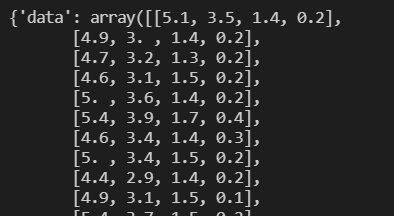
《机器学习》上机实践

031802326-苏炜斌

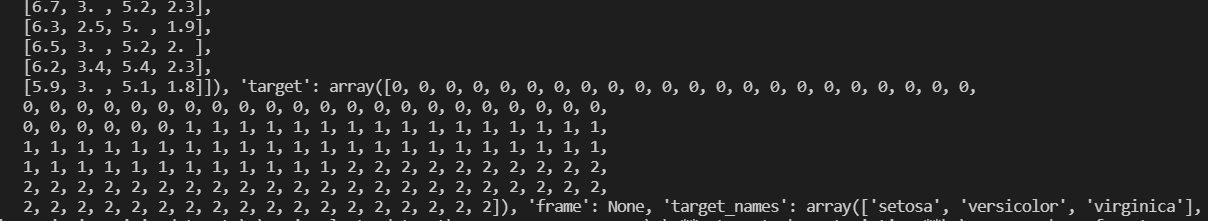
**实验题：**

Iris数据集（鸢尾花数据集）是常用的分类实验数据集，由Fisher于1936收集整理。数据集包含150个数据样本，分为3类，每类50个数据，每个数据包含4个属性。4个属性分别为花萼长度，花萼宽度，花瓣长度，花瓣宽度，单位是cm。3个类别分别为Setosa（山鸢尾），Versicolour（杂色鸢尾），Virginica（维吉尼亚鸢尾）。

**1. Iris数据集已与常见的机器学习工具集成，请查阅资料找出MATLAB平台或Python平台加载内置Iris数据集方法，并简要描述该数据集结构。**



……



{

"data":..., # data字段是numpy数据特征

"target":..., # target是花的类型

"target\_names":... # target\_name是指花名

"DESCR":... #DESCR是数据集描述

…..

}

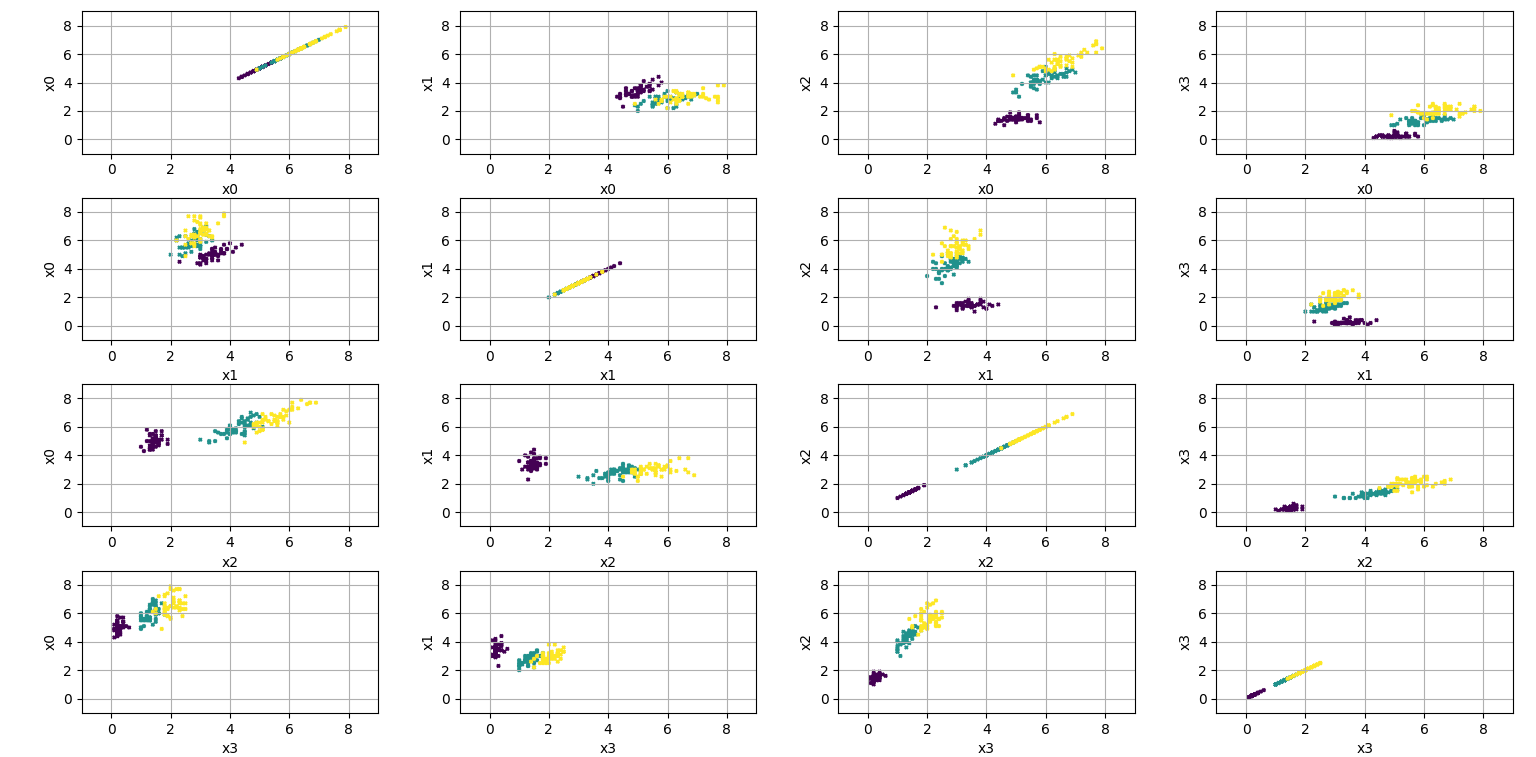
该数据集是一个map，其中data中含有150组数据，每组数据有四种属性值；target中为data中数据对应的分类，有0,1,2共三种；target\_names中为分类值所对应的花名

**核心代码**

1. **iris = datasets.load\_iris()**
2. **iris\_data = iris.data**
3. **iris\_target = iris.target**
4. **iris\_target\_names=iris.target\_names**
5. **print(iris)**
6. **visualization(iris\_data,iris\_target,4,4)**  **#可视化**

**def visualization(feature, label, m, n)见附录**

**2. Iris数据集中有一个种类与另外两个类是线性可分的，其余两个类是线性不可分的。请你通过数据可视化的方法找出该线性可分类并给出判断依据。**



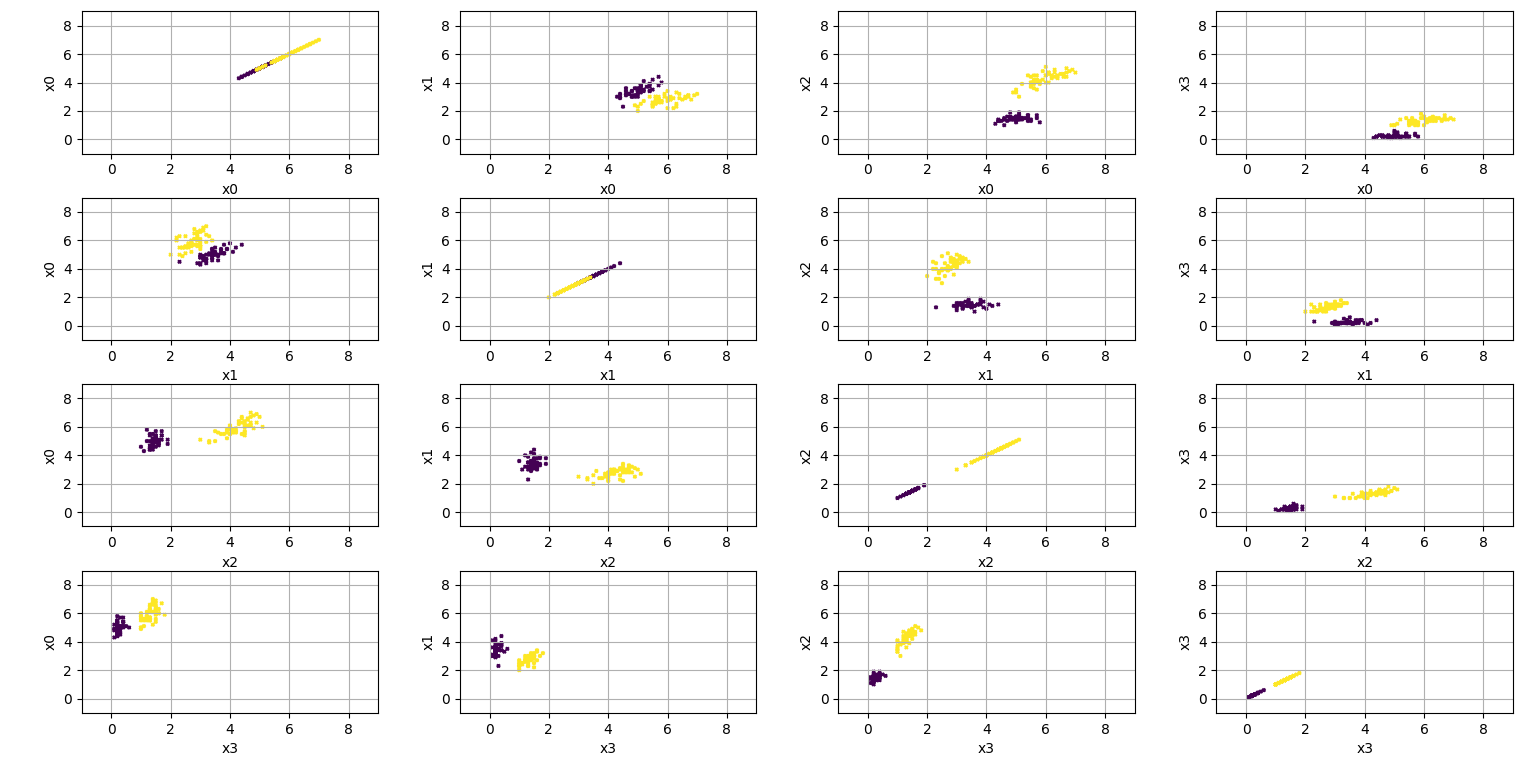
如上图所示0是线性可分的，1,2是线性不可分的，可以通过一个线性函数将其与另外两个类分开

1. **# 核心代码**
2. **iris\_data\_linear, iris\_target\_linear = remove\_from\_data(iris\_data, iris\_target, 2)**
3. **visualization(iris\_data\_linear,iris\_target\_linear,4,4)**

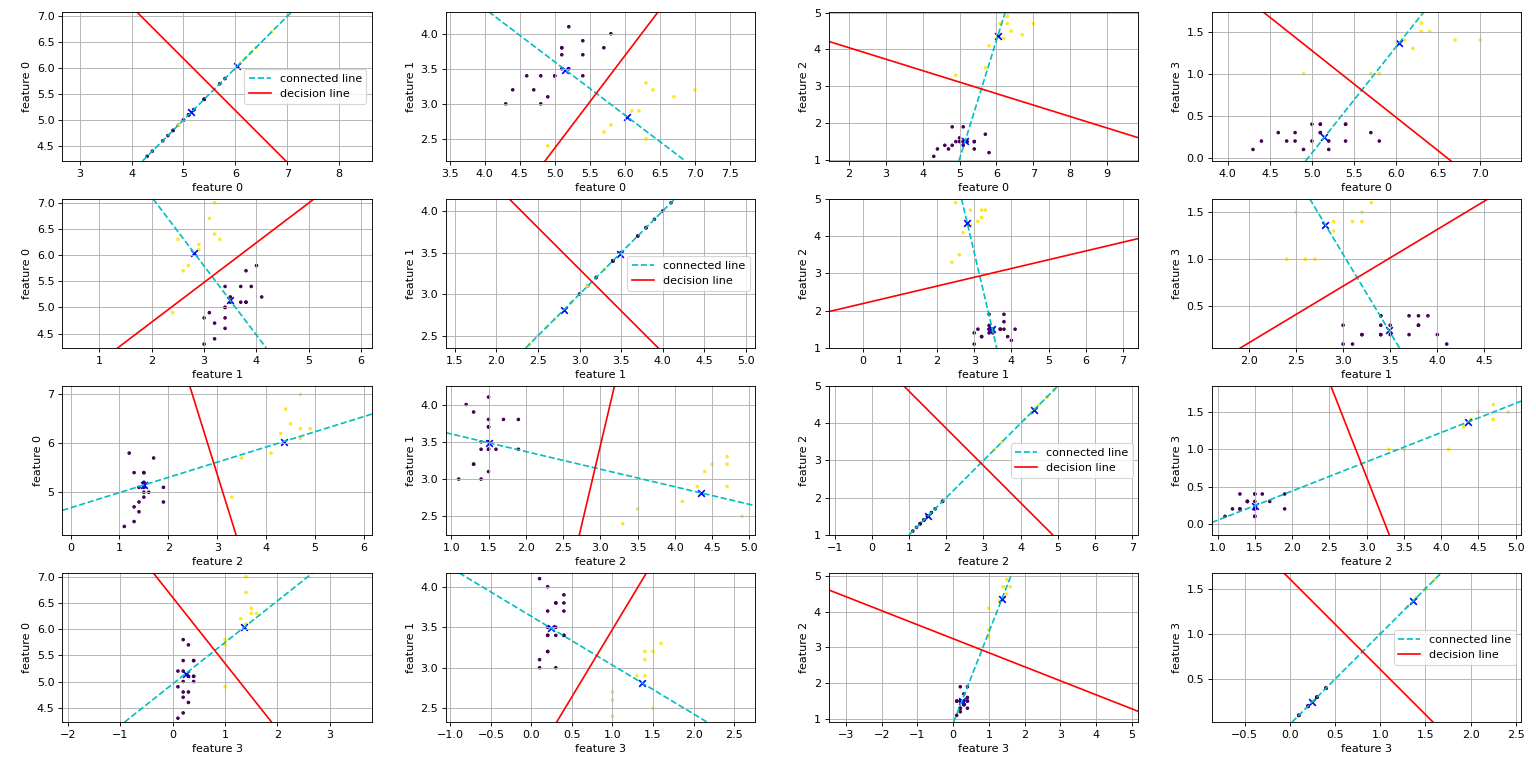
def remove\_from\_data(feature, label, num):见附录

**3.去除Iris数据集中线性不可分的类中最后一个，余下的两个线性可分的类构成的数据集命令为Iris\_linear，请使用留出法将Iris\_linear数据集按7:3分为训练集与测试集，并使用训练集训练一个MED分类器，在测试集上测试训练好的分类器的性能，给出《模式识别与机器学习-评估方法与性能指标》中所有量化指标并可视化分类结果。**

去除2后的散点图



分类效果图



指标：

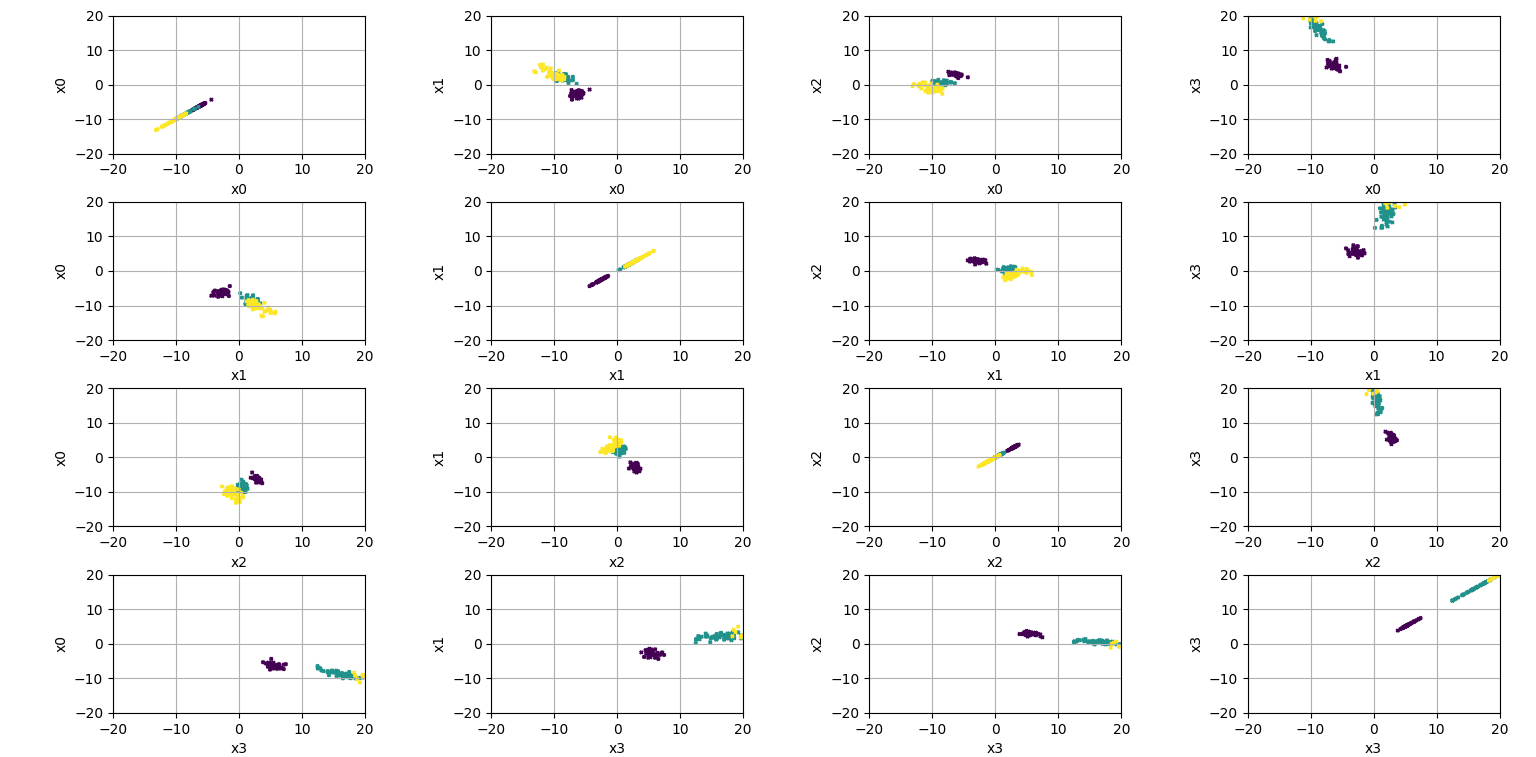


1. **# 核心代码**
2. **#划分训练集、测试集**
3. **x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris\_data\_linear, iris\_target\_linear, test\_size=0.3)**
4. **meds=Medclass()**
5. **meds.train(x\_train,y\_train)**
6. **meds.performance(x\_test, y\_test, 0)**
7. **# 展示每个特征两两对比图，显示决策线**
8. **show\_decision\_line(x\_test, y\_test, meds, class\_1=0, class\_2=1, n=4)**

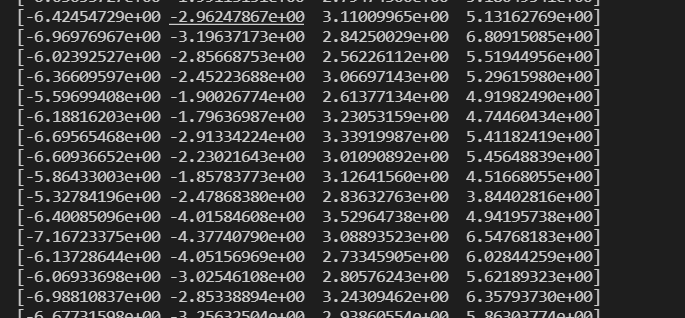
Medclass()详情见附录

**4. 将Iris数据集白化，可视化白化结果并于原始可视化结果比较，讨论白化的作用。**

白化效果图：



白化后的数据



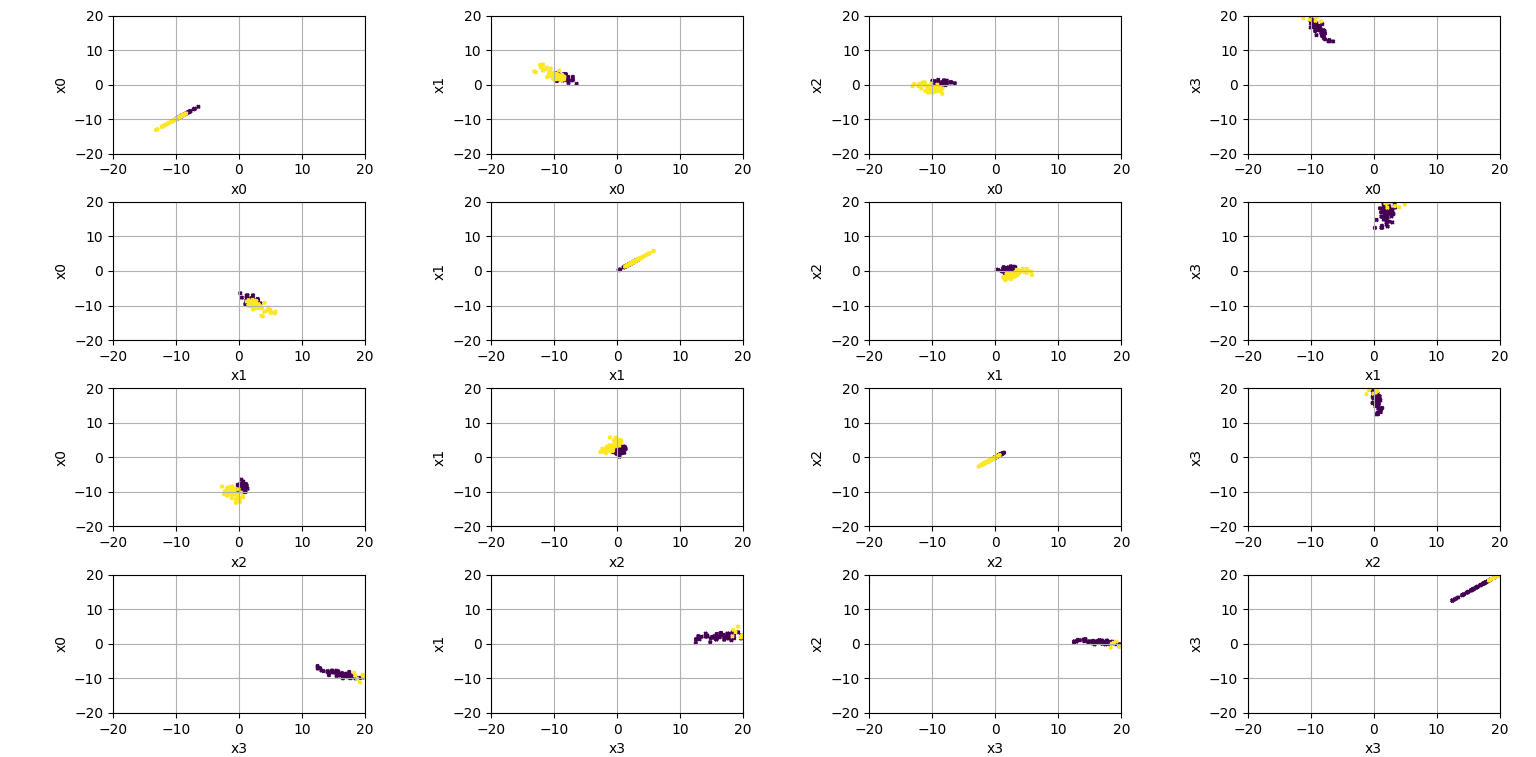
特征白化的目的：将原始特征映射到新的一个特征空间，使得在新空间中特征的协方差为单位矩阵，从而去除特征变化的不同及特征之间的相关性。

特征白化后的数据的特征之间的相关性去除

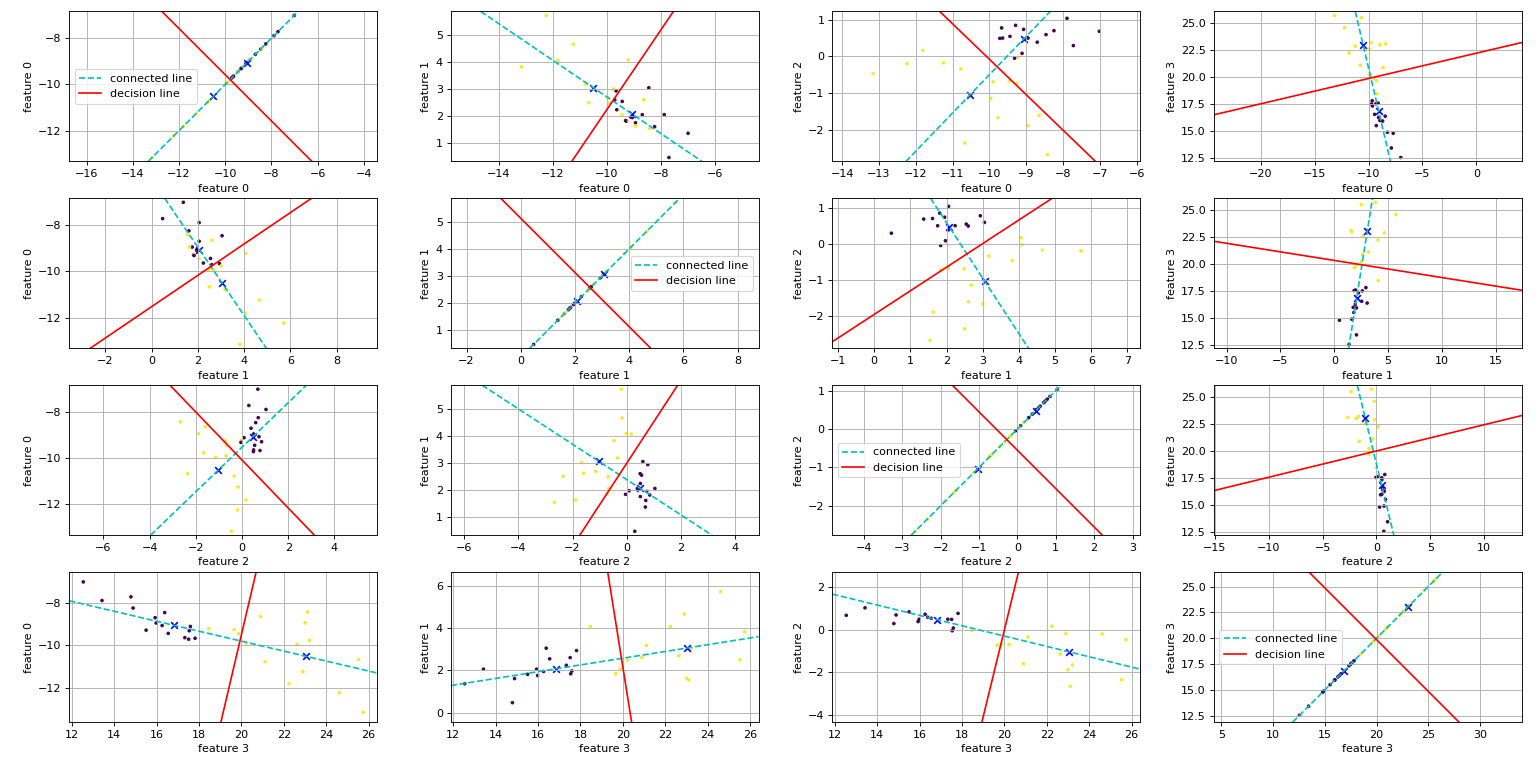
1. **# 核心代码**
2. **iris\_data\_white = whitening(iris\_data)**
3. **print(iris\_data\_white)**
4. **visualization\_white(iris\_data\_white,iris\_target,4,4)**
5. **# 核心代码**
6. **def whitening(data):**
7. **Ex=np.cov(data,rowvar=False) #Ex为data的协方差矩阵**
8. **print(Ex.shape)**
9. **a, b = np.linalg.eig(Ex) #原始特征协方差矩阵Ex的特征值和特征向量**
10. **#特征向量单位化**
11. **modulus=[]**
12. **b=np.real(b)**
13. **for i in range(b.shape[1]):**
14. **sum=0**
15. **for j in range(b.shape[0]):**
16. **sum+=b[i][j]\*\*2**
17. **modulus.append(sum)**
18. **modulus=np.asarray(modulus,dtype="float64")**
19. **b=b/modulus**
20. **#对角矩阵A**
21. **a=np.real(a)**
22. **A=np.diag(a\*\*(-0.5))**
23. **W=np.dot(A,b.transpose())**
24. **X=np.dot(W,np.dot(Ex,W.transpose()))**
25. **for i in range(W.shape[0]):**
26. **for j in range(W.shape[1]):**
27. **if np.isnan(W[i][j]):**
28. **W[i][j]=0**
29. **print(W)**
30. **return np.dot(data,W)print(iris\_data\_white)**

**5. 去除Iris数据集中线性可分的类，余下的两个线性不可分的类构成的数据集命令为Iris\_nonlinear，请使用留出法将Iris\_nonlinear数据集按7:3分为训练集与测试集，并使用训练集训练一个MED分类器，在测试集上测试训练好的分类器的性能，给出《模式识别与机器学习-评估方法与性能指标》中所有量化指标并可视化分类结果。讨论本题结果与3题结果的差异。**

去除0的效果图



分类效果图



指标：



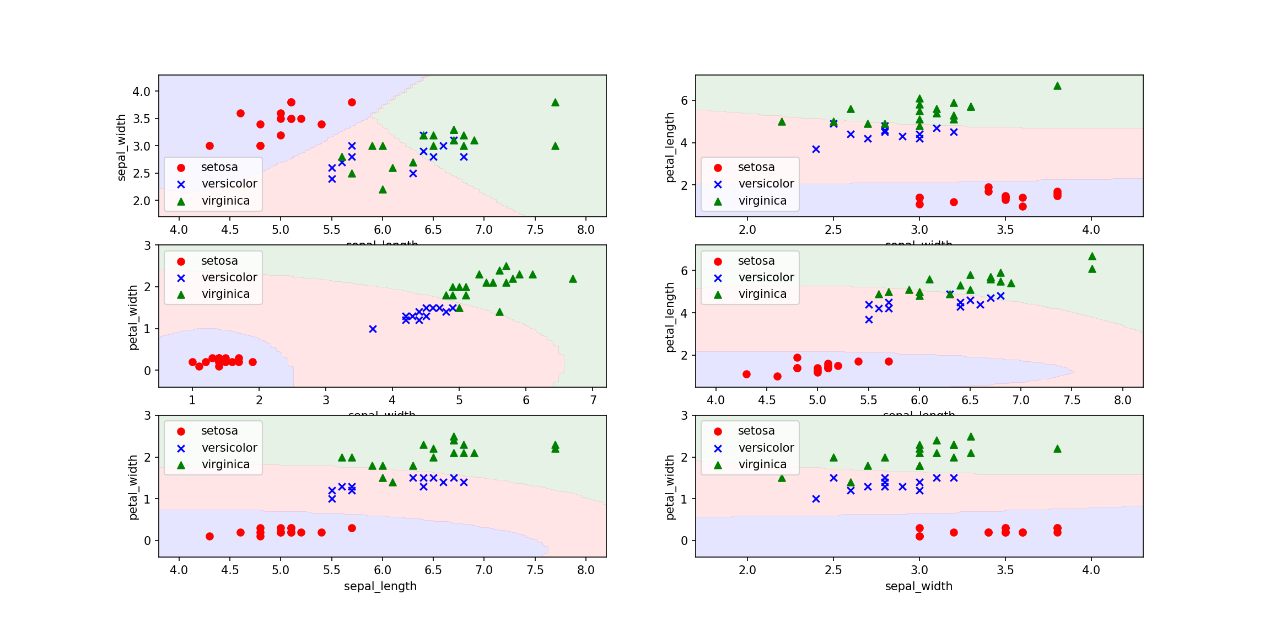
第五题的分类效果与第三题的分类效果较差，正确率下降

1. **# 核心代码**
2. **iris\_data\_nolinear, iris\_target\_nolinear = remove\_from\_data(iris\_data\_white, iris\_target, 0)#白化**
3. **visualization\_white(iris\_data\_nolinear,iris\_target\_nolinear,4,4)**
4. **#划分训练集、测试集**
5. **x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris\_data\_nolinear, iris\_target\_nolinear, test\_size=0.3)**
6. **meds2=Medclass()**
7. **meds2.train(x\_train,y\_train)**
8. **meds2.performance(x\_test, y\_test, 1)**
9. **# 展示每个特征两两对比图，显示决策线**
10. **show\_decision\_line(x\_test, y\_test, meds2, class\_1=1, class\_2=2, n=4)**

**6. 请使用5折交叉验证为Iris数据集训练一个多分类的贝叶斯分类器。给出平均Accuracy，并可视化实验结果。与第3题和第5题结果做比较，讨论贝叶斯分类器的优劣。**

平均accuracy: 0.9933333333333334

分类效果图



贝叶斯分类器与其他方法相比最大的优势或许就在于，它在接受大数据量训练和查询时所具备的高速度。贝叶斯分类器的另一大优势是，对分类器实际学习状况的解释还是相对简单的。贝叶斯分类器的最大缺陷就是，它无法处理基于特征组合所产生的变化结果。

与第3题和第5题结果做比较贝叶斯分类器的分类效果比较好，准确率高

代码见附录Bayess.py

1. **class BayesClassifier(): #贝叶斯分类器,高斯分布概率估计**
2. **def \_\_init\_\_(self):**
3. **self.parameters=[]**
4. **def train(self,X\_data,Y\_data):**
5. **for categorys in set(Y\_data):#遍历每一种类别**
6. **selected= Y\_data==categorys #选中对应该类别的数据**
7. **X\_newData= X\_data[selected] #得到新数据**
8. **mean=np.mean(X\_newData,axis=0) #得到均值**
9. **cov = np.cov(X\_newData.transpose())**
10. **self.parameters.append(BayesParameter(mean,cov,categorys))**
11. **def predit(self,data):**
12. **res=-1**
13. **probability=0**
14. **for parameter in self.parameters:**
15. **if stats.multivariate\_normal.pdf(data, mean=parameter.mean, cov=parameter.cov)>probability:**
16. **res=parameter.category**
17. **probability=stats.multivariate\_normal.pdf(data, mean=parameter.mean, cov=parameter.cov)**
18. **return res**
19. **def K\_Folds\_Cross\_Validation(data,tar,k):**
20. **Set=[]**
21. **Tar=[]**
22. **for i in range(k):**
23. **tempSet=[]**
24. **tempTar=[]**
25. **tempSet.extend(data[i\*10:(i+1)\*10])**
26. **tempTar.extend(tar[i\*10:(i+1)\*10])**
27. **tempSet.extend(data[(i+5) \* 10:(i + 6) \* 10])**
28. **tempTar.extend(tar[(i+5) \* 10:(i + 6) \* 10])**
29. **tempSet.extend(data[(i+10) \* 10:(i + 11) \* 10])**
30. **tempTar.extend(tar[(i+10) \* 10:(i + 11) \* 10])**
31. **Set.append(tempSet)**
32. **Tar.append(tempTar)**
33. **return np.asarray(Set),np.asarray(Tar)**

附录：

Main.py(1-5题)

1. **import numpy as np**
2. **from sklearn import datasets**
3. **from sklearn.model\_selection import train\_test\_split**
4. **import matplotlib.pyplot as plt**
5. **import numpy as np**
6. **from sklearn.model\_selection import train\_test\_split**
7. **# MED分类器**
8. **class Medclass:**
9. **def \_\_init\_\_(self):**
10. **self.center\_dict = {} # 分类中心点，以类别标签为键 label: center\_point(list)**
11. **self.feature\_number = 0 # 特征维度**
12. **self.train\_state = False # 训练状态，True为训练完成，False表示还没训练过**
13. **def train(self, feature\_set, label\_set):**
14. **new\_label\_set = {key: value for key, value in enumerate(label\_set)} # 将标签集合转换为以下标为键的字典 index: label**
15. **self.feature\_number = len(feature\_set[0])**
16. **sample\_num = len(label\_set) # 样本个数**
17. **count = {} # 计算每个类别的样本个数 label: count(int)**
18. **# 计算每个类别的分类中心点**
19. **for index in range(sample\_num):**
20. **if new\_label\_set[index] not in count.keys():**
21. **count[new\_label\_set[index]] = 0**
22. **else:**
23. **count[new\_label\_set[index]] += 1 # 计算对应标签的样本数**
24. **if new\_label\_set[index] not in self.center\_dict.keys():**
25. **self.center\_dict[new\_label\_set[index]] = feature\_set[index]**
26. **else:**
27. **self.center\_dict[new\_label\_set[index]] += feature\_set[index]**
28. **for \_key\_ in self.center\_dict.keys():**
29. **for \_feature\_ in range(self.feature\_number):**
30. **self.center\_dict[\_key\_][\_feature\_] /= count[\_key\_]**
31. **self.train\_state = True**
32. **# 根据输入来进行分类预测，输出以 下标—预测分类 为键值对的字典**
33. **def predict(self, feature\_set):**
34. **# 先判断此分类器是否经过训练**
35. **if not self.train\_state:**
36. **return {}**
37. **sample\_num = len(feature\_set)**
38. **distance\_to = {} # 计算某个样本到各分类中心点距离的平方 label: float**
39. **result = {} # 保存分类结果 index: label**
40. **for \_sample\_ in range(sample\_num):**
41. **for \_key\_ in self.center\_dict.keys():**
42. **delta = feature\_set[\_sample\_] - self.center\_dict[\_key\_]**
43. **distance\_to[\_key\_] = np.dot(delta.T, delta)**
44. **result[\_sample\_] = min(distance\_to, key=distance\_to.get) # 返回最小值的键（即label）**
45. **return result**
46. **# 判断预测准确率**
47. **def accuracy(self, feature\_set, label\_set):**
48. **if not self.train\_state:**
49. **return 0.0**
50. **correct\_num = 0**
51. **total\_num = len(label\_set)**
52. **predict = self.predict(feature\_set)**
53. **for \_sample\_ in range(total\_num):**
54. **if predict[\_sample\_] == label\_set[\_sample\_]:**
55. **correct\_num += 1**
56. **return correct\_num / total\_num**
57. **# 根据指定的阳性类别，计算分类器的性能指标（准确率accuracy，精度precision，召回率recall，特异性specificity，F1\_Score）**
58. **def performance(self, feature\_set, label\_set, positive):**
59. **if not self.train\_state:**
60. **return {}**
61. **total\_num = len(label\_set)**
62. **predict = self.predict(feature\_set)**
63. **true\_positive, false\_positive, true\_negative, false\_negative = 0, 0, 0, 0**
64. **for \_sample\_ in range(total\_num):**
65. **if predict[\_sample\_] == label\_set[\_sample\_]:**
66. **if label\_set[\_sample\_] == positive:**
67. **true\_positive += 1**
68. **else:**
69. **true\_negative += 1**
70. **else:**
71. **if label\_set[\_sample\_] == positive:**
72. **false\_negative += 1**
73. **else:**
74. **false\_positive += 1**
75. **print("tp=",true\_positive,"tn=",true\_negative,"fn=",false\_negative,"fp=",false\_positive)**
76. **accuracy = (true\_positive + true\_negative) / total\_num # 准确率（预测正确的样本与总样本数之比）**
77. **precision = true\_positive / (true\_positive + false\_positive) # 精度（所有 预测 为阳性的样本中， 真值 为阳性的比例）**
78. **recall = true\_positive / (true\_positive + false\_negative) # 召回率（所有 真值 为阳性的样本中， 预测 为阳性的比例）**
79. **specificity = true\_negative / (true\_negative + false\_positive) # 特异性（所有 真值 为阴性的样本中， 预测 为阴性的比例）**
80. **F1\_Score = (2 \* precision \* recall) / (precision + recall) # 精度与召回率的加权平均**
81. **print("accuracy:", accuracy, "precision:", precision, "recall:", recall, "specificity:",specificity, "F1\_Score:", F1\_Score)**
83. **# 获取某一类的样本中心点**
84. **def get\_center(self, key):**
85. **if key in self.center\_dict.keys():**
86. **return self.center\_dict[key]**
87. **else:**
88. **return []**
89. **def get\_center\_dict(self):**
90. **return self.center\_dict**
91. **#end**
92. **#画分割线**
93. **# 展示二维平面上，二分类问题的决策线（class\_1和class\_2）**
94. **# feature是样本特征集合，label是对应的标签集合，对每一维特征进行两两比较，n表示特征维数**
95. **def show\_decision\_line(feature, label, med\_classifier, class\_1=0, class\_2=0, n=0):**
96. **plt.figure(figsize=(16, 12), dpi=80) # 整张画布大小与分辨率**
97. **img = [[] for i in range(n \* n)]**
98. **for i in range(n):**
99. **for j in range(n):**
100. **img[i \* n + j] = plt.subplot(n, n, i \* n + j + 1)**
101. **center\_1 = med\_classifier.get\_center(class\_1)**
102. **center\_2 = med\_classifier.get\_center(class\_2)**
103. **c\_1 = [center\_1[i], center\_1[j]] # class\_1类中心点的i, j两维的分量**
104. **c\_2 = [center\_2[i], center\_2[j]] # class\_2类中心点的i, j两维的分量**
105. **center\_3 = [(c\_1[0] + c\_2[0]) / 2, (c\_1[1] + c\_2[1]) / 2] # 两点连线的中点**
106. **k2, b2 = calculate\_vertical\_line(c\_1, c\_2) # 两点中垂线的斜率和截距**
107. **plt.scatter(feature[:, i], feature[:, j], c=label, s=20, marker='.') # 整个样本集在特征0和2上的散点图**
108. **plt.scatter(c\_1[0], c\_1[1], c='b', marker='x') # 显示med分类器计算的样本中心点**
109. **plt.scatter(c\_2[0], c\_2[1], c='b', marker='x')**
110. **plt.grid(True) # 显示网格线**
111. **plt.axis('equal') # 横纵坐标间隔大小相同**
112. **plt.axline(c\_1, c\_2, color='c', linestyle="--", label="connected line")**
113. **plt.axline(center\_3, slope=k2, color='r', label="decision line")**
114. **if i == j:**
115. **plt.legend() # 对角线上的子图显示出图例**
116. **plt.xlabel("feature " + str(i))**
117. **plt.ylabel("feature " + str(j))**
118. **plt.tight\_layout() # 自动调整子图大小，减少相互遮挡的问题**
119. **plt.show()**
120. **# 计算两点连线，返回斜率和纵截距（假设是二维平面上的点，并且用列表表示）**
121. **def calculate\_connected\_line(point\_1, point\_2):**
122. **if len(point\_1) != 2 or len(point\_2) != 2:**
123. **return None**
124. **k = (point\_1[1] - point\_2[1]) / (point\_1[0] - point\_2[0])**
125. **b = (point\_1[0] \* point\_2[1] - point\_2[0] \* point\_1[1]) / (point\_1[0] - point\_2[0])**
126. **return k, b**
127. **# 计算两点中垂线，返回斜率和纵截距（假设是二维平面上的点，并且用列表表示）**
128. **def calculate\_vertical\_line(point\_1, point\_2):**
129. **if len(point\_1) != 2 or len(point\_2) != 2:**
130. **return None**
131. **k = -(point\_1[0] - point\_2[0]) / (point\_1[1] - point\_2[1])**
132. **b = (point\_1[1] + point\_2[1] + (point\_1[0] + point\_2[0]) \* (point\_1[0] - point\_2[0]) / (point\_1[1] - point\_2[1]))/2**
133. **return k, b**
134. **#画分割线end**
135. **# feature表示样本特征，label表示对应的标签,m行n列共计m\*n个子图**
136. **def visualization(feature, label, m, n):**
137. **plt.figure(figsize=(10, 10), dpi=100)**
138. **img = [[] for i in range(m\*n)]**
139. **for i in range(m):**
140. **for j in range(n):**
141. **img[i\*n+j] = plt.subplot(m, n, i\*n+j+1)**
142. **plt.xlabel("x"+str(i))**
143. **plt.ylabel("x"+str(j))**
144. **plt.xlim(-1, 9)**
145. **plt.ylim(-1, 9)**
146. **plt.scatter(feature[:, i], feature[:, j], s=5, c=label, marker='x')**
147. **plt.grid(True) # 显示网格线**
148. **plt.tight\_layout() # 自动调整子图大小，减少相互遮挡的问题**
149. **plt.show()**
150. **# feature表示样本特征，label表示对应的标签,m行n列共计m\*n个子图**
151. **def visualization\_white(feature, label, m, n):**
152. **plt.figure(figsize=(10, 10), dpi=100)**
153. **img = [[] for i in range(m\*n)]**
154. **for i in range(m):**
155. **for j in range(n):**
156. **img[i\*n+j] = plt.subplot(m, n, i\*n+j+1)**
157. **plt.xlabel("x"+str(i))**
158. **plt.ylabel("x"+str(j))**
159. **plt.xlim(-20, 20)**
160. **plt.ylim(-20, 20)**
161. **plt.scatter(feature[:, i], feature[:, j], s=5, c=label, marker='x')**
162. **plt.grid(True) # 显示网格线**
163. **plt.tight\_layout() # 自动调整子图大小，减少相互遮挡的问题**
164. **plt.show()**
165. **# 去除某个类别的样本，返回两个numpy数组**
166. **def remove\_from\_data(feature, label, num):**
167. **new\_feature = []**
168. **new\_label = []**
169. **for index in range(len(label)):**
170. **if label[index] != num:**
171. **new\_feature.append(feature[index])**
172. **new\_label.append(label[index])**
173. **return np.asarray(new\_feature), np.asarray(new\_label)**
174. **# 特征白化，返回白化后的矩阵（numpy数组格式）**
175. **# 参数为numpy格式的数组，其格式为数学上的矩阵的转置**
176. **def whitening(data):**
177. **Ex=np.cov(data,rowvar=False) #Ex为data的协方差矩阵**
178. **print(Ex.shape)**
179. **a, b = np.linalg.eig(Ex) #原始特征协方差矩阵Ex的特征值和特征向量**
180. **#特征向量单位化**
181. **modulus=[]**
182. **b=np.real(b)**
183. **for i in range(b.shape[1]):**
184. **sum=0**
185. **for j in range(b.shape[0]):**
186. **sum+=b[i][j]\*\*2**
187. **modulus.append(sum)**
188. **modulus=np.asarray(modulus,dtype="float64")**
189. **b=b/modulus**
190. **#对角矩阵A**
191. **a=np.real(a)**
192. **A=np.diag(a\*\*(-0.5))**
193. **W=np.dot(A,b.transpose())**
194. **X=np.dot(W,np.dot(Ex,W.transpose()))**
195. **for i in range(W.shape[0]):**
196. **for j in range(W.shape[1]):**
197. **if np.isnan(W[i][j]):**
198. **W[i][j]=0**
199. **print(W)**
200. **return np.dot(data,W)**
201. **if \_\_name\_\_ == '\_\_main\_\_':**
202. **iris = datasets.load\_iris()**
203. **iris\_data = iris.data**
204. **iris\_target = iris.target**
205. **iris\_target\_names=iris.target\_names**
206. **print(iris)**
207. **#可视化**
208. **visualization(iris\_data,iris\_target,4,4)**
209. **#去除线性不可分的最后一个**
210. **iris\_data\_linear, iris\_target\_linear = remove\_from\_data(iris\_data, iris\_target, 2)**
211. **visualization(iris\_data\_linear,iris\_target\_linear,4,4)**
212. **#划分训练集、测试集**
213. **x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris\_data\_linear, iris\_target\_linear, test\_size=0.3)**
214. **meds=Medclass()**
215. **meds.train(x\_train,y\_train)**
216. **meds.performance(x\_test, y\_test, 0)**
217. **# 展示每个特征两两对比图，显示决策线**
218. **show\_decision\_line(x\_test, y\_test, meds, class\_1=0, class\_2=1, n=4)**
219. **#特征白化**
220. **iris\_data\_white = whitening(iris\_data)**
221. **print(iris\_data\_white)**
222. **visualization\_white(iris\_data\_white,iris\_target,4,4)**
223. **#去除线性可分的类**
224. **#iris\_data\_nolinear, iris\_target\_nolinear = remove\_from\_data(iris\_data, iris\_target, 0) #无白化**
225. **#visualization(iris\_data\_nolinear,iris\_target\_nolinear,4,4)**
226. **iris\_data\_nolinear, iris\_target\_nolinear = remove\_from\_data(iris\_data\_white, iris\_target, 0)#白化**
227. **visualization\_white(iris\_data\_nolinear,iris\_target\_nolinear,4,4)**
228. **#划分训练集、测试集**
229. **x\_train, x\_test, y\_train, y\_test = train\_test\_split(iris\_data\_nolinear, iris\_target\_nolinear, test\_size=0.3)**
230. **meds2=Medclass()**
231. **meds2.train(x\_train,y\_train)**
232. **meds2.performance(x\_test, y\_test, 1)**
233. **# 展示每个特征两两对比图，显示决策线**
234. **show\_decision\_line(x\_test, y\_test, meds2, class\_1=1, class\_2=2, n=4)**

Bayess.py(第6题)

1. **import numpy as np**
2. **from sklearn import datasets**
3. **from sklearn.naive\_bayes import GaussianNB**
4. **from sklearn.model\_selection import train\_test\_split**
5. **import matplotlib.pyplot as plt**
6. **import numpy as np**
7. **from sklearn.model\_selection import train\_test\_split**
8. **import matplotlib.pyplot as plt**
9. **import numpy as np**
10. **from scipy import stats**
11. **plt.rcParams['savefig.dpi'] = 150 #图片像素**
12. **plt.rcParams['figure.dpi'] = 100 #分辨率**
13. **#贝叶斯分类器**
14. **class BayesParameter(): #存储贝叶斯分类器参数**
15. **def \_\_init\_\_(self,mean,cov,category):**
16. **self.mean=mean**
17. **self.cov=cov**
18. **self.category=category**
20. **class BayesClassifier(): #贝叶斯分类器,高斯分布概率估计**
21. **def \_\_init\_\_(self):**
22. **self.parameters=[]**
23. **def train(self,X\_data,Y\_data):**
24. **for categorys in set(Y\_data):#遍历每一种类别**
25. **selected= Y\_data==categorys #选中对应该类别的数据**
26. **X\_newData= X\_data[selected] #得到新数据**
27. **mean=np.mean(X\_newData,axis=0) #得到均值**
28. **cov = np.cov(X\_newData.transpose())**
29. **self.parameters.append(BayesParameter(mean,cov,categorys))**
30. **def predit(self,data):**
31. **res=-1**
32. **probability=0**
33. **for parameter in self.parameters:**
34. **if stats.multivariate\_normal.pdf(data, mean=parameter.mean, cov=parameter.cov)>probability:**
35. **res=parameter.category**
36. **probability=stats.multivariate\_normal.pdf(data, mean=parameter.mean, cov=parameter.cov)**
37. **return res**
38. **def K\_Folds\_Cross\_Validation(data,tar,k):**
39. **Set=[]**
40. **Tar=[]**
41. **for i in range(k):**
42. **tempSet=[]**
43. **tempTar=[]**
44. **tempSet.extend(data[i\*10:(i+1)\*10])**
45. **tempTar.extend(tar[i\*10:(i+1)\*10])**
46. **tempSet.extend(data[(i+5) \* 10:(i + 6) \* 10])**
47. **tempTar.extend(tar[(i+5) \* 10:(i + 6) \* 10])**
48. **tempSet.extend(data[(i+10) \* 10:(i + 11) \* 10])**
49. **tempTar.extend(tar[(i+10) \* 10:(i + 11) \* 10])**
50. **Set.append(tempSet)**
51. **Tar.append(tempTar)**
52. **return np.asarray(Set),np.asarray(Tar)**
53. **def data\_visualization(data,tar):**
54. **trainSet,testSet, trainTar,testTar = train\_test\_split(data, tar, test\_size=0.3)**
55. **bc = BayesClassifier()**
56. **bc.train(trainSet, trainTar)**
57. **testPredict = np.array([bc.predit(x) for x in testSet],dtype="int")**
58. **# 画图部分**
59. **fig = plt.figure(figsize=(8, 8))**
60. **xx = [[0, 1], [1, 2], [2, 3], [0,2],[0,3],[1,3]]**
61. **yy = [["sepal\_length", "sepal\_width"],**
62. **["sepal\_width", "petal\_length"],**
63. **["sepal\_width", "petal\_width"],**
64. **["sepal\_length","petal\_length"],**
65. **["sepal\_length ","petal\_width"],**
66. **["sepal\_width","petal\_width"]]**
67. **for i in range(6):**
68. **ax = fig.add\_subplot(321 + i)**
69. **x\_max,x\_min=testSet.max(axis=0)[xx[i][0]]+0.5,testSet.min(axis=0)[xx[i][0]]-0.5**
70. **y\_max,y\_min=testSet.max(axis=0)[xx[i][1]]+0.5,testSet.min(axis=0)[xx[i][1]]-0.5**
71. **xlist = np.linspace(x\_min, x\_max, 80)**
72. **ylist = np.linspace(y\_min, y\_max, 100)**
73. **XX, YY = np.meshgrid(xlist, ylist)**
74. **bc = GaussianNB()**
75. **bc.fit(trainSet[:, xx[i]],trainTar)**
76. **xys = [np.array([xx, yy]).reshape(1, -1) for xx, yy in zip(np.ravel(XX), np.ravel(YY))]**
77. **zz = np.array([bc.predict(x) for x in xys])**
78. **Z = zz.reshape(XX.shape)**
79. **plt.contourf(XX, YY, Z, 2, alpha=.1, colors=('blue', 'red', 'green'))**
80. **ax.scatter(testSet[testPredict == 0, xx[i][0]], testSet[testPredict == 0, xx[i][1]],**
81. **c='r', marker='o',**
82. **label="setosa")**
83. **ax.scatter(testSet[testPredict==1, xx[i][0]], testSet[testPredict==1, xx[i][1]], c='b', marker='x',**
84. **label="versicolor")**
85. **ax.scatter(testSet[testPredict==2, xx[i][0]], testSet[testPredict==2, xx[i][1]], c='g', marker='^',**
86. **label="virginica")**
87. **ax.set\_xlabel(yy[i][0])**
88. **ax.set\_ylabel(yy[i][1])**
89. **ax.legend(loc=0)**
90. **plt.show()**
92. **datas=datasets.load\_iris()**
93. **data=datas.data**
94. **tar=datas.target**
95. **data\_visualization(data,tar)**
96. **if \_\_name\_\_=="\_\_main\_\_":**
97. **sets,tar=K\_Folds\_Cross\_Validation(data,tar,5)**
98. **accuracy=0**
99. **print(tar[0].shape)**
100. **for i in range(5): #第i个子集作为测试集**
101. **x,y=0,0**
102. **X\_data,Y\_data=None,None**
103. **for j in range(5):**
104. **if i!=j:**
105. **if x\*y==0:**
106. **X0\_data=sets[i]**
107. **Y0\_data=tar[i]**
108. **else:**
109. **X0\_data=np.concatenate((X0\_data,sets[i]),axis=0)**
110. **Y0\_data = np.concatenate((Y0\_data, tar[i]), axis=0)**
111. **x+=1**
112. **y+=1**
113. **bc= BayesClassifier()**
114. **bc.train(X0\_data,Y0\_data)**
116. **y\_predict=[bc.predit(x) for x in sets[i]]**
117. **tempAccuracy=np.sum(y\_predict==tar[i])/tar[i].shape[0]**
118. **accuracy+=tempAccuracy**
119. **accuracy=accuracy/5**
120. **print("accuracy:",accuracy)**