics import homogeneity_score
# from sklearn.metrics import

# example 1
y1  = [0, 0, 0, 1, 1, 1]
y1p = [0, 0, 0, 1, 1, 1]
print ('Example 1 - Homogeneity score:{:0.2f}'.format(homogeneity_score(y1,y1p)))

# example 2
y2  = [0, 0, 0, 1, 1, 1]
y2p = [1, 1, 1, 0, 0, 0]
print ('Example 2 - Homogeneity score:{:0.2f}'.format(homogeneity_score(y2,y2p)))

# example 3
y3  = [0, 0, 0, 1, 1, 1]
y3p = [0, 1, 0, 1, 0, 1]
print ('Example 3 - Homogeneity score:{:0.2f}'.format(homogeneity_score(y3,y3p)))
```

```
Example 1 - Homogeneity score:1.00
Example 2 - Homogeneity score:1.00
Example 3 - Homogeneity score:0.08
```

Other metrics

- Sklearn provides several other external metrics to validate our clustering results.
- The table lists all validation metrics developed under sklearn, but not all of them are external validation, only the highlighted ones

`metrics.calinski_harabasz_score`(X, labels)	Compute the Calinski and Harabasz score.
`metrics.davies_bouldin_score`(X, labels)	Compute the Davies-Bouldin score.
`metrics.silhouette_score`(X, labels, *[, ...])	Compute the mean Silhouette Coefficient of all samples.
`metrics.silhouette_samples`(X, labels, *[, ...])	Compute the Silhouette Coefficient for each sample.

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.cluster

Internal Index: *cohesion & separation*

- Normal setting of clustering is **without GT**
- We can estimate/evaluate clustering quality by the clustering "*cohesion*" and "*separation*"

- **Cohesion(inertia)** is measured by the within-cluster sum of squares

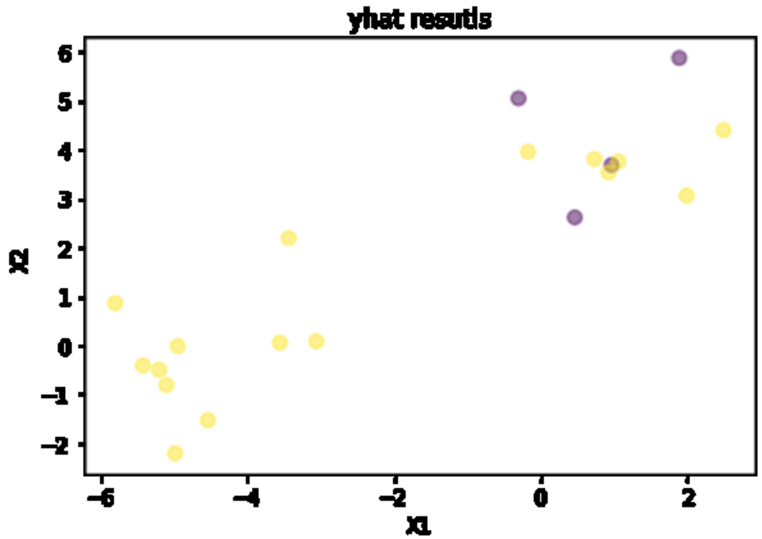
$$WSS = \sum_i \sum_{x \in C_i} (x - m_i)^2$$

- **Separation** is measured by the between-cluster sum of squares

$$BSS = \sum_i |C_i| (m - m_i)^2$$

- Where $|\boldsymbol{C_i}|$ is the size of cluster $\boldsymbol{i}$ and **m** is the centroid of all data
  - BSS + WSS = constant
  - WSS is called Sum of Squared Error (SSE)
  - Larger number of clusters tend to result in smaller SSE, why?

Example




```
[33]
cluster_1_datapoints = X[yhat==0, :]
cluster_1_mean = cluster_1_datapoints.mean(axis=0).reshape(1,2)
X2= np.tile(cluster_1_mean, (cluster_1_datapoints.shape[0],1) )
WS1 = (cluster_1_datapoints - X2) **2


cluster_2_datapoints = X[yhat==1, :]
cluster_2_mean = cluster_2_datapoints.mean(axis=0).reshape(1,2)
X3= np.tile(cluster_2_mean, (cluster_2_datapoints.shape[0],1) )
WS2 = (cluster_2_datapoints - X3)**2


WSS = WS1.sum() + WS2.sum()
print("WSS = %0.3f"%WSS)

WSS = 195.985
```

```
[34]
bs1 = len(cluster_1_datapoints) * ((X.mean(axis=0) - cluster_1_datapoints.mean(axis=0) )** 2).sum()
bs2 = len(cluster_2_datapoints) * ((X.mean(axis=0) - cluster_2_datapoints.mean(axis=0) )** 2).sum()
BSS = bs1 + bs2
print("BSS = %0.3f"%BSS)

BSS = 86.619
```

- WSS: 195.99
- BSS: 86.62

From the results our clustering is not good. As the cohesion is affected by the high value of WSS, while the separation between clusters is low! For good results the values should be the opposite

Internal Index: Silhouette Score

- Given a clustering output, and for each point of the dataset, Silhouette computes $\boldsymbol{\alpha}$ *and* $\boldsymbol{\beta}$ *values to find the Silhouette score.*

- The **silhouette score** for each point is given by

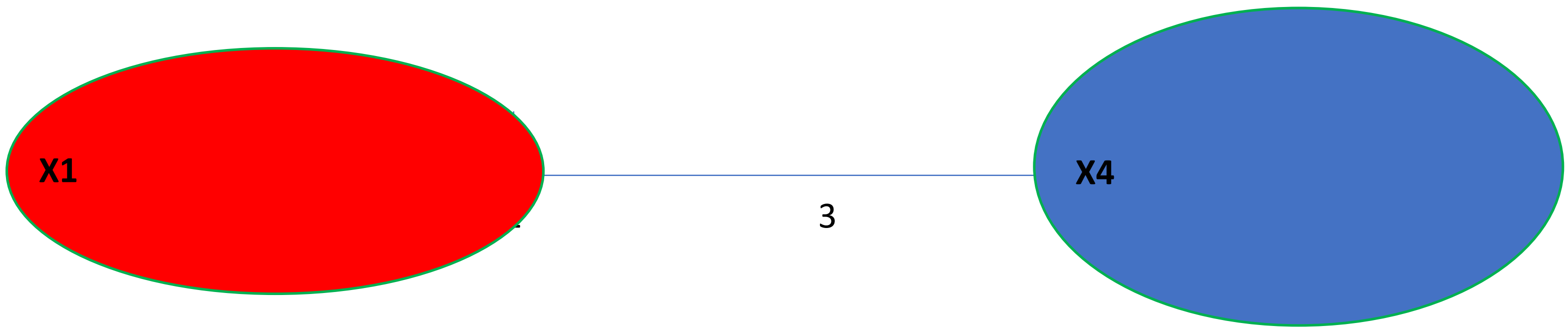
$$SL_i = \begin{cases} 1 - \frac{\alpha}{\beta}, if\ \alpha < \beta \\ \frac{\beta}{\alpha} - 1,\ otherwise \end{cases}$$

- Then, the mean Silhouette scores of all samples is reported

- The metric value ranges between **-1 to 1**

- A value closer to **1** is the better the clustering result, while a value closer to **-1** is the worst the results.
- Values around **0** indicate overlapping clusters results

Example of two samples

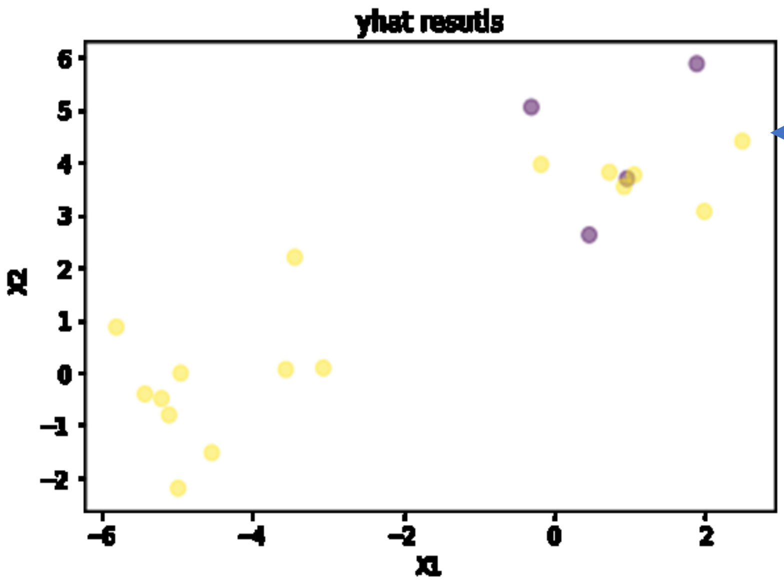


Data Point	Calculation (Euclidean distance)	Scores
[redacted]		
X=4	$\alpha_4 = \frac{((4-4.5)^2+(4-5)^2)}{2} = 0.625$ $\beta_4 = \frac{((4-2)^2+(4-1.5)^2+(4-1)^2)}{3} = 6.42$	$1 - \frac{\alpha_4}{\beta_4} = 0.90$

Silhouette evaluation example

```
[37]  1 from sklearn.metrics.cluster import silhouette_score
      2 print('silhouette:%0.2f'% silhouette_score(X, yhat))

silhouette:0.22
```



Affected greatly due to overlapping

Silhouette Score

- Advantages
  - The score is bounded [-1, 1], as -1 is the incorrect clustering and 1 is the correct clustering. Zero or approaching zero scores means overlapping results
  - When the score is high the clusters dense and separated well.

- Dis-advantages
  - The metric indicates high value with convex clusters than irregular shaped data

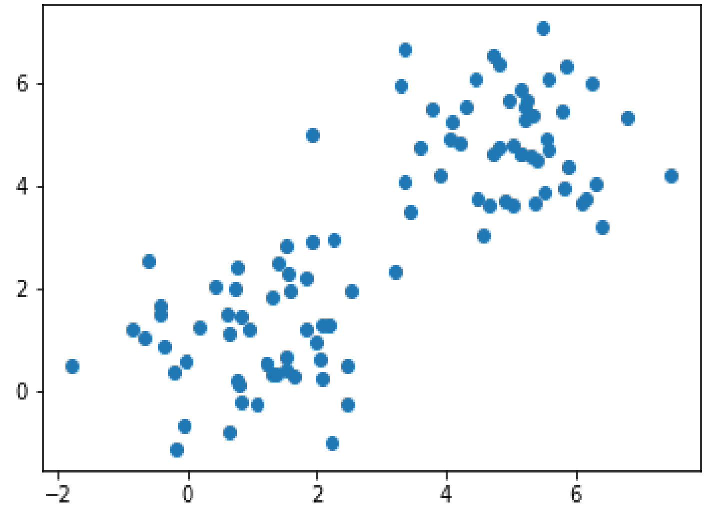
< / >
® Creative Commons licenses

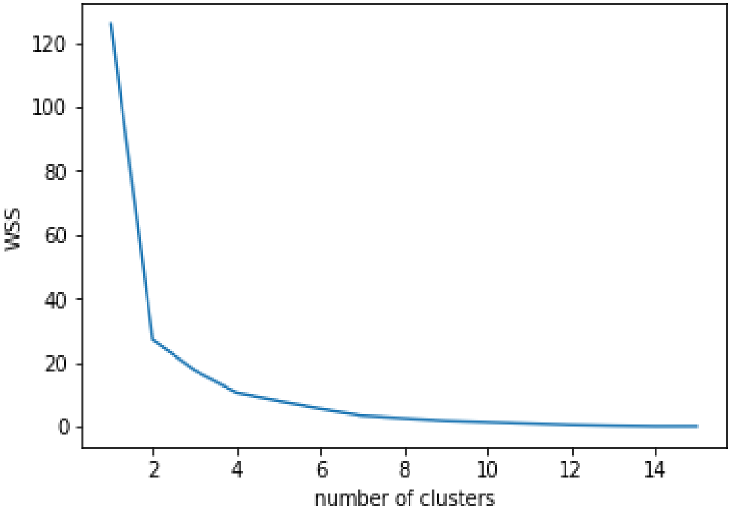
K value selection

- How many clusters should we identify?

What is the right K?

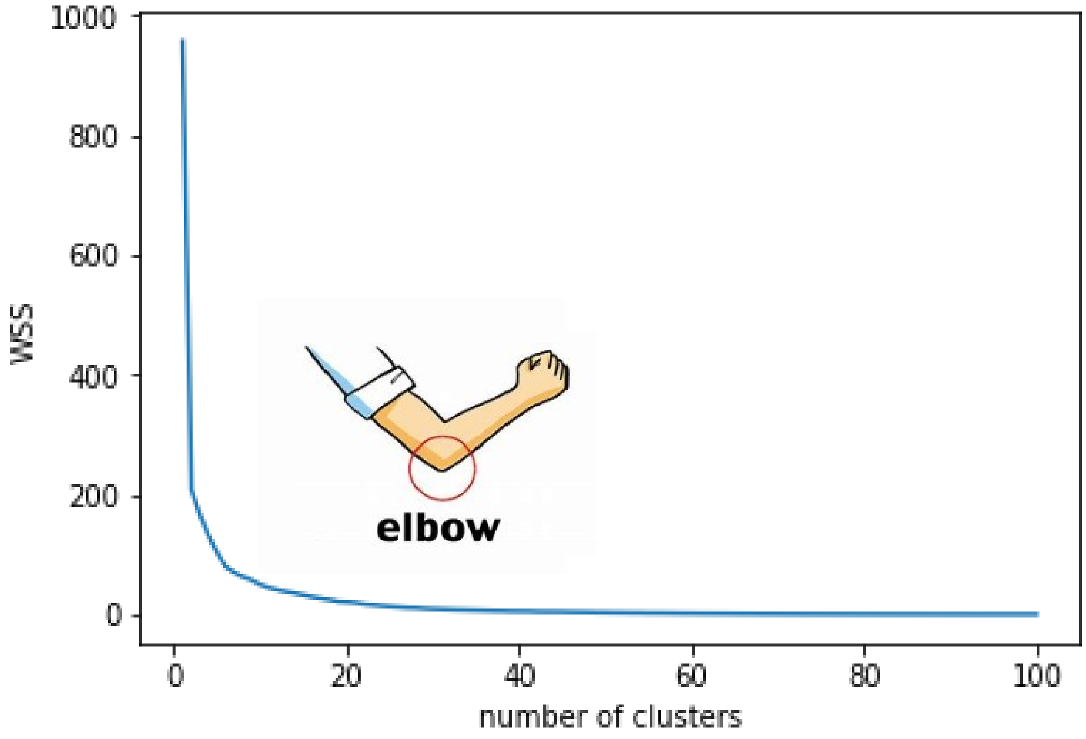
- Is there a criterion for selecting K ?
- The widely adopted method is the so-called Elbow-Method.
- In K-means clustering, we minimize the inertia value (WSS), and maximize the inter distance (BSS) simultaneously.
- By using a repeated clustering with different **K** each time and observe the WSS behavior. It will look something like:





Issue of Elbow method

- As we increase the K too much to select best K value, what happens to the WSS?

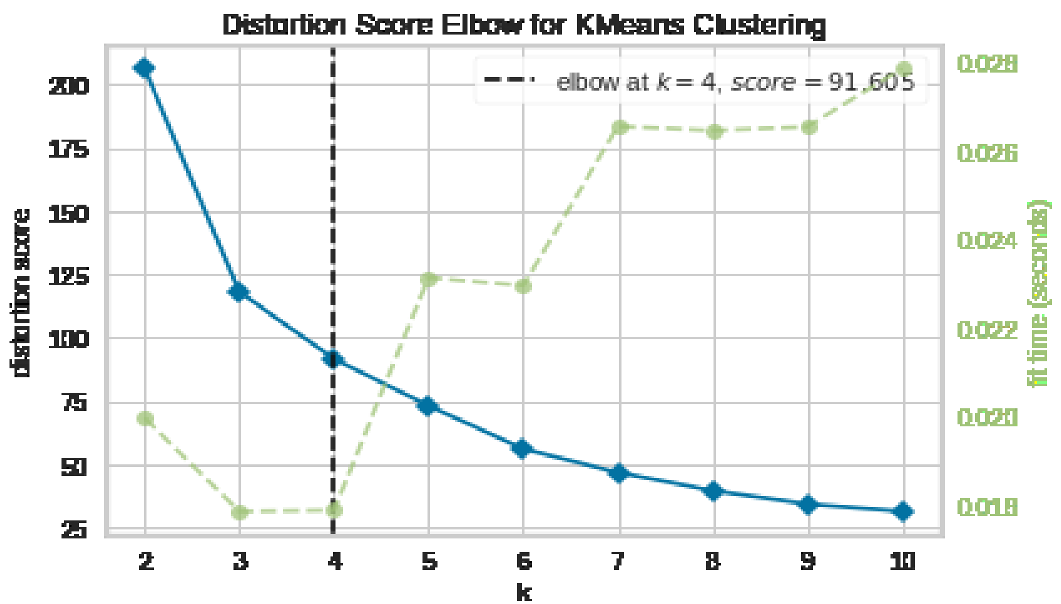


- WSS Decreases and smooths out the curve which makes it tough to decide the elbow location, However, a good guess always works here!

KElbowVisualizer (yellowbrick library)

```python
from yellowbrick.cluster import KElbowVisualizer

Elbow_M = KElbowVisualizer(KMeans(), k=10)
Elbow_M.fit(X)
Elbow_M.show()
plt.show()
```



- Yellowbrick library is an extension to the Scikit-Learn API. It is used for model selection and hyperparameter tuning. Under the hood, it’s using Matplotlib. ([link to doc](#))

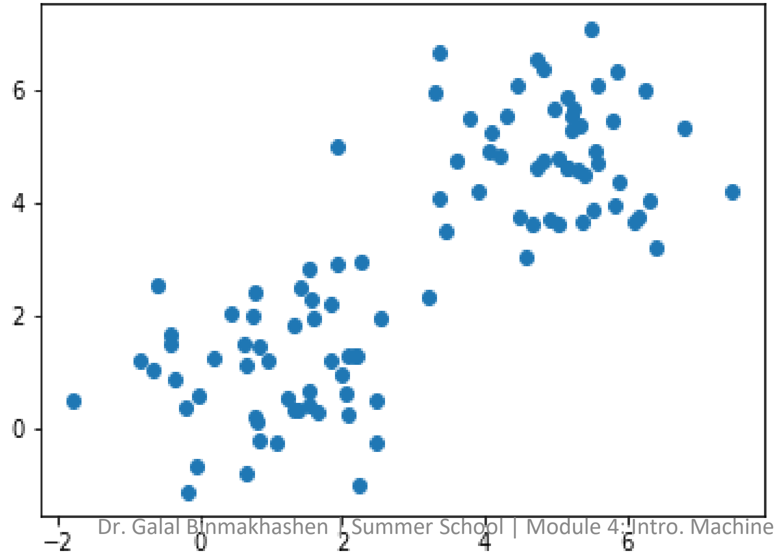
Silhouette-based K selection

- By intuition, as we aim to get better silhouette score, we may keep an eye on that metric to optimize K
- Silhouette tends to approach value of 1 when the cohesion of the clusters is high.
- One method is to repeat clustering of the data for several *Ks*. That is $k = \{1, \ldots, n\}$
- Then, we can analyze the results and choose one of the models with high silhouette scores

Results

- Silhouette score(2 clusters) =0.79
- Silhouette score(3 clusters) =0.63
- Silhouette score(4 clusters) =0.42

- With the above results, we may select the highest or the second highest based on our domain of expertise.

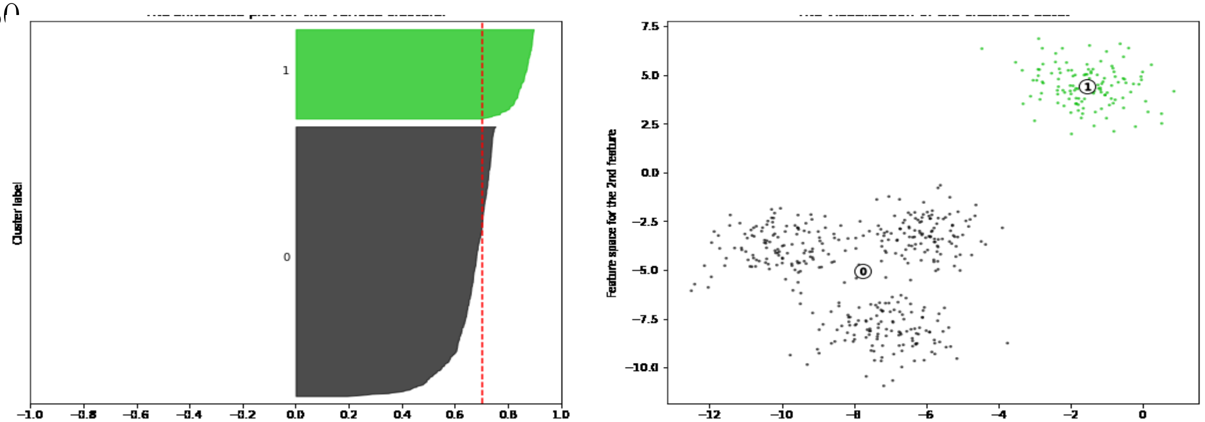


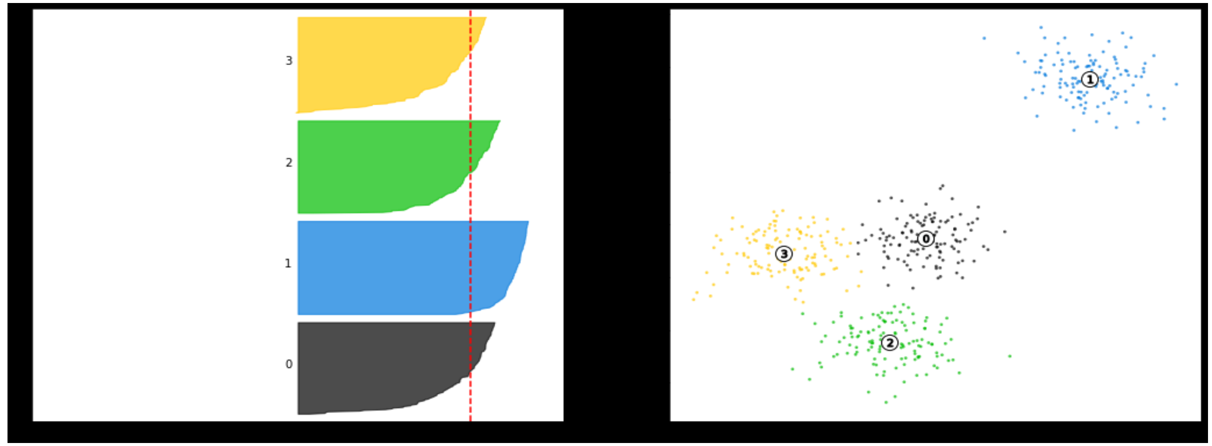
Another example

n_clusters = 2, silhouette_score is : 0.705, ←→ **n_clusters = 4, silhouette_score is : 0.651**

n_clusters = 3 silhouette_score is : 0.588, ←→ n_clusters = 5, silhouette_score is : 0.564

n_clusters = 6, silhouette_score is : 0.45





https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py

Summary

- Pros and Cons, and
- Possible remedies

Pros & Cons (Kmeans)

Pros

- ▷ Simple to understand
- ▷ Fast to cluster
- ▷ Suitable for large datasets
- ▷ Widely adopted
- ▷ Easy to implement

Cons

- ▷ Requires number of clusters as input
- ▷ Sensitive to initialization
- ▷ Lacks Consistency
- ▷ Sensitive to scale and outliers
- ▷ Spherical clusters

Some ideas

Cons

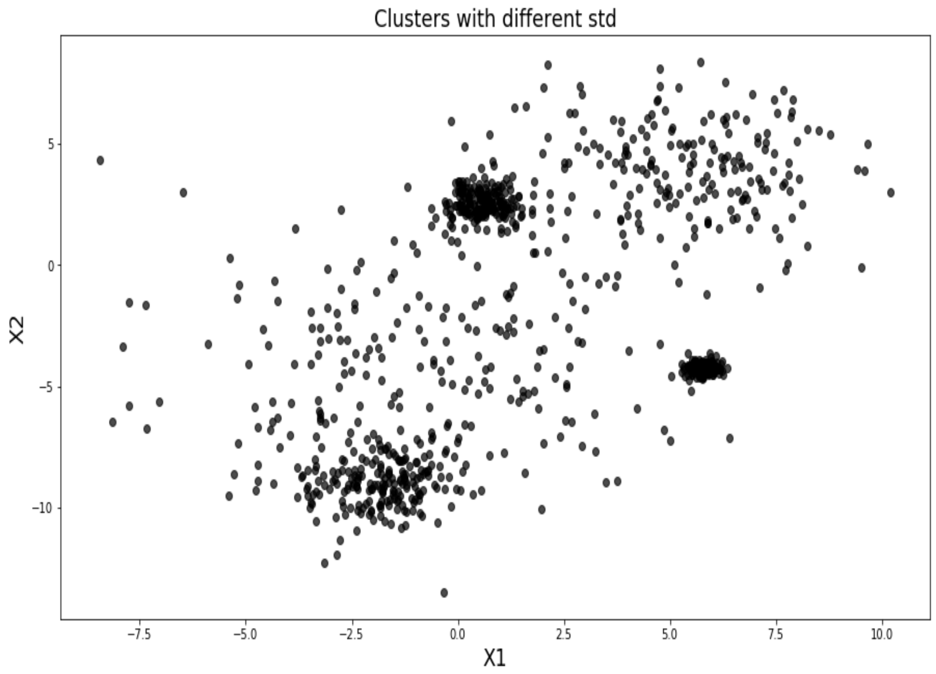
- ▷ Requires number of clusters as input
- ▷ Sensitive to initialization
- ▷ Lacks Consistency
- ▷ Sensitivity to scale and outliers
- ▷ Spherical clusters

Possible Remedy

- ▷ Elbow method
- ▷ Multiple runs with different centroid seeds
- ▷ Always remember to standardize data and get rid of single point clusters if occur twice

What kind of data kmeans can handle

- Generate a data with different spreads.



- Was Kmeans successful?

KMeans

- Data with different densities.

```
# different number of samples per cluster
Xneq_s = np.vstack((Xneq[yneq == 0][:30],
                    Xneq[yneq == 1][:100],
                    Xneq[yneq == 2][:150],
                    Xneq[yneq == 3][:99],
                    Xneq[yneq == 4][:50]))

yneq_s = np.hstack((yneq[yneq == 0][:30],
                    yneq[yneq == 1][:100],
                    yneq[yneq == 2][:150],
                    yneq[yneq == 3][:99],
                    yneq[yneq == 4][:50]))
```

Copy

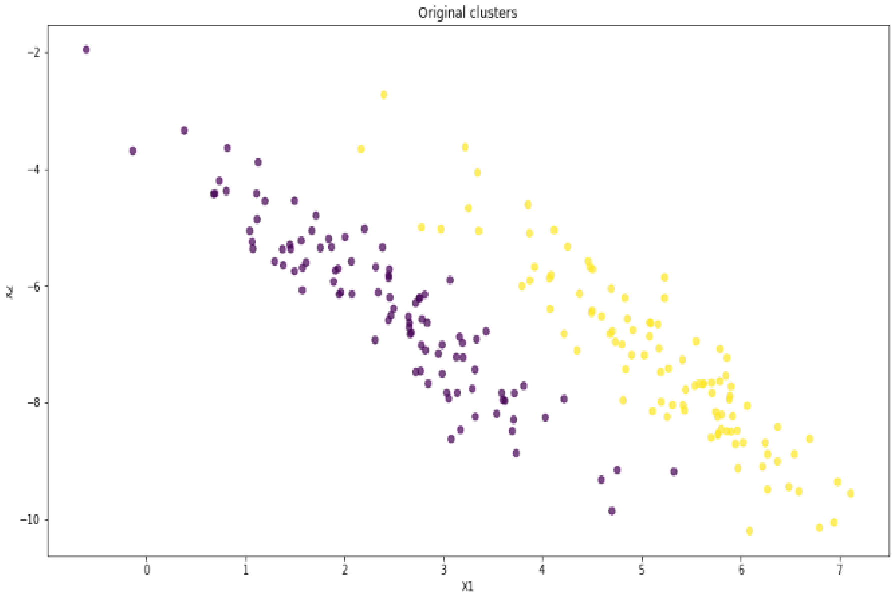
```
# starter code here
# different number of samples per cluster
Xneq_s = np.vstack((Xneq[yneq == 0][:30],
Xneq[yneq == 1][:100],
Xneq[yneq == 2][:150],
Xneq[yneq == 3][:99],
Xneq[yneq == 4][:50]))

yneq_s = np.hstack((yneq[yneq == 0][:30],
yneq[yneq == 1][:100],
yneq[yneq == 2][:150],
yneq[yneq == 3][:99],
yneq[yneq == 4][:50]))
```

- Was Kmeans successful?

Kmeans

- Skewed datasets
- circles, and moons datasets



Copy

To skew

```
# starter code
X, y = datasets.make_blobs (n_samples=200, centers =2, n_features=2, cluster_std=1.5, random_state=40)

# Anisotropicly distributed data Scikit (example)
transformation = [[0.60, -0.63], [-0.40, 0.85]]
X_aniso = np.dot(X, transformation)
```


< / >
® Creative Commons licenses