机器学习第九次作业

- 对于课件4.8-4.10中有关有样本协方差矩阵不等的情况，QDA的例题，采用模拟数据，使用两种等协方差矩阵LDA和Fisher判别的结果如何，请输出训练误差并作图来回答这个问题。

  1. 首先在R中生成QDA例题中的模拟数据

```
# 生成模拟数据
library(MASS)

# 指定均值与方差
mu1 <- c(3,6)
mu2 <- c(3,-2)

covm1 <- matrix(c(0.5,0,0,2),2,2)
covm2 <- matrix(c(2,0,0,2),2,2)

# 生成二元正态分布
mvnorm1 <- mvrnorm(n = 1000, mu1, covm1)
mvnorm2 <- mvrnorm(n = 1000, mu2, covm2)

# 检验是否存在重复值

#any(duplicated(Simudata1))
#any(duplicated(Simudata2))

Simudata1 <- data.frame(mvnorm1)
Simudata2 <- data.frame(mvnorm2)

Simudata1['class'] = 1
Simudata2['class'] = 2

Simudata <- rbind.data.frame(Simudata1,Simudata2)

# 更改标签名
names(Simudata) <- c('x','y','class')

# 输出模拟数据
write.csv(Simudata,file = 'C:/Users/mi/Desktop/Simudata.csv',row.names =
FALSE)
```

  2. 在Python中实现Fisher判别分析与等协方差矩阵LDA

```
# 模块调用
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# 读入数据
Simudata = pd.read_csv('C:/Users/mi/Desktop/Simudata.csv')
```

```
Simudata1 = Simudata[Simudata['class'] == 1]
Simudata2 = Simudata[Simudata['class'] == 2]

#按照80% 20%的方式划分数据集
X1_train, X1_test, y1_train, y1_test =
train_test_split(Simudata1.iloc[:,0:2], Simudata1.iloc[:,2], test_size =
0.2)
X2_train, X2_test, y2_train, y2_test =
train_test_split(Simudata2.iloc[:,0:2], Simudata2.iloc[:,2], test_size =
0.2)

X_train = X1_train.append(X2_train)
y_train = y1_train.append(y2_train)
```

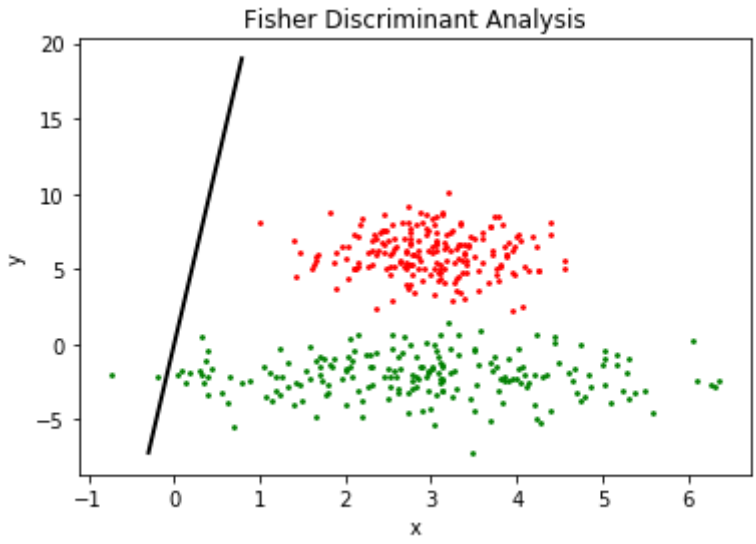
3. 实现Fisher判别分析算法

```
# 函数设计
def cov_avg(X):
    # 计算给定数据的均值和协方差阵

    row, col = X.shape
    x_bar = np.mean(X)
    cov_m = np.dot((X - np.mean(X)).T, (X - np.mean(X)))/row
    return x_bar, cov_m

def fisher(X1, X2):
    # Fisher算法
    avg1, cov_m1 = cov_avg(X1)
    avg2, cov_m2 = cov_avg(X2)
    Sw = cov_m1 + cov_m2
    u, s, v = np.linalg.svd(Sw)
    Sw_inv = np.dot(np.dot(v.T, np.linalg.inv(np.diag(s))), u.T)
    return np.dot(Sw_inv, avg1 - avg2)

def judge(X, w, avg1, avg2):

    pos = np.dot(X, w)
    center1 = np.dot(avg1, w)
    center2 = np.dot(avg2, w)

    dist1 = np.linalg.norm(pos - center1)
    dist2 = np.linalg.norm(pos - center2)

    if dist1 < dist2:
        return 1

    else:
        return 2
```

4. 绘制投影平面并输出预测准确率

```
avg1, cov_m1 = cov_avg(X1_train)
avg2, cov_m2 = cov_avg(X2_train)
avg, cov_m = cov_avg(X_train)
```

```
w = fisher(X1_train, X2_train)

X_test = X1_test.append(X2_test)
y_test = y1_test.append(y2_test)

pred = []
for i in range(X_test.shape[0]):
    label = judge(X_test.iloc[i,:], w, avg1, avg2)
    pred.append(label)



xx = np.arange(-0.3,0.8,0.01)
yy = xx*w[1]/w[0]

# 预测结果为第一类的点
xp1 = X_test.iloc[list(np.where(np.array(pred) == 1)[0]),0]
yp1 = X_test.iloc[list(np.where(np.array(pred) == 1)[0]),1]

# 预测结果为第二类的点
xp2 = X_test.iloc[list(np.where(np.array(pred) == 2)[0]),0]
yp2 = X_test.iloc[list(np.where(np.array(pred) == 2)[0]),1]

fig, ax = plt.subplots()
line = ax.plot(xx, yy, color = 'black',linewidth = 2)
plt.scatter(xp1, yp1, color = 'red', s = 3)
plt.scatter(xp2, yp2, color = 'green', s = 3)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Fisher Discriminant Analysis')
plt.show()

print('Accuracy Score of Fisher Discriminant Analysis: %f' %
(accuracy_score(y_test,pred)))
```



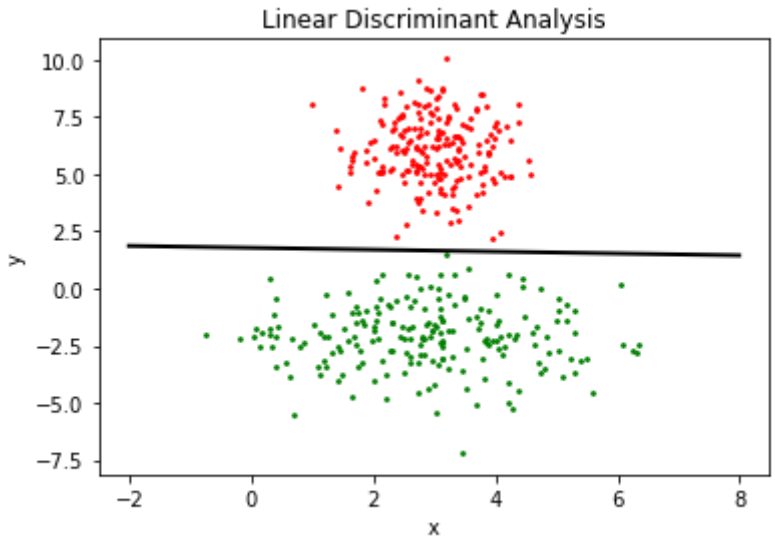

```
Accuracy Score of Fisher Discriminant Analysis: 0.997500
```

5. 训练LDA模型

```
LDA = LinearDiscriminantAnalysis()
LDA.fit(X_train, y_train)
pred = []
pred = LDA.predict(X_test)
para = LDA.get_params()
```

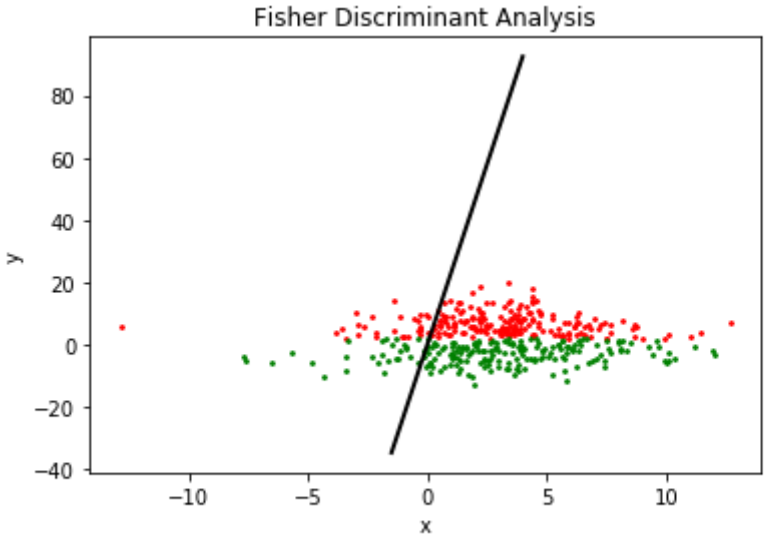
6. 绘制决策边界并输出预测准确率

```
w1 = np.dot(np.linalg.inv(cov_m), (avg1 - avg2).T)
w0 = -0.5*np.dot((avg1 - avg2).T, w1)

xx = np.arange(-2,8,0.01)
yy = -xx*w1[0]/w1[1] - w0

# 预测结果为第一类的点
xp1 = X_test.iloc[list(np.where(np.array(pred) == 1)[0]),0]
yp1 = X_test.iloc[list(np.where(np.array(pred) == 1)[0]),1]

# 预测结果为第二类的点
xp2 = X_test.iloc[list(np.where(np.array(pred) == 2)[0]),0]
yp2 = X_test.iloc[list(np.where(np.array(pred) == 2)[0]),1]

fig, ax = plt.subplots()
line = ax.plot(xx, yy, color = 'black',linewidth = 2)
plt.scatter(xp1, yp1, color = 'red', s = 3)
plt.scatter(xp2, yp2, color = 'green', s = 3)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Discriminant Analysis')
plt.show()

print('Accuracy Score of LDA: %f' %(accuracy_score(y_test,pred)))
```

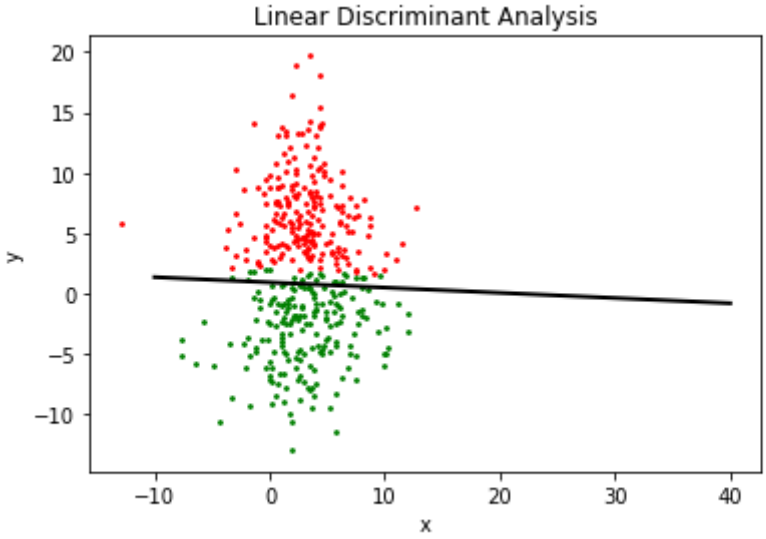



```
Accuracy Score of LDA: 0.997500
```

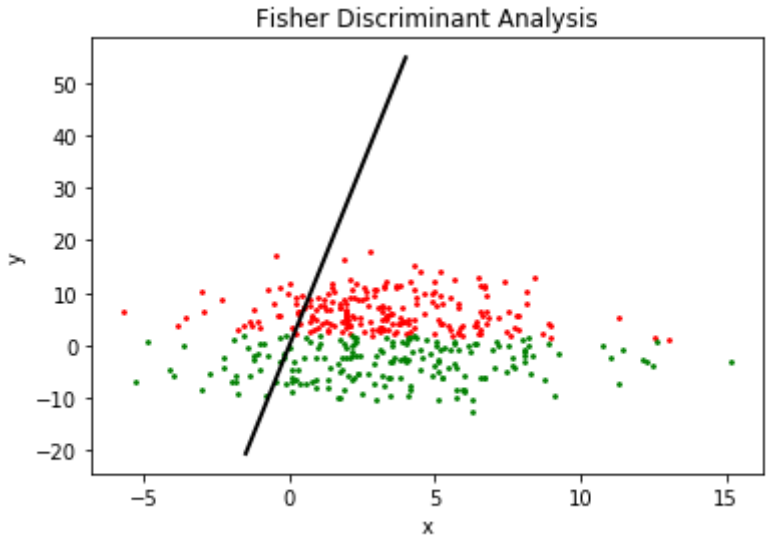
Fisher判别分析与等协方差矩阵LDA（实际上两类的协方差并不相等）的判别效果一致，难分优劣，因此在增大例题中的协方差后又进行了多次实验。



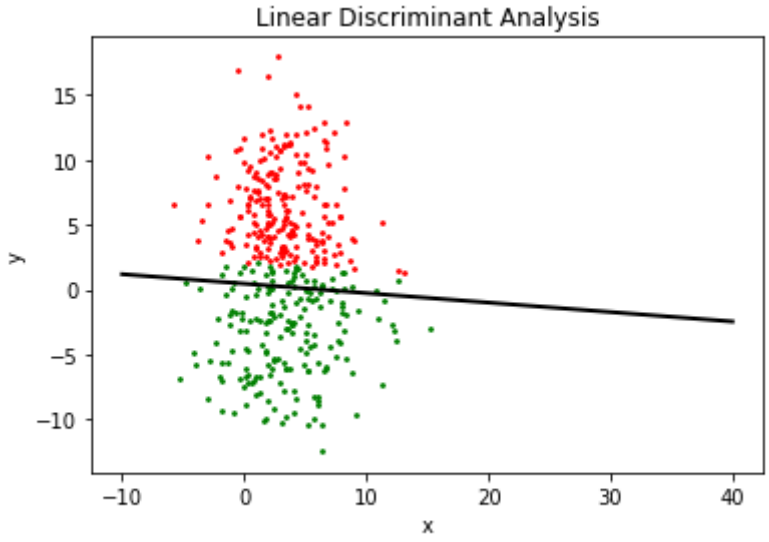

```
Accuracy Score of Fisher Discriminant Analysis: 0.800000
```




```
Accuracy Score of LDA: 0.800000
```




```
Accuracy Score of Fisher Discriminant Analysis: 0.832500
```




```
Accuracy Score of LDA: 0.832500
```

两种方式的判准率始终相同，可能是由于此实验中分层等比例划分数据集导致先验概率相等

比较Fisher判别分析和LDA的划分函数，可以看出在假定等协方差矩阵时，LDA和Fisher判别分析的决策超平面的法向量通过训练集的经验数据进行估计十分接近$(\Sigma_0+\Sigma_1)^{-1}\approx\Sigma^{-1}$

$$w=S_w^{-1}(\mu_0-\mu_1)$$

$$\log\frac{\pi_k}{\pi_l}-\frac{1}{2}(\mu_k+\mu_l)^T\Sigma^{-1}(\mu_k-\mu_l)+x^T\Sigma^{-1}(\mu_k-\mu_l)$$

决策平面相近，导致预测准确率相同

- 对于Boston房价数据，用最后一列房价，按照1/4分位数，中位数和3/4分位数将其分成四段，请使用industry,river,nox,rooms，age,distance几个自变量进行OvO设计和MvM设计，编写程序对LDA进行多分类分析。

1. 模块调用声明

```
import pandas as pd
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

2. 数据预处理

```
Boston = pd.read_csv('C:/Users/mi/Desktop/Boston.csv')
percentile = np.percentile(Boston['medv'], (25,50,75), interpolation =
'midpoint')

# 划分数据集
medv1 = Boston[Boston['medv'] < percentile[0]]
medv2 = Boston[Boston['medv'] >= percentile[0]][Boston['medv'] < percentile[1]]
medv3 = Boston[Boston['medv'] >= percentile[1]][Boston['medv'] < percentile[2]]
medv4 = Boston[Boston['medv'] >= percentile[2]]

# 添加类别标签
medv1['class'] = 1
medv2['class'] = 2
medv3['class'] = 3
medv4['class'] = 4

# 划分训练集与测试集
X1_train, X1_test, y1_train, y1_test =
train_test_split(medv1[['industry','river','nox','rooms','age','distance']],medv
1['class'] , test_size = 0.2)
X2_train, X2_test, y2_train, y2_test =
train_test_split(medv2[['industry','river','nox','rooms','age','distance']],medv
2['class'] , test_size = 0.2)
X3_train, X3_test, y3_train, y3_test =
train_test_split(medv3[['industry','river','nox','rooms','age','distance']],medv
3['class'] , test_size = 0.2)
X4_train, X4_test, y4_train, y4_test =
train_test_split(medv4[['industry','river','nox','rooms','age','distance']],medv
4['class'] , test_size = 0.2)
```

3. OVO分类器

```
LDA1 = LinearDiscriminantAnalysis()
LDA2 = LinearDiscriminantAnalysis()
LDA3 = LinearDiscriminantAnalysis()
LDA4 = LinearDiscriminantAnalysis()
LDA5 = LinearDiscriminantAnalysis()
LDA6 = LinearDiscriminantAnalysis()

# 训练分类器
X12_train = X1_train.append(X2_train)
y12_train = y1_train.append(y2_train)
LDA1.fit(X12_train, y12_train)

X13_train = X1_train.append(X3_train)
y13_train = y1_train.append(y3_train)
LDA2.fit(X13_train, y13_train)

X14_train = X1_train.append(X4_train)
y14_train = y1_train.append(y4_train)
LDA3.fit(X14_train, y14_train)

X23_train = X2_train.append(X3_train)
y23_train = y2_train.append(y3_train)
LDA4.fit(X23_train, y23_train)
```

```
X24_train = X2_train.append(X4_train)
y24_train = y2_train.append(y4_train)
LDA5.fit(X24_train, y24_train)

X34_train = X3_train.append(X4_train)
y34_train = y3_train.append(y4_train)
LDA6.fit(X34_train, y34_train)

X_test = X1_test.append([X2_test,X3_test,X4_test])
y_test = y1_test.append([y2_test,y3_test,y4_test])

# 测试分类效果
pred = []
for i in range(X_test.shape[0]):
    y_pred = [LDA1.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))
[0],LDA2.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))
[0],LDA3.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))
[0],LDA4.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))
[0],LDA5.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))
[0],LDA6.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))[0]]
    y_preds = pd.Series(data = y_pred)
    pred.append(y_preds.mode()[0])

# 输出分类正确率
print('Accuracy Score of LDA OvO: %f' %(accuracy_score(y_test,pred)))
```

Accuracy Score of LDA OvO: 0.650485

4. MvM分类器

```
# MvM
LDAECOC1 = LinearDiscriminantAnalysis()
LDAECOC2 = LinearDiscriminantAnalysis()
LDAECOC3 = LinearDiscriminantAnalysis()
LDAECOC4 = LinearDiscriminantAnalysis()
LDAECOC5 = LinearDiscriminantAnalysis()

f_train = X1_train.append([X2_train,X3_train,X4_train])
# 二元ECOC码
# f1训练数据
ECOC11_train= [-1]*X1_train.shape[0]
ECOC12_train =  [1]*X2_train.shape[0]
ECOC13_train= [-1]*X3_train.shape[0]
ECOC14_train = [-1]*X4_train.shape[0]

f1_ECOC = ECOC11_train + ECOC12_train + ECOC13_train + ECOC14_train
LDAECOC1.fit(f_train, f1_ECOC)

# f2训练数据
ECOC21_train= [1]*X1_train.shape[0]
ECOC22_train = [-1]*X2_train.shape[0]
ECOC23_train= [1]*X3_train.shape[0]
ECOC24_train = [-1]*X4_train.shape[0]

f2_ECOC = ECOC21_train + ECOC22_train + ECOC23_train + ECOC24_train
LDAECOC2.fit(f_train, f2_ECOC)
```

```
# f3训练数据
ECOC31_train= [-1]*X1_train.shape[0]
ECOC32_train =[-1]*X2_train.shape[0]
ECOC33_train= [1]*X3_train.shape[0]
ECOC34_train =[1]*X4_train.shape[0]

f3_ECOC = ECOC31_train + ECOC32_train + ECOC33_train + ECOC34_train
LDAECOC3.fit(f_train, f3_ECOC)

# f4训练数据
ECOC41_train= [1]*X1_train.shape[0]
ECOC42_train =[1]*X2_train.shape[0]
ECOC43_train= [-1]*X3_train.shape[0]
ECOC44_train =[1]*X4_train.shape[0]

f4_ECOC = ECOC41_train + ECOC42_train + ECOC43_train + ECOC44_train
LDAECOC4.fit(f_train, f4_ECOC)

# f5训练数据
ECOC51_train= [1]*X1_train.shape[0]
ECOC52_train =[-1]*X2_train.shape[0]
ECOC53_train= [1]*X3_train.shape[0]
ECOC54_train =[-1]*X4_train.shape[0]

f5_ECOC = ECOC51_train + ECOC52_train + ECOC53_train + ECOC54_train
LDAECOC5.fit(f_train, f5_ECOC)

# 类别向量
vector1 = np.array([-1, 1, -1, 1, 1])
vector2 = np.array([1, -1, -1, 1, -1])
vector3 = np.array([-1, 1, 1, -1, 1])
vector4 = np.array([-1, -1, 1, -1, 1])

# 测试分类效果
pred = []
for i in range(X_test.shape[0]):
    y_pred = np.array([LDAECOC1.predict(np.array(X_test.iloc[i,:]).reshape(1,
-1))[0],LDAECOC2.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))
[0],LDAECOC3.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))
[0],LDAECOC4.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))
[0],LDAECOC5.predict(np.array(X_test.iloc[i,:]).reshape(1, -1))[0]])
    hm_dist1 = np.linalg.norm(vector1 - y_pred, ord = 1)
    eu_dist1 = np.linalg.norm(vector1 - y_pred)
    hm_dist2 = np.linalg.norm(vector2 - y_pred, ord = 1)
    eu_dist2 = np.linalg.norm(vector2 - y_pred)
    hm_dist3 = np.linalg.norm(vector3 - y_pred, ord = 1)
    eu_dist3 = np.linalg.norm(vector3 - y_pred)
    hm_dist4 = np.linalg.norm(vector4 - y_pred, ord = 1)
    eu_dist4 = np.linalg.norm(vector4 - y_pred)

    hm_dist = [hm_dist1,hm_dist2,hm_dist3,hm_dist4]
    eu_dist = [eu_dist1,eu_dist2,eu_dist3,eu_dist4]

    #根据海明距离确定分类
    pred.append(hm_dist.index(min(hm_dist)) + 1)

    # 根据欧氏距离确定分类
```

```
    pred.append(eu_dist.index(min(eu_dist)) + 1)

# 输出分类正确率
print('Accuracy Score of LDA MvM: %f' %(accuracy_score(y_test,pred)))
```

Accuracy Score of LDA MvM: 0.320388