

# 哈尔滨工业大学计算机科学与技术学院

## 机器学习实验报告

姓名	徐亚楠
学号	1190201224
实验题目	逻辑回归
报告时间	2021. 10. 14

### 一、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。

### 二、实验要求及实验环境

#### 2.1 实验要求：

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。

#### 验证：

1. 可以手工生成两个分别类别数据（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，找一实际数据加以测试。

#### 2.2 实验环境：

Pycharm 2021 windows10 python3.8

### 三、设计思想（本程序中的用到的主要算法及数据结构）

$$\begin{aligned}
P(Y = 1|X) &= \frac{P(Y = 1)P(X|Y = 1)}{P(X)} \\
&= \frac{P(Y = 1)P(X|Y = 1)}{P(Y = 1)P(X|Y = 1) + P(Y = 0)P(X|Y = 0)} \\
&= \frac{1}{1 + \frac{P(Y = 0)P(X|Y = 0)}{P(Y = 1)P(X|Y = 1)}} \\
&= \frac{1}{1 + \exp(\ln \frac{P(Y = 0)P(X|Y = 0)}{P(Y = 1)P(X|Y = 1)})}
\end{aligned}$$

$$\text{令 } \pi = P(Y = 1), \quad 1 - \pi = P(Y = 0)$$

$$P(Y = 1|X) = \frac{1}{1 + \exp(\ln \frac{1 - \pi}{\pi} + \sum_i \ln \frac{P(X_i|Y = 0)}{P(X_i|Y = 1)})}$$

假设类条件分布服从正态分布且方差不依赖于  $k$ 。  $P(x|y_k) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(x - \mu_k)^2}{2\sigma_i^2}}$

$$\begin{aligned}
P(Y = 1|X) &= \frac{1}{1 + \exp(\ln \frac{1 - \pi}{\pi} + \sum_i \ln \frac{\frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}}}{\frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2}}})} \\
&= \frac{1}{1 + \exp(\ln \frac{1 - \pi}{\pi} + \sum_i \ln \frac{e^{-\frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}}}{e^{-\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2}}})} \\
&= \frac{1}{1 + \exp(\ln \frac{1 - \pi}{\pi} + \sum_i (\frac{-(x_i - \mu_{i0})^2}{2\sigma_i^2} - \frac{-(x_i - \mu_{i1})^2}{2\sigma_i^2}))} \\
&= \frac{1}{1 + \exp(\ln \frac{1 - \pi}{\pi} + \sum_i (\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2} - \frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}))} \\
&= \frac{1}{1 + \exp(\ln \frac{1 - \pi}{\pi} + \sum_i (\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} x_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}))}
\end{aligned}$$

$$\text{令 } w_0 = \ln \frac{1-\pi}{\pi} + \sum_i \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}, \quad w_i = \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2}$$

则有

$$P(Y = 1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

使用极大条件似然估计来计算损失函数  $l(w)$ ，此时有

$$P(Y = 0|X, W) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

所以损失函数

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1|X^l, W) + (1 - Y^l) \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1|X^l, W)}{P(Y^l = 0|X^l, W)} - \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_{i=1}^n w_i X_i) - \ln (1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)) \end{aligned}$$

我们要求  $\arg \max_w$  也就是求  $\arg \min_w -l(w)$ ，用梯度下降法求解

$$\frac{\partial l(W)}{\partial w_i} = \sum_l X_i^l (Y^l - \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i^l)})$$

$$w_i = w_i - \eta \sum_l X_i^l (Y^l - \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i^l)})$$

向量形式

$$W = W - \eta \sum_l X^l \left( Y^l - \frac{1}{1 + \exp(WX^l)} \right)$$

为了控制过拟合问题增加正则项

$$w_i = w_i - \eta \lambda w_i - \eta \sum_l X_i^l (Y^l - \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i^l)})$$

向量形式

$$W = W - \eta \lambda W - \eta \sum_l X^l (Y^l - \frac{1}{1 + \exp(WX^l)})$$

这种直接将式 求导得到式 进行迭代的方式，在数据特别多，即特别大的情况

下，有可能 导致上溢出发生，基于此，我们将式归一化，防止其溢出，得到下式

$$l(W) = \frac{1}{l} \sum_l Y^l (w_0 + \sum_{i=1}^n w_i X_i) - \ln (1 + \exp(w_0 + \sum_{i=1}^n w_i X_i))$$

$$\frac{\partial l(W)}{\partial w_i} = \frac{1}{l} \sum_l X_i^l (Y^l - \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i^l)})$$

$$w_i = w_i - \frac{\eta}{l} \sum_l X_i^l \left( Y^l - \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i^l)} \right)$$

$$W = W - \frac{\eta}{l} \sum_l X^l \left( Y^l - \frac{1}{1 + \exp(WX^l)} \right)$$

加入正则项后

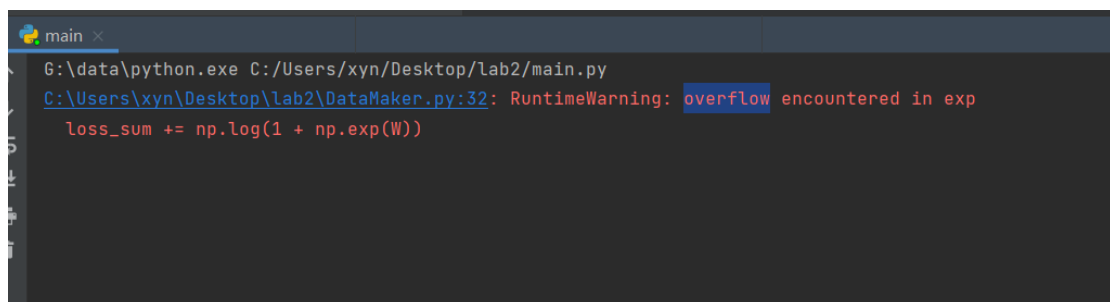
$$w_i = w_i - \eta \lambda w_i - \frac{\eta}{l} \sum_l X_i^l \left( Y^l - \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i^l)} \right)$$

$$W = W - \eta \lambda W - \frac{\eta}{l} \sum_l X^l \left( Y^l - \frac{1}{1 + \exp(WX^l)} \right)$$

## 四、实验结果与分析

### 4.1

这次实验要注意的事就是数据处理问题，一个是 sigmoid 函数 **numpy** 数组 **x** 中可能有绝对值比较大的负数，这样会导致 **np.exp(-x)** 溢出，一个是 loss 函数里的指数函数会产生数据溢出



```
main x
G:\data\python.exe C:/Users/xyn/Desktop/lab2/main.py
C:\Users\xyn\Desktop\lab2\DataMaker.py:32: RuntimeWarning: overflow encountered in exp
  loss_sum += np.log(1 + np.exp(W))
```

针对 sigmoid 和指数函数

#### 1.

```
1. if x.all()>0:
```

```
2.     return 1 / (1 + np.exp(-x))
3. else:
4.     return (np.exp(-x))/(1 + np.exp(x))
```

## 2. z-score 标准化

最常见的标准化方法就是 Z 标准化，也是 SPSS 中最为常用的标准化方法，spss 默认的标准化方法就是 z-score 标准化。

也叫标准差标准化，这种方法给予原始数据的均值（mean）和标准差（standard deviation）进行数据的标准化。

经过处理的数据符合标准正态分布，即均值为 0，标准差为 1，其转化函数为： $x^* = \frac{x - \mu}{\sigma}$  其中  $\mu$  为所有样本数据的均值， $\sigma$  为所有样本数据的标准差。z-score 标准化方法适用于属性 A 的最大值和最小值未知的情况，或有超出取值范围的离群数据的情况。

```
1. mean=np.mean(W)
2.     var=np.std(W)
3.     for i in range(size):
4.         W[i]=(W[i]-mean)/var
```

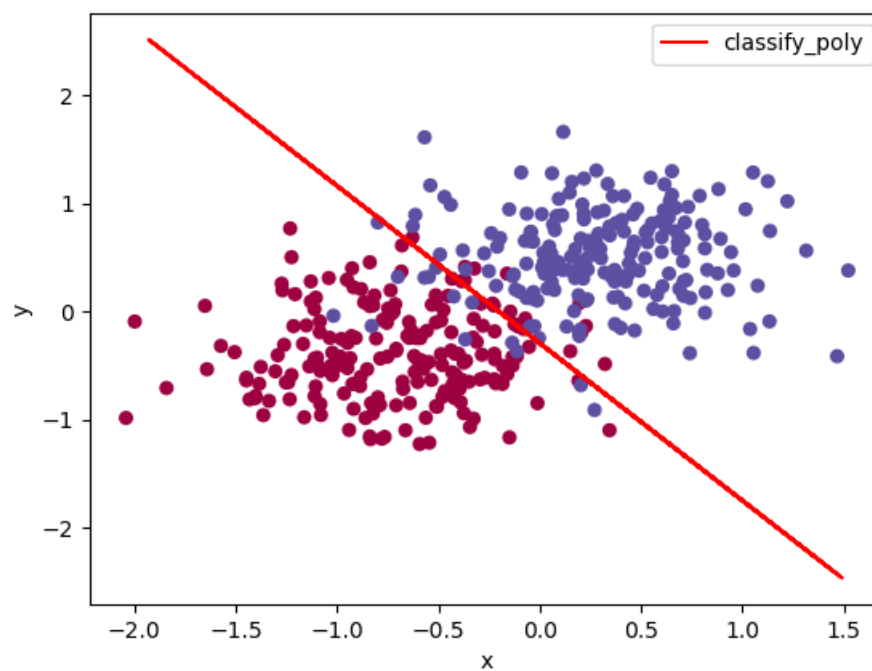
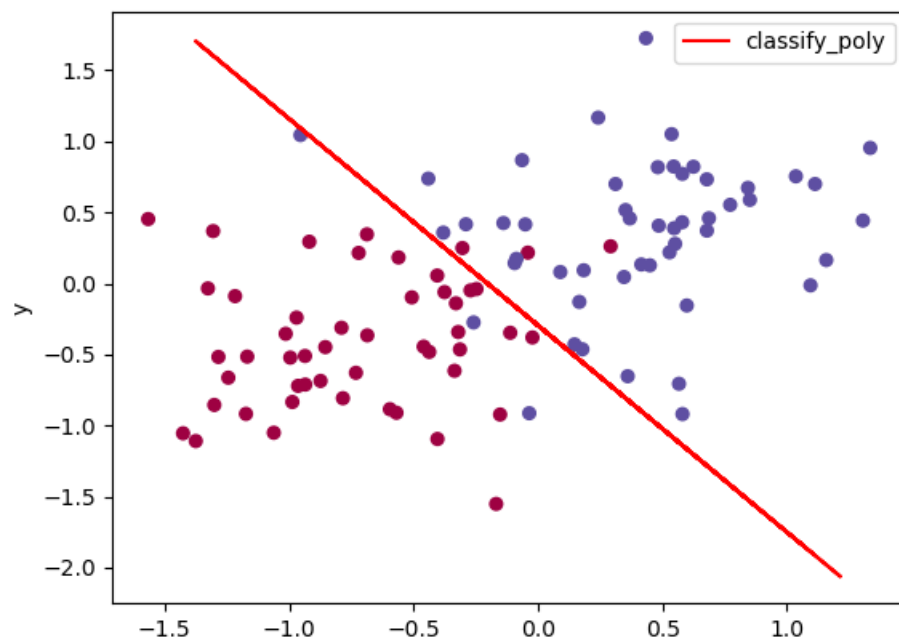
## 4.2 生成数据

利用高斯分布生成数据，注意均值所在区间选取要适当，否则他的重心会严重偏向某边出现两类数据严重交叉，线性可分很困难。我们选取反例均值为 $[-0.7, -0.3]$ ，正例均值为 $[0.3, 0.5]$ ，方差均为 0.2，不满足朴素贝叶斯假设时，两个维度数据的协方差为不为 0，协方差为 0 两属性独立这是高斯分布的特有属性可证明。我们取梯度下降学习率 0.01，迭代精度  $10^{-6}$

### 4.2.1 无惩罚，满足朴素贝叶斯假设

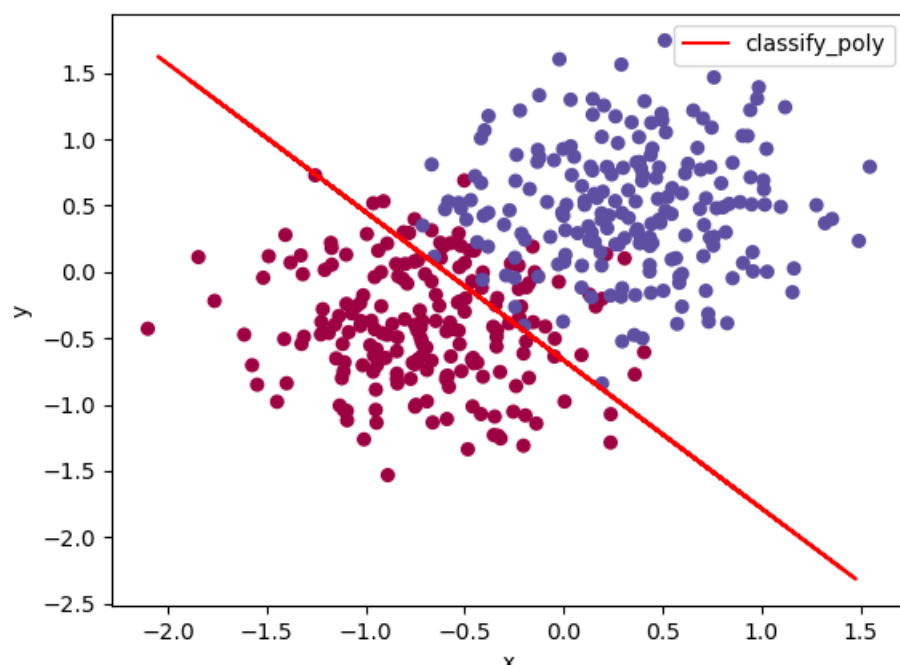
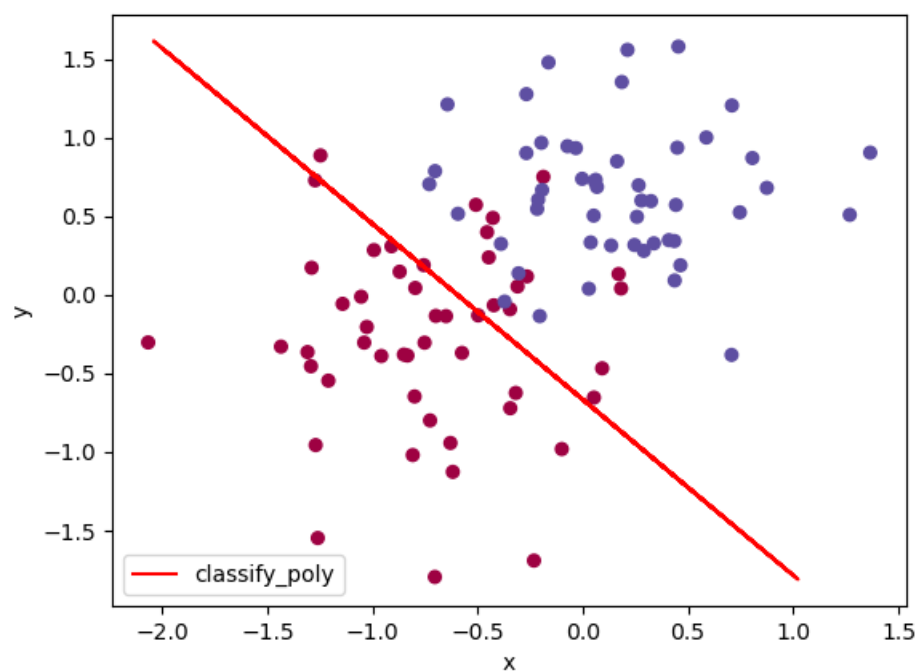
训练集正反数据量都为 50 测试集正反数据量都为 200

$y = -1.455x - 0.2954$       准确率 0.89



#### 4.2.2 无惩罚，不满足朴素贝叶斯假设

$Y = -1.118x - 0.666$  准确率 0.89



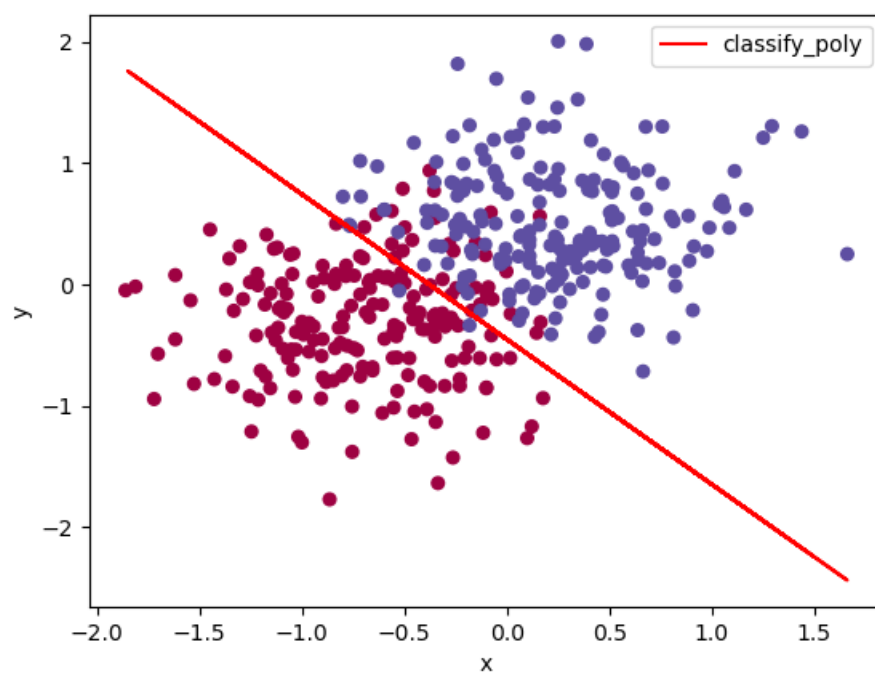
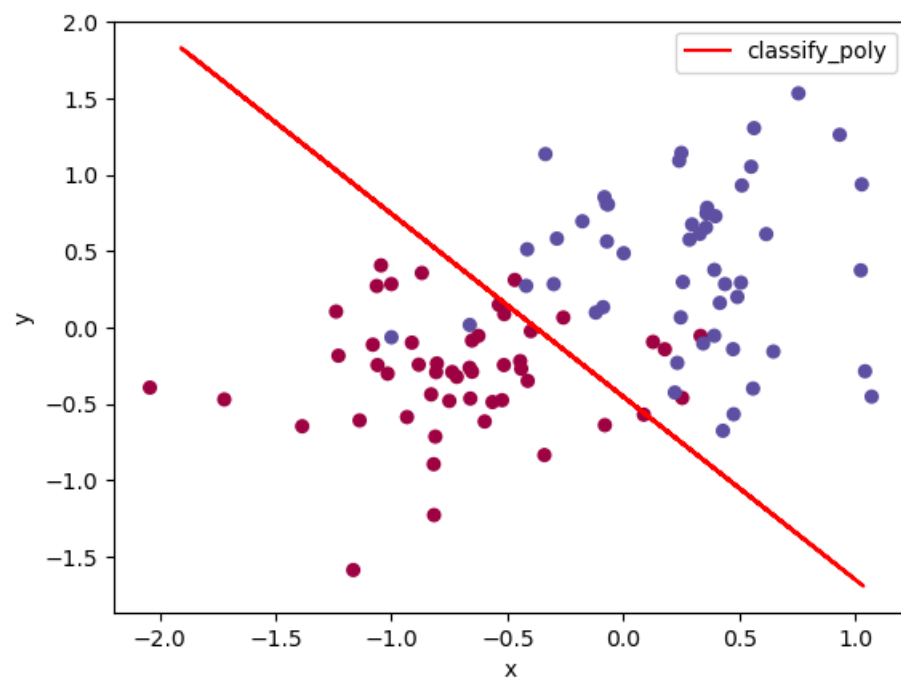
#### 4.2.3 有惩罚，满足朴素贝叶斯假设

$$w_i = w_i - \eta \lambda w_i - \frac{\eta}{l} \sum_l X_i^l \left( Y^l - \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i^l)} \right)$$

上式中，lamda 是用于调节目标函数和惩罚项之间关系的，lamda 越大，惩罚力

度越大，所得到的 $w$ 的最优解越趋近于或者说参数向量越稀疏； $\lambda$  越小，惩罚力度越小，模型越复杂，也越能体现训练集的数据特征。依据实验一经验取  $\lambda=e^{-8}$

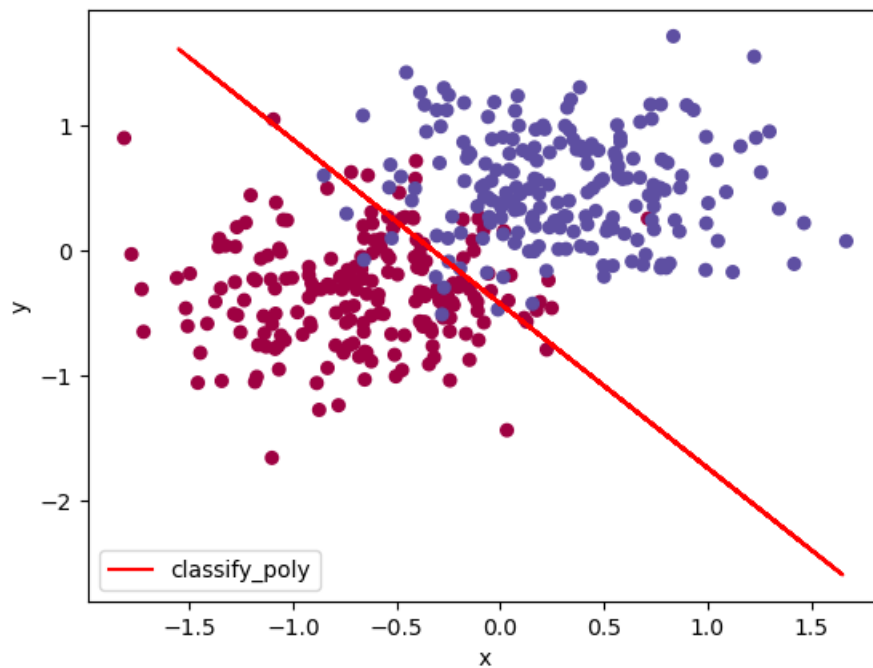
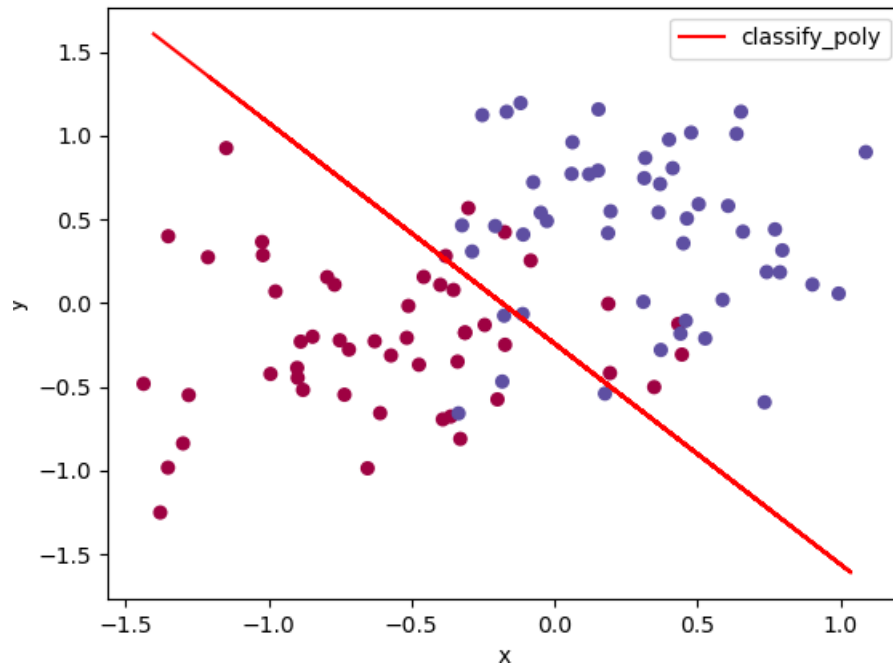
$Y=-1.197x - 0.4519$       准确率 0.92





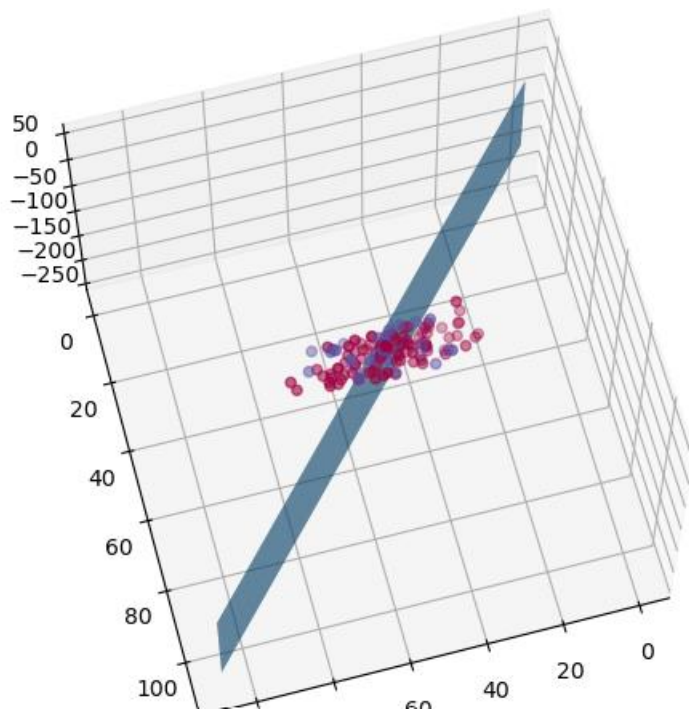
#### 4.2.4 有惩罚，不满足朴素贝叶斯假设

准确率 0.9075



#### 4.2 UCI 数据集 Haberman's Survival Data Set

此数据集一共有三个属性，两类问题，根据官网描述这不是完全线性可分数据集。所以其准确率不是很高只有 0.73 左右。



## 五、结论

1. 逻辑回归并没有对数据的分布进行建模，也就是说，逻辑回归模型并不知道数据的具体分布，而是直接根据已有的数据求解分类超平面。
2. 逻辑回归可以很好地解决线性分类问题，而且收敛速度较快，在真实的数据集上往往只需要数百次的迭代就能得到结果。
3. 正则项在数据量较大时，对结果的影响不大。在数据量较小时，可以有效解决过拟合问题。
4. 从结果中可以看出，类条件分布在满足朴素贝叶斯假设时的分类表现略好于不满足朴素贝叶斯假设时。

## 六、参考文献

[1] 周志华 著. 机器学习, 北京: 清华大学出版社, 2016 年 1 月

## 七、附录：源代码（带注释）

```

1. import numpy as np
2. import DataMaker
3.
4. if __name__ == '__main__':
5.     mean_neg,mean_pos = [-0.7, -0.3],[0.3, 0.5]
6.     var = 0.2
7.     cov_xy = 0.2
8.     size_pos,size_neg, lamda,epsilon,alpha= 50,50,np.exp(-8),1e-6,0.01
9.     # train_x,train_y=DataMaker.generate_data(mean_neg, var, size_neg, mean_
pos, var, size_pos,0.0)
10.    # coefficient, w = DataMaker.gradient_descent(train_x, train_y, lamda, a
lpha, epsilon,100000)
11.    # poly = np.poly1d(coefficient[::-1])
12.    # DataMaker.show_poly(train_x,train_y,poly)
13.    # test_x, test_y=DataMaker.generate_data(mean_neg, var, 200, mean_pos, v
ar, 200,0.0)
14.    # accuracy = DataMaker.accuracy(test_x, test_y, w)
15.    # print(poly)
16.    # print('accuracy=',accuracy)
17.    # DataMaker.show_poly(test_x, test_y, poly)
18.
19.
20.    train_x,train_y,test_x,test_y=DataMaker.uci_data('./haberman.data')
21.    coefficient, w = DataMaker.gradient_descent(train_x, train_y, lamda, alp
ha, epsilon, 100000)
22.    print(coefficient)
23.    accuracy=DataMaker.accuracy(test_x,test_y,w)
24.    print('accuracy=',accuracy)
25.    DataMaker.uci_show(train_x,train_y,coefficient)
26. import numpy as np
27. import matplotlib.pyplot as plt
28. from mpl_toolkits.mplot3d import Axes3D
29. ....
30. 生成数据
31. @:param mu_neg 反例均值
32. @:param mu_pos 正例均值
33.
34. @:param cov_xy
35. @:param var_neg 反例的方差
36. @:param
37. '''
38. def generate_data(mu_neg, var_neg, size_neg, mu_pos, var_pos, size_pos, cov_
xy=0.0):
39.     train_x = np.zeros((size_pos + size_neg, 2))

```

```

40.     train_y = np.zeros(size_pos + size_neg)
41.     train_x[:size_neg, :] = np.random.multivariate_normal(mu_neg, [[var_neg,
    cov_xy], [cov_xy, var_neg]], size=size_neg)
42.     train_x[size_neg:, :] = np.random.multivariate_normal(mu_pos, [[var_pos,
    cov_xy], [cov_xy, var_pos]], size=size_pos)
43.     train_y[size_neg:] = 1 #正例的标签
44.     return train_x, train_y
45.
46. def sigmoid(x):
47.
48.     if x.all()>0:
49.         return 1 / (1 + np.exp(-x))
50.     else:
51.         return (np.exp(-x))/(1 + np.exp(x))
52. '''
53. 利用极大条件似然得到 loss
54. 在计算 e 的指数幂会溢出 所以一定要注意归一化处理
55.
56. '''
57. def loss(train_x, train_y, w, lamda):
58.     size = train_x.shape[0]
59.     loss_sum=0
60.     W=np.zeros((size, 1))
61.     for i in range(size):
62.         W[i]=w @ train_x[i].T
63.
64.     mean=np.mean(W)
65.     var=np.std(W)
66.     for i in range(size):
67.         W[i]=(W[i]-mean)/var
68.
69.     for i in range(size):
70.         loss_sum += np.log(1 + np.e ** W[i])
71.     loss_mcle = train_y @ W - loss_sum
72.     return -loss_mcle / size
73.
74. '''
75. 梯度下降
76.
77. @:param alpha: 步长
78. @:param epsilon: 精度, 算法终止距离
79. @:param times: 最大迭代次数
80. @:param lamda:正则项
81. '''

```

```

82. def gradient_descent(train_x, train_y, lamda, alpha, epsilon, times):
83.     size = train_x.shape[0]
84.     dimension = train_x.shape[1] #train_x 的列数
85.     X = np.ones((size, dimension + 1))
86.     for i in range(dimension):
87.         X[:, i + 1] = train_x[:, i]
88.     w = np.ones((1, X.shape[1]))
89.     new_loss = loss(X, train_y, w, lamda)
90.     for i in range(times):
91.         old_loss = new_loss
92.         t = np.zeros((size, 1))
93.         for j in range(size):
94.             t[j] = w @ X[j].T
95.
96.         gradient_w = - (train_y - sigmoid(t.T)) @ X / size
97.         old_w = w
98.         w = w - alpha * lamda * w - alpha * gradient_w
99.         new_loss = loss(X, train_y, w, lamda)
100.        if old_loss < new_loss: # 不下降了, 说明步长过大
101.            w = old_w
102.            alpha /= 2
103.            continue
104.            #np.linalg.norm(gradient_w)<=epsilon
105.            if abs(new_loss - old_loss)<epsilon:
106.                break
107.        print(i)
108.        w = w.reshape(dimension + 1) # 得到的 w
109.        coefficient = -(w / w[dimension])[0:dimension] # 对 w 做归一化得到方程系
            数
110.        return coefficient, w
111.    """
112.    计算准确率
113.    @:param x: 测试数据
114.    @:param y: 测试数据标签
115.    @:param w: w
116.    """
117. def accuracy(x, y, w):
118.     size = x.shape[0]
119.     dimension = x.shape[1]
120.     correct_count = 0
121.     X = np.ones((size, dimension + 1)) # 构造 X 矩阵, 第一维都设置成 1, 方便与
        w 相乘
122.     X[:, 1:dimension + 1] = x
123.     for i in range(size):

```

```

124.         label = 0
125.         if w @ X[i].T >= 0:
126.             label = 1
127.         if label == y[i]:
128.             correct_count += 1
129.     accuracy = correct_count / size
130.     return accuracy
131.
132. def show_poly(x, y, poly):
133.
134.     plt.scatter(x[:, 0], x[:, 1], c=y, s=30, marker='o', cmap=plt.cm.Spectral)
135.     real_x = min(x[:, 0]) + (max(x[:, 0]) - min(x[:, 0])) * np.random.random(50)
136.     real_y = poly(real_x)
137.     plt.plot(real_x, real_y, 'r', label='classify_poly')
138.     plt.xlabel('x')
139.     plt.ylabel('y')
140.     plt.legend(loc=0)
141.     plt.show()
142.
143. def uci_data(path):
144.
145.     data_uci=np.loadtxt(path, dtype=np.float32)
146.     np.random.shuffle(data_uci) # 打乱原数据，以便分成训练集和测试集
147.     data_size=np.size(data_uci, axis=0) # 一共多少行
148.     train_data = data_uci[:int(0.4* data_size), :]
149.     test_data = data_uci[int(0.4 * data_size):, :] #训练集测试集 4:6
150.     dim = np.size(data_uci, axis=1) - 1 # 训练集样本维度 分离最后一列
151.     train_x = train_data[:, 0:dim]
152.     train_y = train_data[:, dim:dim + 1]
153.     train_size = np.size(train_x, axis=0)
154.     train_y = train_y.reshape(train_size) # 矩阵转化为行向量
155.     test_x = test_data[:, 0:dim]
156.     test_y = test_data[:, dim:dim + 1]
157.     test_size = np.size(test_x, axis=0)
158.     test_y = test_y.reshape(test_size) # 矩阵转化为行向量
159.     return train_x,train_y,test_x,test_y
160.
161.
162. # 三维的
163. def uci_show(train_x, train_y, poly):
164.     fig = plt.figure()
165.     ax = Axes3D(fig)

```

```
166.     ax.scatter(train_x[:, 0], train_x[:, 1], train_x[:, 2], c=train_y, cmap
    =plt.cm.Spectral)
167.     real_x = np.linspace(np.min(train_x[:,0])-
    30, np.max(train_x[:,0])+30, 255)
168.     real_y = np.linspace(np.min(train_x[:,0])-
    30, np.max(train_x[:,0])+30, 255)
169.     real_X, real_Y = np.meshgrid(real_x, real_y)
170.     real_z = poly[0] + poly[1] * real_X + poly[2] * real_Y
171.     ax.plot_surface(real_x, real_y, real_z, rstride=1, cstride=1)
172.     ax.legend(loc='best')
173.     plt.show()
```