## Fisher线性判别

### 环境

* pycharm专业版2019.3.3
* python3.7.4
* 外部库：numpy=1.16.5；sklearn=0.21.3；matplotlib=3.3.1；openpyxl=3.0.0

### 原理简介

1. 将高纬度问题降维，即假设数据存在于n维空间中，在数学上，通过投影使数据到一条直线上。然后根据投影点在直线上的分布对原始点进行分类。
2. 怎么找到合适的直线方向，能使不同类别数据映射到该条直线上易于区分，这就是Fisher线性判别要解决的问题。

### 一些计算公式

#### 在n维X空间

* 各类样本均值向量:

* 各类类内离散度矩阵:

* 总类内离散度矩阵:

* 样本类间离散度矩阵:

#### 在1维Y空间

* 各类样本均值:

* 各类内离散度:

* 最佳投影方向:
* 决策点:

#### 分类评价指标

* 总体分类精度OA：
* 类别分类精度AA:
* kappa系数:
* pe:

### 数据标准化

由于两种数据集中特征的量纲都一样，所以为了消除量纲，将每个特征的数值除以该特征中的最大值即可，这样将所有特征数值映射到了区间[0,1]之间

### 划分训练集和测试集

利用sklearn包中的train*test*split函数进行训练集和测试集的划分，其中训练集占比40%，测试集占比60%。

其中train*test*split函数的一般形式如下：

X\_train,X\_test, y\_train, y\_test =train\_test\_split(train\_data,train\_target,test\_size=0.4, random\_state=0)

参数解释：

* **train\_data：**所要划分的样本特征集
* **train\_target：**所要划分的样本结果集
* **test\_size：**测试样本占比，如果是整数的话就是样本的数量
* **random\_state：**是随机数的种子
* **随机种子：**其实就是该组随机数的编号，在需要重复试验的时候，保证得到一组一样的随机数

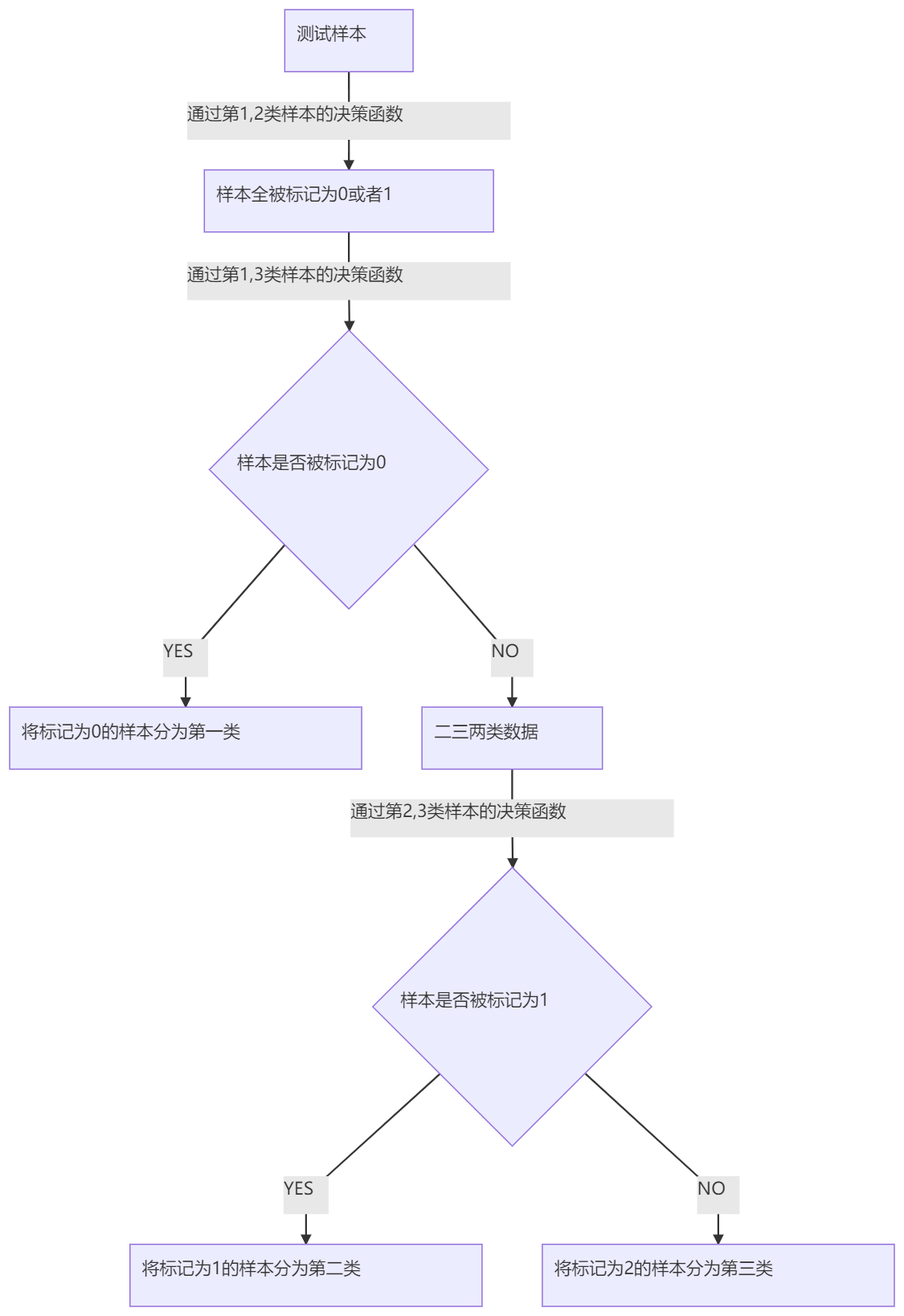
通过该函数，只要每次传入不同的随机种子，就可以得到不同的训练集、测试集

### 分类策略

1. sonar数据集

* sonar数据集为一个二分类问题，只需要计算出最优投影方向，投影后计算出决策点，再进行分类即可

1. iris数据集

* iris数据集为一个三分类问题，可以将其转化为三个二分类问题进行分类，具体分类思路如下：
* 

### 分类结果

1. sonar数据集

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| * 试验次数 | * 整体分类精度OA | * kappa系数 | * 第一类类别分类精度 | * 第二类类别分类精度 |
| * 1 | * 0.603 | * 0.213 | * 0.678 | * 0.537 |
| * 2 | * 0.690 | * 0.370 | * 0.559 | * 0.806 |
| * 3 | * 0.690 | * 0.380 | * 0.695 | * 0.687 |
| * 4 | * 0.587 | * 0.171 | * 0.559 | * 0.612 |
| * 5 | * 0.690 | * 0.385 | * 0.763 | * 0.627 |
| * 6 | * 0.698 | * 0.389 | * 0.610 | * 0.776 |
| * 7 | * 0.675 | * 0.355 | * 0.763 | * 0.597 |
| * 8 | * 0.683 | * 0.371 | * 0.780 | * 0.597 |
| * 9 | * 0.706 | * 0.410 | * 0.678 | * 0.731 |
| * 10 | * 0.690 | * 0.383 | * 0.729 | * 0.657 |
| * 11 | * 0.643 | * 0.284 | * 0.627 | * 0.657 |
| * 12 | * 0.730 | * 0.467 | * 0.847 | * 0.627 |
| * 13 | * 0.643 | * 0.279 | * 0.576 | * 0.701 |
| * 14 | * 0.683 | * 0.353 | * 0.542 | * 0.806 |
| * 15 | * 0.675 | * 0.351 | * 0.712 | * 0.642 |
| * 16 | * 0.603 | * 0.198 | * 0.525 | * 0.672 |
| * 17 | * 0.794 | * 0.589 | * 0.847 | * 0.746 |
| * 18 | * 0.611 | * 0.223 | * 0.627 | * 0.597 |
| * 19 | * 0.627 | * 0.246 | * 0.542 | * 0.701 |
| * 20 | * 0.778 | * 0.551 | * 0.712 | * 0.836 |
| * 21 | * 0.675 | * 0.358 | * 0.797 | * 0.567 |
| * 22 | * 0.683 | * 0.365 | * 0.695 | * 0.672 |
| * 23 | * 0.754 | * 0.500 | * 0.644 | * 0.851 |
| * 24 | * 0.738 | * 0.475 | * 0.729 | * 0.746 |
| * 25 | * 0.730 | * 0.460 | * 0.746 | * 0.716 |
| * 26 | * 0.667 | * 0.335 | * 0.695 | * 0.642 |
| * 27 | * 0.667 | * 0.333 | * 0.678 | * 0.657 |
| * 28 | * 0.706 | * 0.417 | * 0.780 | * 0.642 |
| * 29 | * 0.754 | * 0.508 | * 0.780 | * 0.731 |
| * 30 | * 0.730 | * 0.462 | * 0.780 | * 0.687 |
| * **平均值** | * **0.687** | * **0.373** | * **0.690** | * **0.684** |

1. iris数据集

|  |  |  |  |
| --- | --- | --- | --- |
| * 实验次数 | * 整体分类精度OA | * kappa系数 | * 类别分类精度AA（依次为第1，2，3类） |
| * 1 | * 0.900 | * 0.850 | * 1.00 0.767 0.933 |
| * 2 | * 0.911 | * 0.867 | * 1.00 0.800 0.933 |
| * 3 | * 0.833 | * 0.750 | * 1.00 0.800 0.700 |
| * 4 | * 0.956 | * 0.933 | * 1.00 0.900 0.967 |
| * 5 | * 0.922 | * 0.883 | * 1.00 0.900 0.867 |
| * 6 | * 0.967 | * 0.950 | * 1.00 0.900 1.000 |
| * 7 | * 0.900 | * 0.850 | * 1.00 0.800 0.900 |
| * 8 | * 0.944 | * 0.917 | * 1.00 0.900 0.933 |
| * 9 | * 0.933 | * 0.900 | * 1.00 0.967 0.833 |
| * 10 | * 0.956 | * 0.933 | * 1.00 0.933 0.933 |
| * 11 | * 0.900 | * 0.850 | * 1.00 0.833 0.867 |
| * 12 | * 0.822 | * 0.733 | * 1.00 0.667 0.800 |
| * 13 | * 0.833 | * 0.75 | * 1.00 0.800 0.700 |
| * 14 | * 0.967 | * 0.950 | * 1.00 0.933 0.967 |
| * 15 | * 0.878 | * 0.817 | * 1.00 0.633 1.000 |
| * 16 | * 0.922 | * 0.883 | * 1.00 0.933 0.833 |
| * 17 | * 0.644 | * 0.467 | * 1.00 0.433 0.500 |
| * 18 | * 0.900 | * 0.850 | * 1.00 0.833 0.867 |
| * 19 | * 0.900 | * 0.850 | * 1.00 0.867 0.833 |
| * 20 | * 0.889 | * 0.833 | * 1.00 0.867 0.800 |
| * 21 | * 0.900 | * 0.850 | * 1.00 0.900 0.800 |
| * 22 | * 0.900 | * 0.850 | * 1.00 0.833 0.867 |
| * 23 | * 0.911 | * 0.867 | * 1.00 0.967 0.767 |
| * 24 | * 0.944 | * 0.917 | * 1.00 0.833 1.000 |
| * 25 | * 0.944 | * 0.917 | * 1.00 0.967 0.867 |
| * 26 | * 0.922 | * 0.883 | * 0.97 0.867 0.933 |
| * 27 | * 0.956 | * 0.933 | * 1.00 0.900 0.967 |
| * 28 | * 0.922 | * 0.883 | * 1.00 0.900 0.867 |
| * 29 | * 0.956 | * 0.933 | * 1.00 0.933 0.933 |
| * 30 | * 0.878 | * 0.817 | * 1.00 0.767 0.867 |
| * **平均值** | * **0.904** | * **0.856** | * **0.999 0.844 0.868** |

### 结果分析

1. sonar

* 从结果可以看出，sonar数据集的分类精度并不理想，只有68.7%，kappa系数也只有0.373，这说明分类效果较为一般，个人认为分类效果一般的原因如下：
* sonar数据集的维度较高，足足有60维，而我们直接将其降到了一维，降维的过程中避免不了有效信息的损失，有可能是因为有效信息损失过多，导致分类效果不理想。
* 可行的改进方法是不降成一维，降成2维或者稍低一点的维度进行分类。

1. iris数据集

* 从结果可以看出，三分类的准确率达到了90.4%，kappa系数达到了0.856，说明分类效果很好，与实际情况几乎完全一致。
* 类别分类精度中第一类的精度最高，二三类稍低一点，可以看出：第一类鸢尾花在四个特征上与另外两类有较为明显的差别，很容易跟另外两类区分开来；而第二三类可能是在四个特征上的差别没有特别明显，所以分类精度会有所下降。

### 代码展示

1. 鸢尾花数据集分类（数据来源于sklearn内部封装的数据集）

|  |
| --- |
| 1. # -\*- coding: utf-8 -\*- 2. **from** sklearn**.**datasets **import** load\_iris 3. **from** sklearn**.**model\_selection **import** train\_test\_split 4. **import** numpy **as** np 5. **from** matplotlib **import** pyplot **as** plt 6. **import** random 7. # 计算并返回均值向量 8. **def** junzhi**(**iris**):** 9. a **=** np**.**zeros**([**4**,** 1**])** 10. a**[**0**]** **=** np**.**mean**(**iris**[:,** 0**])** 11. a**[**1**]** **=** np**.**mean**(**iris**[:,** 1**])** 12. a**[**2**]** **=** np**.**mean**(**iris**[:,** 2**])** 13. a**[**3**]** **=** np**.**mean**(**iris**[:,** 3**])** 14. **return** a 15. # 计算类内离散度矩阵S\_i 16. **def** S\_i**(**iris**):** 17. a **=** junzhi**(**iris**)** 18. b **=** np**.**zeros**([**iris**.**shape**[**1**],** iris**.**shape**[**1**]])** 19. **for** i **in** range**(**iris**.**shape**[**0**]):** 20. b **=** b **+** np**.**matmul**((**iris**[**i**,** **:].**T **-** a**),** **(**iris**[**i**,** **:].**T **-** a**).**T**)** 21. **return** b 22. # 计算类间离散度矩阵S\_b 23. **def** S\_b**(**iris1**,** iris2**):** 24. b\_1 **=** S\_i**(**iris1**)** 25. b\_2 **=** S\_i**(**iris2**)** 26. c **=** b\_1 **+** b\_2 27. **return** c 28. # 划分训练集、测试集的函数 29. **def** train\_test**(**iris**,** target**,** num**):** 30. train\_iris**,** test\_iris**,** train\_target**,** test\_target **=**\ 31. train\_test\_split**(**iris**,** target**,** test\_size**=**0.6**,** random\_state**=**num**,** shuffle**=True)** 32. **return** **{**'train\_iris'**:** train\_iris**,** 'test\_iris'**:** test\_iris**,** 'train\_target'**:** train\_target**,** 'test\_target'**:** test\_target**}** 33. # 计算出最优的投影方向并计算出决策点 34. **def** best\_w**(**iris1**,** iris2**,** target1**,** target2**,** num**):** 35. train\_iris1 **=** train\_test**(**iris1**,** target1**,** num**)[**'train\_iris'**]** 36. train\_iris2 **=** train\_test**(**iris2**,** target2**,** num**)[**'train\_iris'**]** 37. s\_0 **=** S\_b**(**train\_iris1**,** train\_iris2**)** 38. best\_w **=** np**.**matmul**(**np**.**linalg**.**inv**(**s\_0**),** junzhi**(**train\_iris1**)** **-** junzhi**(**train\_iris2**))** 39. y\_0 **=** 0.5**\***np**.**mean**(**np**.**matmul**(**train\_iris1**,** best\_w**))** **+** 0.5**\***np**.**mean**(**np**.**matmul**(**train\_iris2**,** best\_w**))** 40. # print(best\_w) 41. **return** best\_w**,** y\_0 42. # 对测试样本进行分类并计算相关评价指标 43. **def** classify**(**iris1**,**iris2**,**iris3**,**target1**,**target2**,**target3**,**num**):** 44. # print(num) 45. ## 训练集得到的最佳方向和决策点 46. w\_best12**,**y0\_12**=**best\_w**(**iris1**,** iris2**,** target1**,** target2**,** num**)** 47. w\_best13**,**y0\_13**=**best\_w**(**iris1**,** iris3**,** target1**,** target3**,** num**)** 48. w\_best23**,**y0\_23**=**best\_w**(**iris2**,** iris3**,** target2**,** target3**,** num**)** 49. ## 测试集 50. test1**=**train\_test**(**iris1**,**target1**,**num**)[**'test\_iris'**]** 51. test2**=**train\_test**(**iris2**,**target2**,**num**)[**'test\_iris'**]** 52. test3**=**train\_test**(**iris3**,**target3**,**num**)[**'test\_iris'**]** 53. test**=**np**.**vstack**((**test1**,**test2**,**test3**))** 54. ## 当前测试集对应的标签 55. test\_target1**=**train\_test**(**iris1**,**target1**,**num**)[**'test\_target'**]** 56. test\_target2**=**train\_test**(**iris2**,**target2**,**num**)[**'test\_target'**]** 57. test\_target3**=**train\_test**(**iris3**,**target3**,**num**)[**'test\_target'**]** 58. test\_target**=**np**.**hstack**((**test\_target1**,**test\_target2**,**test\_target3**))** 59. # print(test\_target) 60. ## 存放预测得到的标签 61. predict\_target**=**np**.**zeros\_like**(**test\_target**)** 62. ## 先通过第一二类决策函数 63. y**=**np**.**matmul**(**test**,**w\_best12**)** 64. **for** i **in** range**(**len**(**test**)):** 65. **if** y**[**i**]>**y0\_12 **or** y**[**i**]==**y0\_12**:** 66. predict\_target**[**i**]=**0 67. **else:** 68. predict\_target**[**i**]=**1 69. ## 再通过第一三类决策函数 70. y**=**np**.**matmul**(**test**,**w\_best13**)** 71. **for** i **in** range**(**len**(**test**)):** 72. **if** y**[**i**]>**y0\_13 **or** y**[**i**]==**y0\_13**:** 73. predict\_target**[**i**]=**0 74. **else:** 75. predict\_target**[**i**]=**2 76. ## 剩余的通过第二三类决策函数 77. y**=**np**.**matmul**(**test**,**w\_best23**)** 78. **for** i **in** range**(**len**(**test**)):** 79. **if** predict\_target**[**i**]!=**0**:** 80. **if** y**[**i**]>**y0\_23 **or** y**[**i**]==**y0\_23**:** 81. predict\_target**[**i**]=**1 82. **else:** 83. predict\_target**[**i**]=**2 84. # print(predict\_target) 85. ## 计算OA、AA、kappa系数 86. ### 记录三类样本分类正确的数量 87. num\_1**=**0 88. num\_2**=**0 89. num\_3**=**0 90. ### 记录三类样本实际分类的数量 91. real\_num\_1**=**0 92. real\_num\_2**=**0 93. real\_num\_3**=**0 94. **for** i **in** range**(**len**(**test\_target**)):** 95. ## 统计分类正确的数量 96. **if** i**<**len**(**test\_target**)/**3**:** 97. **if** predict\_target**[**i**]==**test\_target**[**i**]:** 98. num\_1**=**num\_1**+**1 99. **elif** i**<**2**\*(**len**(**test\_target**))/**3**:** 100. **if** predict\_target**[**i**]==**test\_target**[**i**]:** 101. num\_2**=**num\_2**+**1 102. **else:** 103. **if** predict\_target**[**i**]==**test\_target**[**i**]:** 104. num\_3**=**num\_3**+**1 105. ## 统计实际的分类数量 106. **if** predict\_target**[**i**]==**0**:** 107. real\_num\_1**=**real\_num\_1**+**1 108. **elif** predict\_target**[**i**]==**1**:** 109. real\_num\_2**=**real\_num\_2**+**1 110. **else:** 111. real\_num\_3**=**real\_num\_3**+**1 112. # print(num\_1) 113. ### 计算相关指标 114. OA**=(**num\_1**+**num\_2**+**num\_3**)/**len**(**test\_target**)** 115. AA\_1**=**num\_1**\***3**/**len**(**test\_target**)** 116. AA\_2**=**num\_2**\***3**/**len**(**test\_target**)** 117. AA\_3**=**num\_3**\***3**/**len**(**test\_target**)** 118. pe**=(**real\_num\_1**\***len**(**test\_target1**)+**real\_num\_2**\***len**(**test\_target2**)**\ 119. **+**real\_num\_3**\***len**(**test\_target3**))/**np**.**square**(**len**(**test\_target**))** 120. kappa**=(**OA**-**pe**)/(**1**-**pe**)** 121. **return** OA**,[**AA\_1**,**AA\_2**,**AA\_3**],**kappa 122. **if** \_\_name\_\_ **==** "\_\_main\_\_"**:** 123. # 导入数据并去量纲 124. data **=** load\_iris**()** 125. iris1 **=** data**.**data**[**0**:**50**,** 0**:**4**]** 126. iris2 **=** data**.**data**[**50**:**100**,** 0**:**4**]** 127. iris3 **=** data**.**data**[**100**:**150**,** 0**:**4**]** 129. max\_feature**=**np**.**array**([**np**.**max**(**data**.**data**[:,**0**]),**np**.**max**(**data**.**data**[:,**1**]),**\ 130. np**.**max**(**data**.**data**[:,**2**]),**np**.**max**(**data**.**data**[:,**3**])])** 131. iris1**=**iris1**/**max\_feature 132. iris2**=**iris2**/**max\_feature 133. iris3**=**iris3**/**max\_feature 134. # 导入标签 135. target1 **=** data**.**target**[**0**:**50**].**T 136. target2 **=** data**.**target**[**50**:**100**].**T 137. target3 **=** data**.**target**[**100**:**150**].**T 138. # 存储相关的指标值 139. OAs**=**np**.**zeros**([**30**,**1**])** 140. AAs**=**np**.**zeros**([**30**,**3**])** 141. kappas**=**np**.**zeros**([**30**,**1**])** 142. #随机给出30个随机种子，用于train\_test函数 143. # nums=[703,5205,8248,4998,1027,8528,7063,6513,793,2805,1524,8985,3939,9000\ 144. # ,3796,3178,628,9359,582,265,5920,8866,7960,5090,5481,4928,526,8763,5333,6596] 145. nums**=**random**.**sample**(**range**(**0**,**10000**),**100**)** 146. j**=**0**;**k**=**0 147. **while** j**<**30**:** 148. **try:** 149. OAs**[**j**],**AAs**[**j**,],**kappas**[**j**]=**classify**(**iris1**,**iris2**,**iris3**,**target1**,**target2**,**target3**,**nums**[**k**])** 150. j**=**j**+**1 151. k**=**k**+**1 152. **except:** 153. k**+=**1 154. # print(OAs) 155. temp**=**0 156. **for** i **in** range**(**len**(**OAs**)):** 157. **if** OAs**[**i**]!=**0**:** 158. temp**=**temp**+**1 159. **print(**AAs**)** 160. **print(**OAs**)** 161. **print(**kappas**)** 162. **print(**"AA值分别为：\n{:.3f}\n{:.3f}\n{:.3f}"**.**format**(**np**.**sum**(**AAs**[:,**0**])/**temp**,**np**.**sum**(**AAs**[:,**1**]/**temp**),**np**.**sum**(**AAs**[:,**2**])/**temp**))** 163. **print(**"OA值为：\n{:.3f}"**.**format**(**np**.**sum**(**OAs**)/**temp**))** 164. **print(**"kappa值为：\n{:.3f}"**.**format**(**np**.**sum**(**kappas**)/**temp**))** |

1. sonar数据集分类（数据来源见附件sonar.xlsx）

|  |
| --- |
| 1. # -\*- coding: utf-8 -\*- 2. **from** sklearn**.**model\_selection **import** train\_test\_split 3. **import** numpy **as** np 4. **from** matplotlib **import** pyplot **as** plt 5. **import** random 6. **from** openpyxl **import** load\_workbook 7. # 计算并返回均值向量 8. **def** junzhi**(**sonar**):** 9. a **=** np**.**zeros**([**60**,** 1**])** 10. **for** i **in** range**(**60**):** 11. a**[**i**]=**np**.**mean**(**sonar**[:,**i**])** 12. **return** a 13. # 计算类内离散度矩阵S\_i 14. **def** S\_i**(**sonar**):** 15. a **=** junzhi**(**sonar**)** 16. b **=** np**.**zeros**([**sonar**.**shape**[**1**],** sonar**.**shape**[**1**]])** 17. **for** i **in** range**(**sonar**.**shape**[**0**]):** 18. b **=** b **+** np**.**matmul**((**sonar**[**i**,** **:].**T **-** a**),** **(**sonar**[**i**,** **:].**T **-** a**).**T**)** 19. **return** b 20. # 计算类间离散度矩阵S\_b 21. **def** S\_b**(**sonar1**,** sonar2**):** 22. b\_1 **=** S\_i**(**sonar1**)** 23. b\_2 **=** S\_i**(**sonar2**)** 24. c **=** b\_1 **+** b\_2 25. **return** c 26. # 划分训练集、测试集的函数 27. **def** train\_test**(**sonar**,** target**,** num**):** 28. train\_sonar**,** test\_sonar**,** train\_target**,** test\_target **=**\ 29. train\_test\_split**(**sonar**,** target**,** test\_size**=**0.6**,** random\_state**=**num**,** shuffle**=True)** 30. **return** **{**'train\_sonar'**:** train\_sonar**,** 'test\_sonar'**:** test\_sonar**,** 'train\_target'**:** train\_target**,** 'test\_target'**:** test\_target**}** 31. # 计算出最优的投影方向并计算出决策点 32. **def** best\_w**(**sonar1**,** sonar2**,** target1**,** target2**,** num**):** 33. train\_sonar1 **=** train\_test**(**sonar1**,** target1**,** num**)[**'train\_sonar'**]** 34. train\_sonar2 **=** train\_test**(**sonar2**,** target2**,** num**)[**'train\_sonar'**]** 35. s\_0 **=** S\_b**(**train\_sonar1**,** train\_sonar2**)** 36. best\_w **=** np**.**matmul**(**np**.**linalg**.**inv**(**s\_0**),** junzhi**(**train\_sonar1**)** **-** junzhi**(**train\_sonar2**))** 37. y\_0 **=** **(**58**/**124**)\***np**.**mean**(**np**.**matmul**(**train\_sonar1**,** best\_w**))** **+** **(**66**/**124**)\***np**.**mean**(**np**.**matmul**(**train\_sonar2**,** best\_w**))** 38. # print(best\_w) 39. **return** best\_w**,** y\_0 40. # 对测试样本进行分类并计算相关评价指标 41. **def** classify**(**sonar1**,**sonar2**,**target1**,**target2**,**num**):** 42. # print(num) 43. ## 训练集得到的最佳方向和决策点 44. w\_best12**,**y0\_12**=**best\_w**(**sonar1**,** sonar2**,** target1**,** target2**,** num**)** 45. ## 测试集 46. test1**=**train\_test**(**sonar1**,**target1**,**num**)[**'test\_sonar'**]** 47. test2**=**train\_test**(**sonar2**,**target2**,**num**)[**'test\_sonar'**]** 48. test**=**np**.**vstack**((**test1**,**test2**))** 49. ## 当前测试集对应的标签 50. test\_target1**=**train\_test**(**sonar1**,**target1**,**num**)[**'test\_target'**]** 51. test\_target2**=**train\_test**(**sonar2**,**target2**,**num**)[**'test\_target'**]** 52. # print(test\_target1.shape) 53. # print(test\_target2.shape) 54. test\_target**=**np**.**vstack**((**test\_target1**,**test\_target2**))** 55. # print(test\_target.shape) 56. # print(test\_target) 57. ## 存放预测得到的标签 58. predict\_target**=**np**.**zeros\_like**(**test\_target**)** 59. ## 通过决策函数 60. y**=**np**.**matmul**(**test**,**w\_best12**)** 61. **for** i **in** range**(**len**(**test**)):** 62. **if** y**[**i**]>**y0\_12 **or** y**[**i**]==**y0\_12**:** 63. predict\_target**[**i**]=**0 64. **else:** 65. predict\_target**[**i**]=**1 66. # print(predict\_target) 67. ## 计算OA、AA、kappa系数 68. ### 记录三类样本分类正确的数量 69. num\_1**=**0 70. num\_2**=**0 71. ### 记录三类样本实际分类的数量 72. real\_num\_1**=**0 73. real\_num\_2**=**0 74. **for** i **in** range**(**len**(**test\_target**)):** 75. ## 统计分类正确的数量 76. **if** i**<**len**(**test\_target1**):** 77. **if** predict\_target**[**i**]==**test\_target**[**i**]:** 78. num\_1**=**num\_1**+**1 79. **else:** 80. **if** predict\_target**[**i**]==**test\_target**[**i**]:** 81. num\_2**=**num\_2**+**1 82. ## 统计实际的分类数量 83. **if** predict\_target**[**i**]==**0**:** 84. real\_num\_1**=**real\_num\_1**+**1 85. **else:** 86. real\_num\_2**=**real\_num\_2**+**1 87. # print(num\_1) 88. ### 计算相关指标 89. OA**=(**num\_1**+**num\_2**+**20**)/**len**(**test\_target**)** 90. AA\_1**=(**10**+**num\_1**)/**len**(**test\_target1**)** 91. AA\_2**=(**10**+**num\_2**)/**len**(**test\_target2**)** 92. pe**=(**real\_num\_1**\***len**(**test\_target1**)+**real\_num\_2**\***\ 93. len**(**test\_target2**))/**np**.**square**(**len**(**test\_target**))** 94. kappa**=(**OA**-**pe**)/(**1**-**pe**)** 95. **return** OA**,[**AA\_1**,**AA\_2**],**kappa 96. **if** \_\_name\_\_ **==** "\_\_main\_\_"**:** 97. # 导入数据 98. workbook**=**load\_workbook**(**filename**=**'sonar.xlsx'**)** 99. # print(workbook.sheetnames) 100. sheet**=**workbook**[**'Sheet1'**]** 101. # 存储样本特征集 102. sonar**=**np**.**zeros**([**208**,**60**])** 103. # 存储标签 104. target**=**np**.**zeros**([**208**,**1**])** 105. **for** i **in** range**(**208**):** 106. **for** j **in** range**(**60**):** 107. sonar**[**i**,**j**]=**sheet**.**cell**(**row**=**i**+**1**,**column**=**j**+**1**).**value 108. # print(sonar.shape) 109. # print(sonar[0,59]) 110. **for** i **in** range**(**208**):** 111. **if** sheet**.**cell**(**row**=**i**+**1**,**column**=**61**).**value**==**'R'**:** 112. target**[**i**]=**0 113. **else:** 114. target**[**i**]=**1 115. # print(target[97]) 116. sonar1**=**sonar**[**0**:**97**,:]** 117. sonar2**=**sonar**[**97**:**208**,:]** 118. target1**=**target**[**0**:**97**,:]** 119. target2**=**target**[**97**:**208**,:]** 120. # 存储相关的指标值 121. OAs**=**np**.**zeros**([**30**,**1**])** 122. AAs**=**np**.**zeros**([**30**,**2**])** 123. kappas**=**np**.**zeros**([**30**,**1**])** 124. #随机给出30个随机种子，用于train\_test函数 125. # nums=[703,5205,8248,4998,1027,8528,7063,6513,793,2805,1524,8985,3939,9000\ 126. # ,3796,3178,628,9359,582,265,5920,8866,7960,5090,5481,4928,526,8763,5333,6596] 127. nums**=**random**.**sample**(**range**(**0**,**10000**),**100**)** 128. j**=**0**;**k**=**0 129. **while** j**<**30**:** 130. **try:** 131. OAs**[**j**],**AAs**[**j**,],**kappas**[**j**]=**classify**(**sonar1**,**sonar2**,**target1**,**target2**,**nums**[**k**])** 132. j**=**j**+**1 133. k**=**k**+**1 134. **except:** 135. k**+=**1 137. # OAs[j],AAs[j,],kappas[j]=classify(sonar1,sonar2,target1,target2,nums[k]) 138. temp**=**0 139. **for** i **in** range**(**len**(**OAs**)):** 140. **if** OAs**[**i**]!=**0**:** 141. temp**=**temp**+**1 142. **print(**OAs**)** 143. **print(**AAs**)** 144. **print(**kappas**)** 145. **print(**"AA值分别为：\n{:.3f}\n{:.3f}"**.**format**(**np**.**sum**(**AAs**[:,**0**])/**temp**,**np**.**sum**(**AAs**[:,**1**]/**temp**)))** 146. **print(**"OA值为：\n{:.3f}"**.**format**(**np**.**sum**(**OAs**)/**temp**))** 147. **print(**"kappa值为：\n{:.3f}"**.**format**(**np**.**sum**(**kappas**)/**temp**))** |