机器学习 第一次作业

K-means 作业

201250076 袁家乐

2023 年 3 月 2 日

一、k-means 聚类算法原理阐述

k-means 聚类算法是动态聚类算法的一种，对于动态聚类算法而言，其首先选择若干个样本点作为聚类中心，再按某种聚类准则使样本点向各中心聚集，从而得到初始聚类；然后判断初始分类是否合理，若不合理，则修改聚类；如此反复进行修改聚类的迭代算法，直至合理为止。

具体到 K-means 聚类算法的步骤上。首先我们选择一个聚类数量 k，通过随机选择样本点的方式初始化 k 个聚类中心。然后进入迭代部分，在每一次迭代中，对每个样本点，计算样本点到 k 个聚类中心的距离（使用距离度量算法，如欧氏距离等），将样本点划入与其最近邻的聚类中；在所有样本点划分完毕后，重新计算聚类中心，聚类中心设定为此聚类中所有样本的均值。直到各样本所属聚类完全稳定下来，停止迭代。

K-means 算法的 k 值选择往往需要试探，且更适用于具备优良的独立区域分布的样本。

二、实验步骤与核心代码解说

本次实验使用 python 代码，用到的类库有 numpy、random 和 PIL.Image 等。主要实现了 k-means 聚类算法和使用 k-means 聚类算法实现图像分割、结果可视化。封装的函数如下：

main 函数入口

def imgPreProcess(path) #图像读取和预处理

def calculateDistance(vector1,vector2) #计算两个向量间的欧氏距离（平方）

def myKmeans(data,k) #KMeans 算法实现

下面，从 main 函数入口开始，解说实验步骤，并针对各封装的函数阐明核心代码的设计。

```
'''
    图像读取和预处理
    :param path 图像路径
    :return imgData 预处理完成的图像数据
'''
def imgPreProcess(path):...
```

```
'''
    计算两个向量间的欧氏距离（平方）
    :param vector1 向量1
    :param vector2 向量2
    :return ans 计算结果
'''
def calculateDistance(vector1, vector2):...
```

```
'''
    KMeans算法实现
    :param data 图像预处理后存储的矩阵数据
    :param k 分类中心数
    :return [centerPoints,pointsCluster] 聚类中心，样本点聚类情况
'''
def myKmeans(data,k):...
```

如下图所示，main 函数入口进入后，首先调用 imgPreProcess()方法，将原始图片读取并预处理成 numpy 的 matrix 格式。接着，k 值从 1 取至 6，分别执行在不同 k 值下的聚类方法图形分割操作，调用 myKmeans()方法，得到生成的聚类中心矩阵和聚类标签矩阵。最后，新建一个画布 newPicture，按聚类标签矩阵找到各像素点样本的簇号，再以簇号为索引在聚类中心矩阵中获取对应的 RGB 值，以聚类中心的值替代像素点样本值填入新的画布，最后将结果保存为.jpg 格式。

```
if __name__=='__main__':
    # 图片选自：麦田里的丝柏树，梵高，1889
    # Wheat Field with Cypresses,Vincent Willem van Gogh,1889
    pre_data=imgPreProcess("picture.jpg")
    for k in range(1,7,1):
        centerPoints,pointsCluster=myKmeans(pre_data,k)
        # 将聚类后的图片可视化保存
        resultPath="result-"+str(k)+".jpg"
        file=open("picture.jpg","rb")
        tmpImg=pilImage.open(file)
        imgLength,imgWidth=tmpImg.size
        file.close()
        newPicture = pilImage.new("RGB", (imgLength,imgWidth))
        num= np.shape(pointsCluster)[0]
        for i in range(num):
            clusterIndex = int(pointsCluster[i, 0])
            r = int(float(centerPoints[clusterIndex, 0]) * 256)
            g = int(float(centerPoints[clusterIndex, 1]) * 256)
            b = int(float(centerPoints[clusterIndex, 2]) * 256)
            newPicture.putpixel((int(i / imgWidth), (i % imgWidth)), (r, g, b))
        newPicture.save(resultPath,"JPEG")
```

如下所示，在 imgPreProcess()中，根据路径读取原图文件，提取 RGB 值以 numpy 的 matrix 格式存储，填入 imgData。对于原图文件访问时，应进行只读的权限控制，并及时关闭文件。

```
def imgPreProcess(path):
    # 读入图像
    file=open(path,"rb")
    imgData=[]
    tmpImg=pilImage.open(file)
    # 获取图像长度、宽度
    imgLength,imgWidth=tmpImg.size
    # 依据像素点读取图像rgb值，填入imgData
    for i in range(0,imgLength,1):
        for j in range(0,imgWidth,1):
            imgPixel=[]
            r,g,b=tmpImg.getpixel((i,j))
            imgPixel.append(r/256.0)
            imgPixel.append(g/256.0)
            imgPixel.append(b/256.0)
            imgData.append(imgPixel)
    file.close()
    # 将imgData以numpy矩阵形式返回
    imgData=np.mat(imgData)
    return imgData
```

如下所示，在 calculateDistance()中，根据欧式距离计算公式计算欧式距离，考虑到时间复杂度的问题，此处采纳欧式距离的平方形式，不再冗余地进行开根号操作，此处的距离度量只用做数值比较。

```
def calculateDistance(vector1, vector2):
    ans = (vector1 - vector2) * (vector1 - vector2).T
    return ans[0, 0]
```

最主要的 myKmeans()如下，首先需完成的是初始聚类中心的选取。此处使用 random.sample()方法，在样本中无重复的随机选定 k 个点作为初始样本点。

```
def myKmeans(data,k):
    # 随机选取初始的k个中心
    pointsNum,attributeNum=np.shape(data)
    centerPoints=np.mat(np.zeros((k,attributeNum)))
    randomList=random.sample(range(0,pointsNum-1),k)
    for i in range(0,k,1):
        for j in range(0,attributeNum,1):
            centerPoints[i,j]=data[randomList[i],j]
```

接下来，使用 while 循环进行迭代。结合机器的算力和实际呈现的效果，确

定迭代的终止条件为达到稳态（每次需调整聚类的样本点数不超过 2%）或迭代超过 10 次。

各次迭代中，计算各样本点到各聚类中心的距离（采用欧氏距离的平方，调用我自己封装的 calculateDistance()函数），记录最近距离的聚类簇号，然后将此样本点划入最近的聚类中。

```
# 迭代执行聚类工作
print("start cluster loop...")
pointsCluster=np.mat(np.zeros((pointsNum,2)))
loopCounter=0
loopFlag=True
# 迭代停止条件：达到稳态无修改或迭代超过10次
while loopFlag==True and loopCounter<10:
    loopCounter+=1
    loopFlag=False
    # 计算每个样本点到各聚类中心的距离，确定与之最近的聚类中心,并划入该聚
    changeCounter=0.0
    for i in range(0,pointsNum,1):
        # 确定最近邻聚类中心
        minDistance = np.inf
        minCenterIndex = -1
        for j in range(0,k,1):
            distance=calculateDistance(data[i,],centerPoints[j,])
            if distance < minDistance:
                minDistance = distance
                minCenterIndex = j
        # 划入该聚类中
        if pointsCluster[i, 0] != minCenterIndex:
            changeCounter+=1.0
            pointsCluster[i,]=np.mat([minCenterIndex,minDistance])
```

样本划分完毕后，更新聚类中心值（以各聚类样本的均值作为新的聚类中心值），依据修改量判断是否继续迭代。

迭代全部完成后，返回聚类中心矩阵和聚类标签矩阵。

```
    # 针对各聚类，重新计算聚类中心
    for i in range(0,k,1):
        clusterSum=np.mat(np.zeros((1,attributeNum)))
        clusterPointsNum=0
        for j in range(0,pointsNum,1):
            if pointsCluster[j,0]==i:
                clusterSum+=data[j,]
                clusterPointsNum+=1
        for j in range(0,attributeNum,1):
            if clusterPointsNum!=0:
                centerPoints[i,j]=clusterSum[0,j]/clusterPointsNum
    if changeCounter/pointsNum>0.02:
        loopFlag=True
    print("loop",loopCounter,"finished")
print("cluster loop all finished!")
return [centerPoints,pointsCluster]
```

三、运行结果与图像可视化

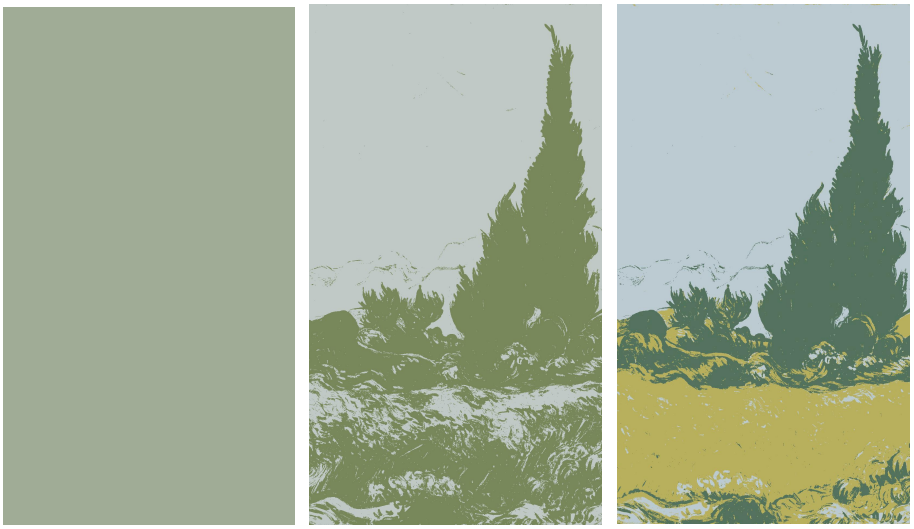
执行 main.py 文件，控制台信息如下，各次迭代开始、结束、每轮迭代完成

均有提示。本次作业采用的图片是梵高著名的画作《麦田里的丝柏树》（Wheat Field with Cypresses, Vincent Willem van Gogh, 1889）。

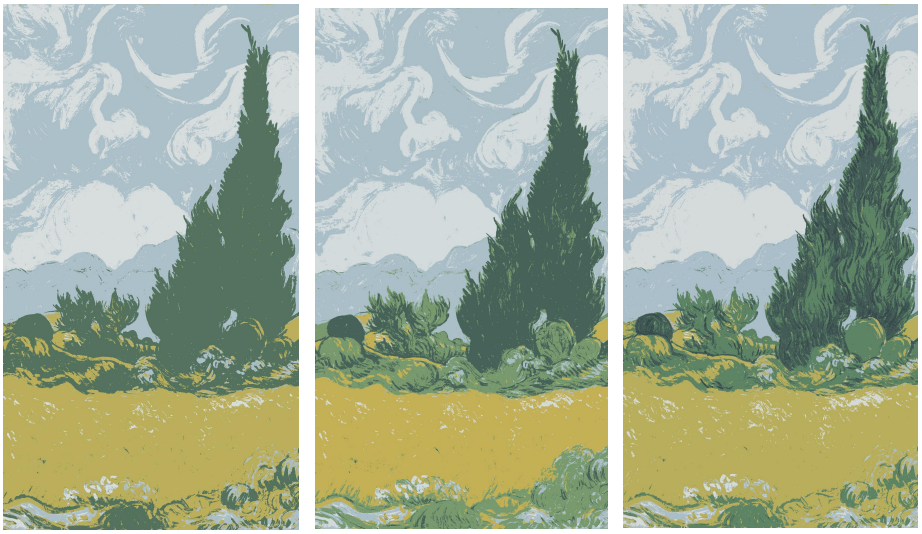
```
Run:  main
G:\python.exe D:/2023春/机器学习/作业1/code/main.py
start cluster loop...
loop 1 finished
cluster loop all finished!
start cluster loop...
loop 1 finished
loop 2 finished
loop 3 finished
loop 4 finished
loop 5 finished
cluster loop all finished!
start cluster loop...
loop 1 finished
loop 2 finished
loop 3 finished
```

画作原图和取 k 从 1 至 6 的运行结果图如下所示。可见此聚类算法取得了较好的图像分割效果。原图和结果图可视化存储在 code 文件夹中。

（原图）



（从左至右，k=1~3）



（从左至右，k=4~6）