K-means

Unsupervised classification

Abdelwahid Benslimane
wahid.benslimane@gmail.com

Index

Abdelwahid Benslimane
wahid.benlimane@gmail.com

Generalities (1/2)

Purpose of unsupervised automatic classification :

- To group data without a priori information about their class, based on their similarity (partitioning).
- Eventually, a representative data item ("prototype") can be found for each group.
- The number of searched groups may or may not be known

Example of two-dimensional data partitioning into 3 groups with K-means:

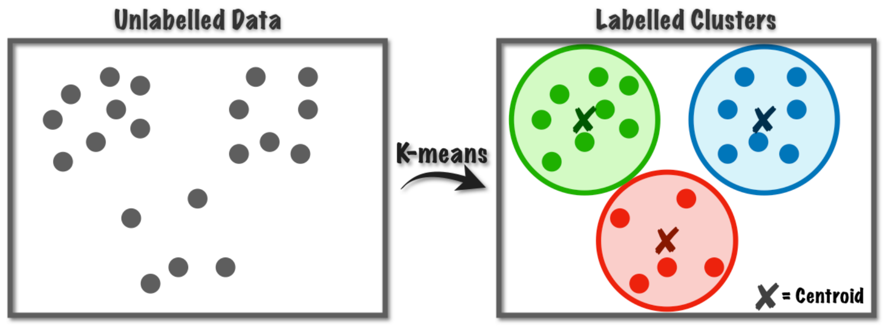


Image from https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c

Abdelwahid Benslimane
wahid.benlimane@gmail.com

Generalities (2/2)

Unsupervised automatic classification methods generally work on vector representations, which can be used to define the centers of gravity, probability densities and so on. Their complexity is generally $O(n)$.

Other, more general, and also more complex methods ($> O(n^2)$) are satisfied with a simple metric structure on the data space (to be able to calculate a distance).

Clusters can be mutually exclusive or not

crisp clustering: a data item may or may not belong to a cluster

fuzzy clustering: a data item may or may not belong to several clusters (FCM: Fuzzy C-means algorithm)

Some methods look for compact, relatively distant groups, as in the example on the previous slide, while others focus on dense, not necessarily compact, groups separated by less dense regions, as in the example below.

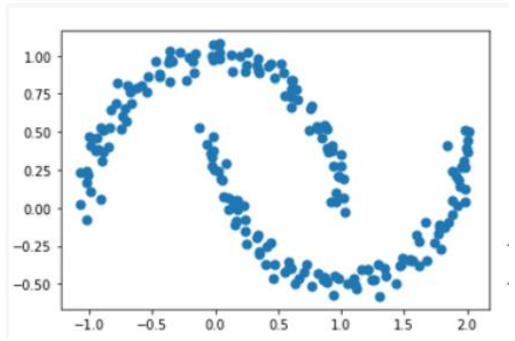


Image taken from https://www.niser.ac.in/~smishra/teach/cs460/2020/lectures/lec22/

Abdelwahid Benslimane
wahid.benlimane@gmail.com

K-means (1/4)

ϵ: set of N (vector) data described by p variables with values in R

To gather these data into k a priori unknown groups ϵ_1, ϵ_2, …, we minimize a certain cost function (sum of the squared distances of the data to the center of their class).

C : set of gravity centers m_j for the k groups ($j = 1, 2, \dots, k$)

$C(l)$: index of the group to which the data x_l belongs

$$\Phi_\epsilon(C) = \sum_{j=1}^{k} \sum_{x_i \in \epsilon_j} d^2(x_i, m_j) = \sum_{l=1}^{N} d^2(x_l, m_{C(l)})$$

The lower the value of $\Phi_\epsilon(C)$, the more compact the groups

Abdelwahid Benslimane
wahid.benlimane@gmail.com

K-means (2/4)

K-means algorithm:

```
1.  Random initialization of the group centers selected among the data
2.  While the centers are not stabilized, do:
        1. Assign each data item to the group of which the center is the nearest
        2. Replace the centers with centers of gravity of the new groups (they become the new centers)
    End while
```

As the number of classes increases, $\Phi_{\epsilon}(C)$ decreases (demonstrable, but also intuitive!), because by increasing the number of centers, for a certain number of data the squared distance to the nearest center will decrease. And if the number of classes corresponds to the number of observations, the value of $\Phi_{\epsilon}(C)$ will be zero (obviously not a realistic or relevant case).

Abdelwahid Benslimane
wahid.benlimane@gmail.com

K-means (3/4)

We can show that $\Phi_\epsilon(C)$ decreases at each of the two stages of the iterative process.

1. Assignment of each data item to the group associated with the nearest center: if x_i switches from group associated with the center m_p to group associated with the center m_q, it means that:

$$d^2(x_i, m_p) > d^2(x_i, m_q).$$

Thus:

$$d^2(x_i, m_p) + \sum_{l \neq i} d^2(x_l, m_{C(l)}) > d^2(x_i, m_q) + \sum_{l \neq i} d^2(x_l, m_{C(l)})$$

$\Phi_\epsilon(C)$ before the new affectation $\Phi_\epsilon(C)$ after the new affectation

Abdelwahid Benslimane
wahid.benlimane@gmail.com

K-means (4/4)

2. Replacement of old centers by the new centers of gravity of the groups: if $\widetilde{m_j}$ (consider cluster j) is the old center of the group and m_j the new center, then :

$$\sum_{x_i \in \epsilon_j} d^2(x_i, \widetilde{m_j}) = \sum_{x_i \in \epsilon_j} ||x_i - m_j + m_j - \widetilde{m_j}||^2$$

$$= \sum_{x_i \in \epsilon_j} ||x_i - m_j||^2 + \sum_{|x_i|} ||\, m_j - \widetilde{m_j}||^2 + 2\left(m_j - \widetilde{m_j}\right)^T \sum_{x_i \in \epsilon_j} (x_i - \ m_j) \qquad \left(\sum_{x_i \in \epsilon_j} (x_i - \ m_j) = 0\right)$$

$$\geq \sum_{x_i \in \epsilon_j} ||\, x_i - m_j||^2 \left(= \sum_{x_i \in \epsilon_j} d^2(x_i, m_j)\right)$$

Contribution to $\Phi_\epsilon(C)$ of group j decreases, so $\Phi_\epsilon(C)$ decreases.

Abdelwahid Benslimane
wahid.benlimane@gmail.com

Initialiation with K-means++ (1/1)

Good K-means initialization should result in a higher-quality solution with faster convergence (fewer iterations).

Input: set ϵ of N data in R^d and number k of centers/clusters to be searched for

Output: set C of k initial centers (chosen among the data) to be used when starting the K-means algorithm

```
1. $C \leftarrow x$ random selection of a data item in ϵ
2. While $|C| < k$, do:
        1. Select another $x \in \epsilon$ with the highest (probability) value $\frac{d^2(x,C)}{\phi_\epsilon(C)}$
        2. $C \leftarrow x$
   End of While
```

$$d^2(x,C) = min_{j=1,\dots,t}\, d^2\left(x, C_j\right), 1 \le t \le k$$

$$\phi_\epsilon(C) = \sum_{x \in \epsilon} d^2(x, C)$$

The idea behind K-means++ initialization is quite simple: we choose centers successively, following a non-uniform distribution that evolves after each center has been chosen, and which favors candidates who are far from centers that have already been selected

Abdelwahid Benslimane
wahid.benlimane@gmail.com

Limitations (1/1)

- An obvious limitation is the need for vector data in order to calculate group (gravity) centers, a limitation lifted in the k-medoids method, which we haven't seen here, and which is satisfied with the existence of a metric.

- The K-means method is not suited to the discovery of dense, non-compact groups separated by less dense regions, and is limited to classes of globally spherical shape. This is due to the use of the usual Euclidean distance. This limitation can be overcome by using other metrics (such as the Mahalanobis metric).

$$d(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

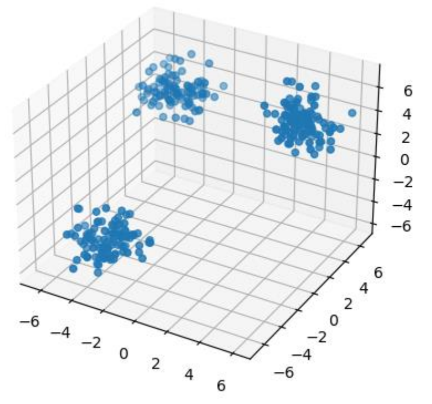
S^{-1}: Inverse of the variance-covariance matrix of the data. If it is the identity matrix, then Mahalanobis distance = Euclidean distance.

Abdelwahid Benslimane
wahid.benlimane@gmail.com

Demo (1/3)

```
import numpy as np
from sklearn.utils import shuffle

# generation of 3 x 100 3D points following a normal law
# + translation of each group of points
d1 = np.random.randn(100,3) + [4,4,4]
d2 = np.random.randn(100,3) + [-4,4,4]
d3 = np.random.randn(100,3) + [-4,-4,-4]

# generation of labels (they won't be used by k-means)
l1 = np.ones(100)
l2 = 2 * np.ones(100)
l3 = 3 * np.ones(100)

# data are concatenated in a matrix (same for labels)
data = np.concatenate((d1,d2,d3))
labels = np.concatenate((l1, l2, l3))

# random shuffling of the data matrix (same for labels)
data, labels = shuffle(data, labels)
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#visualization of the data points
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(data[:,0], data[:,1], data[:,2])
plt.show()
```

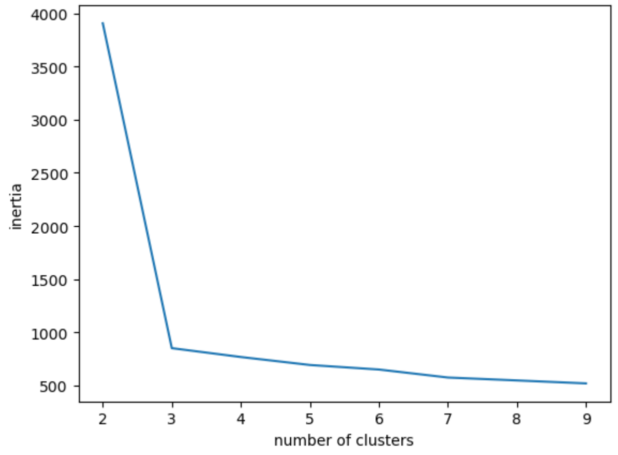


Abdelwahid Benslimane
wahid.benlimane@gmail.com

Demo (2/3)

```
from sklearn.cluster import KMeans

#k-means is applied with different numbers of centers/clusters
#the different values of the inertia (sum of squared distances
#of samples to their closest cluster center ) are registed in a table
#documentation: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

inertia = []


for i in range(2, 10):
    kmeans = KMeans(n_clusters = i, n_init=1, init = 'k-means++', random_state = 10)
    kmeans.fit(data)
    inertia.append(kmeans.inertia_)
```

```
import matplotlib.pyplot as plt

#evolution of the inertia depending on ne the number of clusters

fig, ax = plt.subplots()
ax.plot([2, 3, 4, 5, 6, 7, 8, 9], inertia)
plt.xlabel("number of clusters")
plt.ylabel("inertia")
plt.show()
```



Abdelwahid Benslimane
wahid.benlimane@gmail.com

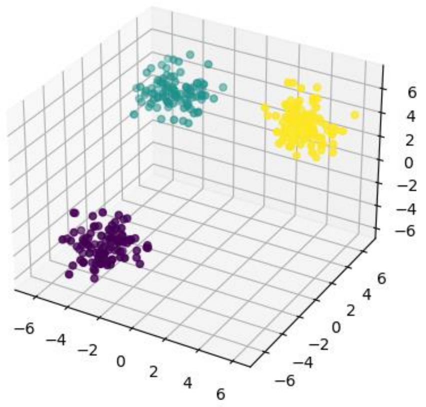
Demo (3/3)

```
#the optimal number of clusers is 3 according to the "elbow method"

kmeans = KMeans(n_clusters = 3, n_init=1, init = 'k-means++', random_state = 10)
kmeans.fit(data)
```

```
KMeans(n_clusters=3, n_init=1, random_state=10)
```

```
#prediction of the labels with he k-means

labels_pred = kmeans.predict(data)
```

```
#visualization of the labels prediction with k-means
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
#colors are based on labels
ax.scatter(data[:,0], data[:,1], data[:,2], c=labels_pred)
plt.show()
```



```
#quality of the prediction assessed with the adjusted Rand index
#https://en.wikipedia.org/wiki/Rand_index

from sklearn import metrics
metrics.adjusted_rand_score(labels_pred, labels)
```

```
1.0
```

Abdelwahid Benslimane
wahid.benlimane@gmail.com