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。

1. 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。

2. **class** Point:
3. """
4. 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；
5. 实际聚类标签answer；用来进行编号的order
6. """
7. **def** \_\_init\_\_(self, position, answer, order, label="null"):
8. self.position = position  # 用ndarray数组存储
9. self.answer = answer
10. self.order = order
11. self.label = label
13. **def** show\_point(self):  # 打印输出
14. **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

1. **def** load\_file():
2. # 读取文件，数据存放在二维ndarray数组data
3. file\_name = "./breast.txt"
4. with open(file\_name, 'r') as fp:
5. fp = open(file\_name, 'r')
6. lines = fp.readlines()
7. data = np.array([[float(\_) **for** \_ **in** line.split()] **for** line **in** lines])
8. # 将数据赋值给以point为元素的列表
9. seq = 0
10. **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
11. pos = \_[:-1:1]
12. 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
13. point\_list.append(Point(pos, ans, seq))
14. 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。

2. **def** calc\_label():
3. # 根据聚类中心计算点的标签
4. # print("division：", division)
5. **for** point **in** point\_list:
6. distance = [np.sqrt(np.sum(np.square(point.position - center))) **for** center **in** k\_list]
7. # print("distance: ", distance)
8. 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
9. 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。

1. # 根据各个点的标签，更新聚类中心
2. **def** update\_cluster():
3. 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
4. **for** center\_order **in** range(len(k\_list)):  # 对于每个聚类中心
5. **for** p **in** division[center\_order]:  # 对于属于该聚类中心的各个点
6. 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
7. tmp = [tmp[i] / len(division[center\_order]) **for** i **in** range(len(tmp))]  # 计算加权中心
8. 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
9. # 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

1. **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
2. """
3. 参考链接1：https://www.jianshu.com/p/43318a3dc715
4. 参考链接2：https://blog.csdn.net/hang916/article/details/88783931
5. :return: 打印输出，并返回NMI值
6. """
7. 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
8. p\_grp = [0 **for** i **in** range(k)]  # grp表示聚类后的group边界分布
9. p\_gnd = [0 **for** i **in** range(ans\_count)]  # gnd表示ground truth边界分布
10. **for** p **in** point\_list:
11. p\_grp[p.label] += 1  # 统计聚类产生的标签
12. p\_gnd[p.answer] += 1  # 统计实际分类的标签
13. p\_grp\_gnd[p.label][p.answer] += 1  # 用于计算联合概率分布
14. p\_grp = [i / len(point\_list) **for** i **in** p\_grp]  # 计算聚类为group的边界概率分布
15. p\_gnd = [i / len(point\_list) **for** i **in** p\_gnd]  # 计算实际ground truth的边界概率分布
16. 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
17. h\_grp = -sum([i\*math.log(i, 2) **for** i **in** p\_grp])  # 计算聚类结果的信息熵
18. h\_gnd = -sum([i\*math.log(i, 2) **for** i **in** p\_gnd])  # 计算实际结果的信息熵
19. # 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
20. 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)])
21. nmi = 2 \* tmp / (h\_grp + h\_gnd)
22. **print**("NMI: %.4f" % nmi)
23. **return** nmi

主函数

2. **if** \_\_name\_\_ == "\_\_main\_\_":
3. # 初始化数据
4. point\_list = []  # 以Point为元素的列表，用于存储输出点的信息
5. k = 2  # K-means参数
6. ans\_count = 2  # 实际聚类的标签种类
7. load\_file()  # 加载数据
8. # 初始化k\_list和division
9. 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
10. 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
12. n = 1  # 迭代计算
13. **while** n < 20:
14. **print**("第 ", n, " 次迭代：")
15. calc\_label()
16. update\_cluster()
17. verify()
18. 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(b) 运行结果截图

图3.1(a) 运行结果截图