机器学习-第六讲课后作业

姓名：周宝航 学号：2120190442 专业：计算机科学与技术

一、问题描述

文件“Ionosphere Dataset.csv”中包含 351 个样本，前 200 个样本组成训练样本集，后 151 个样本组成测试样本集。每个样本由 34 个特征描述，两个类别分别使用 g 和 b 来标记。

1. 编程实现线性判别式最小二乘法算法。使用训练样本得到线性判别式模型，分别给出训练样本错误率和测试样本错误率。

2. 编程实现 Fisher 线性判别式算法。使用训练样本得到线性判别式模型，分别给出训练样本错误率和测试样本错误率。

3. 编程实现概率生成式模型（假设类条件概率密度为正太分布）。使用训练样本得到概率生成式模型，分别给出训练样本错误率和测试样本错误率。

4. 比较线性判别最小二乘法算法和 Fisher 线性判别式算法的结果，试解释原因。

二、基本思路

1. 我们根据线性判别式最小二乘法构建目标值矩阵与输入矩阵，然后利用训练集求解模型参数，得到训练集预测错误率。待训练结束，将模型应用在测试集得到测试集预测错误率。

2. 我们首先求解样本中各个类别的均值向量与协方差矩阵。然后

根据 Fisher 判别式函数求出最优参数，得到训练完成的模型。待模型训练完成，分别在训练集和测试集进行预测得到各自的错误率。

3. 根据生成式模型的最大似然解，我们对训练样本中各个类别的先验概率、特征均值向量与协方差矩阵进行计算。然后利用后验概率公式对样本所属类别进行判断。待训练完成后，我们将模型分别应用在训练集和测试集进行预测得到各自的错误率。

三、解题步骤

1. 算法描述

1.1 线性判别最小二乘算法

我们考虑 K 类分类问题，且目标变量 t 使用 1-of-K 二值编码。针对每类c_k使用线性模型表示：$y_k(x)=w_k^Tx+w_{k0},\ \ k=1,...,K$，转换为矩阵表示：$\mathbf{y}(x)=\tilde{W}^T\tilde{x}$，其中矩阵$\tilde{W}$的第 k 列是 D+1 维矢量$\tilde{w}_k=(w_{k0},w_k^T)^T$，$\tilde{x}$是对应的增广输入矢量$(1,x^T)^T$。由此，我们可得到未知输入 x 的类别是：$c_k=\arg\max_k y_k=\arg\max_k \tilde{w}_k^T\tilde{x}$。现给定训练数据集：$\{x_n,t_n\},n=1,...,N$，我们定义：矩阵 T 的第 n 行是矢量$t_n^T$，矩阵$\tilde{X}$的第 n 行是$\tilde{x}_n^T$。该模型的平方和误差函数为：$\mathrm{E}_D(\tilde{W})=\frac{1}{2}Tr\{(\tilde{X}\tilde{W}-T)^T(\tilde{X}\tilde{W}-T)\}$。我们令误差函数对$\tilde{W}$的导数为 0，得到解：$\tilde{W}=(\tilde{X}^T\tilde{X})^{-1}\tilde{X}^TT$。

1.2 Fisher 线性判别式算法

现给定类别c_1有N_1个点，类别c_2有N_2个点，可以求得各类别的均值向量：$m_1=\frac{1}{N_1}\sum_{n\in c_1}x_n$，$m_2=\frac{1}{N_2}\sum_{n\in c_2}x_n$。我们的优化目标为：类均值投

影之间具有较大分离间隔，且每个类别内部具有较小方差。由 Fisher 准则定义，我们有优化目标：$J(w)=\frac{(m_2-m_1)^2}{s_1^2+s_2^2}=\frac{w^T S_B w}{w^T S_w w}$，其中类间协方差矩阵为：$S_B=(m_2-m_1)(m_2-m_1)^T$，类内协方差矩阵为：$S_w=\sum_{n\in c_1}(x_n-m_1)(x_n-m_1)^T+\sum_{n\in c_2}(x_n-m_2)(x_n-m_2)^T$。考虑目标函数求导并去掉无关因素，我们得到 Fisher 线性判别式：$w\propto S_w^{-1}(m_2-m_1)$。对未知矢量 x，其预测类别为：$\mathrm{x}\in\begin{cases}c_1 & y(x)=w^T(x-m)>0\\ c_2 & \text{otherwise}\end{cases}$。

1.3 概率生成式模型算法

在两分类问题中，生成式概率模型对类条件密度 $p(x|c_k)$ 和类先验概率 $p(c_k)$ 进行建模。类 c_1 的后验概率为：

$$p(c_1|x)=\frac{p(x|c_1)p(c_1)}{p(x|c_1)p(c_1)+p(x|c_2)p(c_2)}=\frac{1}{1+\exp(-\alpha)}=\sigma(\alpha),$$

其中：$\alpha=\ln\frac{p(x|c_1)p(c_1)}{p(x|c_2)p(c_2)}$。我们假设类条件密度为高斯分布，且所有类别具有相同的协方差矩阵。根据前面的后验概率公式，我们重写两分类问题的后验概率为：$p(c_1|x)=\sigma(w^Tx+w_0)$，其中：$w=\Sigma^{-1}(\mu_1-\mu_2)$，$w_0=-\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1+\frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2+\ln\frac{p(c_1)}{p(c_2)}$。现给定数据集 $\{x_n,t_n\}$，$t_n=1$ 表示类别 c_1，$t_n=0$ 表示类别 c_2。我们求得各参数的最大似然解为：先验概率 $p(c_1)=\frac{N_1}{N_1+N_2}$，$p(c_2)=\frac{N_2}{N_1+N_2}$；概率分布参数估计为：$\mu_1=\frac{1}{N_1}\sum_{n=1}^{N}t_nx_n$，$\mu_2=\frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)x_n$，$\Sigma=\frac{1}{N}(\sum_{n\in c_1}(x_n-\mu_1)(x_n-\mu_1)^T+\sum_{n\in c_2}(x_n-\mu_2)(x_n-\mu_2)^T)$。

2. 算法实现

2.1 线性判别最小二乘算法

```
1.  class LeastSquareMethod(object):
2.      '''''
3.          最小二乘线性分类方法
4.      '''
5.      def __init__(self):
6.          self.params = None
7.
8.      def fit(self, X, y):
9.          # 统计类别
10.         classes = np.unique(y).tolist()
11.         m, _ = X.shape
12.         # 目标值矩阵
13.         T = np.zeros((m, len(classes)))
14.         for i in range(m):
15.             T[i, classes.index(y[i])] = 1.
16.
17.         c_X = np.hstack((np.ones((m, 1)), X))
18.         # 求解参数
19.         w = np.linalg.pinv(c_X.T @ c_X) @ c_X.T @ T
20.
21.         self.params = w
22.         self.labels = classes
23.         # 预测训练集误差
24.         pred = self.predict(X)
25.         acc = np.mean(pred == y)
26.         print(f"\n{self.__class__.__name__}\n 训练错误率: {1-acc}")
27.
28.     def predict(self, X):
29.         m, _ = X.shape
30.         X = np.hstack((np.ones((m, 1)), X))
31.         y = [
32.             self.labels[i]
33.             for i in np.argmax(X @ self.params, axis=-1)
34.         ]
35.         return y
```

上面是我们对线性判别最小二乘算法的实现类，其中第 19 行是利用求解方程对最优参数进行求解。在预测目标值时，我们利用判别式函数得到最大值对应的类别作为预测类别。

2.2 Fisher 线性判别式算法

```
class LinearDiscriminantAnalysis(object):
    '''''
        线性判别分析方法
    '''
    def __init__(self):
        self.params = None

    def fit(self, X, y):
        _, n = X.shape
        # 全局均值
        mu_t = np.mean(X, axis=0)
        # 统计类别
        classes = np.unique(y).tolist()
        # 类内散度矩阵
        c_mu = {}
        Sw = np.zeros((n, n))
        for c in classes:
            index = np.where(y == c)[0]
            subData = X[index, :]
            # 类别均值向量
            mu = np.mean(subData, axis=0, keepdims=True)
            c_mu[c] = mu
            # 类别协方差矩阵
            Sw += (subData - mu).T @ (subData - mu)
        # 求解参数
        w = np.linalg.pinv(Sw) @ (c_mu[classes[1]] - c_mu[classes[0]]).T
        self.mu = mu_t
        self.params = w
        self.c_mu = c_mu
        # 预测训练误差
        pred = self.predict(X)
        acc = np.mean(pred == y)
        print(f"\n{self.__class__.__name__}\n 训练错误率: {1-acc}")
    def predict(self, X):
        # 判别函数
        distance = (X - self.mu) @ self.params
        # 预测类别
        y = [
            list(self.c_mu.keys())[int(distance[i, 0] > 0)]
            for i in range(len(distance))
        ]
        return y
```

上面是我们对 Fisher 线性判别式算法的实现类，其中第 16-24 行是对类内散度矩阵进行求解。而第 26 行是利用判别式函数求解参数。

2.3 概率生成式模型算法

```
1.  class GenerativeModel(object):
2.      '''''
3.          概率生成式模型
4.      '''
5.      def __init__(self):
6.          self.params = None
7.
8.      def fit(self, X, y):
9.          _, n = X.shape
10.         cache = {}
11.         # 统计类别
12.         classes = np.unique(y).tolist()
13.         for c in classes:
14.             index = np.where(y == c)[0]
15.             subData = X[index, :]
16.             tmp = {}
17.             # 类别均值向量
18.             mu = np.mean(subData, axis=0, keepdims=True)
19.             tmp['mean'] = mu
20.             # 类别协方差矩阵
21.             tmp['var'] = (subData - mu).T @ (subData - mu) / len(subData)
22.             # 先验概率
23.             tmp['prior'] = len(subData) / len(X)
24.             cache[c] = tmp
25.         # 类别加权协方差矩阵
26.         sigma = np.zeros((n, n))
27.         for _, v in cache.items():
28.             sigma += v['prior'] * v['var']
29.         # 类别 1 均值向量
30.         mu1 = cache[classes[0]]['mean'].T
31.         # 类别 2 均值向量
32.         mu2 = cache[classes[1]]['mean'].T
33.         # 类别 1 先验概率
34.         p1 = cache[classes[0]]['prior']
35.         # 类别 2 先验概率
36.         p2 = cache[classes[1]]['prior']
37.         # 求解参数
38.         w = np.linalg.pinv(sigma) @ (mu1 - mu2)
```

```
39.         w0 = -0.5 * mu1.T @ np.linalg.pinv(sigma) @ mu1\
40.              +0.5 * mu2.T @ np.linalg.pinv(sigma) @ mu2 + np.log(p1 / p2)
41.
42.         self.labels = classes
43.         self.params = {'w':w, 'w0':w0}
44.         # 训练集测试误差
45.         pred = self.predict(X)
46.         acc = np.mean(pred == y)
47.         print(f"\n{self.__class__.__name__}\n 训练错误率: {1-acc}")
48.
49.     def predict(self, X):
50.         w = self.params['w']
51.         w0 = self.params['w0']
52.         # 后验概率
53.         prob = 1 / (1 + np.exp(- (X @ w + w0)))
54.         # 预测类别
55.         y = [
56.             self.labels[int(p < 0.5)]
57.             for p in prob
58.         ]
59.         return y
```

上面是我们对概率生成式模型的实现类，其中第 13-24 行是对样本中各类别求其均值矢量、协方差矩阵和先验概率。而第 38-40 行是利用模型最大似然解对参数进行求解。

四、结果与分析

1. 实验内容与步骤

本次实验数据集共包含 351 个样本，每个样本由 34 个特征描述。样本一共分为“g”和“b”两类。在实验中，我们将数据集进行如下划分：前 200 个样本组成训练集，后 151 个样本组成测试集。我们分别采用线性判别最小二乘法模型、Fisher 线性判别式模型和概率生成式模型在训练集进行训练，并得到训练集上的预测错误率。然后，我

们再将这些训练完成的模型作用在测试集，得到测试集上的预测错误率。预测错误率作为评价指标衡量模型的优劣。模型预测错误率定义为：$\text{error} = 1 - \frac{1}{\text{N}} \sum_{i=1}^{N} y_i == \hat{y}_i$，只统计预测标签与实际标签一致的样本数量。

2. 实验结果

方法	线性判别 最小二乘法	Fisher 线性 判别式算法	概率 生成式模型
训练集错误率	0.125	0.13	0.125
测试集错误率	0.0927	0.0927	0.0927

表 1 模型错误率评价

上表是对线性判别最小二乘模型、Fisher 线性判别式模型与概率生成式模型的性能评价比较结果。Fisher 线性判别式模型的训练集错误率比另外两个模型的结果要差，而三者的测试集错误率是一样的。在该数据集上，这三种模型的泛化能力是相同的，只是 Fisher 线性判别模型在拟合训练集的能力上不如另外两种模型。这是因为 Fisher 线性判别模型的贝叶斯最优解的限制条件是：所有类别的协方差矩阵是相同的。而实际数据集中，这种情况是很难保证的。

3. 结果分析

我们知道：在二分类问题中，Fisher 线性判别式模式是线性判别最小二乘法的特殊情况。但实验中，Fisher 线性判别模型的训练集错误率是高于最小二乘法的，两者的测试集错误率是相同的。这种情况

出现的原因是：Fisher 线性判别模型的贝叶斯最优解是要求数据集中各类别的协方差矩阵是一致的。但该数据集中没有满足这个要求。但是，这种偏差没有影响模型的泛化能力，两者的测试错误率是相同的。还有一个原因可能是：测试集样本的复杂度不高，训练集样本复杂程度可以覆盖测试集样本。