ML: Clustering

ECE30007 Intro to AI Project

한동대학교
HANDONG GLOBAL UNIVERSITY

Contents

- PCA review
- Unsupervised learning?
- Clustering
  - Hierarchical methods → bottom-up/top-down +code
  - Centroid model → K-means
- k-means
  - algorithm
  - pros and cons
  - + basic code

- K-means sklearn +code
- application to MNIST +code
  - PCA
  - clustering

Exercise (1) – Let's make 2-dim MNIST data

```
import pickle
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

print('... loading data')
with open('data/mnist.pkl', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')
```

```
... loading data
```

```
train_x, train_y = train_set
test_x, test_y = test_set

train_x = pd.DataFrame(train_x)
train_y = pd.DataFrame(train_y, columns=['label'])
test_x = pd.DataFrame(test_x)
test_y = pd.DataFrame(test_y, columns=['label'])
```

Exercise (1) – Let's make 2-dim MNIST data

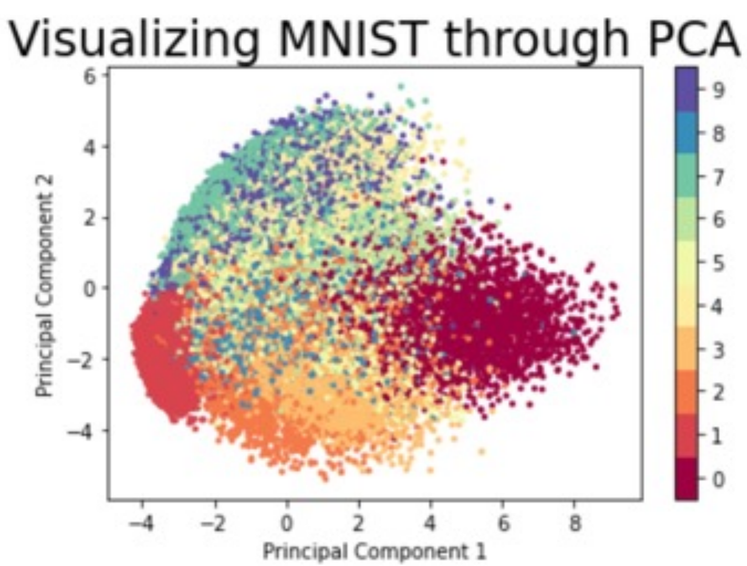
```
from sklearn.decomposition import PCA

mypca = PCA(n_components = 2)
PCA_train_x = mypca.fit_transform(train_x)
PCA_test_x = mypca.transform(test_x)
```

```
print('PCA shape: ', PCA_train_x.shape, PCA_test_x.shape)
```

```
PCA shape:  (50000, 2) (10000, 2)
```

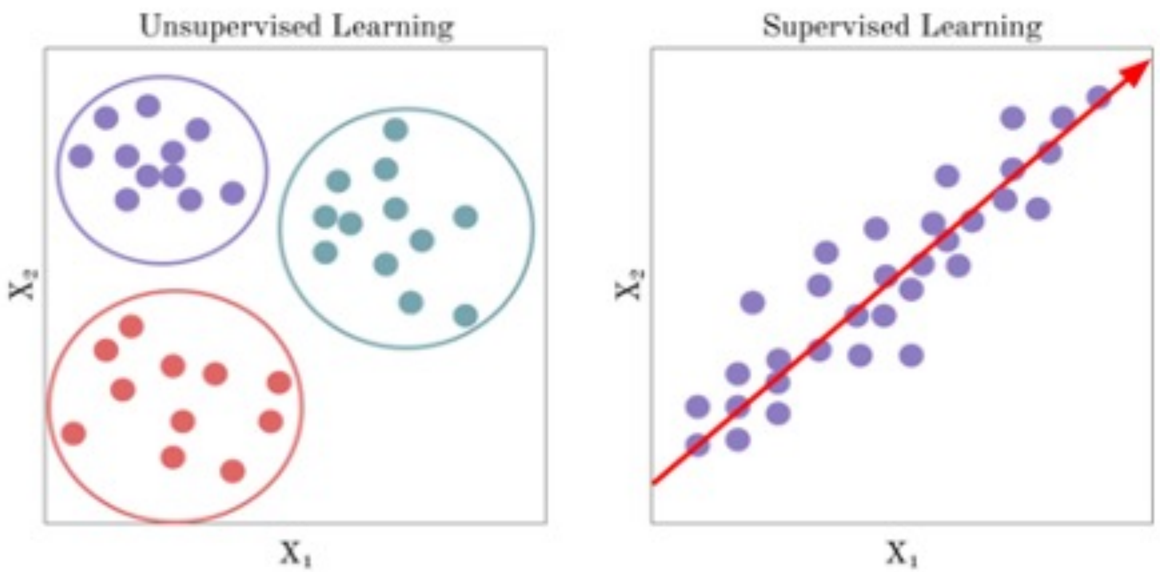
```
## Plot on the graph
plt.scatter(PCA_train_x[:, 0], PCA_train_x[:, 1], s=5, c=train_y['label'], cmap='Spectral')
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))

plt.title('Visualizing MNIST through PCA', fontsize=24);
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

```
Text(0, 0.5, 'Principal Component 2')
```



Unsupervised learning

- Not using labeled data (cf. supervised learning), but instead focuses on the data's feature.
- **Goal**
  - Analyze data and find important features



Clustering

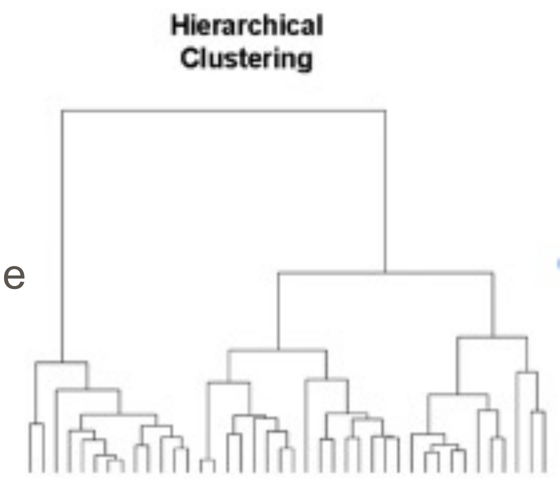
- Task of discovering innate groups of data
- Given a cloud of data points, we would like to understand their structure
- Example

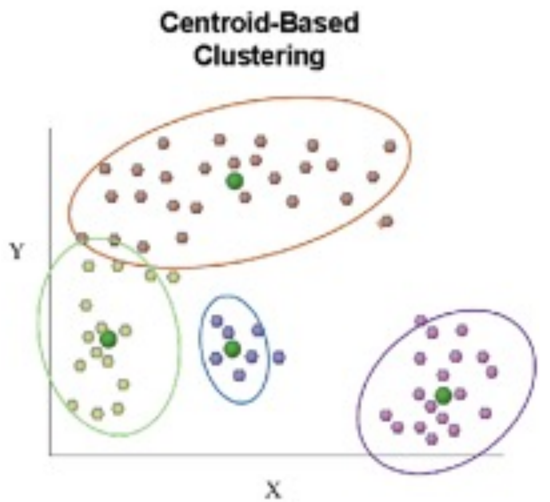
Clustering

- Given a set of data points, group the points into a specified number of clusters, such that
  - Members of a cluster are similar to each other
  - Members of different clusters are dissimilar
    - Similarity is defined by a distance metric e.g.,
      - Euclidean distance, Cosine dissimilarity…
- Why do we need clustering?
  - Problems often involve too many data points in a high-dimensional space

Clustering

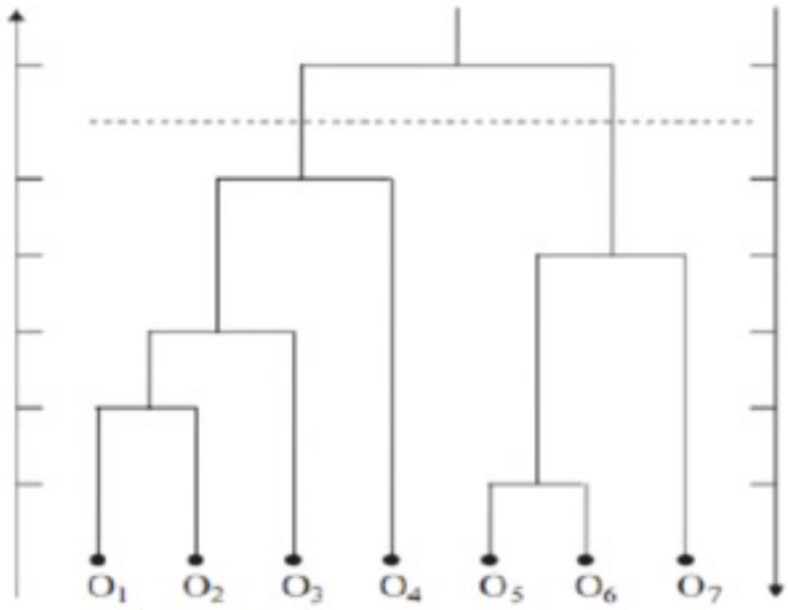
- Example types of methods
  - **Hierarchical model**
    - Agglomerative (bottom-up)
      - Initially, each data point is one cluster.
      - Repeatedly combine two “nearest” clusters into one
    - Divisive (top-down)
      - Initially, all data points are in one big cluster
      - Recursively split clusters
  - **Centroid model**
    - K-means
      - Maintain a set of clusters
      - Each data point belongs to its ”nearest” cluster





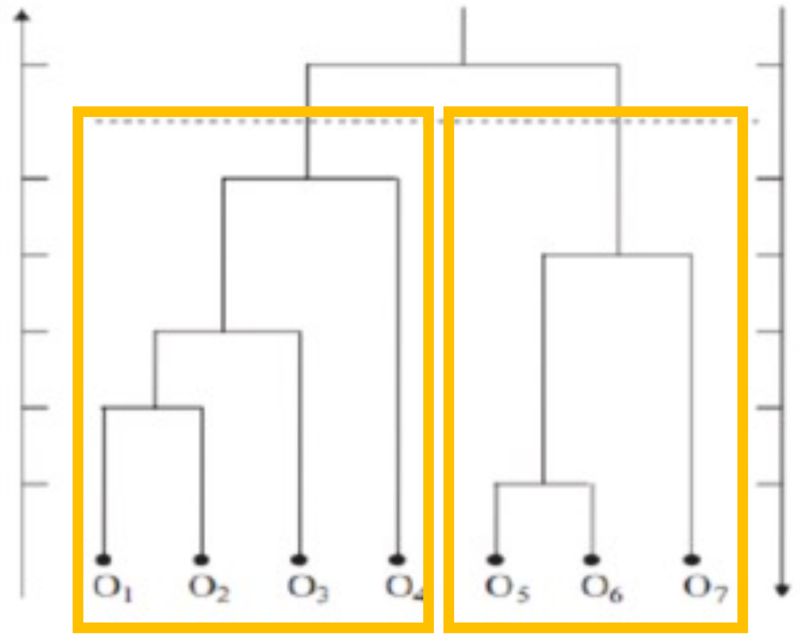
Hierarchical Clustering

- Idea: build a tree-like hierarchical taxonomy(dendrogram) from a set of data points



Hierarchical Clustering

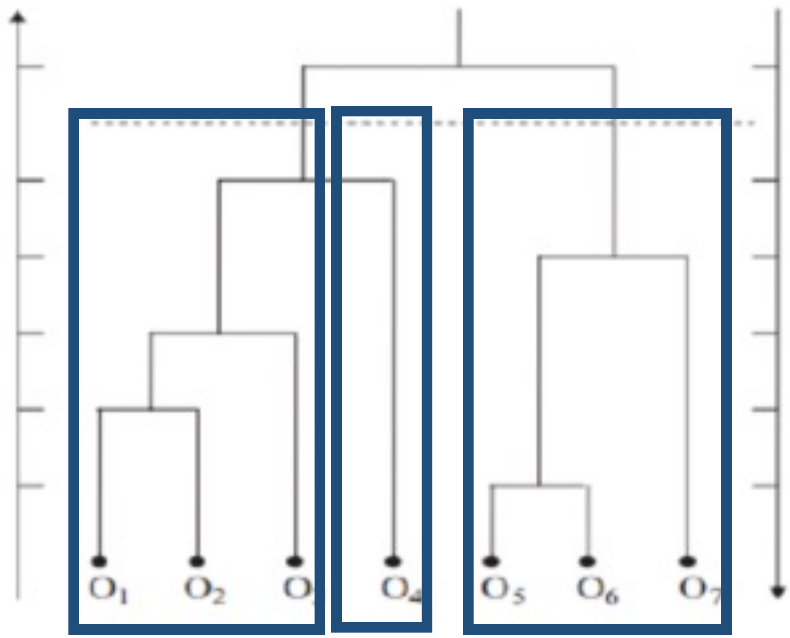
- Dendrogram in hierarchical clustering
  - Clustering obtained by cutting the dendrogram at a desired level
  - Each connected component forms a cluster



When k=2, Cluster 1 Cluster 2

Hierarchical Clustering

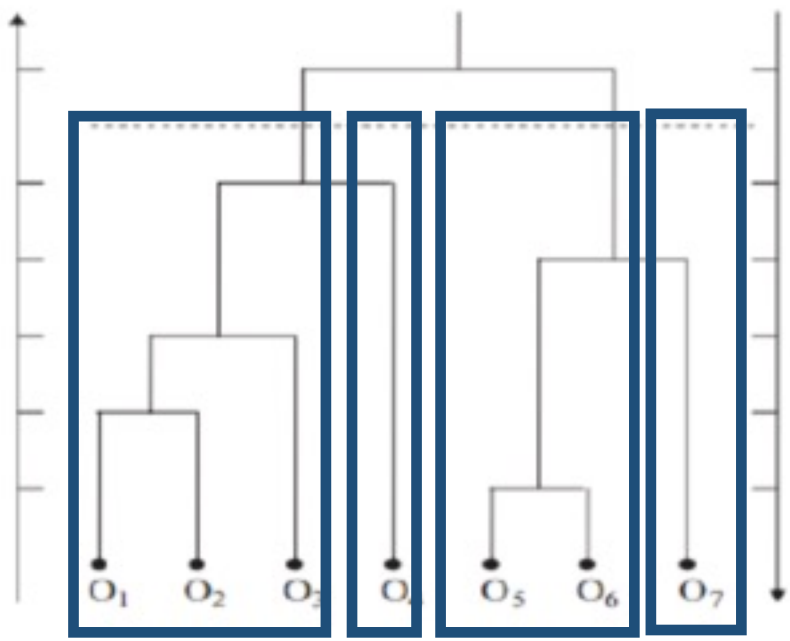
- Dendrogram in hierarchical clustering
  - Clustering obtained by cutting the dendrogram at a desired level
  - Each connected component forms a cluster



When k=3, Cluster 1 Cluster 2 Cluster 3

Hierarchical Clustering

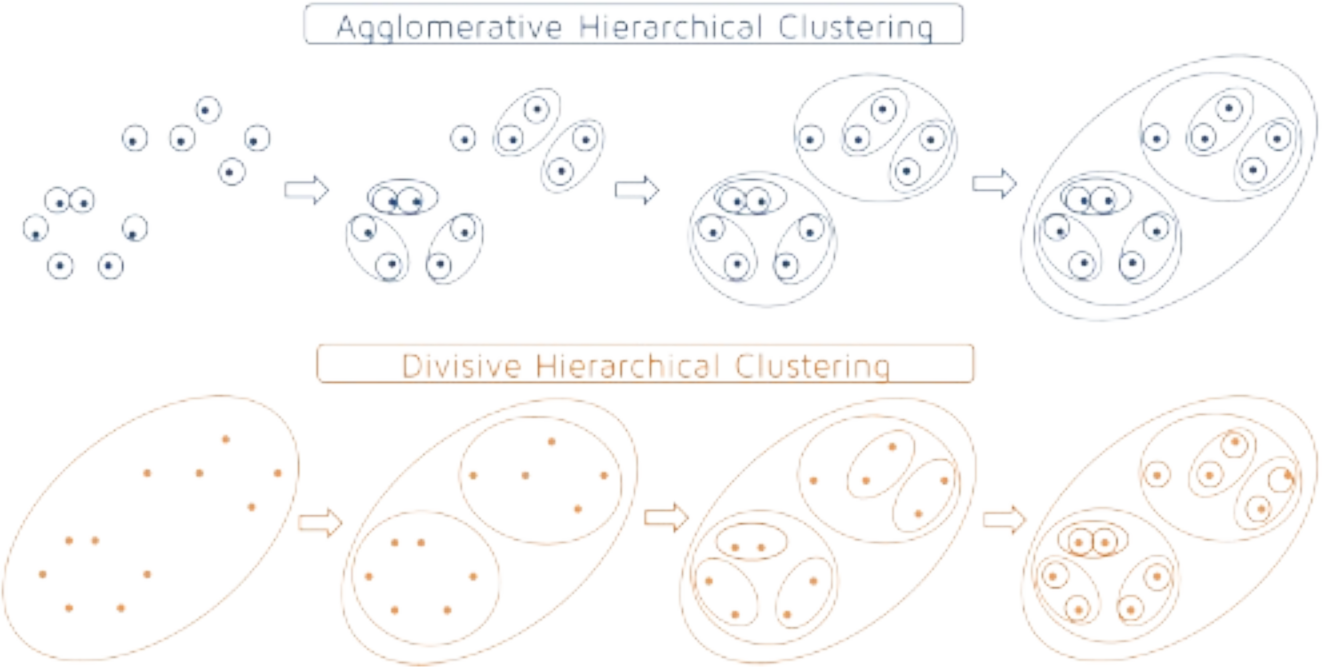
- Dendrogram in hierarchical clustering
  - Clustering obtained by cutting the dendrogram at a desired level
  - Each connected component forms a cluster



When k=4, Cluster 1 Cluster 2 3 4

Hierarchical Clustering

- Type
  1. **Agglomerative (bottom-up)**
  2. Divisive (top-down)



https://quantdare.com/hierarchical-clustering/

Agglomerative Clustering

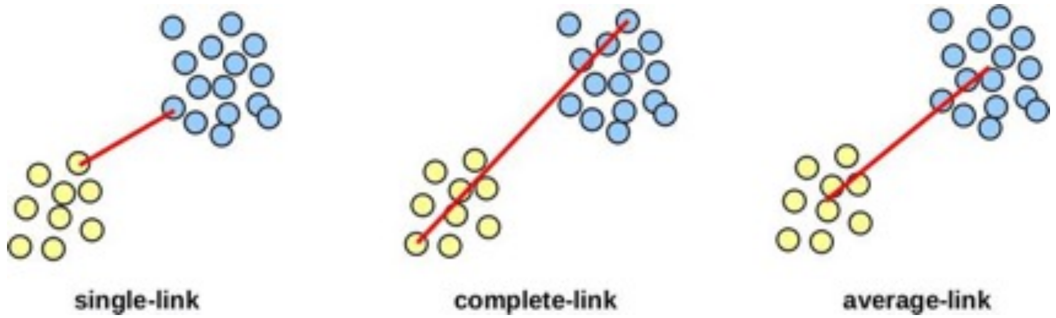
- Idea
  1. Initially, each data point is one cluster
  2. Repeatedly combine two “nearest” clusters into one
  - The history of the combining process forms a hierarchy(dendrogram)

- Key questions
  1. How do you represent a cluster of more than one point?
  2. How do you determine the “nearness” of clusters?
  3. When do you stop combining clusters?

Agglomerative Clustering

- Key questions
  1. **How do you represent a cluster of more than one point?**
     - How do you represent the location of each cluster, to tell which cluster is closest?
     - Represent each cluster by its centroid = *average of the cluster members*

  2. How do you determine the **"nearness"** of clusters?
     1. Measure the distance between centroids (group average link)
     2. Measure the distance between two closest points (single link)
     3. Measure the distance between two furthest points (complete link)



Agglomerative Clustering

- Key questions
  3. When do you **stop combining clusters?**
     1) Approach 1 : Pick a number *k upfront,* and stop when we have k clusters
        - Makes sense when we know that the data naturally falls into k classes

     2) Approach 2 : Stop when the next merge would create a cluster with low cohesion (a bad cluster)

Agglomerative Clustering

- Nearest neighbor

	A	B	C	D
A	0	10	3	5
B		0	25	8
C			0	12
D				0

A B C D

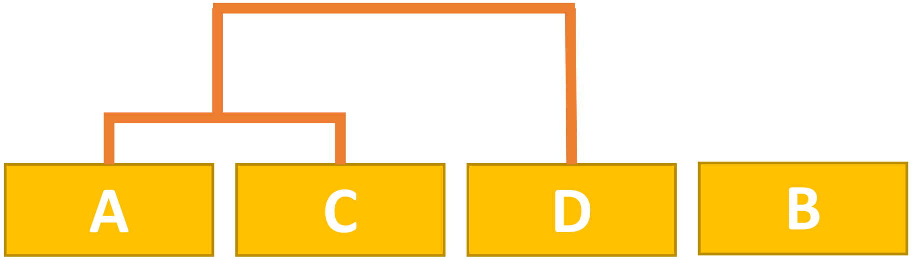
Agglomerative Clustering

- Nearest neighbor

	AC	B	D
AC	0	10	5
B		0	8
D			0

Agglomerative Clustering

- Nearest neighbor



	ACD	B
ACD	0	8
B		0

It costs expensive time and space!

Exercise(2) – Agglomerative clustering

- Let's just use 1000 data samples.

```
sub_PCA_train_x = PCA_train_x[:1000, :]
print('sub_PCA_train_x.shape: ', sub_PCA_train_x.shape)
```

```
sub_PCA_train_x.shape:  (1000, 2)
```

- Doing agglomerative clustering with sklearn library

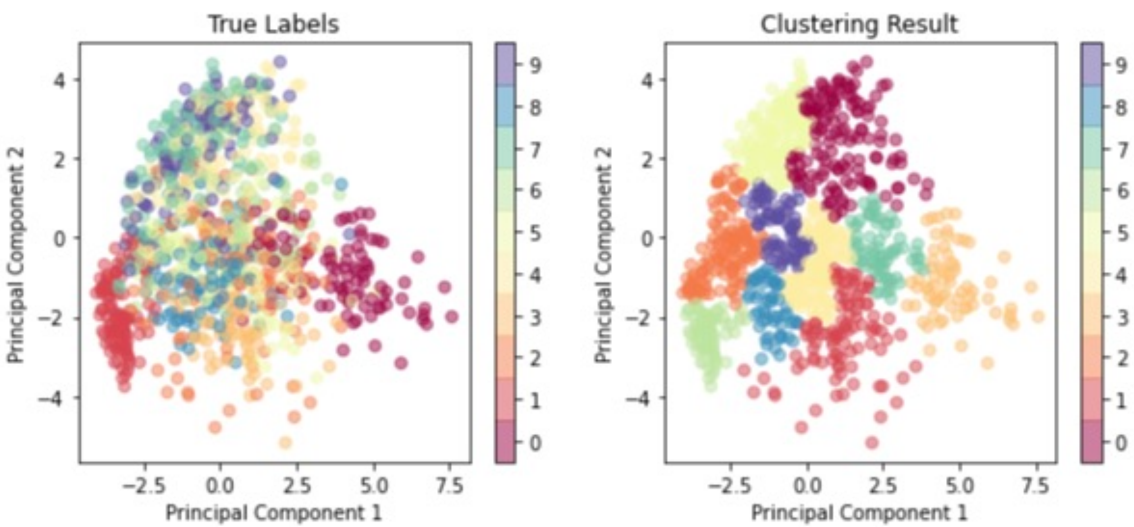
```
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt

hier = AgglomerativeClustering(n_clusters=10, affinity='euclidean')
hier_clusters = hier.fit(sub_PCA_train_x)
```

Exercise(2) – Agglomerative clustering

true label

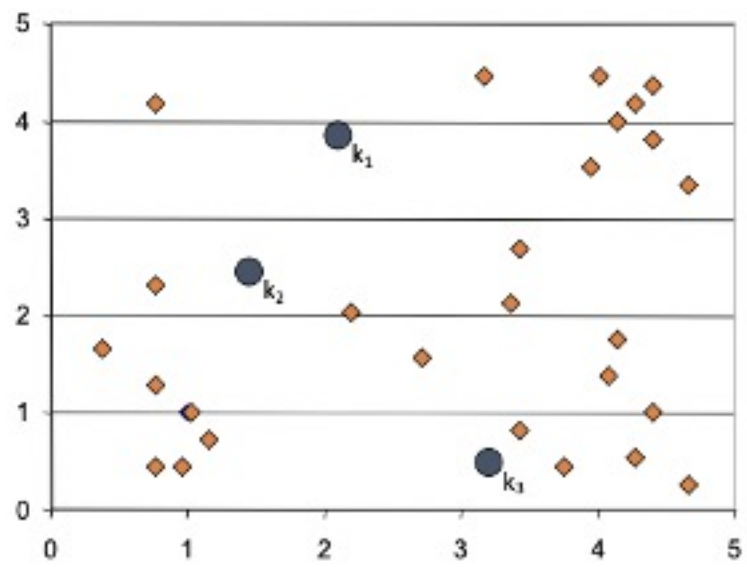
cluster index

```
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.scatter(sub_PCA_train_x[:, 0], sub_PCA_train_x[:, 1], c=train_y['label'][:1000], cmap='Spectral', alpha=0.5)
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title("True Labels")
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.subplot(1,2,2)
plt.scatter(sub_PCA_train_x[:, 0], sub_PCA_train_x[:, 1], c=hier_clusters.labels_[:1000], cmap='Spectral', alpha=0.5)
plt.colorbar(boundaries=np.arange(11)-0.5).set_ticks(np.arange(10))
plt.title("Clustering Result")
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.show()
```



Centroid Clustering : k-Means

- Idea
  1. Start by picking **k**, the number of clusters
  2. Initialize clusters by picking one point per cluster(initial cluster center)
     - For the moment, we assume to pick the k points **at random**

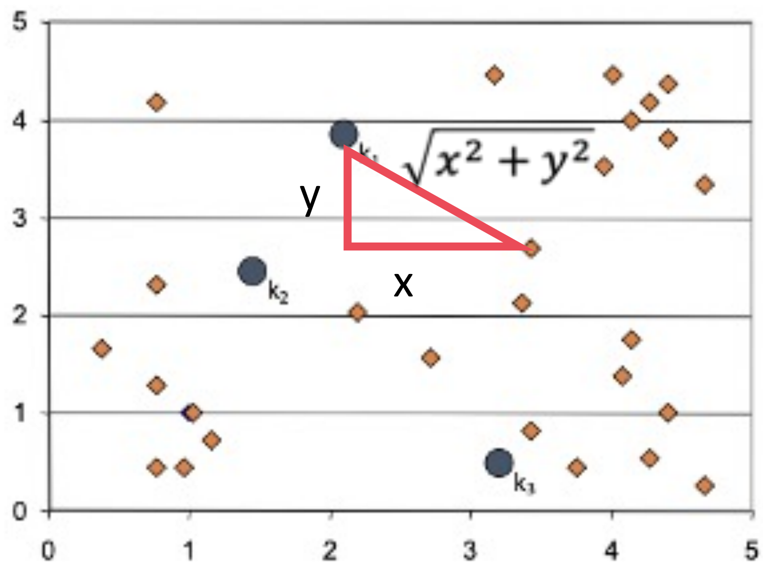


k = 3

Centroid Clustering : k-Means

- Idea

3. For each data point, decide the cluster membership by assigning it to the nearest cluster center
    - Distance Metric: **Euclidean Distance**



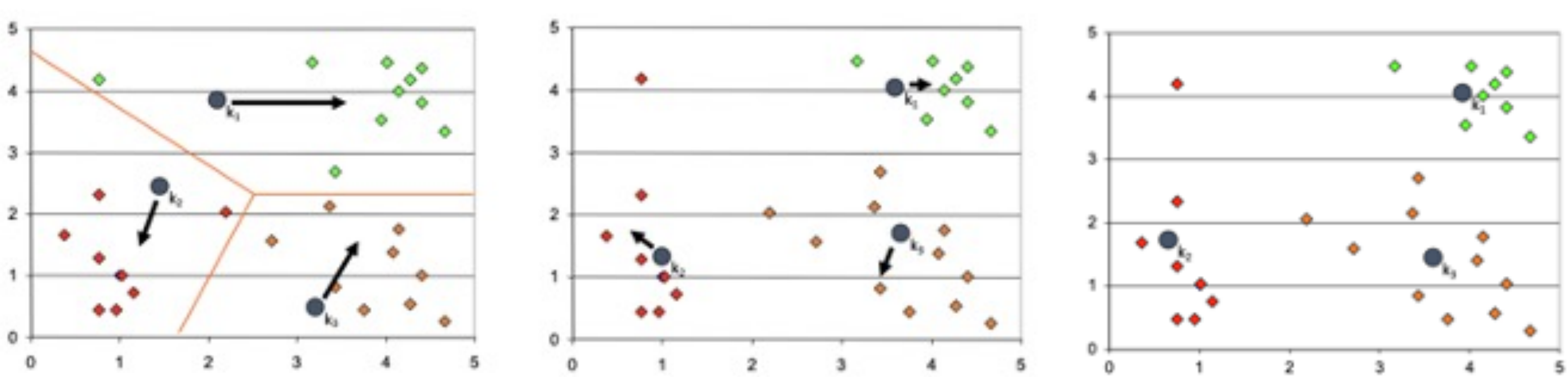
Centroid Clustering : k-Means

- Idea

  3. For each data point, decide the cluster membership by assigning it to the nearest cluster center
  4. Re-estimate the cluster centers, by taking the average of the cluster members

  - Repeat 3-4 until convergence

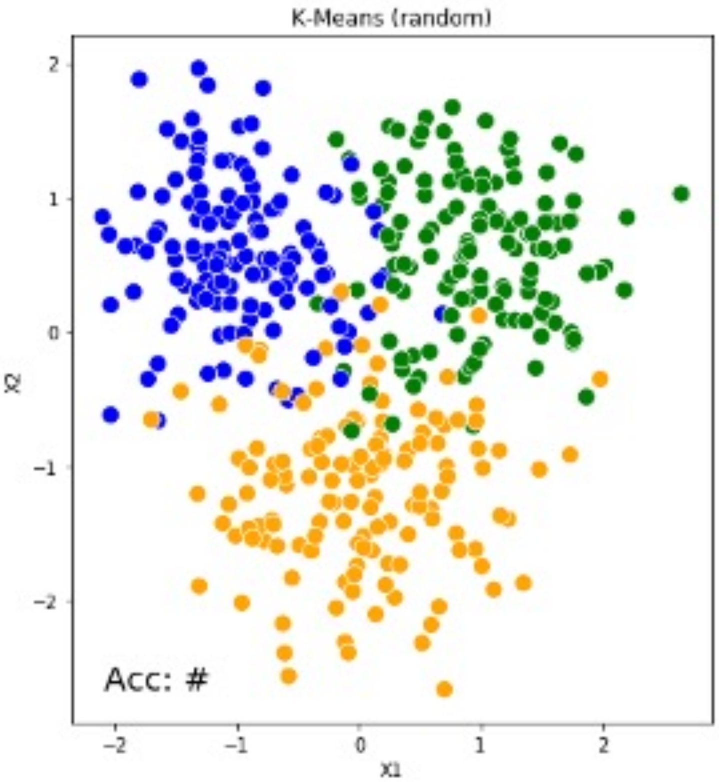
Data points do not move between clusters & centroids stabilize



Centroid Clustering : k-Means

- Idea
  - Repeat 3-4 until convergence

Data points do not move between clusters & centroids stabilize



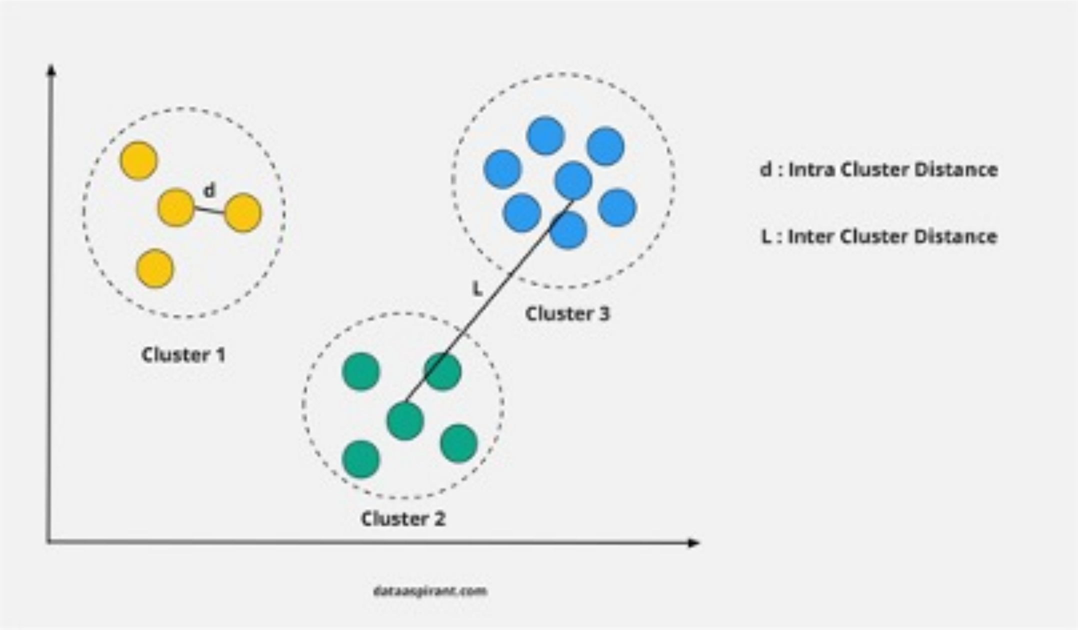
k = 3

Centroid Clustering : k-Means

- **Advantages**
  - Easy to understand and implement
  - Efficient: O(nKT)
    - n = number of data points
    - K = number of clusters
    - T = number of iterations
    - Normally, K, T, << n

What is Good Clustering?

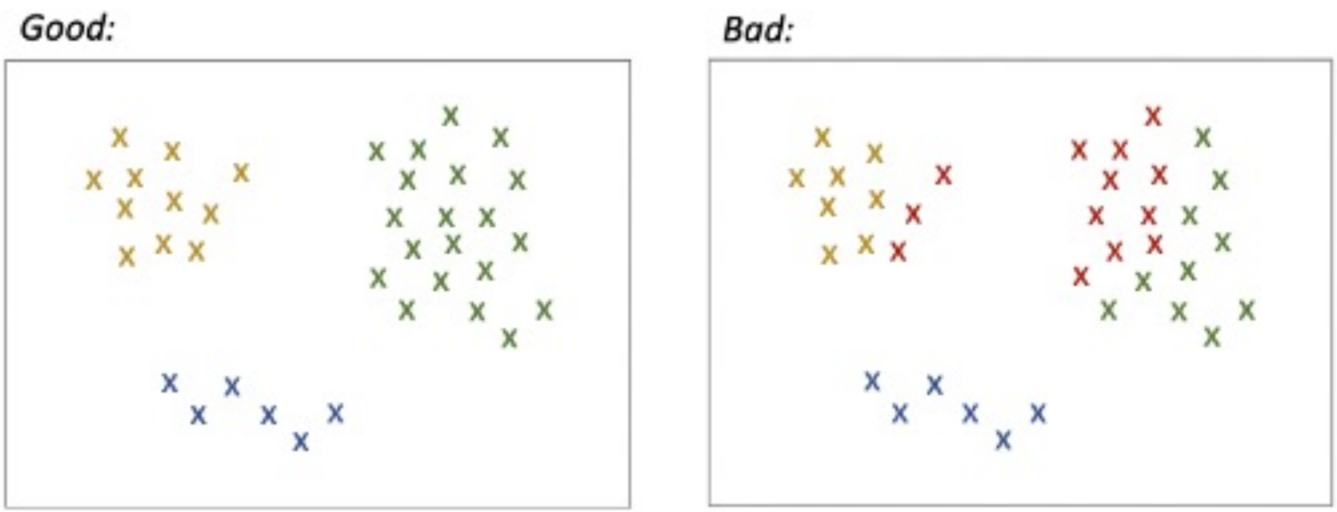
- **Internal criterion:** a good clustering will result in
  - The intra-cluster similarity is high
  - The inter-cluster similarity is low



https://dataaspirant.com/8-intra-cluster-distance-and-inter-cluster-distance/

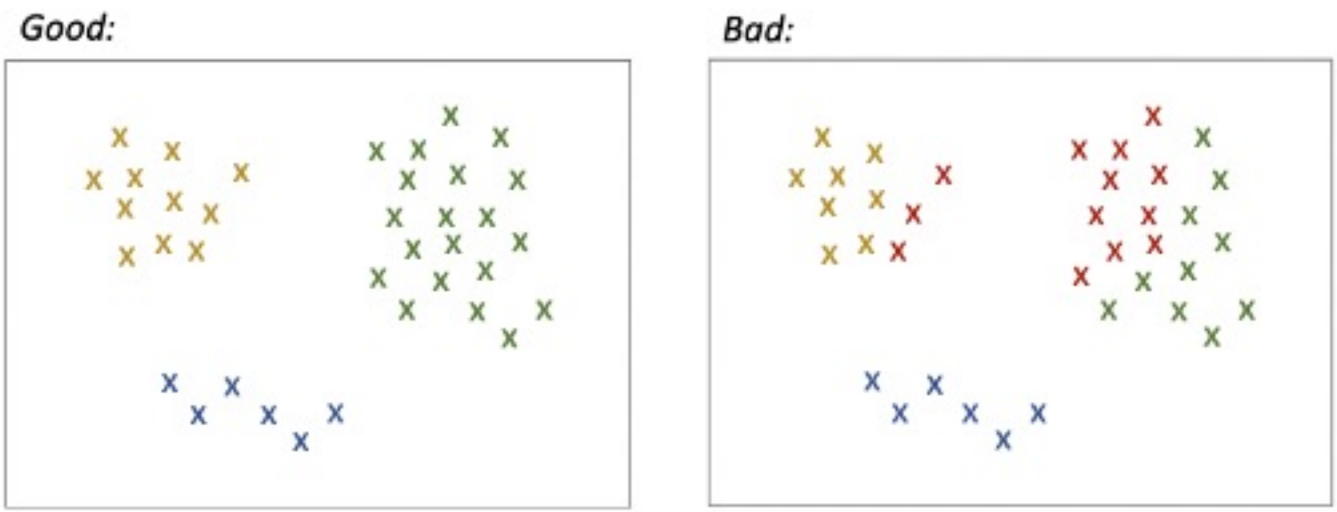
What is Good Clustering?

- **Internal criterion:** a good clustering will result in
  - The intra-cluster similarity is high
  - The inter-cluster similarity is low



What is Good Clustering?

- **External criterion:** Quality evaluated by its ability to discover some or all of the hidden patterns or latent classes in ground truth information
  - The requires labeled data

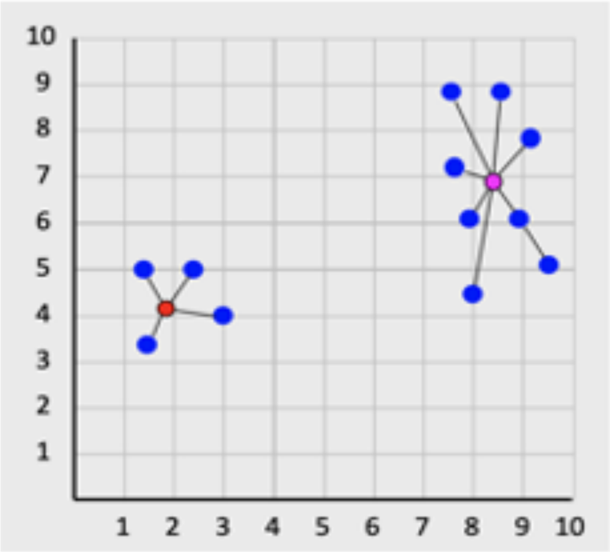


Can we pick *k Automatically?*

- Metric : Squared error

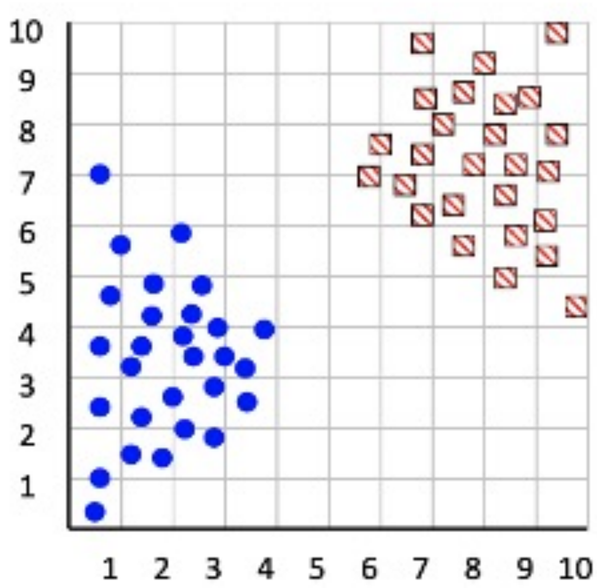
$$\sum_{i=1}^{k}\sum_{x\in S_i}\|x-m_i\|^2$$

where S_i denotes the sample set of i^{th} cluster and m_i is centroid of i^{th} cluster



Example: Can We Pick *k* Automatically?

- The katydid/grasshopper dataset

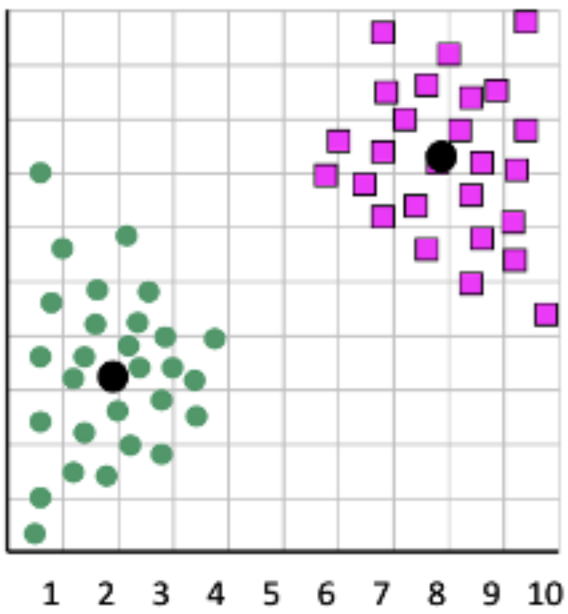


Example: Can We Pick *k* Automatically?

- When k=1, the objective function yields 873.0

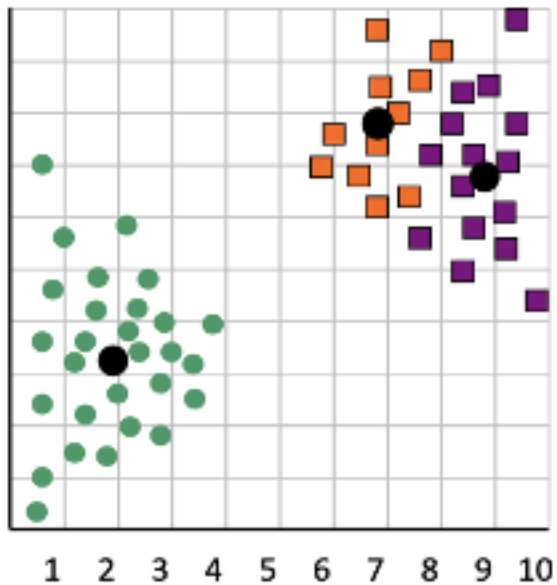
Example: Can We Pick *k* Automatically?

- When k=2, the objective function yields 173.1



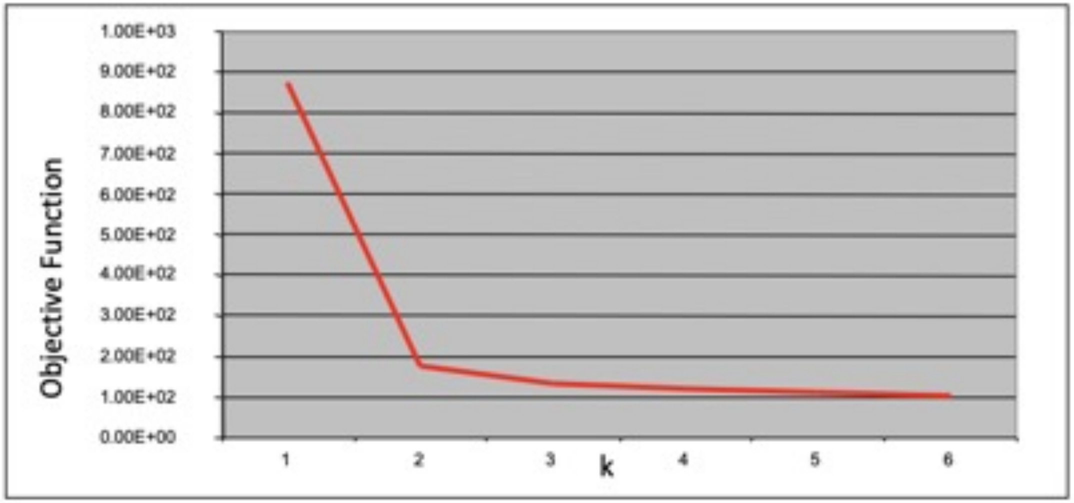
Example: Can We Pick *k* Automatically?

- When k=3, the objective function yields 133.6



Example: Can We Pick *k* Automatically?

- We can plot the objective function values for k equals 1 to 6
    - The abrupt change at *k* = 2 is highly suggestive of two clusters in the data
    - This technique for determining the number of clusters is known as “knee finding” or “elbow finding”



k-Means scratch

```python
a = np.array([1, 3, 4])
A = np.tile(a, (3,1))
A
```

```
array([[1, 3, 4],
       [1, 3, 4],
       [1, 3, 4]])
```

Construct an array by repeating A the number of times given by reps.

- **STEP1** : Normalize data

```python
def apply_normalizer(dataset, offset, divisor):
    dataset_normalized = np.zeros(dataset.shape)
    N = dataset.shape[0]
    dataset_normalized = dataset - np.tile(offset, (N,1))
    dataset_normalized = dataset_normalized / np.tile(divisor, (N,1))

    return dataset_normalized


def normalize_minmax(dataset):
    minval = dataset.min(0)
    maxval = dataset.max(0)

    dataset_normalized = apply_normalizer(dataset, minval, maxval-minval)

    return dataset_normalized, minval, maxval-minval


def normalize_meanstd(dataset):
    meanval = dataset.mean(0)
    stdval = dataset.std(0)

    dataset_normalized = apply_normalizer(dataset, meanval, stdval)

    return dataset_normalized, meanval, stdval
```

k-Means scratch

- **STEP1** : Normalize data

```
normalized_PCA_train_x, off, div = normalize_minmax(sub_PCA_train_x)
print("Original data: ", sub_PCA_train_x[0], '\nNormalized data: ', normalized_PCA_train_x[0])
print("offset:", off, ";  divisor:", div, '\n')

normalized_PCA_train_x, off, div = normalize_meanstd(sub_PCA_train_x)
print("Original data: ", sub_PCA_train_x[0], '\nNormalized data: ', normalized_PCA_train_x[0])
print("offset:", off, ";  divisor:", div)
```

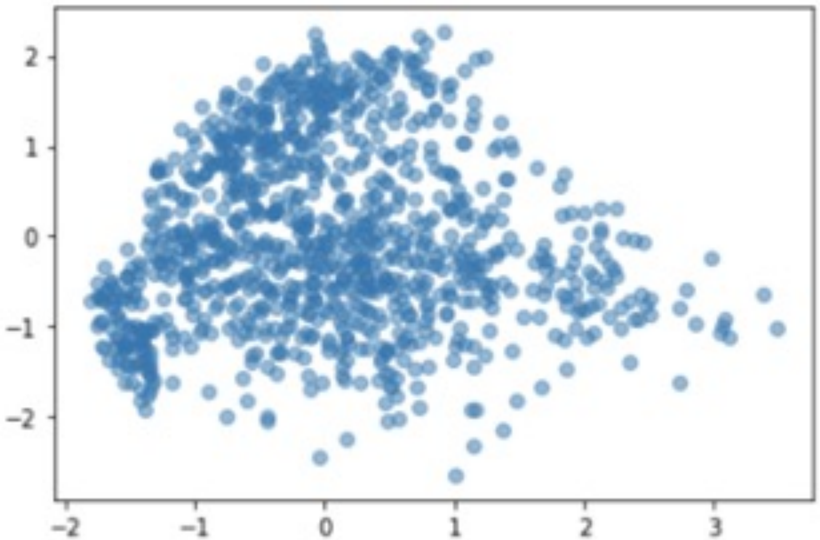
```
Original data:  [ 0.46153015 -1.2470015 ]
Normalized data:  [0.38819835 0.4091687 ]
offset: [-4.0606523 -5.1655836] ;  divisor: [11.649155  9.576935]

Original data:  [ 0.46153015 -1.2470015 ]
Normalized data:  [ 0.24439529 -0.6388204 ]
offset: [-0.0758791  -0.00492685] ;  divisor: [2.1989346 1.9443254]
```

k-Means scratch

- **STEP1** : Normalize data

```
plt.scatter(normalized_PCA_train_x[:, 0], normalized_PCA_train_x[:, 1], cmap='Spectral', alpha=0.5)
```

```
<matplotlib.collections.PathCollection at 0x12af95510>
```



k-Means scratch

- **STEP2** : Initialize centroids randomly or with first *k instances*

```
k = int(input("How many cluster do you want? "))
print(k)
```

```
How many cluster do you want? 10
10
```

```
import random

def init_centroids_random(dataset, k):
    centroids = {}
    init_centroids = random.sample(range(0, len(dataset)), k)

    for i, c in enumerate(init_centroids):
        centroids[i] = dataset[c]
    return centroids
```

```
def init_centroids_index(dataset, k):
    centroids = {}
    for i in range(k):   # first k instances become the initial centroids
        centroids[i] = dataset[i]
    return centroids
```

k-Means scratch

- **STEP2** : Initialize centroids randomly or with first *k instances*

```
# initialize_centroids(centroids, sub_PCA_train_x)
centroids = init_centroids_random(sub_PCA_train_x, k)
```

```
cet_df = pd.DataFrame(centroids).transpose()
cet_df.columns = ['X', 'Y']
cet_df.head()
```

Transpose index and columns.

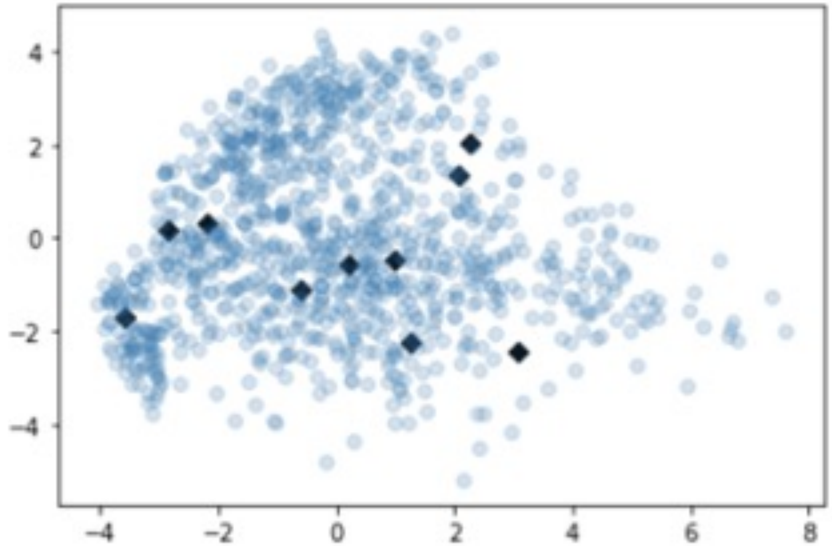
	X	Y
0	1.260929	-2.252848
1	-2.195889	0.337217
2	3.076864	-2.450219
3	0.983099	-0.469126
4	-2.872622	0.170720

k-Means scratch

- **STEP2** : Initialize centroids randomly or with first *k instances*

```
plt.figure()
plt.scatter(cet_df['X'], cet_df['Y'], color='black', marker='D')
plt.scatter(sub_PCA_train_x[:, 0], sub_PCA_train_x[:, 1], alpha=0.2)
```

```
<matplotlib.collections.PathCollection at 0x1a2f010210>
```



k-Means scratch

- **STEP3** : (Re)assigning every data to its closest centroid

```python
# a distance function
def Euclidean_distance(vecA, vecB):
    return np.sqrt(sum(np.power([a - b for a, b in zip(vecA, vecB)], 2)))
```

```python
def re_assign_data(dataset, centroids):
    # (Re)assigning every instance to its closest centroid
    cluster_memberships = {}
    for i in centroids:
        cluster_memberships[i] = []

    for row in dataset:
        # Calculate euclidean distance between each centroid and each data.
        dist_to_centroids = [Euclidean_distance(row, centroids[c]) for c in centroids]

        # Find the centroid with a minimum distance
        membership = dist_to_centroids.index(min(dist_to_centroids))
        cluster_memberships[membership].append(row)
    return cluster_memberships
```

k-Means scratch

- **STEP4** : Recalculate average of each cluster and calculate SSE value

```
def re_calc_avg_sse(centroids, cluster_memberships):
    # Re-calculate the average of each cluster and calculate SSE.
    curr_sse = 0

    for membership in cluster_memberships:
        centroids[membership] = np.average(cluster_memberships[membership], axis=0)

        for row in cluster_memberships[membership]:
            curr_sse += np.power(Euclidean_distance(row, centroids[membership]), 2)

    return centroids, curr_sse
```

k-Means scratch

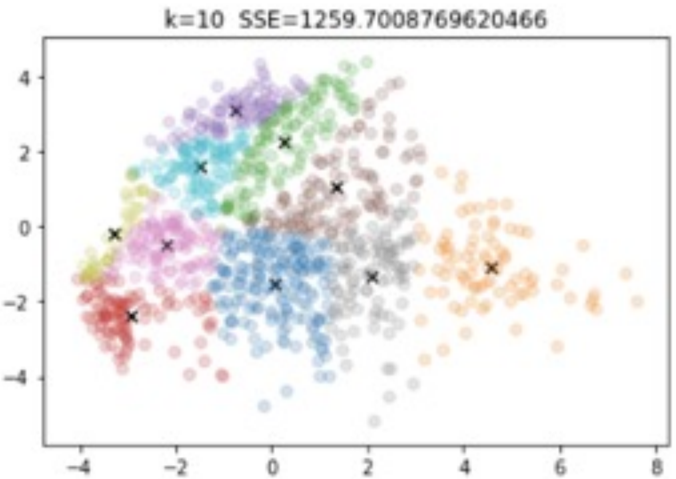
- **STEP5** : Iterate STEP3 and STEP4 until SSE is less than ‘tol’ value

```python
## k-Means algorithm
def kmeans(dataset, k, max_iter = 300, tol = 0.001):
    centroids = init_centroids_random(dataset, k) # or init_centroids_random(dataset, k)

    ## 1. Initiate SSE (sse = sum of squared error) into 'np.inf'
    curr_sse = np.inf

    ## 2. Clustering
    for i in range(max_iter):

        ## (Re)Aassign datas to its closest centroids
        cluster_memberships = re_assign_data(dataset, centroids)

        ## Re-calculate the average of each cluster and calculate SSE.
        prev_sse = curr_sse
        centroids, curr_sse = re_calc_avg_sse(centroids, cluster_memberships)

        ## Plot center points and assigned data.
        plt.figure(i)
        c_df = pd.DataFrame(centroids).transpose()
        plt.scatter(c_df.loc[:, 0], c_df.loc[:, 1], color='black', marker='x')
        for key in cluster_memberships:
            plt.scatter(*zip(*cluster_memberships[key]), alpha=0.2)
            plt.title('k={} '.format(k) + ' SSE=' + str(curr_sse))
        plt.show()
        print('iteration#{} | prev_sse= {:.4f};  curr_sse= {:.4f}'.format(i, prev_sse, curr_sse))

        # Terminal Condition
        if (prev_sse - curr_sse) / curr_sse < tol:
            break

    return cluster_memberships, curr_sse
```

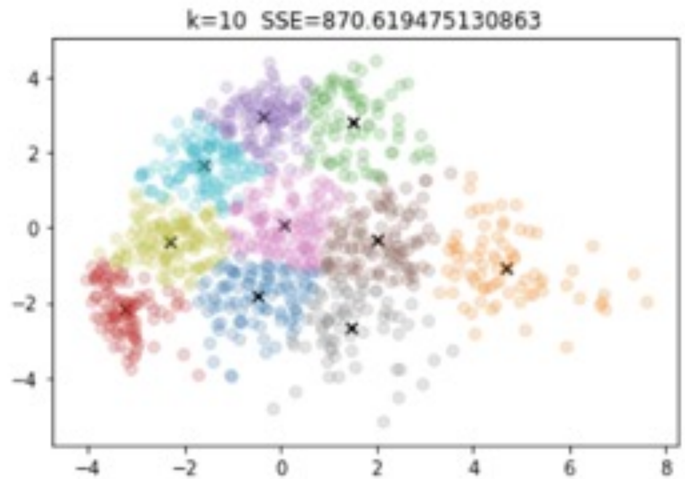
k-Means scratch

- **STEP5** : Iterate STEP3 and STEP4 until SSE is less than ‘tol’ value

```
cluster_memberships, curr_sse = kmeans(sub_PCA_train_x, k)
```




```
iteration#0 | prev_sse= inf;  curr_sse= 1259.7009
```




```
iteration#21 | prev_sse= 871.3678;  curr_sse= 870.6195
```

k-Means with sklearn

- **STEP6** : Implement k-means with sklearn library and check crosstab

cross tabulation

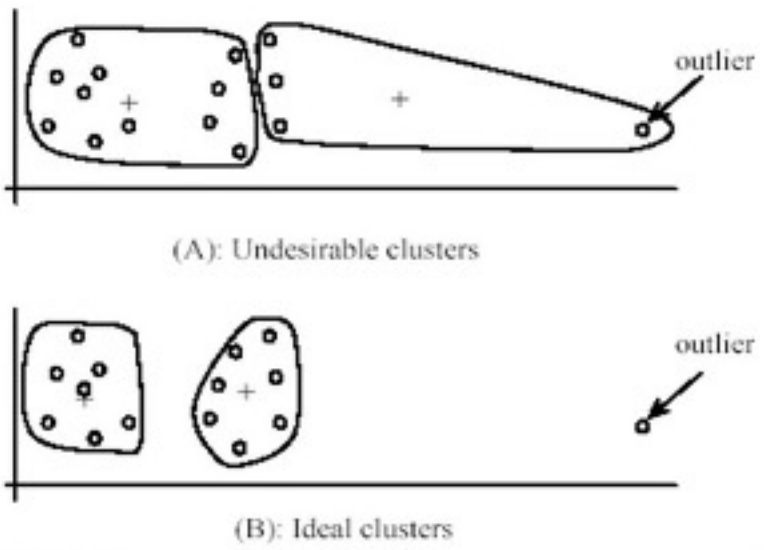
```python
from sklearn.cluster import KMeans

model = KMeans(n_clusters=k)
model.fit(PCA_train_x)

result = model.predict(PCA_test_x)
```

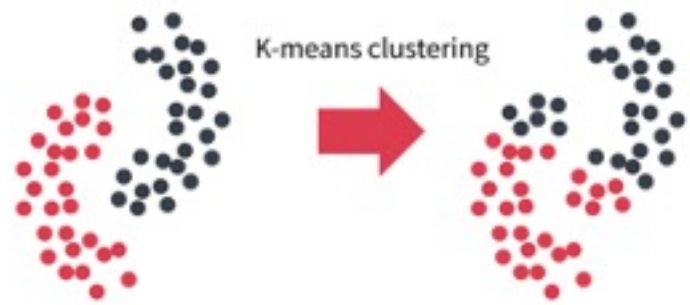
```python
import pandas as pd

df = pd.DataFrame({'labels': test_y['label'], 'result': result})
ct = pd.crosstab(df['labels'], df['result'])
ct
```

result	0	1	2	3	4	5	6	7	8	9
labels										
0	71	4	81	0	1	435	376	5	2	5
1	0	918	2	0	0	0	0	0	214	1
2	24	89	232	2	55	209	8	12	219	182
3	3	129	98	5	21	59	0	1	294	400
4	47	51	15	50	355	1	2	461	0	0
5	17	144	291	0	106	168	3	17	61	85
6	230	73	313	0	49	157	37	51	38	10
7	11	95	20	235	337	0	1	323	5	1
8	21	169	286	1	92	98	22	23	208	54
9	32	59	10	104	340	5	9	449	1	0

K-means disadvantages

- converges always (global convergence), but only a local optimum.
- sensitive to initialization
- sensitive to outliers
- not applicable when metric is not defined.
  - should be carefully chosen.
- not suitable for non-convex clusters.





https://www.quora.com/What-are-the-weaknesses-of-the-standard-k-means-algorithm-aka-Lloyds-algorithm

HW – Answer to given 6 questions

Q1: In the normalization methods, what is the meaning of offset and divisor, respectively?

Q2: After normalization, how does the data range change?

Q3: Before the iterations, how are the centroids defined?

Q4: One metric to evaluate the clustering results is sum of squared error (SSE). Describe the meaning of SSE in terms of the relationship between data and centroids.

Q5: What is the terminal condition? Describe it with tol and max_iter.