模式识别 Fisher 线性判别

一、Fisher 线性判别算法介绍

1.1 介绍

Fisher 两类的判别问题可以看作是把所有样本都投影到一个方向上，然后在这个一维空间中确定一个分类的阈值。过这个阈值点且与投影方向垂直的超平面就是两类的分类面。问题是如何根据实际情况找到这条最好的、最易于分类的投影线，这就是 Fisher 线性判别算法要解决问题。

Fisher 线性判别的思想就是：选择投影方向，使投影后两类相隔尽可能远，而同时每一类内部的样本又尽可能聚集。以下部分仅讨论两类问题。

1.2 Fisher 准则函数中的基本参量

（1）样本

① 训练样本集是(每个样本是一个 d 维向量)：

$$\chi=\{\boldsymbol{x_1},\boldsymbol{x_2},\cdots,\boldsymbol{x_N}\},$$

② 其中ω1类的样本是：

$$\chi_1=\{\boldsymbol{x_1^1},\boldsymbol{x_2^1},\cdots,\boldsymbol{x_{N_1}^1}\}$$

③ 其中ω1类的样本是：

$$\chi_2=\{\boldsymbol{x_1^2},\boldsymbol{x_2^2},\cdots,\boldsymbol{x_{N_2}^2}\}$$

目标:寻找一个投影 w(w 也是一个 d 维列向量),使得投影后的样本变成：

$$y_i=\boldsymbol{w^T x_i},\ \ \mathrm{i}=1,2,\cdots,\mathrm{N}$$

（2）在原来的样本空间

① **类均值向量**为:

$$\boldsymbol{m_i} = \frac{1}{N_i}\sum_{x_j\in\chi_i}\boldsymbol{x_j},\ \ i = 1,2$$

② **各类的类内离散度矩阵**(within-class scatter matrix)为:

$$\boldsymbol{S_i} = \sum_{x_j\in\chi_i}(\boldsymbol{x_j}-\boldsymbol{m_i})(\boldsymbol{x_j}-\boldsymbol{m_i})^{\boldsymbol{T}},\ \ i = 1,2$$

③ **总类内离散度**矩阵(pooled within-class scatter matrix)为:

$$\boldsymbol{S_w} = \boldsymbol{S_1}+\boldsymbol{S_2}$$

④ 类间离散度矩阵(between-class scatter matrix)定义为:

$$\boldsymbol{S_i} = (\boldsymbol{m_1}-\boldsymbol{m_2})(\boldsymbol{m_1}-\boldsymbol{m_2})^T$$

(3) 在投影以后的一维空间

① 两类均值分别为:

$$\widetilde{m}_i = \frac{1}{N_i}\sum_{y_j\in\zeta_i}y_i = \frac{1}{N_i}\sum_{x_j\in\chi_i}\boldsymbol{w^T x_j} = \boldsymbol{w^T m_i},\ \ i = 1,2$$

② 类内离散度矩阵为:

$$\tilde{S}_i^2 = \sum_{y_j\in\zeta_i}(y_i-\widetilde{m}_i)^2,\ \ i = 1,2$$

③ 总类内离散度矩阵为:

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

⑤ 类间离散度矩阵为:

$$\tilde{S}_b = (\widetilde{m}_1-\widetilde{m}_2)^2$$

1.3 衡量标准与分类

两类判别，就是希望寻找的投影方向使投影以后两类尽可能分开，而各类内部又尽可能聚集，这一目标可以表示成如下的准则：

$$max\ \boldsymbol{J_F}(\boldsymbol{\omega})=\frac{\tilde{S}_b}{\tilde{S}_w}=\frac{(\tilde{\boldsymbol{m}}_1+\tilde{\boldsymbol{m}}_2)^2}{\tilde{\boldsymbol{S}}_1+\tilde{\boldsymbol{S}}_2}$$

这就是 Fisher 准则函数(Fisher's Criterion)，可变换为：

$$max\ \boldsymbol{J_F}(\boldsymbol{\omega})=\frac{\boldsymbol{w^T S_b w}}{\boldsymbol{w^T S_w w}}$$

可求解得 Fisher 判别准则下的**最佳投影方向**：

$$\boldsymbol{w^*}=\boldsymbol{S_w^{-1}}(\boldsymbol{m_1}-\boldsymbol{m_2})$$

若不考虑先验概率，**阈值**w_0可按以下规则选取：

$$\omega_0=-\frac{1}{2}(\tilde{m}_1+\tilde{m}_2)=-\frac{1}{2}(\boldsymbol{w^T m_1}+\boldsymbol{w^T m_2})$$

两类线性判别的一般**决策规则**为：

$$\text{若}g(x)=\boldsymbol{w^T x}+w_0\begin{cases}>0,\ \text{则}x\in\omega_1\\<0,\ \text{则}x\in\omega_2\end{cases}$$

若考虑先验概率，决策规则可以写成：

$$\text{若}g(x)=\boldsymbol{w^T}(\boldsymbol{x}-\frac{1}{2}(\boldsymbol{m_1}+\boldsymbol{m_2}))\begin{cases}>\log\frac{P(\omega_1)}{P(\omega_2)},\ \text{则}x\in\omega_1\\<\log\frac{P(\omega_1)}{P(\omega_2)},\ \text{则}x\in\omega_2\end{cases}$$

在此处键入公式。

二、实验数据集介绍

2.1 Iris 数据集介绍

<table>
<tr><th colspan="4">Iris 数据集</th></tr>
<tr><td>类别</td><td colspan="3">3</td></tr>
<tr><td>维度</td><td colspan="3">4</td></tr>
<tr><td rowspan="3">数据长度</td><td rowspan="3">150</td><td>50</td><td>Iris-setosa</td></tr>
<tr><td>50</td><td>Iris-versicolor</td></tr>
<tr><td>50</td><td>Iris-setosa</td></tr>
</table>

2.2 Sonar 数据集介绍

<table>
<tr><th colspan="4">Sonar 数据集</th></tr>
<tr><td>类别</td><td colspan="3">2</td></tr>
<tr><td>维度</td><td colspan="3">60</td></tr>
<tr><td rowspan="2">数据长度</td><td rowspan="2">208</td><td>97</td><td>R</td></tr>
<tr><td>111</td><td>M</td></tr>
</table>

三、实验设置

对两类数据集，sonar 是两分类问题，直接按决策公式判别；iris 是三类问题，需要求出样本之间两两组合的判别函数，按照第二种情况进行三分类（每个模式类和其他模式类之间分别用判别点分开）。

两类数据集处理方法具有相似性，因此实验设置基本一致，具体如下：

1、读取数据信息。首先从数据集文件（iris.data、sonar.all-data）中读取数据，将读取到的数据按照一定比例（默认 2/5 测试）**随机**存放到训练集和测试集中，再将训练集中数据按标签分类（iris 分三类、sonar 分两类）。

2、求各类样本的基本参量。根据 1.2（2）中的①②③三个公式，分别求解类样本的均值向量、类内离散度矩阵、两两样本间总类内离散度矩阵（iris 中需要求解$\boldsymbol{S}_{w_12}$ $\boldsymbol{S}_{w\_13}$ $\boldsymbol{S}_{w\_23}$，sonar 中求解$\boldsymbol{S}_{w_12}$）。

3、求解权向量和阈值。根据 1.3 中相应公式，求解样本间的权向量和阈值（iris 三类样本，需要求解两两样本间的参数，一共求解三次；sonar 只需要求解一次），求解完毕之后即可按照第二类情况规则进行分类。

4、绘图验证训练效果。利用判别函数分别将各类样本降至一维，按权向量方向投影至坐标轴，并分颜色绘制各点位置（iris 分别按照三个投影方向绘制三次），观察不同色点的分布情况，观察训练效果。

5、用测试集测试并计算准确率，绘图显示分类效果。

6、重复 1-5 步 20 次，计算准确率，绘图显示。

四、实验结果展示与分析

4.1 Iris 数据集分类结果分析

1、首先将三类样本数据两两组合，利用权向量和阈值将 4 维数据降至 1 维，在坐标轴上进行绘制，结果展示如下：

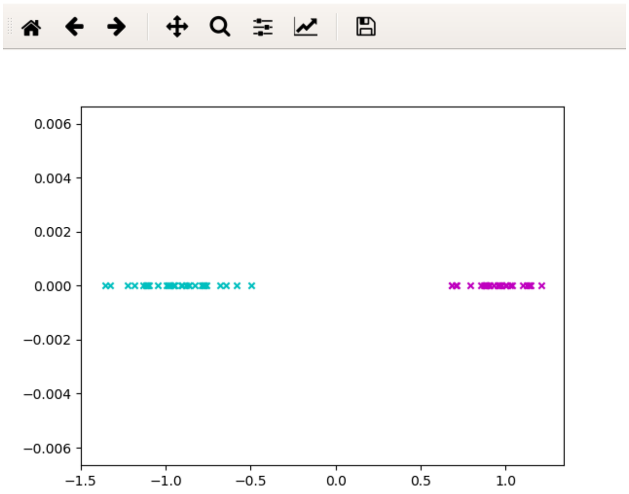


图 4.1：Setosa 与 Versicolor 数据训练分类结果

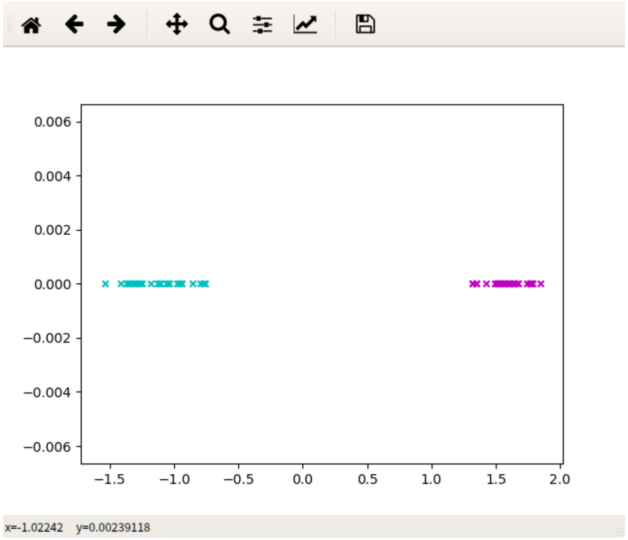


图 4.2：Setosa 与 Virginica 数据训练分类结果

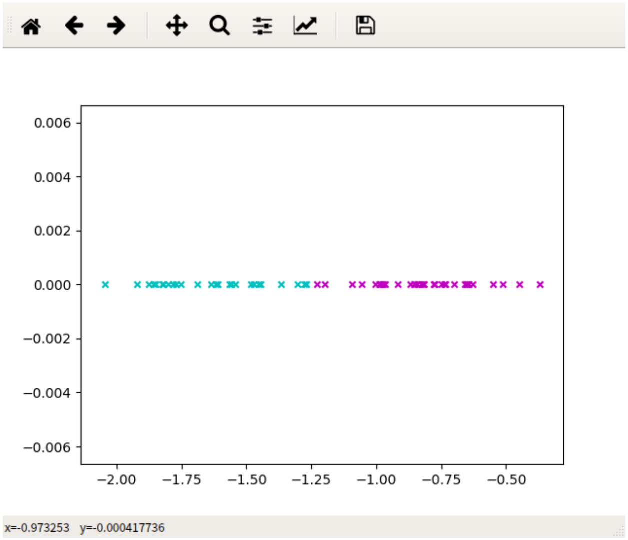


图 4.3:Versicolor 与 Virginica 数据训练分类结果

从上述图示可以看出：经训练后，两两样本投影至一维分别聚集到轴的两侧，可以寻找到非常明确的分类点分开两类样本。

2、将训练样本分别按 G_{12}、G_{13}、G_{23}（三类样本两两之间的判别函数）进行计算，按照 1.3 中判别规则进行判别，并将降维后的数据分别投影在 Y=3、Y=2、Y=1 三条轴上，通过青（3）、紫（2）、蓝（1）三类数据区别三类数据，

分类点均为 X=0，结果如下图：

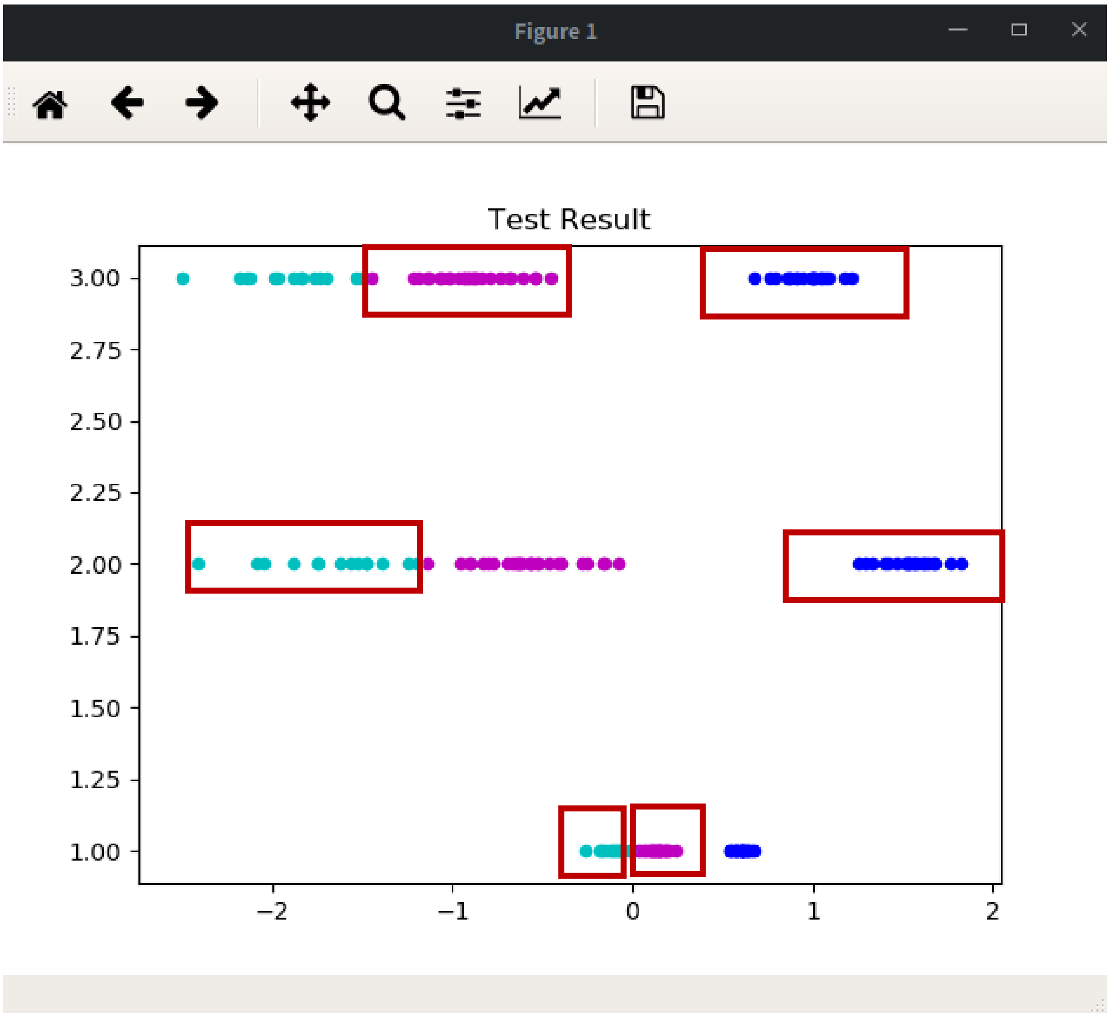
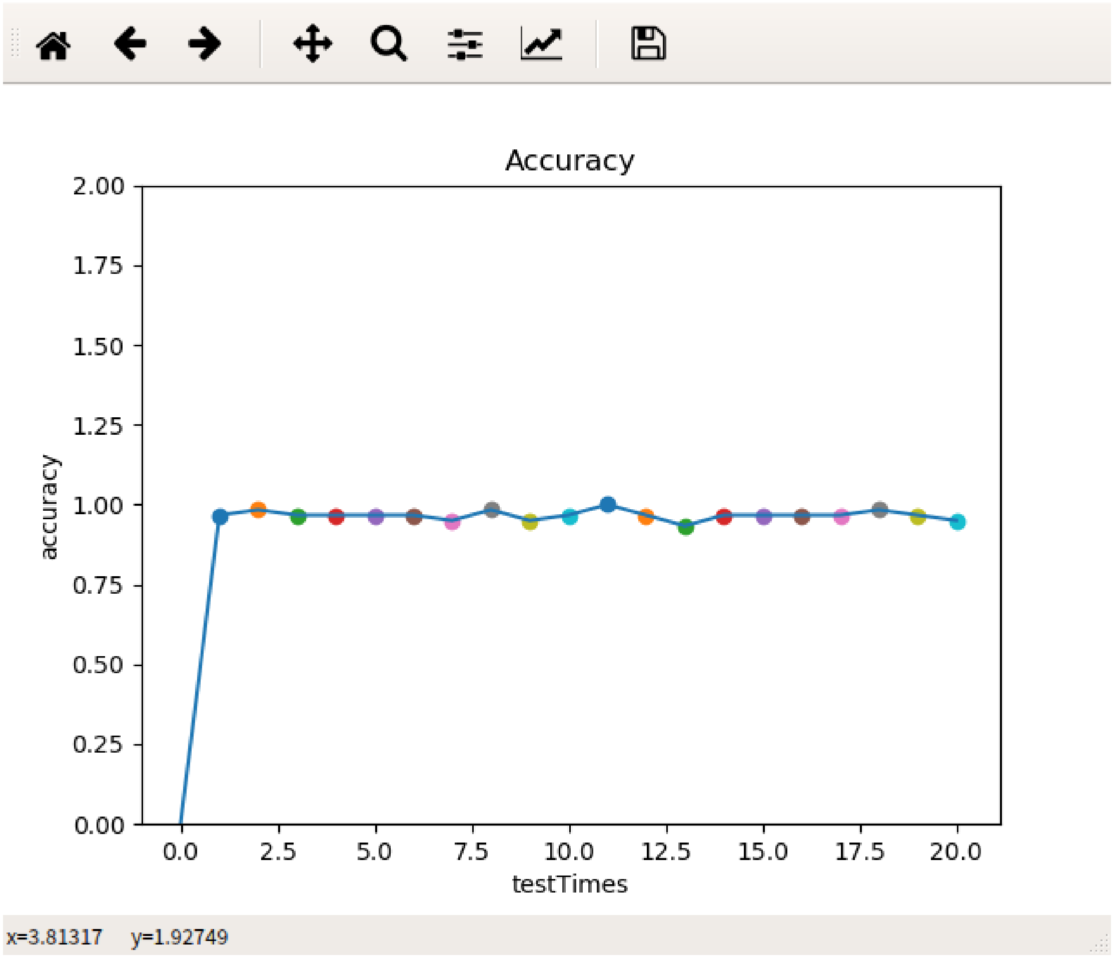


图 4.4：分别按三类判别函数将测试数据投影的结果

从上图可以看出按照相对应的判别函数 g，两类测试样本均能在分类点 X=0 两侧很好的区分开来，甚至也能够区分第三类数据。

3、根据第二类分类情况（$G_{12}>0$、$G_{13}>0$，则属于第一类；$G_{12}<0$、$G_{23}>0$，则属于第二类，$G_{23}<0$、$G_{13}<0$，则属于第三类）的分类规则，对测试样本进行分类，对比已知标签，分类正确和错误次数，计算正确率，重新测试二十次，计算平均准确率。结果如下




```
1 :Accuracy: 0.9666666666666667
2 :Accuracy: 0.9833333333333333
3 :Accuracy: 0.9666666666666667
4 :Accuracy: 0.9666666666666667
5 :Accuracy: 0.9666666666666667
6 :Accuracy: 0.9666666666666667
7 :Accuracy: 0.95
8 :Accuracy: 0.9833333333333333
9 :Accuracy: 0.95
10 :Accuracy: 0.9666666666666667
11 :Accuracy: 1.0
12 :Accuracy: 0.9666666666666667
13 :Accuracy: 0.9333333333333333
14 :Accuracy: 0.9666666666666667
15 :Accuracy: 0.9666666666666667
16 :Accuracy: 0.9666666666666667
17 :Accuracy: 0.9666666666666667
18 :Accuracy: 0.9833333333333333
19 :Accuracy: 0.9666666666666667
20 :Accuracy: 0.95
Accuracy_AVE: 0.9666666666666666
```

图 4.5：测试样本测试准确率

上述结果可以看出测试准确率稳定到 96%左右，平均准确率 96.7%。

4.2 sonar 数据集分类结果分析

由于两类数据集处理具有相似性，这里只简要描述

1、求解出判别函数后，把训练集、测试集的投影点分别绘制在一维轴上，两类样本依据颜色区分（训练集：X=0，测试集 1 类：X=1，测试集二类：X=-1），结果战术如下：

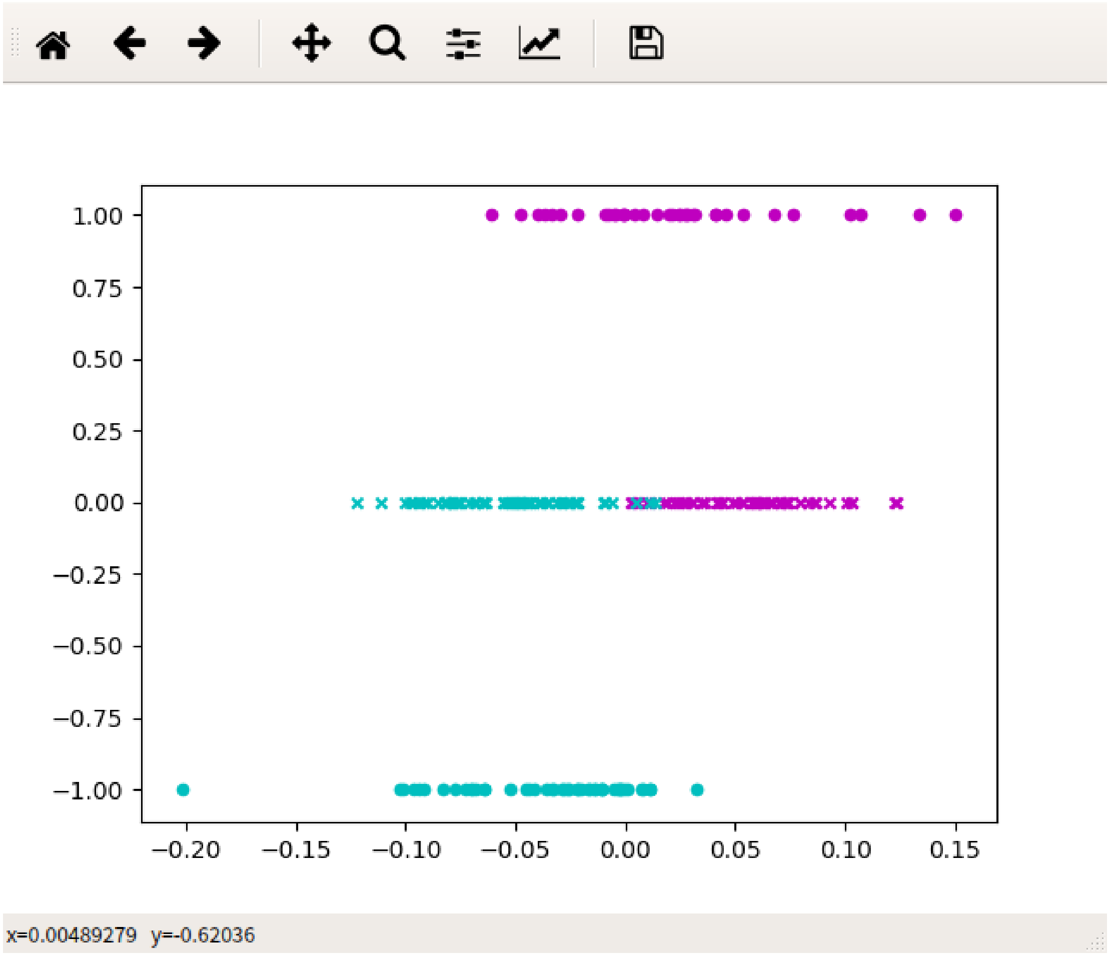
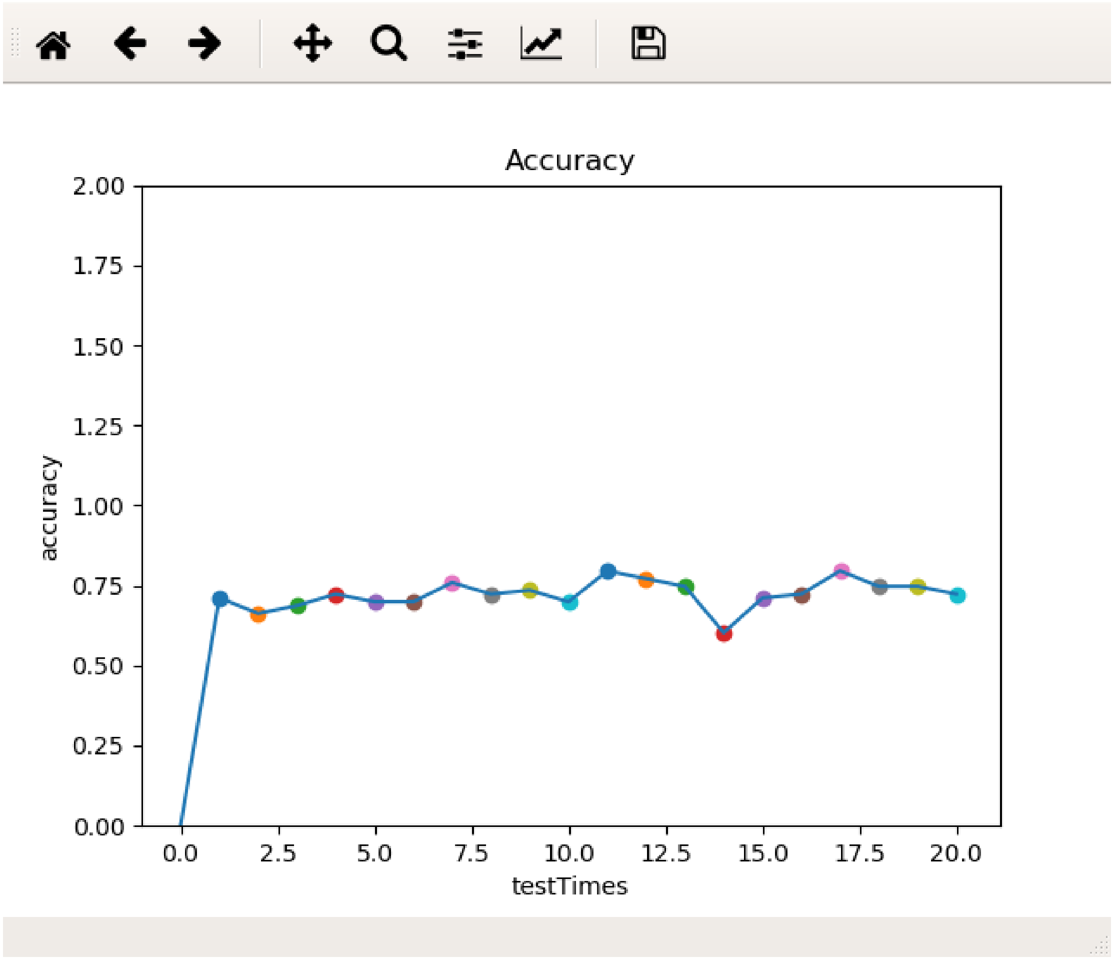


图 4.6：训练、测试样本投影到一维的分布

上述结果可以看出，训练集在分类点 X=0 处有部分重合，测试集的两类均有少量错误判别情况（理论上紫色圆点均在 X>0，青色圆点均在 X<0）。

2、根据上一步计算结果，对测试样本进行分类，对比已知标签，分类正确和错误次数，计算正确率，重新测试二十次，计算平均准确率。结果如下图所示：




```
1 :Accuracy: 0.7108433734939759
2 :Accuracy: 0.6626506024096386
3 :Accuracy: 0.6867469879518072
4 :Accuracy: 0.7228915662650602
5 :Accuracy: 0.6987951807228916
6 :Accuracy: 0.6987951807228916
7 :Accuracy: 0.7590361445783133
8 :Accuracy: 0.7228915662650602
9 :Accuracy: 0.7349397590361446
10 :Accuracy: 0.6987951807228916
11 :Accuracy: 0.7951807228915663
12 :Accuracy: 0.7710843373493976
13 :Accuracy: 0.7469879518072289
14 :Accuracy: 0.6024096385542169
15 :Accuracy: 0.7108433734939759
16 :Accuracy: 0.7228915662650602
17 :Accuracy: 0.7951807228915663
18 :Accuracy: 0.7469879518072289
19 :Accuracy: 0.7469879518072289
20 :Accuracy: 0.7228915662650602
Accuracy_AVE: 0.7228915662650602
```

图 4.6：测试样本测试准确率

上述结果可以看出测试准确率稳定到 70%左右，平均准确率 72.3%。

五、Python 代码

5.1 iris 数据集

```
import os
import sys
import numpy as np
from numpy import *
import random
import matplotlib.pyplot as plt

def readData(DATA_PATH,DATA_SIZE,DATA_DIMENSION,test_rate):
    test_Labels = random.sample(range(0,DATA_SIZE-
1),int(DATA_SIZE*test_rate))
    test_Data = [[0 for i in range(5)] for j in range(len(test_Labels))
]
    train_Data = [[0 for i in range(5)] for j in range(DATA_SIZE-
len(test_Labels))]
    #随机提取训练、测试数据
    f = open(DATA_PATH)
    line = f.readline()
    i = 0
    train_Label = 0
    test_Label = 0
    while line:
        temp = line.split(",",4)
        if i in test_Labels:
            test_Data[test_Label] = temp
            test_Label = test_Label+1
        else:
            train_Data[train_Label] = temp
            train_Label = train_Label+1
        i = i+1
        line = f.readline()
    #将要训练的数据分类保存
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][4] == 'Iris-setosa\n':
            _1_Num = _1_Num+1
        if train_Data[i][4] == 'Iris-versicolor\n':
            _2_Num = _2_Num+1
        if train_Data[i][4] == 'Iris-virginica\n':
            _3_Num = _3_Num+1
    _1_Data = mat(zeros((_1_Num,DATA_DIMENSION)))
```

```
    _2_Data = mat(zeros((_2_Num,DATA_DIMENSION)))
    _3_Data = mat(zeros((_3_Num,DATA_DIMENSION)))
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][4] == 'Iris-setosa\n':
            _1_Data[_1_Num] = train_Data[i][0:4]
            _1_Num = _1_Num+1
        if train_Data[i][4] == 'Iris-versicolor\n':
            _2_Data[_2_Num] = train_Data[i][0:4]
            _2_Num = _2_Num+1
        if train_Data[i][4] == 'Iris-virginica\n':
            _3_Data[_3_Num] = train_Data[i][0:4]
            _3_Num = _3_Num+1
    f.close()
    return train_Data,test_Data,_1_Data.T,_2_Data.T,_3_Data.T
#计算样本均值
def compute_mean(samples):
    mean_mat=mean(samples, axis=1)
    return mean_mat
#计算样本类内离散度
def compute_withinclass_scatter(samples, mean):
    #获取样本维数, 样本个数
    dimens,nums=samples.shape
    #将所有样本向量减去均值向量
    samples_mean=samples-mean
    #初始化类内离散度矩阵
    s_in=0
    for i in range(nums):
        x=samples_mean[:,i]
        s_in+=dot(x,x.T)
    return s_in

def showTrainResule(group1,group2,w,w0):
    dimens,nums=group1.shape
    for i in range(nums):
        position = dot(w.T,group1[:,i])+w0
        plt.scatter(float(position),0,20,'m','x')
    dimens,nums=group2.shape
    for i in range(nums):
        position = dot(w.T,group2[:,i])+w0
        plt.scatter(float(position),0,20,'c','x')
    plt.show()
```

```
def LDA_Fisher(dataPath,dataSize,dataDimension,dataTypeNum,testRate = 2
/5):
    Accuracy = 0.0
    correctNum = wrongNum = 0
    train_Data,test_Data,group1,group2,group3 = readData(dataPath,dataS
ize,dataDimension,testRate)

    #求均值向量
    mean1 = compute_mean(group1)
    mean2 = compute_mean(group2)
    mean3 = compute_mean(group3)

    #求类内离散度
    s_in1 = compute_withinclass_scatter(group1, mean1)
    s_in2 = compute_withinclass_scatter(group2, mean2)
    s_in3 = compute_withinclass_scatter(group3, mean3)

    #求总类内离散度矩阵
    s_w_12 = s_in1+s_in2
    s_w_13 = s_in1+s_in3
    s_w_23 = s_in2+s_in3

    #求解权向量
    w_12 = dot(s_w_12.I,mean1-mean2)
    w_13 = dot(s_w_13.I,mean1-mean3)
    w_23 = dot(s_w_23.I,mean2-mean3)

    #求解阈值
    w0_12 = -0.5*(dot(w_12.T,mean1)+dot(w_12.T,mean2))
    w0_13 = -0.5*(dot(w_13.T,mean1)+dot(w_13.T,mean3))
    w0_23 = -0.5*(dot(w_23.T,mean2)+dot(w_23.T,mean3))

    #显示训练结果
    showTrainResule(group1,group2,w_12,w0_12)
    showTrainResule(group1,group3,w_12,w0_13)
    showTrainResule(group2,group3,w_12,w0_23)

    #测试结果
    for i in range(len(test_Data)):
        test1 = mat(zeros((1,4)))
        test1[0] = test_Data[i][0:4]
        g_12 = dot(w_12.T,test1.T)+w0_12
        g_13 = dot(w_13.T,test1.T)+w0_13
        g_23 = dot(w_23.T,test1.T)+w0_23
```

```
        if test_Data[i][4] == 'Iris-setosa\n':
            color = 'b'
            if g_12>0 and g_13>0:
                correctNum += 1
            else:
                wrongNum += 1
        if test_Data[i][4] == 'Iris-versicolor\n':
            color = 'm'
            if g_12<0 and g_23>0:
                correctNum += 1
            else:
                wrongNum += 1
        if test_Data[i][4] == 'Iris-virginica\n':
            color = 'c'
            if g_13<0 and g_23<0:
                correctNum += 1
            else:
                wrongNum += 1
        # plt.scatter(float(g_12),3,20,color)
        # plt.scatter(float(g_13),2,20,color)
        # plt.scatter(float(g_23),1,20,color)
    Accuracy = correctNum/(correctNum+wrongNum)
    # plt.title("Test Result")
    # plt.show()
    return Accuracy
if __name__ == "__main__":
    Accuracy_AVE = 0
    Accuracy = []
    Accuracy.append(0)
    testTimes = 20
    for i in range(testTimes):
        temp = LDA_Fisher("iris.data",150,4,3,2/5)
        Accuracy.append(temp)
        print(i+1,":Accuracy:",temp)
        plt.scatter(i+1,temp)
        Accuracy_AVE += temp
    Accuracy_AVE /= testTimes
    print("Accuracy_AVE:",Accuracy_AVE)
    plt.title("Accuracy")
    plt.xlabel("testTimes")
    plt.ylabel("accuracy")
    plt.ylim(0,2)
    plt.plot(Accuracy)
    plt.show()
```

5.2 sonar 数据集

```
import os
import sys
import numpy as np
from numpy import *
import random
import matplotlib.pyplot as plt

def readData(DATA_PATH,DATA_SIZE,DATA_DIMENSION,test_rate):
    test_Labels = random.sample(range(0,DATA_SIZE-
1),int(DATA_SIZE*test_rate))
    test_Data = [[0 for i in range(5)] for j in range(len(test_Labels))
]
    train_Data = [[0 for i in range(5)] for j in range(DATA_SIZE-
len(test_Labels))]
    #随机提取训练、测试数据
    f = open(DATA_PATH)
    line = f.readline()
    i = 0
    train_Label = 0
    test_Label = 0
    while line:
        temp = line.split(",",DATA_DIMENSION)
        if i in test_Labels:
            test_Data[test_Label] = temp
            test_Label = test_Label+1
        else:
            train_Data[train_Label] = temp
            train_Label = train_Label+1
        i = i+1
        line = f.readline()
    #将要训练的数据分类保存
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][DATA_DIMENSION] == 'R\n':
            _1_Num = _1_Num+1
        if train_Data[i][DATA_DIMENSION] == 'M\n':
            _2_Num = _2_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-virginica\n':
            _3_Num = _3_Num+1
    _1_Data = mat(zeros((_1_Num,DATA_DIMENSION)))
    _2_Data = mat(zeros((_2_Num,DATA_DIMENSION)))
    _3_Data = mat(zeros((_3_Num,DATA_DIMENSION)))
```

```
    _1_Num = _2_Num = _3_Num = 0
    for i in range(len(train_Data)):
        if train_Data[i][DATA_DIMENSION] == 'R\n':
            _1_Data[_1_Num] = train_Data[i][0:DATA_DIMENSION]
            _1_Num = _1_Num+1
        if train_Data[i][DATA_DIMENSION] == 'M\n':
            _2_Data[_2_Num] = train_Data[i][0:DATA_DIMENSION]
            _2_Num = _2_Num+1
        if train_Data[i][DATA_DIMENSION] == 'Iris-virginica\n':
            _3_Data[_3_Num] = train_Data[i][0:DATA_DIMENSION]
            _3_Num = _3_Num+1
    f.close()
    return train_Data,test_Data,_1_Data.T,_2_Data.T,_3_Data.T

#计算样本均值
def compute_mean(samples):
    mean_mat=mean(samples, axis=1)
    return mean_mat

#计算样本类内离散度
def compute_withinclass_scatter(samples, mean):
    #获取样本维数，样本个数
    dimens,nums=samples.shape
    #将所有样本向量减去均值向量
    samples_mean=samples-mean
    s_in=0
    for i in range(nums):
        x=samples_mean[:,i]
        s_in+=dot(x,x.T)
    return s_in

def showTrainResule(group1,group2,w,w0):
    dimens,nums=group1.shape
    for i in range(nums):
        position = dot(w.T,group1[:,i])+w0
        plt.scatter(float(position),0,20,'m','x')
    dimens,nums=group2.shape
    for i in range(nums):
        position = dot(w.T,group2[:,i])+w0
        plt.scatter(float(position),0,20,'c','x')
    plt.show()

def LDA_Fisher(dataPath,dataSize,dataDimension,dataTypeNum,testRate = 2
/5):
```

```
    Accuracy = 0.0
    correctNum = wrongNum = 0
    train_Data,test_Data,group1,group2,group3 = readData(dataPath,dataS
ize,dataDimension,testRate)

    #求均值向量
    mean1 = compute_mean(group1)
    mean2 = compute_mean(group2)
    mean3 = compute_mean(group3)

    #求类内离散度
    s_in1 = compute_withinclass_scatter(group1, mean1)
    s_in2 = compute_withinclass_scatter(group2, mean2)
    s_in3 = compute_withinclass_scatter(group3, mean3)

    #求总类内离散度矩阵
    s_w = s_in1+s_in2

    #求解权向量
    w = dot(s_w.I,mean1-mean2)

    #求解阈值
    w0 = -0.5*(dot(w.T,mean1)+dot(w.T,mean2))

    #绘图，训练样本投影
    # showTrainResule(group1,group2,w,w0)

    #绘图，测试样本投影
    for i in range(len(test_Data)):
        test1 = mat(zeros((1,dataDimension)))
        test1[0] = test_Data[i][0:dataDimension]
        g = dot(w.T,test1.T)+w0
        if test_Data[i][dataDimension] == 'R\n':
            if g>0:
                correctNum += 1
            else:
                wrongNum += 1
            # plt.scatter(float(g),1,20,'m')
        if test_Data[i][dataDimension] == 'M\n':
            if g<0:
                correctNum += 1
            else:
                wrongNum += 1
            # plt.scatter(float(g),-1,20,'c')
```

```
    Accuracy = correctNum/(correctNum+wrongNum)
    # plt.show()
    return Accuracy



if __name__ == "__main__":
    Accuracy_AVE = 0
    Accuracy = []
    Accuracy.append(0)
    testTimes = 20
    for i in range(testTimes):
        temp = LDA_Fisher("sonar.all-data",208,60,2,2/5)
        Accuracy.append(temp)
        print(i+1,":Accuracy:",temp)
        plt.scatter(i+1,temp)
        Accuracy_AVE += temp
    Accuracy_AVE /= testTimes
    print("Accuracy_AVE:",Accuracy_AVE)
    plt.title("Accuracy")
    plt.xlabel("testTimes")
    plt.ylabel("accuracy")
    plt.ylim(0,2)
    plt.plot(Accuracy)

    plt.show()
```