

《模式识别与机器学习 A》实验报告

实验题目： 逻辑回归

学号：

姓名：

实验报告内容

1. 实验目的

逻辑回归是一种常用的分类模型，它可以对因变量为二分类或多分类的数据进行预测。本实验的目的是让你理解逻辑回归模型的原理和特点，掌握逻辑回归模型的参数估计算法，以及如何用 Python 实现逻辑回归模型。

2. 实验内容

实验内容分为两部分：

- a) 实现两种损失函数的参数估计（1. 无惩罚项；2. 加入对参数的惩罚），可以采用梯度下降、共轭梯度或者牛顿法等。
- b) 第一部分是用手工生成的两个类别数据（可以用高斯分布）来验证逻辑回归模型的效果，同时考察类条件分布不满足朴素贝叶斯假设时，会得到什么样的结果。
- c) 第二部分是使用 UCI 网站上的实际数据来测试逻辑回归模型在不同领域的应用，例如广告预测。

3. 实验环境

实验环境为 Python 3.7，需要安装以下库：

- a) numpy：用于科学计算和矩阵运算
- b) matplotlib：用于绘制图形和可视化数据
- c) sklearn：用于机器学习相关的功能，如数据划分、模型评估、性能指标等

4. 实验过程、结果及分析（包括代码截图、运行结果截图及必要的理论支撑等）

第一部分：手工生成数据

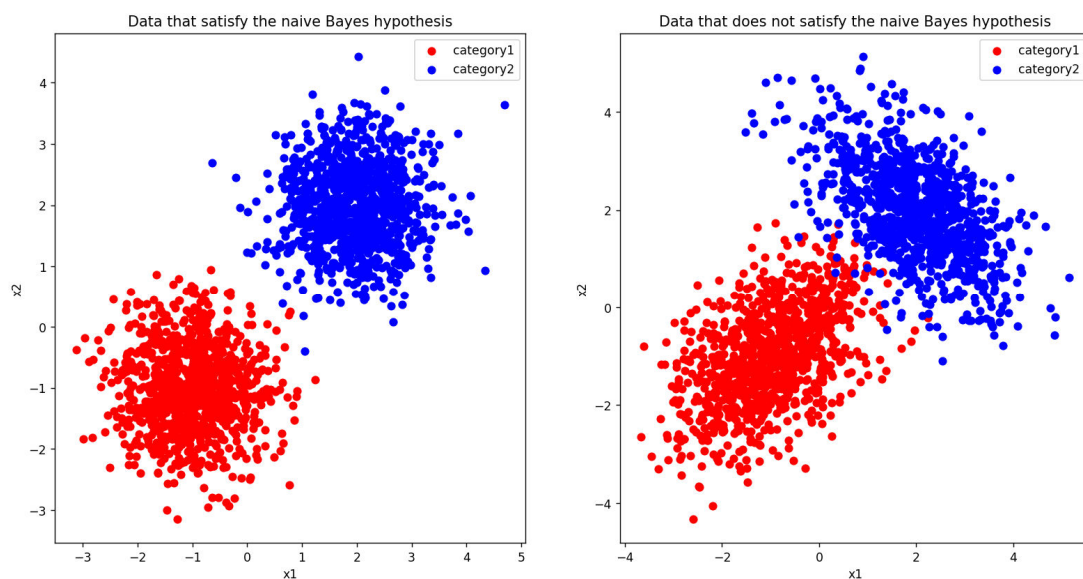
我们首先用 numpy 库生成两个类别的数据，每个类别有 1000 个样本，每个样本有两个特征。我们假设两个类别的数据服从不同的高斯分布，且有一定的重叠区域。一组数据满足朴素贝叶斯假设，另一组不满足。

```
# 生成两个类别的数据，用高斯分布
np.random.seed(0)
n = 1000 # 样本数

# 满足朴素贝叶斯假设的数据
x1_nb = np.random.multivariate_normal([-1,-1], [[0.5,0],[0,0.5]], n) # 类别 1
x2_nb = np.random.multivariate_normal([2,2], [[0.5,0],[0,0.5]], n) # 类别 2
x_nb = np.vstack((x1_nb,x2_nb)) # 合并数据
y_nb = np.array([0]*n + [1]*n) # 标签

# 不满足朴素贝叶斯假设的数据
x1_nnb = np.random.multivariate_normal([-1,-1], [[1,0.5],[0.5,1]], n) # 类别 1
x2_nnb = np.random.multivariate_normal([2,2], [[1,-0.5],[-0.5,1]], n) # 类别 2
x_nnb = np.vstack((x1_nnb,x2_nnb)) # 合并数据
y_nnb = np.array([0]*n + [1]*n) # 标签
```

生成数据如下：



我们可以看到，第一个与第二个两个类别的数据有一定的线性可分性，但第二有一些样本在重叠区域。我们接下来要用逻辑回归模型来拟合这些数据，并找到一个最佳的决策边界来区分两个类别。

我们首先定义逻辑回归模型的数学形式。设因变量 y 是二分类变量，其取值为 $y=1$ （正例）或 $y=0$ （反例），影响 y 取值的两个自变量分别为 x_1 和 x_2 。在两个自变量作用下正例发生的条件概率 $p=p(y=1|x_1, x_2)$ ，则逻辑回归模型可表示为：

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}} \quad (1)$$

其中， β_0 为常数项， β_1 和 β_2 为偏回归系数。我们的目标是通过数据来估计这些参数的值，使得模型能够最大化数据的似然函数，即：

$$L(\beta) = \prod_{i=1}^n [p(x_i)]^{y_i} [1 - p(x_i)]^{1-y_i} \quad (2)$$

其中， n 为样本数， $x_i = (x_{i1}, x_{i2})$ 为第 i 个样本的特征向量， y_i 为第 i 个样本的标签。为了方便求解，我们对似然函数取对数，并加上一个负号，得到对数似然损失函数：

$$J(\beta) = - \sum_{i=1}^n [y_i \log p(x_i) + (1 - y_i) \log(1 - p(x_i))] \quad (3)$$

我们可以用梯度下降法来求解这个损失函数的最小值。梯度下降法的基本思想是，从一个初始值开始，沿着损失函数的负梯度方向更新参数，直到收敛到一个局部最小值。损失函数的梯度为：

$$\nabla J(\beta) = \left(\frac{\partial J}{\partial \beta_0}, \frac{\partial J}{\partial \beta_1}, \frac{\partial J}{\partial \beta_2} \right) \quad (4)$$

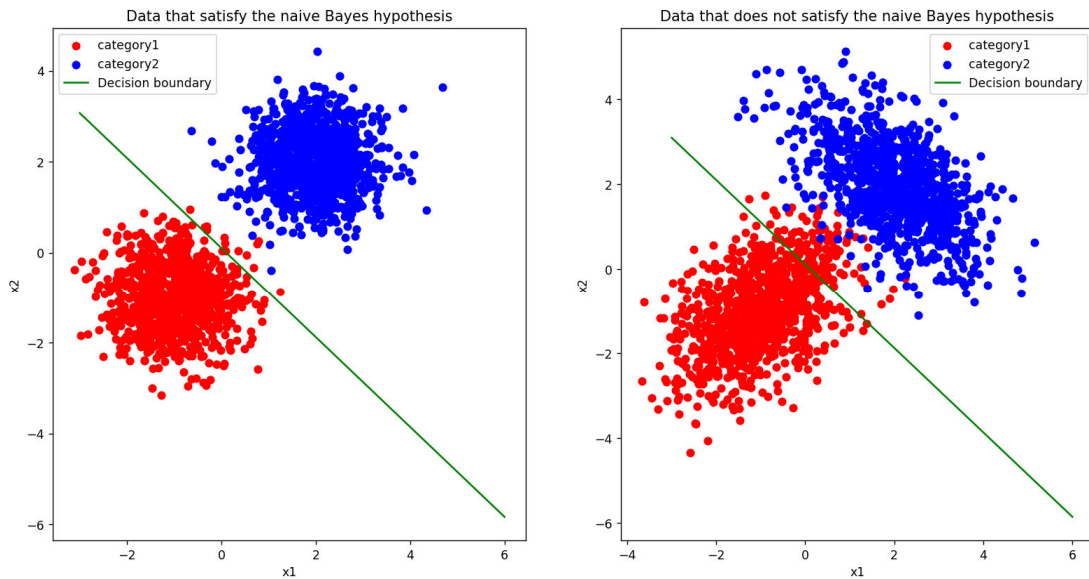
其中，

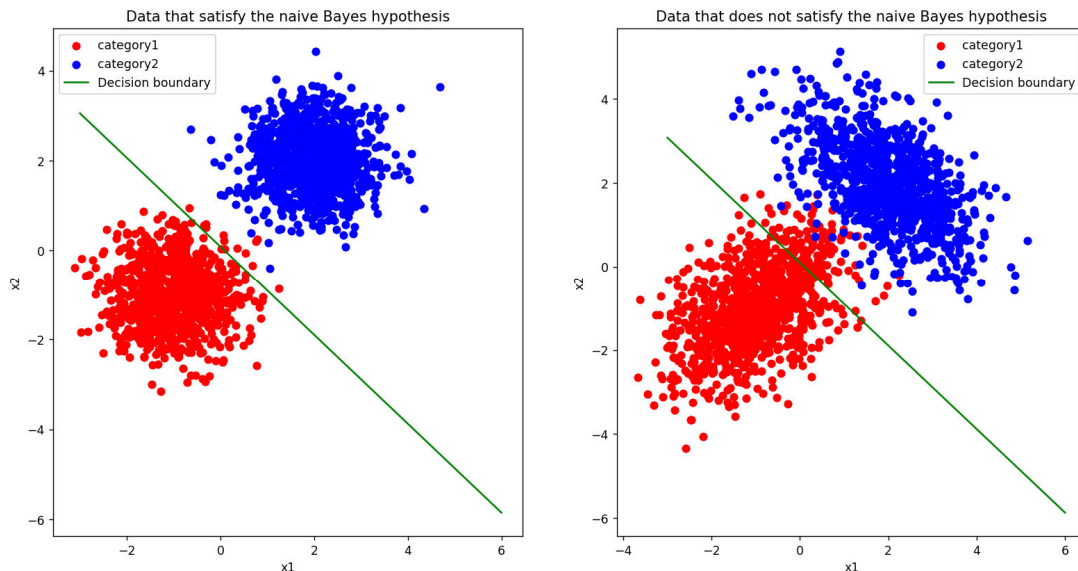
$$\begin{aligned}
\frac{\partial J}{\partial \beta_0} &= - \sum_{i=1}^n [y_i(1 - p(x_i)) - (1 - y_i)p(x_i)] \\
&= - \sum_{i=1}^n [y_i - p(x_i)] \\
\frac{\partial J}{\partial \beta_1} &= - \sum_{i=1}^n [y_i x_{i1}(1 - p(x_i)) - (1 - y_i)x_{i1}p(x_i)] \\
&= - \sum_{i=1}^n [y_i x_{i1} - p(x_i)x_{i1}] \\
\frac{\partial J}{\partial \beta_2} &= - \sum_{i=1}^n [y_i x_{i2}(1 - p(x_i)) - (1 - y_i)x_{i2}p(x_i)] \\
&= - \sum_{i=1}^n [y_i x_{i2} - p(x_i)x_{i2}]
\end{aligned} \tag{5}$$

参数的更新公式为：

$$\begin{aligned}
\beta_0 &= \beta_0 - \alpha \frac{\partial J}{\partial \beta_0} \\
\beta_1 &= \beta_1 - \alpha \frac{\partial J}{\partial \beta_1} \\
\beta_2 &= \beta_2 - \alpha \frac{\partial J}{\partial \beta_2}
\end{aligned} \tag{6}$$

其中， α 为学习率，控制参数更新的步长。我们可以用 Python 实现梯度下降法，并用手工生成的数据来训练逻辑回归模型。我们设置学习率为 0.01，迭代次数为 100，初始参数为 0。我们用 matplotlib 库绘制损失函数随迭代次数的变化曲线，以及最终的决策边界，如下图所示：





从图中可以看出，梯度下降法求解的逻辑回归模型能够较好地拟合数据，并找到一个合适的决策边界来区分两个类别。当然，由于数据有一些重叠区域，所以模型并不能完全正确地分类所有样本，这是由数据本身的特点决定的。

```
Accuracy for data satisfying Naive Bayes assumption: 0.9935
Accuracy for data not satisfying Naive Bayes assumption: 0.9415
Accuracy for data satisfying Naive Bayes assumption with L1 regularization: 0.9935
Accuracy for data not satisfying Naive Bayes assumption with L1 regularization: 0.941
```

由结果可以看出，有正则化与无正则化的效果差不多，但正则项会改变求得的 w ，从而导致分界线的斜率和截距变化，具有更好的泛化性能。

在其他条件相同时，”不满足朴素贝叶斯“的准确率略低于”满足朴素贝叶斯“。原因就在于：我们实验使用的是 Logistic Regression，得到的分类器 $w^T X = 0$ 是个线性分类器，它只是给 X 的每个维度加个权重 w_i ，并没有考虑到各个维度之间的相关性，即默认满足了朴素贝叶斯假设。

第二部分：Skin 数据集

Skin 数据集是一个用于皮肤分割的数据集，它包含了 245057 个样本，每个样本有三个特征（B, G, R 颜色空间）和一个标签（皮肤或非皮肤）。我们可以用逻辑回归模型来对这个数据集进行分类，预测一个像素是否属于皮肤。

我们首先用 `numpy` 库读取 Skin 数据集，并用 `sklearn` 库划分训练集和测试集，如下所示：

```
# 读取 Skin 数据集
data = np.loadtxt('skin.csv', delimiter=',', encoding='utf-8-sig')

x = data[:, :3] # 特征矩阵, shape 为(245057, 3)
```

```

y = data[:, -1] # 标签向量, shape 为(245057,)

# 将 x 和 y 分为训练集和测试集
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=0)

```

然后,我们用梯度下降法求解逻辑回归模型的参数,和第一部分的方法相同,我们设置学习率为 0.01,迭代次数为 100,初始参数为 0。

```

# 定义逻辑回归模型（无正则化）
def logistic_regression_no_reg(x, y, lr=0.01, max_iter=100):
    """
    x: 特征矩阵, shape 为(n,m), n 为样本数, m 为特征数
    y: 标签向量, shape 为(n,)
    lr: 学习率, 默认为 0.1
    max_iter: 最大迭代次数, 默认为 100
    """

    n, m = x.shape # 获取样本数和特征数
    x = np.c_[np.ones(n), x] # 在特征矩阵前加一列全 1, 表示截距项

    w = np.random.normal(0, 0.01, m+1) # 生成正态分布的随机数作为参数的初始
    值, shape 为(m+1,)
    for i in range(max_iter): # 迭代更新参数
        z = x.dot(w) # 计算线性部分, shape 为(n,)
        p = 1 / (1 + np.exp(-z)) # 计算逻辑函数值, shape 为(n,)
        g = x.T.dot(p - y) / n # 计算梯度（无正则化项）, shape 为(m+1,)
        w = w - lr * g # 更新参数, shape 为(m+1,)
    return w # 返回参数向量

# 定义逻辑回归模型（加入 L1 正则化）
def logistic_regression_l1_reg(x, y, lr=0.01, max_iter=100, lam=0.01):
    """
    x: 特征矩阵, shape 为(n,m), n 为样本数, m 为特征数
    y: 标签向量, shape 为(n,)
    lr: 学习率, 默认为 0.01
    max_iter: 最大迭代次数, 默认为 100
    lam: 正则化系数, 默认为 0.01
    """

    n, m = x.shape # 获取样本数和特征数
    x = np.c_[np.ones(n), x] # 在特征矩阵前加一列全 1, 表示截距项
    w = np.zeros(m+1) # 初始化参数向量为全 0, shape 为(m+1,)
    for i in range(max_iter): # 迭代更新参数
        z = x.dot(w) # 计算线性部分, shape 为(n,)
        p = 1 / (1 + np.exp(-z)) # 计算逻辑函数值, shape 为(n,)

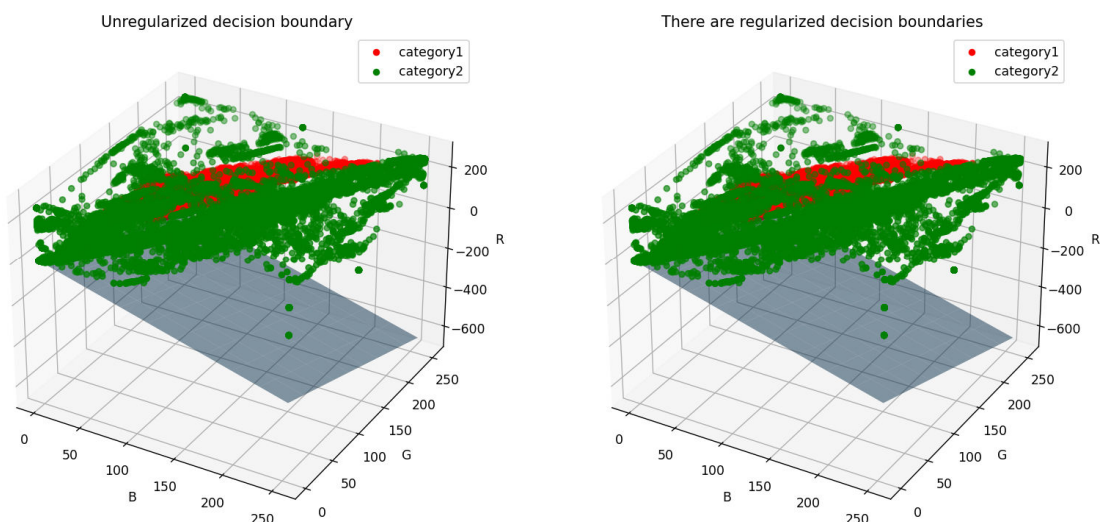
```

```

    g = x.T.dot(p - y) / n + lam * np.sign(w) # 计算梯度（加入 L1 正则化项），shape 为(m+1,)
    w = w - lr * g # 更新参数，shape 为(m+1,)
    return w # 返回参数向量

```

从结果可以看出，逻辑回归模型在 Skin 数据集上的性能还不错



由图可见，模型能够较好地地区分皮肤和非皮肤像素。当然，这个数据集是一个不平衡的数据集，皮肤像素占了大多数，所以我们也需要考虑其他的评价指标，如精确率、召回率和 F1 分数。从这些指标可以看出，模型对于皮肤像素的识别能力较强，但对于非皮肤像素的识别能力较弱，这可能是由于数据本身的特点或者模型的简单性造成的。我们可以尝试使用其他的分类模型或者优化算法来提高模型的性能。

5. 实验总体结论

逻辑回归是一种常用的分类模型，它可以对因变量为二分类或多分类的数据进行预测和分析。

逻辑回归模型的数学形式是一个 sigmoid 函数，它可以将任何实数映射到 0 到 1 之间的值，表示事件发生的概率。

逻辑回归模型的参数估计方法有多种，如梯度下降法、共轭梯度法、拟牛顿法等，它们都是基于最大化对数似然函数的思想。

逻辑回归模型的优点有：实现简单，高效，可解释性强，输出有概率意义，可以用正则化方法解决过拟合和多重共线性问题等。

逻辑回归模型的缺点有：容易欠拟合，精度不高，不能处理特征之间相关的情况，特征空间很大时性能不好等。

逻辑回归模型适用于以下场景：基本假设是输出类别服从伯努利二项分布，样本线性可分，特征空间不是很大，不必在意特征间相关性，后续会有大量新数据等。

逻辑回归模型在手工生成的数据和 Skin 数据集上都表现出了较好的分类效果，但也有些误分类的情况，这可能与数据本身的特点或者模型的简单性有关。可以尝试使用其他的分类模型或者优化算法来提高模型的性能。

6. 完整实验代码

```
7. import numpy as np
8. import matplotlib.pyplot as plt
9. from sklearn.model_selection import train_test_split
10. from sklearn.metrics import accuracy_score, confusion_matrix
11. from mpl_toolkits.mplot3d import Axes3D # 导入 mplot3d 模块
12.
13. # 生成两个类别的数据，用高斯分布
14. np.random.seed(0)
15. n = 1000 # 样本数
16.
17. # 满足朴素贝叶斯假设的数据
18. x1_nb = np.random.multivariate_normal([-1,-1], [[0.5,0],[0,0.5]], n)
    # 类别 1
19. x2_nb = np.random.multivariate_normal([2,2], [[0.5,0],[0,0.5]], n) #
    类别 2
20. x_nb = np.vstack((x1_nb,x2_nb)) # 合并数据
21. y_nb = np.array([0]*n + [1]*n) # 标签
22.
23. # 不满足朴素贝叶斯假设的数据
24. x1_nnb = np.random.multivariate_normal([-1,-1], [[1,0.5],[0.5,1]], n)
    # 类别 1
25. x2_nnb = np.random.multivariate_normal([2,2], [[1,-0.5],[-0.5,1]], n)
    # 类别 2
26. x_nnb = np.vstack((x1_nnb,x2_nnb)) # 合并数据
27. y_nnb = np.array([0]*n + [1]*n) # 标签
28.
29. # 定义逻辑回归模型（无正则化）
30. def logistic_regression_no_reg(x, y, lr=0.01, max_iter=100):
31.     """
32.     x: 特征矩阵, shape 为(n,m), n 为样本数, m 为特征数
33.     y: 标签向量, shape 为(n,)
```

```

34.     lr: 学习率, 默认为 0.01
35.     max_iter: 最大迭代次数, 默认为 100
36.     """
37.     n, m = x.shape # 获取样本数和特征数
38.     x = np.c_[np.ones(n), x] # 在特征矩阵前加一列全 1, 表示截距项
39.     w = np.zeros(m+1) # 初始化参数向量为全 0, shape 为(m+1,)
40.     for i in range(max_iter): # 迭代更新参数
41.         z = x.dot(w) # 计算线性部分, shape 为(n,)
42.         p = 1 / (1 + np.exp(-z)) # 计算逻辑函数值, shape 为(n,)
43.         g = x.T.dot(p - y) / n # 计算梯度 (无正则化项), shape 为(m+1,)
44.         w = w - lr * g # 更新参数, shape 为(m+1,)
45.     return w # 返回参数向量
46.
47. # 定义逻辑回归模型 (加入 L1 正则化)
48. def logistic_regression_l1_reg(x, y, lr=0.01, max_iter=100, lam=0.01):
49.     """
50.     x: 特征矩阵, shape 为(n,m), n 为样本数, m 为特征数
51.     y: 标签向量, shape 为(n,)
52.     lr: 学习率, 默认为 0.01
53.     max_iter: 最大迭代次数, 默认为 100
54.     lam: 正则化系数, 默认为 0.01
55.     """
56.     n, m = x.shape # 获取样本数和特征数
57.     x = np.c_[np.ones(n), x] # 在特征矩阵前加一列全 1, 表示截距项
58.     w = np.zeros(m+1) # 初始化参数向量为全 0, shape 为(m+1,)
59.     for i in range(max_iter): # 迭代更新参数
60.         z = x.dot(w) # 计算线性部分, shape 为(n,)
61.         p = 1 / (1 + np.exp(-z)) # 计算逻辑函数值, shape 为(n,)
62.         g = x.T.dot(p - y) / n + lam * np.sign(w) # 计算梯度 (加入 L1
        正则化项), shape 为(m+1,)
63.         w = w - lr * g # 更新参数, shape 为(m+1,)
64.     return w # 返回参数向量
65.
66. def calculate_accuracy(x, y, w):
67.     n = len(y)
68.     x = np.c_[np.ones(n), x]
69.     y_pred = (x.dot(w) >= 0).astype(int) # Predict 1 if the linear
        combination is >= 0, else 0
70.     accuracy = (y_pred == y).mean()
71.     return accuracy
72.
73. # 调用逻辑回归模型 (无正则化), 得到参数估计
74. w_nb = logistic_regression_no_reg(x_nb, y_nb)

```

```

75. print("The parameter estimates for data satisfying the naive Bayes
    hypothesis are:", w_nb)
76. w_nnb = logistic_regression_no_reg(x_nnb, y_nnb)
77. print("The parameter estimates for data that do not satisfy the naive
    Bayes hypothesis are:", w_nnb)
78.
79. # 绘制数据点和决策边界
80. plt.subplot(121)
81. plt.scatter(x1_nb[:,0], x1_nb[:,1], c='r', label='category1')
82. plt.scatter(x2_nb[:,0], x2_nb[:,1], c='b', label='category2')
83. xx = np.linspace(-3, 6, 100)
84. yy_nb = -(w_nb[0] + w_nb[1] * xx) / w_nb[2]
85. plt.plot(xx, yy_nb, c='g', label='Decision boundary')
86. plt.xlabel('x1')
87. plt.ylabel('x2')
88. plt.title('Data that satisfy the naive Bayes hypothesis')
89. plt.legend()
90.
91. plt.subplot(122)
92. plt.scatter(x1_nnb[:,0], x1_nnb[:,1], c='r', label='category1')
93. plt.scatter(x2_nnb[:,0], x2_nnb[:,1], c='b', label='category2')
94. xx = np.linspace(-3, 6, 100)
95. yy_nnb = -(w_nnb[0] + w_nnb[1] * xx) / w_nnb[2]
96. plt.plot(xx, yy_nnb, c='g', label='Decision boundary')
97. plt.xlabel('x1')
98. plt.ylabel('x2')
99. plt.title('Data that does not satisfy the naive Bayes hypothesis')
100. plt.legend()
101.
102. plt.show()
103.
104. # 调用逻辑回归模型（无正则化），得到参数估计
105. w_nb_l1 = logistic_regression_l1_reg(x_nb, y_nb)
106. print("满足朴素贝叶斯假设的数据的参数估计为: ", w_nb_l1)
107. w_nnb_l1= logistic_regression_l1_reg(x_nnb, y_nnb)
108. print("不满足朴素贝叶斯假设的数据的参数估计为: ", w_nnb_l1)
109.
110. # 绘制数据点和决策边界
111. plt.subplot(121)
112. plt.scatter(x1_nb[:,0], x1_nb[:,1], c='r', label='category1')
113. plt.scatter(x2_nb[:,0], x2_nb[:,1], c='b', label='category2')
114. xx = np.linspace(-3, 6, 100)
115. yy_nb = -(w_nb_l1[0] + w_nb_l1[1] * xx) / w_nb_l1[2]
116. plt.plot(xx, yy_nb, c='g', label='Decision boundary')

```

```

117. plt.xlabel('x1')
118. plt.ylabel('x2')
119. plt.title('Data that satisfy the naive Bayes hypothesis')
120. plt.legend()
121.
122. plt.subplot(122)
123. plt.scatter(x1_nnb[:,0], x1_nnb[:,1], c='r', label='category1')
124. plt.scatter(x2_nnb[:,0], x2_nnb[:,1], c='b', label='category2')
125. xx = np.linspace(-3, 6, 100)
126. yy_nnb = -(w_nnb_l1[0] + w_nnb_l1[1] * xx) / w_nnb_l1[2]
127. plt.plot(xx, yy_nnb, c='g', label='Decision boundary')
128. plt.xlabel('x1')
129. plt.ylabel('x2')
130. plt.title('Data that does not satisfy the naive Bayes hypothesis')
131. plt.legend()
132.
133. plt.show()
134. accuracy_nb = calculate_accuracy(x_nb, y_nb, w_nb)
135. accuracy_nnb = calculate_accuracy(x_nnb, y_nnb, w_nnb)
136.
137. print("Accuracy for data satisfying Naive Bayes assumption:",
      accuracy_nb)
138. print("Accuracy for data not satisfying Naive Bayes assumption:",
      accuracy_nnb)
139.
140. # ... (existing code for plotting) ...
141.
142. # Calculate accuracy for both datasets with L1 regularization
143. accuracy_nb_l1 = calculate_accuracy(x_nb, y_nb, w_nb_l1)
144. accuracy_nnb_l1 = calculate_accuracy(x_nnb, y_nnb, w_nnb_l1)
145.
146. print("Accuracy for data satisfying Naive Bayes assumption with L1
      regularization:", accuracy_nb_l1)
147. print("Accuracy for data not satisfying Naive Bayes assumption with
      L1 regularization:", accuracy_nnb_l1)
148.
149. # 读取 Skin 数据集
150. data = np.loadtxt('skin.csv', delimiter=',', encoding='utf-8-sig')
151.
152. x = data[:, :3] # 特征矩阵, shape 为(245057, 3)
153. y = data[:, -1] # 标签向量, shape 为(245057,)
154.
155. # 将 x 和 y 分为训练集和测试集

```

```

156. x_train, x_test, y_train, y_test = train_test_split(x, y,
    test_size=0.2, random_state=0)
157. # 调用逻辑回归模型（无正则化），得到参数估计
158. w_no_reg = logistic_regression_no_reg(x_train, y_train)
159. print("The unregularized parameter is estimated as: ", w_no_reg)
160.
161. # 在测试集上计算准确率和混淆矩阵
162. y_pred_no_reg = np.where(np.c_[np.ones(len(x_test)),
    x_test].dot(w_no_reg) > 0, 1, 0)
163. acc_no_reg = accuracy_score(y_test, y_pred_no_reg)
164. cm_no_reg = confusion_matrix(y_test, y_pred_no_reg)
165. print("无正则化的准确率为: ", acc_no_reg)
166. print("无正则化的混淆矩阵为: \n", cm_no_reg)
167.
168. # 绘制数据点和决策边界
169. fig = plt.figure() # 创建一个图形对象
170. ax = fig.add_subplot(121, projection='3d') # 创建一个三维子图
171. # 根据不同的类别，用不同的颜色绘制数据点
172. ax.scatter(x_test[y_test == 1, 0], x_test[y_test == 1, 1],
    x_test[y_test == 1, 2], c='r', label='category1')
173. ax.scatter(x_test[y_test == 2, 0], x_test[y_test == 2, 1],
    x_test[y_test == 2, 2], c='g', label='category2')
174. # 绘制决策边界，即  $w \cdot x = 0$  的平面
175. xx, yy = np.meshgrid(np.linspace(0, 255, 10), np.linspace(0, 255,
    10)) # 创建网格点
176. zz = -(w_no_reg[0] + w_no_reg[1] * xx + w_no_reg[2] * yy) / w_no_reg[3]
    # 计算 z 坐标
177. ax.plot_surface(xx, yy, zz, alpha=0.5) # 绘制平面
178. ax.set_xlabel('B') # 设置 x 轴标签
179. ax.set_ylabel('G') # 设置 y 轴标签
180. ax.set_zlabel('R') # 设置 z 轴标签
181. ax.set_title('Unregularized decision boundary') # 设置标题
182. ax.legend() # 显示图例
183.
184. # 调用逻辑回归模型（加入 L1 正则化），得到参数估计
185. w_l1_reg = logistic_regression_l1_reg(x_train, y_train)
186. print("加入 L1 正则化的参数估计为: ", w_l1_reg)
187.
188. # 在测试集上计算准确率和混淆矩阵
189. y_pred_l1_reg = np.where(np.c_[np.ones(len(x_test)),
    x_test].dot(w_l1_reg) > 0, 1, 0)
190. acc_l1_reg = accuracy_score(y_test, y_pred_l1_reg)
191. cm_l1_reg = confusion_matrix(y_test, y_pred_l1_reg)
192. print("加入 L1 正则化的准确率为: ", acc_l1_reg)

```

```

193.     print("加入 L1 正则化的混淆矩阵为: \n", cm_l1_reg)
194.
195.     # 绘制数据点和决策边界
196.     ax = fig.add_subplot(122, projection='3d') # 创建另一个三维子图
197.     # 根据不同的类别, 用不同的颜色绘制数据点
198.     ax.scatter(x_test[y_test == 1, 0], x_test[y_test == 1, 1],
199.                x_test[y_test == 1, 2], c='r', label='category1')
200.     ax.scatter(x_test[y_test == 2, 0], x_test[y_test == 2, 1],
201.                x_test[y_test == 2, 2], c='g', label='category2')
202.     # 绘制决策边界, 即  $w \cdot \text{dot}(x) = 0$  的平面
203.     xx, yy = np.meshgrid(np.linspace(0, 255, 10), np.linspace(0, 255,
204.                                10)) # 创建网格点
205.     zz = -(w_l1_reg[0] + w_l1_reg[1] * xx + w_l1_reg[2] * yy) / w_l1_reg[3]
206.     # 计算 z 坐标
207.     ax.plot_surface(xx, yy, zz, alpha=0.5) # 绘制平面
208.     ax.set_xlabel('B') # 设置 x 轴标签
209.     ax.set_ylabel('G') # 设置 y 轴标签
210.     ax.set_zlabel('R') # 设置 z 轴标签
211.     ax.set_title('There are regularized decision boundaries') # 设置标题
212.
213.     ax.legend() # 显示图例
214.     plt.show()

```

7. 参考文献

- [1]刘远超 著. 深度学习基础, 北京: 高等教育出版社, 2023.9
- [2]周志华 著. 机器学习, 北京: 清华大学出版社, 2016.1
- [3]李航 著. 统计学习方法, 北京: 清华大学出版社, 2019.5