**实验5 Python数值计算与可视化**

**学号：** 2020281024 **姓名： 陈志鸿 日期：2021-11-22**

* **实验正式报告请交纸质版。截止日期：2021年11月29日**
* **请注意报告版面整洁，截图清晰，内容组织逻辑清晰，易于阅读**
* **代码请附在报告中，并打包发邮件。代码要能复现报告中的结果。**

**目的和要求：**

（1）运用Numpy数值分析，解决实际问题

（2）运用matplotlib 进行数据可视化

**一、基本题**

**使用Python Numpy或者 matplotlib回答下面的问题：**

1. 生成一个长度为10的随机向量，并排序

1. **import** numpy as np
2. x = np.random.randint(0, 100, size = 10)
3. x = np.sort(x)



2. 生成一个长度为10的随机向量，并把最大值替换为0

1. x = np.random.randint(0, 100, size = 10)
2. x[np.argmax(x)] = 0



3. 对于一个给定的向量，如何找到与给定数值a 最近的元素？

1. x = np.random.randint(0, 100, 10)
2. val = 10
3. **print**(x, x[np.argmin(np.abs(x - val))])

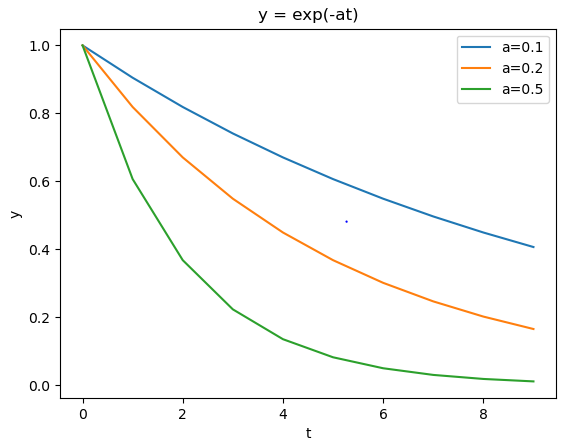


4. 创建一个 10x10 的随机数组并找到它的全局最大值、全局最小值，每一行的最小值？

1. np.min(x), np.max(x), np.min(x, axis = 1)

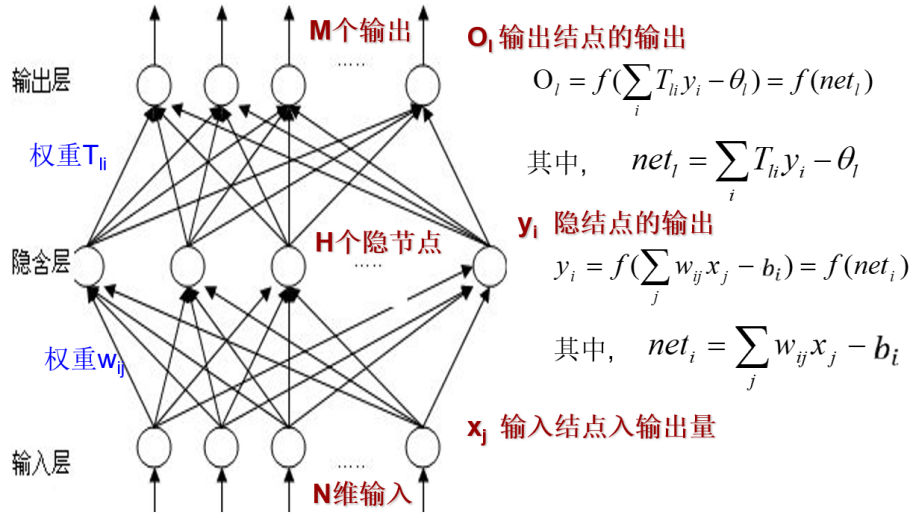
5. 有一组测量数据满足y = exp(-at) ，t的变化范围为0~10，用不同的线型和标记点画出a=0.1、a=0.2和a=0.5三种情况下的曲线。注意设置好title, xlabel, ylable, legend 等。

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
4. t = np.arange(10)
5. a = [0.1, 0.2, 0.5]
6. **for** i **in** range(3) :
7. plt.plot(t, np.exp(-a[i] \* t))
8. plt.title('y = exp(-at)'), plt.xlabel('t'), plt.ylabel('y'),plt.legend(['a=0.1', 'a=0.2', 'a=0.5'])
9. plt.show()



**二、向量化编程**

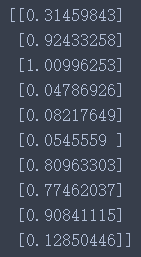
**BP神经网络推导**



如图所示为包含一个隐含层的神经网络。图右侧部分为前向推导，即给定模型参数 和输入X，可以求出模型的输出O。请写出前向过程函数：output = forward()，并给出一个测试案例及其结果。 图中(N,H,M)定义了模型的架构，决定参数的维度。

**注意：**请尽量使用numpy 向量/矩阵操作，少使用循环。

1. **import** numpy as np
3. **def** forward(X, N, H, M) :
4. W = np.random.randn(H, N)
5. b = np.full((H, 1), 0.01)
6. y = 1 / (1 + np.exp(np.dot(W, X)) - b)
8. T = np.random.randn(M, H)
9. sigma = np.full((M, 1), 0.01)
10. O = 1 / (1 + np.exp(np.dot(T, y)) - sigma)
12. **return** O
14. X = np.random.randn(100, 1)
15. output = forward(X, 100, 50, 10)
16. **print**(output)

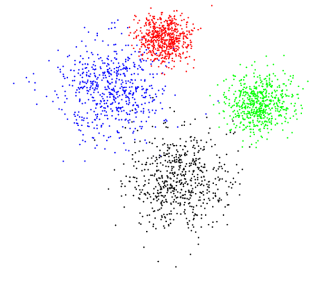


**三、综合题**

**Kmeans 聚类算法-续**

使用Numpy编写代码，不要使用其他工具包，勿直接抄袭网上代码！

聚类就是根据数据之间的相似度将数据集划分为多个类别或组，使类别内的数据相似度较大而类别间的数据相似度较小。如下图所示，左边是原始数据，右边是聚类之后的效果，不同的颜色代表不同的类别。



**实验数据：**文件Lab5.dat 中含有2400个二维空间的点坐标XY。

**Kmeans 算法简要**

请参考相关书籍或网络，了解kmeans 算法。大致的步骤如下：

1. 设置初始类别中心和类别数
2. 根据类别中心对全部数据进行类别划分：每个点分到离自己距离最小的那个类
3. 重新计算当前类别划分下每个类的中心：例如可以取每个类别里所有的点的平均值作为新的中心。如何求多个点的平均值？ 分别计算X坐标的平均值，y坐标的平均值，从而得到新的点。**注意：**类的中心可以不是真实的点，虚拟的点也不影响。
4. 在新的类别中心下继续进行类别划分;
5. 如果连续两次的类别划分结果不变则停止算法; 否则循环2～5。例如当类的中心不再变化时，跳出循环。

**影响Kmeans算法的可能因素：**

* **如何选择距离的定义：**对数据点进行类别划分时，需要计算点到点之间的距离。距离有很多种。 例如欧式距离，, 其中和是两个不同的点; L1范数距离。
* **如何确定合理的K值？**
* **如何选择K个初始类簇的中心点？**

**1. 实现函数 cidx, ctrs = kmeans(X, K),** 其中

* 输入X是的矩阵，每行一个点，K是类别的个数，
* 输出ctrs 是类的中心坐标，对应的size应该为：。cidx 是各个点的类别信息，表示每个点属于哪一类，其size 为, 例如cidx(0) = 2，表示第一个点属于第二个类。
* 函数写好后，测试当K = 2, 3, 4的效果，并用散点图画出分类的效果。例如，下图显示了当K=4的效果。

请查阅matplotlib.pyplot.scatter散点图的画法。

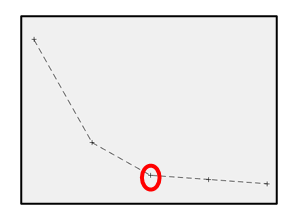
**2. 确定最优的参数K：手肘法**

参考：<https://blog.csdn.net/qq_15738501/article/details/79036255>

对每一个K值，计算分类的SSE(sum of the squared errors，误差平方和)。

其中N是点的个数，是第个点，是对应的中心。

编写函数函数y = calSSE(X, cidx) 计算聚类效果的SSE，其中X是待聚类的数据，cidx 是kmeans 函数返回的聚类结果， 画出当类别数K分别为2,3,4,5,6等值时的SSE， 肉眼能否看出最佳的K值？类似下图的效果？



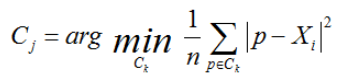
**以上步骤已经在Lab4中完成**

**本次试验的任务是实现下面的步骤3和4.**

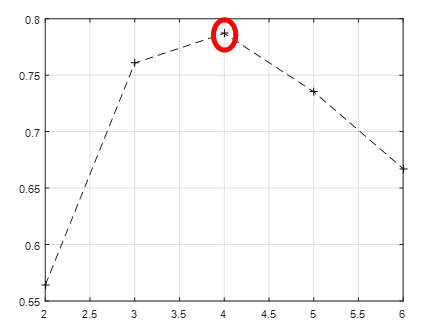
**3. 轮廓系数法确定最佳K值**

采用轮廓系数法确定最佳的K值，即计算分类的Silhouette Coefficient。手肘法需要目测曲线的拐点，而本方法可以自动的确定最佳的K值。

参考：<https://blog.csdn.net/qq_15738501/article/details/79036255>

某个样本点的轮廓系数定义如下：。其中是与同簇的其他样本的平均距离，称为凝聚度。是与其他最近簇中所有样本的平均距离，称为分离度。而最近簇的定义是

其中是某个簇中的样本。简单点讲，就是用到某个簇所有样本平均距离作为衡量该点到该簇的距离后，选择离最近的一个簇作为最近簇。求出所有样本的轮廓系数后再求平均值就得到了平均轮廓系数。平均轮廓系数的取值范围为[-1, 1]，且簇内样本的距离越近，簇间样本距离越远，**平均轮廓系数越大，聚类效果越好, 平均轮廓系数最大的K便是最佳聚类数**。



编写函数 cal\_Silhouette\_Coeff() 计算分类的轮廓系数，并画出K=2,3,4,5,6时的曲线，如上图所示。

**4. 改善初始类中心点的选取**

参考：<https://blog.csdn.net/zhihaoma/article/details/48649489>

前面的kmeans 算法是采用随机选择K个点作为初始的类簇中心点。现在采用新的方法，即选择彼此距离尽可能远的K个点。该方法的要点是：首先随机选择一个点作为第一个初始类簇中心点，然后选择距离该点最远的那个点作为第二个初始类簇中心点，然后再选择距离前两个点的最近距离最大的点作为第三个初始类簇的中心点，以此类推，直至选出K个初始类簇中心点。【详见下面的说明】

【本算法说明】

如图所示，当前已经有两个中心C1和C2，如何在其余的点中选取第三个中心呢？ 以A，B为例，选A还是B？

…….

A

B

C1

C2

设A到C1, C2的距离分别为, B到C1, C2的距离分别为。

我们的目的是要选择离红色点最远的点，那么我们可以定义一个点到红色点集合的距离为该点到所有红色点的最短路径，即。 于是，A到当前中心点集的距离为, 同理，B到中心点集的距离为。

据此，如果， 选在A作为第三个中心，否则选B。用同样的方法，选取后面的中心点。

编写函数 centers = get\_farthest\_k\_center(X, K)，其中X是数据，K是中心数目，返回K个中心。

1. **import** numpy as np
2. **import** matplotlib.pyplot as plt
4. data = np.loadtxt('Lab5.dat')
6. **def** CalcDistance(x, y) :
7. **return** np.sum(np.square(x - y))
9. **def** get\_farthest\_k\_center(X, K) :
10. row, col = X.shape
11. centroids = np.zeros((K, col))
12. #第一个点
13. index = int(np.random.uniform(0, row))
14. centroids[0, : ] = X[index, : ]
15. #第二个点
16. distance = np.zeros(row)
17. **for** i **in** range(row) :
18. distance[i] = CalcDistance(X[i, : ], centroids[0, : ])
19. max\_index = np.argmax(distance)
20. centroids[1, : ] = X[max\_index, : ]
21. #第三个点及以后，
22. #然后再选择距离前两个点的最短距离最大的那个点作为第三个初始类簇的中心点
23. j = 1
24. **while** j <= K - 2:
25. distance = np.zeros(row)
26. **for** i **in** range(row) :
27. distance1 = CalcDistance(centroids[j - 1, : ], X[i, : ])
28. distance2 = CalcDistance(centroids[j, : ], X[i, : ])
29. distance[i] = min(distance1, distance2)
30. max\_index = np.argmax(distance)
31. centroids[j + 1, : ] = X[max\_index, : ]
32. j += 1
33. **return** centroids
35. **def** cal\_Silhouette\_Coeff(X, cidx, K) :
36. c, b = [], [0] \* (K - 1)
37. distance\_a, distance\_b = 0, np.zeros(K - 1)
38. **for** i **in** range(K) :
39. subX = X[cidx == 1 + i]#获取每个簇类
40. length = len(subX)
41. **for** j **in** range(length) :
42. distance\_a = CalcDistance(subX[j, : ], subX[ : , : ]) / (length)#凝聚度
43. **for** k **in** range(K) :#求最近簇
44. **if** k < i :
45. b[k] = CalcDistance(subX[j, : ], X[cidx == k + 1, : ]) / len(X[cidx == k + 1])
46. **elif** k > i :
47. b[k - 1] = CalcDistance(subX[j, : ], X[cidx == k + 1, : ]) / len(X[cidx == k + 1])
48. distance\_b = np.min(b)#分离度
49. c.append((distance\_b - distance\_a) / max(distance\_a, distance\_b))
51. **return** sum(c) / len(c)
53. **def** calSSE(X, cidx, ctrs) :
54. SSE = 0
55. **for** i, ctr **in** enumerate(ctrs) :
56. SSE += np.sum(np.square(X[np.where(cidx == i + 1)] - ctr))
58. **return** SSE / X.shape[0]
60. **def** kmeans(X, K) :
61. center\_point = get\_farthest\_k\_center(X, K)#初始化簇点
62. cluter = np.zeros(X.shape[0])#建立簇类初始化为0
63. item = 5
64. **while** item > 0:#迭代
65. **for** i **in** range(X.shape[0]) :#计算每一组数据与每个簇心的欧氏距离，距离最小者记为此组数据为所标类别
66. distance = center\_point
67. distance = np.sum(np.square(distance - X[i]), axis = 1)#注意x， y都计算所以要求和，注意求和维度
68. cluter[i] = np.argmin(distance) + 1#最小值的下标
70. New\_center\_point = np.zeros((K, 2), dtype = np.float64)
72. **for** i **in** range(K) :#更新簇点，取每一簇类的平均值作为新簇心
73. subX = X[np.where(cluter == i + 1)]
74. length = len(subX)
75. **if** length > 0:#防止分母出现0
76. New\_center\_point[i] = np.mean(subX, axis = 0)
77. **else** :
78. New\_center\_point[i] = center\_point[i]
80. **if** (New\_center\_point == center\_point).all() :#当新簇点与之前的簇心近似相等时退出迭代
81. **break**
83. center\_point = New\_center\_point
84. item -= 1
85. **return** cluter, center\_point#返回每一组数据所对应的簇类和簇心
87. SSE = []
88. C = []
89. mark = [ 'r', 'c', 'y', 'k', 'm', 'g']
91. plt.ion()
92. **for** K **in** range(2, 7) :
93. cidx, ctrs = kmeans(data, K)
94. **print**(f'K为{K}时的簇心 : \n {ctrs}')
95. SSE.append(calSSE(data, cidx, ctrs))#手肘法求最好分类的K值
96. C.append(cal\_Silhouette\_Coeff(data, cidx, K))#求轮廓系数
98. plt.subplot(2, 3, K - 1)
99. **for** i **in** range(K) :
100. plt.scatter(data[np.where(cidx == i + 1), 0], data[np.where(cidx == i + 1), 1], marker = '.', color = mark[i])#作图
101. plt.scatter(ctrs[ : , 0], ctrs[ : , 1], marker = '\*', color = 'g')#做出簇心
102. plt.title(f'K is {K}')
103. plt.tight\_layout()
104. plt.xticks([]), plt.yticks([])

107. plt.figure(), plt.plot(list(range(2, 7)), SSE, '+--')
108. plt.figure(), plt.plot(list(range(2, 7)), C, '+--'), plt.grid(), plt.xticks([2, 3, 4, 5, 6]), plt.xlim(2, 6)
109. plt.ioff()
110. plt.show()

