实验一：Fisher 辨别分析

1 问题描述

使用 Fisher 线性判别处理分类问题，并在 UCI 数据集上的 Iris 和 sonar 数据上验证算法的有效性，其中，Iris 数据 3 类，4 维，150 个数据；sonar 数据 2 类，60 维，208 个样本。

2 数据集说明

UCI 数据库是加州大学欧文分校 (University of CaliforniaIrvine) 提出的用于机器学习的数据库，这个数据库目前共有 488 个数据集，其数目还在不断增加，UCI 数据集是一个常用的标准测试数据集。

下表为部分 Iris 数据展示：

5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
...				
7	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
...				
6.3	3.3	6	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3	5.9	2.1	virginica

3 Fisher 原理分析

判别分析是一种经典的现行分析方法，其利用已知类别的样本建立判别模型，对未知类别的样本进行分类。

3.1 Fisher 思想

Fisher 的主要思想是投影，选择一个适当的投影轴，使所有的样本点都投影到这个轴上得到一个投影值，将多维问题转化为一维问题来处理。在计算投影轴的方向时，需保证每一类内的投影距离尽可能小，而不同类的投影距离尽可能大。

3.2 公式推导

Fisher 线性判别需根据实际数据找到一条最易于分类的投影方向。

- 在 n 维 X 空间
  1. 各类样本的均值向量 $\mu_i : \mu_i = \frac{1}{N_i} \sum_{\mathbf{x}_j \in D_i} \mathbf{x}_j, i = 1, 2$
  2. 样本类内离散度矩阵 S_i 和总样本类内离散度矩阵 S_w

$$S_i = \sum_{x_j \in D_i} (x_j - \mu_i)(x_j - \mu_i)^T, i = 1, 2$$

$$S_w = S_1 + S_2$$

3. 样本类间离散度矩阵 S_b

$$S_b=(\mu_1-\mu_2)(\mu_1-\mu_2)^T$$

- 在一维 Y 空间
  1. 各类样本的均值向量 $\overline{\mu_i}:\bar{\mu}_i=\frac{1}{N_i}\sum_{y_j\in D_i'}y_j, i=1,2$
  2. 样本类内离散度矩阵 $\overline{S_i}$ 和总样本类内离散度矩阵 $\overline{S_w}$

$$\bar{S}_i^2=\sum_{y_j\in D_i'}(y_j-\bar{\mu}_i)^2, i=1,2$$

$$\bar{S}_w=\bar{S}_1^2+\bar{S}_2^2$$

  3. 样本类间离散度矩阵 $\overline{S_b}$

$$\bar{S}_b=(\bar{\mu}_1-\bar{\mu}_2)^2$$

目标：投影后，在一维 Y 空间中各类样本尽可能分得开些，即使原样本向量在该方向上的投影能兼顾类间分布尽可能分开，类内样本投影尽可能密集的要求。

由此构造 Fisher 准则函数为

$$J(\mathbf{w})=\frac{\bar{S}_b}{\bar{S}_w}=\frac{(\bar{\mu}_1-\bar{\mu}_2)^2}{\bar{S}_1^2+\bar{S}_2^2}$$

Fisher 最佳投影方向的求解为

$$\mathbf{w}^*=argmaxJ(\mathbf{w})$$

将 $J(\mathbf{w})$ 变成 w 的显函数之后,

由各类样本均值可推出：

$$\bar{\mu}_i=\frac{1}{N_i}\sum_{y_j\in D_i'}y_j=\frac{1}{N_i}\sum_{\mathbf{x}_j\in D_i}\mathbf{w}^T\mathbf{x}_j=\mathbf{w}^T\left(\frac{1}{N_i}\sum_{\mathbf{x}_j\in D_i}\mathbf{x}_j\right)=\mathbf{w}^T\mu_i$$

投影样本均值之差可以展开为：

$$\begin{aligned}&(\bar{\mu}_1-\bar{\mu}_2)^2=\left(\mathbf{w}^T\mu_1-\mathbf{w}^T\mu_2\right)^2\\&=\mathbf{w}^T(\mu_1-\mu_2)(\mu_1-\mu_2)^T\mathbf{w}\\&=\mathbf{w}^T\mathbf{S}_b\mathbf{w}\end{aligned}$$

由类内散布矩阵可推出：

$$\begin{aligned}&\bar{S}_i^2=\textstyle\sum_{y_j\in D_i'}(y_j-\bar{\mu}_i)^2=\sum_{\mathbf{x}_j\in D_i}\left(\mathbf{w}^T\mathbf{x}_j-\mathbf{w}^T\mu_i\right)^2\\&=\mathbf{w}^T\left[\textstyle\sum_{\mathbf{x}_j\in D_i}(\mathbf{x}_j-\mu_i)(\mathbf{x}_j-\mu_i)^T\right]\mathbf{w}\\&=\mathbf{w}^T\mathbf{S}_i\mathbf{w}\end{aligned}$$

于是有：

$$\bar{S}_1^2+\bar{S}_2^2=\mathbf{w}^T(\mathbf{S}_1+\mathbf{S}_2)\mathbf{w}=\mathbf{w}^T\mathbf{S}_w\mathbf{w}$$

准则函数可以写为：

$$J(\mathbf{w})=\frac{\mathbf{w}^T\mathbf{S}_b\mathbf{w}}{\mathbf{w}^T\mathbf{S}_w\mathbf{w}}$$

要求使 J(w) 最大的 w，可以采用 Lagrange 乘子法求解。

假设分母等于非零常数，定义 Lagrange 函数为：

$$L(\mathbf{w},\lambda)=\mathbf{w}^TS_b\mathbf{w}-\lambda\left(\mathbf{w}^TS_w\mathbf{w}-c\right)$$

对 w 求偏导数，令偏导数为 0：

$$\frac{\partial L(\mathbf{w},\lambda)}{\partial \mathbf{w}} = 2\mathbf{S}_b\mathbf{w} - 2\lambda\mathbf{S}_w\mathbf{w} = 0$$

即：

$$\mathbf{S}_w^{-1}\mathbf{S}_b\mathbf{w}^* = \lambda\mathbf{w}^*$$

化简得：

$$\mathbf{w}^* = \frac{1}{\lambda}\mathbf{S}_w^{-1}\mathbf{S}_b\mathbf{w}^* = \frac{1}{\lambda}\mathbf{S}_w^{-1}(\mu_1-\mu_2)(\mu_1-\mu_2)^T\,\mathbf{w}^*$$

该式中，$(\mu_1-\mu_2)^T\,\mathbf{w}^*$ 和 λ 为标量，对于投影向量，我们仅需考虑它的方向，因此

$$\mathbf{w}^* = \mathbf{S}_w^{-1}(\mu_1-\mu_2)$$

最后通过 $\mathrm{y} = w^{*T}x$ 对样本集合作线性变换得到 n 个样本投影后的样本值 y_1,y_2...y_n，对这些投影之后的样本值进行分类，确定一个阈值 w_0，分类规则如下：

$$\mathrm{y} > w_0 \rightarrow x \in \omega_1$$
$$\mathrm{y} < w_0 \rightarrow x \in \omega_2$$

阈值可以通过下面三种方法进行取值：

$$w_0 = \frac{\bar{\mu}_1+\bar{\mu}_2}{2}$$

$$w_0 = \frac{N_1\bar{\mu}_1+N_2\bar{\mu}_2}{N_1+N_2}$$

$$w_0 = \frac{\bar{\mu}_1+\bar{\mu}_2}{2} + \frac{\ln\left[P(\omega_1)/P(\omega_2)\right]}{N_1+N_2-2}$$

本次实验阈值根据第一种方式进行选取。

4 实验结果

4.1 Iris 数据集（3 类，4 维）

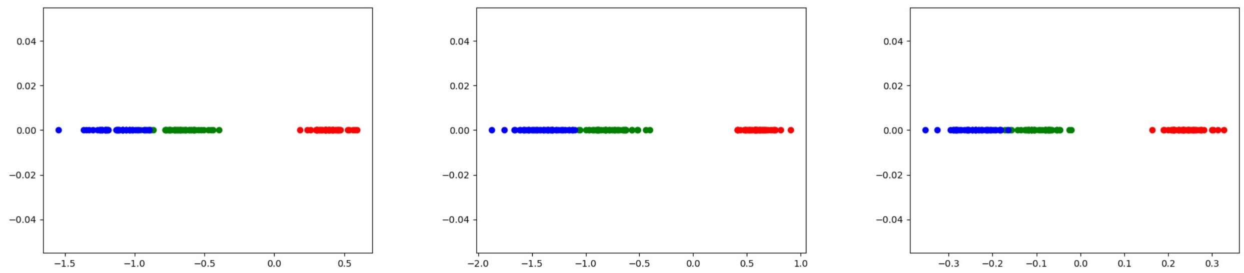
本次实验对 Iris 数据集采用了三种方法求解分类准确率，结果如下表所示：

表 1: Iris 数据集三种分类方法的准确率

分类方法	准确率
留出法	97.11%
10 折交叉验证法	97.33%
留一法	98.00%

其中，留出法对训练集和测试集按照 7/3 的比例进行随机划分，计算 10 次准确率之后求取平均值。

由于 fish 线性判别单次运行只能投影两类样本，因此，建立了三个最佳投影向量，使数据能够进行两两投影比较。Iris 数据集分别在三个投影向量上的投影结果如下图所示。



(a) 在第一投影方向上的投影 (b) 在第二投影方向上的投影 (c) 在第三投影方向上的投影

图 1: Iris 数据集在三类投影方向上的投影

从图中可以看出，Iris 数据集在三个方向上投影分隔比较明显，重叠部分很少，这印证了该分类算法在 Iris 数据集上准确率很高。

4.2 sonar 数据集（2 类，60 维）

本次实验对 sonar 数据集采用了三种方法求解分类准确率，结果如下表所示：

表 2: sonar 数据集三种分类方法的准确率

分类方法	准确率
留出法	72.22%
10 折交叉验证法	59.19%
留一法	75.00%

其中，留出法对训练集和测试集按照 7/3 的比例进行随机划分，计算 10 次准确率之后求取平均值。

从上表中可以发现，对于 K 折交叉验证法，取 K 为 10 时，准确率明显较低。当增大 K 时，准确率得到明显提高，当 K 取值为 80 时，K 折交叉验证法准确率逼近留一法。考虑到留一法是特殊的 K 折交叉验证法（即 K=N-1），说明当样本量较大时，适当增大 K 值有利于提升该分类算法的准确率。

sonar 数据集在投影向量上的投影结果如下图所示。

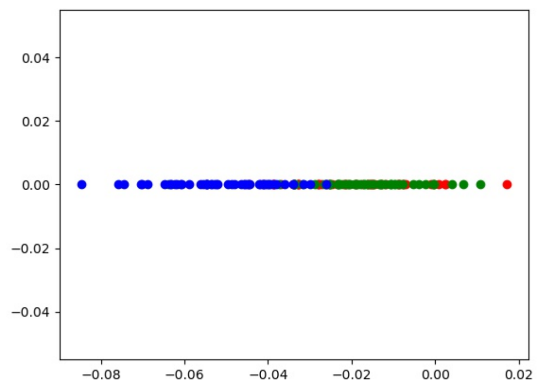


图 2: sonar 数据集在三类投影方向上的投影

从图中可以看出，三类样本无明显间隔区分，部分混叠严重，这也印证了该分类算法在 sonar 数据集上准确率不高。

4.3 分类准确率和维度之间的关系

为了进一步探究分类准确率和维度之间的关系，本实验选取了 60 维的 sonar 数据集作为研究对象，采取了分类准确率最高的留一法，得到结果如下图所示：

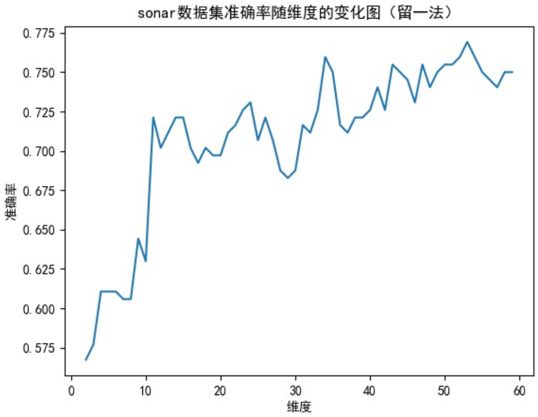


图 3: sonar 数据集上分类准确率和维度之间的关系

从图中可以得出结论，随着维度的升高，该算法的分类准确率也随之增大，但分类准确率的增大幅度逐渐减缓，并且，增大幅度具有波动性。

维度的升高，会使算法的计算成本增大，因此，可以在图中选取一个拐点作为最佳维度的选取点。对于 sonar 数据集来说，当维度为 11 时，准确率已经达到了 72.12%, 在这一点之后，准确率的增大幅度趋于缓慢。因此，在本实验中，维度最适合的取值为 11。

5 程序代码

对 Iris 数据集分类的代码如下：

```
import numpy as np
from sklearn.model_selection import train_test_split, KFold, LeaveOneOut
import matplotlib.pyplot as plt


def load_dataset():
data = np.genfromtxt('iris.txt', delimiter=',', usecols=(0, 1, 2, 3))
target = np.genfromtxt('iris.txt', delimiter=',', usecols=(4), dtype=str)
t = np.zeros(len(target))
t[target == 'setosa'] = 1
t[target == 'versicolor'] = 2
t[target == 'virginica'] = 3
return data, t


def fisher(class1, class2):
class1 = np.mat(class1)
class2 = np.mat(class2)
```

```
# 求解每一个特征的均值，按列求解
a1 = np.mean(class1, axis=0)
a2 = np.mean(class2, axis=0)

# 直接代入公式求解类内离散度矩阵
s1 = (class1 - a1).T * (class1 - a1)
s2 = (class2 - a2).T * (class2 - a1)
sw = s1 + s2
# 这里是求解离散度矩阵的另一种思路：通过协方差公式求解，49为样本数量-1(n-1)
# s = np.cov(class0.T) * 49

# w 为最佳变换向量w*，w0为阈值
w = (a1 - a2) * np.linalg.inv(sw)
w0 = (a1 * w.T + a2 * w.T) / 2
return w, w0


# 计算分类准确率
def accuracy(pre, tar):
total = len(pre)
acc = 0
for i in range(total):
if pre[i] == tar[i]:
acc += 1
return acc / total


# 修改三个类别标签
def transform_target(data, target):
class1 = []
class2 = []
class3 = []
for i in range(len(data)):
if target[i] == 1:
class1.append(data[i])
elif target[i] == 2:
class2.append(data[i])
else:
class3.append(data[i])
return class1, class2, class3
```

```
# method1 留出法，随机划分训练测试集，多次平均求结果
def method1():
data, target = load_dataset()

# 使用留出法随机划分数据集，训练集/测试集=7/3，每次划分具有随机性
X_train, X_test, Y_train, Y_test = train_test_split(data, target,
    test_size=0.30)

class1, class2, class3 = transform_target(X_train, Y_train)
# w12代表第一类和第二类比较的投影向量，w012代表第一类和第二类比较时的阈
    值，其它同理。
w12, w012 = fisher(class1, class2)
w13, w013 = fisher(class1, class3)
w23, w023 = fisher(class2, class3)

# 3分类的比较思路：两两进行比较，若两次均分类正确才算正确
y12 = X_test * w12.T
y13 = X_test * w13.T
y23 = X_test * w23.T
res = np.zeros(len(X_test))
for i in range(len(res)):
if y12[i] > w012 and y12[i] > w013:
res[i] = 1
if y12[i] < w012 and y23[i] > w023:
res[i] = 2
if y13[i] < w013 and y23[i] < w023:
res[i] = 3
# print(res)
acc = accuracy(res, Y_test)
# print("分类准确率为", acc)
return acc


# method2 k折交叉验证法
def method2():
data, target = load_dataset()
acc = 0
K = 10  # 这里设定k为10
kf = KFold(n_splits=K)
for train_index, test_index in kf.split(data):
X_train = data[train_index]
X_test = data[test_index]
Y_train = target[train_index]
Y_test = target[test_index]
```

```
class1, class2, class3 = transform_target(X_train, Y_train)
# w12代表第一类和第二类比较的投影向量，w012代表第一类和第二类比较时的阈值，其它同理。
w12, w012 = fisher(class1, class2)
w13, w013 = fisher(class1, class3)
w23, w023 = fisher(class2, class3)

# 3分类的比较思路：两两进行比较，若两次均分类正确才算正确
y12 = X_test * w12.T
y13 = X_test * w13.T
y23 = X_test * w23.T
res = np.zeros(len(X_test))
for i in range(len(res)):
if y12[i] > w012 and y12[i] > w013:
res[i] = 1
if y12[i] < w012 and y23[i] > w023:
res[i] = 2
if y13[i] < w013 and y23[i] < w023:
res[i] = 3
# print(res)
acc += accuracy(res, Y_test)
# print("分类准确率为", acc)
acc = acc / K
return acc

# method3 留一法
def method3():
data, target = load_dataset()
loo = LeaveOneOut()
acc = 0
for train_index, test_index in loo.split(data):
X_train = data[train_index]
X_test = data[test_index]
Y_train = target[train_index]
Y_test = target[test_index]
class1, class2, class3 = transform_target(X_train, Y_train)
# w12代表第一类和第二类比较的投影向量，w012代表第一类和第二类比较时的阈值，其它同理。
w12, w012 = fisher(class1, class2)
w13, w013 = fisher(class1, class3)
w23, w023 = fisher(class2, class3)

# 3分类的比较思路：两两进行比较，若两次均分类正确才算正确
y12 = X_test * w12.T
```

```
y13 = X_test * w13.T
y23 = X_test * w23.T
res = np.zeros(len(X_test))
for i in range(len(res)):
if y12[i] > w012 and y12[i] > w013:
res[i] = 1
if y12[i] < w012 and y23[i] > w023:
res[i] = 2
if y13[i] < w013 and y23[i] < w023:
res[i] = 3
# print(res)
acc += accuracy(res, Y_test)
# print("分类准确率为", acc)
acc = acc / len(data)
return acc

# 绘制投影图
def draw():
data, target = load_dataset()
class1, class2, class3 = transform_target(data, target)
# w12代表第一类和第二类比较的投影向量，w012代表第一类和第二类比较时的阈
    值，其它同理。
w12, w012 = fisher(class1, class2)
w13, w013 = fisher(class1, class3)
w23, w023 = fisher(class2, class3)

# 3分类的比较思路：两两进行比较，若两次均分类正确才算正确
y12 = data * w12.T
y13 = data * w13.T
y23 = data * w23.T

# y12方向上的投影
plt.figure(1)
plt.plot(y12[0:49], np.zeros([49, 1]), 'ro')
plt.plot(y12[50:99], np.zeros([49, 1]), 'go')
plt.plot(y12[100:149], np.zeros([49, 1]), 'bo')
plt.savefig('./iris-1.jpg')
plt.show()


# y13方向上的投影
plt.figure(2)
plt.plot(y13[0:49], np.zeros([49, 1]), 'ro')
plt.plot(y13[50:99], np.zeros([49, 1]), 'go')
```

```
plt.plot(y13[100:149], np.zeros([49, 1]), 'bo')
plt.savefig('./iris-2.jpg')
plt.show()


# y23方向上的投影
plt.figure(3)
plt.plot(y23[0:49], np.zeros([49, 1]), 'ro')
plt.plot(y23[50:99], np.zeros([49, 1]), 'go')
plt.plot(y23[100:149], np.zeros([49, 1]), 'bo')
plt.savefig('./iris-3.jpg')
plt.show()



def main():
# 10次计算方法一的留出法，取平均准确率作为结果(保留两位小数输出)
total_accuary = 0
for i in range(10):
total_accuary += method1()
total_accuary = total_accuary / 10
print("留出法的分类准确率为：", "{:.2%}".format(total_accuary))
draw()
total_accuary2 = method2()
print("10折交叉验证法的分类准确率为：", "{:.2%}".format(total_accuary2))
total_accuary3 = method3()
print("留一法的分类准确率为：", "{:.2%}".format(total_accuary3))

if __name__ == '__main__':
        main()
```

对 sonar 数据集分类的代码如下：

```
import numpy as np
from sklearn.model_selection import train_test_split, KFold, LeaveOneOut
import matplotlib.pyplot as plt


# 正常导入数据
def load_dataset():
data = np.genfromtxt('sonar.txt', delimiter=',', usecols=np.arange(0, 59)
    )
target = np.genfromtxt('sonar.txt', delimiter=',', usecols=(60), dtype=
    str)
t = np.zeros(len(target))
```

```
t[target == 'R'] = 1
t[target == 'M'] = 2
return data, t


# 自定义导入数据维度
def load_dataset_dimension(dimension):
data = np.genfromtxt('sonar.txt', delimiter=',', usecols=np.arange(0,
    dimension))
target = np.genfromtxt('sonar.txt', delimiter=',', usecols=(60), dtype=
    str)
t = np.zeros(len(target))
t[target == 'R'] = 1
t[target == 'M'] = 2
return data, t


def fisher(class1, class2):
class1 = np.mat(class1)
class2 = np.mat(class2)

# 求解每一个特征的均值，按列求解
a1 = np.mean(class1, axis=0)
a2 = np.mean(class2, axis=0)

# 直接代入公式求解类内离散度矩阵
s1 = (class1 - a1).T * (class1 - a1)
s2 = (class2 - a2).T * (class2 - a1)
sw = s1 + s2
# 这里是求解离散度矩阵的另一种思路：通过协方差公式求解，49为样本数量-1(n
    -1)
# s = np.cov(class0.T) * 49

# w 为最佳变换向量w*，w0为阈值
w = (a1 - a2) * np.linalg.inv(sw)
w0 = (a1 * w.T + a2 * w.T) / 2
return w, w0


# 计算分类准确率
def accuracy(pre, tar):
total = len(pre)
acc = 0
for i in range(total):
```

```
if pre[i] == tar[i]:
acc += 1
return acc / total


# 修改两个类别标签
def transform_target(data, target):
class1 = []
class2 = []
for i in range(len(data)):
if target[i] == 1:
class1.append(data[i])
elif target[i] == 2:
class2.append(data[i])
return class1, class2


# method1 留出法，随机划分训练测试集，多次平均求结果
def method1():
data, target = load_dataset()

# 使用留出法随机划分数据集，训练集/测试集=7/3，每次划分具有随机性
X_train, X_test, Y_train, Y_test = train_test_split(data, target, test_size=0.30)

class1, class2 = transform_target(X_train, Y_train)
# w代表投影向量，w0代表第一类和第二类比较时的阈值。
w, w0 = fisher(class1, class2)

y = X_test * w.T
res = np.zeros(len(X_test))
for i in range(len(res)):
if y[i] > w0:
res[i] = 1
else:
res[i] = 2
# print(res)
acc = accuracy(res, Y_test)
# print("分类准确率为", acc)
return acc


# method2 k折交叉验证法
def method2():
```

```
data, target = load_dataset()
acc = 0
K = 10 # 这里设定k为10
kf = KFold(n_splits=K)
for train_index, test_index in kf.split(data):
X_train = data[train_index]
X_test = data[test_index]
Y_train = target[train_index]
Y_test = target[test_index]
class1, class2 = transform_target(X_train, Y_train)
# w代表投影向量，w0代表第一类和第二类比较时的阈值。
w, w0 = fisher(class1, class2)

y = X_test * w.T
res = np.zeros(len(X_test))
for i in range(len(res)):
if y[i] > w0:
res[i] = 1
else:
res[i] = 2
# print(res)
acc += accuracy(res, Y_test)
# print("分类准确率为", acc)
acc = acc / K
return acc


# method3 留一法
def method3():
data, target = load_dataset()
loo = LeaveOneOut()
acc = 0
for train_index, test_index in loo.split(data):
X_train = data[train_index]
X_test = data[test_index]
Y_train = target[train_index]
Y_test = target[test_index]
class1, class2 = transform_target(X_train, Y_train)
# w代表投影向量，w0代表第一类和第二类比较时的阈值。
w, w0 = fisher(class1, class2)

y = X_test * w.T
res = np.zeros(len(X_test))
for i in range(len(res)):
```

```
            if y[i] > w0:
                res[i] = 1
            else:
                res[i] = 2
        # print(res)
        acc += accuracy(res, Y_test)
        # print("分类准确率为", acc)
    acc = acc / len(data)
    return acc


# testdension 以留一法为基础，测试维度和准确率的关系
def testdension(dimension):
    data, target = load_dataset_dimension(dimension)
    loo = LeaveOneOut()
    acc = 0
    for train_index, test_index in loo.split(data):
        X_train = data[train_index]
        X_test = data[test_index]
        Y_train = target[train_index]
        Y_test = target[test_index]
        class1, class2 = transform_target(X_train, Y_train)
        # w代表投影向量，w0代表第一类和第二类比较时的阈值。
        w, w0 = fisher(class1, class2)

        y = X_test * w.T
        res = np.zeros(len(X_test))
        for i in range(len(res)):
            if y[i] > w0:
                res[i] = 1
            else:
                res[i] = 2
        # print(res)
        acc += accuracy(res, Y_test)
        # print("分类准确率为", acc)
    acc = acc / len(data)
    return acc


# 绘制投影图
def draw():
    data, target = load_dataset()
    class1, class2 = transform_target(data, target)

```

```
w, w0 = fisher(class1, class2)
y = data * w.T

plt.figure(1)
plt.plot(y[0:49], np.zeros([49, 1]), 'ro')
plt.plot(y[50:99], np.zeros([49, 1]), 'go')
plt.plot(y[100:149], np.zeros([49, 1]), 'bo')
plt.savefig('./sonar.jpg')
plt.show()


def main():
# 10次计算方法一的留出法，取平均准确率作为结果(保留两位小数输出)
total_accuary1 = 0
for i in range(10):
total_accuary1 += method1()
total_accuary1 = total_accuary1 / 10
print("留出法的分类准确率为：", "{:.2%}".format(total_accuary1))
# draw()
total_accuary2 = method2()
print("K折交叉验证法的分类准确率为：", "{:.2%}".format(total_accuary2))
total_accuary3 = method3()
print("留一法的分类准确率为：", "{:.2%}".format(total_accuary3))


# 绘制维度与准确率的关系图
total_accuary = []
plt.figure(2)
for demension in range(2, 60):
total_accuary.append(testdension(demension))
print(total_accuary)
plt.plot(np.arange(2, 60), total_accuary)
# 解决中文显示问题
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
plt.xlabel("维度")
plt.ylabel("准确率")
plt.title("sonar数据集准确率随维度的变化图（留一法）")
plt.savefig('./demension.jpg')
plt.show()



if __name__ == '__main__':
```

```
main()
```