Unsupervised Learning: K-means Clustering

by Prof. Seungchul Lee
iSystems Design Lab
http://isystems.unist.ac.kr/
UNIST

Table of Contents

- I. 1. Supervised vs. Unsupervised Learning
- II. 2. K-means
  - II. 2.1. (Iterative) Algorithm
  - II. 2.2. Summary: K-means Algorithm
  - III. 2.3. K-means Optimization Point of View (optional)
- III. 3. Python code
- IV. 4. Some Issues in K-means
  - II. 4.2. Choosing the Number of Clusters
  - III. 4.3. K-means: Limitations

1. Supervised vs. Unsupervised Learning

- Supervised: building a model from labeled data
- Unsupervised: clustering from unlabeled data

Supervised Learning

\[
\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\} \quad \Rightarrow \quad \text{Classification}
\]

\[
\{y^{(1)}, y^{(2)}, \ldots, y^{(m)}\}
\]
Unsupervised Learning

- Data clustering is an unsupervised learning problem
- Given:
  - \( m \) unlabeled examples \( \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\} \)
  - the number of partitions \( k \)
- Goal: group the examples into \( k \) partitions

\[
\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\} \Rightarrow \text{Clustering}
\]

- the only information clustering uses is the similarity between examples
- clustering groups examples based of their mutual similarities
- A good clustering is one that achieves:
  - high within-cluster similarity
  - low inter-cluster similarity
- it is a "chicken and egg" problem (dilemma)
  - Q: if we knew \( c_i \) s, how would we determine which points to associate with each cluster center?
  - A: for each point \( x^{(i)} \), choose closest \( c_i \)
  - Q: if we knew the cluster memberships, how do we get the centers?
  - A: choose \( c_i \) to be the mean of all points in the cluster

2. K-means
2.1. (Iterative) Algorithm

1) Initialization

Input:

- $k$: the number of clusters
- Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

Randomly initialized anywhere in $\mathbb{R}^n$.
2) Iteration

- Repeat until convergence (a possible convergence criteria: cluster centers do not change anymore)

3) Output

Output: model

- \( c \) (label): index (1 to \( k \)) of cluster centroid \( \{ c_1, c_2, \ldots, c_k \} \)
- \( \mu \): averages (mean) of points assigned to cluster \( \{ \mu_1, \mu_2, \ldots, \mu_k \} \)
2.2. Summary: K-means Algorithm

Randomly initialize \( k \) cluster centroids \( \mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n \)

Repeat{
  for \( i = 1 \) to \( m \)
    \( c_i := \) index (from 1 to \( k \)) of cluster centroid closest to \( x^{(i)} \)
  for \( k = 1 \) to \( k \)
    \( \mu_k := \) average (mean) of points assigned to cluster \( k \)
}
2.3. K-means Optimization Point of View (optional)

- $c_i =$ index of cluster $(1, 2, \cdots, k)$ to which example $x^{(i)}$ is currently assigned
- $\mu_k =$ cluster centroid $k \ (\mu_k \in \mathbb{R}^n)$
- $\mu_{c_i} =$ cluster centroid of cluster to which example $x^{(i)}$ has been assigned

- Optimization objective:

$$J(c_1, \cdots, c_m, \mu_1, \cdots, \mu_k) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c_i}||^2$$

$$\min_{c_1, \cdots, c_m, \mu_1, \cdots, \mu_k} J(c_1, \cdots, c_m, \mu_1, \cdots, \mu_k)$$

3. Python code

In [2]:

```python
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```
In [3]:

# data generation
G0 = np.random.multivariate_normal([1, 1], np.eye(2), 100)
G1 = np.random.multivariate_normal([3, 5], np.eye(2), 100)
G2 = np.random.multivariate_normal([9, 9], np.eye(2), 100)

X = np.vstack([G0, G1, G2])
X = np.asmatrix(X)
print(X.shape)

plt.figure(figsize=(10, 6))
plt.plot(X[:,0], X[:,1], 'b.'
plt.show()

(300, 2)
In [4]:

# The number of clusters and data
k = 3
m = X.shape[0]

# randomly initialize mean points
mu = X[np.random.randint(0,300,k),:]
pre_mu = mu.copy()
print(mu)

plt.figure(figsize=(10, 6))
plt.plot(X[:,0], X[:,1], 'b.' )
plt.plot(mu[:,0], mu[:,1], 'ko')
plt.show()

[[ 1.08723812  7.02505378]
 [ 9.37822573  8.07630763]
 [ 3.09510078  4.684617  ]]
In [5]:

```python
y = np.empty([m,1])

# Run K-means
for n_iter in range(500):
    for i in range(m):
        d0 = np.linalg.norm(X[i,:] - mu[0,:],2)
        d1 = np.linalg.norm(X[i,:] - mu[1,:],2)
        d2 = np.linalg.norm(X[i,:] - mu[2,:],2)
        y[i] = np.argmin([d0, d1, d2])
    err = 0
    for i in range(k):
        mu[i,:] = np.mean(X[np.where(y == i)[0]], axis=0)
        err += np.linalg.norm(pre_mu[i,:] - mu[i,:],2)
    pre_mu = mu.copy()
    if err < 1e-10:
        print("Iteration: ", n_iter)
        break
```

Iteration: 4

In [6]:

```python
X0 = X[np.where(y==0)[0]]
X1 = X[np.where(y==1)[0]]
X2 = X[np.where(y==2)[0]]

plt.figure(figsize=(10, 6))
plt.plot(X0[:,0], X0[:,1], 'b.'
plt.plot(X1[:,0], X1[:,1], 'g.'
plt.plot(X2[:,0], X2[:,1], 'r.'
plt.show()
```

![Plot of data points colored by K-means clusters.](image)
4. Some Issues in K-means

4.1. K-means: Initialization issues

- k-means is extremely sensitive to cluster center initialization
  
  - Bad initialization can lead to
    - Poor convergence speed
    - Bad overall clustering

- Safeguarding measures:
  - Choose first center as one of the examples, second which is the farthest from the first, third which is the farthest from both, and so on.
  - Try multiple initialization and choose the best result
4.2. Choosing the Number of Clusters

- Idea: when adding another cluster does not give much better modeling of the data
- One way to select $k$ for the K-means algorithm is to try different values of $k$, plot the K-means objective versus $k$, and look at the 'elbow-point' in the plot

In [8]:

```python
# data generation
G0 = np.random.multivariate_normal([1, 1], np.eye(2), 100)
G1 = np.random.multivariate_normal([3, 5], np.eye(2), 100)
G2 = np.random.multivariate_normal([9, 9], np.eye(2), 100)
X = np.vstack([G0, G1, G2])
X = np.asmatrix(X)
```

In [9]:

```python
cost = []
for i in range(1,11):
    kmeans = KMeans(n_clusters=i, random_state=0).fit(X)
    cost.append(abs(kmeans.score(X)))

plt.figure(figsize=(10,6))
plt.stem(range(1,11),cost)
plt.show()
```
4.3. K-means: Limitations

- Make hard assignments of points to clusters
  - A point either completely belongs to a cluster or not belongs at all
  - No notion of a soft assignment (i.e., probability of being assigned to each cluster)
  - Gaussian mixture model (we will study later) and Fuzzy K-means allow soft assignments

- Sensitive to outlier examples (such example can affect the mean by a lot)
  - K-medians algorithm is a more robust alternative for data with outliers

- Works well only for round shaped, and of roughly equal sizes/density cluster

- Does badly if the cluster have non-convex shapes
  - Spectral clustering (we will study later) and Kernelized K-means can be an alternative

- Non-convex/non-round-shaped cluster: standard K-means fails!

Clusters with different densities