Fisher线性判别分析

降维作为一种减少特征冗余的方法，也可以应用在线性分类当中。在K分类问题中，Fisher线性判别分析通过最大化类间方差和最小化类内方差，将数据映射到K-1维空间进行分类。本文将着重讨论推导多分类的情况。

1. 符号标识

符号	意义
N_k	属于第K类的样本数量
N	样本总数
K	类别总数
$\mathbf{x} \in \mathbb{R}^D$	D维样本向量
$\mathbf{X} \in \mathbb{R}^{N \times D}$	样本矩阵
$\mathbf{S}_W \in \mathbb{R}^{D \times D}$	类内散度矩阵
$\mathbf{S}_B \in \mathbb{R}^{D \times D}$	类间散度矩阵
$\mathbf{W} \in \mathbb{R}^{D \times K-1}$	投影矩阵
$\mathbf{y} \in \mathbb{R}^{K-1}$	投影后样本向量
$\mathbf{u} \in \mathbb{R}^{K-1}$	投影后样本均值
$\mathbf{P}_W \in \mathbb{R}^{K-1 \times K-1}$	投影后类内散度矩阵
$\mathbf{P}_B \in \mathbb{R}^{K-1 \times K-1}$	投影后类间散度矩阵
$Tr(\cdot)$	矩阵的迹

2. 散度矩阵(Scatter Matrices)

定义类内散度矩阵

$$\mathbf{S}_W = \sum_{k=1}^{K} \mathbf{\Sigma}_k \cdots\cdots (2.1)$$

$$\mathbf{\Sigma}_k = \sum_{n \in C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T \cdots\cdots (2.2)$$

其中

$$\mathbf{m}_k = \frac{1}{N_k}\sum_{n\in C_k}\mathbf{x}_n \cdots\cdots(2.3)$$

定义类间散度矩阵

$$\mathbf{S}_B = \sum_{k=1}^{K} N_k(\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T \cdots\cdots(2.4)$$

其中

$$\mathbf{m}_k = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n = \frac{1}{N}\sum_{k=1}^{K} N_k\mathbf{m}_k \cdots\cdots(2.5)$$

可得混合散度矩阵(the mixture scatter matrix)

$$\mathbf{S}_M = \mathbf{S}_W + \mathbf{S}_B \cdots\cdots(2.6)$$

3. 二分类求解

$$设均值向量\mathbf{m}_1 = \frac{1}{N_1}\sum_{n\in C_1}\mathbf{x}_n, \mathbf{m}_2 = \frac{1}{N_2}\sum_{n\in C_2}\mathbf{x}_n,投影方向为\mathbf{w}, y_n = \mathbf{w}^T\mathbf{x}_n.$$

投影后均值为$u_1 = \mathbf{w}^T\mathbf{m}_1, u_2 = \mathbf{w}^T\mathbf{m}_2$,类间距离$(u_1 - u_2)^2$.

$$类内总方差s_1^2 = \sum_{n\in C_1}(y_n - u_1)^2, s_2^2 = \sum_{n\in C_2}(y_n - u_2)^2.最大化类间距离，最小化类内方差。$$

$$J = \frac{(u_1 - u_2)^2}{s_1 + s_2} = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}}$$

其中,

$$\mathbf{S}_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T, \mathbf{W}_W = \sum_{n\in C_1}(\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n\in C_2}(\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T$$

J对$\mathbf{w}$求导可得

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{2(\mathbf{S}_B\mathbf{w}\mathbf{w}^T\mathbf{S}_W\mathbf{w} - \mathbf{w}^T\mathbf{S}_B\mathbf{w}\mathbf{S}_W\mathbf{w})}{(\mathbf{w}^T\mathbf{S}_W\mathbf{w})^2} = 0$$

$$\mathbf{S}_W\mathbf{w} \propto \mathbf{S}_B\mathbf{w} = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T\mathbf{w} \propto (\mathbf{m}_1 - \mathbf{m}_2)$$

$$\rightarrow \mathbf{w} \propto \mathbf{S}_w^{-1}(\mathbf{m}_1 - \mathbf{m}_2),我们只关注投影的方向,也可以给其范数加上约束。$$

4. 多分类求解

由于我们有K个类别，根据贝叶斯分类器对此类问题的处理，是得到K个后验概率$p_1(\mathbf{x})...p_K(\mathbf{x})$, 然而我们知道$\sum_i p_i = 1$，因此，只有$K-1$个是线性无关的。**那么我们讲$D$维样本空间映射到$K-1$维空间是没有分类信息的损失**

的。

于是，有线性映射

$$\mathbf{y} = \mathbf{W}^T\mathbf{x} \cdots\cdots (4.1)$$

$$\mathbf{P}_W = \mathbf{W}^T\mathbf{P}_W\mathbf{W} \cdots\cdots (4.2)$$

$$\mathbf{P}_B = \mathbf{W}^T\mathbf{P}_W\mathbf{W} \cdots\cdots (4.3)$$

在二分类时的思想是最大化类间方差，最小化类内方差，于是可得二分类时的损失函数

$$J(\mathbf{W}) = \frac{\mathbf{w}^T\mathbf{S}_B\mathbf{w}}{\mathbf{w}^T\mathbf{S}_W\mathbf{w}}$$

与之不同的是，多分类情况下分子分母都是矩阵而不是标量，且矩阵没有除法，因此需要采用另一种判别准则。

判别准则有多种，我们这里使用其中一种。可以先从直觉上理解，具体是为什么等我明白了再补充吧。

$$J(\mathbf{W}) = Tr(\mathbf{P}_W^{-1}\mathbf{P}_B) = Tr((\mathbf{W}^T\mathbf{S}_W\mathbf{W})^{-1}\mathbf{W}^T\mathbf{S}_B\mathbf{W}) \cdots\cdots (4.4)$$

对其求微分可得

$$\begin{aligned}
d(J) &= Tr[d(\mathbf{W}^T\mathbf{S}_W\mathbf{W})^{-1})\mathbf{W}^T\mathbf{S}_B\mathbf{W})] + Tr[2(\mathbf{W}^T\mathbf{S}_W\mathbf{W})^{-1}\mathbf{W}^T\mathbf{S}_B d(\mathbf{W})] \\
&= Tr[-(\mathbf{W}^T\mathbf{S}_W\mathbf{W})^{-1})d(\mathbf{W}^T\mathbf{S}_W\mathbf{W})(\mathbf{W}^T\mathbf{S}_W\mathbf{W})^{-1}\mathbf{W}^T\mathbf{S}_B\mathbf{W}] + Tr[2(\mathbf{W}^T\mathbf{S}_W\mathbf{W})^{-1}\mathbf{W}^T\mathbf{S}_B d(\mathbf{W})] \\
&= Tr[-2\mathbf{P}_W^{-1}\mathbf{W}^T\mathbf{S}_W d(W)\mathbf{P}_W^{-1}\mathbf{P}_B] + Tr[2\mathbf{P}_W^{-1}\mathbf{W}^T\mathbf{S}_B d(\mathbf{W})] \\
&= Tr[(-2\mathbf{P}_W^{-1}\mathbf{P}_B\mathbf{P}_W^{-1}\mathbf{W}^T\mathbf{S}_W + 2\mathbf{P}_W^{-1}\mathbf{W}^T\mathbf{S}_B)d(\mathbf{W})]
\end{aligned} \quad (4.5)$$

可得

$$\frac{\partial J}{\partial \mathbf{W}} = -2\mathbf{S}_W\mathbf{W}\mathbf{P}_W^{-1}\mathbf{P}_B\mathbf{P}_W^{-1} + 2\mathbf{S}_B\mathbf{W}\mathbf{P}_W^{-1} = 0 \cdots\cdots (4.6)$$

$$\mathbf{S}_W^{-1}\mathbf{S}_B\mathbf{W} = \mathbf{W}\mathbf{P}_W^{-1}\mathbf{P}_B \cdots\cdots (4.7)$$

式4.7的形式容易与矩阵的特征值联系起来。式中的散度矩阵$\mathbf{S}_B$是不满秩的，它是由K个秩为1的矩阵相加得到的，而在式2.3的约束下，只有$K-1$个矩阵是线性无关的，因此它的秩最多为$K-1$。而$\mathbf{S}_W$是满秩的，则$\mathbf{S}_W^{-1}\mathbf{S}_B$只有$K-1$个非零特征值。

命题1：存在一个线性变换$Q \in \mathbb{R}^{K-1\times K-1}$且$Q^{-1}$存在，使得

$$\mathbf{Q}^T\mathbf{P}_W\mathbf{Q} = \mathbf{I}, \mathbf{Q}^T\mathbf{P}_B\mathbf{Q} = \mathbf{\Lambda} \cdots\cdots (4.8)$$

其中，$\mathbf{I}$为$K-1$阶单位矩阵，$\mathbf{\Lambda}$为$K-1$阶对角矩阵。

证明：

$\because \mathbf{P}_W$正定，存在C使得$\mathbf{C}^T\mathbf{P}_W\mathbf{C} = I$,而$\mathbf{C}^T\mathbf{P}_B\mathbf{C}$是实对称矩阵，从而存在正交变换

$$\mathbf{D}^T(\mathbf{C}^T\mathbf{P}_B\mathbf{C})\mathbf{D} = diag(\lambda_1, \ldots, \lambda_{K-1})$$

令$\mathbf{Q} = \mathbf{C}\mathbf{D}$,则$\mathbf{Q}^T\mathbf{P}_W\mathbf{Q} = \mathbf{D}^T(\mathbf{C}^T\mathbf{P}_W\mathbf{C})\mathbf{D} = \mathbf{D}^T\mathbf{D} = \mathbf{I}$,得证。

将式4.8带入式4.7可得

$$\mathbf{S}_W^{-1}\mathbf{S}_B(\mathbf{WQ}) = (\mathbf{WQ})\mathbf{\Lambda}\cdots\cdots(4.9)$$

可以发现，**$\mathbf{\Lambda}$不仅是$\mathbf{P}_B$的特征值矩阵，还是$\mathbf{S}_W^{-1}\mathbf{S}_B$的特征值矩阵**。则有，

$$J(\mathbf{W}) = Tr(\mathbf{P}_W^{-1}\mathbf{P}_B) = \sum_i^{K-1}\lambda_i$$

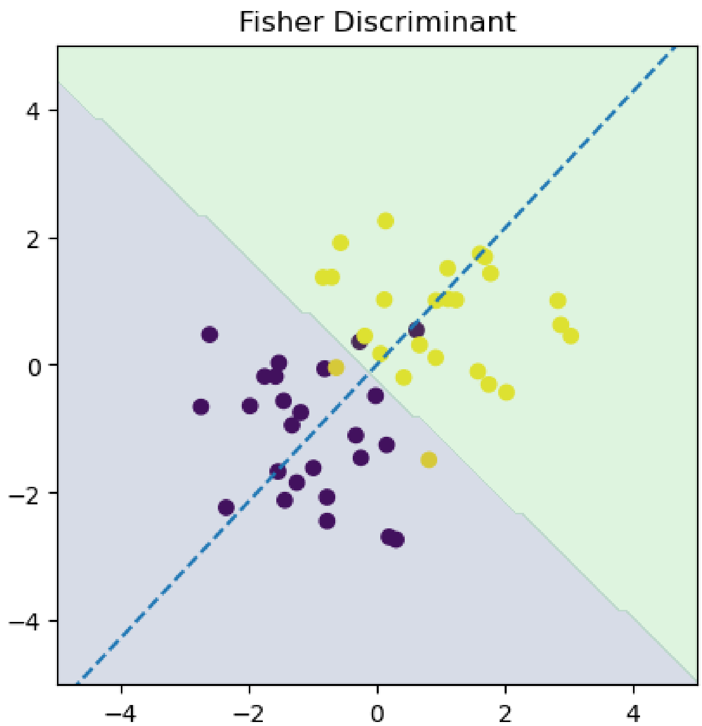
$$Tr(\mathbf{S}_W^{-1}\mathbf{S}_B) = \sum_i^{D}\mu_i\cdots\cdots(4.10)$$

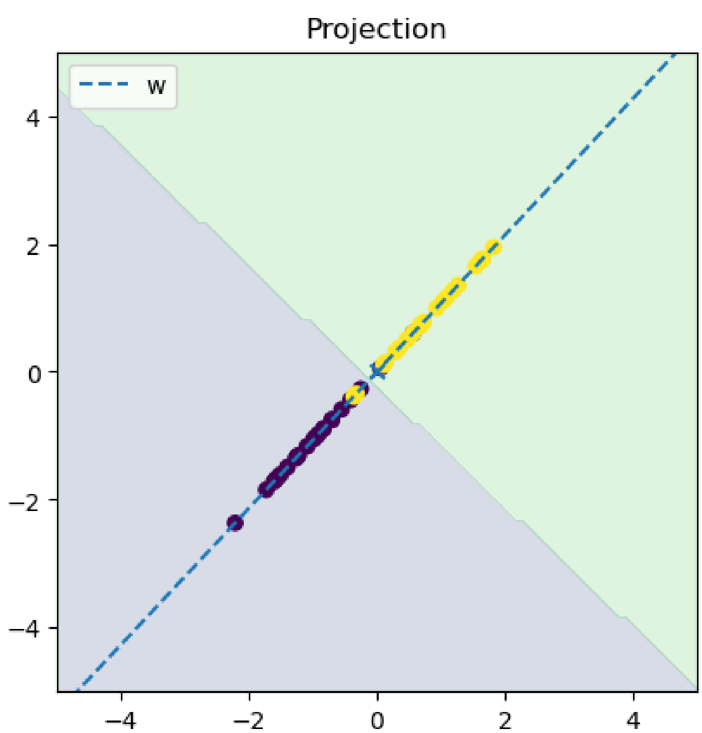
注意，这里$\mathbf{S}_W^{-1}\mathbf{S}_B$是我们可以通过观测到的样本计算出来的，所以特征值是确定的$\mu_i, i = 1, ..., D$,式4.10给出了与目标函数之间的关系，并且由正交变换的不变性，我们可得知$\mathbf{W}$就是由$\mathbf{S}_W^{-1}\mathbf{S}_B$最大的$K-1$个特征值对应的特征向量构成的。

代码实现

```python
# 二分类
class FisherLinearDiscriminant:
    """
    Only for 2 classes
    """
    def __init__(self, w=None, threshold=None):
        self.w = w
        self.threshold = threshold

    def fit(self, x_train: np.ndarray, y_train: np.ndarray):
        x0 = x_train[y_train == 0]
        x1 = x_train[y_train == 1]
        u1 = np.mean(x0, axis=0)
        u2 = np.mean(x1, axis=0)
        cov = np.cov(x0, rowvar=False) + np.cov(x1, rowvar=False)
        w = np.linalg.inv(cov) @ (u2 - u1)
        self.w = w / np.linalg.norm(w)
        g0 = Gaussian()
        g0.fit(x0 @ self.w)
        g1 = Gaussian()
        g1.fit(x1 @ self.w)
        x = np.roots([g1.var - g0.var,
                      2*(g1.mean*g0.var - g0.mean*g1.var),
                      g1.var * g0.mean ** 2 - g0.var * g1.mean ** 2
                      - g1.var * g0.var * np.log(g1.var / g0.var)
                      ])
        if g0.mean < x[0] < g1.mean or g1.mean < x[0] < g0.mean:
            self.threshold = x[0]
        else:
            self.threshold = x[1]

    def project(self, x: np.ndarray):
        return x @ self.w

    def classify(self, x: np.ndarray):
        return (x @ self.w > self.threshold).astype(int)


class MultiFisherLinearDiscriminant:
    def __init__(self, W=None, threshold=None, n_classes=3):
        self.W = W
        self.threshold = threshold
        self.n_classes = n_classes

    def fit(self, x_train: np.ndarray, y_train: np.ndarray):
        cov_b = []  # between
        cov_w = []  # within
        mean = []
        mu = x_train.mean(0, keepdims=True) # 1 D
        for k in range(self.n_classes):
            x_k = x_train[y_train == k] # N_k D
            mean_k = np.mean(x_k, axis=0, keepdims=True)  # 1 D
            mean.append(mean_k)
            dist = x_k[:, None, :] - mean_k[:, :, None]  # N_K D D
            cov_k = np.einsum('nde,nde->ed', dist, dist)
            cov_w.append(cov_k)
            dist = mean_k - mu
            cov_k = (y_train == k).sum() * dist * dist.T
            cov_b.append(cov_k)
```

```
        cov_b = np.sum(cov_b, 0)    # D D
        cov_w = np.sum(cov_w, 0)
        A = np.linalg.inv(cov_w) @ cov_w
        _, vectors = np.linalg.eig(A)
        self.W = vectors[:, -(self.n_classes-1):]


#测试
def create_data(size=50, add_outlier=False, add_class=False):
    assert size % 2 == 0
    x0 = np.random.normal(size=size).reshape(-1, 2) - 1
    x1 = np.random.normal(size=size).reshape(-1, 2) + 1
    if add_outlier:
        x = np.random.normal(size=10).reshape(-1, 2) + np.array([5, 10])
        return np.concatenate([x0, x1, x]), np.concatenate([np.zeros(size//2), np.ones(size//2 + 5)])
    if add_class:
        x = np.random.normal(size=size).reshape(-1, 2) + 3
        return np.concatenate([x0, x1, x]), np.concatenate([np.zeros(size//2), np.ones(size//2), 2*np.ones(size,
    return np.concatenate([x0, x1]), np.concatenate([np.zeros(size//2), np.ones(size//2)])

model = FisherLinearDiscriminant()
model.fit(x_train, y_train)

plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
x1_test, x2_test = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
x_test = np.concatenate([x1_test, x2_test]).reshape(2, -1).T
y_pred = model.classify(x_test)
x = np.linspace(-5, 5, 20)
plt.contourf(x1_test, x2_test, y_pred.reshape(100, -1), alpha=0.2, levels=np.linspace(0,1,3))
plt.plot(x, x * model.w[1]/model.w[0], label='w', linestyle='--')
plt.title('Fisher Discriminant')
plt.gca().set_aspect('equal', adjustable='box')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.show()

plt.plot(x, x * model.w[1]/model.w[0], label='w', linestyle='--')
w = model.w
rollmat = np.zeros((2,2))
div = np.sqrt(w[0] ** 2 + w[1] ** 2)
rollmat[0,0] = w[0]/div
rollmat[0,1] = w[1]/div
rollmat[1,0] = -w[1]/div
rollmat[1,1] = w[0]/div
x_proj = x_train@w
x_proj = np.concatenate([x_proj[:,None], np.zeros_like(x_proj[:,None])],axis=-1).reshape(-1, 2)
#plt.scatter(x_proj[:,0], x_proj[:,1]-5, c=y_train)
x_roll = x_proj @ rollmat
plt.contourf(x1_test, x2_test, y_pred.reshape(100, -1), alpha=0.2, levels=np.linspace(0,1,3))
plt.scatter(x_roll[:, 0], x_roll[:,1], c=y_train)
plt.scatter(0, 0, marker='x', alpha=1)
plt.title('Projection')
plt.gca().set_aspect('equal', adjustable='box')
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.legend()
plt.show()
```





后记

有些地方还没整明白，明白了再回来补充.

参考文献

[1] Fukunaga, K. (1990). Introduction to Statistical Pattern Recognition (Second ed.). Academic Press. 441-454.
[2] Christopher M. Bishop.(2007). Pattern Recognition and Machine Learning. 187-192.