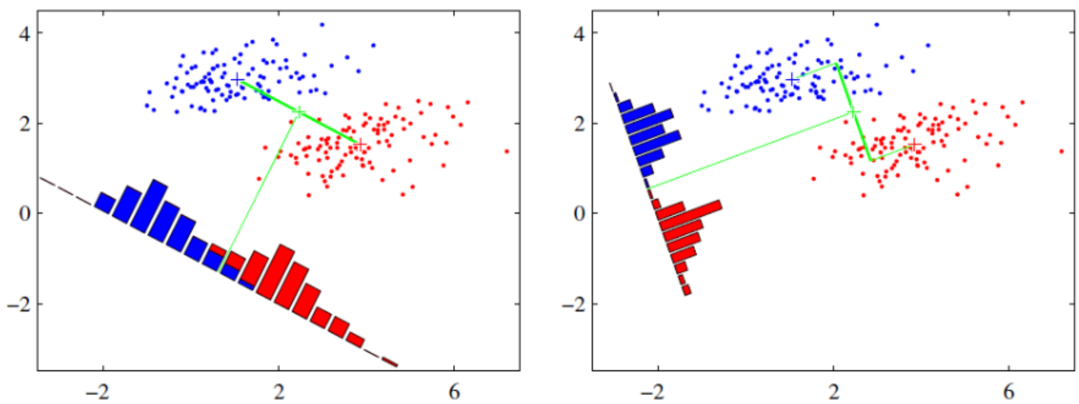
# 机器学习实验 8. 线性判别分析 LDA

## 线性判别分析算法原理概述

LDA 是一种有监督学习的降维技术，也就是说它的数据集的每个样本都是有类别输出的。这点和 PCA 不同。PCA 是不考虑样本类别输出的无监督降维技术。LDA 的思想可以用一句话概括，就是“投影后类内方差最小，类间方差最大”。这句话的理解就是，要将数据在低纬度上进行投影，投影后希望每一种类别数据的投影点尽可能的接近，而不同类别的数据的类别中心之间的距离尽可能的大。

先看最简单的情况。假设有两类数据，分别为红色和蓝色，如下图所示，这些数据特征是二维的，要求这些数据投影到一维的一条直线，让每一种类别的数据的投影点尽可能的接近，而红色和蓝色数据中心之间的距离尽可能的大。




上图中提供了两种投影方式，哪一种能更好的满足标准呢？从直观上可以看出，右图要比左图的投影效果好，因为右图的黑色数据和蓝色数据比较更为集中，且类别之间的距离明显。左图则在边界处数据混杂。以上是二类

数据的线性判别分析（LDA）算法的思想。在实际应用中，数据可能是多个类别的，特征也是高维的，投影后的也一般不是直线，而是原始高维空间中的超平面（低维子空间）。

## 线性判别分析的瑞利商模型

- 瑞利商是指这样的函数 $R(A,x)$:

$$R(A,x)=\frac{x^H Ax}{x^H x}$$

其中 $x$ 是非零向量，而 $A$ 为 $n\times n$ 的 Hermitian 矩阵就是满足共轭转置矩阵和自己相等的矩阵，即 $A^H=A$。如果我们的矩阵 $A$ 是实矩阵，则满足 $A^H=A$ 的矩阵即为 Hermitian 矩阵。

瑞利商 $R(A,x)$有一个非常重要的性质，即它的最大值等于矩阵 $A$ 最大的特征值，而最小值等于矩阵 $A$ 的最小特征值，也就满足

$$\lambda_{\min}\le\frac{x^H Ax}{x^H x}\le\lambda_{\max}$$

具体的证明这里就不给出了。当向量 $x$ 是标准正交基时，即满足 $x^H x=1$ 时，瑞利商退化为：$R(A,x)=x^H Ax$，这个形式在谱聚类和 PCA 中都有出现。

- 广义瑞利商是指这样的函数 $R(A,B,x)$:

$$R(A,B,x)=\frac{x^H Ax}{x^H Bx}$$

其中 $x$ 为非零向量，而 $A$、$B$ 为 n×n 的 Hermitian 矩阵。$B$ 为正定矩阵。它的最大值和最小值是什么呢？其实只要通过将其标准化就可以转化为瑞利商的格式。令 $x=B^{-1/2}x'$，则分母转化为：

$$x^H Bx=x'^H(B^{-1/2})^H BB^{-1/2}x'=x'^H B^{-1/2}BB^{-1/2}x'=x'^H x'$$

而分子转化为:

$$x^H Ax = x^{'H} B^{-1/2} A B^{-1/2} x^{'}$$

此时我们的 R(A,B,x)转化为$R(A,B,x^{'})$：

$$R(A,B,x^{'}) = \frac{x^{'H} B^{-1/2} A B^{-1/2} x^{'}}{x^{'H} x^{'}}$$

利用前面的瑞利商的性质可以很快知道，$R(A,B,x)$的最大值为矩阵$B^{-1/2}AB^{-1/2}$的最大特征值，或者说矩阵$B^{-1}A$的最大特征值，而最小值为矩阵$B^{-1}A$的最小特征值。

## 线性判别分析的求解

假设数据集 $D=\{(x_1,y_1), (x_2,y_2),\ldots, (x_m,y_m)\}$，其中任意样本 $x_i$ 为 $n$ 维向量，$y_i \in \{C_1, C_2, ..., C_k\}$。我们定义 $N_j (j=1,2,...,k)$ 为第 $j$ 类样本个数，$X_j (j=1,2,...,k)$为第 j 类样本的集合，而$\mu_j (j=1,2,...,k)$为第 j 类样本的均值向量，定义$\sum_j (j=1,2,...,k)$为第 j 类样本的协方差矩阵。

二类 LDA 向低维投影，投影到低维空间的结果是一条直线，而多类 LDA 则是一个超平面了。假设我们投影到低维空间的维度为 d，对应的基向量就是（w1,w2,…,wd)，基向量组成的矩阵为 W，它是一个 n×d 的矩阵。

由此优化目标可以变为:

$$\frac{W^T S_b W}{W^T S_W W}$$

其中，$S_b = \sum_{j-1}^{k} N_j (\mu_j - \mu)(\mu_j - \mu)^T$ 为所有样本的均值向量。$S_w = \sum_{i=1}^{k} S_{wj} = \sum_{j=1}^{k} \sum_{x \in X_j} (x-\mu_j)(x-\mu_j)^T$。

注意，这里$W^T S_b W$和$W^T S_W W$都是矩阵，不是标量，无法作为一个标量函数来优化。也就是说，无法直接用二类 LDA 的优化方法，因此一般采用其它的一些代替优化目标来实现。

常见的一个多类 LDA 优化目标函数定义为：

$$\arg\max J(W) = \frac{\prod_{diag} W^T S_b W}{\prod_{diag} W^T S_w W}$$

其中$\prod_{diag} A$为$A$的主对角线元素的乘积，W 为 n×d 的矩阵。

J(W)的优化过程可以转化为：

$$J(W) = \frac{\prod_{i=1}^{d} w_i^T S_b wi}{\prod_{i=1}^{d} w_i^T S_w wi} = \prod_{i=1}^{d} \frac{w_i^T S_b wi}{w_i^T S_w wi}$$

上式的最右边就是广义瑞利商。最大值是矩阵的最大特征值，最大的 $d$ 个值的乘积就是矩阵$S_w^{-1} S_b$的最大的 $d$ 个特征乘积，此时对应的矩阵 $W$ 为这最大的 $d$ 个特征值对应的特征向量张成的矩阵。

由于 $W$ 是一个利用了样本类别得到的投影矩阵，因此它的降维到的最低维度 $d$ 最大值为 $k$-1。

上述目标函数也可以使用矩阵的迹（Trace）或行列式代替。尤其是使用矩阵的迹，可以获得更为明显的瑞利商的解。

## 线性判别分析算法流程

输入：数据集集 $D=\{(x_1,y_1), (x_2,y_2),\ldots, (x_m,y_m)\}$，其中任意样本 $x_i$ 为 $n$ 维向量，$y_i \in \{C_1, C_2, ..., C_k\}$，降维到维度 d。

输出：降维后的样本集 $D'$

1）计算类内散度矩阵 Sw

2）计算类间散度矩阵 Sb

3）计算矩阵 $S_w^{-1}S_b$

4）计算 $S_w^{-1}S_b$ 的最大的 d 个特征值和对应的 d 个特征向量(w1,w2,…,wd)，得到投影矩阵 W

5）对样本集中的每一个样本特征 $x_i$ 转化为新的样本 $z_i = W^T x_i$

6）得到输出样本集 $D' = \{(z_1, y_1), (z_2, y_2), \ldots, (zm, y_m)\}$

## 线性判别分析算法的一般实现

1）数据生成：这里直接通过 scikit-learn 的接口来生成数据

```
from sklearn.datasets import make_multilabel_classification
import numpy as np

x, y = make_multilabel_classification(n_samples=20, n_features=2,
                                      n_labels=1, n_classes=1,
                                      random_state=2)  # 设置随机数种子，保证每次产生相同的数据。

# 根据类别分个类
index1 = np.array([index for (index, value) in enumerate(y) if value == 0])  # 获取类别1的indexs
index2 = np.array([index for (index, value) in enumerate(y) if value == 1])  # 获取类别2的indexs

c_1 = x[index1]   # 类别1的所有数据(x1, x2) in X_1
c_2 = x[index2]  # 类别2的所有数据(x1, x2) in X_2
#print(c_1)
#print(c_2)
```

2）LDA 算法实现

```
def cal_cov_and_avg(samples):
    """
    给定一个类别的数据，计算协方差矩阵和平均向量
    :param samples:
    :return:
    """
    u1 = np.mean(samples, axis=0)
    cov_m = np.zeros((samples.shape[1], samples.shape[1]))
    for s in samples:
        t = s - u1
        cov_m += t * t.reshape(2, 1)
    return cov_m, u1


def fisher(c_1, c_2):
    """
    fisher算法实现
    :param c_1:
    :param c_2:
    :return:
    """
    cov_1, u1 = cal_cov_and_avg(c_1)
    cov_2, u2 = cal_cov_and_avg(c_2)
    s_w = cov_1 + cov_2
    u, s, v = np.linalg.svd(s_w)  # 奇异值分解
    s_w_inv = np.dot(np.dot(v.T, np.linalg.inv(np.diag(s))), u.T)
    return np.dot(s_w_inv, u1 - u2)
```

3）判定分析

```
def judge(sample, w, c_1, c_2):
    """
    true 属于1
    false 属于2
    :param sample:
    :param w:
    :param center_1:
    :param center_2:
    :return:
    """
    u1 = np.mean(c_1, axis=0)
    u2 = np.mean(c_2, axis=0)
    center_1 = np.dot(w.T, u1)
    center_2 = np.dot(w.T, u2)
    pos = np.dot(w.T, sample)
    return abs(pos - center_1) < abs(pos - center_2)


w = fisher(c_1, c_2)  # 调用函数，得到参数w
out = judge(c_1[1], w, c_1, c_2)   # 判断所属的类别
print(out)
```

4）可视化显示

```
import matplotlib.pyplot as plt

plt.scatter(c_1[:, 0], c_1[:, 1], c='#99CC99')
plt.scatter(c_2[:, 0], c_2[:, 1], c='#FFCC00')
line_x = np.arange(min(np.min(c_1[:, 0]), np.min(c_2[:, 0])),
                   max(np.max(c_1[:, 0]), np.max(c_2[:, 0])),
                   step=1)

line_y = - (w[0] * line_x) / w[1]
plt.plot(line_x, line_y)
plt.show()
```