机器学习 **Lecture 9**

KMeans

一、 算法

Algorithm 1 KMeans

Input: 数据集 $D = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m\}$; 聚类簇数 k

1: 初始化 k 个均指向量 $\{\mu_1, \mu_2, \cdots, \mu_k\}$
2: **repeat**
3:     **for** $i = 1, 2, \cdots, m$ **do**
4:         **for** $j = 1, 2, \cdots, k$ **do**
5:             **Compute** $d_{ij} = d(\mathbf{x}_i, \mu_j)$
6:         **end for**
7:         确定第 i 个样本的簇标记 $\lambda_i = \arg\min_j d(ij)$
8:         将 $\mathbf{x}_i$ 划分到对应的簇：$C_{\lambda_j} = C_{\lambda_j} \cup \{\mathbf{x_i}\}$
9:     **end for**
10:     **for** $j = 1, 2, \cdots, k$ **do**
11:         **Update** $\mu_j = \frac{1}{|C_{\lambda_j}|} \sum_{\mathbf{x_i} \in C_{\lambda_j}} \mathbf{x}_i$
12:     **end for**
13: **until** 所有均值向量均未被更新

二、 理论

Theorem 1. *K-means 算法等价于对如下目标函数进行优化*

$$J = \frac{1}{m} \sum_{i=1}^{m} d(\mathbf{x_i}, \mu_j)$$

当采用欧式距离时，$\min J$ 等价于

$$\min \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{k} r_{ij} ||\mathbf{x}_i - \mu_j||^2 \tag{1}$$

其中 $r_{ij} = 0, 1, \sum_{j=1}^{k} r_{ij} = 1$

使用 BCD 算法对 (1) 式进行求解，即先在固定 r 的情况下更新 μ，再在固定 μ 的情况下更新 r

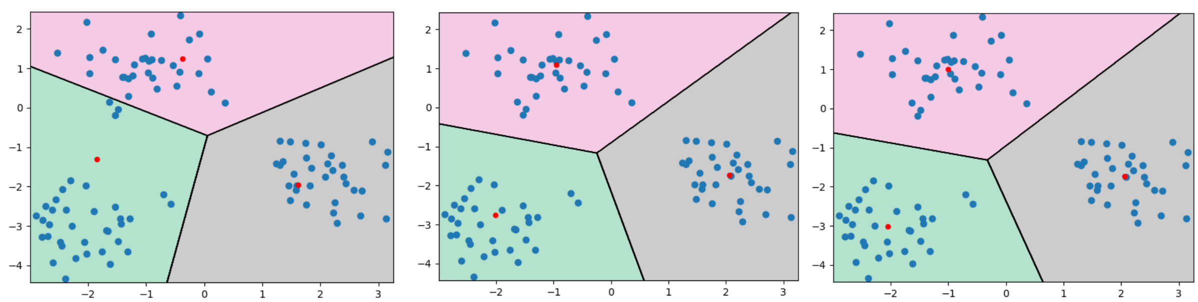
$$\nabla_{\mu_j} l(R, \mu) = -2 \sum_{i=1}^{m} r_{ij} (\mathbf{x}_i - \mu_j) = 0$$

$$\therefore \mu_j = \frac{\sum_{i=1}^{m} r_{ij} \mathbf{x}_i}{\sum_{i=1}^{m} r_{ij}} = \frac{1}{m} \sum_{i=1}^{m} r_{ij} \mathbf{x}_i$$

$$
\begin{aligned}
r_{ij} &= \arg\min_{r_{ij}} \sum_{i=1}^{m}\sum_{j=1}^{k} r_{ij}||\mathbf{x}_i-\mu_j||^2 \\
&= \arg\min_{r_{ij}} \sum_{j=1}^{k} r_{ij}||\mathbf{x}_i-\mu_j||^2 \\
&= \begin{cases} 1 & \arg\min\limits_{1\le j\le k} ||\mathbf{x}_i-\mu_j||^2 \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

三、　K-Means 迭代过程

构造中心为 [-1,1],[2,-2],[-2,-3]，标准差为 0.6，样本数量为 100 的数据集。经过三轮迭代，算法收敛，结果如图1所示。



(a)　　(b)　　(c)

图 1: K-Means 迭代过程.

四、　通过手肘法挑选 k

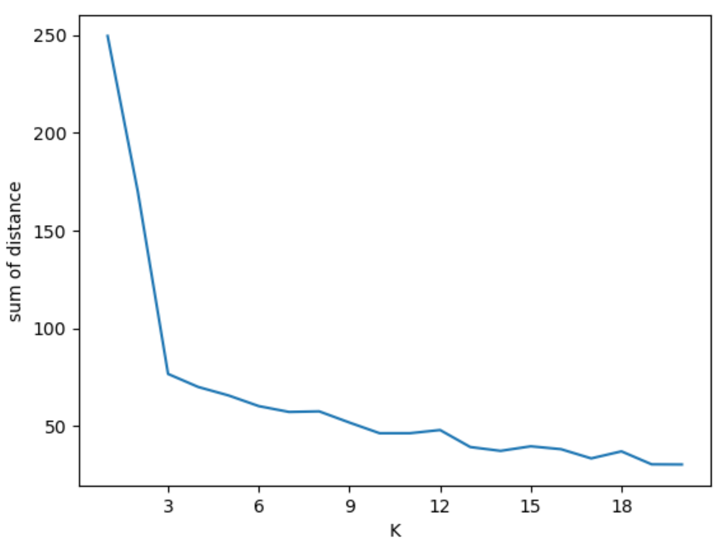


图 2: 手肘法.

手肘法的核心思想是：随着聚类数 k 的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和 SSE 自然会逐渐变小。并且，当 k 小于真实聚类数时，由于 k 的增大会大幅增加每个簇的聚合程度，故 SSE 的下降幅度会很大，而当 k 到达真实聚类数时，再增加 k 所得到的聚合程度回报会迅速变小，所以 SSE 的下降幅度会骤减，然后随着 k 值的继续增大而趋于平缓，也就是说 SSE 和 k 的关系图是一个手肘的形状，而这个肘部对应的 k 值就是数据的真实聚类数。

从图2中可以看出，$k = 3$ 为一个拐点，因此 k 取 3 时效果最优，这与构造的数据相吻合。

五、 代码实现

```python
import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt
from tqdm import tqdm
from matplotlib.ticker import MaxNLocator

class KMeans:
    def __init__(self, x, k, resolustion=1000, plot=True):
        self.x = np.array(x)
        self.k = k
        self.m = len(x)
        self.meanVectors = self.x[np.random.choice(self.m,k,replace=False)]
        self.clusters = [[]for i in range(self.k)]
        self.resolustion = resolustion
        self.plot = plot

    def train(self):
        '''
        KMeans 迭代训练
        :return: 均值向量
        '''
        cnt = 0
        while True:
            # ------------------------------------------- #
            # 初始化所有簇
            # ------------------------------------------- #
            self.clusters = [[]for i in range(self.k)]
            # ------------------------------------------- #
            # 计算最近簇
            # ------------------------------------------- #
            for i in range(self.m):
                d = float('inf')
                index = 0
                for j in range(self.k):
                    d_ij = norm(self.x[i]-self.meanVectors[j])
                    if d_ij < d:
                        d = d_ij
                        index = j
                self.clusters[index].append(self.x[i])
```

```
            # ------------------------------------------ #
            # 更新均值向量
            # ------------------------------------------ #
            done = True
            for i in range(self.k):
                if len(self.clusters[i])!=0:
                    newMeanVector = np.average(np.array(self.clusters[i]),axis=0)
                else:
                    newMeanVector = self.meanVectors[i]
                if not np.array_equal(newMeanVector,self.meanVectors[i]):
                    self.meanVectors[i] = newMeanVector
                    done = False
            # print('# ------------------------------------ #')
            # print(self.meanVectors)
            # ------------------------------------------ #
            # 绘制决策边界
            # ------------------------------------------ #
            if self.plot:
                self.plot_decision_boundaries(iteration=cnt,resolustion=self.resolustion)
            cnt += 1
            # ------------------------------------------ #
            # 判断结束条件 退出循环
            # ------------------------------------------ #
            if done:
                return self.meanVectors

    def predict(self, x):
        '''
        预测
        :param x: 某个给定向量
        :return: 距离x最近的均指向量的编号
        '''
        d = float('inf')
        index = 0
        for i in range(self.k):
            d_ij = norm(x-self.meanVectors[i])
            if d_ij < d:
                d = d_ij
                index = i
        return index

    def select_k(self, start=1, end=20, plot=False):
        '''
        手肘法确定k的取值
        :return:
        '''
        all_distance = []
        for k in range(start,end+1):
            self.__init__(self.x, k, plot=plot)
            self.train()
```

```python
            all_distance.append(self.sumDistance())
        plt.plot(range(1,len(all_distance)+1),all_distance)
        plt.ylabel('sum of distance')
        plt.xlabel('K')
        plt.gca().xaxis.set_major_locator(MaxNLocator(integer=True))
        plt.show()


    def sumDistance(self):
        '''
        计算点到聚点的距离之和
        :return:
        '''
        sumdistance = 0
        for i in range(self.m):
            d = float('inf')
            for j in range(self.k):
                d_ij = norm(self.x[i]-self.meanVectors[j])
                if d_ij<d:
                    d = d_ij
            sumdistance += d
        return sumdistance

    def plot_decision_boundaries(self, resolustion=1000, iteration=0):
        '''
        绘制决策边界
        :param resolustion: 网格密度
        :param iteration: 迭代次数
        :return:
        '''
        mins = self.x.min(axis=0) - 0.1
        maxs = self.x.max(axis=0) + 0.1
        xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolustion),
                             np.linspace(mins[1], maxs[1], resolustion))
        grid = np.c_[xx.ravel(), yy.ravel()]
        predict = []
        for i in tqdm(grid):
            predict.append(self.predict(i))
        predict = np.array(predict)
        predict = predict.reshape(xx.shape)
        # print(predict)

        plt.contourf(predict, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                   cmap='Pastel2')

        plt.contour(predict, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                  linewidths=1, colors='k')

        plt.scatter(self.x[:,0],self.x[:,1])
        plt.scatter(self.meanVectors[:,0],self.meanVectors[:,1],s=20,c='r')
```

```python
        # plt.show()
        plt.savefig('kmeans_{}.png'.format(iteration))
        plt.close()
```