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：
  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

## 实验环境

OS： Windows 11

Python： 3.7.9

# 三、设计思想

单位阶跃函数：

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

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：

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

**几率（odds）：**

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

**对数几率：**

$$ln(odds)=w^Tx+b$$

重写公式：

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

设：

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

似然函数：

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

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

为了方便求解，取对数，得对数似然函数：

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

- 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：

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

  求导，得到梯度：

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

使用梯度下降法求解：

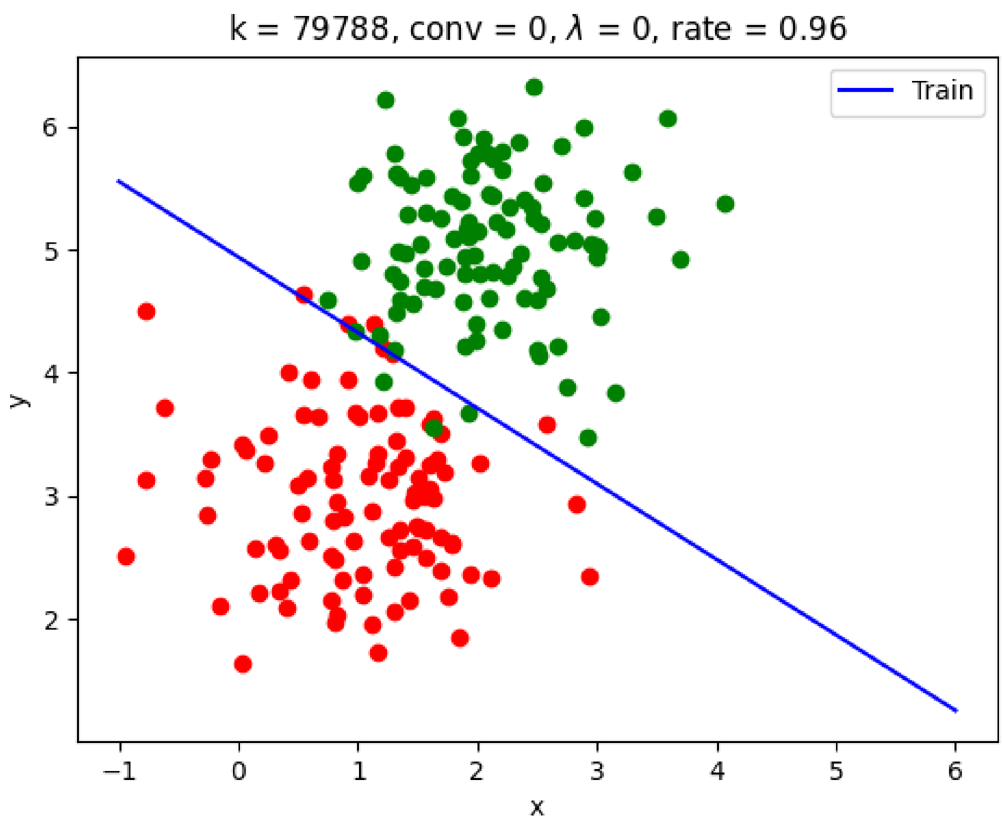
$$
W_{i+1} = W_i - \alpha \nabla J
$$

# 四、实验结果分析

## 1、生成数据

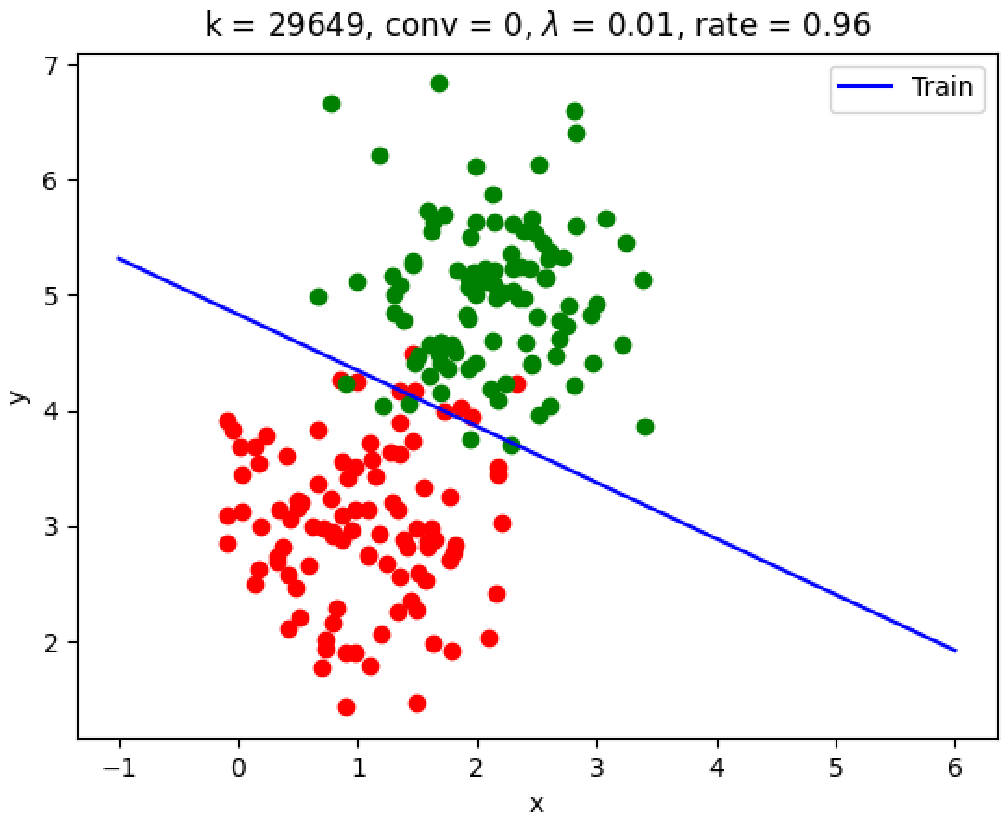
利用高斯分布生成中心点分别为 (1, 3)(2, 5) 的两组点，有正则项时 $\lambda$ 为0.01，不满足贝叶斯假设时相关系数为0.2，梯度下降学习率为0.5，精度为 $10^{-6}$ 。

## 2、满足贝叶斯假设，无正则项



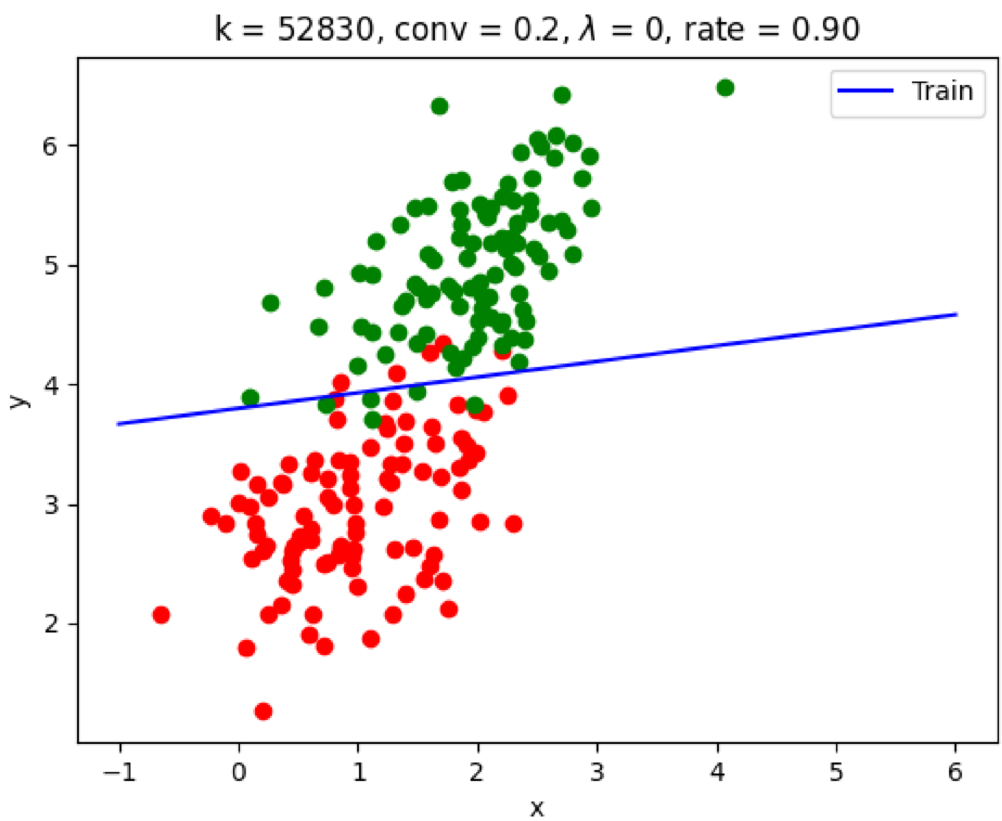

迭代次数为79788，正确率为0.96

## 3、满足贝叶斯假设，有正则项



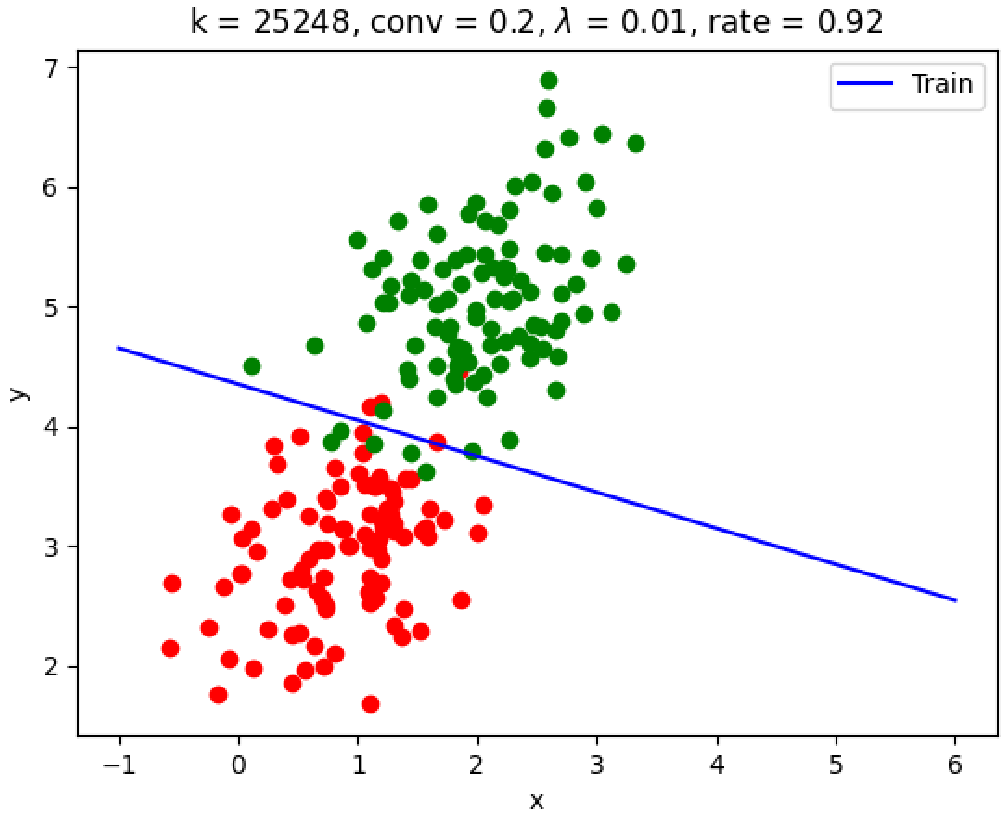

迭代次数为29649，正确率为0.96

## 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)

```python
#### LogisticRegression.py ####
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x_i):
    return 1 / (1 + np.exp(-x_i))

def model(X, W):
    """
    预测函数
    """
    return sigmoid(np.dot(X, W))

def cal_loss(W, X, Y, _lambda):
    size = X.shape[0]
    ln = np.mean(np.log(1 + np.exp(X @ W)))
    loss = (-Y @ X @ W + ln + 0.5 * _lambda * np.dot(W.T,W))/ size
    return loss

def cal_gradient(W, X, Y, _lambda):
    """
    计算梯度值
    _lambda为0时即为无正则项
    """
    return ((X.T @ model(X, W) - X.T @ Y.T) + _lambda * W) / X.shape[0]

def gradient_descent(train_X, train_Y, _lambda, times, _alpha, epsilon):
    W = np.zeros((train_X.shape[1], 1))    #### 初始化 W
    new_loss = abs(cal_loss(W, train_X, train_Y, _lambda))
    k = 0
    for i in range(times):
        old_loss = new_loss
        gradient_loss = cal_gradient(W, train_X, train_Y, _lambda)
        W -= gradient_loss * _alpha    # W_i+1 = W_i - _alpha * gradient_loss
        new_loss = abs(cal_loss(W, train_X, train_Y, _lambda))
        if abs(old_loss - new_loss) < epsilon:
            k = i
            break
    return W, k

def data(size, locat, conv):
    cov=[[0.4, conv], [conv, 0.4]]
    def generater(n):
        X = np.zeros((n * 2, 2))
```

```
        Y = np.zeros((n * 2, 1))
        X[:n, :] = np.random.multivariate_normal(locat[0], cov, n)
        X[n:, :] = np.random.multivariate_normal(locat[1], cov, n)
        Y[n:] = 1
        return X, Y.T
    train_data = generater(size[0])
    test_data = generater(size[1])
    return train_data, test_data

def cal_rate(test_X, test_Y, W):
    y = model(test_X, W)
    Y = np.zeros((y.shape[0], 1))
    test_Y = test_Y.T
    for i in range(y.shape[0]):
        if y[i] >= 0.5:
            Y[i] = 1
        elif y[i] < 0.5:
            Y[i] = 0
    correct = 0
    for i in range(Y.shape[0]):
        if Y[i] == test_Y[i]:
            correct += 1
    rate = correct / Y.shape[0]
    return rate

"""
W = [3*1] , X = [N*3] , Y = [1*N]
"""
if __name__ == "__main__":
    size = (100,150)
    locat = np.array([[1, 3],[2, 5]])
    conv = 0  # 相关系数
    _lambda = 0
    times = 1000000
    _alpha = 0.5
    epsilon = 1e-6
    ### 生成数据
    train_data, test_data = data(size, locat, conv)
    train_X = np.zeros((size[0] * 2, 3))
    train_X[:,:2] = train_data[0]
    train_X[:,2:] = 1
    train_Y = train_data[1]

    test_X = np.zeros((size[1] * 2, 3))
    test_X[:,:2] = test_data[0]
    test_X[:,2:] = 1
    test_Y = test_data[1]
    ################
    W, k = gradient_descent(train_X, train_Y, _lambda, times, _alpha, epsilon)
    rate = cal_rate(test_X, test_Y, W)

    x = np.linspace(-1, 6, 10000)
    y = -(W[0] * x + W[2]) / W[1]
    A = []
    B = []
    for i in range(size[0] * 2):
        if train_data[1][0][i] == 0:
```

```
            A.append(train_data[0][i])
        elif train_data[1][0][i] == 1:
            B.append(train_data[0][i])
    A = np.array(A)
    B = np.array(B)
    plt.title("k = {}, conv = {}, $\lambda$ = {}, rate = {:.2f}".format(k, conv, _lambda, rate))
    plt.xlabel("x")
    plt.ylabel("y")
    plt.plot(x, y, c="b",label = "Train")
    plt.scatter(A[:,:1], A[:, 1:], c="r")
    plt.scatter(B[:,:1], B[:, 1:], c="g")
    plt.legend()
    plt.show()

#### LogisticRegression_uci.py ####

import numpy as np
import re

def sigmoid(x_i):
    return 1 / (1 + np.exp(-x_i))

def model(X, W):
    """
    预测函数
    """
    return sigmoid(np.dot(X, W))

def cal_loss(W, X, Y, _lambda):
    size = X.shape[0]
    ln = np.mean(np.log(1 + np.exp(X @ W)))
    loss = (-Y @ X @ W + ln + 0.5 * _lambda * np.dot(W.T,W))/ size
    return loss

def cal_gradient(W, X, Y, _lambda):
    """
    计算梯度值
    _lambda为0时即为无正则项
    """
    return ((X.T @ model(X, W) - X.T @ Y.T) + _lambda * W) / X.shape[0]

def gradient_descent(train_X, train_Y, _lambda, times, _alpha, epsilon):
    W = np.zeros((train_X.shape[1], 1))   #### 初始化 W
    new_loss = abs(cal_loss(W, train_X, train_Y, _lambda))
    k = 0
    for i in range(times):
        old_loss = new_loss
        gradient_loss = cal_gradient(W, train_X, train_Y, _lambda)
        W -= gradient_loss * _alpha   # W_i+1 = W_i - _alpha * gradient_loss
        new_loss = abs(cal_loss(W, train_X, train_Y, _lambda))
        if abs(old_loss - new_loss) < epsilon:
            k = i
            break
    return W, k

def cal_rate(test_X, test_Y, W):
```

```python
    y = model(test_X, W)
    Y = np.zeros((y.shape[0], 1))
    test_Y = test_Y.T
    for i in range(y.shape[0]):
        if y[i] >= 0.5:
            Y[i] = 1
        elif y[i] < 0.5:
            Y[i] = 0
    correct = 0
    for i in range(Y.shape[0]):
        if Y[i] == test_Y[i]:
            correct += 1
    rate = correct / Y.shape[0]
    return rate

"""
W = [3*1] , X = [N*3] , Y = [1*N]
"""
if __name__ == "__main__":
    conv = 0  # 相关系数
    _lambda = 0
    times = 1000000
    _alpha = 0.5
    epsilon = 1e-6
    ### 获取数据
    bl = open("uci.data", encoding="UTF-8")
    bl_list = bl.readlines()
    LEN = len(bl_list)
    dim = 4
    X = np.zeros((LEN, dim))
    Y = np.zeros((1, LEN))
    label = []
    i = 0
    for line in bl_list:
        l = re.split("[,\n]",line)
        for j in range(dim):
            X[i, j] = l[j + 1]
        if l[0] == "L":
            Y[0, i] = 0
        elif l[0] == "R":
            Y[0, i] = 1
        i += 1
    train_X = X[:200, :]  ## 训练集
    train_Y = Y[:,: 200]
    test_X = X[200:, :]   ## 测试集
    test_Y = Y[:, 200:]

    ################
    W, k = gradient_descent(train_X, train_Y, _lambda, times, _alpha, epsilon)
    rate = cal_rate(test_X, test_Y, W)
    print("k = {}, lambda = {}, rate = {}".format(k, _lambda, rate))
```