《模式识别与机器学习 A》实验报告

实验题目：逻辑回归

学号：2021110683

姓名：徐柯炎

实验报告内容

1. 实验目的

理解逻辑回归模型，掌握逻辑回归模型的参数估计算法。

2. 实验内容

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。

要求：1.可以手工生成两个类别数据（可以用高斯分布），验证你的算法。考察类条件分布不满足朴素贝叶斯假设，会得到什么样的结果。

2. 逻辑回归有广泛的用处，例如广告预测。可以到 UCI 网站上，找一些实际数据加以测试。

3. 实验环境

Windows10; python3.9;PyCharm 2021.2.2

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）

（1） 逻辑回归模型原理

逻辑回归主要是针对二分类预测问题，通过直接计算后验概率来对类别进行判断，当计算结果大于 0.5 时，判别结果为 1，结果小于 0.5 则判别为 0；因此， 逻辑回归实际是一种概率估计。

为了能将数据信息映射到 0 到 1 之间，我们需要寻找一种函数，这就是 sigmoid 函数，也就是判别函数：

$$g(z) = \frac{1}{1 + e^{-z}}$$

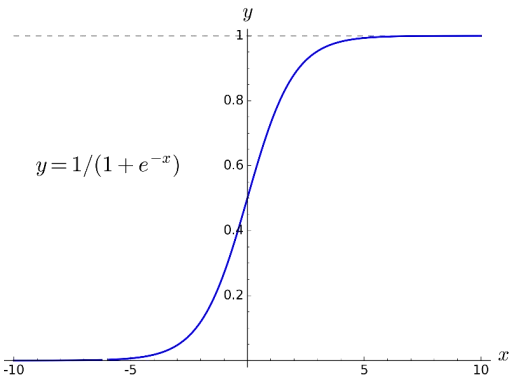
其中z为参数θ和样本X的线性组合，即

$$z = \theta X^T$$

$$\theta = [\omega_0 \quad \omega_1 \ldots \omega_m]$$

$$X = [x_0^T \quad x_1^T \ldots x_{num}^T]$$

该函数的图像如下图所示：



当 z 为 0 时，函数值为 0.5；当 z 趋近正无穷时，函数值逼近 1；当z趋近负无穷时，函数值逼近 0。

逻辑回归模型的数学形式确定后，剩下就是如何去求解模型中的参数。在统计学中，常常使用极大似然估计法来求解，即找到一组参数，使得在这组参数下，我们的数据的似然度（概率）最大。

我们假设：

$$P(Y=1|x)=p(x)$$
$$P(Y=0|x)=1-p(x)$$

则最大似然函数为：

$$L(w)=\prod[p(x_i)]^{y_i}[1-p(x_i)]^{1-y_i}$$

对等式两边同取对数得：

$$\begin{aligned}L(w) &= \sum[y_i \ln p(x_i)+(1-y_i)\ln(1-p(x_i))]\\ &= \sum[y_i(\theta\cdot x_i)-\ln(1+e^{\theta\cdot x_i})]\end{aligned}$$

在加上惩罚项后，我们取损失函数为：

$$J(\theta)=\frac{1}{n}\sum_{i=1}^{n}[y_i(\theta\cdot x_i)-\ln(1+e^{\theta\cdot x_i})]+\frac{\lambda}{2}||\theta||_2$$

总的来说，逻辑回归使用线性组合和 Sigmoid 函数来建立一个概率模型，可以用于二元分类问题。它是一种简单而有效的分类方法，并在许多应用中广泛使用，包括医学诊断、金融风险评估、自然语言处理等。

（2） 梯度下降法原理

那么如何用合适的方法来得到逻辑回归的参数呢？最简单的方法就是梯度下降法。接下来将简要介绍一下梯度下降法的原理。

首先定义损失函数。损失函数就是一个自变量为算法的参数，函数值为误差值的函数。梯度下降就是找让误差值最小时候这个算法对应的参数。损失函数用公式表示为：

$$J(\theta)=\frac{1}{n}\sum_{i=1}^{n}[y_i(\theta\cdot x_i)-\ln(1+e^{\theta\cdot x_i})]+\frac{\lambda}{2}||\theta||_2$$

其中y为样本的真实标签（值），这里的损失函数为均方损失函数，λ为惩罚项，$||\theta||_2$为θ的 L2 范数。

接着初始化参数θ，一般采用随机化的方法初始化参数，并设置学习率α。

然后迭代更新参数：

$$\theta^t=\theta^{t-1}-\alpha\frac{\partial J(\theta^{t-1})}{\partial\theta^{t-1}}$$

其中t表示当前迭代的次数，$\frac{\partial J(\theta^{t-1})}{\partial\theta^{t-1}}$表示损失函数关于$\theta$的梯度。

接着检查终止条件：

a) 可以设置迭代次数的最大值。

b) 或者可以检查参数更新的幅度是否足够小，也就是设置一个阈值。

如果终止条件不满足，则进入下一个循环迭代参数。

这个过程将不断地更新参数，使损失函数或目标函数逐渐收敛到最小值或局

部最小值。学习率α的选择和终止条件的设置都是梯度下降法的关键，它们会影响算法的性能和收敛速度。

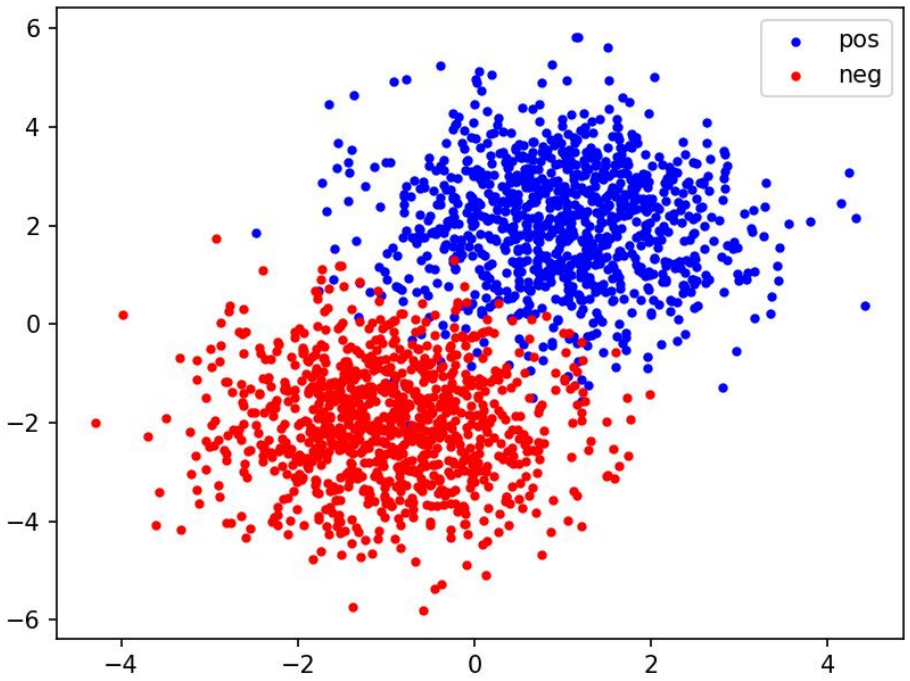
（3） 实验过程

首先用以下代码来生成数据：

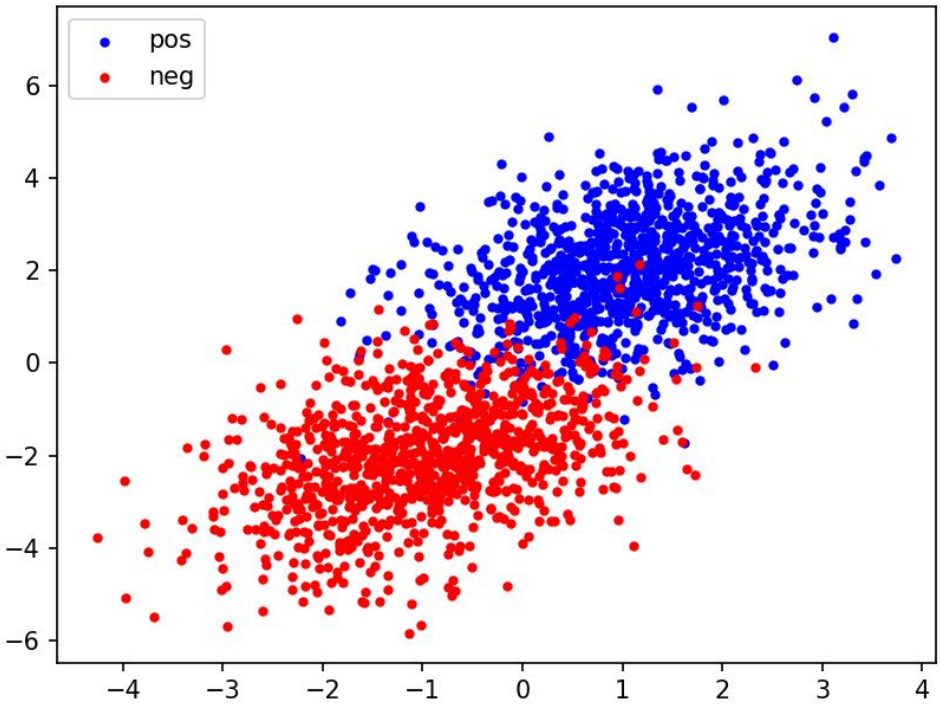
```
def generate_data(num=1000, m=2, plot=True):
    pos_mean = [1, 2]  # 正例的两维度均值
    neg_mean = [-1, -2]  # 反例的两维度均值
    X = np.zeros((2 * num, 2))
    Y = np.zeros((2 * num, 1))
    # 生成符合贝叶斯假设的数据
    cov = np.mat([[1, 0], [0, 1.5]])
    # 生成不符合贝叶斯假设的数据
    # cov = np.mat([[1, 0.5], [0.5, 1.5]])
    X[:num, :] = np.random.multivariate_normal(pos_mean, cov, num)
    X[num:, :] = np.random.multivariate_normal(neg_mean, cov, num)
    Y[:num] = 1
    Y[num:] = 0
    plt.scatter(X[:num, 0], X[:num, 1], c='b', marker='.', label='pos')
    plt.scatter(X[num:, 0], X[num:, 1], c='r', marker='.', label='neg')
    b = np.ones((2 * num, 1))
    X = np.hstack((X, b))
    # print(X.T.shape, Y.T.shape)
    return X.T, Y.T
```

用此函数可以生成符合贝叶斯假设的数据和不符合贝叶斯假设的数据，两类数据的直观表示如下：

首先是符合贝叶斯假设的数据：



接着是不符合贝叶斯假设的数据：



其中 pos 表示正例，neg 表示负例。
接着生成损失函数，用如下函数来完成：

```
# 损失函数，带正则项(极大似然），并对loss做归一化处理
def loss(X, Y, theta, lamuda=0):
    """
    :param theta: shape(1, m+1)
    :param X: shape(m+1, num)
    :param Y: shape(1, num)
    :param lamuda: penalty term
    :return: loss
    """
    num = X.shape[1]
    theta_x = np.dot(theta, X)  # shape(1, num)
    part1 = np.dot(Y, theta_x.T)
    temp = np.log(1 + np.exp(theta_x))
    part2 = np.sum(temp)
    Loss = part1 - part2 - lamuda * np.dot(theta, theta.T) / 2
    return -Loss[0][0] / num
```

然后进行梯度下降拟合逻辑回归的参数，代码如下：

```python
def gradient_descent(x_train, y_train, alpha=0.01, m=2, iter=1000, lamuda=1, threshold=1e-5):
    """
    :param threshold
    :param m: degree
    :param x_train: shape(m+1, num)
    :param y_train: shape(1, num)
    :param alpha: learning rate
    :param iter: iteration
    :param lamuda: penalty term
    :return: theta
    """
    # theta: shape(1, m+1)
    theta = np.random.rand(m + 1).reshape(1, m + 1)
    cost = 1e10
    for i in range(iter):
        gradient = gradient_theta(x_train, y_train, theta, lamuda)
        theta = theta - alpha * gradient
        if abs(cost - loss(x_train, y_train, theta, lamuda)) < threshold:
            cost = loss(x_train, y_train, theta, lamuda)
            print(f'iter: {i},\tcost: {cost}')
            break
        cost = loss(x_train, y_train, theta, lamuda)
        if i % 10 == 0:
            print(f'iter: {i},\tcost: {cost}')
    print(f'拟合的参数theta分别为：', end='')
    for t in theta[0]:
        t = round(t, 5)
```

最后用以下函数来进行测试：

```python
def test(theta, num=1000, label="test"):
    de = theta.shape[1]
    theta = theta[0]
    b = theta[2]
    x = np.linspace(-4, 4, num).reshape(1, num)
    y = -theta[0] / theta[1] * x - b / theta[1]
    plt.plot(x[0], y[0], label=label)
```

此函数用来测试拟合曲线的直观图像显示。

```python
# 测试准确率，并返回准确率和测试标签
def test_acc(x_test, y_test, theta):
    test_label = sigmoid(theta, x_test)[0]
    num = test_label.shape[0]
    # print(test_label)
    label = np.zeros(num)
    cnt = 0
    for i in range(num):
        label[i] = 1 if test_label[i] >= 0.5 else 0
        if label[i] == y_test[0][i]:
            cnt += 1
    # print(label)
    return cnt / num, label
```

此函数用来测试准确率，并返回准确率和测试标签。

最后总的测试代码如下：

```python
def test_main():
    m = 2
    num = 1000
    alpha = 0.01
    iter = 1000
    lamuda = 1
    threshold = 1e-5
    dit = {'m': 2, 'num': 1000, 'alpha': 0.01, 'iter': 1000, 'lamuda': 1, 'threshold': 1e-5}
    x_train, y_train = generate_data(num, m, plot=False)
    theta = gradient_descent(x_train, y_train, alpha=alpha, iter=iter, lamuda=lamuda, threshold=threshold)
    test(theta, num=num)
    x_test, y_test = generate_data(int(num/10), m)
    acc, _ = test_acc(x_test, y_test, theta)
    print(f'准确率为：{round(acc, 3)}')
    plt.legend()
    plt.show()
```

（4） 实验测试

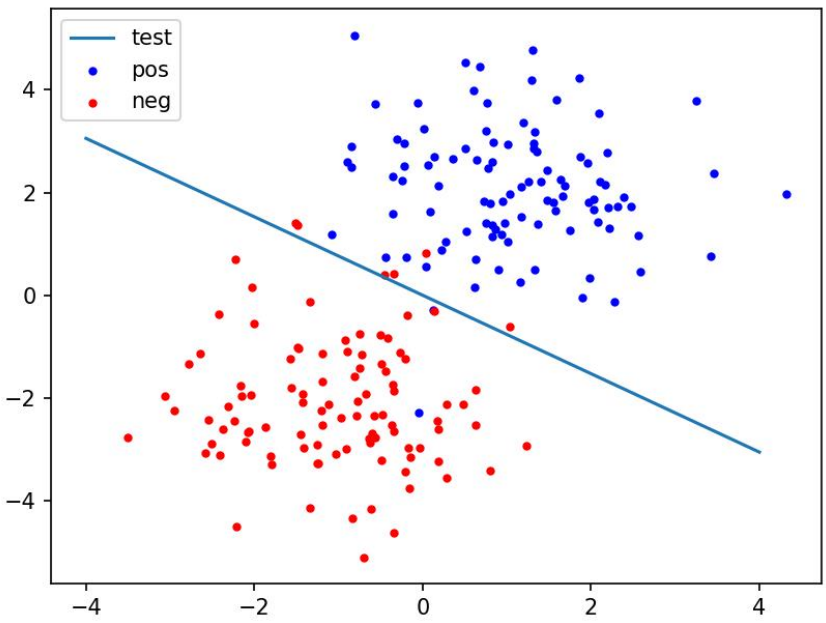
首先测试两类数据集，也就是不符合贝叶斯假设的数据集和符合贝叶斯假设的数据集。

用来测试的各超参数如下：

```
m = 2
num = 1000
alpha = 0.01
iter = 1000
lamuda = 1
threshold = 1e-5
```

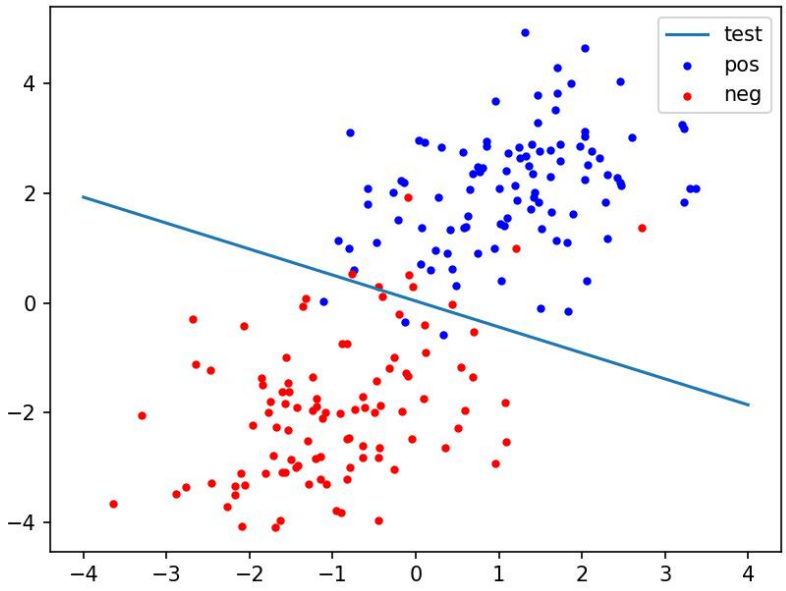
测试结果如下：

符合贝叶斯假设的数据集：



```
iter: 0,    cost: 0.13686638614771396
iter: 10,   cost: 0.09878690276232864
iter: 20,   cost: 0.08565614440780644
iter: 30,   cost: 0.08455627187152491
iter: 32,   cost: 0.0845346401796549
拟合的参数theta分别为：1.85814 2.43524 -0.00661
准确率为：0.96
```

不符合贝叶斯假设的数据集：




```
iter: 0,    cost: 0.11887275164937665
iter: 10,   cost: 0.09687980336528572
iter: 20,   cost: 0.09445885571098457
iter: 25,   cost: 0.09438256714954682
拟合的参数theta分别为：1.30499 2.76175 -0.09735
准确率为：0.945
```

从实验结果可以看出，不符合贝叶斯假设的数据集拟合的准确率更低，相比较符合贝叶斯假设的数据集而言拟合的难度更大，也就是分类的难度更大。

接下来我们来测试惩罚项对拟合结果的影响。

本次测试采用的超参数如下：

```
m = 2
num = 1000
alpha = 0.01
iter = 1000
threshold = 1e-7
```

无惩罚项的情况下结果如下：

```
iter: 0,    cost: 0.22576283389704258
iter: 10,   cost: 0.16330963947931376
iter: 20,   cost: 0.13274592317042333
iter: 30,   cost: 0.11015940527172825
iter: 40,   cost: 0.09196818510022695
iter: 50,   cost: 0.08061509579231096
iter: 60,   cost: 0.07705324607336024
iter: 70,   cost: 0.07669609801618571
iter: 80,   cost: 0.07668204063146276
iter: 82,   cost: 0.07668183001656416
拟合的参数theta分别为：1.99503 2.77754 0.05916
准确率为：0.969
```

有惩罚项（$\lambda = 1$）的情况如下：

```
iter: 0,    cost: 0.12956104224353251
iter: 10,   cost: 0.08676907805503539
iter: 20,   cost: 0.07619205489692643
iter: 30,   cost: 0.07463393365112761
iter: 40,   cost: 0.07451510097345414
iter: 50,   cost: 0.07450356832908388
iter: 56,   cost: 0.07450252576615837
拟合的参数theta分别为：1.83666 2.60241 0.09223
准确率为：0.978
```

从上图可以看出，在有惩罚项的情况下，在测试集上的准确率更高，因为有惩罚项的情况防止了过拟合的情况的发生，对参数的拟合有着更好的效果。

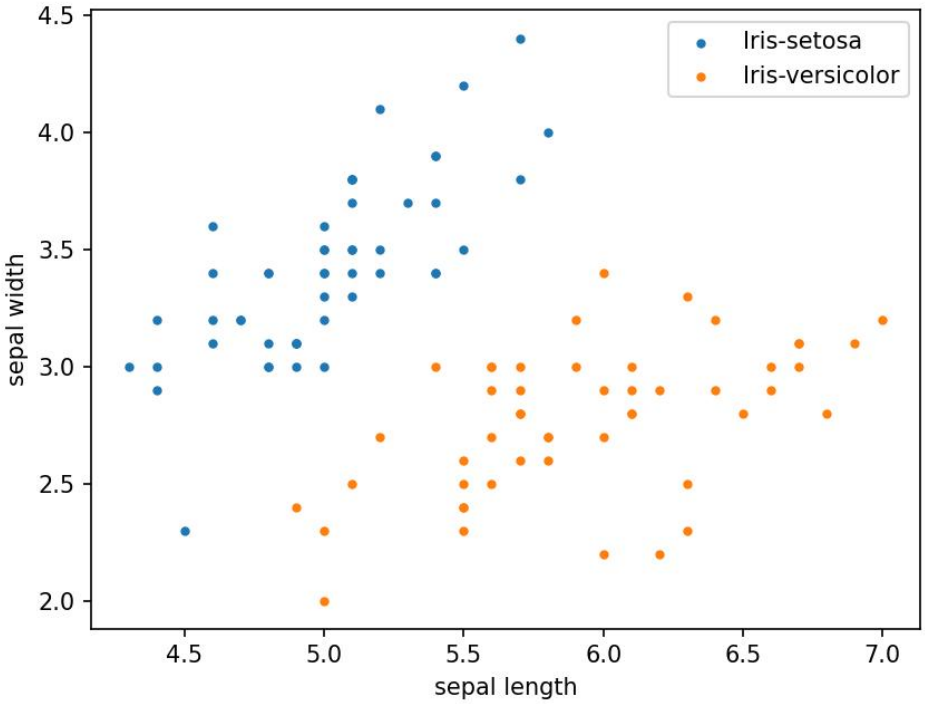
下面将选取 uci 网站上的数据进行测试。

参数选择如下：

```
# 参数如下
m = x_train.shape[0] - 1
num = x_train.shape[1]
alpha = 0.01
iter = 1000
lamuda = 1
threshold = 1e-3
```

测试结果如下：

```
iter: 0,    cost: 5.481323285602521
iter: 6,    cost: 0.09184877475664313
拟合的参数theta分别为：-1.08478 -1.07226 2.76152 1.51377 0.38158
准确率为：1.0
```



其中上图为分类任务的可视化展示。x 轴为 sepal length，y 轴为 sepal width，蓝色的点为 iris-setosa，橙色的点为 iris-versicolor。从以上测试结果可以看出，分类的准确率高达 100%，并且收敛的也非常快，可以说拟合的效果很好，超出了预期。

5. 实验总体结论

（1） 逻辑回归的优缺点

优点：

1) 简单和易理解： 逻辑回归是一种直观且易于理解的算法。它建立在线性模型的基础上，使用 Sigmoid 函数进行分类，因此容易解释和可视化。
2) 适用性广泛： 逻辑回归可用于各种分类问题，特别是二元分类。它可以应用于医学诊断、金融风险评估、自然语言处理、图像分类等各种领域。
3) 概率输出： 逻辑回归提供了样本属于正类别的概率估计，而不仅仅是类别标签。这对于需要可靠的概率估计的问题非常有用。
4) 抗噪声能力： 逻辑回归在处理具有噪声的数据时表现良好，因为它可以通过正则化来减小模型的过拟合风险。

缺点：

1) 线性决策边界限制： 逻辑回归建立在线性模型的基础上，因此只能学习

线性决策边界，对于复杂的非线性问题性能可能较差。

2) 特征选择困难： 当特征空间非常大时，逻辑回归可能需要特征选择来改善性能。这可能需要领域知识或特征工程的支持。
3) 可能受样本不平衡影响： 当类别不平衡时，逻辑回归可能偏向于多数类别，需要采取类别平衡技术来处理不平衡数据。
4) 不能很好地处理高维数据： 当特征数量远远大于样本数量时，逻辑回归可能表现不佳。在这种情况下，维度约减和特征选择可能是必要的。

（2） 实验过程中的结论

1) 符合贝叶斯假设的数据更好拟合，不符合贝叶斯假设的数据集拟合的准确率更低，相比较符合贝叶斯假设的数据集而言拟合的难度更大，也就是分类的难度更大。
2) 在有惩罚项的情况下，在测试集上的准确率更高，因为有惩罚项的情况防止了过拟合的情况的发生，对参数的拟合有着更好的效果。

6. 完整实验代码

logistic_regression.py

```
# --*-- coding:utf-8 --*--
"""
作者：徐柯炎
日期：2023 年 10 月 07 日
"""
import math
import numpy as np
import matplotlib.pyplot as plt


# sigmoid 函数
def sigmoid(theta, x):
    z = np.dot(theta, x)
    sig = 1 / (1 + np.exp(-z))
    return sig


# 生成数据（符合贝叶斯假设、不符合贝叶斯假设）
def generate_data(num=1000, m=2, plot=True):
    pos_mean = [1, 2]  # 正例的两维度均值
    neg_mean = [-1, -2]  # 反例的两维度均值
    X = np.zeros((2 * num, 2))
    Y = np.zeros((2 * num, 1))
    # 生成符合贝叶斯假设的数据
    cov = np.mat([[1, 0], [0, 1.5]])
    # 生成不符合贝叶斯假设的数据
    # cov = np.mat([[1, 0.5], [0.5, 1.5]])
```

```
28.     X[:num, :] = np.random.multivariate_normal(pos_mean, co
    v, num)
29.     X[num:, :] = np.random.multivariate_normal(neg_mean, co
    v, num)
30.     Y[:num] = 1
31.     Y[num:] = 0
32.     if plot:
33.         plt.scatter(X[:num, 0], X[:num, 1], c='b', marker='.
    ', label='pos')
34.         plt.scatter(X[num:, 0], X[num:, 1], c='r', marker='.
    ', label='neg')
35.     b = np.ones((2 * num, 1))
36.     X = np.hstack((X, b))
37.     # print(X.T.shape, Y.T.shape)
38.     return X.T, Y.T
39.
40.
41. # 损失函数，带正则项(极大似然），并对 loss 做归一化处理
42. def loss(X, Y, theta, lamuda=0):
43.     """
44.     :param theta: shape(1, m+1)
45.     :param X: shape(m+1, num)
46.     :param Y: shape(1, num)
47.     :param lamuda: penalty term
48.     :return: loss
49.     """
50.     num = X.shape[1]
51.     theta_x = np.dot(theta, X)  # shape(1, num)
52.     part1 = np.dot(Y, theta_x.T)
53.     temp = np.log(1 + np.exp(theta_x))
54.     part2 = np.sum(temp)
55.     Loss = part1 - part2 - lamuda * np.dot(theta, theta.T)
    / 2
56.     return -Loss[0][0] / num
57.
58.
59. # 求解梯度
60. def gradient_theta(x_train, y_train, theta, lamuda):
61.     gradient = np.dot(sigmoid(theta, x_train) - y_train, x_
    train.T) + lamuda / x_train.shape[1] * np.sum(theta)
62.     return gradient
63.
64.
65. # 梯度下降
```

```
66. def gradient_descent(x_train, y_train, alpha=0.01, m=2, ite
      r=1000, lamuda=1, threshold=1e-5):
67.     """
68.     :param threshold
69.     :param m: degree
70.     :param x_train: shape(m+1, num)
71.     :param y_train: shape(1, num)
72.     :param alpha: learning rate
73.     :param iter: iteration
74.     :param lamuda: penalty term
75.     :return: theta
76.     """
77.     # 初始化 theta
78.     # theta: shape(1, m+1)
79.     theta = np.random.rand(m + 1).reshape(1, m + 1)
80.     cost = 1e10
81.     for i in range(iter):
82.         # 当前梯度
83.          gradient = gradient_theta(x_train, y_train, theta,
      lamuda)
84.         # 更新参数
85.         theta = theta - alpha * gradient
86.         # 终止迭代的条件
87.         if abs(cost - loss(x_train, y_train, theta, lamuda))
       < threshold:
88.             cost = loss(x_train, y_train, theta, lamuda)
89.             print(f'iter: {i},\tcost: {cost}')
90.             break
91.         # 计算损失
92.         cost = loss(x_train, y_train, theta, lamuda)
93.         if i % 10 == 0:
94.             print(f'iter: {i},\tcost: {cost}')
95.     print(f'拟合的参数 theta 分别为: ', end='')
96.     for t in theta[0]:
97.         t = round(t, 5)
98.         print(t, end=' ')
99.     print()
100.      return theta
101.
102.
103. def test_main():
104.     # 参数如下
105.     m = 2
106.     num = 1000
```

```
    alpha = 0.01
    iter = 1000
    threshold = 1e-7
    lamuda = 1
    dit = {'m': 2, 'num': 1000, 'alpha': 0.01, 'iter': 1000, 'lamuda': 1, 'threshold': 1e-5}
    x_train, y_train = generate_data(num, m, plot=False)
    theta = gradient_descent(x_train, y_train, alpha=alpha, iter=iter, lamuda=lamuda, threshold=threshold)
    test(theta, num=num)
    x_test, y_test = generate_data(num, m)
    acc, _ = test_acc(x_test, y_test, theta)
    print(f'准确率为：{round(acc, 3)}')
    plt.legend()
    plt.show()


# 测试拟合效果，画图
def test(theta, num=1000, label="test"):
    de = theta.shape[1]
    theta = theta[0]
    b = theta[2]
    x = np.linspace(-4, 4, num).reshape(1, num)
    y = -theta[0] / theta[1] * x - b / theta[1]
    plt.plot(x[0], y[0], label=label)


# 测试准确率，并返回准确率和测试标签
def test_acc(x_test, y_test, theta):
    test_label = sigmoid(theta, x_test)[0]
    num = test_label.shape[0]
    # print(test_label)
    label = np.zeros(num)
    cnt = 0
    for i in range(num):
        label[i] = 1 if test_label[i] >= 0.5 else 0
        if label[i] == y_test[0][i]:
            cnt += 1
    # print(label)
    return cnt / num, label


if __name__ == '__main__':
    test_main()
```

iris_test.py

```
# --*-- coding:utf-8 --*--
"""
作者：徐柯炎
日期：2023 年 10 月 07 日
"""
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from logistic_regression import *
from ucimlrepo import fetch_ucirepo

# 从 uci 获取 iris 数据集
iris = fetch_ucirepo(id=53)

# 数据（pd.dataframe 格式）
X = iris.data.features
y = iris.data.targets
# print(X.info())
# print(y.iloc[0])
# print(y.iloc[55])
# 这里进行前两种花的二分类
y = y[0:100]
num = y.shape[0]
# print(num)
# print(y)
# print(X[y['class'] == 'Iris-setosa'])
# print(X[50:100])
label = np.zeros((1, num))
# 数据标注
for i in range(100):
    label[0][i] = 0 if y.loc[i, 'class'] == 'Iris-setosa' else 1
# print(label)
X = np.array(X).T


# 分割训练集和测试集
def train_test_split(X, y):
    X = np.vstack((X, np.ones((1, X.shape[1]))))
    x_train = np.hstack((X[:, 0:40], X[:, 50:90]))
    y_train = np.hstack((y[:, 0:40], y[:, 50:90]))
    x_test = np.hstack((X[:, 40:50], X[:, 90:100]))
    y_test = np.hstack((y[:, 40:50], y[:, 90:100]))
```

```
43.         # print(x_test, y_test)
44.         return x_train, x_test, y_train, y_test
45.
46.
47.     # 画出直观的分类图，x 轴为 sepal length，y 轴 sepal width
48.     def plt_iris(X, y):
49.         x1 = []
50.         x2 = []
51.         for i in range(X.shape[1]):
52.             if y[i] == 0:
53.                 x1.append(X[:, i])
54.             else:
55.                 x2.append(X[:, i])
56.         x1 = np.array(x1)
57.         x2 = np.array(x2)
58.         fig = plt.figure()
59.         ax = fig.add_subplot(111)
60.         ax.set_xlabel('sepal length')
61.         ax.set_ylabel('sepal width')
62.         ax.scatter(x1[:, 0], x1[:, 1], marker='.', label='Iri
    s-setosa')
63.         ax.scatter(x2[:, 0], x2[:, 1], marker='.', label='Iri
    s-versicolor')
64.
65.
66.     # 测试准确率，并返回准确率和测试标签
67.     def test_acc(x_test, y_test, theta):
68.         test_label = sigmoid(theta, x_test)[0]
69.         num = test_label.shape[0]
70.         # print(test_label)
71.         label = np.zeros(num)
72.         cnt = 0
73.         for i in range(num):
74.             label[i] = 1 if test_label[i] >= 0.5 else 0
75.             if label[i] == y_test[0][i]:
76.                 cnt += 1
77.         # print(label)
78.         return cnt / num, label
79.
80.
81.     if __name__ == '__main__':
82.         x_train, x_test, y_train, y_test = train_test_split(X,
     label)
83.
```

```
        # 参数如下
        m = x_train.shape[0] - 1
        num = x_train.shape[1]
        alpha = 0.01
        iter = 1000
        lamuda = 1
        threshold = 1e-3

        # 梯度下降
        theta = gradient_descent(x_train, y_train, alpha=alpha, iter=iter, lamuda=lamuda, threshold=threshold, m=m)

        # 在测试集上测试
        # acc, label = test_acc(x_test, y_test, theta)
        X = np.hstack((x_train, x_test))
        y = np.hstack((y_train, y_test))
        acc, label = test_acc(X, y, theta)
        print(f'准确率为：{round(acc, 3)}')

        # 画图
        plt_iris(X[0:2, :], label)
        plt.legend()
        plt.show()
```

7. 参考文献

逻辑回归（Logistic Regression）（一）