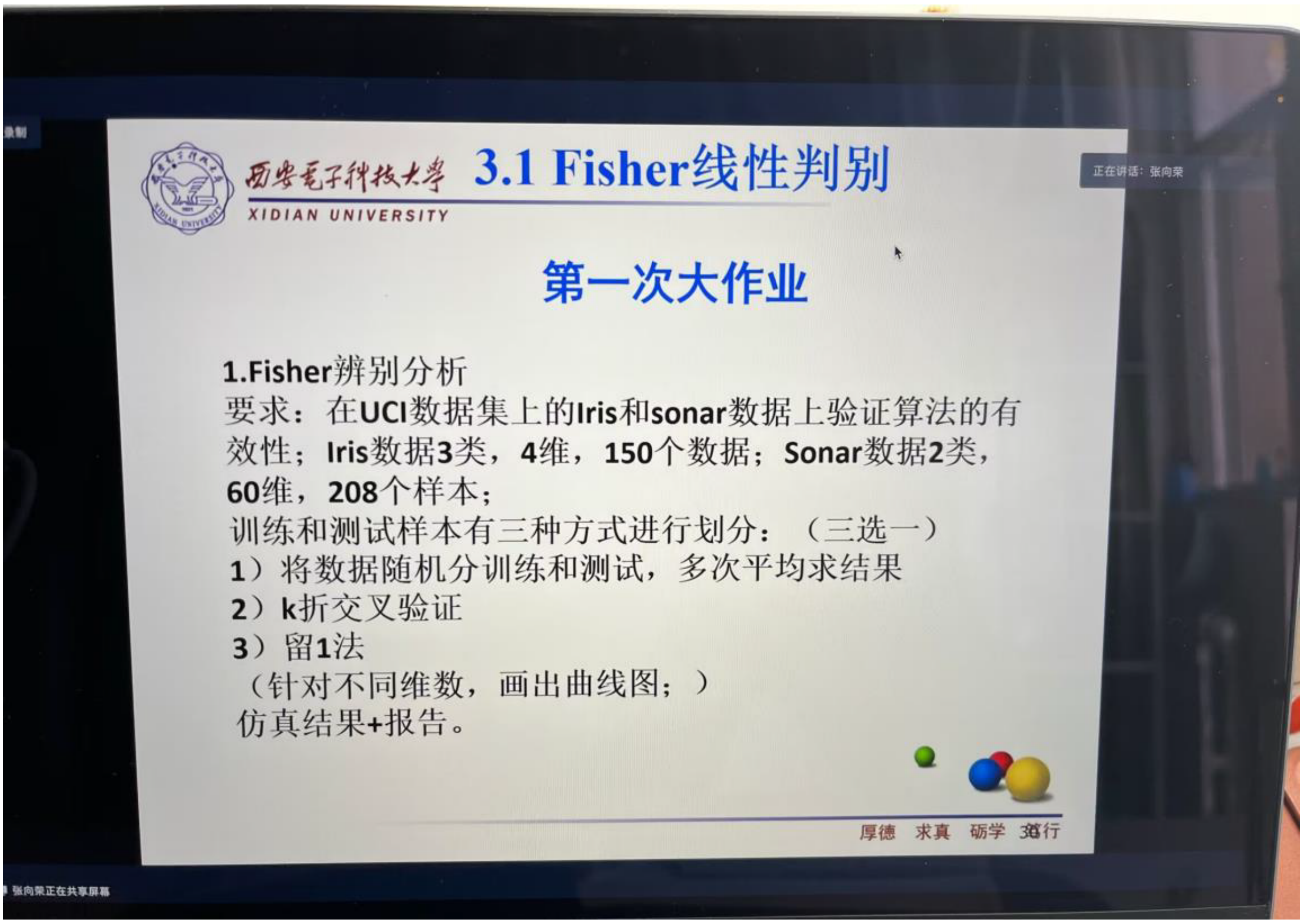
模式识别第一次大作业

姓名：XXXX　学号：XXXXXXXXXX

created： 2023/10/20

一、问题描述：



二、数据集简述：

UCI_Iris数据集：

Iris数据集是一组用于分类问题的常用数据集,来自UCI机器学习库。Iris数据集包含了3种类型鸢尾花的4个特征信息。Iris数据集特征数量少,样本　数量适中,适合作为分类算法的「Hello World」示例。

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
```

基本信息

- 样本数量:150个
- 类别:3类
    - Iris Setosa(山鸢尾)
    - Iris Versicolour(杂色鸢尾)
    - Iris Virginica(维吉尼亚鸢尾)
- 每类样本数量:50个
- 特征数:4个
    - 花萼长度
    - 花萼宽度
    - 花瓣长度
    - 花瓣宽度

数据描述

Iris数据集包含了3种类型鸢尾花的4个特征信息。这是一个多分类问题,常被用来测试分类算法的性能。数据集中的每个样本都属于其中一类,类别是均衡的,每个类型50个样本。4个特征表示花萼和花瓣的长度和宽度,都是正实数。

使用场景

Iris数据集特征数量少,样本数量适中,适合作为分类算法的「Hello World」示例。常用来测试分类算法的效果,如KNN、SVM、决策树等。也可用来比较不同算法的分类性能。由于样本量小,可视化分类结果。

UCI _Sonar数据集介绍

> *UCI 机器学习库（UCI Machine Learning Repository）是一个广泛用于机器学习和数据挖掘研究的资源，其中包含了许多开源数据集，Sonar数据集就是其中之一。Sonar数据集（也称为声纳数据集）是一个经典的二分类问题数据集，用于声纳信号处理和目标检测领域的研究。*
>
> *网址：https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/sonar.all-data*

	attribute_1	attribute_2	...	attribute_58	attribute_59	attribute_60	Class
1	0.02	0.0371	...	0.0084	0.009	0.0032	Rock
2	0.0453	0.0523	...	0.0049	0.0052	0.0044	Rock
3	0.0262	0.0582	...	0.0164	0.0095	0.0078	Rock
4	0.01	0.0171	...	0.0044	0.004	0.0117	Rock
...	...	...	...	...	...	...	...
208	0.026	0.0363	...	0.0036	0.0061	0.0115	Mine

基本信息

- 样本数量:208
- 类别:2类
  - Rock(矿石)
  - Metal(金属)
- 特征数:60
- 特征信息:声纳返回信号在60个频带中的能量强度

数据描述

这是一个二分类问题,根据声纳返回信号区分矿石和金属。数据来自实际的声纳信号采集。相对Iris数据集,它有更多的特征数,样本不均衡,是一个更难的二分类问题。

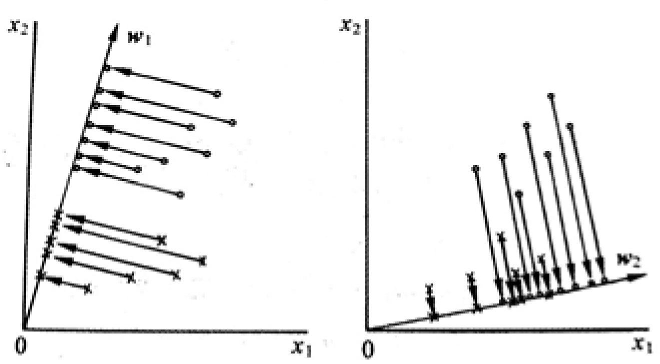
使用场景

Sonar数据集可以用来测试二分类算法在高维特征上的表现。样本量适中,可用于比较不同二分类算法的效果。也可研究特征工程技术在高维数据上的应用。因为样本不均衡,还可研究分类算法处理样本不均衡的技巧。

三、算法分析

Fisher线性判别分析（ LDA）

两类的线性判别问题可以看作是把所有的样本都投影到一个方向上，然后在这个然后在这个一维空间中确定一个分类的阈值。通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。 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 。



这里只讨论两分类的问题，训练样本集是X ={x1,...,xN}, 每一个样本是一个 d 维向量，其中 w1 类的样本是 X1 ={x11,...,xN11} w2 类的样本是 X2 ={x12,...,xN12} 。我们要寻找一个投影方向 W W 也是一个 d 维向量），投影以后的样本变成

$$y_i = W^T X_i,\quad i=1,\ 2,\ \cdots N$$

在原样本空间中，类均值向量为

$$m_i = \frac{1}{N_i} \sum_{x_j \epsilon X_j} x_j \ ,\qquad i = 1,2, \dots , N$$

定义各类的类内离散度矩阵(within-class scatter matrix)为

$$S_i = \sum_{x_j \epsilon X_j} (x_j - m_i)(x_j - m_i)^T\ ,\qquad i = 1,2$$

总类内离散度矩阵(pooled within-class scatter matrix)为

$$S_w = S_1 + S_2$$

类间离散度矩阵(between-class scatter matrix)为

$$S_b = (m_1 - m_2)(m_1 - m_2)^T$$

Fisher 判别准则变为 Rayleigh 商

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

最佳投影方向

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

决策规则

$$\mathrm{g(x)} = w^T x + w_0 \ \lesseqgtr 0,\ 则\ x \epsilon \begin{cases} w_1 \\ w_2 \end{cases}$$

四、源代码如下:

一:

```
#%%
import numpy as np
import pandas as pd
```

```
from sklearn import datasets
import matplotlib.pyplot as plt
#%%
iris = datasets.load_iris()
#%%
def Fisher(X1, X2, n):
    X1 = X1[:, 0:n]
    X2 = X2[:, 0:n]
    m1 = (np.mean(X1, axis=0))
    m2 = (np.mean(X2, axis=0))
    m1 = m1.reshape(n, 1)   # 将行向量转换为列向量以便于计算
    m2 = m2.reshape(n, 1)

    # 计算类内离散度矩阵
    S1 = np.zeros((n, n))   # m1 = within_class_scatter_matrix1
    S2 = np.zeros((n, n))   # m2 = within_class_scatter_matrix2
    for i in range(0, X1.shape[0]):
        S1 += (X1[i].reshape(n, 1) - m1).dot((X1[i].reshape(n, 1) -
m1).T)
    for i in range(0, X2.shape[0]):
        S2 += (X2[i].reshape(n, 1) - m2).dot((X2[i].reshape(n, 1) -
m2).T)
    # 计算总类内离散度矩阵S_w
    S_w = S1 + S2

    # 计算最优投影方向 W
    W = np.linalg.inv(S_w).dot(m1 - m2)
    # 在投影后的一维空间求两类的均值
    m_1 = (W.T).dot(m1)
    m_2 = (W.T).dot(m2)

    # 计算分类阈值 W0(为一个列向量)
    W0 = 0.5 * (m_1 + m_2)

    return W, W0

#%%
def Classify(X, W, W0, n):
    y = (W.T).dot(X[0:n, :]) - W0
    return y
#%%
P1 = iris.data[0:50, 0:4]
P2 = iris.data[50:100, 0:4]
P3 = iris.data[100:150, 0:4]

result = np.zeros(100)
Accuracy = np.zeros((3, 4))
#%%
```

```
for k in range(0,3):
    if k == 0:
        P_1 = P1
        P_2 = P2
    if k == 1:
        P_1 = P2
        P_2 = P3
    if k == 2:
        P_1 = P1
        P_2 = P3
    for n in range(1, 5):
        count = 0
        for i in range(100):
            if i <= 49:
                test = P_1[i]
                test = test.reshape(4, 1)
                train = np.delete(P_1, i, axis=0)
                W, W0 = Fisher(train, P_2, n)
                if (Classify(test, W, W0, n)) >= 0:
                    count += 1
                    result[i] = Classify(test, W, W0, n)
            else:
                test = P_2[i-50]
                test = test.reshape(4, 1)
                train = np.delete(P_2, i-50, axis=0)
                W, W0 = Fisher(P_1, train, n)
                if (Classify(test, W, W0, n)) < 0:
                    count += 1
                    result[i] = Classify(test, W, W0, n)
        Accuracy[k][n-1] = count/100
        if k == 0:
            print("第一类和第二类的分类准确率在维数取%d时为:%.3f" % (n,
Accuracy[k][n-1]))
        if k == 1:
            print("第二类和第三类的分类准确率在维数取%d时为:%.3f" % (n,
Accuracy[k][n-1]))
        if k == 2:
            print("第一类和第三类的分类准确率在维数取%d时为:%.3f" % (n,
Accuracy[k][n-1]))

#%%
#画曲线图
x = np.arange(1,5,1)
plt.title("Fisher in Iris")
plt.xlabel('dimension')
plt.ylabel('Accuracy')
plt.xlim((1, 4))
plt.ylim((0, 1.0))
plt.plot(x, Accuracy[0], 'r-o', label = "class1 and class2")
```

```
plt.plot(x, Accuracy[1], 'g-o', label = "class2 and class3")
plt.plot(x, Accuracy[2], 'b-o', label = "class1 and class3")
plt.legend()
plt.savefig('Fisher in Iris')
plt.show()
#%%
```

二:

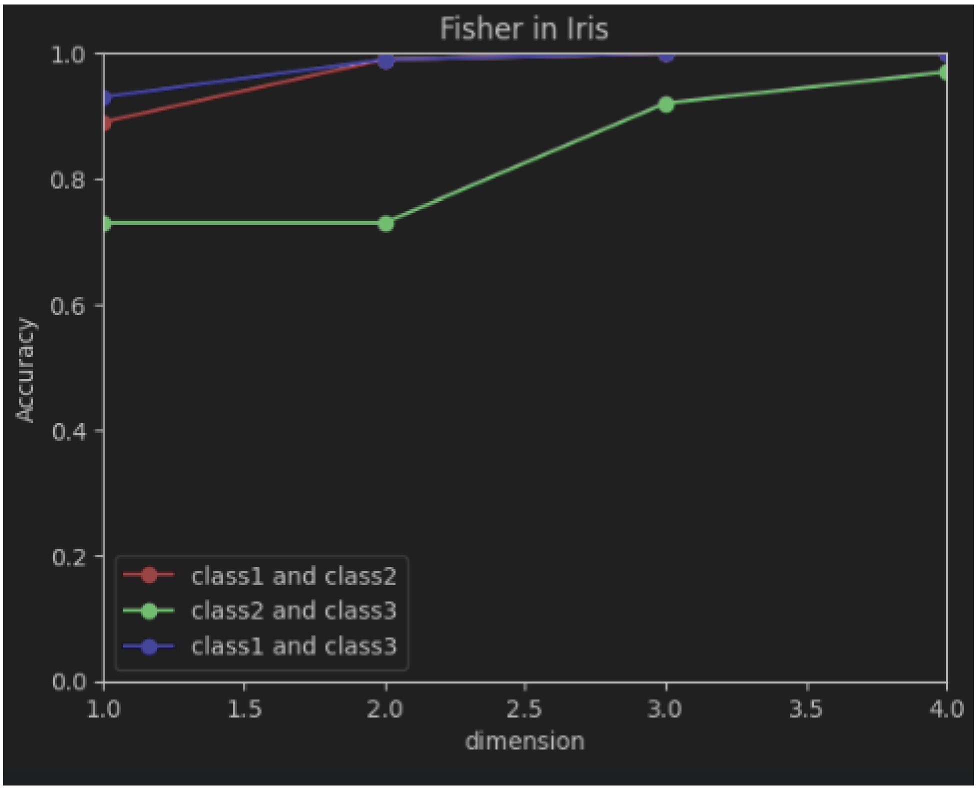
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

sonar = pd.read_csv('sonar.all-data', header=None, sep=',')
sonar1 = sonar.iloc[0:208,0:60]
sonar2 = np.array(sonar1)

def Fisher(X1, X2, n):  #Fisher线性判别过程
    X1 = X1[:, 0:n]
    X2 = X2[:, 0:n]
    m1 = (np.mean(X1, axis=0))
    m2 = (np.mean(X2, axis=0))
    m1 = m1.reshape(n, 1)   # 将行向量转换为列向量以便于计算
    m2 = m2.reshape(n, 1)

    # 计算类内离散度矩阵
    S1 = np.zeros((n, n))
    S2 = np.zeros((n, n))
    for i in range(0, X1.shape[0]):
        S1 += (X1[i].reshape(n, 1) - m1).dot((X1[i].reshape(n, 1) -
m1).T)
    for i in range(0, X2.shape[0]):
        S2 += (X2[i].reshape(n, 1) - m2).dot((X2[i].reshape(n, 1) -
m2).T)
    # 计算总类内离散度矩阵S_w
    S_w = S1 + S2

    # 计算最优投影方向 W
    W = np.linalg.inv(S_w).dot(m1 - m2)
    # 在投影后的一维空间求两类的均值
    m_1 = (W.T).dot(m1)
    m_2 = (W.T).dot(m2)

    # 计算分类阈值 W0(为一个列向量)
    W0 = 0.5 * (m_1 + m_2)

    return W, W0

def Classify(X, W, W0, n):
```

```
    y = (W.T).dot(X[0:n, :]) - W0
    return y

P1 = sonar2[0:104, 0:60]
P2 = sonar2[104:208, 0:60]

result = np.zeros(208)
Accuracy = np.zeros(60)

for n in range(1, 61):
    count = 0
    for i in range(208):
        if i <= 103:
            test = P1[i]
            test = test.reshape(60, 1)
            train = np.delete(P1, i, axis=0)
            W,W0 = Fisher(train, P2, n)
            if (Classify(test, W, W0, n)) >= 0:
                count += 1
                result[i] = Classify(test, W, W0, n)
        else:
            test = P2[i-104]
            test = test.reshape(60, 1)
            train = np.delete(P2, i-104, axis=0)
            W,W0 = Fisher(P1, train, n)
            if (Classify(test, W, W0, n)) < 0:
                count += 1
                result[i] = Classify(test, W, W0, n)
    Accuracy[n-1] = count/208
    print("分类准确率在维数取%d时为:%.3f" % (n, Accuracy[n-1]))

#画曲线图
x = np.arange(1, 61, 1)
plt.title("Fisher in Sonar")
plt.xlabel('dimension')
plt.ylabel('Accuracy')
plt.xlim((1, 60))
plt.ylim((0, 1.0))
plt.plot(x, Accuracy, 'r-,')
plt.show()
```

五、实验结果

Iris数据集

```
deprecated, and will error in future. Ensure you ex
  result[i] = Classify(test, W, W0, n)

第一类和第二类的分类准确率在维数取1时为:0.890
第一类和第二类的分类准确率在维数取2时为:0.990
第一类和第二类的分类准确率在维数取3时为:1.000
第一类和第二类的分类准确率在维数取4时为:1.000
第二类和第三类的分类准确率在维数取1时为:0.730
第二类和第三类的分类准确率在维数取2时为:0.730
第二类和第三类的分类准确率在维数取3时为:0.920
第二类和第三类的分类准确率在维数取4时为:0.970
第一类和第三类的分类准确率在维数取1时为:0.930
第一类和第三类的分类准确率在维数取2时为:0.990
第一类和第三类的分类准确率在维数取3时为:1.000
第一类和第三类的分类准确率在维数取4时为:1.000
```



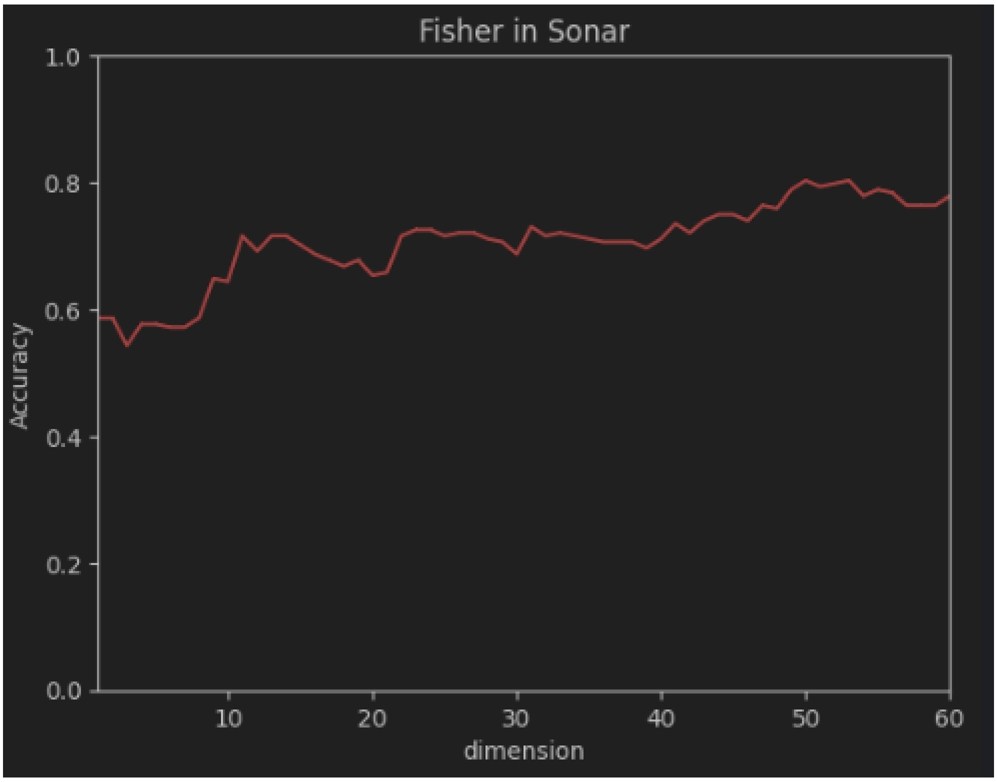
根据给定的三组类别在不同维数下Fisher线性判别的分类准确率,我进行以下分析:

第一类与第二类:随着维数增加,分类准确率迅速提高,在3维时就达到100%,说明这两类样本在3维空间就可完全线性可分。第二类与第三类:随着维数增加,分类准确率逐步提高,但低维时仅达70%左右,至4维时才接近100%,说明这两类样本线性可分性较弱,需要更高维才能达到完全可分。第一类与第三类:分类准确率随维数提高也很快接近100%,2维时就达99%,说明两类在较低维就基本可分。第一类与第二类最易分类,两者在低维就可完全分开;第二类与第三类最难分类,需要更高维才能提高可分性。随着维数增加,三组准确率均明显提高,说明样本在高维空间更可线性可分。

可以看出第二类和第 三类的数据相似度还是蛮高的，在低维的时候的分类效果不是很好。

Sonar数据集

```
performing this operation. (Deprecated NumPy 1.25.)
  result[i] = Classify(test, W, W0, n)
分类准确率在维数取2时为:0.587
分类准确率在维数取3时为:0.543
分类准确率在维数取4时为:0.577
分类准确率在维数取5时为:0.577
分类准确率在维数取6时为:0.572
分类准确率在维数取7时为:0.572
分类准确率在维数取8时为:0.587
分类准确率在维数取9时为:0.649
分类准确率在维数取10时为:0.644
分类准确率在维数取11时为:0.716
分类准确率在维数取12时为:0.692
分类准确率在维数取13时为:0.716
分类准确率在维数取14时为:0.716
分类准确率在维数取15时为:0.702
```



整体趋势:随着维数的增加,分类准确率整体呈上升趋势,但在中间区间(约20-40维)会出现一定程度的波动。低维效果不佳:当维数较低时(1-10维),分类准确率比较低,在50%~70%左右,说明线性可分性不强。高维效果良好:当维数增加到50维以上时,分类准确率达到了约80%左右,说明样本在高维空间更可线性可分。可能存在维数过fitting:分类误差先下降后上升,说明存在

最优维数,过高的维数导致过fitting。最优维数:根据曲线走势,最优维数约在40-60维。此时分类准确率达到最大值,约为80%左右。

随着维数增加分类准确率在逐渐增加，但效果始终不好，可能是数据过于复杂，对于 Fisher 线性判别的方法来说较难处理 。