实验四 KMEANS 报告

江辉--171250506

2020-4-20

南京大学

软件学院

目录

一、 实验内容描述.....1

1、 项目目录描述.....1

二、 实验结果图.....2

三、 算法原理阐述.....4

四、 核心代码讲解.....4

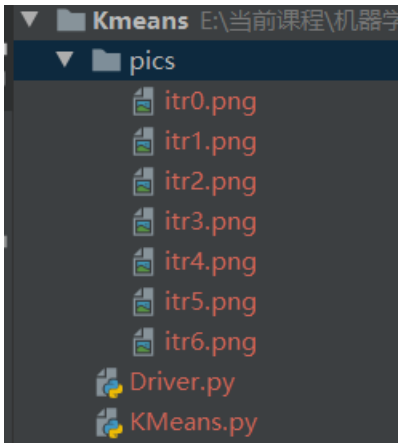
1、 KMeans 算法类.....4

一、实验内容描述

本程序用 python 编写，手写 KMeans 算法的实现。100 个随机点和随机数量个核(我代码里随机范围是 3-13)，通过 Matplotlib 绘制每一次迭代的聚类过程

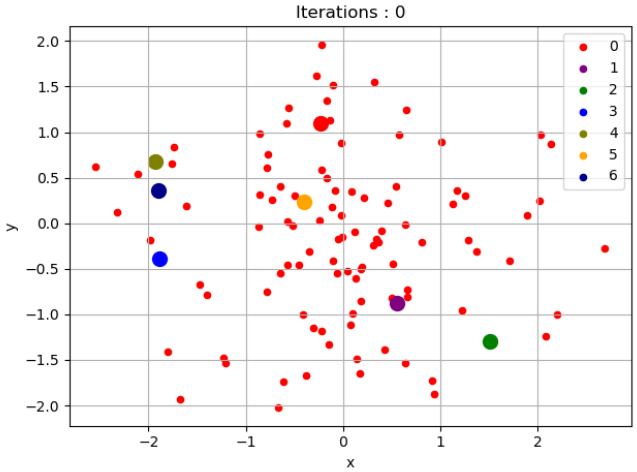
1、项目目录描述

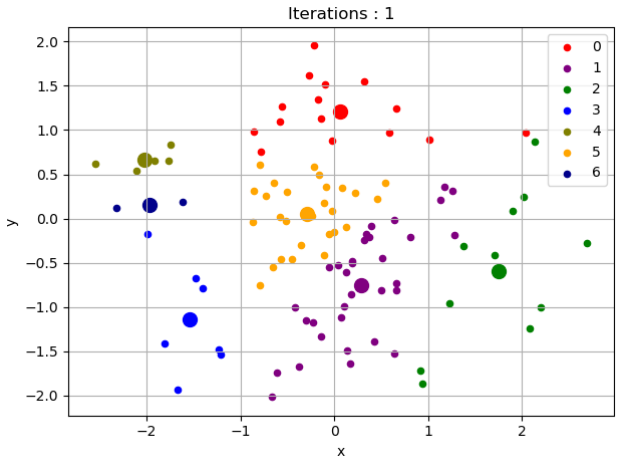
本项目推荐使用 pycharm 打开
KMeans.py 里写了实现的代码，同时附带了 pyplot 画图代码
Driver.py 里是 main 函数入口
pics 目录下是每一次迭代画出来的图

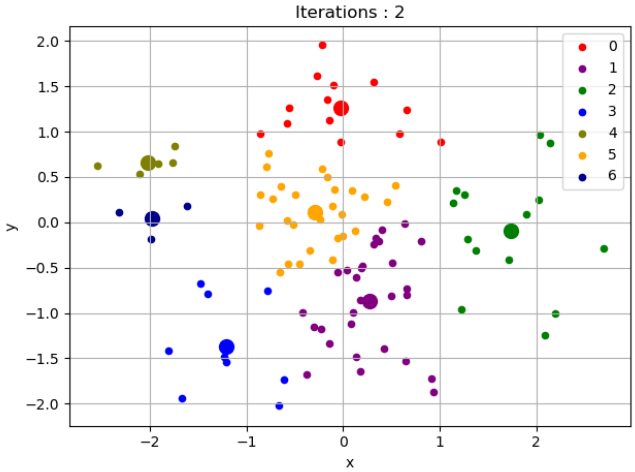


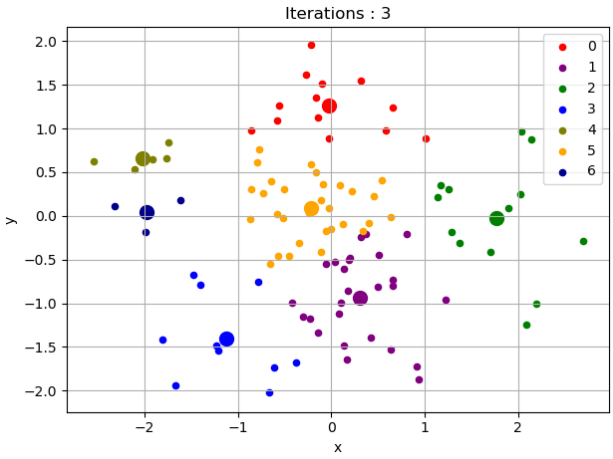
二、实验结果图

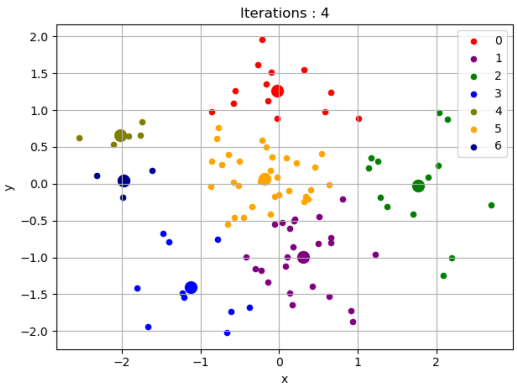
以下是用 pyplot 画的各迭代的图。该用例运行的时候随机到有 7 个聚类中心，各个颜色类的中心点用比较大的点表示。可以看到其实在前几次计算后各个中心点就大致确定方位了，在后续的聚类过程中只有个别中心点的微调(从相对网格的位置线可以看出来，图中蓝色点中心变化比较明显)，所以每次迭代时中心点的移动都不大，在第 6 次迭代的时候就完全稳定了

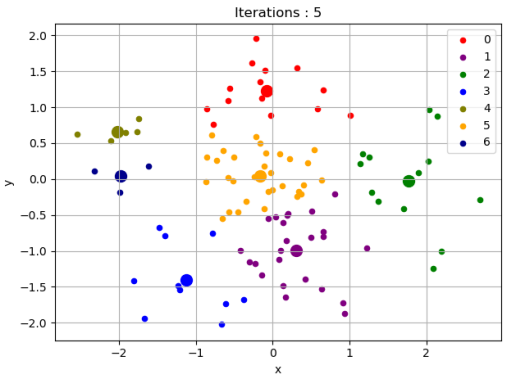


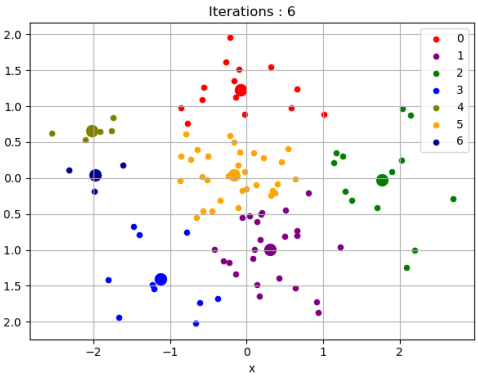












下面是 cost 变化和中心点坐标变化，可以发现，cost 在刚开始下降很快，后面就下降很少了，也就是说基本稳定了，只有个别点还在变动，第 6 次迭代就已经收敛了

```
kernel num : 7
iteration : 0
cost: 351.3194064267526
kernels:
[[-0.23277029  1.09763498]
 [ 0.55620221 -0.87815548]
 [ 1.50712195 -1.29607395]
 [-1.88372681 -0.39044637]
 [-1.93487597  0.68065662]
 [-0.39565876  0.2377784 ]
 [-1.89931709  0.36023053]]
iteration : 1
cost: 52.11847736708145
kernels:
[[ 0.06095316  1.21123239]
 [ 0.28937723 -0.7513473 ]
 [ 1.74530977 -0.598994  ]
 [-1.54036986 -1.14372306]
 [-2.01531905  0.6603246 ]
 [-0.29026322  0.0557305 ]
 [-1.96806023  0.150799  ]]
iteration : 2
cost: 37.55262050598384
kernels:
[[-0.02642128  1.26477641]
 [ 0.27004348 -0.87322656]
 [ 1.73044734 -0.09549378]
 [-1.20665188 -1.37050714]
 [-2.01531905  0.6603246 ]
 [-0.28985189  0.11171141]
 [-1.97394731  0.04069896]]
```

```
iteration : 3
cost: 36.55004828076579
kernels:
[[-0.02642128  1.26477641]
 [ 0.3100942  -0.93640206]
 [ 1.76916204 -0.02888153]
 [-1.12385397 -1.40092871]
 [-2.01531905  0.6603246 ]
 [-0.21504535  0.09174537]
 [-1.97394731  0.04069896]]
iteration : 4
cost: 36.234819359023994
kernels:
[[-0.02642128  1.26477641]
 [ 0.30754801 -0.99536893]
 [ 1.76916204 -0.02888153]
 [-1.12385397 -1.40092871]
 [-2.01531905  0.6603246 ]
 [-0.18031448  0.07171131]
 [-1.97394731  0.04069896]]
iteration : 5
cost: 36.14143286621628
kernels:
[[-0.07997194  1.22861284]
 [ 0.30754801 -0.99536893]
 [ 1.76916204 -0.02888153]
 [-1.12385397 -1.40092871]
 [-2.01531905  0.6603246 ]
 [-0.16109461  0.04955728]
 [-1.97394731  0.04069896]]
```

```
iteration : 6
cost: 36.14143286621628
kernels:
[[-0.07997194  1.22861284]
 [ 0.30754801 -0.99536893]
 [ 1.76916204 -0.02888153]
 [-1.12385397 -1.40092871]
 [-2.01531905  0.6603246 ]
 [-0.16109461  0.04955728]
 [-1.97394731  0.04069896]]
finish at iteration : 6
```

三、 算法原理阐述

K-Means 算法是一种无监督分类算法，只需要数据，而不需要标签。假设预先给定 n 个训练样本（无标记），如：{X1,X2,⋯,Xn}，同时给定聚类的个数 K。
目标：把比较“接近”的样本放到一个簇类（cluster）里，总共得到 K 个簇类（cluster）

具体做法：
1、初始化 k 个中心点 n1、n2、、、nk
2、计算所有点 Xi 与所有中心点 n 的距离(可以是欧氏距离、曼哈顿距离核函数映射后距离等)，取距离最近的的中心点作为该点的簇中心
3、根据归入各个簇的样本点，分别计算各个簇新的簇中心(可以取簇内所有样本点均值，也可以取最中心的样本点作为簇中心等)
4、重复步骤 2、3 直至所有簇中心不再变化

四、 核心代码讲解

1、KMEANS 算法类

下图是我实现的 KMeans 算法类
is_convergent 函数用于判断是否收敛
classify 用于重新将素有点进行归类
cal_new_kernel 是重新计算中心点坐标
cal_cost 是计算误差值
run 是 kmeans 主体
log 是做绘图、打印中心点坐标、cost 值等
plot 是绘图方法

```
class KMeans:
    def __init__(self, pt_num=100, kn_num=5, max_itr=100):...

    def is_convergent(self):...

    def classify(self):...

    def cal_cost(self):...

    def cal_new_kernel(self):...

    def run(self):...

    def plot(self, itr_num):...

    def log(self, itr):...
```

__init__中可以看到就是默认参数是 100 个随机点，5 个中心点，最多 100 次迭代(在 main 函数里刷随机数传入即可修改）。classes 存了各个点对应的簇中心序号，last_kernels 是记录上一次迭代时的中心点坐标，用于判断是否收敛

```
class KMeans:
    def __init__(self, pt_num=100, kn_num=5, max_itr=100):
        """
        :param pt_num: 总点数
        :param kn_num: 核数，不要超过定义好的颜色数(20)
        :param max_itr: 最大迭代次数
        """
        self.pt_num = pt_num
        self.kn_num = kn_num
        self.max_itr = max_itr
        self.points = np.random.randn(pt_num, 2)
        self.classes = np.zeros(pt_num)
        self.kernels = np.random.randn(kn_num, 2)
        self.last_kernels = None
        self.colors = ["red", "purple", "green", "blue", "olive",
                       "orange", "darkblue", 'yellowgreen', "pink", "yellow",
                       "brown", "black", "peru", "orchid", "palegreen",
                       "navajowhite", "navy", "oldlace", "rosybrown", "saddlebrown"]
```

整个 KMeans 算法流程就是对每一次迭代先判断是否已经收敛，收敛则退出，不收敛则依次进行点归类、重算中心点、画图、打印中心点坐标和计算 cost

```
def run(self):
    self.log(0)
    for i in range(self.max_itr):
        if not self.is_convergent():
            self.classify()
            self.cal_new_kernel()
            self.log(i + 1)
        else:
            self.plot(i)
            print("finish at iteration : " + str(i))
            break
```

记录并更新簇中心用的是簇内平均坐标

```
def cal_new_kernel(self):
    """
    记录上一次迭代的中心坐标，并更新类内平均坐标作为新中心点
    """
    self.last_kernels = self.kernels.copy()
    for i in range(self.kn_num):
        self.kernels[i] = np.average(self.points[self.classes == i], axis=0)
```

距离计算采用的是欧氏距离，下面的 classify 方法是将每个点归类，计算与所有簇中心的欧氏距离，取最小距离的类保留在 classes 数组内

```
def classify(self):
    """
    采用欧氏距离
    """
    for i in range(self.pt_num):
        self.classes[i] = np.argmin(np.sum((self.points[i] - self.kernels) ** 2, axis=1))

def cal_cost(self):
    """
    按各点和所属簇中心计算欧氏距离作为cost
    """
    cost = 0
    for i in range(self.kn_num):
        cost += np.sum((self.points[self.classes == i] - self.kernels[i]) ** 2)
    return cost
```

判断聚类是否停止，我在每次迭代后都会记录的上一次的簇中心坐标，因此只需要比对每次迭代所有簇中心坐标是否都和上一次迭代一样即可

```
def is_convergent(self):
    if self.last_kernels is None:
        return False
    else:
        return (self.kernels == self.last_kernels).all()
```