Cluster analysis (1)

Dr. Huiping Cao

Clustering Algorithms

- K-means and its variants
- Hierarchical clustering
- Density-based clustering

K-means clustering

- It is one type of **partition based** or **prototype based** clustering.
- There are other types of clustering including **hierarchical** and **density-based clustering**.
- In real-world applications of clustering, we do not have any ground truth information about the instances. Our goal is to group the instances based on their feature **similarity**. The similarity is generally measured as the opposite of **distance**.

K-means clustering

- Typically **squared Euclidean distance** is used. For example, the distance between two points $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ is defined as follows:

$$d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \| \mathbf{x}^{(i)} - \mathbf{x}^{(j)} \|_2^2 = \sum_{l=1}^{m} \left(\mathbf{x}_l^{(i)} - \mathbf{x}_l^{(j)}\right)^2$$

- **Partitioning method criterion**: Construct a partition of a database *D* of *n* objects into a set of *k* clusters, s.t., minimum sum of squared error, which is also called **within-cluster variation**.

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} \left(p - \mu^{(i)}\right)^2$$

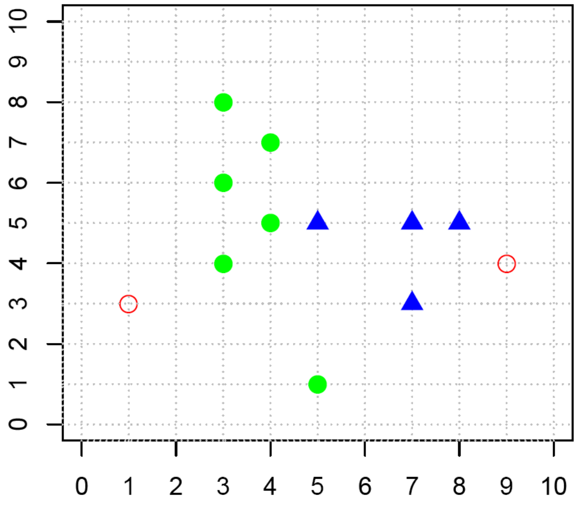
K-means clustering

- Given *k*, the *k*-means algorithm is implemented in steps.
  - Partitional clustering approach
  - Number of clusters, K, must be specified
  - Each cluster is associated with a centroid (center point)
  - Each point is assigned to the cluster with the closest centroid
  - The basic algorithm is very simple

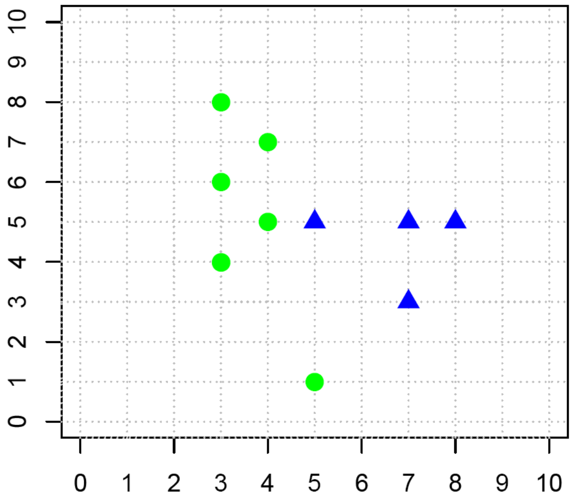
1: Select K points as the initial centroids.
2: **repeat**
3:     Form K clusters by assigning all points to the closest centroid.
4:     Recompute the centroid of each cluster.
5: **until** The centroids don't change

K-means algorithm - example

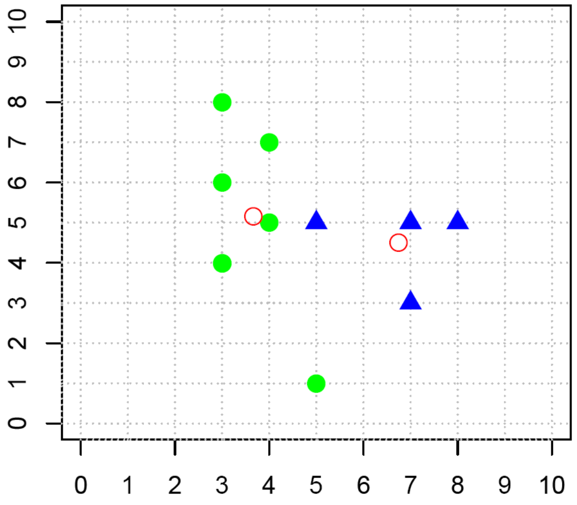
- k=2



S1: Arbitrarily choose *k* points as initial cluster center

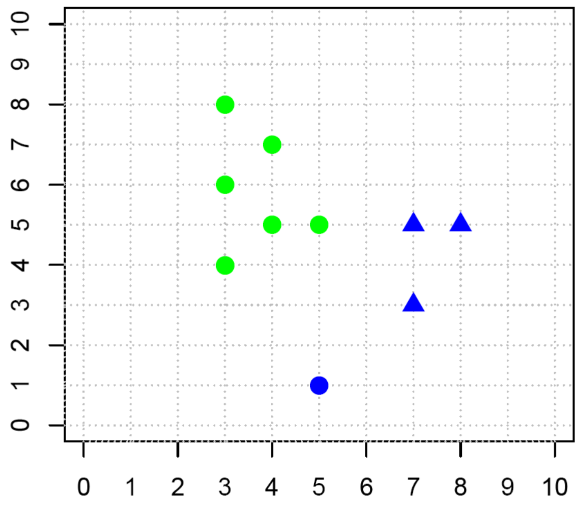


S2: Assign each object to the most similar centroid

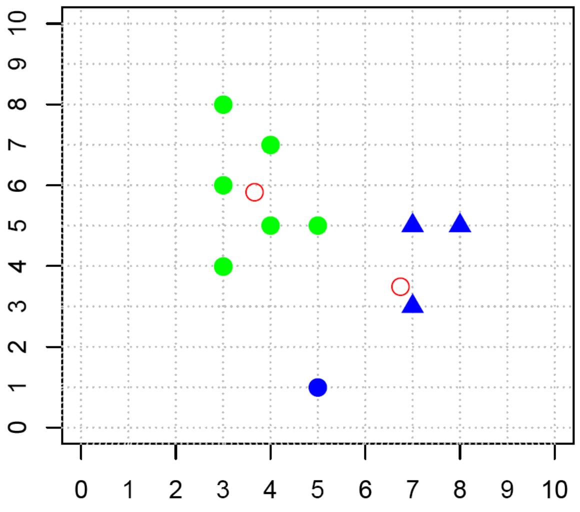


S3: Update the cluster means

K-means algorithm - example



S4: Re-assign points

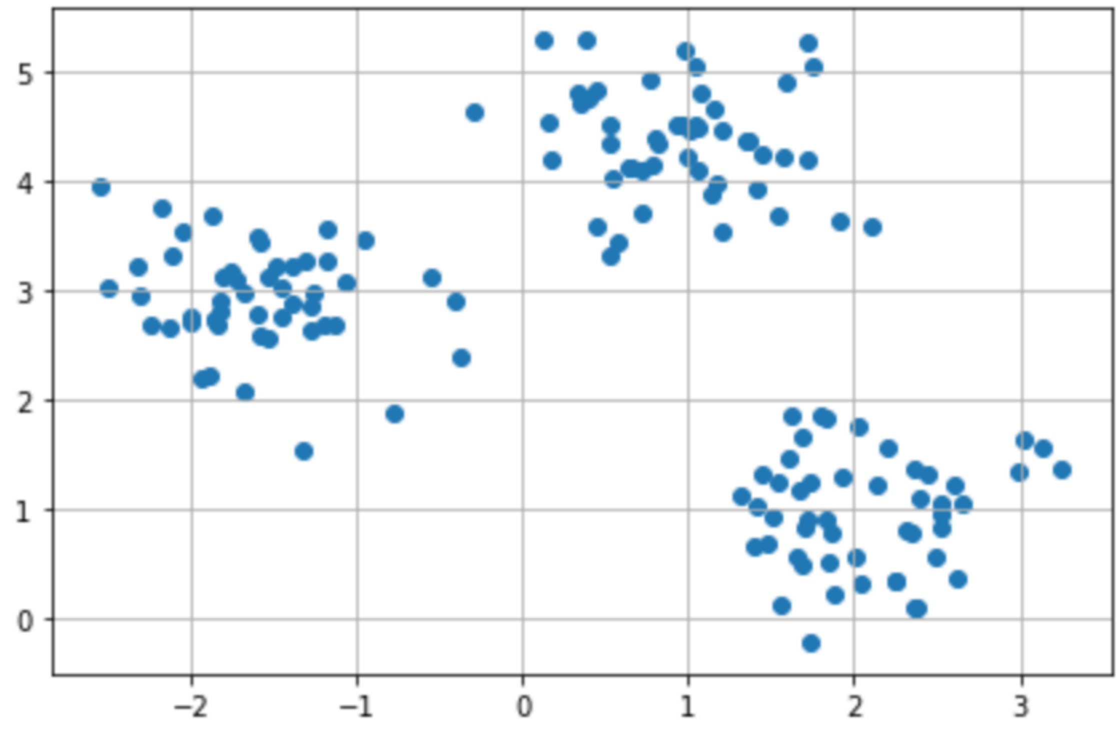


S5: Update the cluster means

S6: Re-assign points …

K-means code – generating synthetic data

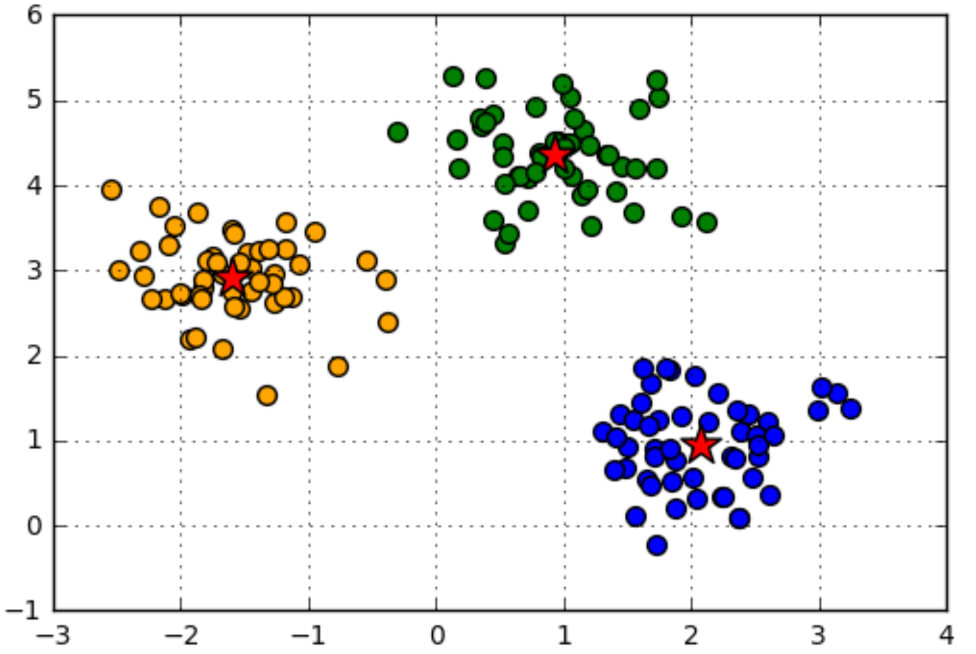
```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

#Generate synthetic data
X, y = make_blobs(n_samples=150, n_features=2,
                centers=3, cluster_std=0.5,
                shuffle=True,random_state =0)

#Plot X
plt.scatter(X[:, 0], X[:, 1])
plt.grid()
plt.tight_layout()
plt.show()
```



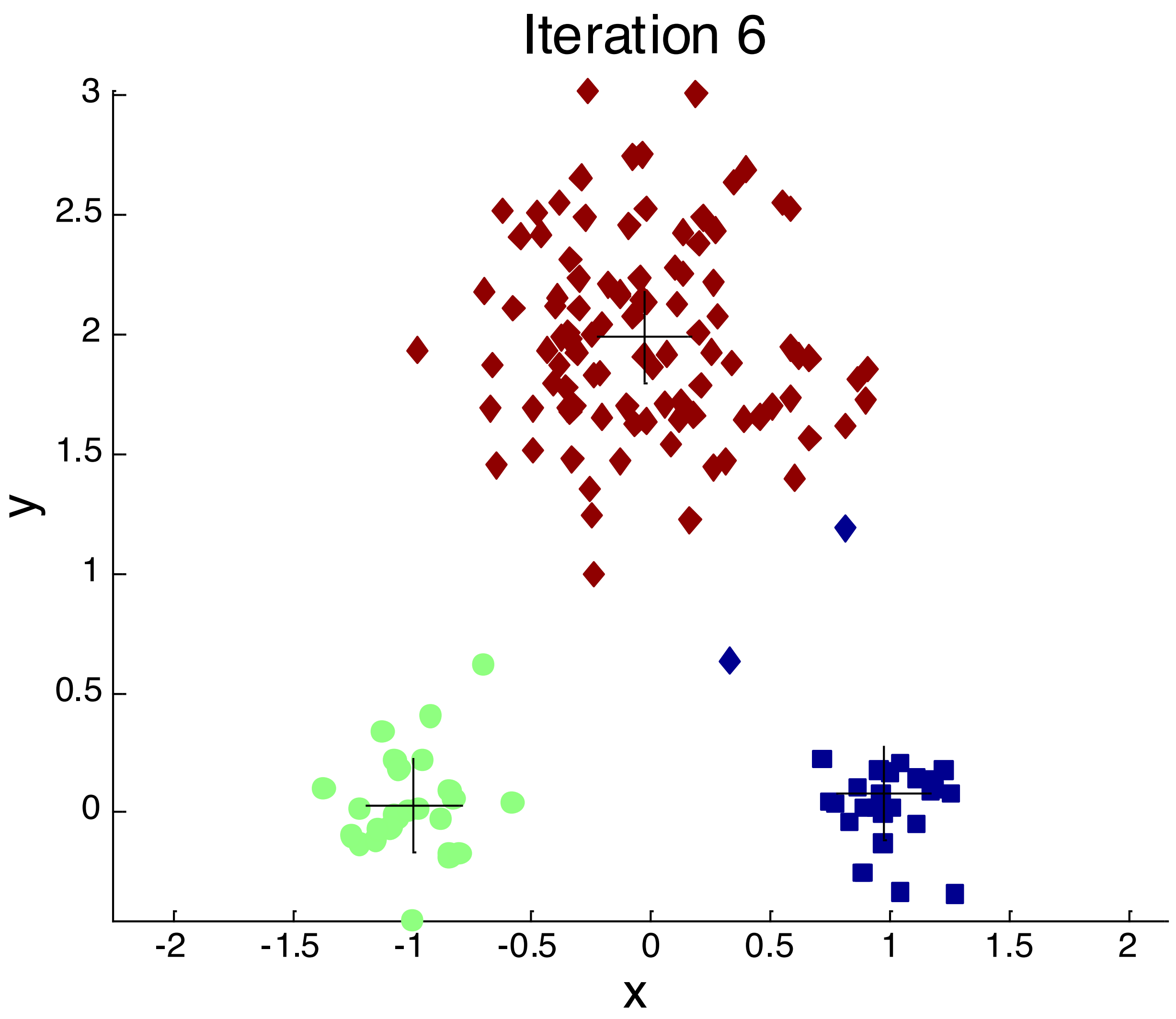
K-means code

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3, init='random',
          n_init =10, max_iter=300,
          tol=1e-04, random_state=0)

y_km = km.fit_predict(X)
print(y_km)
print(km.cluster_centers_ )
print('SE = %.3f' %km.inertia_)

# Plot the points in three clusters and the centroids
# Code details see textbook)
```

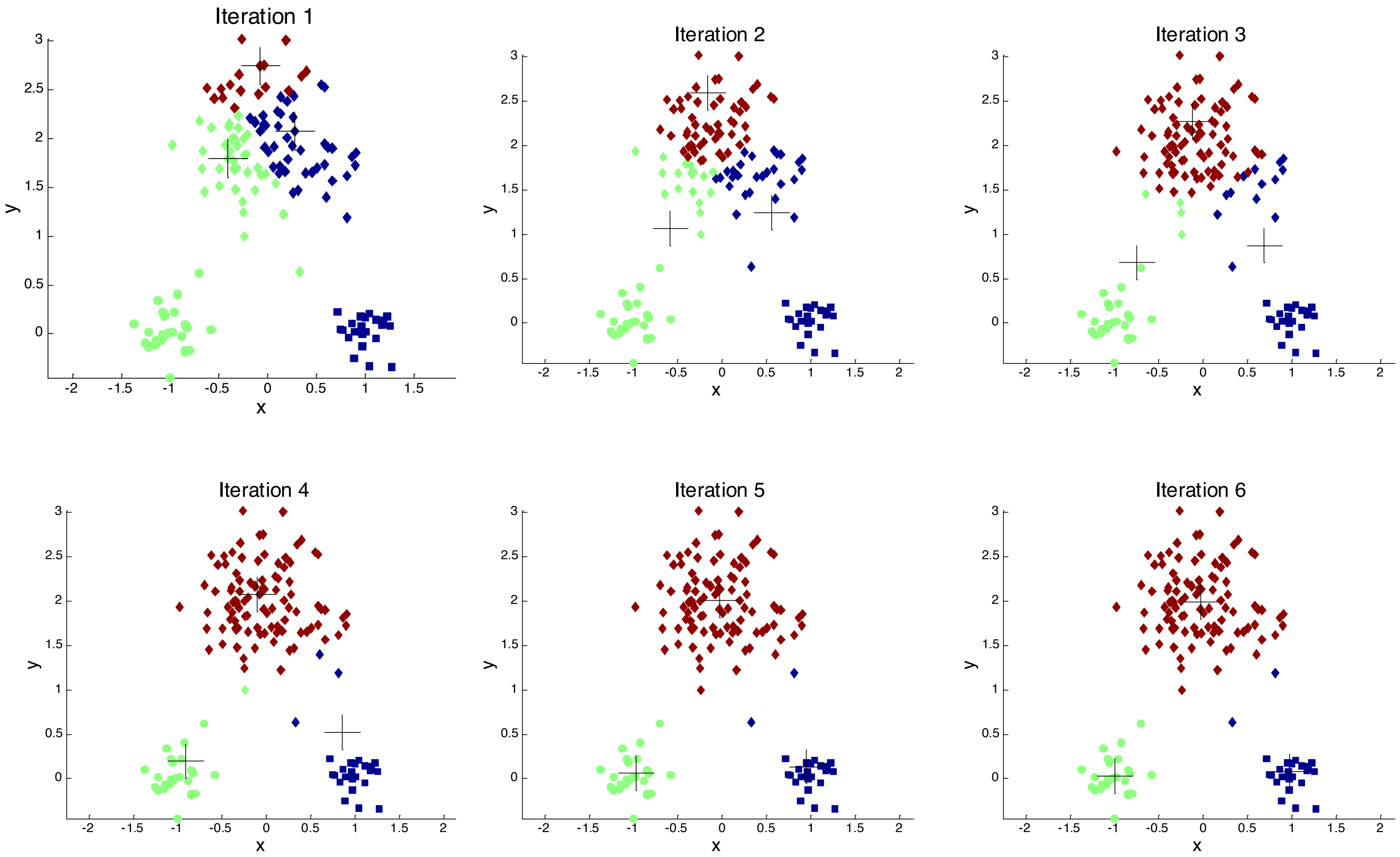
```
[2 1 1 1 2 1 1 2 0 1 2 0 0 1 1 0 0 2 0 2 1 2 1 1 0 2 2 1 0 2
0 0 0 0 1 2 2 2 1 1 0 0 1 2 2 2 0 1 0 1 2 1 1 2 2 0 1 2 0 1 0
0 0 0 1 0 1 2 1 1 1 2 2 1 2 1 1 0 0 1 2 2 1 1 2 2 2 0 0 2 2 1
2 1 2 1 0 0 2 2 2 2 0 2 2 1 0 1 1 1 0 1 2 0 1 0 1 1 0 0 1 2 1
1 2 2 0 2 0 0 0 0 2 0 0 0 1 0 2 0 1 1 2 2 0 0 0 0 2 2]
[[-1.5947298 2.92236966]
 [ 0.9329651 4.35420712]
 [ 2.06521743 0.96137409]]
SE = 72.476
```



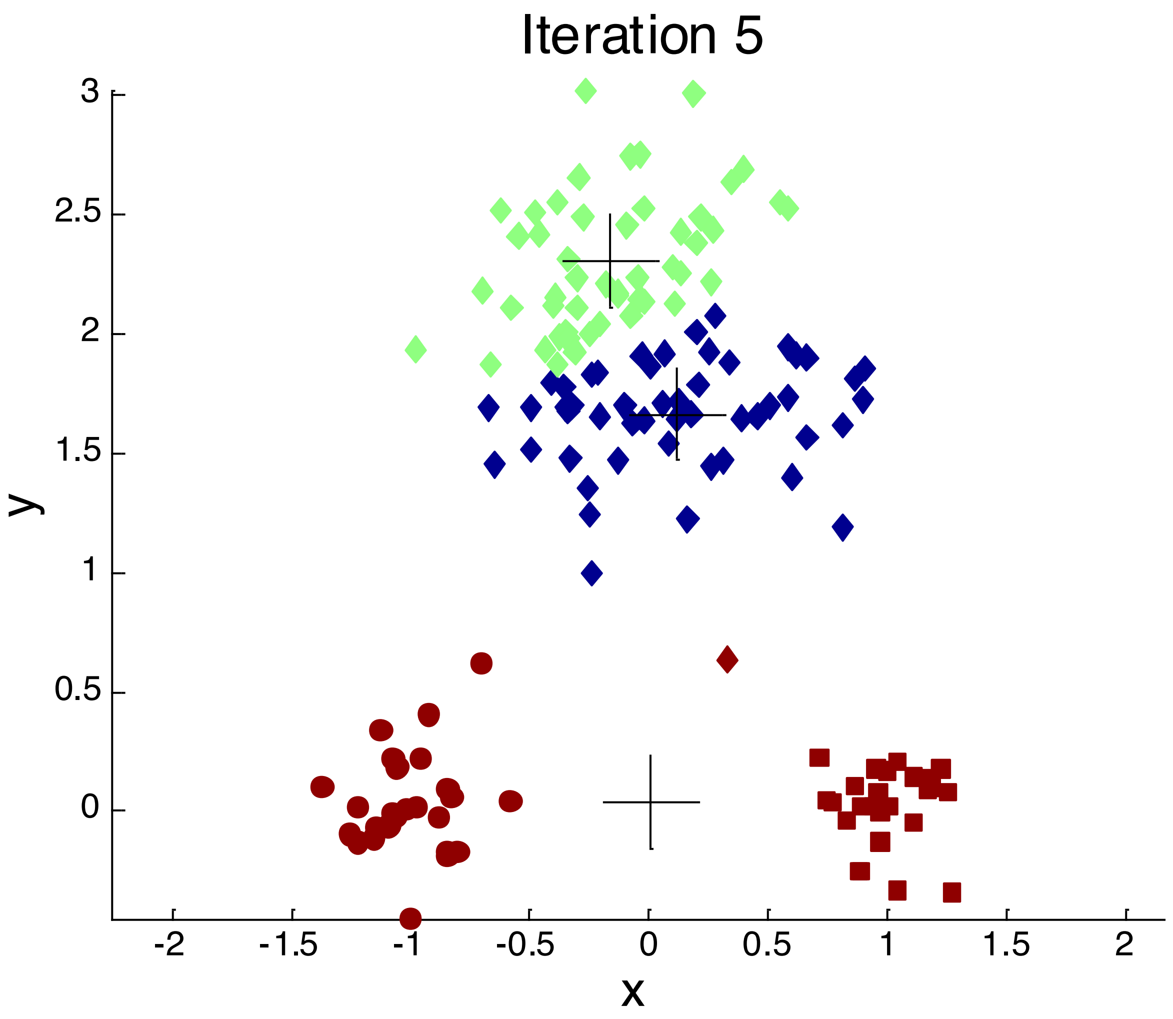
Importance of Choosing Initial Centroids



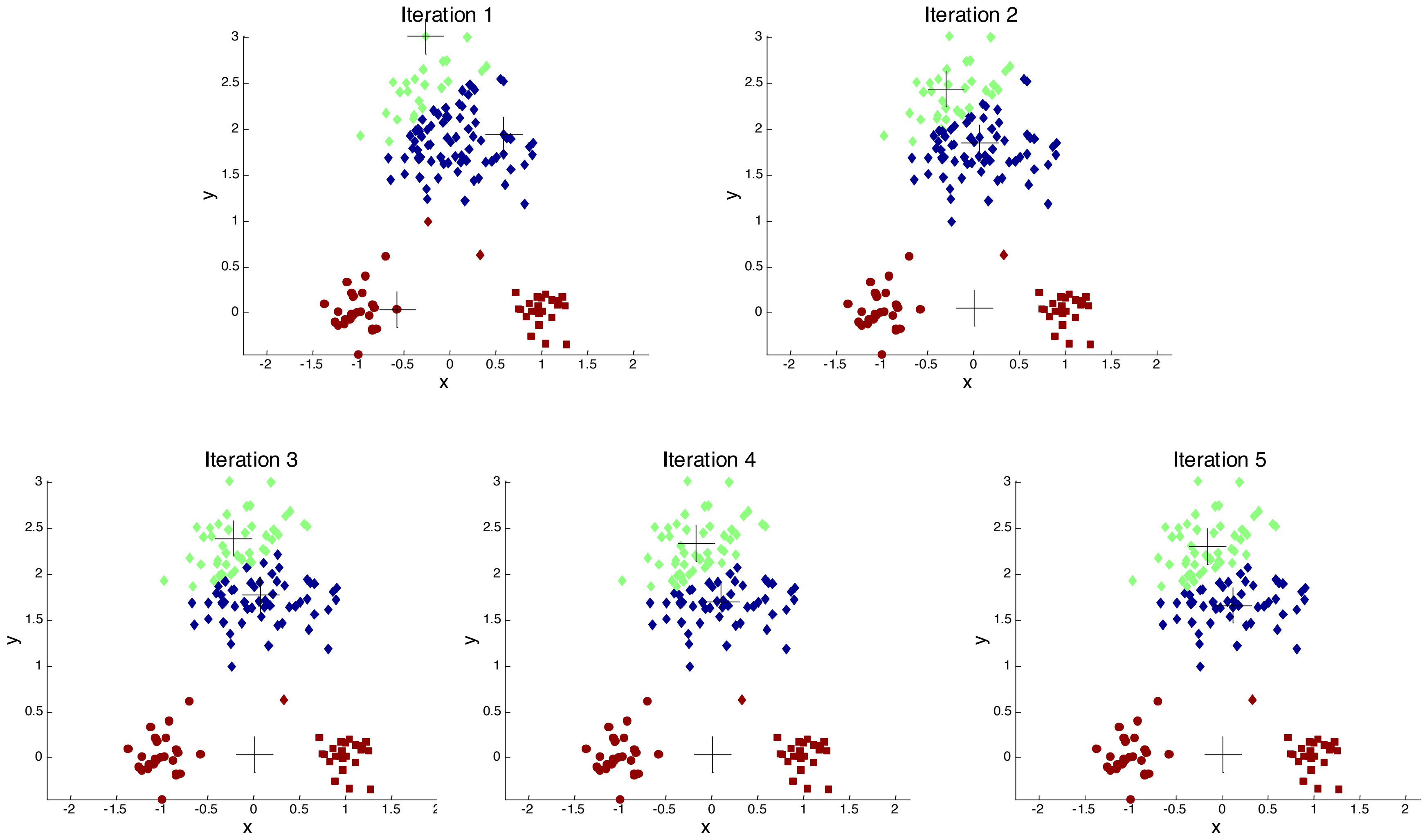
Importance of Choosing Initial Centroids



Importance of Choosing Initial Centroids



Importance of Choosing Initial Centroids



K-means Clustering – Details

- Initial centroids are often chosen randomly.
  - Clusters produced vary from one run to another.
- The centroid is (typically) the mean of the points in the cluster.
- ‘**Closeness**’ is measured by Euclidean distance, cosine similarity, correlation, etc.
- K-means will converge for common similarity measures mentioned above.
- Most of the **convergence** happens in the first few iterations.
  - Often the **stopping condition** is changed to ‘Until relatively few points change clusters’
- **Complexity** is O(n*K*I*d)
  - n = number of points, K = number of clusters,
    I = number of iterations, d = number of attributes

Solutions to Initial Centroids Problem

- **Multiple runs**
  - Helps, but probability is not on your side
- **Sample and use hierarchical clustering** to determine initial centroids
- Select more than **k initial centroids** and then select among these initial centroids
  - Select most widely separated
- **Postprocessing**
- **Generate a larger number of clusters** and then perform a hierarchical clustering
- **Bisecting K-means**
  - Not as susceptible to initialization issues

k-means++

- Difference from k-means: place initial cluster centroids in a smarter way.
- This approach can be slower than random initialization, but very consistently produces better results in terms of SSE
  - The k-means++ algorithm guarantees an approximation ratio $O(\log k)$ in expectation, where k is the number of centers

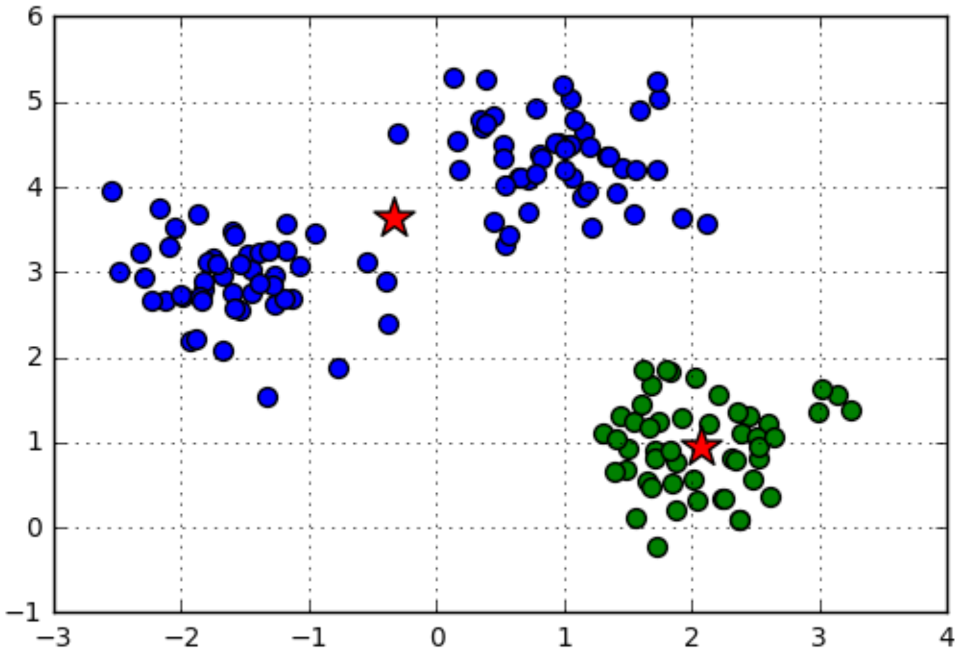
K-means++

1. Select an initial point at random to be the first centroid
2. For k – 1 steps
   1) For each of the N points, x_i, $1 \le i \le N$, find the minimum squared distance to the currently selected centroids, $C_1, \ldots, C_j$, $1 \le j < k$, i.e., $\min_j d^2(C_j, x_i)$
   2) Randomly select a new centroid by choosing a point with probability proportional to $\frac{\min_j d^2(C_j, x_i)}{\sum_i \min_j d^2(C_j, x_i)}$ is
3. End For

```
km = KMeans(n_clusters=2, init='k−means++', n_init =10, max_iter=300, tol=1e−04, random_state=0)
```

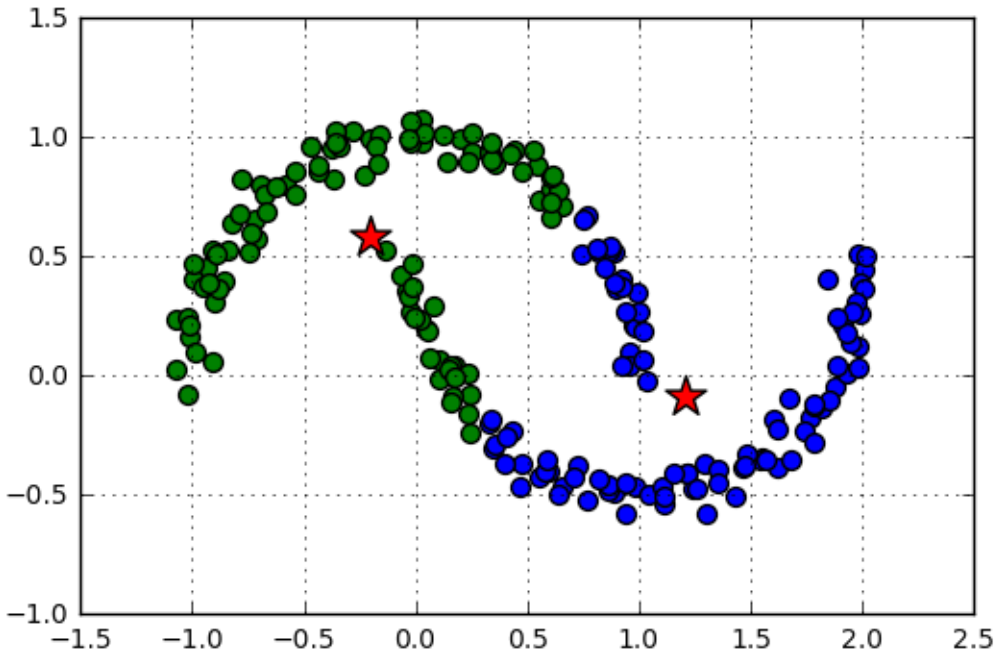
Comments on K-means algorithms

- Weakness 1. Need to specify *k* in advance
  - **ONE example for** k = 2



Comments on K-means algorithms

- Weakness 2. Not suitable to discover clusters with **non-convex shapes** (one example for moon-shaped dataset)

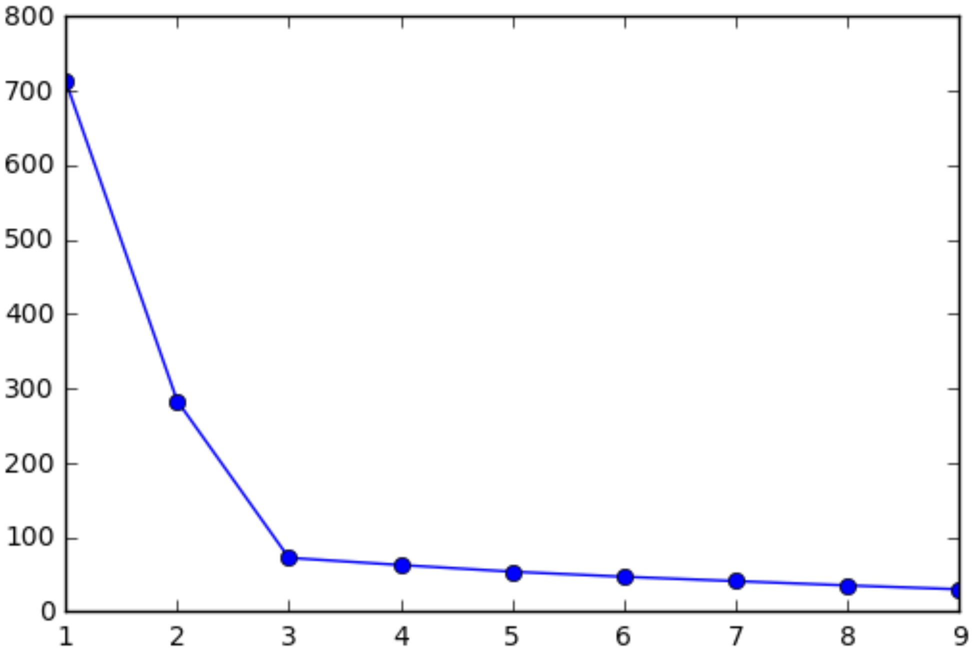


Comments on K-means algorithms

- Weakness 3. Unable to handle (sensitive to) noisy data and outliers: An object with an extremely large value may substantially distort the distribution of the data.
  - Example: given seven points in 1D space: 1,2,3,8,9,10,25 and *k* = 2
  - Intuitively, the clusters can be {1,2,3},{8,9,10,25}
  - $SSE = (1-2)^2 + (2-2)^2 + (3-2)^2 + (8-13)^2 + (9-13)^2 + (10-13)^2 + (25-13)^2 = 196$
  - Clusters gotten from the K-means algorithm: $\{1,2,3,8\}, \{9,10,25\}$
  - $SSE = (1-3.5)^2 + (2-3.5)^2 + (3-3.5)^2 + (8-3.5)^2 + (9-14.67)^2 + (10-14.67)^2 + (25-14.67)^2$ = 189.67

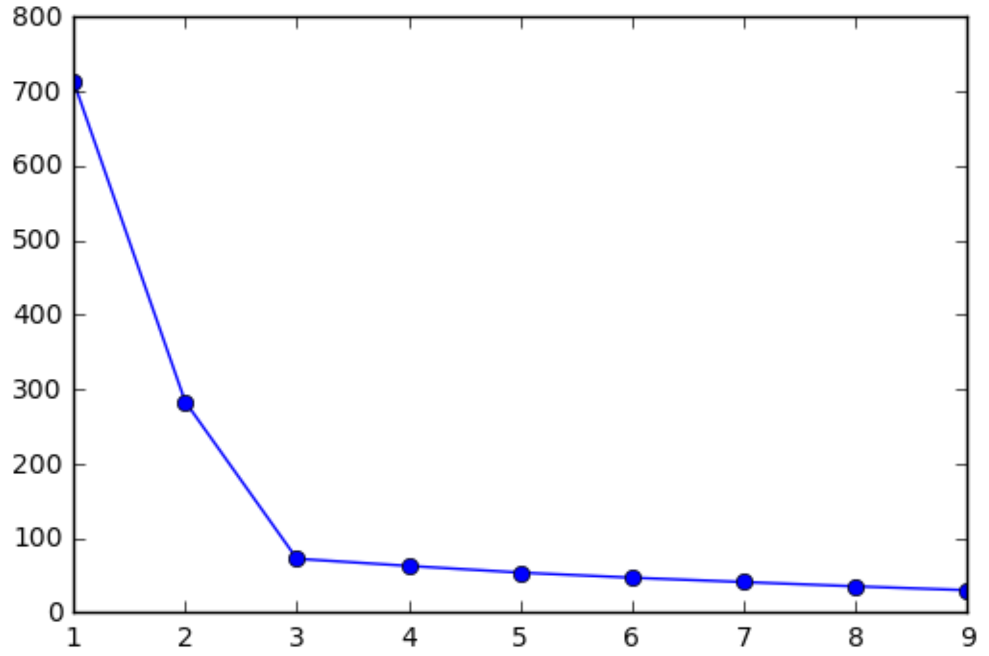
Elbow method: find *k*

- **Elbow method** is a graphical tool.
- Generally, if *k* increases, the distortion will decrease. Elbow method is to identify the value of *k* where the **distortion *begins to increase* most rapidly**.



Elbow method

```
distortions = []
# Calculate distortions
for i in range(1, 11):
    km = KMeans(n_clusters=i,
                init='k-means++',
                n_init=10,
                max_iter=300,
                random_state=0)
    km.fit(X)
    distortions.append(km.inertia_)

#Plot distortions for different K
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()
plt.show()
```



Reference

- Chapter 11, Sebastian Raschka and Vahid Mirjalili: Python Machine Learning (Machine learning and deep learning with Python, scikit-learn, and TensorFlow), 3rd Edition.
- Chapter 7, Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar: Introduction to Data Mining, 2nd Edition.