K-means 聚类算法

1. 实验说明

本次实验主要通过使用 K-means 聚类算法，对数据样本进行聚类处理，并结合实际分类标签，计算 NMI 来判断聚类结果的好坏。

2. 实验方法

2.1 算法

K-means 算法：

输入：输入各个点的坐标，坐标用 n 维向量表示

输出：对于每个点，计算其聚类标签，输出 NMI

1.随机选择 K 各点，编号为 1-K，作为最初始聚类中心，将各个点的位置向量存放于 k_list 列表中，对于聚类中心 i，维护一个聚类列表 division[i]，存放属于该聚类中心的点的编号。对于训练样本的一个点，保留其位置、聚类标签、实际标签（用于计算 NMI 值）、编号。

2.对于每一个点，计算到每一个聚类中心的距离，将其标签设为距离最小的聚类中心在 k_list 中的编号，将属于聚类中心 i 的数据点的编号，加入对应的聚类列表 division[i]。

3.对于聚类列表 division[i]，加权平均、计算新的聚类中心，并用新的聚类中心的坐标，更新 k_list，重新计算聚类列表

4.计算 NMI 值，并输出

5.重复 2-4，直至一定次数

2.2 具体代码

定义了一个 Point 类，用来存储点的位置，k-means 贴上的标签，实际的标签，order 用于标识不同的点。

```
1.
2.  class Point:
3.      """
```

```
    存储每一个数据点，属性包括：位置 position(n 维列表)；聚类产生的标签 label；
    实际聚类标签 answer；用来进行编号的 order
    """
    def __init__(self, position, answer, order, label="null"):
        self.position = position  # 用 ndarray 数组存储
        self.answer = answer
        self.order = order
        self.label = label

    def show_point(self):  # 打印输出
        print("position: ", self.position, "\nlabel: ", self.label, "\norder:", self.order, "\nanswer: ", self.answer)

```

读取文件，存放于以 Point 为元素的 point_list 中

```
def load_file():
    # 读取文件，数据存放在二维 ndarray 数组 data
    file_name = "./breast.txt"
    with open(file_name, 'r') as fp:
        fp = open(file_name, 'r')
        lines = fp.readlines()
        data = np.array([[float(_) for _ in line.split()] for line in lines])
    # 将数据赋值给以 point 为元素的列表
    seq = 0
    for _ in data:  # 数据的最后一列为实际的分类标签，用于计算准确率
        pos = _[:-1:1]
        ans = int(_[-1] / 2) - 1  # 将用于验证的实际分类标签从 2.0、4.0 转化为 0、1
        point_list.append(Point(pos, ans, seq))
        seq = seq + 1

```

对于每个点，计算到各个聚类中心的距离，选择最近的聚类中心的编号作为该点的标签。

```

def calc_label():
    # 根据聚类中心计算点的标签
    # print("division: ", division)
    for point in point_list:
```

```
        distance = [np.sqrt(np.sum(np.square(point.position - center))) for center in k_list]
        # print("distance: ", distance)
        point.label = distance.index(min(distance))  # 将标签设为距离最近的聚类中心在 k_list 中的编号
        division[point.label].append(point.order)  # 对每个聚类中心，维护一个属于该聚类中心的点的集合（用 order 表示）
```

对于属于同一个聚类标签的点，计算加权位置，作为新的聚类标签。

```
# 根据各个点的标签，更新聚类中心
def update_cluster():
    tmp = [0 for _ in range(len(point_list[0].position))]  # 用于保存新的计算所得的聚类中心
    for center_order in range(len(k_list)):  # 对于每个聚类中心
        for p in division[center_order]:  # 对于属于该聚类中心的各个点
            tmp = [tmp[i] + point_list[p].position[i] for i in range(len(point_list[p].position))]  # 计算各个维度累加距离
        tmp = [tmp[i] / len(division[center_order]) for i in range(len(tmp))]  # 计算加权中心
        k_list[center_order] = [tmp[i] for i in range(len(tmp))]  # 用计算所得的加权中心，更新 k_list 列表中聚类中心的位置
        # print("k_list[center_order]: ", k_list[center_order])  # 用于打印聚类中心点的坐标

```

计算 NMI 值

```
def verify():  # 计算 NMI，即归一化互信息，所用变量为存储在 data_list 中已经分类好的点
    """
    参考链接 1：https://www.jianshu.com/p/43318a3dc715
    参考链接 2：https://blog.csdn.net/hang916/article/details/88783931
    :return: 打印输出，并返回 NMI 值
    """
    p_grp_gnd = [[0 for i in range(k)] for j in range(ans_count)]  # 联合条件概率分布：grp 表示聚类后的 group，gnd 表示 ground truth
    p_grp = [0 for i in range(k)]  # grp 表示聚类后的 group 边界分布
    p_gnd = [0 for i in range(ans_count)]  # gnd 表示 ground truth 边界分布
    for p in point_list:
        p_grp[p.label] += 1  # 统计聚类产生的标签
        p_gnd[p.answer] += 1  # 统计实际分类的标签
        p_grp_gnd[p.label][p.answer] += 1  # 用于计算联合概率分布
```

```
    p_grp = [i / len(point_list) for i in p_grp]  # 计算聚类为 group 的边界概率分布
    p_gnd = [i / len(point_list) for i in p_gnd]  # 计算实际 ground truth 的边界概率分布
    p_grp_gnd = [[i / len(point_list) for i in t] for t in p_grp_gnd]  # 计算联合概率分布
    h_grp = -sum([i*math.log(i, 2) for i in p_grp])  # 计算聚类结果的信息熵
    h_gnd = -sum([i*math.log(i, 2) for i in p_gnd])  # 计算实际结果的信息熵
    # h_grp_gnd = sum([p_grp[i] * (math.log(p_gnd[i], 2) - math.log(p_grp[i], 2)) for i in range(len(p_grp))])  # 计算相对熵
    tmp = sum([sum([p_grp_gnd[i][j]*(math.log(p_grp_gnd[i][j], 2) - math.log(p_grp[i]*p_gnd[j], 2)) for i in range(2)]) for j in range(2)])
    nmi = 2 * tmp / (h_grp + h_gnd)
    print("NMI: %.4f" % nmi)
    return nmi

```

主函数

```

if __name__ == "__main__":
    # 初始化数据
    point_list = []  # 以 Point 为元素的列表，用于存储输出点的信息
    k = 2  # K-means 参数
    ans_count = 2  # 实际聚类的标签种类
    load_file()  # 加载数据
    # 初始化 k_list 和 division
    k_list = np.array([point_list[random.randint(0, len(point_list) - 1)].position for i in range(k)])  # 初始随机生成的聚类中心的坐标
    division = [[] for _ in range(len(k_list))]  # 全局变量，用于保存属于聚类中心的点的编号

    n = 1  # 迭代计算
    while n < 20:
        print("第 ", n, " 次迭代: ")
        calc_label()
        update_cluster()
        verify()
        n = n + 1
```

3. 实验结果

比较好的一次结果截图，容易看到，在不稳定时 NMI 值可能达到 0.8。多次运行，稳定时结果在 0.72-0.76 之间

图 3.1(a) 运行结果截图

图 3.1(b) 运行结果截图