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：1170300418

姓名：于新蕊

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

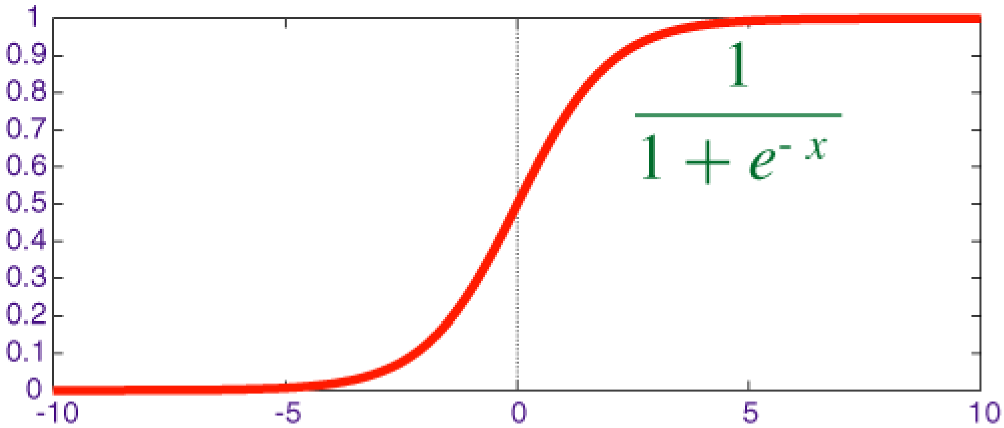
windows64，pycharm，python3.0，anaconda

三、设计思想（本程序中的用到的主要算法及数据结构）

逻辑回归

逻辑回归是一种经典的分类模型。

定义**sigmoid函数**：$y = \mathrm{sigmoid}(x) = \frac{1}{1+e^{-x}}$，图像如下：



假设函数$h_\theta(x) = \mathrm{simgoid}(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$。

sigmoid函数值域$(0,1)$，所以$h_\theta(x)$可以理解为预测$y = 1$的概率。

即$h_\theta(x) >= 0.5$时，预测$y = 1$；$h_\theta(x) < 0.5$预测$y = 0$。

那么

$$P(y=1|x;\theta)=h_\theta(x)$$
$$P(y=0|x;\theta)=1-h_\theta(x)$$

即

$$P(y|x;\theta)=(h_\theta(x))^y(1-h_\theta(x))^{1-y}$$

我们用**最大似然法**来估计参数θ:

$$\begin{aligned}L(\theta)&=\prod_{i=1}^{m}P\left(y^{(i)}|x^{(i)};\theta\right)\\&=\prod_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)\right)^{y^{(i)}}\left(1-h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}\end{aligned}$$

$$l(\theta)=\log(L(\theta))=\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right)$$

最大似然估计就是要求得使$l(\theta)$取最大值时的θ。

因此，我们定义**代价函数**$J(\theta)=-\frac{1}{m}l(\theta)$。最大化$l(\theta)$相当于最小化$J(\theta)$。

可以证明$J(\theta)$是个凸函数，因此我们可以用梯度下降法来求解。

梯度下降法（无正则项）

选择一个学习率α，迭代次数$epcho$，每次沿梯度下降最快的方向，即导数方向下降，具体来讲：每次迭代可以表示为$\theta_j=\theta_j-\alpha\frac{\partial J(\theta)}{\partial\theta_j}$。即$\theta=\theta-\alpha\frac{\partial J(\theta)}{\partial\theta}$。

$$\begin{aligned}\frac{\partial}{\partial\theta_j}l(\theta)&=\left(y\frac{1}{h_\theta(x)}-(1-y)\frac{1}{1-h_\theta(x)}\right)\frac{\partial}{\partial\theta_j}h_\theta(x)\\&=\left(y\frac{1}{h_\theta(x)}-(1-y)\frac{1}{1-h_\theta(x)}\right)h_\theta(x)\left(1-h_\theta(x)\frac{\partial}{\partial\theta_j}\theta^Tx\right)\\&=\left(y\left(1-h_\theta(x)\right)-(1-y)h_\theta(x)\right)x_j\\&=\left(y-h_\theta(x)\right)x_j\end{aligned}$$

所以

$$\frac{\partial}{\partial\theta_j}J(\theta)=-\frac{1}{m}\left(y-h_\theta(x)\right)x_j$$

和线性回归代价函数的导数表达式一致，不同的时$h_\theta(x)$的表达式。

所以迭代为

$$\begin{aligned}&\text{repeat epcho times}\{\\&\qquad\theta_j=\theta_j-\tfrac{\alpha}{m}\textstyle\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)x_j^{(i)}\\&\}\text{for } j=0\cdots n,\ \text{simultaneous update}\end{aligned}$$

梯度下降法（有正则项）

根据贝叶斯公式：

$$p(\omega|\mathbf{D})=\frac{p(\mathbf{D}|\omega)p(\omega)}{p(\mathbf{D})}$$

$p(\omega|\mathbf{D})$是后验概率，$p(\mathbf{D}|\omega)$是似然函数，$p(\omega)$是先验概率。

之前我们考虑的是求出θ使得$\prod_{i=1}^{m}p(\mathbf{X}|\theta)$最大，也就是求"哪个$\theta$可以使训练集中的所有情况出现的概率最大"。现在我们考虑$\prod_{i=1}^{m}p(\theta|\mathbf{X})$最大，也就使求"出现这样的训练集最可能是哪个$\theta$"。

即

$$\theta=\underset{\theta}{\operatorname{argmax}}\,P(\theta|\mathbf{X})$$

由贝叶斯公式知：$P(\theta|\mathbf{X})\propto P(\mathbf{X}|\theta)P(\theta)$

所以最大后验估计$\theta=\underset{\theta}{\operatorname{argmax}}\,P(\mathbf{X}|\theta)P(\theta)$

设先验函数服从高斯分布$P(\theta)=se^{-\frac{\alpha}{2}\theta^T\theta}$

类似似然函数取对数，那么最大后验函数为

$$\begin{aligned}\widehat{M(\theta)}&=\ln P(\theta)+\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right)\\&=-\frac{\alpha}{2}\theta^T\theta+\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right)\\&=-\frac{\lambda}{2}\theta^T\theta+\sum_{i=1}^{m}\left(y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right)+\left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right)\end{aligned}$$

代价函数$J(\theta)=-\frac{1}{m}l(\theta)+\frac{\lambda}{2m}\theta^T\theta$

$\frac{\partial}{\partial\theta_j}J(\theta)=-\frac{1}{m}(y-h_\theta(x))\,x_j+\frac{\lambda}{m}\theta_j$

迭代方法为

$$\begin{aligned}&\text{repeat epcho times}\{\\&\quad\theta_j=\theta_j(1-\frac{\alpha\lambda}{m})-\frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)}\\&\}\text{for } j=0\ \cdots\ n,\text{simultaneous update}\end{aligned}$$

生成数据

满足朴素贝叶斯假设

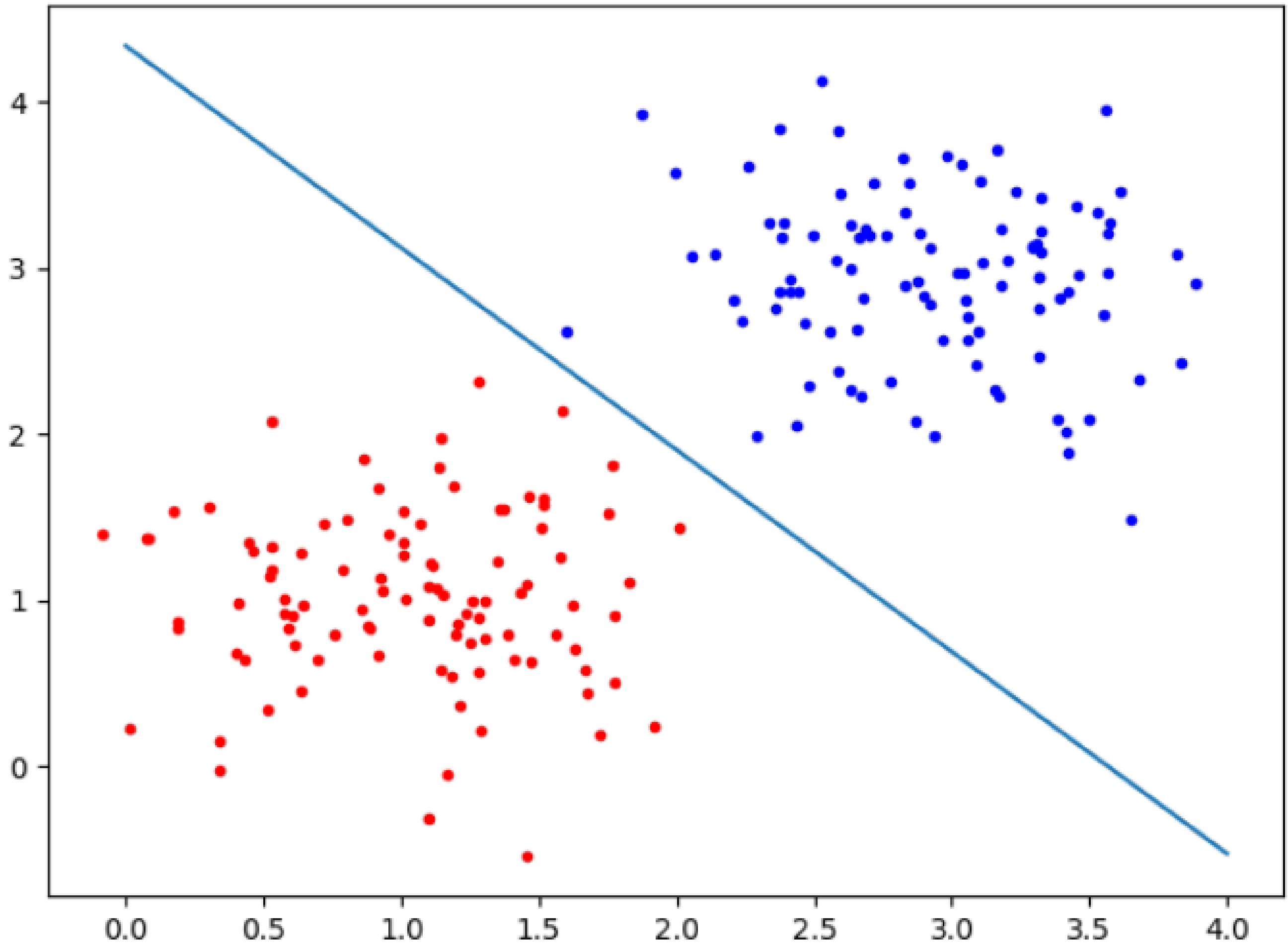
`generate_data()`每一维的根据传入的均值、方差参数单独生成高斯分布的数据

不满足朴素贝叶斯假设

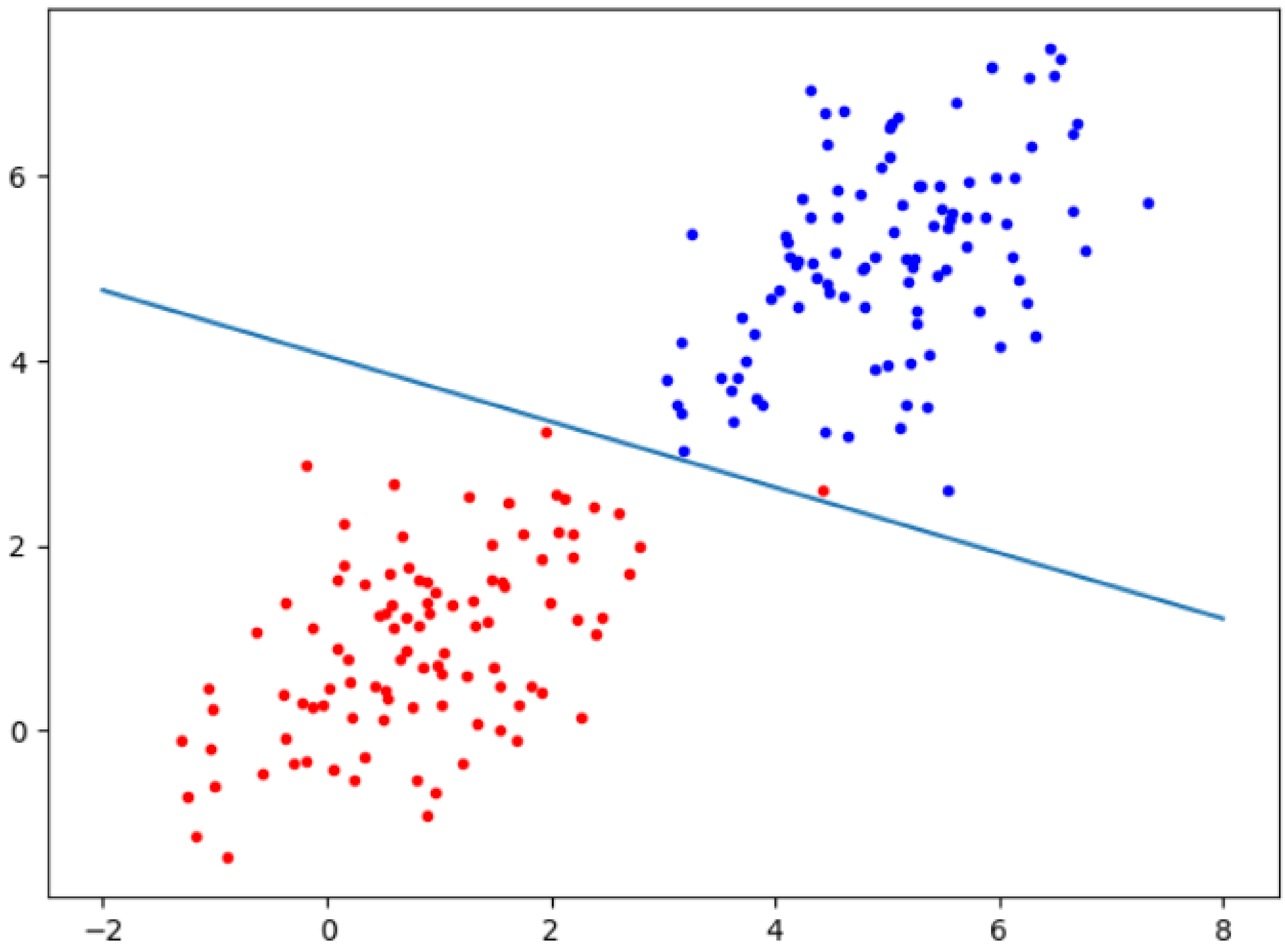
利用协方差矩阵对角线非零构造相关数据，协方差矩阵应该半正定。

四、实验结果与分析

满足朴素贝叶斯假设



不满足朴素贝叶斯假设



真实数据

breast-cancer.data（9个特征，训练集150，测试集136）

```
[[-0.96943628]
 [-0.3322818 ]
 [-4.19733521]
 [-2.85282353]
 [-7.59293382]
 [ 6.39738371]
 [ 1.08975777]
 [-2.89424029]
 [-1.93887257]
 [-4.81113044]]
accuracy: 136/136
```

分析

lr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依然不错，原因是因为维数只有二维，不会引发过多的错乱。该分类器在真实数据集上表现良好。

五、结论

逻辑回归属于概率性判别式模型，之所谓是概率性模型，是因为逻辑回归模型是有概率意义的；之所以是判别式模型，是因为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，而是直接将判别函数，或者说是分类超平面求解了出来。

六、参考文献

吴恩达机器学习(Coursera)

逻辑回归

七、附录：源代码（带注释）

```
import numpy as np
import matplotlib.pyplot as plt


def generate_data(loc1, scale1, size1, loc2, scale2, size2):
    """
    生成数据
    :param loc1: 类别1的均值
    :param scale1: 类别1的方差
    :param size1: 类别1数据个数
    :param loc2: 类别2的均值
    :param scale2: 类别2的方差
    :param size2: 类别2数据个数
    :return: 数据集
    """
    x1 = np.empty((size1, loc1.size))
    x2 = np.empty((size2, loc2.size))
    y1 = np.ones(size1)
    y2 = np.zeros(size2)
    for i in range(size1):
        for j in range(loc1.size):
            x1[i][j] = np.random.normal(loc1[j], scale1[j], 1)
    for i in range(size2):
        for j in range(loc2.size):
            x2[i][j] = np.random.normal(loc2[j], scale2[j], 1)
    if loc1.size == 2:
        plt.plot(x1[:, 0:1], x1[:, 1:], '.', color='red')
        plt.plot(x2[:, 0:1], x2[:, 1:], '.', color='blue')
    x1 = np.column_stack((x1, y1))
    x2 = np.column_stack((x2, y2))
    x1 = np.row_stack((x1, x2))
    np.random.shuffle(x1)
```

```
    return x1[:, :-1], x1[:, -1:]


def sigmoid(x):
    """
    sigmoid函数
    :param x: x
    :return: sigmoid(x)
    """
    return 1 / (1 + np.exp(-x))


def cost_function(theta, x, y):
    """
    J(theta) =  -1/m * sum(y_i * log(sigmoid(x_i * theta)) + (1 - y_i) * log(1 - 
sigmoid(x_i * theta)))
    :param theta: 参数向量theta
    :param x: x
    :param y: y
    :return: 代价函数的值
    """
    cost = np.sum(y.T * np.log(sigmoid(np.dot(theta, x))) + (1 - y).T * np.log(1 - 
sigmoid(np.dot(theta, x))))
    cost = -1 / y.size * cost
    return cost


def gradient_descent(x, y, alpha=0.05, epochs=100000, _lambda=0.0):
    """
    梯度下降法
    :param x: X矩阵
    :param y: y向量
    :param alpha: 学习率
    :param epochs: 迭代次数
    :param _lambda: 正则项系数，默认无正则项
    :return: 系数向量theta
    """
    x = np.column_stack((np.ones(y.size).T, x))
    n, m = x.shape
    theta = np.zeros(m).reshape(m, 1)
    # y1 = np.array([])
    while epochs > 0:
        hx = (sigmoid(np.dot(x, theta)) - y).T
        theta = (1 - _lambda * alpha / n) * theta - alpha / n * np.dot(hx, x).T
        epochs -= 1
    return theta


def get_accuracy(theta, x, y):
    """
    准确率
    :param theta:
    :param x:
```

```python
    :param y:
    :return:
    """
    x = np.column_stack((np.ones(y.size).T, x))
    predict_y = sigmoid(np.dot(x, theta))
    predict_y = predict_y.flatten()
    y = y.flatten()
    acc = 0
    for i in range(y.size):
        if (predict_y[i] >= 0.5) and (y[i] == 1):
            acc += 1
        if (predict_y[i] < 0.5) and (y[i] == 0):
            acc += 1
    print('accuracy: ' + str(acc) + '/' + str(y.size))


def uci_test():
    """
    uci数据测试
    :return:
    """
    f = open('breast-cancer.data', 'r')
    data = []
    lines = f.readlines()
    dic = dict()
    cnt = np.zeros(10)
    for line in lines:
        line = line.strip()
        line = line.split(',')
        for i in range(len(line)):
            if dic.get(line[i], 0) == 0:
                cnt[i] += 1
            dic[line[i]] = int(cnt[i])
            line[i] = int(cnt[i])
        data.append(line)
    data = np.array(data)
    np.random.shuffle(data)
    train_x = data[0:150:, :-1:]
    train_y = data[0:150:, -1::]
    test_x = data[150::, :-1:]
    test_y = data[150::, -1::]
    theta = gradient_descent(train_x, train_y)
    print(theta)
    get_accuracy(theta, test_x, test_y)


def plot(theta, start, end):
    x1 = np.linspace(start, end, 1000)
    theta = -theta / theta[2][0]
    func = np.poly1d(np.array(theta.T)[0][-2::-1])
    y1 = func(x1)
    plt.plot(x1, y1)
    plt.show()
```

```python
def work():
    """
    二元测试
    :return:
    """
    loc1 = np.array([1, 1])
    loc2 = np.array([3, 3])
    scale1 = 0.5 * np.ones(2)
    scale2 = 0.5 * np.ones(2)
    size1 = 100
    size2 = 100
    data_set, y = generate_data(loc1, scale1, size1, loc2, scale2, size2)
    # print(_data_set, _y)
    theta = gradient_descent(data_set, y, 0.01)
    print(theta)
    """
    theta0 + theta1 x + theta2 y = 0
    y = -theta1 / theta2 x - theta0 / theta2
    """
    if loc1.size == 2:
        plot(theta, 0, 4)


def generate_related_data(mean1, cov1, size1, mean2, cov2, size2):
    """
    生成不独立的数据
    :param mean1: 类别1均值
    :param cov1: 类别1的2个特征协方差矩阵
    :param size1: 类别1的数据个数
    :param mean2: 类别2均值
    :param cov2: 类别2的2个特征协方差矩阵
    :param size2: 类别2的数据个数
    :return:
    """
    x1 = np.random.multivariate_normal(mean=mean1, cov=cov1, size=size1)
    x2 = np.random.multivariate_normal(mean=mean2, cov=cov2, size=size2)
    plt.plot(x1[:, 0:1], x1[:, 1:], '.', color='red')
    plt.plot(x2[:, 0:1], x2[:, 1:], '.', color='blue')
    y1 = np.ones(size1)
    y2 = np.zeros(size2)
    x1 = np.column_stack((x1, y1))
    x2 = np.column_stack((x2, y2))
    x1 = np.row_stack((x1, x2))
    np.random.shuffle(x1)
    return x1[:, :-1], x1[:, -1:]


def related_test():
    """
    相关数据测试
    :return:
```

```python
    """
    mean1 = np.array([1, 1])
    mean2 = np.array([5, 5])
    cov1 = np.array([[1, 0.5], [0.5, 1]])
    cov2 = np.array([[1, 0.5], [0.5, 1]])
    x, y = generate_related_data(mean1, cov1, 100, mean2, cov2, 100)
    theta = gradient_descent(x, y)
    plot(theta, -2, 8)


if __name__ == '__main__':
    # uci_test()
    work()
    # related_test()
```