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： 选修

实验题目： 逻辑回归

学号： 1170300520

姓名： 郭子阳

# 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

1. 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）；
2. 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；
3. 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

- Python 3.7.0
- JupyterLab 1.1.3

# 三、设计思想（本程序中的用到的主要算法及数据结构）

## 算法原理

逻辑回归（Logistic Regression）是一种用于解决二分类（0 or 1）问题的机器学习方法，用于估计某种事物的可能性。

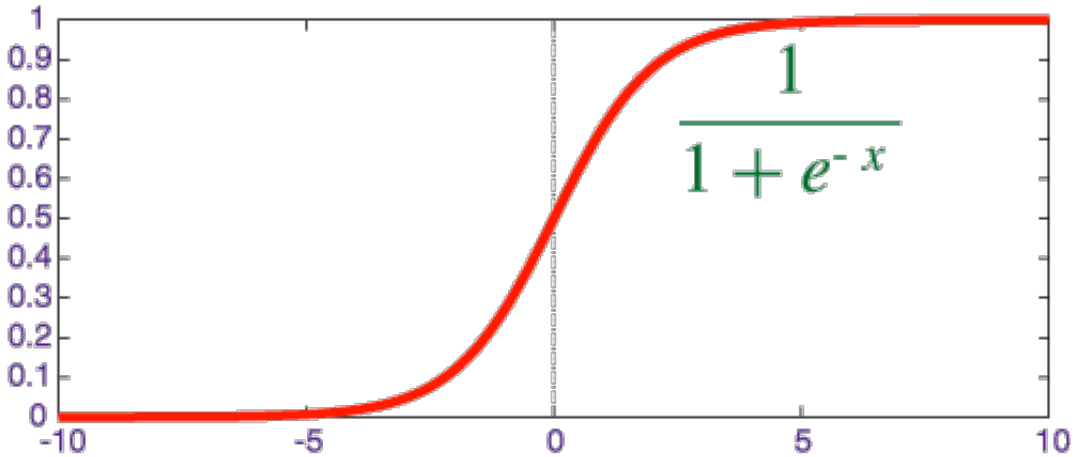
对于Logistic Regression来说，其思想也是基于线性回归（Logistic Regression属于广义线性回归模型）。其公式如下：

$$h_\theta(x) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-\theta^T x}}$$

其中，

$$y = \frac{1}{1+e^{-z}}$$

称为Sigmoid函数，逻辑回归实质是将线性回归函数的结果映射到了(0, 1)中。Sigmoid函数图像如下：



于是，$h_\theta(x)$表明了数据属于某一类别的概率，即可对样本进行分类。

## 算法实现

由于使用的Titanic数据集可能导致溢出，Sigmoid函数替换为如下的激活函数：

$$mapping(z) = 0.5(1 + tanh(0.5z))$$

Cost函数定义为：

$$J = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log(a^{(i)}) + (1-y^{(i)})\log(1-a^{(i)})$$

加上惩罚项的Cost函数为：

$$J = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log(a^{(i)}) + (1-y^{(i)})\log(1-a^{(i)}) + \frac{\lambda}{2m}w^2$$

于是，训练的单次迭代如下：

```
def propagate(w, b, X, Y, theta):
    m = X.shape[1]
    # 前向传播
    theta = 1e-5
    A = mapping((np.dot(w.T, X) + b ) * theta)
    cost = -1 / m * np.sum(Y * np.log(A) + (1 - Y) * np.log(1 - A)) +
theta/(2*m)*np.dot(w.T, w)

    # 计算梯度，反向传播
    dw = 1 / m * np.dot(X, (A - Y).T) + theta/m*w
    db = 1 / m * np.sum(A - Y)

    cost = np.squeeze(cost)

    grad = {
        "dw": dw,
        "db": db
    }
    return grad, cost
```

训练如下：

```
def optimize(w, b, X, Y, num_iterations, learning_rate, theta):
    costs = []
    for i in range(num_iterations):
        grad, cost = propagate(w, b, X, Y, theta)

        dw = grad['dw']
        db = grad['db']

```

```
        # 更新w和b
        w = w - learning_rate * dw
        b = b - learning_rate * db

        if (i+1)%2000 == 0:
            costs.append(cost)

    params = {
        "w": w,
        "b": b
    }
    return params, costs
```

自己生成的两个特征的数据集如下：

```
X0 = 0.0
Y0 = 0.0
X1 = 5.0
Y1 = 5.0

XG_Raw = np.zeros((2, 200))
YG_Raw = np.zeros((1, 200))

for i in range(100):
    XG_Raw[0, i] = X0 + random.gauss(0, 4)
    XG_Raw[1, i] = Y0 + random.gauss(0, 4)
    YG_Raw[0, i] = 0
for i in range(100, 200):
    XG_Raw[0, i] = X1 + random.gauss(0, 4)
    XG_Raw[1, i] = Y1 + random.gauss(0, 4)
    YG_Raw[0, i] = 1

plt.scatter(XG_Raw[0,0:100], XG_Raw[1,0:100], c='r', marker='.')
plt.scatter(XG_Raw[0,100:200], XG_Raw[1,100:200], c='b', marker='.')
plt.show()
```

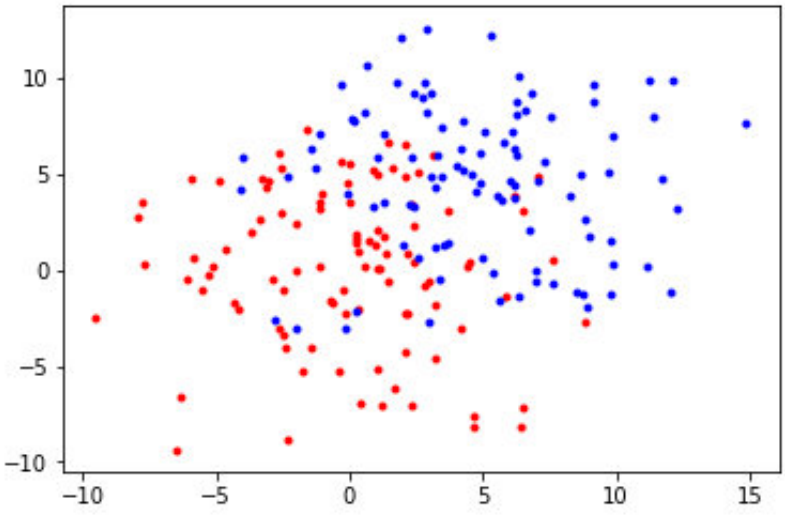
生成不满足朴素贝叶斯的数据集：

```
X0 = 0.0
Y0 = 0.0
X1 = 5.0
Y1 = 5.0

XG_Raw = np.zeros((2, 200))
YG_Raw = np.zeros((1, 200))

for i in range(100):
    XG_Raw[0, i] = X0 + random.gauss(0, 4)
```

```
    XG_Raw[1, i] = 2 * XG_Raw[0, i] + 1
    YG_Raw[0, i] = 0
for i in range(100, 200):
    XG_Raw[0, i] = X1 + random.gauss(0, 4)
    XG_Raw[1, i] = 2 * XG_Raw[0, i] + 1
    YG_Raw[0, i] = 1

plt.scatter(XG_Raw[0,0:100], XG_Raw[1,0:100], c='r', marker='.')
plt.scatter(XG_Raw[0,100:200], XG_Raw[1,100:200], c='b', marker='.')
plt.show()
```

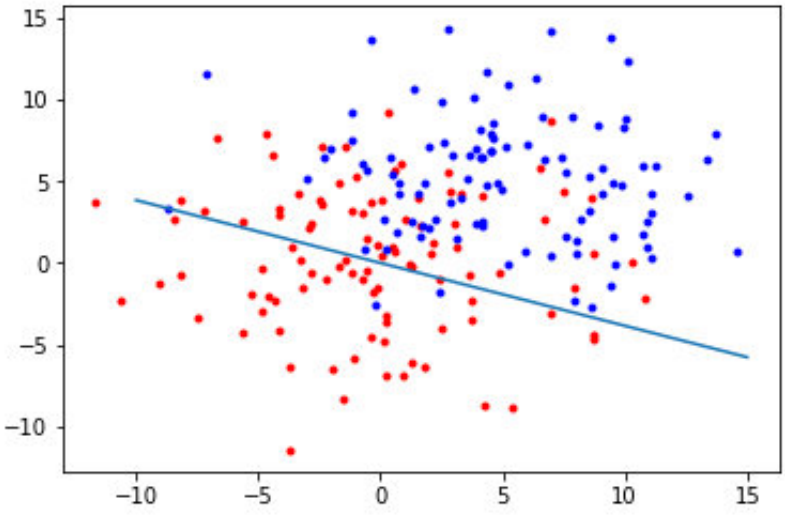
## 四、实验结果与分析

生成满足朴素贝叶斯的数据集：

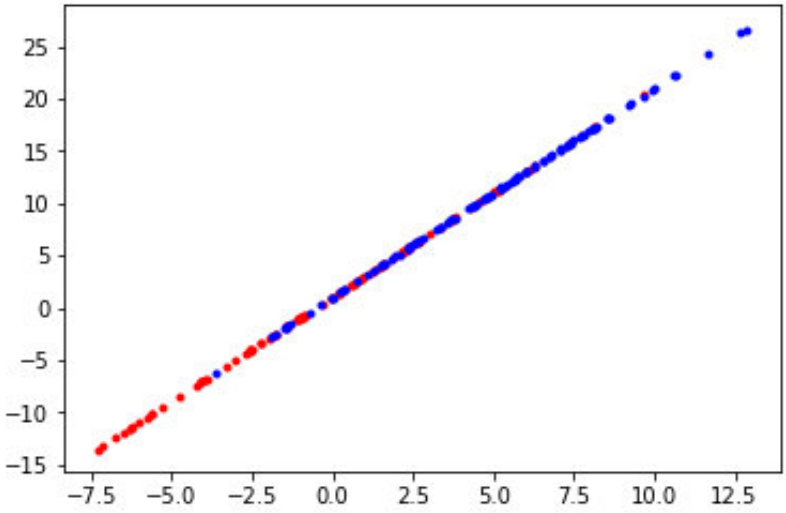



对满足朴素贝叶斯数据集进行分类时得到的结果为：

```
训练集准确率：97.14285714285714%
测试集准确率：93.33333333333333%
```




生成不满足朴素贝叶斯的数据集：

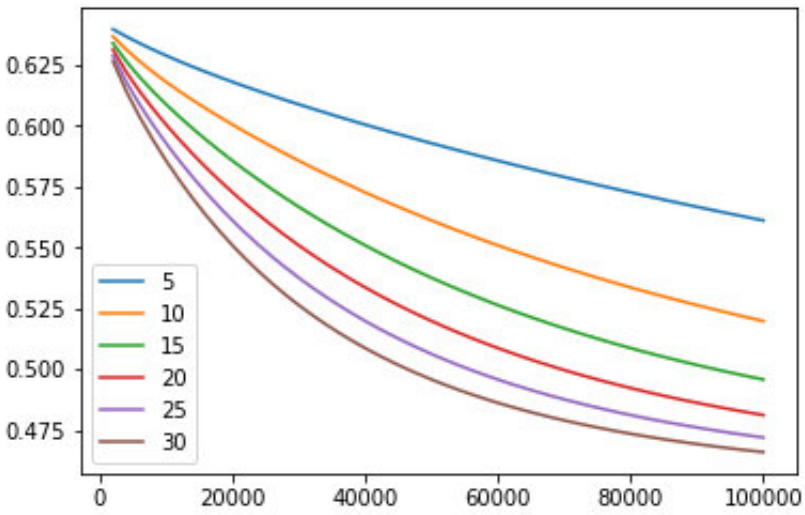



对不满足朴素贝叶斯数据集进行分类时得到的结果为：

```
训练集准确率：92.85714285714286%
测试集准确率：86.66666666666667%
```

使用不同的`learning_rate`对Titanic进行分类时得到的结果以及cost变化：

```
learning rate为5时准确率：76.86567164179104%
learning rate为10时准确率：81.34328358208955%
learning rate为15时准确率：83.58208955223881%
learning rate为20时准确率：82.83582089552239%
learning rate为25时准确率：82.83582089552239%
learning rate为30时准确率：82.46268656716418%
```




## 五、结论

1. 在数据集比较小时，使用朴素贝叶斯较逻辑回归效果好
2. 朴素贝叶斯在处理不严格独立的数据集时效果较差，但是逻辑回归效果较好

## 六、参考文献

[1]. Logistic_regression, https://en.wikipedia.org/wiki/Logistic_regression

[2]. Deeplearning.ai, https://mooc.study.163.com/smartSpec/detail/1001319001.htm