1.
原理：

混合高斯密度估计是指对于一个由 k 个高斯成分组成的分布，利用给定数据估计各高斯成分的参数，并给出样本所属各高斯成分的先验概率。K-Means 算法可以看作混合高斯密度函数的“硬划分”版本，是后者的特殊情形，主要引入了如下假设：

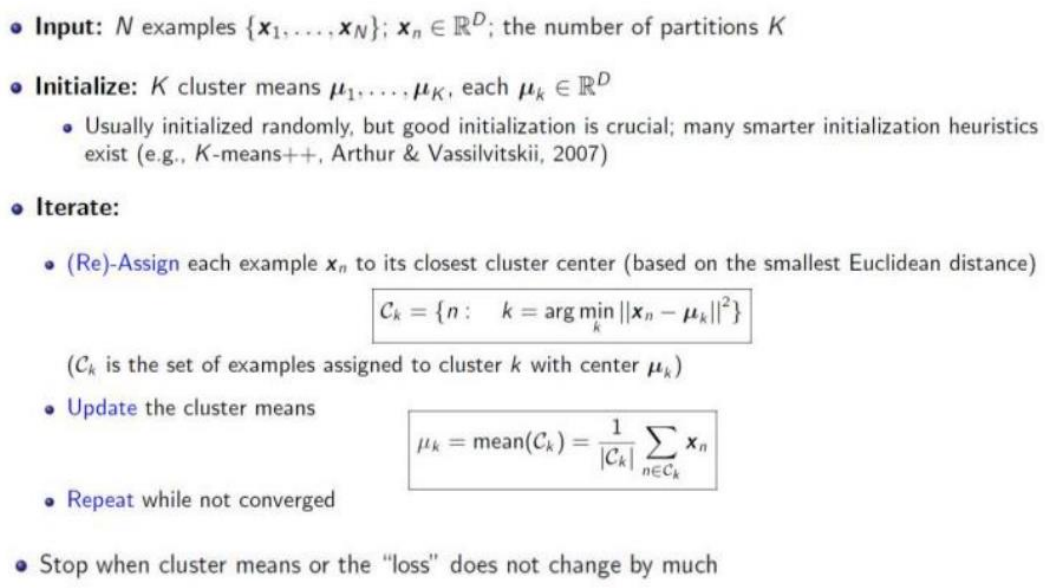
（1）各类别的先验概率相等；

（2）每一个样本以概率 1 属于某一个类别，而不属于其他类别（概率为 0）；

（3）每个高斯分量的协方差矩阵均为单位矩阵 I。

K-Means 算法是一种基于距离的无监督学习算法，通过迭代计算，每次迭代首先根据样本点与每一个聚类中心的距离，分配样本点给对应的聚类，随后计算更新每一个聚类中心。聚类中心可以看作高斯成分的均值。

计算步骤：



- **Input:** N examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$; $\mathbf{x}_n \in \mathbb{R}^D$; the number of partitions K
- **Initialize:** K cluster means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, each $\boldsymbol{\mu}_k \in \mathbb{R}^D$
  - Usually initialized randomly, but good initialization is crucial; many smarter initialization heuristics exist (e.g., K-means++, Arthur & Vassilvitskii, 2007)
- **Iterate:**
  - (Re)-Assign each example $\mathbf{x}_n$ to its closest cluster center (based on the smallest Euclidean distance)

$$\boxed{\mathcal{C}_k = \{n : \quad k = \arg\min_k ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2\}}$$

  ($\mathcal{C}_k$ is the set of examples assigned to cluster k with center $\boldsymbol{\mu}_k$)
  - Update the cluster means

$$\boxed{\boldsymbol{\mu}_k = \text{mean}(\mathcal{C}_k) = \frac{1}{|\mathcal{C}_k|}\sum_{n\in\mathcal{C}_k}\mathbf{x}_n}$$

  - Repeat while not converged
- Stop when cluster means or the “loss” does not change by much

影响因素：

聚类数目 K 的选择；初始聚类中心的选择；距离的计算方式；数据的分布（流形形状），K-Means 在均衡数据、类似球形分布时性能较好，在复杂流形或非凸分布性能不好。

2.
原理：

谱聚类算法建立在图论的谱图理论基础之上，其本质是将聚类问题转化为一个图上的关于顶点划分的最优问题。谱聚类算法建立在点对亲和性基础之上，理论上能对任意分布形状的样本空间进行聚类。

计算步骤：

Unnormalized Spectral Clustering

1. Construct a similarity graph and compute the unnormalized graph Laplacian L.
2. Compute the k smallest eigenvectors $u_1, u_2, \cdots, u_k$ of L.
3. Let $U = [\, u_1 \, u_2 \, \cdots \, u_k \,] \in \mathbb{R}^{n \times k}$.
4. Let $y_i \in \mathbb{R}^k$ be the vector corresponding to the ith row of U.

$$U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1k} \\ u_{21} & u_{22} & \cdots & u_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n1} & u_{n2} & \cdots & u_{nk} \end{bmatrix} = \begin{bmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_n^T \end{bmatrix}$$

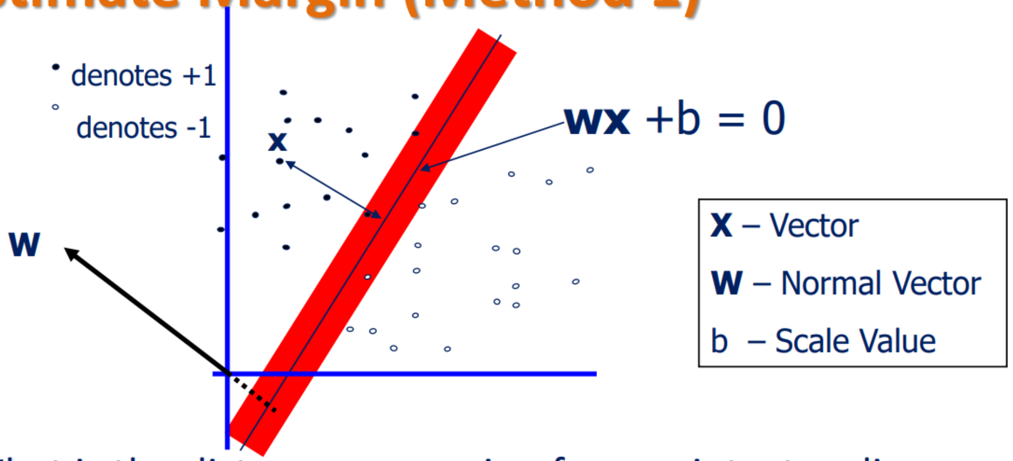
5. Thinking of y_i's as points in $\mathbb{R}^k$, cluster them with k-means algorithms.

影响因素：

相似度的计算方法：距离度量方式；图构造方式：全局或局部；局部相似度阈值ε或近邻个数k；图 Laplacian 矩阵是否归一化及归一化方式等。

3.

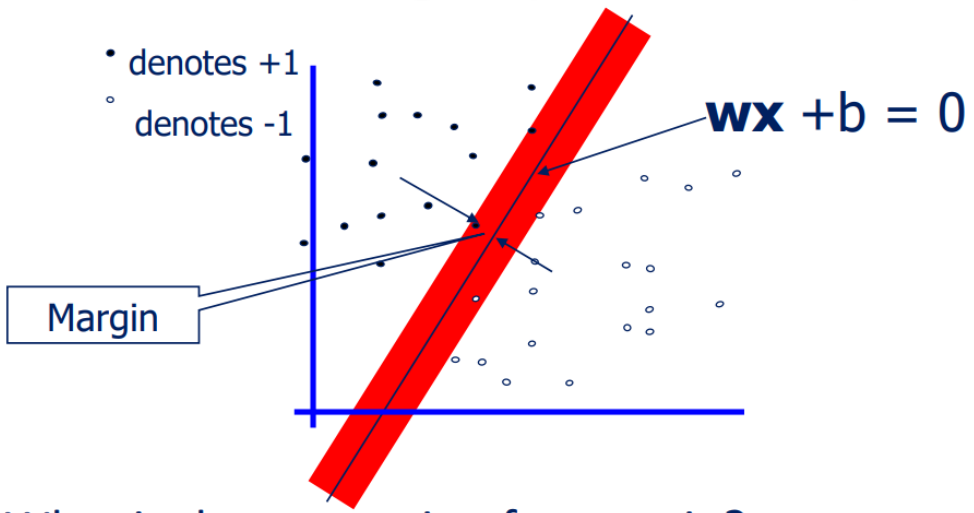
Estimate Margin (Method 1)



- What is the distance expression for a point **x** to a line **wx**+b= 0?

$$d(\mathbf{x}) = \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

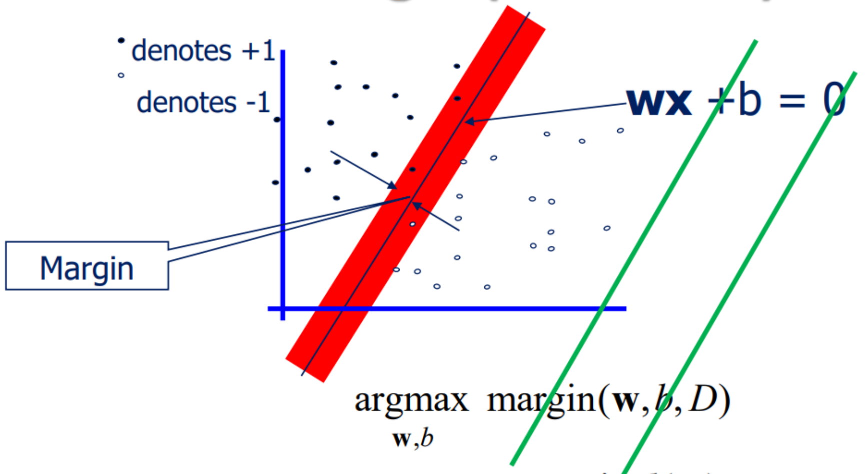
Estimate Margin (Method 1)



- What is the expression for margin?

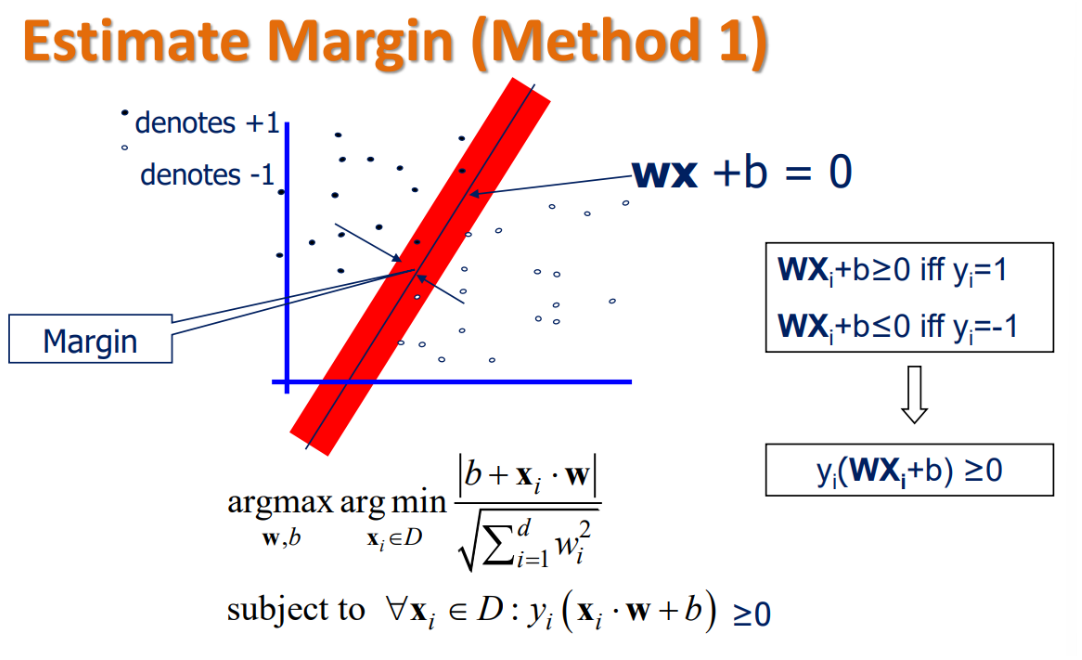
$$\text{margin} \equiv \arg\min_{\mathbf{x}\in D} d(\mathbf{x}) = \arg\min_{\mathbf{x}\in D} \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

Estimate Margin (Method 1)



Is this all?

$$\underset{\mathbf{w},b}{\text{argmax}}\ \text{margin}(\mathbf{w},b,D)$$
$$= \underset{\mathbf{w},b}{\text{argmax}} \arg\min_{\mathbf{x}_i\in D} d(\mathbf{x}_i)$$
$$= \underset{\mathbf{w},b}{\text{argmax}} \arg\min_{\mathbf{x}_i\in D} \frac{|b+\mathbf{x}_i\cdot\mathbf{w}|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$



$\mathbf{WX}_i+b\geq 0$ iff $y_i=1$

$\mathbf{WX}_i+b\leq 0$ iff $y_i=-1$

$$\Downarrow$$

$y_i(\mathbf{WX}_i+b) \geq 0$

$$\underset{\mathbf{w},b}{\operatorname{argmax}} \underset{\mathbf{x}_i\in D}{\arg\min} \frac{|b+\mathbf{x}_i\cdot\mathbf{w}|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

$$\text{subject to } \forall \mathbf{x}_i \in D: y_i\left(\mathbf{x}_i\cdot\mathbf{w}+b\right) \geq 0$$

4.

Hinge loss 公式为:

$$\sum_{i=1}^{N}[1-y_i(w\cdot x_i+b)]_+ + \lambda||w||^2$$

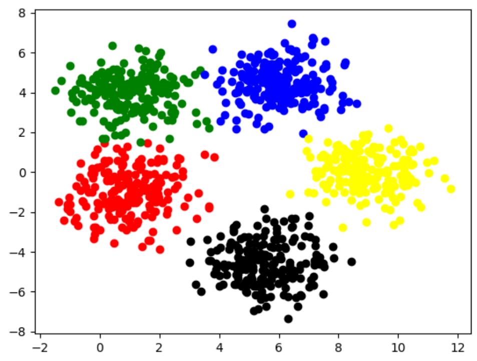
$$[z]_+ = \begin{cases} z, z>0 \\ 0.z\leq 0 \end{cases}$$

第一项是损失，第二项是正则化项。比感知机损失函数更为严格，因为当其为 0 时，说明不仅要分类正确，而且置信度要足够高。

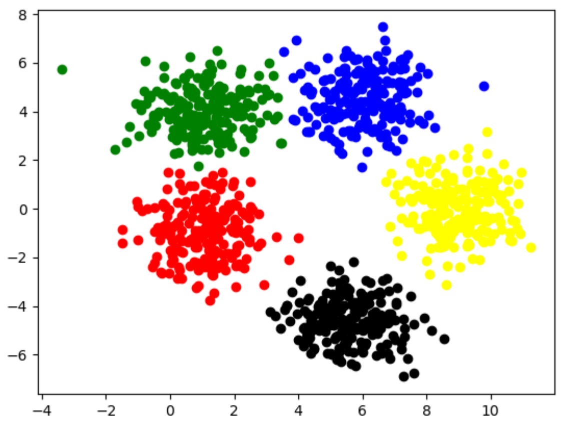
编程 1

```python
from sklearn.cluster import KMeans
import numpy as np
from numpy.linalg import cholesky
import random
import matplotlib.pyplot as plt

def get_data():
    sampleNo = 200

    mu1 = np.array([[1, -1]])
    mu2 = np.array([[5.5, -4.5]])
    mu3 = np.array([[1, 4]])
    mu4 = np.array([[6, 4.5]])
    mu5 = np.array([[9, 0]])

    Sigma = np.array([[1, 0], [0, 1]])
    R = cholesky(Sigma).T
    va,vc = np.linalg.eig(Sigma); R2 = (np.diag(va)**0.5)@vc.T

    s1 = np.random.randn(sampleNo, 2) @ R + mu1
    s2 = np.random.randn(sampleNo, 2) @ R + mu2
    s3 = np.random.randn(sampleNo, 2) @ R + mu3
    s4 = np.random.randn(sampleNo, 2) @ R + mu4
    s5 = np.random.randn(sampleNo, 2) @ R + mu5
    s = np.vstack((s1,s2,s3,s4,s5))
    real_mean_vector = [mu1,mu2,mu3,mu4,mu5]

    return s,real_mean_vector

    # plt.plot(*s1.T,'.',label = 's1')
    # plt.plot(*s2.T,'.',label = 's2')
    # plt.plot(*s3.T,'.',label = 's3')
    # plt.plot(*s4.T,'.',label = 's4')
    # plt.plot(*s5.T,'.',label = 's5')
    # plt.axis('scaled')
    # plt.legend()
    # plt.show()

if __name__ == "__main__":
    melons,real_mean_vector = get_data()
    kmeans = KMeans(n_clusters=2, random_state=0)

    k = 5
    rnd = 0
    ROUND_LIMIT = 10
    THRESHOLD = 1e-10
    clusters = []
    mean_vectors = [[1,2],[3,4],[5,6],[5,3],[6,5]]

    while True:
        rnd += 1
        change = 0
        clusters = []
        for i in range(k):
            clusters.append([])
        for melon in melons:
            c = np.argmin(
                list(map(lambda vec: np.linalg.norm(melon - vec, ord=2),
mean_vectors))
            )

            clusters[c].append(melon)

        for i in range(k):

            new_vector = np.zeros((1, 2))
            for melon in clusters[i]:
                new_vector += melon
            new_vector /= len(clusters[i])

            change += np.linalg.norm(mean_vectors[i] - new_vector, ord=2)
            mean_vectors[i] = new_vector

        if rnd > ROUND_LIMIT or change < THRESHOLD:
            break

    print('最终迭代%d 轮' % rnd)
    colors = ['red', 'green', 'blue', 'black', 'yellow']
    for i, col in zip(range(k), colors):
        for melon in clusters[i]:
            plt.scatter(melon[0], melon[1], color=col)
    plt.show()
    error = [real_mean_vector[i] - mean_vectors[i] for i in
range(len(real_mean_vector))]
    print(error)
```

聚类结果：
1）
最终迭代 11 轮
初始化：
mean_vectors = [[1,2],[3,4],[5,6],[5,3],[6,5]]
结果：
[array([[0.94486978, -0.99414189]]), array([[1.04340334, 3.97452782]]), array([[6.02617479, 4.47221539]]), array([[5.48752479, -4.56247998]]), array([[8.92532792, 0.01747956]])]



2）
初始化
mean_vectors = [[1,1],[1.3,1.5],[1.5,1.6],[1.5,1.3],[1.6,1.5]]
最终迭代 8 轮
结果：
[array([[1.05333391, -0.97379168]]), array([[1.04473189, 4.04757586]]), array([[5.97699707, 4.51343309]]), array([[5.59030353, -4.58271968]]), array([[8.96083237, 0.03115038]])]



不同初始化相差不大