哈尔滨工业大学计算学部

# 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：1183710109

姓名：郭茁宁

# 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。

## 验证方法

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:Windows10
- Python3.7

# 三、算法设计和原理分析

## <一>算法原理

$Logistic$回归的主要实现方式有两种，一是梯度下降算法，二是牛顿法。后者相对于前者的主要优势是速度快，迭代次数少。

梯度下降的主要算法依旧是对于多项式的梯度计算，然后在梯度方向来进行下降以找到更好的结果（即得到的函数对于所有样本点而言进行了□较好的划分）。

对于得到的划分我们可以表示为

$$0 = w_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m \tag{1}$$

这在二维上表现为直线。

特别的对于二维特征，我们可以表示为

$$Data = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{n1} & x_{n2} \end{bmatrix}, Result = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} \tag{2}$$

其中$Data$为二维特征，$Result$为每个样本□点的类型，后者只有0和1之分（严格的伯努利分布）。

从概率上讲，一件事发生的概率为$p$的话，那么不发生的概率就是$1-p$，对于样本而言，属于第一类的概率符合这个理论，只不过这里的概率非0即1。我们定义这件事发生的几率，并给出表达

$$Logit(p) = \frac{p}{1-p} \tag{3}$$

那么我们将此带入(1)中有

$$\begin{aligned} Logit(P(y=1|x)) &= w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n \\ &= \sum_{i=1}^{n} w_ix_i \\ &= W^TX \end{aligned}$$

其中

$$W = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix}, X = Data$$

输入所有的样本之后为了实现划分，我们其实都设置了一个限制，即超过多少的概率时为1反之为0。我们有□以下函数计算概率

$$\phi(z) = \frac{1}{1+e^z} \tag{4}$$

称之为$sigmoid$函数。

致此我们终于可以通过函数来计算概率并且进行判断。

在线性感知器中，我们通过梯度下降算法来使得预测值与实际值的误差的平方和最小，来求得权重和阈值。而在逻辑斯蒂回归中所定义的代价函数就是使得该件事情发生的几率最大，也就是某个样本属于其真实标记样本的概率越大越好。这里我们使用梯度上升。

我们有最大似然函数

$$L(\omega) = P(y|x;\omega) = \prod_{i=1}^{n} P(y^{(i)}|x^{(i)};\omega) = (\phi(z^{(i)}))^{y^{(i)}}(1-\phi(z^{(i)}))^{1-y^{(i)}}$$

为了防止溢出取对数

$$l(\omega) = L(\omega) = P(y|x;\omega) = \sum_{i=1}^{n}(y^{(i)}log(\phi(z^{(i)})) + (1-y^{(i)})log(1-\phi(z^{(i)})) \tag{5}$$

为了判断是否足够收敛，我们定义□$loss$

$$J(\phi(z), y;\omega) = \begin{cases} log(\phi(z)), & y=1, \\ log(1-\phi(z)), & y=0. \end{cases} \tag{6}$$

式(5)与式(6)相等。

在进行了最初的计算之后我们需要对权重$W$进行更新。

我们对似然函数求偏导数得到

$$\frac{\partial}{\partial\omega_j}l(\omega) = (y-\phi(z))x_j \tag{7}$$

这便是梯度，更新后的$W$为

$$W = W - \frac{1}{m}(Data^T \cdot (Result - \phi(z))) \tag{8}$$

添加正则化惩罚之后变为

$$J(\omega) = l(\omega) + \frac{\lambda}{2m}||\omega||_2^2 \tag{9}$$

$$W = W - \frac{1}{m}(Data^T \cdot (Result - \phi(z))) - \frac{\lambda}{m}W$$

以上所有均适用于

$$z = X^T \cdot W \tag{10}$$

对于牛顿法，假设$L(w)$具有二阶连续偏导数，若第$k$次的迭代值为$w^{(k)}$，则可将$L(w)$在$w^{(k)}$附近进行二阶泰勒展开：

$$L(w) = L(w^{(k)}) + g_k^T(w - w^{(k)}) + \frac{1}{2}(w - w^{(k)})^T H(w^{(k)})(w - w^{(k)}) \tag{11}$$

这里，$g_k = g(w^{(k)}) = \nabla L(w^{(k)})$是$L(w)$的梯度向量在$w^{(k)}$处的值，$H(w^{(k)})$是$L(w)$的海森矩阵。

$$H(w) = [\frac{\partial^2 L}{\partial w_i \partial w_j}]_{n\times n} \tag{12}$$

在$w^{(k)}$处的值。函数$L(w)$取得极值的必要条件是一阶导数为0（即梯度为0）

$$\nabla L(w) = 0 \tag{13}$$

假设在迭代过程中第$k+1$次迭代使得$\nabla L(w) = 0$，则有：

$$\nabla L(w) = g_k + H_k(w - w^{(k)}) \tag{14}$$

将$H_k = H(w^(k))$代入，有：

$$g_k + H_k(w^{(k+1)} - w^{(k)}) = 0 \tag{15}$$

因此可得迭代式：

$$w^{(k+1)} = w^{(k)} - H_k^{-1} g_k \tag{16}$$

# <二>算法实现

## 梯度下降：

对于加入正则的梯度下降，在计算出加入正则的权重更新函数之后，必须对于loss也加入正则项使得在必要的地方停止迭代。

```
def cost_function(W, X, Y):
    """
    cost1为对数似然函数
    cost2为损失函数
    """
    cost = -np.dot(Y.T, np.log(model(X, W))) - np.dot(
        (np.ones((scale_of_example * 2, 1)) - Y).T,
        np.log(np.ones((scale_of_example * 2, 1)) - model(X, W)),
    )
    cost = sum(cost) / len(X) + (hyper_parameter * (np.dot(W.T, W))) / (2 *
len(X))
    print(cost)
    return cost
```

```
def gradient_function(W, X, Y):
    """
    计算梯度
    """
    gradient = np.dot(X.T, (model(X, W) - Y))
    # print(gradient)
    return gradient


def gradient_decent(W, example_added, label, threshold):
    """
    梯度下降法
    """
    alpha = 0.0001
    gradient = gradient_function(W, example_added, label)
    while cost_function(W, example_added, label) > threshold:
        W = W - alpha * gradient
        gradient = gradient_function(W, example_added, label)
    return W
```

### 牛顿法：

```
def Hessian(W, X, Y):
    """
    生成黑塞矩阵
    """
    hessianMatrix = np.zeros((dim + 1, dim + 1))
    for t in range(scale_of_example * 2):
        X_mat = np.mat(X[t]).T
        XXT = np.array(X_mat * X_mat.T)
        hessianMatrix += sigmoid(np.dot(X[t], W)) * (sigmoid(np.dot(X[t], W)) -
1) * XXT
    return hessianMatrix


def newton_method(W, example_added, label, threshold):
    """
    牛顿法
    """
    gradient = gradient_function(W, example_added, label)
    alpha = 0.01
    while cost_function(W, example_added, label) > threshold:
        H = np.linalg.inv(Hessian(W, example_added, label))
        W = W + alpha * np.dot(H, gradient)
        gradient = gradient_function(W, example_added, label)
    return W
```

# 四、实验结果分析

## 1、生成数据结果

## (1)满足朴素贝叶斯无正则项结果如下图：

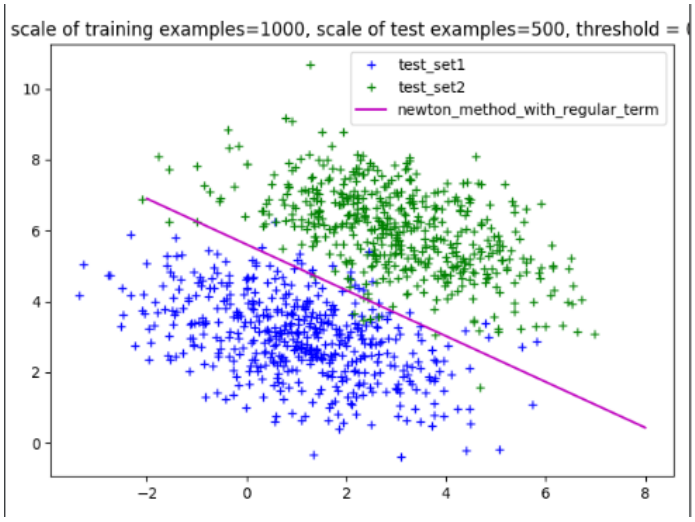
使用牛顿法对生成的训练集计算逻辑回归曲线并使用使用改逻辑回归曲线测试生成的测试集，正确率和图像如图。



## (2)满足朴素贝叶斯有正则项结果如下图：

使用牛顿法对生成的训练集计算逻辑回归曲线，并使用使用改逻辑回归曲线测试生成的测试集，正确率和图像。


### (3)不满足朴素贝叶斯无正则项结果如下图：





### (4)不满足朴素贝叶斯有正则项结果如下图：



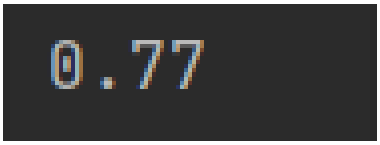
通过以上4种不条件下的测试可以看出，逻辑回归分类器在在满足朴素贝叶斯假设时分类良好，在不满足朴素贝叶斯假设时分类效果也与满足时相差并不大，可能是由于数据维度只有二维，不会发生较大的偏差。并且在二维条件下，是否有惩罚项对其影响也不大，但当减小训练集时，所得到的判别函数确实存在过拟合现象，加入正则项可以预防此现象的发生。

## 2、使用UCI数据训练并测试

在UCI网站找到数据，储存在文件balance-scale.data中，数据量有200000，先随机打乱数据的顺序，再选取30%的数据作为训练集，再把剩余70%的数据作为测试集。

## (1)无正则项的结果

准确率如下：




## (2)有正则项的结果

准确率如下：


使用UCI数据测试时有时候也会出现震荡的情况。

# 五、结论

- 关于牛顿法：牛顿法每次迭代的时间代价为O( ) ,相比梯度下降法，每次的时间开销和空间占 用会更大。但是牛顿法仅需大约10-15次就能找到最小值，比梯度下降法快得多（200次左右）。但 是牛顿法的计算过程中涉及求黑塞矩阵的逆，如果矩阵奇异，则牛顿法不再适用。
- 牛顿法可以得到很好的结果，相比于梯度下降法，牛顿法的迭代收敛的速度较快。
- 对于进行数学推导时使用的假设(包括朴素贝叶斯假设和 仅与类别相关与维度无关)，当打破假设时，得到的分类结果仍然较好，这与[Domingos&Pazzani, 1996]中的阐述相符合。
- 关于精度：python编译器默认浮点数为float32，有时候精度丢失会比较严重，如果需要使用float64 表示数据，需要自己手动设置。
- 关于sigmoid函数，sigmoid函数可能会发生溢出，主要是当z<<0时时 >>0,会发生溢出。

# 六、代码

```
//手动生成文件测试
import matplotlib.pyplot as plt
import numpy as np


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


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


def cal_loss(W, X, Y):
    """
    cost1为对数似然函数
    cost2为损失函数
    """
    cost = -np.dot(Y.T, np.log(model(X, W))) - np.dot(
```

```
        (np.ones((scale_of_example * 2, 1)) - Y).T,
        np.log(np.ones((scale_of_example * 2, 1)) - model(X, W)),
    )
    cost = sum(cost) / len(X) + (hyper_parameter * (np.dot(W.T, W))) / (2 *
len(X))
    print(cost)
    return cost


def cal_gradient(W, X, Y, lamb):
    """
    计算梯度
    """
    gradient = np.dot(X.T, (model(X, W) - Y))
    # print(gradient)
    return gradient+lamb*W


def gradient_decent(W, example_added, label, threshold):
    """
    梯度下降法
    """
    alpha = 0.001
    gradient = cal_gradient(W, example_added, label)
    while cal_loss(W, example_added, label) > threshold:
        W = W - alpha * gradient
        gradient = cal_gradient(W, example_added, label)
    return W


def Hessian(W, X, Y):
    """
    生成黑塞矩阵
    """
    hessianMatrix = np.zeros((dim + 1, dim + 1))
    for t in range(scale_of_example * 2):
        X_mat = np.mat(X[t]).T
        XXT = np.array(X_mat * X_mat.T)
        hessianMatrix += sigmoid(np.dot(X[t], W)) * (sigmoid(np.dot(X[t], W)) -
1) * XXT
    return hessianMatrix


def newton(W, example_added, label, threshold, step = 10):
    """
    牛顿法
    """
    gradient = cal_gradient(W, example_added, label, 0.01)
    alpha = 0.01
    while cal_loss(W, example_added, label) > threshold:
        H = np.linalg.inv(Hessian(W, example_added, label))
        W = W + alpha * np.dot(H, gradient)
        gradient = cal_gradient(W, example_added, label, 0.01)
    return W


def judge(W):
    """
```

```
    判断回归效果
    """
    judge_scale = 500
    s1 = np.dot(np.random.randn(judge_scale, dim), R1) + mu1
    plt.plot(s1[:, 0], s1[:, 1], "+", label="test_set1", color="b")
    s2 = np.dot(np.random.randn(judge_scale, dim), R2) + mu2
    plt.plot(s2[:, 0], s2[:, 1], "+", label="test_set2", color="g")
    example = np.vstack((s1, s2))
    label1 = np.zeros((judge_scale, 1))
    label2 = np.ones((judge_scale, 1))
    test_label = np.vstack((label1, label2))
    test_set = np.hstack((np.ones((judge_scale * 2, 1)), example))
    result = np.zeros((judge_scale * 2, 1))
    correct_num = 0
    for i in range(judge_scale * 2):
        if model(test_set, W)[i - 1][0] > 0.5:
            result[i - 1][0] = 1
    for i in range(judge_scale * 2):
        if result[i - 1][0] == test_label[i - 1][0]:
            correct_num += 1
    return correct_num / (judge_scale * 2)


if __name__ == "__main__":
    """
    生成训练数据，为二维随机高斯分布
    label为二分类分别为0和1
    hyper_parameter为\lambda
    """
    scale_of_example = 1000
    dim = 2
    mu1 = np.array([[1, 3]])
    Sigma1 = np.array([[2, -1], [-1, 2]])
    R1 = np.linalg.cholesky(Sigma1)
    s1 = np.dot(np.random.randn(scale_of_example, dim), R1) + mu1
    # plt.plot(s1[:, 0], s1[:, 1], ".", label="training_set1", color="red")

    mu2 = np.array([[3, 6]])
    Sigma2 = np.array([[2, -1], [-1, 2]])
    R2 = np.linalg.cholesky(Sigma2)
    s2 = np.dot(np.random.randn(scale_of_example, dim), R2) + mu2
    # plt.plot(s2[:, 0], s2[:, 1], ".", label="training_set2", color="yellow")

    example = np.vstack((s1, s2))
    label1 = np.zeros((scale_of_example, 1))
    label2 = np.ones((scale_of_example, 1))
    label = np.vstack((label1, label2))
    data = np.hstack((example, label))
    W = np.ones((dim + 1, 1))

    hyper_parameter = 0.01

    example_added = np.hstack((np.ones((scale_of_example * 2, 1)), example))
    cal_loss(W, example_added, label)

    # W = gradient_decent(W, example_added, label, 0.1)     # 梯度下降法
    W = newton(W, example_added, label, 0.1)  # 牛顿法
    print(judge(W))
```

```
    X1 = np.linspace(-2, 8, 20)
    X2 = -W[0][0] / W[2][0] - np.dot(W[1][0], X1) / W[2][0]
    plt.plot(X1, X2, label="newton_method_with_regular_term", color="m")
    plt.title("scale of training examples=1000, scale of test examples=500,
threshold = 0.1")
    plt.legend()
    plt.show()

//UCI测试
import numpy as np
import io
import re
import operator


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


def model(X, W):
    return sigmoid(np.dot(X, W))


def cal_loss(W, X, Y):
    cost = -np.dot(Y.T, np.log(model(X, W))) - np.dot(
        (np.ones((scale_of_example * 2, 1)) - Y).T,
        np.log(np.ones((scale_of_example * 2, 1)) - model(X, W)),
    )
    cost = sum(cost) / len(X) + (hyper_parameter * (np.dot(W.T, W))) / (2 *
len(X))
    print(cost)
    return cost


def cal_gradient(W, X, Y, lamb):
    gradient = np.dot(X.T, (model(X, W) - Y))
    return gradient + lamb*W


def gradient_decent(W, example_added, label, threshold):
    alpha = 0.0001
    gradient = cal_gradient(W, example_added, label, 0.05)
    while cal_loss(W, example_added, label) > threshold:
        W = W - alpha * gradient
        gradient = cal_gradient(W, example_added, label, 0.05)
    return W


def judge(W):
    judge_scale = 50
    result = np.zeros((judge_scale * 2, 1))
    correct_num = 0
    for i in range(judge_scale * 2):
        if model(test_set, W)[i - 1][0] > 0.5:
            result[i - 1][0] = 1
    for i in range(judge_scale * 2):
        if result[i - 1][0] == test_label[i - 1][0]:
            correct_num += 1
```

```
    return correct_num / (judge_scale * 2)


if __name__ == "__main__":
    bl = io.open("balance-scale.data", encoding="UTF-8")
    bl_list = bl.readlines()
    scale_of_example = 200
    dim = 4
    example = [1, 1, 1, 1, 1]
    label = []
    i = 0
    '''
    二分类只选取左倾或者右倾的数据
    平衡状态不考虑
    选取前400个数据训练
    后一百个数据测试
    '''
    for line in bl_list:
        l = re.split('[,\n]', line)
        if operator.eq(l[0], 'L'):
            label.append(0)
            tmp = np.mat([1, int(l[1]), int(l[2]), int(l[3]), int(l[4])])
            example = np.vstack((example, tmp))
        elif operator.eq(l[0], 'R'):
            label.append(1)
            tmp = np.mat([1, int(l[1]), int(l[2]), int(l[3]), int(l[4])])
            example = np.vstack((example, tmp))
        else:
            continue
        i = i + 1

    example_added = example[1:401, :]
    test_set = example[401:501, :]
    training_label = np.mat(label).T[1:401, :]
    test_label = np.mat(label).T[401:501, :]
    label = training_label
    W = np.zeros((dim + 1, 1))
    hyper_parameter = 0.00001
    cal_loss(W, example_added, label)
    W = gradient_decent(W, example_added, label, 0.402)
    print(judge(W))
```