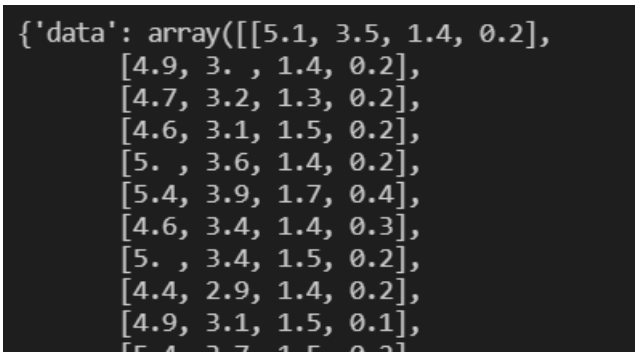
《机器学习》上机实践

031802326-苏炜斌

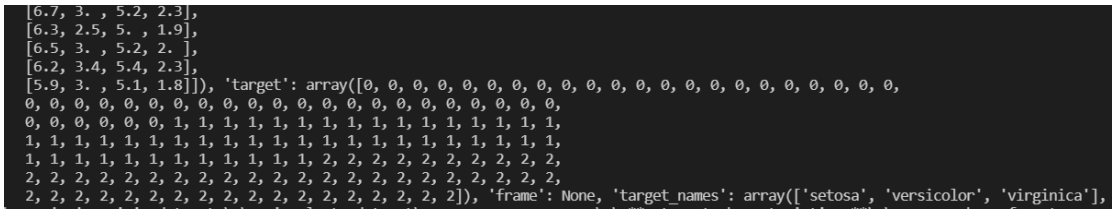
实验题：

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

…..

}

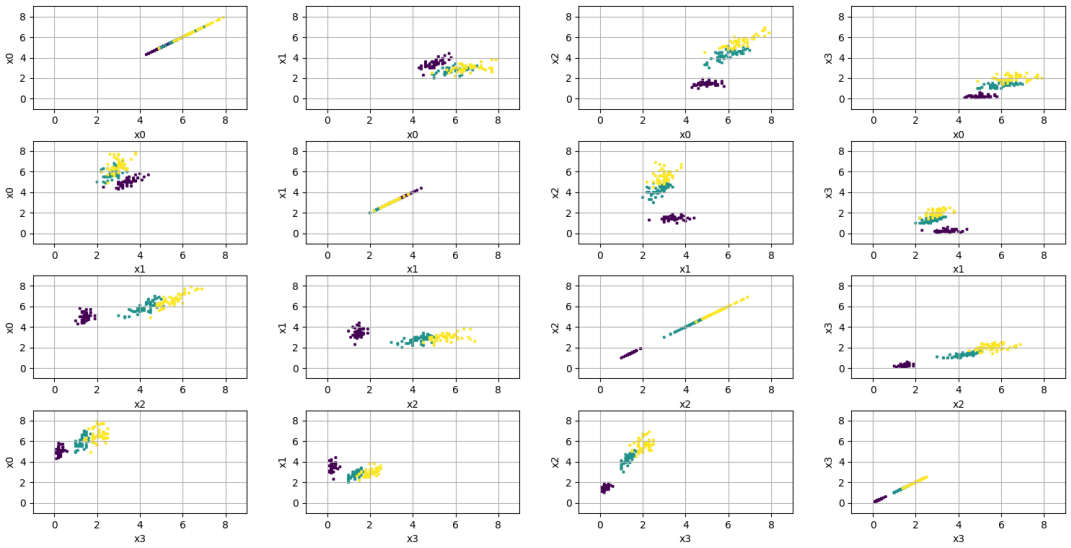
该数据集是一个 map，其中 data 中含有 150 组数据，每组数据有四种属性值；target 中为 data 中数据对应的分类，有 0,1,2 共三种；target_names 中为分类值所对应的花名

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。



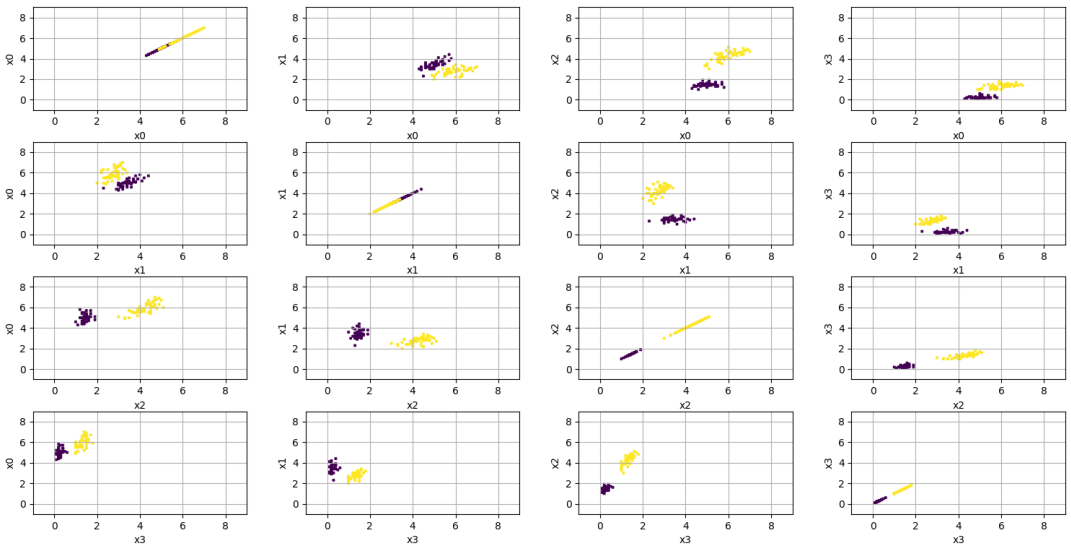
如上图所示 0 是线性可分的，1,2 是线性不可分的，可以通过一个线性函数将其与另外两个类分开

```
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)
```

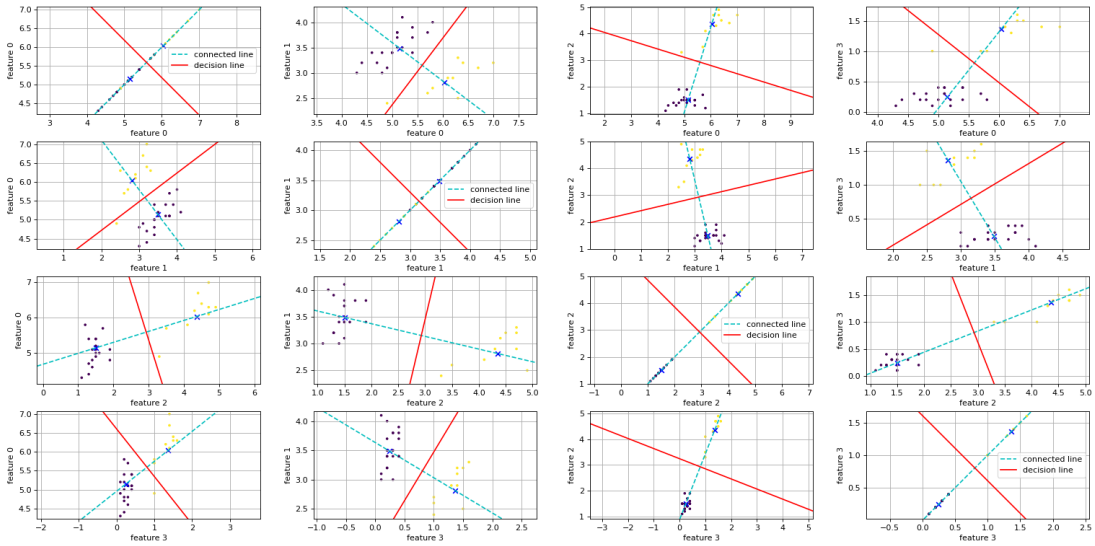
def remove_from_data(feature, label, num):见附录

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：

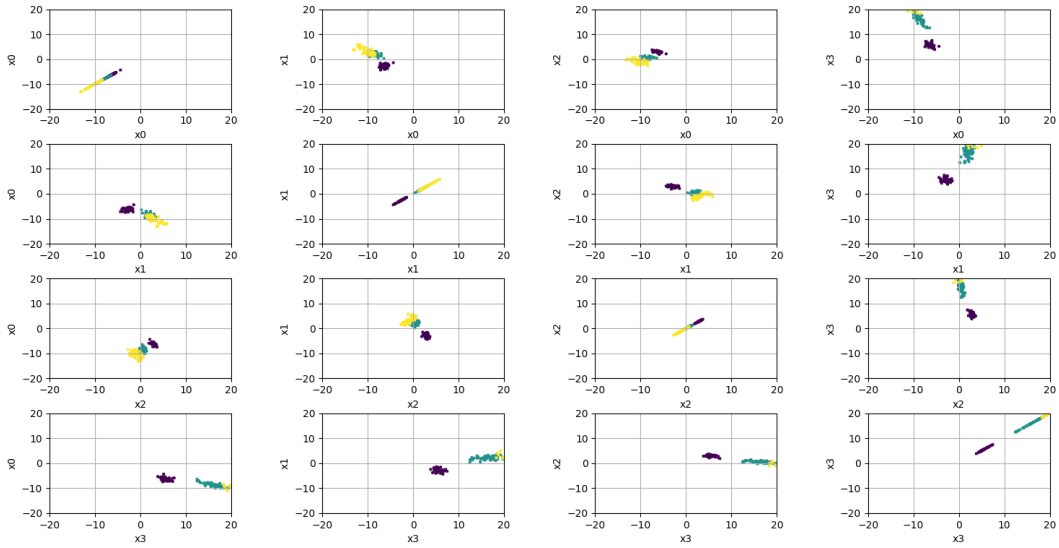
```
tp= 20 tn= 10 fn= 0 fp= 0
accuracy: 1.0 precision: 1.0 recall: 1.0 specificity: 1.0 F1_Score: 1.0
```

```
# 核心代码
#划分训练集、测试集
 x_train, x_test, y_train, y_test = train_test_split(iris_data_linear, iris_target_linear, test_size=0.3)
      meds=Medclass()
meds.train(x_train,y_train)
meds.performance(x_test, y_test, 0)
# 展示每个特征两两对比图，显示决策线
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

```
[-6.42454729e+00 -2.96247867e+00  3.11009965e+00  5.13162769e+00]
[-6.96976967e+00 -3.19637173e+00  2.84250029e+00  6.80915085e+00]
[-6.02392527e+00 -2.85668753e+00  2.56226112e+00  5.51944956e+00]
[-6.36609597e+00 -2.45223688e+00  3.06697143e+00  5.29615980e+00]
[-5.59699408e+00 -1.90026774e+00  2.61377134e+00  4.91982490e+00]
[-6.18816203e+00 -1.79636987e+00  3.23053159e+00  4.74460434e+00]
[-6.69565468e+00 -2.91334224e+00  3.33919987e+00  5.41182419e+00]
[-6.60936652e+00 -2.23021643e+00  3.01090892e+00  5.45648839e+00]
[-5.86433003e+00 -1.85783773e+00  3.12641560e+00  4.51668055e+00]
[-5.32784196e+00 -2.47868380e+00  2.83632763e+00  3.84402816e+00]
[-6.40085096e+00 -4.01584608e+00  3.52964738e+00  4.94195738e+00]
[-7.16723375e+00 -4.37740790e+00  3.08893523e+00  6.54768183e+00]
[-6.13728644e+00 -4.05156969e+00  2.73345905e+00  6.02844259e+00]
[-6.06933698e+00 -3.02546108e+00  2.80576243e+00  5.62189323e+00]
[-6.98810837e+00 -2.85338894e+00  3.24309462e+00  6.35793730e+00]
```

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。

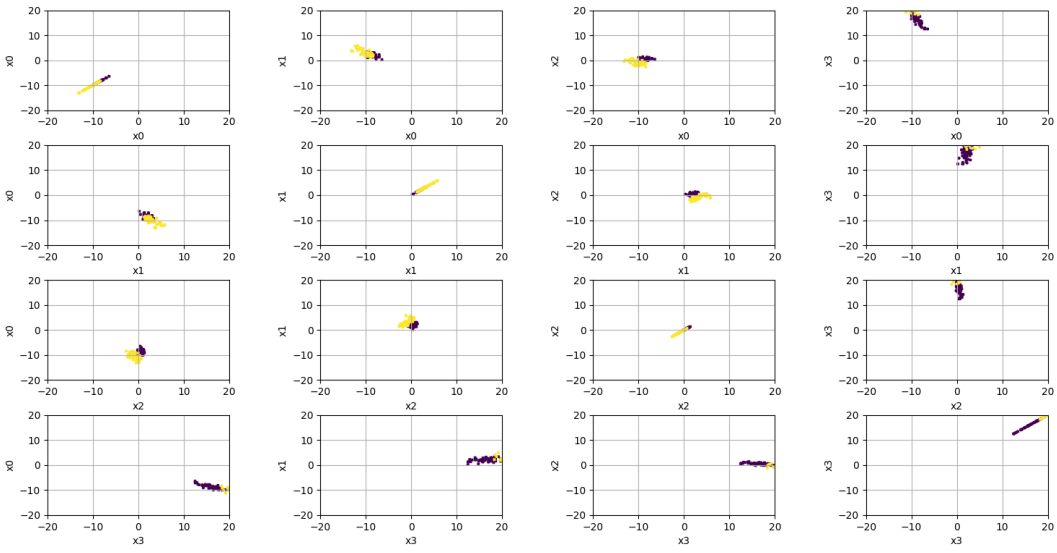
特征白化后的数据的特征之间的相关性去除

```
# 核心代码
iris_data_white = whitening(iris_data)
print(iris_data_white)
visualization_white(iris_data_white,iris_target,4,4)
# 核心代码
def whitening(data):
    Ex=np.cov(data,rowvar=False) #Ex 为 data 的协方差矩阵
    print(Ex.shape)
```

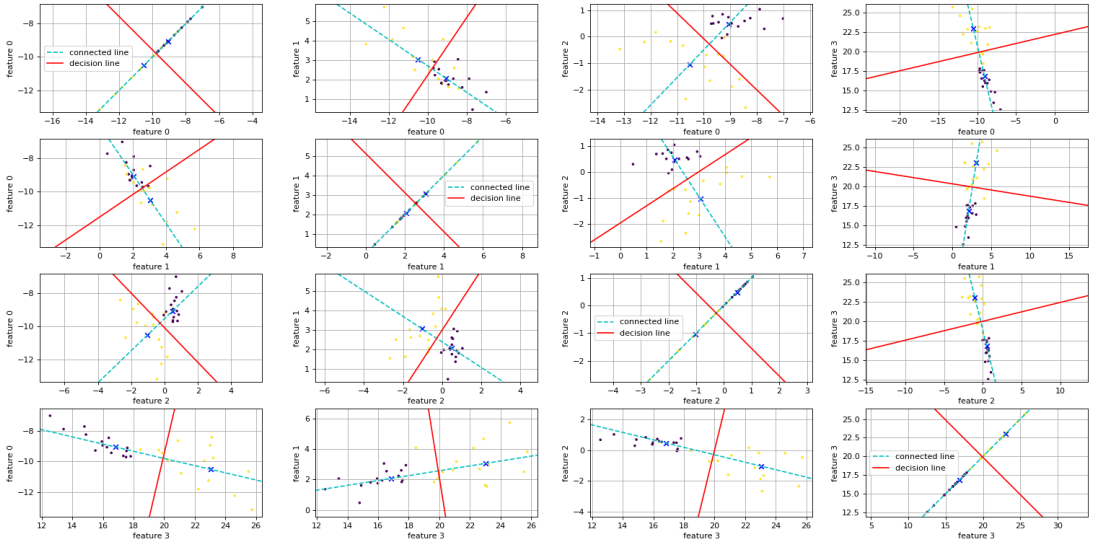
```
    a, b = np.linalg.eig(Ex) #原始特征协方差矩阵 Ex 的特征值和特征向量
    #特征向量单位化
    modulus=[]
    b=np.real(b)
    for i in range(b.shape[1]):
        sum=0
        for j in range(b.shape[0]):
            sum+=b[i][j]**2
        modulus.append(sum)
    modulus=np.asarray(modulus,dtype="float64")
    b=b/modulus
    #对角矩阵 A
    a=np.real(a)
    A=np.diag(a**(-0.5))
    W=np.dot(A,b.transpose())
    X=np.dot(W,np.dot(Ex,W.transpose()))
    for i in range(W.shape[0]):
        for j in range(W.shape[1]):
            if np.isnan(W[i][j]):
                W[i][j]=0
    print(W)
    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：

```
tp= 15 tn= 12 fn= 0 fp= 3
accuracy: 0.9 precision: 0.8333333333333334 recall: 1.0 specificity: 0.8 F1_Score: 0.9090909090909091
```

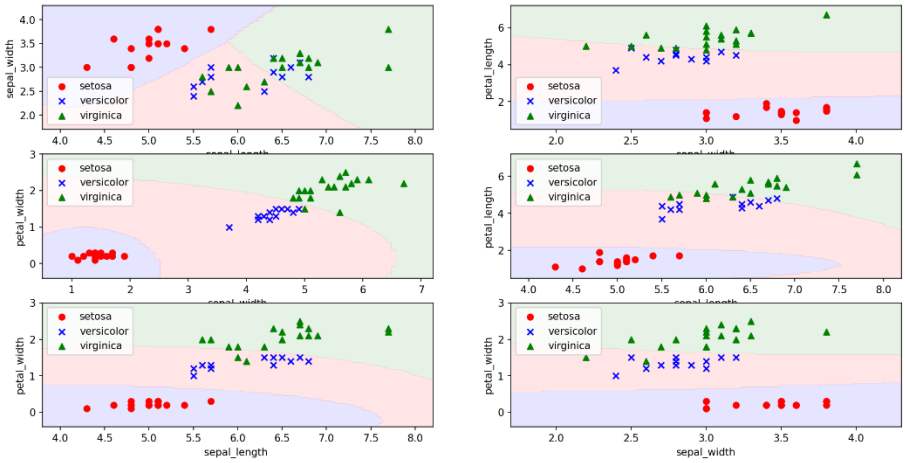
第五题的分类效果与第三题的分类效果较差，正确率下降

```
# 核心代码
iris_data_nolinear, iris_target_nolinear = remove_from_data(iris_data_white, iris_target, 0)#白化
visualization_white(iris_data_nolinear,iris_target_nolinear,4,4)
    #划分训练集、测试集
x_train, x_test, y_train, y_test = train_test_split(iris_data_nolinear, iris_target_nolinear,
test_size=0.3)
meds2=Medclass()
meds2.train(x_train,y_train)
meds2.performance(x_test, y_test, 1)
# 展示每个特征两两对比图，显示决策线
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

```
52. class   BayesClassifier():   #贝叶斯分类器,高斯分布概率估计
53.
54. def __init__(self):
55.         self.parameters=[]
56.
57. def train(self,X_data,Y_data):
58.
59.         for categorys in set(Y_data):#遍历每一种类别
60.             selected= Y_data==categorys #选中对应该类别的数据
61.             X_newData= X_data[selected] #得到新数据
62.             mean=np.mean(X_newData,axis=0) #得到均值
63.             cov = np.cov(X_newData.transpose())
64.             self.parameters.append(BayesParameter(mean,cov,categorys))
65.
66.  def predit(self,data):
67.         res=-1
68.         probability=0
69.         for parameter in self.parameters:
70.             if stats.multivariate_normal.pdf(data, mean=parameter.mean,
    cov=parameter.cov)>probability:
```

```
                    res=parameter.category
                    probability=stats.multivariate_normal.pdf(data, mean=parameter.mean,
cov=parameter.cov)
        return res

def     K_Folds_Cross_Validation(data,tar,k):
  Set=[]
  Tar=[]
  for i in range(k):
        tempSet=[]
        tempTar=[]
        tempSet.extend(data[i*10:(i+1)*10])
        tempTar.extend(tar[i*10:(i+1)*10])
        tempSet.extend(data[(i+5) * 10:(i + 6) * 10])
        tempTar.extend(tar[(i+5) * 10:(i + 6) * 10])
        tempSet.extend(data[(i+10) * 10:(i + 11) * 10])
        tempTar.extend(tar[(i+10) * 10:(i + 11) * 10])
        Set.append(tempSet)
        Tar.append(tempTar)
  return np.asarray(Set),np.asarray(Tar)
```

附录：
Main.py(1-5 题)

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split

# MED 分类器
class Medclass:
    def __init__(self):
        self.center_dict = {}  # 分类中心点，以类别标签为键    label: center_point(list)
        self.feature_number = 0  # 特征维度
        self.train_state = False  # 训练状态，True 为训练完成，False 表示还没训练过

    def train(self, feature_set, label_set):
        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
        self.feature_number = len(feature_set[0])
        sample_num = len(label_set)  # 样本个数
        count = {}  # 计算每个类别的样本个数  label: count(int)
        # 计算每个类别的分类中心点
        for index in range(sample_num):
            if new_label_set[index] not in count.keys():
                count[new_label_set[index]] = 0
            else:
                count[new_label_set[index]] += 1  # 计算对应标签的样本数
            if new_label_set[index] not in self.center_dict.keys():
                self.center_dict[new_label_set[index]] = feature_set[index]
            else:
                self.center_dict[new_label_set[index]] += feature_set[index]
        for _key_ in self.center_dict.keys():
            for _feature_ in range(self.feature_number):
                self.center_dict[_key_][_feature_] /= count[_key_]
        self.train_state = True

    # 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
    def predict(self, feature_set):
        # 先判断此分类器是否经过训练
        if not self.train_state:
            return {}
        sample_num = len(feature_set)
```

```
            distance_to = {}  # 计算某个样本到各分类中心点距离的平方  label: float
            result = {}  # 保存分类结果  index: label
            for _sample_ in range(sample_num):
                for _key_ in self.center_dict.keys():
                    delta = feature_set[_sample_] - self.center_dict[_key_]
                    distance_to[_key_] = np.dot(delta.T, delta)
                result[_sample_] = min(distance_to, key=distance_to.get)  # 返回最小值的键（即label）
            return result

        # 判断预测准确率
        def accuracy(self, feature_set, label_set):
            if not self.train_state:
                return 0.0
            correct_num = 0
            total_num = len(label_set)
            predict = self.predict(feature_set)
            for _sample_ in range(total_num):
                if predict[_sample_] == label_set[_sample_]:
                    correct_num += 1
            return correct_num / total_num

        # 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）
        def performance(self, feature_set, label_set, positive):
            if not self.train_state:
                return {}
            total_num = len(label_set)
            predict = self.predict(feature_set)
            true_positive, false_positive, true_negative, false_negative = 0, 0, 0, 0
            for _sample_ in range(total_num):
                if predict[_sample_] == label_set[_sample_]:
                    if label_set[_sample_] == positive:
                        true_positive += 1
                    else:
                        true_negative += 1
                else:
                    if label_set[_sample_] == positive:
                        false_negative += 1
                    else:
                        false_positive += 1
            print("tp=",true_positive,"tn=",true_negative,"fn=",false_negative,"fp=",false_positive)
            accuracy = (true_positive + true_negative) / total_num  # 准确率（预测正确的样本与总样本数之比）
```

```
            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）
            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）
            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）
            F1_Score = (2 * precision * recall) / (precision + recall)  # 精度与召回率的加权平均
            print("accuracy:", accuracy, "precision:", precision, "recall:", recall, "specificity:",specificity, "F1_Score:", F1_Score)


    # 获取某一类的样本中心点
    def get_center(self, key):
        if key in self.center_dict.keys():
            return self.center_dict[key]
        else:
            return []

    def get_center_dict(self):
        return self.center_dict
#end

#画分割线

# 展示二维平面上，二分类问题的决策线（class_1 和 class_2）
# 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
def show_decision_line(feature, label, med_classifier, class_1=0, class_2=0, n=0):
    plt.figure(figsize=(16, 12), dpi=80)  # 整张画布大小与分辨率
    img = [[] for i in range(n * n)]
    for i in range(n):
        for j in range(n):
            img[i * n + j] = plt.subplot(n, n, i * n + j + 1)
            center_1 = med_classifier.get_center(class_1)
            center_2 = med_classifier.get_center(class_2)
            c_1 = [center_1[i], center_1[j]]  # class_1 类中心点的 i, j 两维的分量
            c_2 = [center_2[i], center_2[j]]  # class_2 类中心点的 i, j 两维的分量
            center_3 = [(c_1[0] + c_2[0]) / 2, (c_1[1] + c_2[1]) / 2]  # 两点连线的中点
            k2, b2 = calculate_vertical_line(c_1, c_2)  # 两点中垂线的斜率和截距
            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
            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
            plt.scatter(c_2[0], c_2[1], c='b', marker='x')
            plt.grid(True)  # 显示网格线
```

```
                    plt.axis('equal')   # 横纵坐标间隔大小相同
                    plt.axline(c_1, c_2, color='c', linestyle="--", label="connected line")
                    plt.axline(center_3, slope=k2, color='r', label="decision line")
                    if i == j:
                        plt.legend()   # 对角线上的子图显示出图例
                    plt.xlabel("feature " + str(i))
                    plt.ylabel("feature " + str(j))
                    plt.tight_layout()   # 自动调整子图大小，减少相互遮挡的问题
        plt.show()


# 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）
def calculate_connected_line(point_1, point_2):
        if len(point_1) != 2 or len(point_2) != 2:
            return None
        k = (point_1[1] - point_2[1]) / (point_1[0] - point_2[0])
        b = (point_1[0] * point_2[1] - point_2[0] * point_1[1]) / (point_1[0] - point_2[0])
        return k, b


# 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）
def calculate_vertical_line(point_1, point_2):
        if len(point_1) != 2 or len(point_2) != 2:
            return None
        k = -(point_1[0] - point_2[0]) / (point_1[1] - point_2[1])
        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
        return k, b
#画分割线 end

# 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
def visualization(feature, label, m, n):
        plt.figure(figsize=(10, 10), dpi=100)
        img = [[] for i in range(m*n)]
        for i in range(m):
            for j in range(n):
                img[i*n+j] = plt.subplot(m, n, i*n+j+1)
                plt.xlabel("x"+str(i))
                plt.ylabel("x"+str(j))
                plt.xlim(-1, 9)
                plt.ylim(-1, 9)
                plt.scatter(feature[:, i], feature[:, j], s=5, c=label, marker='x')
                plt.grid(True)   # 显示网格线
                plt.tight_layout()   # 自动调整子图大小，减少相互遮挡的问题
```

```
    plt.show()

# 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
def visualization_white(feature, label, m, n):
    plt.figure(figsize=(10, 10), dpi=100)
    img = [[] for i in range(m*n)]
    for i in range(m):
        for j in range(n):
            img[i*n+j] = plt.subplot(m, n, i*n+j+1)
            plt.xlabel("x"+str(i))
            plt.ylabel("x"+str(j))
            plt.xlim(-20, 20)
            plt.ylim(-20, 20)
            plt.scatter(feature[:, i], feature[:, j], s=5, c=label, marker='x')
            plt.grid(True)   # 显示网格线
            plt.tight_layout()   # 自动调整子图大小，减少相互遮挡的问题
    plt.show()

#  去除某个类别的样本，返回两个 numpy 数组
def remove_from_data(feature, label, num):
    new_feature = []
    new_label = []
    for index in range(len(label)):
        if label[index] != num:
            new_feature.append(feature[index])
            new_label.append(label[index])
    return np.asarray(new_feature), np.asarray(new_label)


#  特征白化，返回白化后的矩阵（numpy 数组格式）
#  参数为 numpy 格式的数组，其格式为数学上的矩阵的转置
def whitening(data):
  Ex=np.cov(data,rowvar=False) #Ex 为 data 的协方差矩阵
  print(Ex.shape)
  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
  #特征向量单位化
  modulus=[]
  b=np.real(b)
  for i in range(b.shape[1]):
      sum=0
      for j in range(b.shape[0]):
          sum+=b[i][j]**2
      modulus.append(sum)
  modulus=np.asarray(modulus,dtype="float64")
```

```
    b=b/modulus
    #对角矩阵 A
    a=np.real(a)
    A=np.diag(a**(-0.5))
    W=np.dot(A,b.transpose())
    X=np.dot(W,np.dot(Ex,W.transpose()))
    for i in range(W.shape[0]):
            for j in range(W.shape[1]):
                    if np.isnan(W[i][j]):
                            W[i][j]=0
    print(W)
    return np.dot(data,W)


if __name__ == '__main__':
        iris = datasets.load_iris()
        iris_data = iris.data
        iris_target = iris.target
        iris_target_names=iris.target_names
        print(iris)
        #可视化
        visualization(iris_data,iris_target,4,4)

        #去除线性不可分的最后一个
        iris_data_linear, iris_target_linear = remove_from_data(iris_data, iris_target, 2)
        visualization(iris_data_linear,iris_target_linear,4,4)
        #划分训练集、测试集
        x_train, x_test, y_train, y_test = train_test_split(iris_data_linear, iris_target_linear,
    test_size=0.3)
        meds=Medclass()
        meds.train(x_train,y_train)
        meds.performance(x_test, y_test, 0)
        #  展示每个特征两两对比图，显示决策线
        show_decision_line(x_test, y_test, meds, class_1=0, class_2=1, n=4)

        #特征白化
        iris_data_white = whitening(iris_data)
        print(iris_data_white)
        visualization_white(iris_data_white,iris_target,4,4)

        #去除线性可分的类
        #iris_data_nolinear, iris_target_nolinear = remove_from_data(iris_data, iris_target, 0) #无白
    化
        #visualization(iris_data_nolinear,iris_target_nolinear,4,4)
```

```
        iris_data_nolinear, iris_target_nolinear = remove_from_data(iris_data_white, iris_target, 0)#白化
        visualization_white(iris_data_nolinear,iris_target_nolinear,4,4)
        #划分训练集、测试集
        x_train, x_test, y_train, y_test = train_test_split(iris_data_nolinear, iris_target_nolinear, test_size=0.3)
        meds2=Medclass()
        meds2.train(x_train,y_train)
        meds2.performance(x_test, y_test, 1)
        # 展示每个特征两两对比图，显示决策线
        show_decision_line(x_test, y_test, meds2, class_1=1, class_2=2, n=4)
```

Bayess.py(第 6 题)

```
import numpy as np
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
plt.rcParams['savefig.dpi'] = 150 #图片像素
plt.rcParams['figure.dpi'] = 100 #分辨率
#贝叶斯分类器
class BayesParameter(): #存储贝叶斯分类器参数

  def __init__(self,mean,cov,category):
        self.mean=mean
        self.cov=cov
        self.category=category

class   BayesClassifier():   #贝叶斯分类器,高斯分布概率估计

  def __init__(self):
        self.parameters=[]

  def train(self,X_data,Y_data):

        for categorys in set(Y_data):#遍历每一种类别
            selected= Y_data==categorys #选中对应该类别的数据
            X_newData= X_data[selected] #得到新数据
            mean=np.mean(X_newData,axis=0) #得到均值
```

```
                cov = np.cov(X_newData.transpose())
                self.parameters.append(BayesParameter(mean,cov,categorys))

    def predit(self,data):
        res=-1
        probability=0
        for parameter in self.parameters:
            if stats.multivariate_normal.pdf(data, mean=parameter.mean,
cov=parameter.cov)>probability:
                res=parameter.category
                probability=stats.multivariate_normal.pdf(data, mean=parameter.mean,
cov=parameter.cov)
        return res


def     K_Folds_Cross_Validation(data,tar,k):
    Set=[]
    Tar=[]
    for i in range(k):
        tempSet=[]
        tempTar=[]
        tempSet.extend(data[i*10:(i+1)*10])
        tempTar.extend(tar[i*10:(i+1)*10])
        tempSet.extend(data[(i+5) * 10:(i + 6) * 10])
        tempTar.extend(tar[(i+5) * 10:(i + 6) * 10])
        tempSet.extend(data[(i+10) * 10:(i + 11) * 10])
        tempTar.extend(tar[(i+10) * 10:(i + 11) * 10])
        Set.append(tempSet)
        Tar.append(tempTar)
    return np.asarray(Set),np.asarray(Tar)

def data_visualization(data,tar):
    trainSet,testSet, trainTar,testTar    = train_test_split(data, tar, test_size=0.3)
    bc = BayesClassifier()
    bc.train(trainSet, trainTar)
    testPredict = np.array([bc.predit(x) for x in testSet],dtype="int")
    #  画图部分
    fig = plt.figure(figsize=(8, 8))
    xx = [[0, 1], [1, 2], [2, 3], [0,2],[0,3],[1,3]]
    yy = [["sepal_length", "sepal_width"],
          ["sepal_width", "petal_length"],
          ["sepal_width", "petal_width"],
          ["sepal_length","petal_length"],
          ["sepal_length ","petal_width"],
```

```
            ["sepal_width","petal_width"]]
    for i in range(6):
        ax = fig.add_subplot(321 + i)
        x_max,x_min=testSet.max(axis=0)[xx[i][0]]+0.5,testSet.min(axis=0)[xx[i][0]]-0.5
        y_max,y_min=testSet.max(axis=0)[xx[i][1]]+0.5,testSet.min(axis=0)[xx[i][1]]-0.5
        xlist = np.linspace(x_min, x_max, 80)
        ylist = np.linspace(y_min, y_max, 100)
        XX, YY = np.meshgrid(xlist, ylist)
        bc = GaussianNB()
        bc.fit(trainSet[:, xx[i]],trainTar)
        xys = [np.array([xx, yy]).reshape(1, -1) for xx, yy in zip(np.ravel(XX), np.ravel(YY))]
        zz = np.array([bc.predict(x) for x in xys])
        Z = zz.reshape(XX.shape)
        plt.contourf(XX, YY, Z, 2, alpha=.1, colors=('blue', 'red', 'green'))
        ax.scatter(testSet[testPredict == 0, xx[i][0]], testSet[testPredict == 0, xx[i][1]],
                   c='r', marker='o',
                    label="setosa")
        ax.scatter(testSet[testPredict==1, xx[i][0]], testSet[testPredict==1, xx[i][1]], c='b',
marker='x',
                    label="versicolor")
        ax.scatter(testSet[testPredict==2, xx[i][0]], testSet[testPredict==2, xx[i][1]], c='g',
marker='^',
                    label="virginica")
        ax.set_xlabel(yy[i][0])
        ax.set_ylabel(yy[i][1])
        ax.legend(loc=0)
    plt.show()

datas=datasets.load_iris()
data=datas.data
tar=datas.target
data_visualization(data,tar)

if __name__=="__main__":
    sets,tar=K_Folds_Cross_Validation(data,tar,5)
    accuracy=0
    print(tar[0].shape)
    for i in range(5): #第 i 个子集作为测试集
        x,y=0,0
        X_data,Y_data=None,None
        for j in range(5):
            if i!=j:
                if x*y==0:
                    X0_data=sets[i]
```

```
                    Y0_data=tar[i]
                else:
                    X0_data=np.concatenate((X0_data,sets[i]),axis=0)
                    Y0_data = np.concatenate((Y0_data, tar[i]), axis=0)
                    x+=1
                    y+=1

        bc= BayesClassifier()
        bc.train(X0_data,Y0_data)

        y_predict=[bc.predit(x) for x in sets[i]]
        tempAccuracy=np.sum(y_predict==tar[i])/tar[i].shape[0]
        accuracy+=tempAccuracy

    accuracy=accuracy/5
print("accuracy:",accuracy)
```