

哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称：机器学习

课程类型：必修

实验题目：逻辑回归

学号：1190200501

姓名：林燕燕

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

实验要求

- 实现两种损失函数的参数估计：
 - 无惩罚项；
 - 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：
 - 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。
 - 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

实验环境

OS: Windows 11

Python: 3.7.9

三、设计思想

单位阶跃函数：

$$P(Y = 1|x) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}, \quad z = w^T x + b$$

由于阶跃函数不可微，使用对数几率函数替代单位阶跃函数：

$$y = \frac{1}{1 + e^{-(w^T x + b)}}$$

几率 (odds) :

$$odds = \frac{y}{1 - y} = e^{w^T x + b}$$

对数几率：

$$\ln(odds) = w^T x + b$$

重写公式：

$$P(Y = 1|x) = \frac{1}{1 + e^{-(w^T x + b)}}$$

设：

$$\begin{aligned} P(Y = 1|x) &= p(x) \\ P(Y = 0|x) &= 1 - p(x) \end{aligned}$$

似然函数：

$$\begin{aligned}
l(w) &= \prod [P(Y=1|x_i)]^{y_i} [P(Y=0|x_i)]^{1-y_i} \\
&= \prod [p(x_i)]^{y_i} [1-p(x_i)]^{1-y_i}
\end{aligned}$$

$$\text{记 } W = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix}_{3 \times 1}, \quad X = \begin{bmatrix} x_{11} & x_{12} & 1 \\ x_{21} & x_{22} & 1 \\ \vdots & \vdots & \vdots \\ x_{N1} & x_{N2} & 1 \end{bmatrix}_{N \times 3}, \quad Y = [y_1, y_2, \dots, y_N]_{1 \times N}$$

为了方便求解，取对数，得对数似然函数：

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

- 不加正则项损失函数为：

$$\begin{aligned}
J(W) &= -\frac{1}{N} \sum_{i=1}^N [y_i(Wx_i) - \ln(1 + e^{Wx_i})] \\
&= \frac{1}{N} (\ln(1 + e^{XW}) - YXW)
\end{aligned}$$

求导，得到梯度：

$$\begin{aligned}
\nabla J &= \frac{\partial J}{\partial W} = \frac{1}{N} \sum_{i=1}^N [x_i (\frac{e^{Wx_i}}{1 + e^{Wx_i}} - y_i)] \\
&= \frac{1}{N} X^T (\frac{1}{1 + e^{-XW}} - Y^T)
\end{aligned}$$

- 加入正则项的损失函数为：

$$\begin{aligned}
\tilde{J}(W) &= -\frac{1}{N} \sum_{i=1}^N [y_i(Wx_i) - \ln(1 + e^{Wx_i})] + \frac{\lambda}{2N} \|W\|^2 \\
&= \frac{1}{N} (\ln(1 + e^{XW}) - YXW) + \frac{\lambda}{2N} W^T W
\end{aligned}$$

求导，得到梯度：

$$\begin{aligned}
\nabla \tilde{J} &= \frac{\partial \tilde{J}}{\partial W} = \frac{1}{N} \sum_{i=1}^N [x_i (\frac{e^{Wx_i}}{1 + e^{Wx_i}} - y_i)] + \frac{\lambda}{N} W \\
&= \frac{1}{N} X^T (\frac{1}{1 + e^{-XW}} - Y^T) + \frac{\lambda}{N} W
\end{aligned}$$

使用梯度下降法求解：

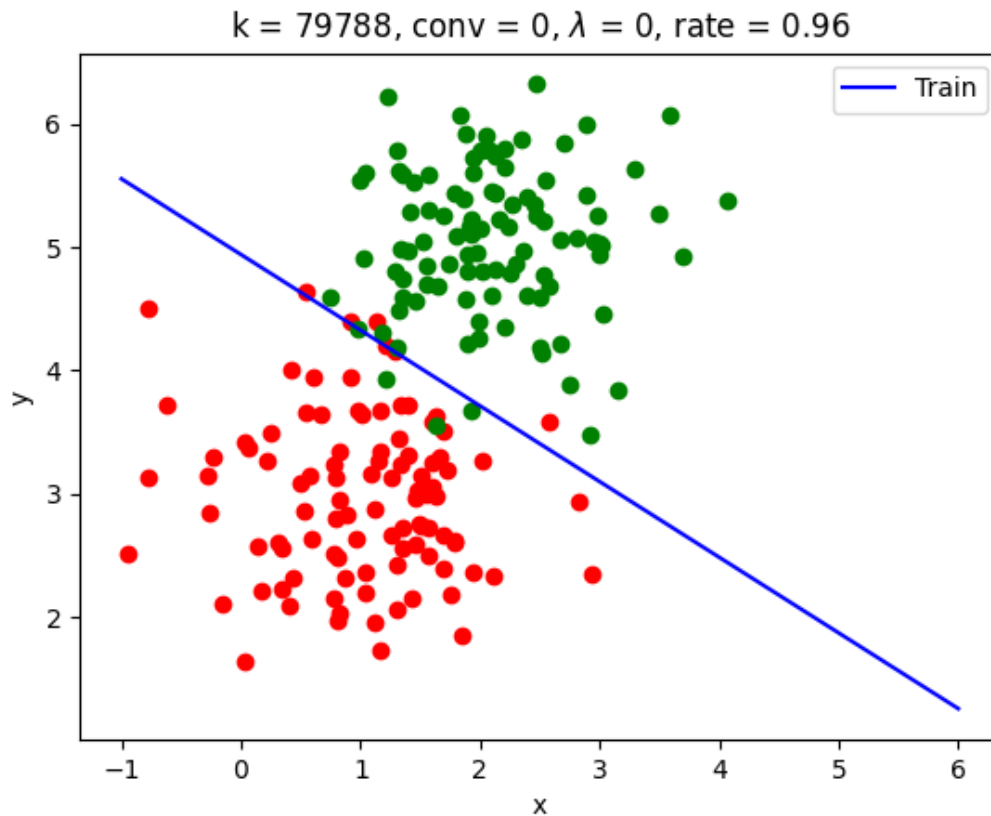
$$W_{i+1} = W_i - \alpha \nabla J$$

四、实验结果分析

1、生成数据

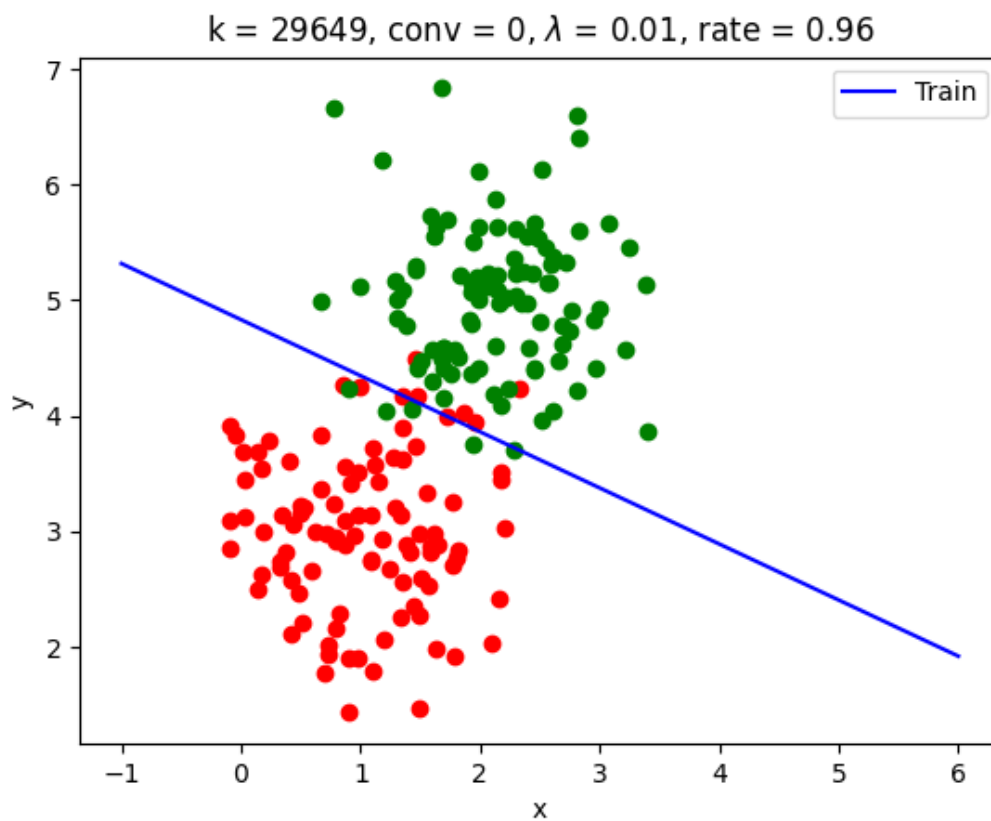
利用高斯分布生成中心点分别为 (1, 3)(2, 5) 的两组点，有正则项时 λ 为0.01，不满足贝叶斯假设时相关系数为0.2，梯度下降学习率为0.5，精度为 10^{-6} 。

2、满足贝叶斯假设，无正则项



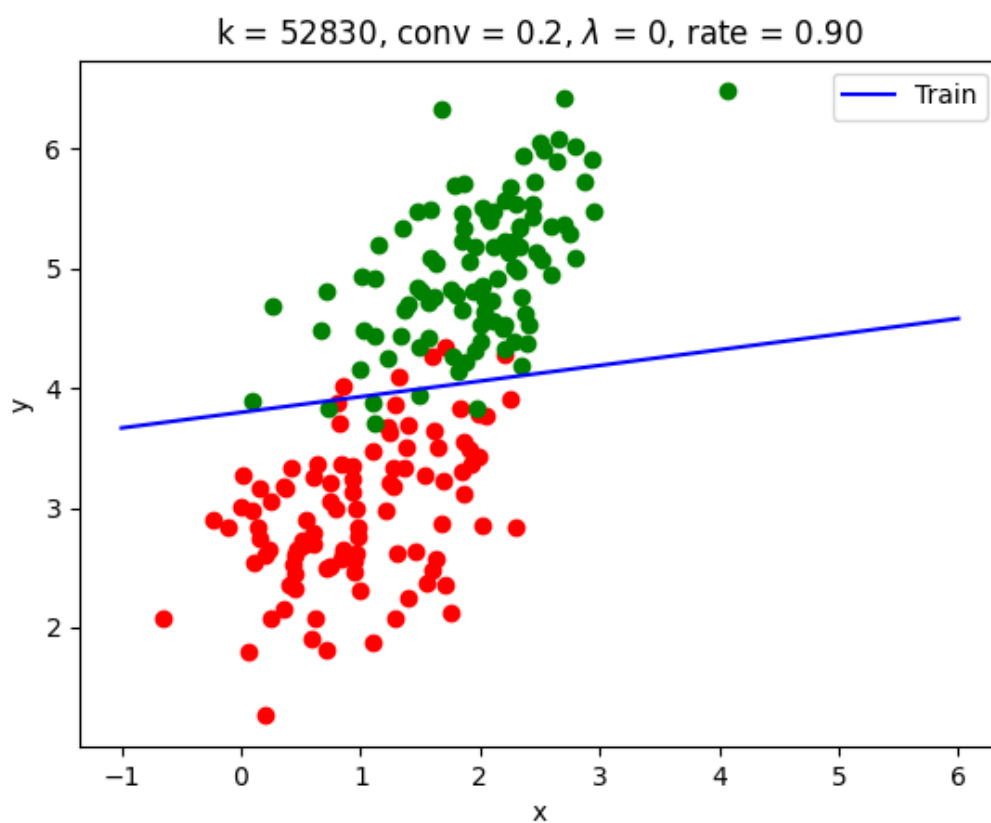
迭代次数为79788，正确率为0.96

3、满足贝叶斯假设，有正则项



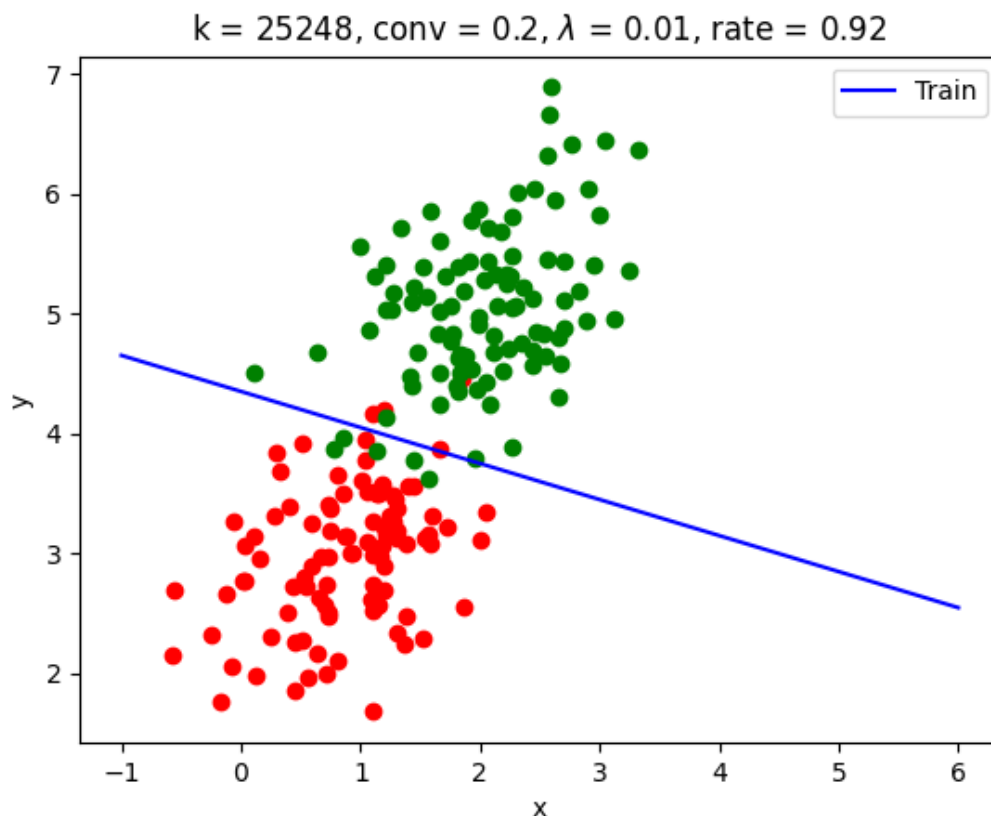
迭代次数为29649, 正确率为0.96

4、不满足贝叶斯假设，无正则项



迭代次数为52830, 正确率为0.90

5、不满足贝叶斯假设，有正则项



迭代次数为25248，正确率为0.92

逻辑回归分类器在满足朴素贝叶斯假设时分类良好，在不满足朴素贝叶斯假设时分类效果稍弱。训练集较小时，存在过拟合现象，加入正则项可以减少此现象。

6、使用UCI数据

共有576个数据，属性个数为4，取前200个作为训练集，剩余数据作为测试集

(1)无正则项

```
PS E:\Program\Machine Learning\2-Logistic regression\Code> python LogisticRegression_uci.py
k = 17334, lambda = 0, rate = 0.8537234042553191
```

迭代次数为17334，正确率为0.854

(2)有正则项

$\lambda=0.2$

```
PS E:\Program\Machine Learning\2-Logistic regression\Code> python LogisticRegression_uci.py
k = 3926, lambda = 0.2, rate = 0.8776595744680851
```

迭代次数为3926，正确率为0.878

$\lambda=0.5$

```
PS E:\Program\Machine Learning\2-Logistic regression\Code> python LogisticRegression_uci.py
k = 2052, lambda = 0.5, rate = 0.8962765957446809
```

迭代次数为2052，正确率为0.896

五、结论

1. 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，在真实的数据集上迭代次数比随机生成数据集小很多；
2. 正则项在数据量较小时，可以有效解决过拟合问题。
3. 从结果中可以看出，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

[【机器学习】逻辑回归 - 知乎](#)

七、附录:源代码(带注释)

```
1  ##### LogisticRegression.py #####
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  def sigmoid(x_i):
6      return 1 / (1 + np.exp(-x_i))
7
8  def model(X, w):
9      """
10     预测函数
11     """
12     return sigmoid(np.dot(X, w))
13
14  def cal_loss(w, X, Y, _lambda):
15     size = X.shape[0]
16     ln = np.mean(np.log(1 + np.exp(X @ w)))
17     loss = (-Y @ X @ w + ln + 0.5 * _lambda * np.dot(w.T, w)) / size
18     return loss
19
20  def cal_gradient(w, X, Y, _lambda):
21     """
22     计算梯度值
23     _lambda为0时即为无正则项
24     """
25     return ((X.T @ model(X, w) - X.T @ Y.T) + _lambda * w) / X.shape[0]
26
27  def gradient_descent(train_X, train_Y, _lambda, times, _alpha, epsilon):
28     w = np.zeros((train_X.shape[1], 1)) ##### 初始化 w
29     new_loss = abs(cal_loss(w, train_X, train_Y, _lambda))
30     k = 0
31     for i in range(times):
32         old_loss = new_loss
33         gradient_loss = cal_gradient(w, train_X, train_Y, _lambda)
34         w -= gradient_loss * _alpha # w_{i+1} = w_i - _alpha *
gradient_loss
35         new_loss = abs(cal_loss(w, train_X, train_Y, _lambda))
36         if abs(old_loss - new_loss) < epsilon:
37             k = i
38             break
39     return w, k
40
41  def data(size, locat, conv):
42     cov=[[0.4, conv], [conv, 0.4]]
43     def generater(n):
44         X = np.zeros((n * 2, 2))
```

```

45     Y = np.zeros((n * 2, 1))
46     X[:n, :] = np.random.multivariate_normal(locat[0], cov, n)
47     X[n:, :] = np.random.multivariate_normal(locat[1], cov, n)
48     Y[n:] = 1
49     return X, Y.T
50 train_data = generater(size[0])
51 test_data = generater(size[1])
52 return train_data, test_data
53
54 def cal_rate(test_X, test_Y, W):
55     y = model(test_X, W)
56     Y = np.zeros((y.shape[0], 1))
57     test_Y = test_Y.T
58     for i in range(y.shape[0]):
59         if y[i] >= 0.5:
60             Y[i] = 1
61         elif y[i] < 0.5:
62             Y[i] = 0
63     correct = 0
64     for i in range(Y.shape[0]):
65         if Y[i] == test_Y[i]:
66             correct += 1
67     rate = correct / Y.shape[0]
68     return rate
69
70 """
71 W = [3*1] , X = [N*3] , Y = [1*N]
72 """
73 if __name__ == "__main__":
74     size = (100,150)
75     locat = np.array([[1, 3],[2, 5]])
76     conv = 0 # 相关系数
77     _lambda = 0
78     times = 1000000
79     _alpha = 0.5
80     epsilon = 1e-6
81     ### 生成数据
82     train_data, test_data = data(size, locat, conv)
83     train_X = np.zeros((size[0] * 2, 3))
84     train_X[:, :2] = train_data[0]
85     train_X[:, 2:] = 1
86     train_Y = train_data[1]
87
88     test_X = np.zeros((size[1] * 2, 3))
89     test_X[:, :2] = test_data[0]
90     test_X[:, 2:] = 1
91     test_Y = test_data[1]
92     #####
93     W, k = gradient_descent(train_X, train_Y, _lambda, times, _alpha,
94                             epsilon)
95     rate = cal_rate(test_X, test_Y, W)
96
97     x = np.linspace(-1, 6, 10000)
98     y = -(W[0] * x + W[2]) / W[1]
99     A = []
100    B = []
101    for i in range(size[0] * 2):
102        if train_data[1][0][i] == 0:

```



```

102         A.append(train_data[0][i])
103         elif train_data[1][0][i] == 1:
104             B.append(train_data[0][i])
105     A = np.array(A)
106     B = np.array(B)
107     plt.title("k = {}, conv = {},  $\lambda$  = {}, rate = {:.2f}".format(k,
conv, _lambda, rate))
108     plt.xlabel("x")
109     plt.ylabel("y")
110     plt.plot(x, y, c="b", label = "Train")
111     plt.scatter(A[:,1], A[:, 1:], c="r")
112     plt.scatter(B[:,1], B[:, 1:], c="g")
113     plt.legend()
114     plt.show()
115
116 ##### LogisticRegression_uci.py #####
117
118 import numpy as np
119 import re
120
121 def sigmoid(x_i):
122     return 1 / (1 + np.exp(-x_i))
123
124 def model(X, w):
125     """
126     预测函数
127     """
128     return sigmoid(np.dot(X, w))
129
130 def cal_loss(w, X, Y, _lambda):
131     size = X.shape[0]
132     ln = np.mean(np.log(1 + np.exp(X @ w)))
133     loss = (-Y @ X @ w + ln + 0.5 * _lambda * np.dot(w.T,w))/ size
134     return loss
135
136 def cal_gradient(w, X, Y, _lambda):
137     """
138     计算梯度值
139     _lambda为0时即为无正则项
140     """
141     return ((X.T @ model(X, w) - X.T @ Y.T) + _lambda * w) / X.shape[0]
142
143 def gradient_descent(train_X, train_Y, _lambda, times, _alpha, epsilon):
144     w = np.zeros((train_X.shape[1], 1)) ##### 初始化 w
145     new_loss = abs(cal_loss(w, train_X, train_Y, _lambda))
146     k = 0
147     for i in range(times):
148         old_loss = new_loss
149         gradient_loss = cal_gradient(w, train_X, train_Y, _lambda)
150         w -= gradient_loss * _alpha #  $w_{i+1} = w_i - \alpha * \text{gradient\_loss}$ 
151         new_loss = abs(cal_loss(w, train_X, train_Y, _lambda))
152         if abs(old_loss - new_loss) < epsilon:
153             k = i
154             break
155     return w, k
156
157 def cal_rate(test_X, test_Y, w):

```

```

158     y = model(test_X, w)
159     Y = np.zeros((y.shape[0], 1))
160     test_Y = test_Y.T
161     for i in range(y.shape[0]):
162         if y[i] >= 0.5:
163             Y[i] = 1
164         elif y[i] < 0.5:
165             Y[i] = 0
166     correct = 0
167     for i in range(Y.shape[0]):
168         if Y[i] == test_Y[i]:
169             correct += 1
170     rate = correct / Y.shape[0]
171     return rate
172
173 """
174 W = [3*1] , X = [N*3] , Y = [1*N]
175 """
176 if __name__ == "__main__":
177     conv = 0 # 相关系数
178     _lambda = 0
179     times = 1000000
180     _alpha = 0.5
181     epsilon = 1e-6
182     ### 获取数据
183     bl = open("uci.data", encoding="UTF-8")
184     bl_list = bl.readlines()
185     LEN = len(bl_list)
186     dim = 4
187     X = np.zeros((LEN, dim))
188     Y = np.zeros((1, LEN))
189     label = []
190     i = 0
191     for line in bl_list:
192         l = re.split("[,\n]", line)
193         for j in range(dim):
194             X[i, j] = l[j + 1]
195         if l[0] == "L":
196             Y[0, i] = 0
197         elif l[0] == "R":
198             Y[0, i] = 1
199         i += 1
200     train_X = X[:200, :] ## 训练集
201     train_Y = Y[:, :200]
202     test_X = X[200:, :] ## 测试集
203     test_Y = Y[:, 200:]
204
205     #####
206     w, k = gradient_descent(train_X, train_Y, _lambda, times, _alpha,
207                             epsilon)
207     rate = cal_rate(test_X, test_Y, w)
208     print("k = {}, lambda = {}, rate = {}".format(k, _lambda, rate))

```

