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：

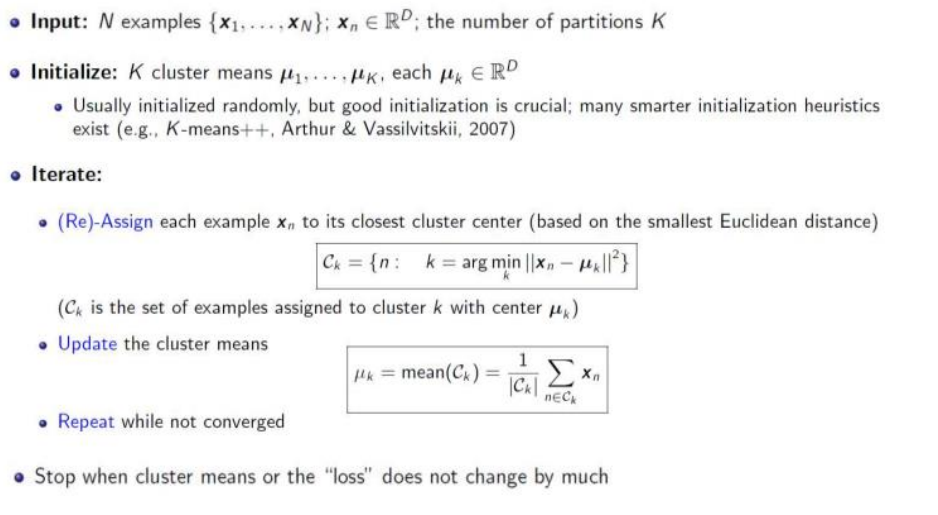
（1）各类别的先验概率相等；

（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：



影响因素：

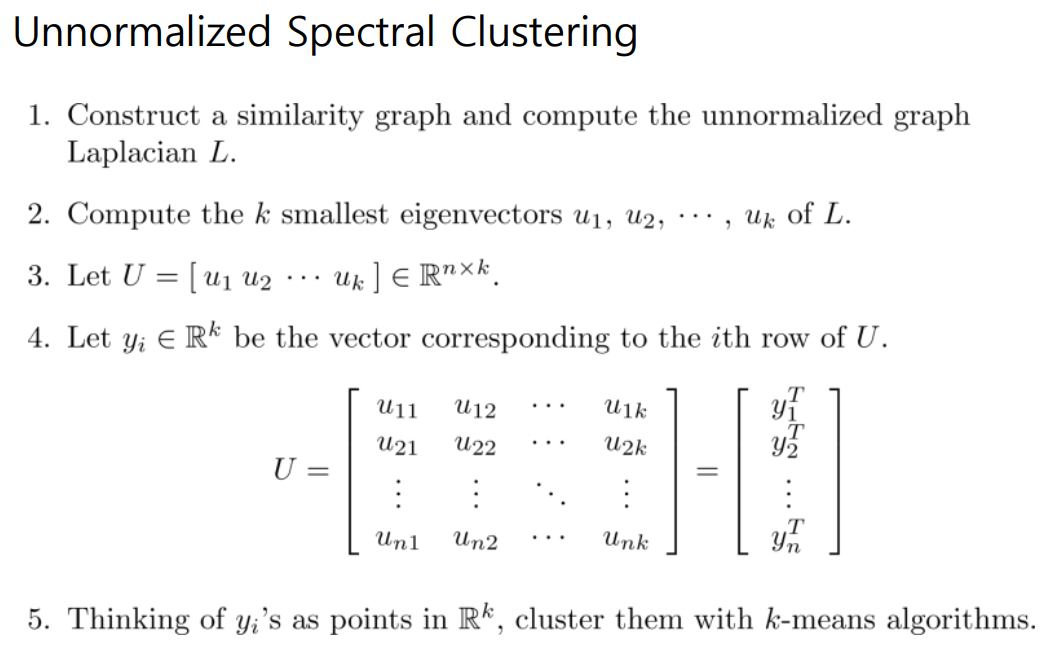
聚类数目K的选择；初始聚类中心的选择；距离的计算方式；数据的分布（流形形状），K-Means在均衡数据、类似球形分布时性能较好，在复杂流形或非凸分布性能不好。

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。

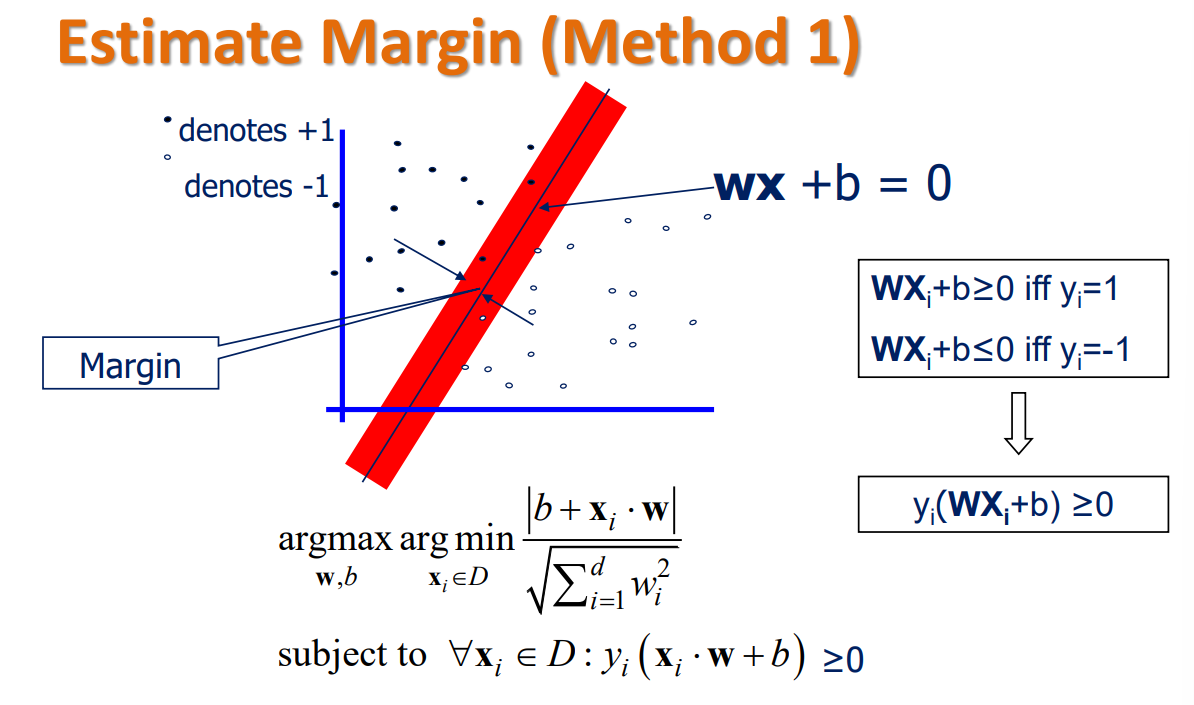
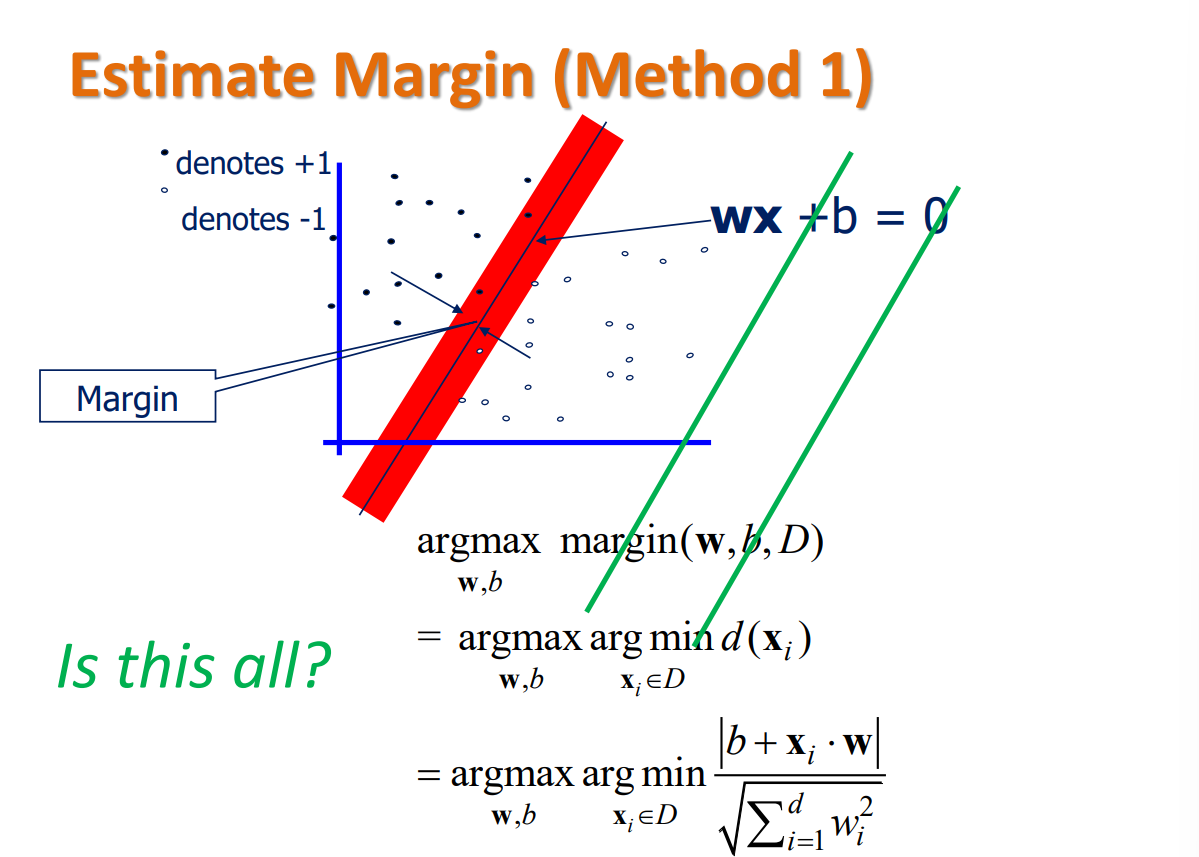
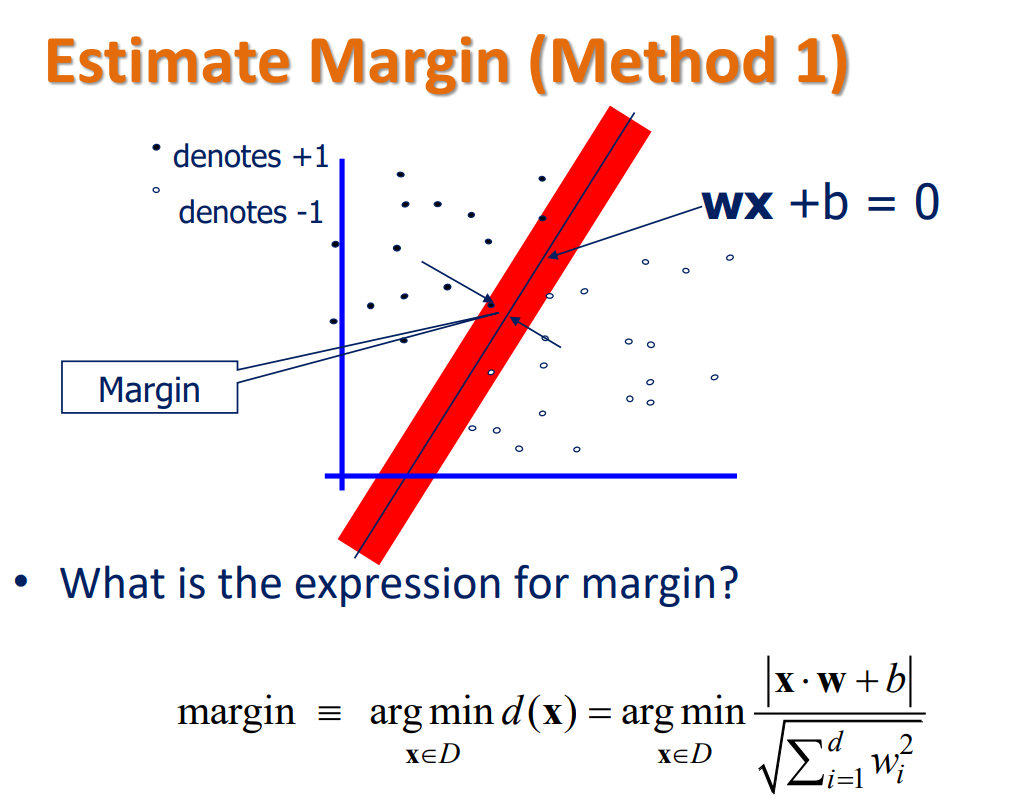
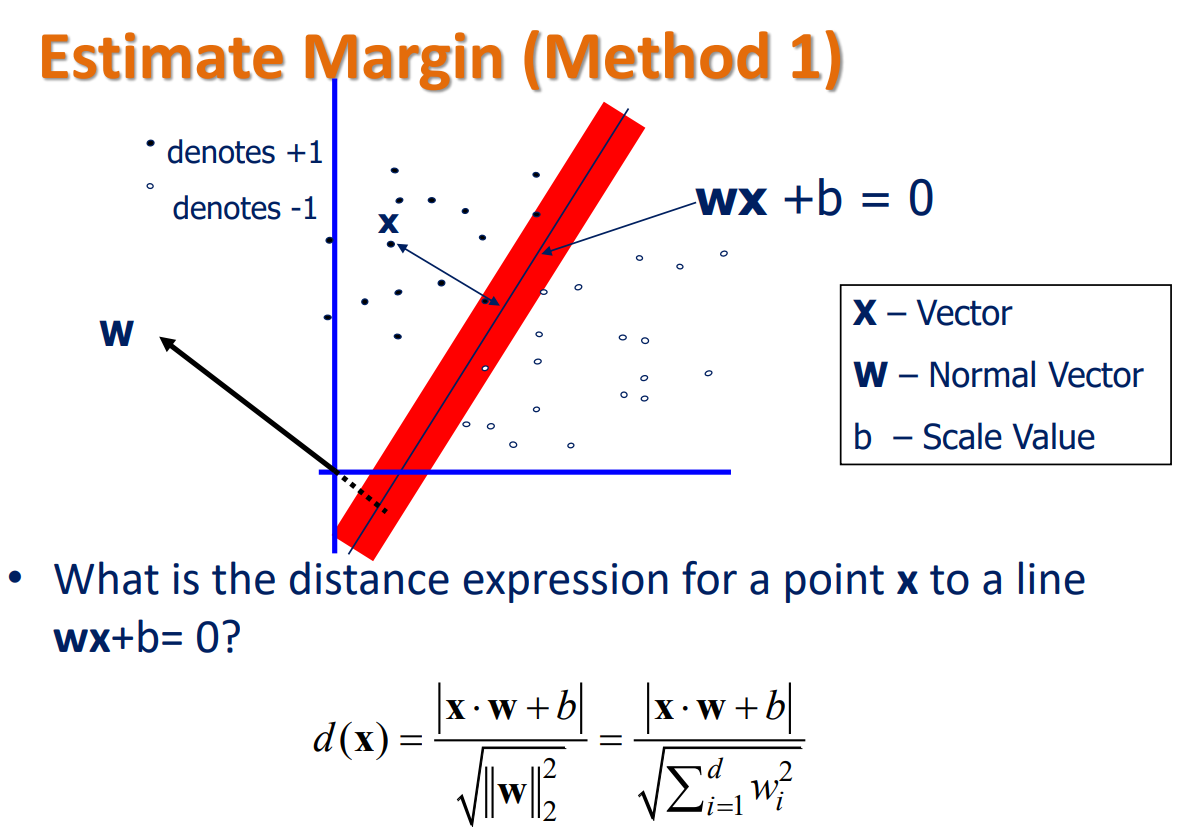
计算步骤：



影响因素：

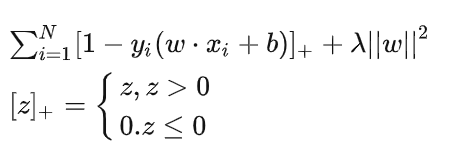
相似度的计算方法：距离度量方式；图构造方式：全局或局部；局部相似度阈值𝜀或近邻个数𝑘；图 Laplacian 矩阵是否归一化及归一化方式等。

3.



4.

Hinge loss公式为：



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

**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）

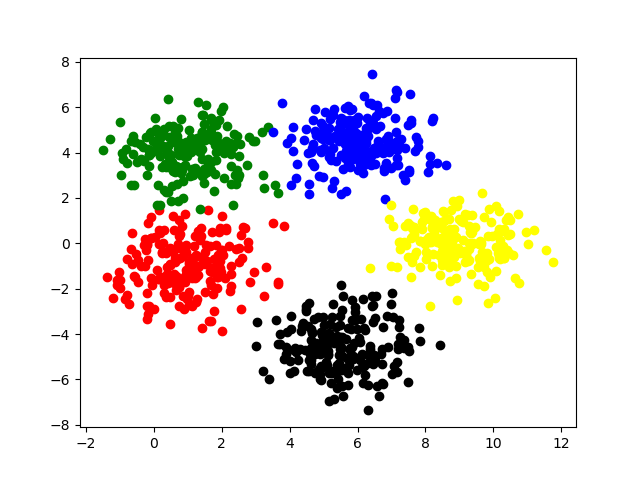
最终迭代11轮

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）

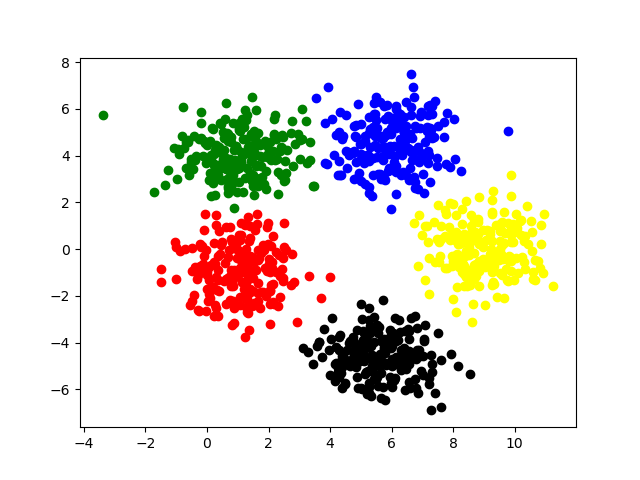
初始化

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