哈尔滨工业大学计算机科学与计算机学院

实验报告

课程名称: 机器学习
课程类型: 选修
实验题目: 多项式拟合正弦函数
学号: 1170300511
姓名: 易亚玲

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（带正则项和不带正则项）

二、实验要求及环境

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

- x86-64,Win 10
- Pycharm 2019.1
- python 3.7

三、设计思想

3.1算法原理

- 二项逻辑回归模型：

$$P(Y=0|x)=\frac{1}{1+e^{\omega\cdot x+b}}$$

$$P(Y=1|x)=\frac{e^{\omega\cdot x+b}}{1+e^{\omega\cdot x+b}}$$

其中$x\in R^n$是输入，Y∈{0,1}是输出，$\omega\in R^n$和$b\in R$是参数，ω称为权值向量，b称为偏置，$\omega\cdot x$为ω和x的内积。有时为了方便，将权值向量和输入向量加以扩充，仍然记为ω和x,但是$\omega=(\omega^1,\omega^2,...,\omega^n,b)^T$，$x=(x^1,x^2,...,x^n,1)^T$。在这种情况下，二项逻辑回归模型如下：

$$P(Y=0|x)=\frac{1}{1+e^{\omega\cdot x}}$$

$$P(Y=1|x)=\frac{e^{\omega\cdot x}}{1+e^{\omega\cdot x}}$$

定义sigmoid函数为

$$sigmoid(z) = \frac{1}{1+e^{-z}}$$

- 似然函数法估计模型参数ω
  设$P(Y=1|x)=\pi(x)$,$P(Y=0|x)=1-\pi(x)$，则似然函数为
  $\prod[\pi(x_i)]^{y_i}[1-\pi(x_i)]^{1-y_i}$
  对数似然函数为

$$L(\omega) = \sum_{i=1}^{N}[y_i log\pi(x_i) + (1-y_i)log(1-\pi(x_i))]$$

$$= \sum_{i=1}^{N}[y_i(\omega\cdot x_i) - log(1+e^{\omega\cdot x_i})]$$

  不加正则项的损失函数为

$$L(\omega) = \sum_{i=1}^{N}[-y_i(\omega\cdot x_i) + log(1+e^{\omega\cdot x_i})]$$

加入正则项的损失函数

$$L(\omega) = -\frac{1}{N}\sum_{i=1}^{N}[y_i log\pi(x_i) + (1-y_i)log(1-\pi(x_i))] + \frac{\lambda}{2N}||\omega||_2^2$$

$$= \frac{1}{N}\sum_{i=1}^{N}[-y_i(\omega\cdot x) + log(1+e^{\omega\cdot x_i})] + \frac{\lambda}{2N}||\omega||_2^2$$

求L(ω)的极大值，得到ω的估计值
不加正则项求ω

$$\frac{\partial L}{\partial w_j} = x_{ij}(-y_i + sigmoid(wx_i))$$

接下来可以由随机梯度下降法求解ω
同理加入正则项的梯度为

$$\frac{\partial L}{\partial w_j} = \frac{1}{N}[x_{ij}(-y_i + sigmoid(wx_i)) + \lambda\cdot\omega]$$

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

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

这里，$g_k=g(\omega^{(k)})=\triangledown L(\omega^{(k)})$是$L(\omega)$的梯度向量在$\omega^{(k)}$处的值，$H(\omega^{(k)})$是$L(\omega)$的黑塞矩阵

$$H(\omega)=[\frac{\partial^2 L}{\partial\omega_i\partial\omega_j}]_{n\times n}$$

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

$$\triangledown L(\omega)=0$$

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

$$\triangledown L(\omega)=g_k+H_k(\omega-\omega^{(k)})$$

将$H_k=H(\omega^{(k)})$代入，有

$$g_k+H_k(\omega^{(k+1)-\omega^{(k)}})=0$$

因此可得迭代式

$$\omega^{(k+1)}=\omega^{(k)}-H_k^{-1}g_k$$

3.2算法的实现

3.2.0.变量：

- matrix = () # 读入数据矩阵
- test_matrix = () # 测试数据矩阵
- y = () # 分类情况，y[i]表示第i组数据的分类情况
- test_y = () # 测试数据集的分类情况
- x = () # 特征矩阵，其中x[i]表示第i个实例的特征取值情况,最后一维为1
- test_x = () # 测试数据集的特征矩阵
- w = () # 对应扩充特征后的w
- n = 0 # 特征数的个数，其中w是n+1维的
- dataSum = 0 # 数据量
- testSum = 0 # 测试数据集大小

3.2.1.生成数据（2维）

满足贝叶斯：协方差矩阵半正定，例如

$$cov=\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$$

不满足贝叶斯：当协方差不等于0时，两个参数相关，则不独立，例如，2维数据均相关，不独立

$$cov = \left(\begin{array}{cc} 2 & 1 \\ 1 & 2 \end{array} \right)$$

可以调用numpy.random.multivariate_normal生成多维高斯分布数据

3.2.2.读取数据

使用pandas.read_csv读取csv格式的数据，然后再将读入的DataFrame结构使用.valus转化为ndarray，然后使用矩阵切片和扩充生成x,y

3.2.3.随机梯度下降法

自行设置迭代次数door，每次选取一组数据，根据以下公式进行求解，观察不同迭代次数的收敛情况

$$\frac{\partial L}{\partial w_j} = x_{ij}(-y_i + sigmoid(wx_i))$$

3.2.4 牛顿法

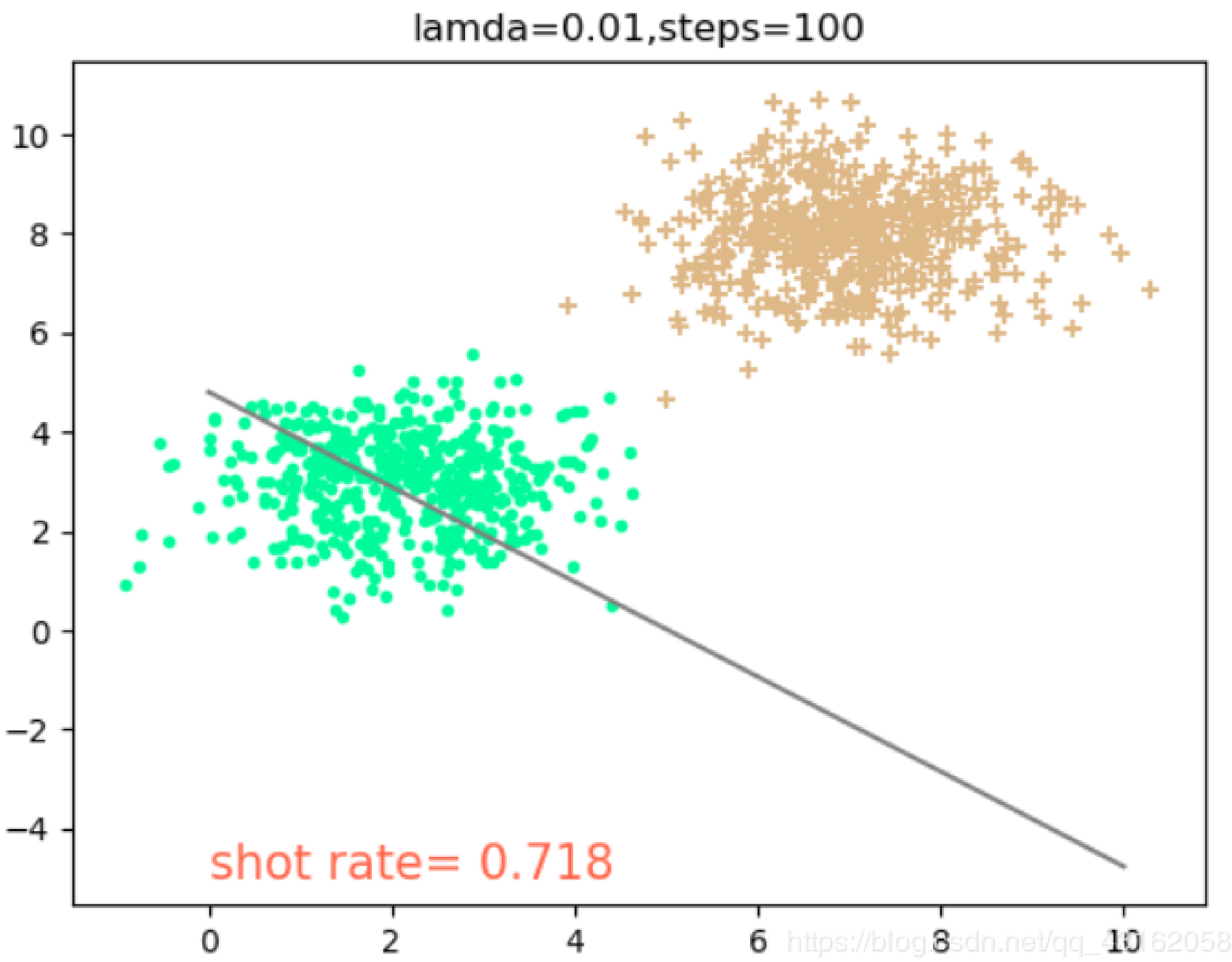
每迭代一次计算一次黑塞矩阵设置迭代次数，按次数迭代可得ω。

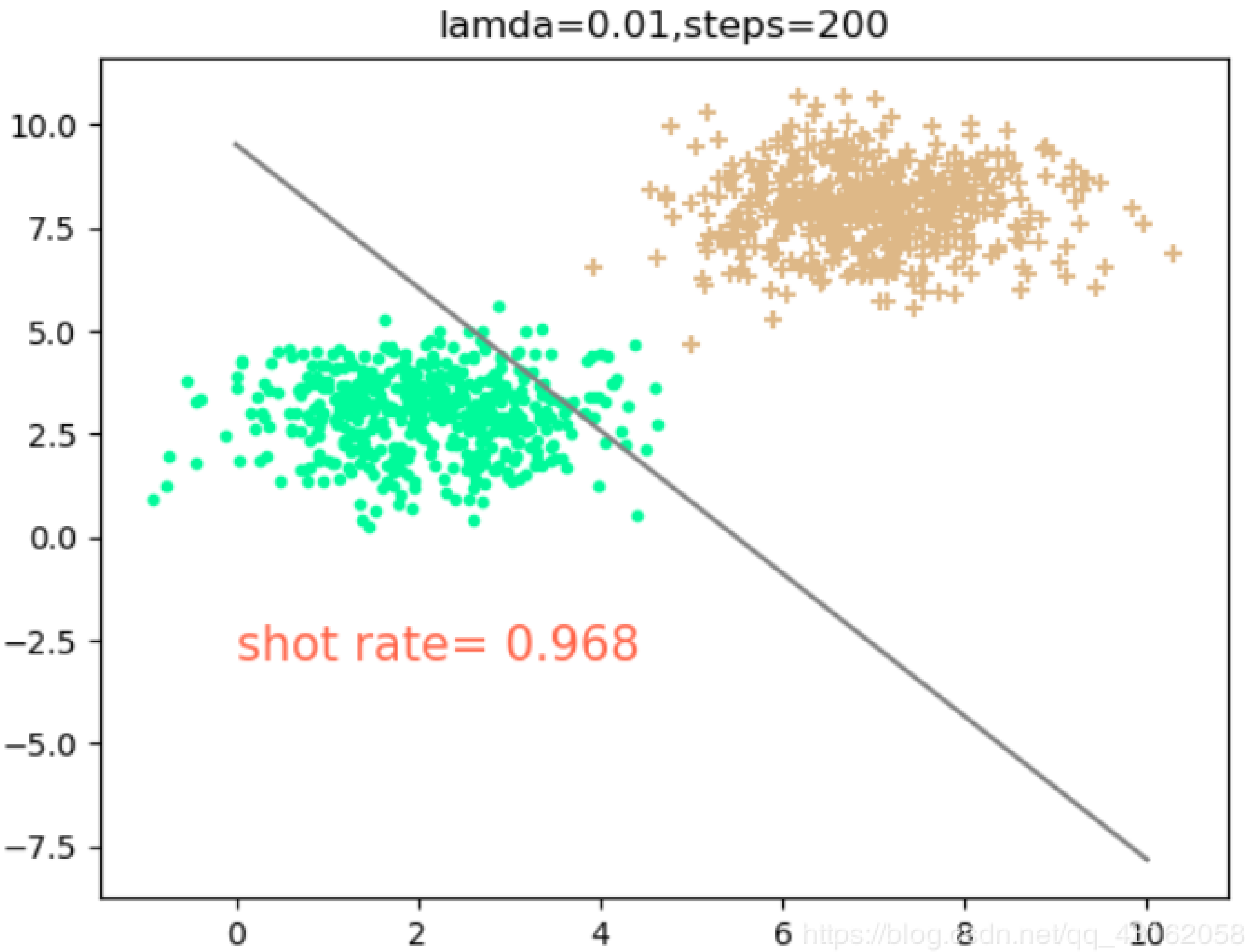
3.2.5. 计算正确率

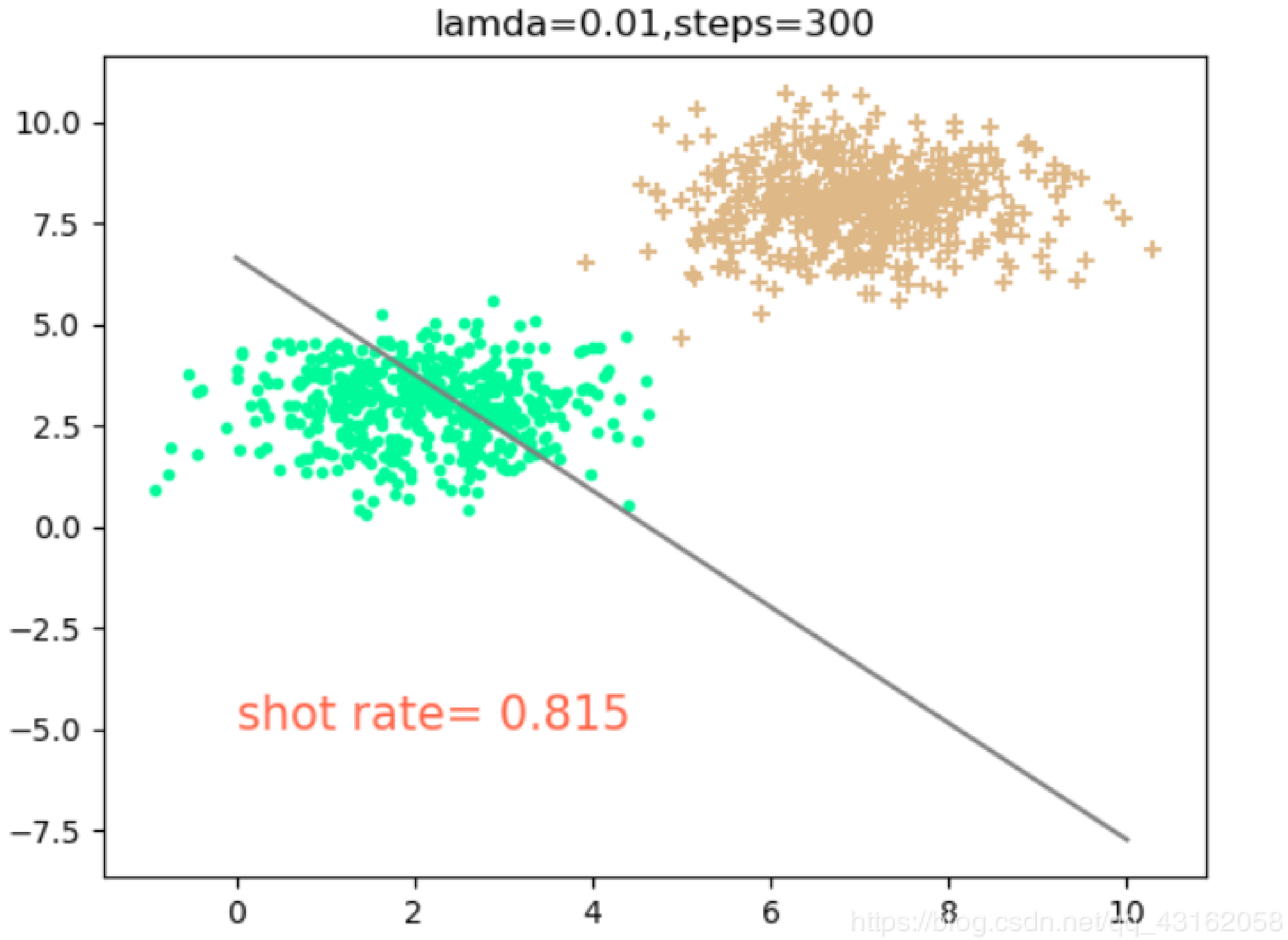
计算$\omega \cdot x$的值，与0比较，若大于或者等于零预测为1；小于0预测为0.统计预测正确的样本数，计算预测的正确率。

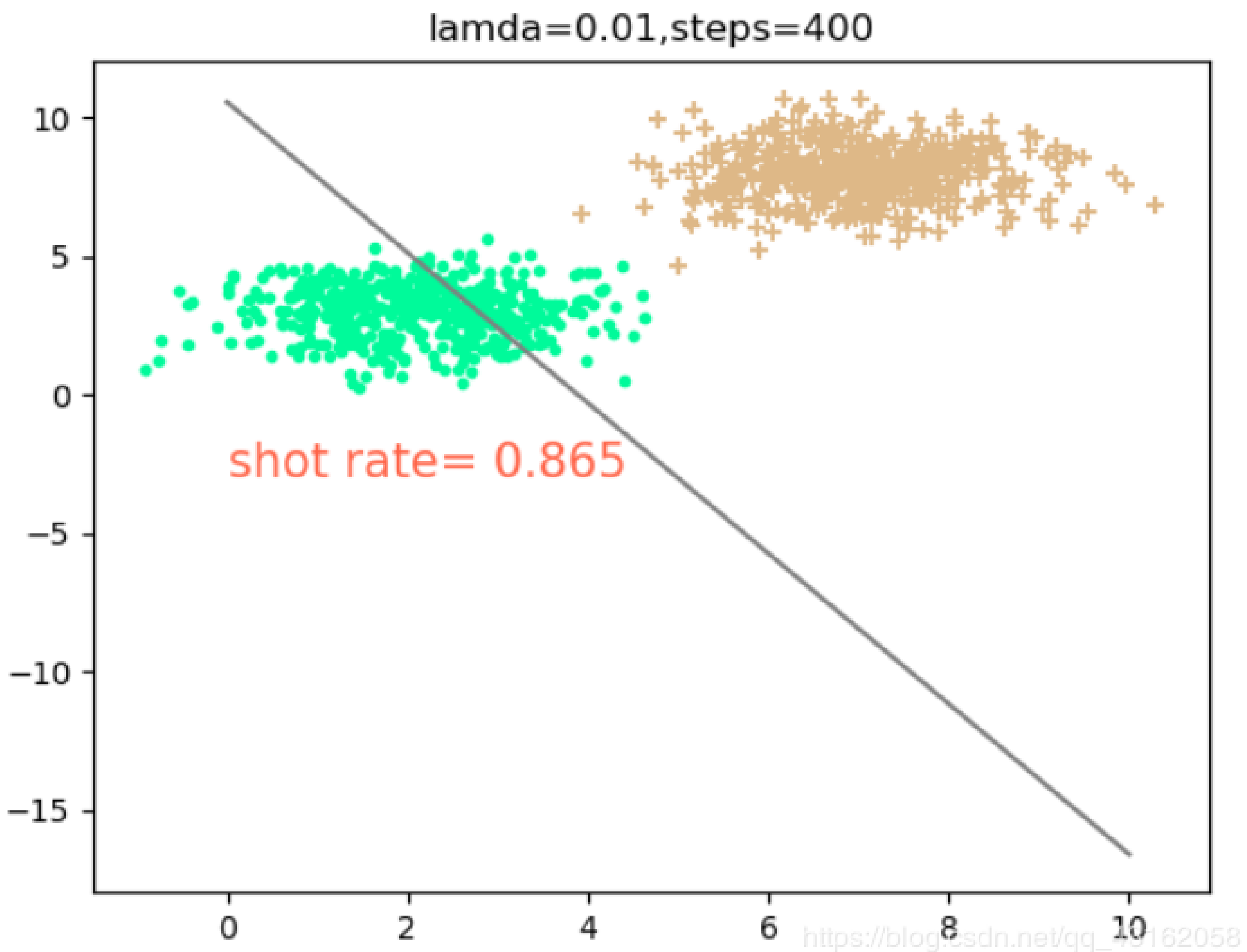
四、实验结果与分析

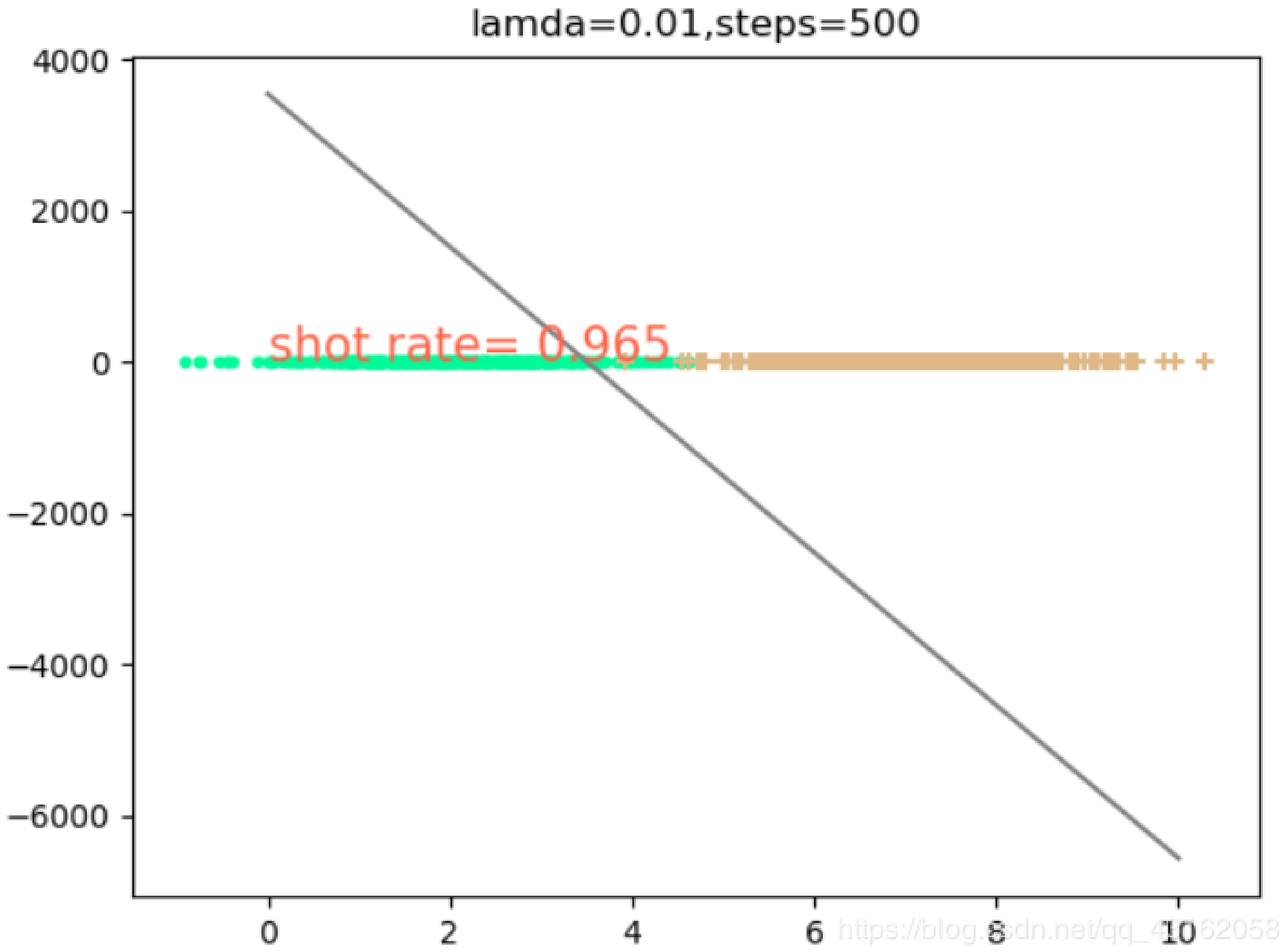
4.1 讨论不同的学习率下，需要的随机梯度下降次数


lamda=0.01,steps=100
10
8
6
4
2
0
−2
−4
shot rate= 0.718
0
2
4
6
8
10
https://blog.csdn.net/qq_41162058


lamda=0.01,steps=200
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
shot rate= 0.968
0
2
4
6
8
10
https://blog.csdn.net/qq_41162058

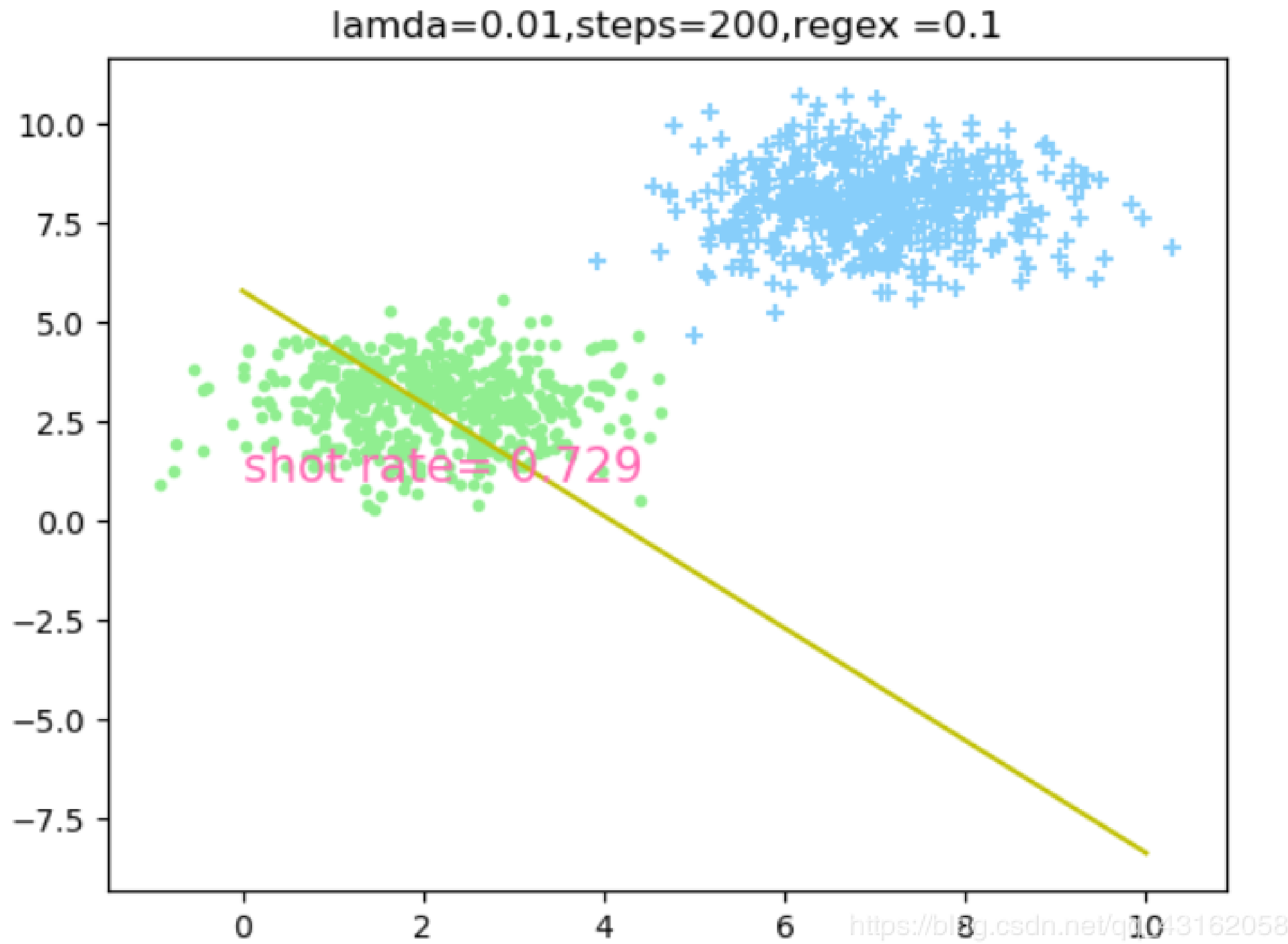

lamda=0.01,steps=300
shot rate= 0.815
https://blog.csdn.net/qq_43162058

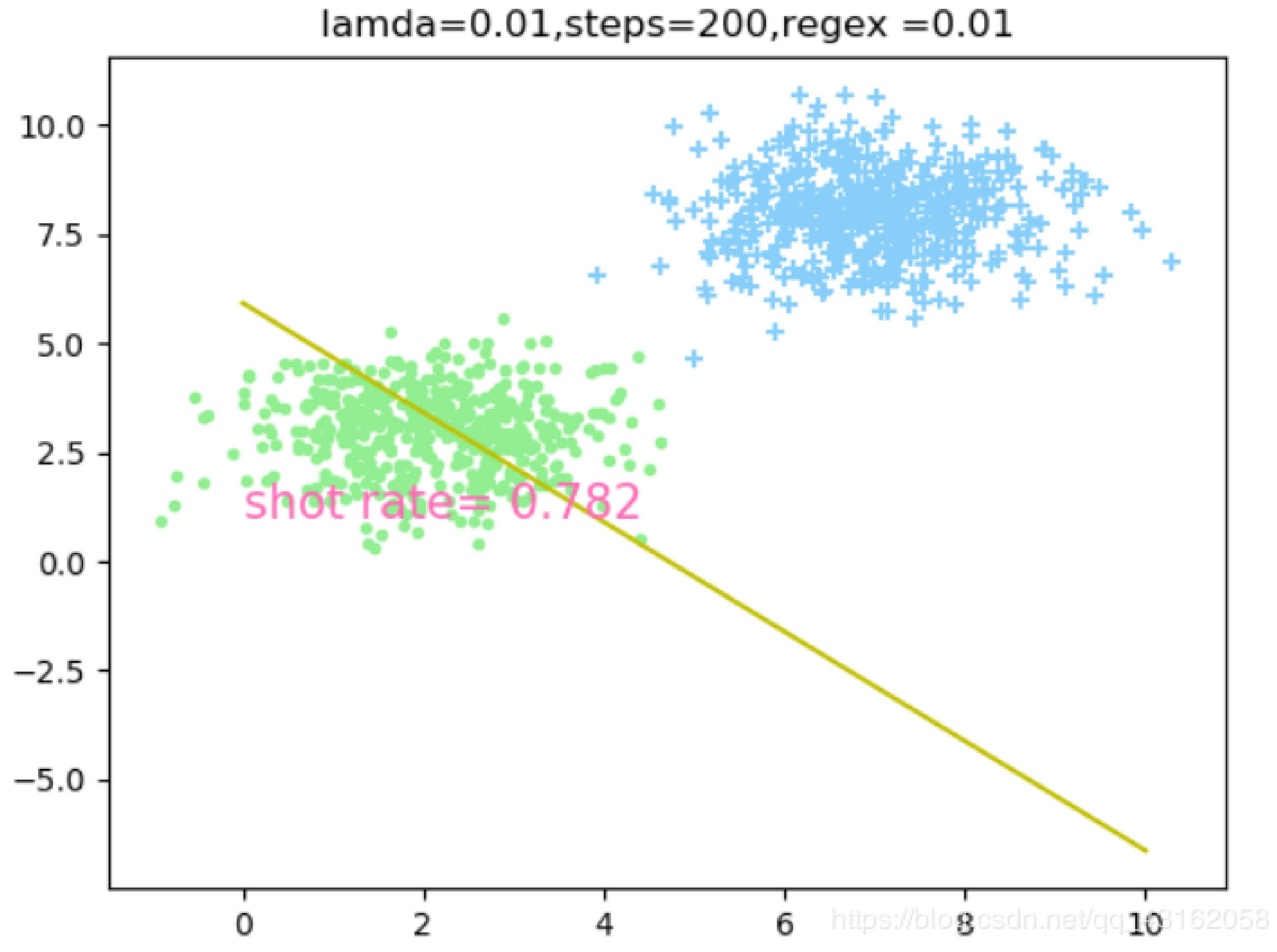

lamda=0.01,steps=400
shot rate= 0.865
https://blog.csdn.net/qq_43162058

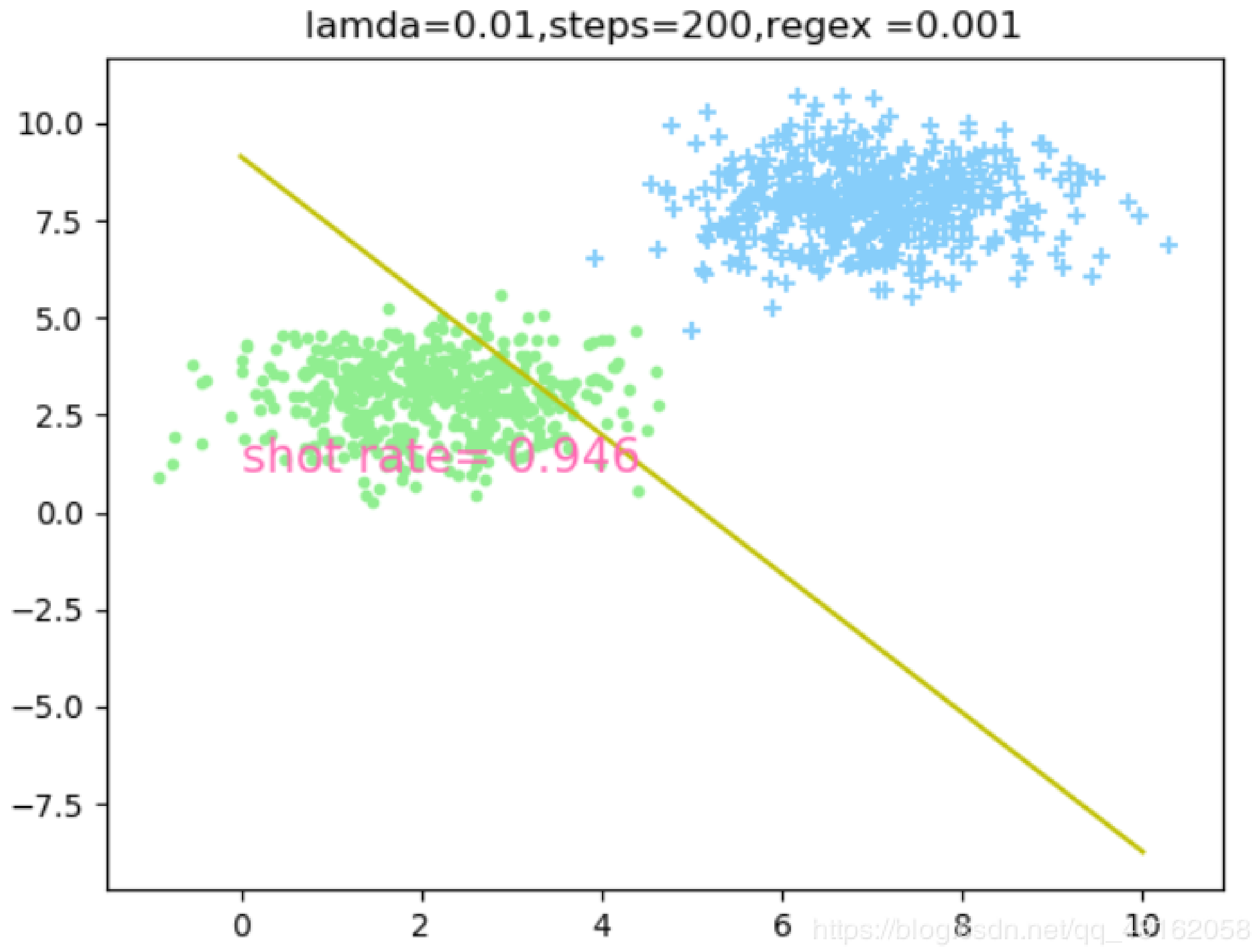


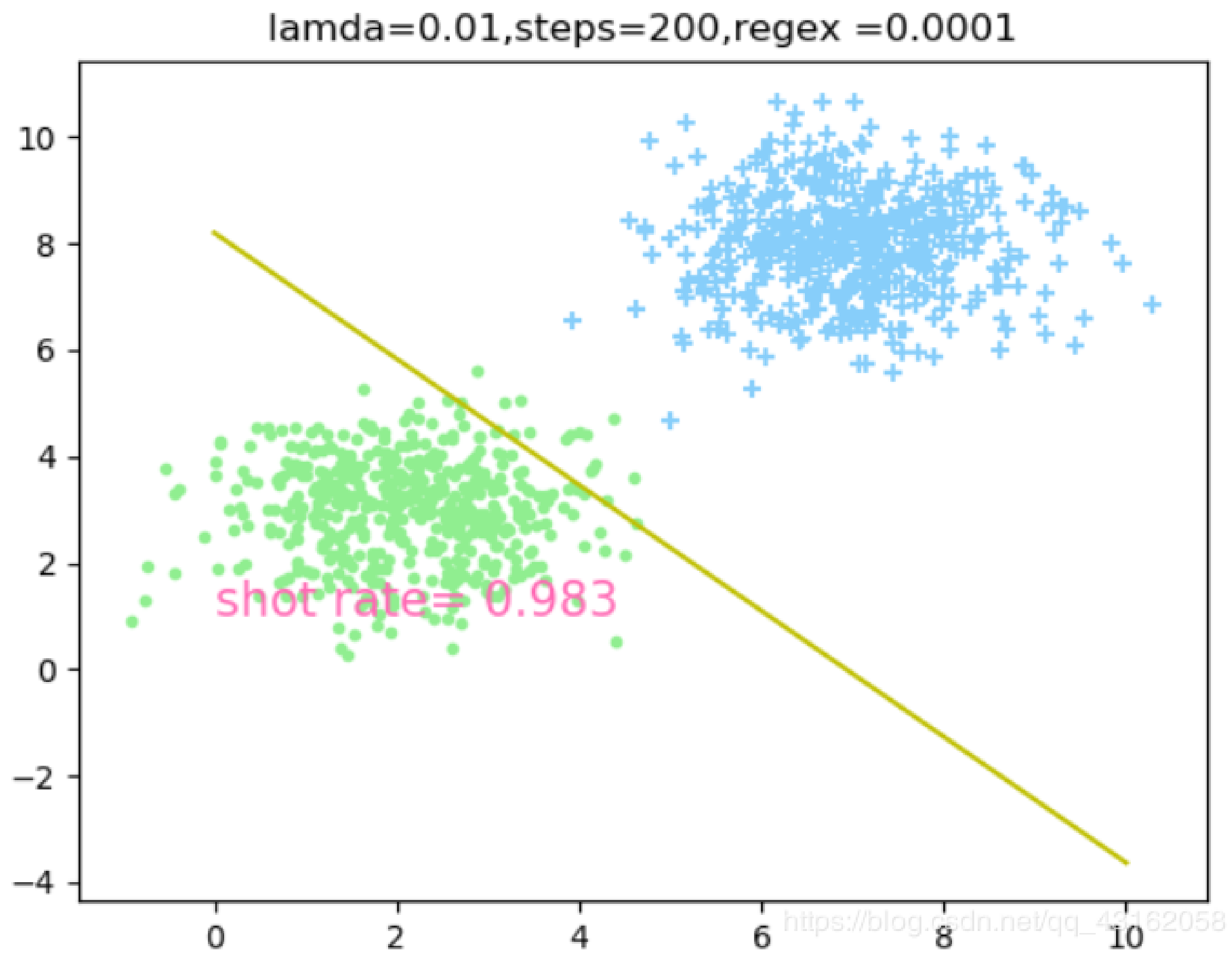
可以得出，当lamda=0.01时，学习率比较大（这里不讨论过拟合），因此所需迭代的步数比较小，大约在100-200次能够拟合得比较好，当大于200次以后出现震荡现象，反而使得正确率下降。但是随机梯度下降法由于每次选择样本是随机的，所以不一定每次分类效果都很好。

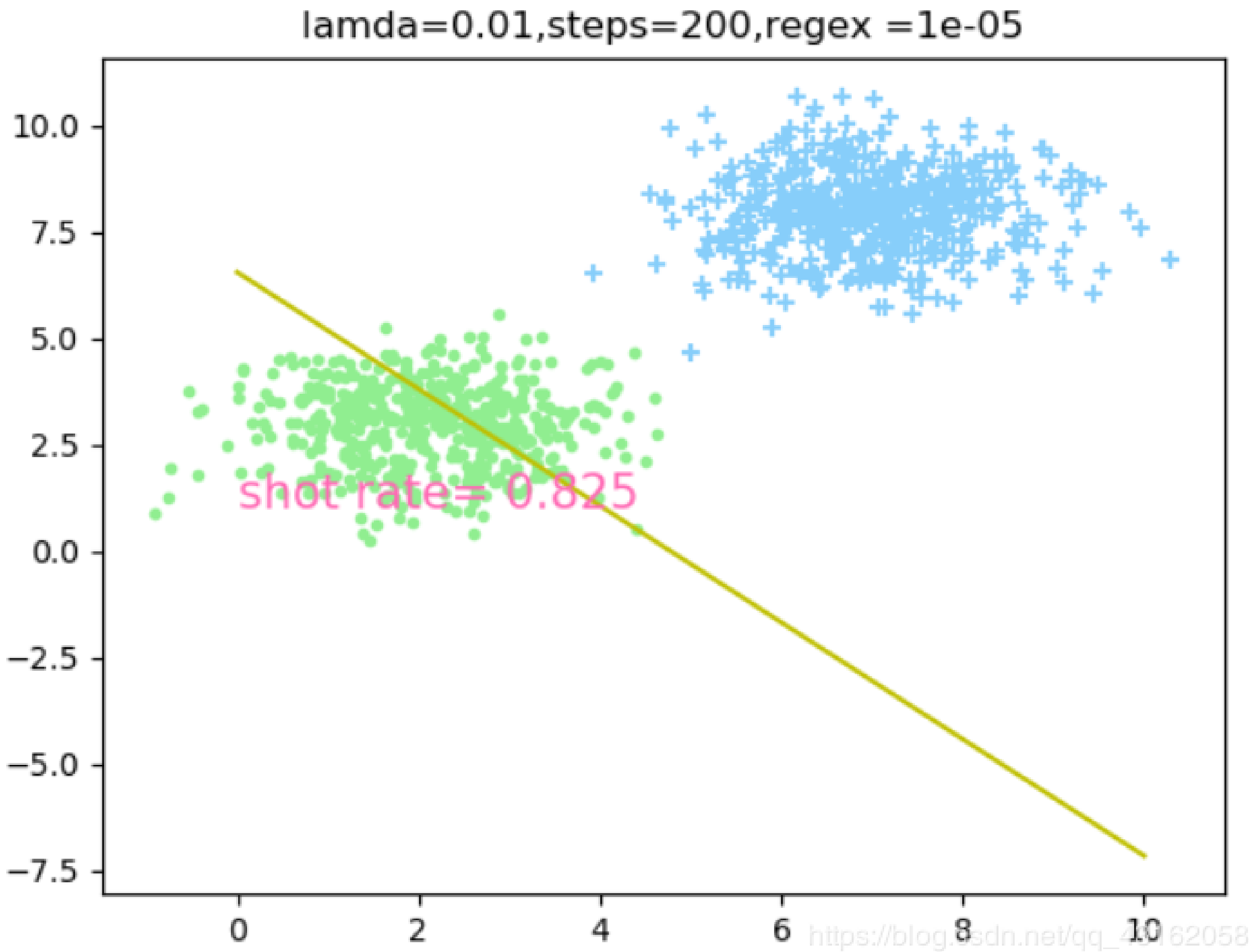
4.2 相同步数和学习率，讨论不同正则因子对分类效果的影响


lamda=0.01,steps=200,regex =0.1
shot rate= 0.729
https://blog.csdn.net/qq_43162058


lamda=0.01,steps=200,regex =0.01
shot rate= 0.782
https://blog.csdn.net/qq_43162058


lamda=0.01,steps=200,regex =0.001
shot rate= 0.946
https://blog.csdn.net/qq_40162058

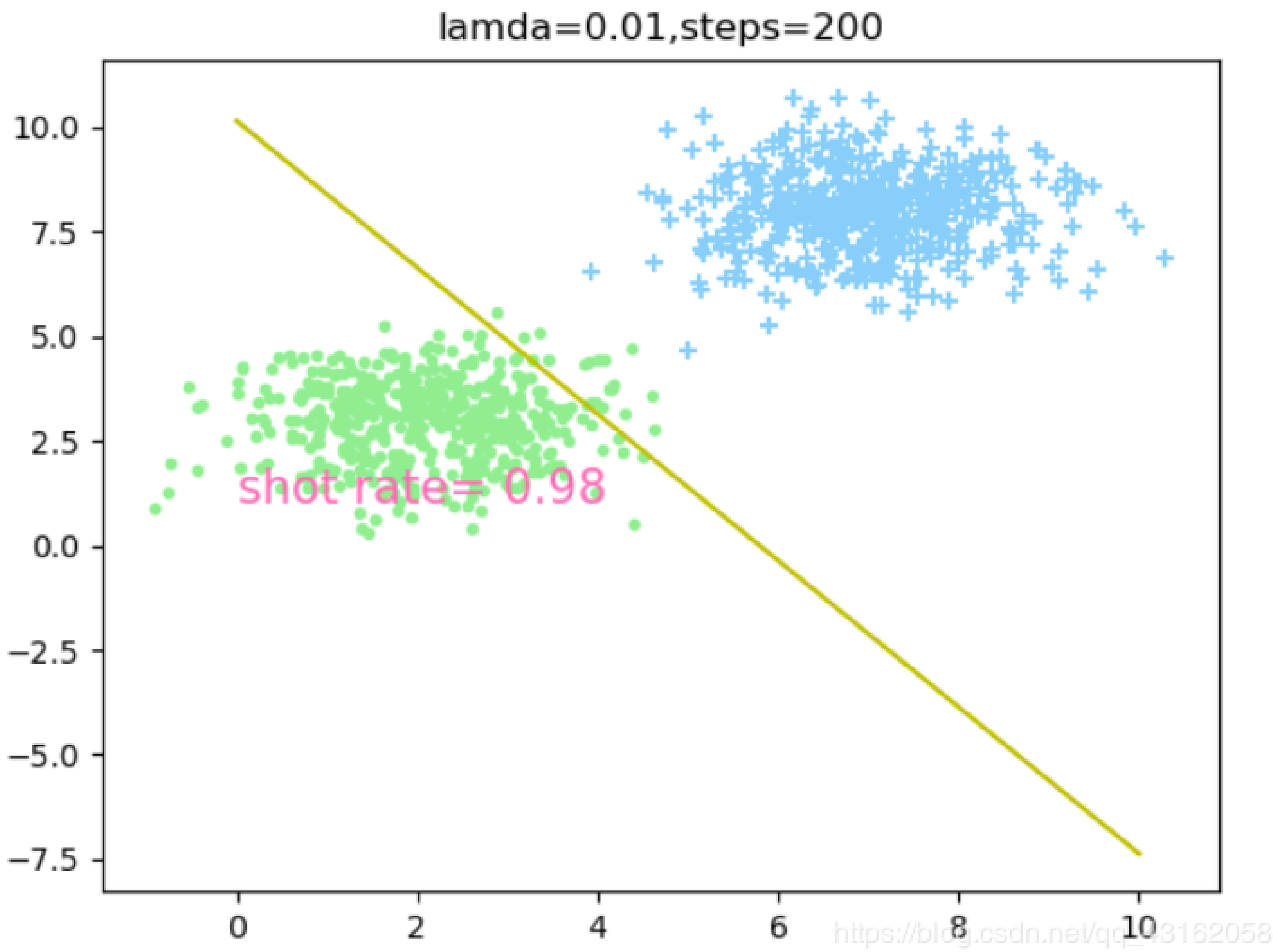

lamda=0.01,steps=200,regex =0.0001
shot rate= 0.983
https://blog.csdn.net/qq_40162058



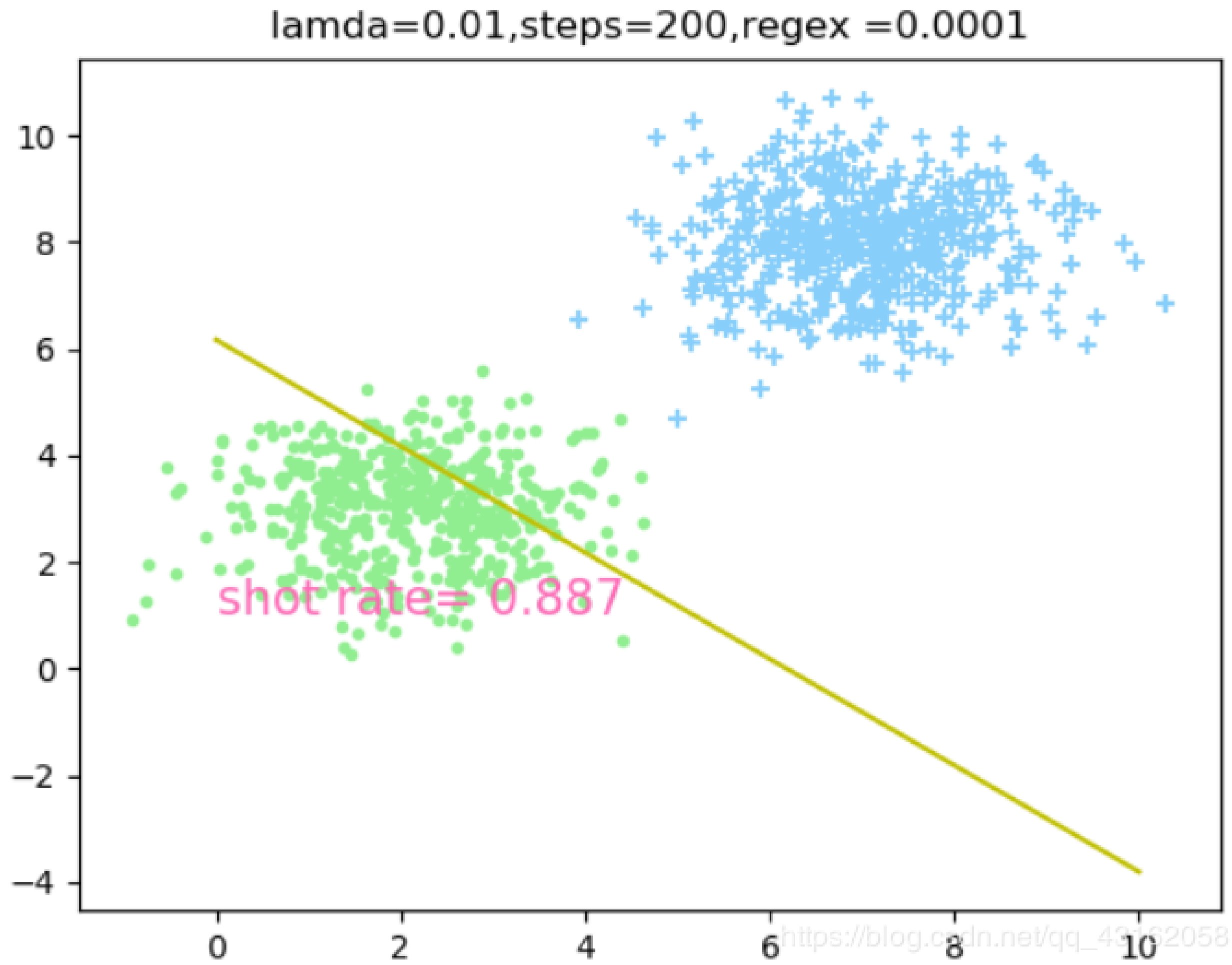
由此可以得出当lamda = 0.01,steps = 200时，正则因子regex = 0.0001时分类效果最好。

4.3生成数据验证

不加正则项



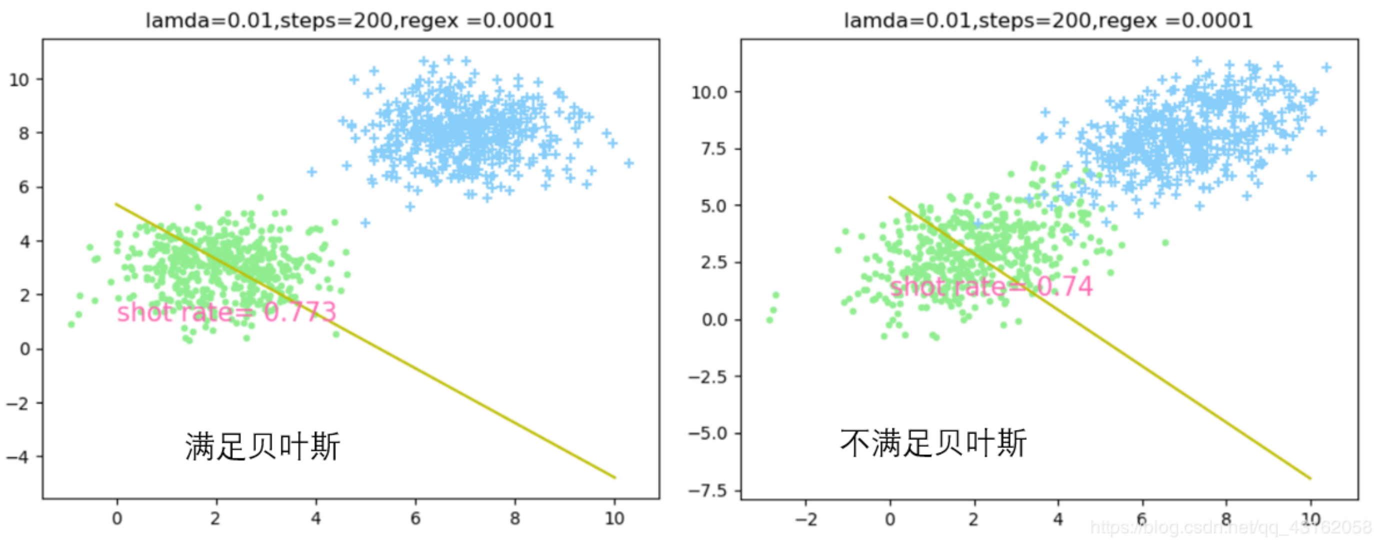
加入正则项



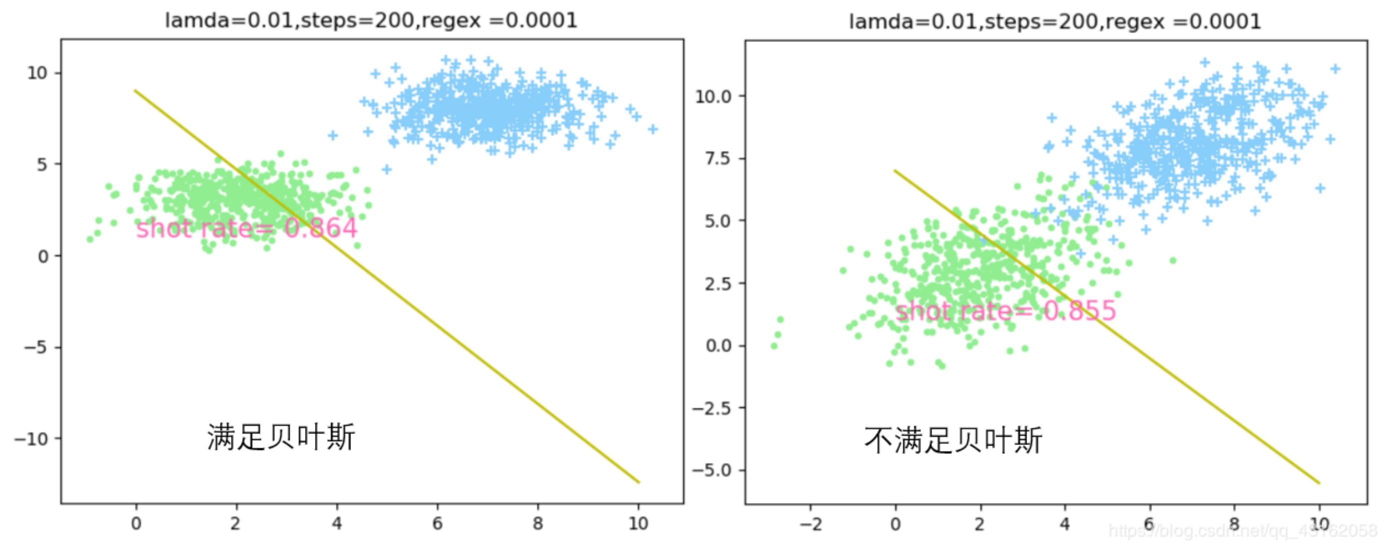
使用随机梯度下降法优化，由于样本选择是随机的，因此每次的正确率不一定相同，但大多数时候正确率都超过0.85，有时候甚至能到达0.95以上。

4.4 数据满足贝叶斯假设和不满足贝叶斯假设进行对比

4.4.1不加正则项



4.4.2 加入正则项



由图片可知，无论加不加正则项，满足贝叶斯假设与否对逻辑回归的正确率的影响不大，但是满足贝叶斯假设的数据分类效果略胜一筹。

4.5 使用UCI数据集测试

在UCI上寻找了一个判断是否会税务欺诈的数据集，筛选特征后只剩四维数据，使用正则和非正则观察分类效果

- 不加正则

  ```
  0.7811244979919679
  ```

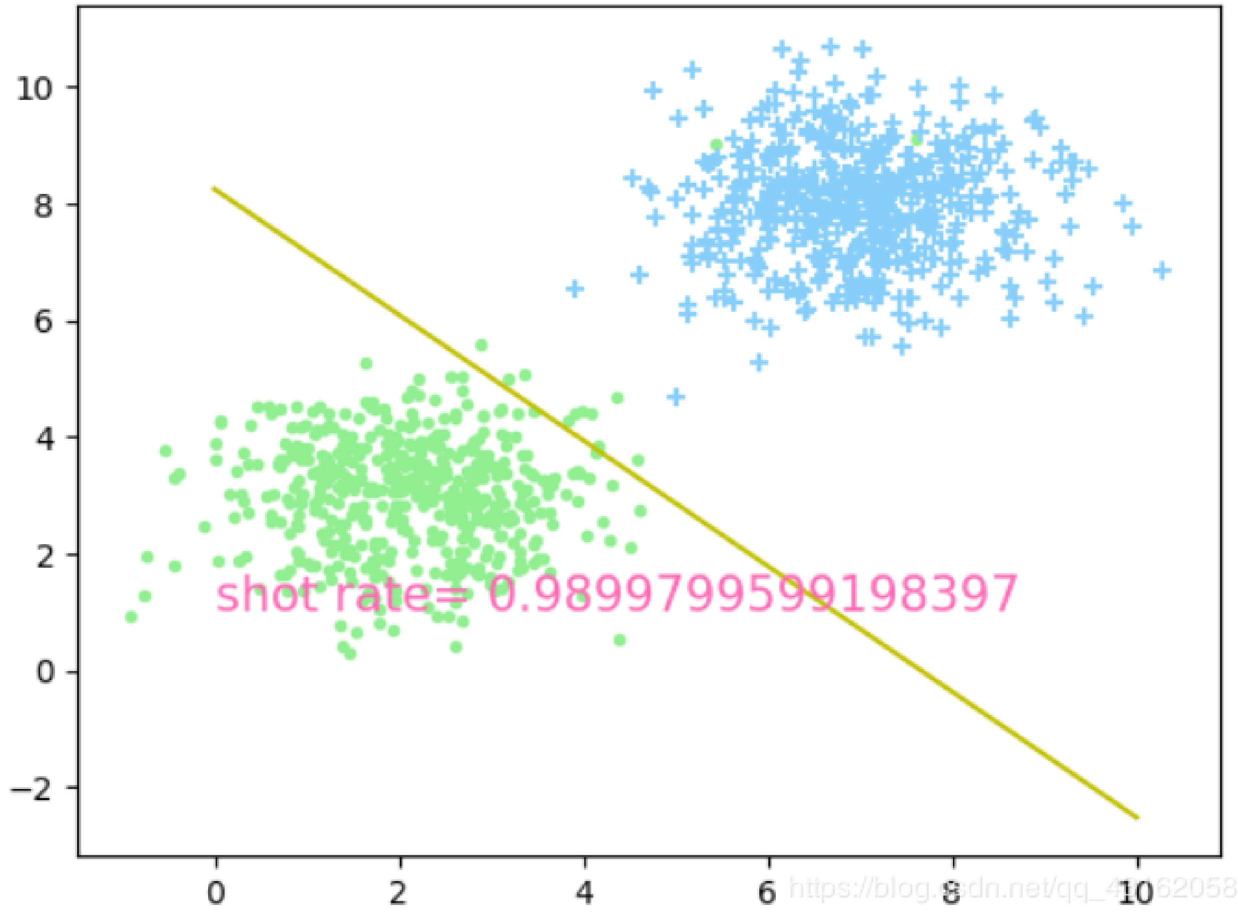
- 加正则

  ```
  0.8453815261044176
  ```

  由于随机梯度下降的不确定性，加正则与不加正则的区别不方便观察。

4.6 牛顿法优化

牛顿法相比梯度下降法，需要的迭代次数更少，而且更容易得到最优解，因此正确率很容易超过0.98，有时候甚至可以达到1.0



4.7 在实验过程中发现的问题

1. 实验中sigmoid函数很容易溢出，可以对自变量进行讨论，如果自变量z的值小于20，则认为sigmoid为0.
2. 计算加入正则因子的ω时，如果数据量过大（约1000以上），而且采用float32来进行计算，会因为精度丢失而无法成功分类。解决方法有两种，一是将loss乘以数据量，这样可以避免精度丢失，但是会有上溢出的风险；而是所有相关数据均使用float64类型，但是float64占用内存较大，运行速度相对缓慢。本实验中我使用的数据可以直接采用乘数据量解决，因此使用的这个办法解决了精度丢失的问题。

五、结论

1. 关于惩罚项: 对于逻辑回归而言，带正则项和不带正则项的差别没有多项式拟合函数那么大。尤其是当使用随机梯度下降法时，由于随机梯度下降法选择样本的不确定性，在相同迭代次数和相同参数条件下，基本无法看出显著差异。但是使用牛顿法进行优化时，由于比较容易找到最小值，所以如果不加正则项会发生过拟合。
2. 关于牛顿法：牛顿法每次迭代的时间代价为O($N\cdot|w|^2$）,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。
3. 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
4. 关于sigmoid函数，sigmoid函数可能会发生溢出，主要是当z<<0时时e^z>>0,会发生溢出。

六、参考文献

【统计学习方法】李航
【机器学习】周志华

七、源代码（含注释）

```
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
import random

"""
by xiaoyi
"""



class Logistic:
    matrix = ()  # 读入数据矩阵
    test_matrix = ()  # 测试数据矩阵
    y = ()  # 分类情况，y[i]表示第i组数据的分类情况
    test_y = ()  # 测试数据集的分类情况
    x = ()  # 特征矩阵，其中x[i]表示第i个实例的特征取值情况,最后一维为1
    test_x = ()  # 测试数据集的特征矩阵
    w = ()  # 对应扩充特征后的w
    n = 0  # 特征数的个数，其中w是n+1维的
    dataSum = 0  # 数据量
    testSum = 0  # 测试数据集大小

    # sigmoid函数
    @staticmethod
    def sig(wx):
        if wx < -10:
            return 0
        else:
            return 1 / (1 + math.exp(-wx))

    # 计算对数似然的值，不加正则，梯度上升法，没有加负号
    def cal_loss1(self):
        loss = 0
        for i in range(self.dataSum):
            w_multi_x = np.dot(self.x[i], self.w)
            # print(w_multi_x)
            loss -= np.dot(self.y[i], w_multi_x)
            # 防止溢出，所以对wx进行讨论
            if w_multi_x > 0:
                loss += w_multi_x + math.log(1 + math.exp(-w_multi_x))
            else:
                loss += math.log(1 + math.exp(w_multi_x))
        return loss

    # 计算损失函数的值，加正则，梯度下降法，加负号
    def cal_loss2(self, regex):
        loss = 0
```

```
        for i in range(self.dataSum):
            # print(self.x[i])
            w_multi_x = np.dot(np.mat(self.x[i]), self.w)
            # print(w_multi_x)
            loss -= np.dot(self.y[i], w_multi_x)
            # 防止溢出，所以对wx进行讨论
            if w_multi_x > 0:
                loss += w_multi_x + math.log(1 + math.exp(-w_multi_x))
            else:
                loss += math.log(1 + math.exp(w_multi_x))
        loss += regex * np.dot(self.w.T, self.w)[0, 0]
        # loss /= self.dataSum
        return loss
```

下降法

```
    def cal_gradient1(self):
        gradient = np.zeros((self.n + 1, 1))
        i = random.randint(0, self.dataSum - 1)
        wx = np.dot(np.mat(self.x[i]), self.w)
        for j in range(self.n + 1):
            gradient[j][0] += self.x[i][j] * (-self.y[i] + Logistic.sig(wx))
        return gradient

    # 计算梯度，带正则，损失函数的梯度
    def cal_gradient2(self, regex):
        gradient = np.zeros((self.n + 1, 1))
        i = random.randint(0, self.dataSum - 1)
        wx = np.dot(np.mat(self.x[i]), self.w)
        for j in range(self.n + 1):
            gradient[j][0] += self.x[i][j] * (-self.y[i] + Logistic.sig(wx))
        gradient += regex * self.w
        # print(gradient)
        # gradient /= self.dataSum
        # print(gradient)
        return gradient

    # 使用梯度下降法优化参数，似然函数，不带正则
    def de_gradient1(self, lamda, door):
        # print(self.w)
        loss0 = self.cal_loss1()
        g0 = self.cal_gradient1()
        w0 = self.w
        self.w -= lamda * g0
        loss1 = self.cal_loss1()
        cnt = 0
        while cnt < door:
            cnt += 1
            loss0 = loss1
            g0 = self.cal_gradient1()
            w0 = self.w
            self.w -= lamda * g0
            loss1 = self.cal_loss1()
            # print(loss0 - loss1)
        self.w = w0
        # print(self.w)
```

```python
        # 返回损失函数的值
        return loss0

    # 使用梯度下降法求解带正则项的w
    def de_gradient2(self, lamda, door, regex):
        loss0 = self.cal_loss2(regex)
        g0 = self.cal_gradient2(regex)
        w0 = self.w
        self.w -= lamda * g0
        loss1 = self.cal_loss2(regex)
        cnt = 0
        while cnt < door:
            # print(loss1 - loss0)
            # print(g0)
            cnt += 1
            loss0 = loss1
            g0 = self.cal_gradient2(regex)
            w0 = self.w
            self.w -= lamda * g0
            loss1 = self.cal_loss2(regex)
        self.w = w0
        # 返回损失函数的值
        return loss0

    # 计算黑塞矩阵
    def hessian(self):
        he = np.zeros((self.n + 1, self.n + 1))
        for i in range(self.dataSum):
            w_multi_x = np.dot(np.mat(self.x[i]), self.w)
            # print(w_multi_x)
            for j in range(self.n + 1):
                for k in range(self.n + 1):
                    if w_multi_x > 20:
                        he[j][k] -= 0
                    else:
                        p = Logistic.sig(w_multi_x)
                        he[j][k] += self.x[i][j] * self.x[i][k] * p * (1 - p)
        return he

    # 牛顿法
    def newton(self, steps):
        cnt = 0
        w0 = self.w
        while cnt < steps:
            cnt += 1
            g = self.cal_gradient1()
            # print(g)
            he = self.hessian()
            # print(np.linalg.inv(he))
            w0 = self.w
            # print(self.w)
            self.w -= np.dot(np.linalg.inv(he), g)
        self.w = w0

    # 读取训练集
```

```python
    def read_data(self, file):
        self.matrix = pd.read_csv(file, header=1).values
        # print(self.matrix)
        # with open(file) as f:
        #     self.matrix = np.loadtxt(f, float, delimiter=",")
        self.dataSum = len(self.matrix)
        self.n = len(self.matrix[0]) - 1
        add = np.ones((self.dataSum, 1))
        self.x = np.hstack((self.matrix[:, :self.n], add))
        # print(self.x)
        self.y = self.matrix[:, self.n]
        self.w = np.ones((self.n + 1, 1))

    # 读取测试集
    def read_test_data(self, file):
        self.test_matrix = pd.read_csv(file, header=1).values
        # with open(file) as f:
        #     self.test_matrix = np.loadtxt(f, float, delimiter=',')
        self.testSum = len(self.test_matrix)
        self.test_x = np.hstack((self.test_matrix[:, :self.n], np.ones((self.testSum, 1))))
        self.test_y = self.test_matrix[:, self.n]

    # 预测
    def pre_test(self):
        cnt = 0
        for i in range(self.testSum):
            pre_wx = np.dot(np.mat(self.test_x[i]), self.w)
            # print(pre_wx)
            if (pre_wx >= 0) and (self.test_y[i] == 1):
                cnt += 1
            elif (pre_wx <= 0) and (self.test_y[i] == 0):
                cnt += 1
        return cnt / self.testSum


def test_model():
    # 测试模型
    test = Logistic()
    train_set = "gauss.csv"
    test_set = "test_gauss.csv"
    test.read_data(train_set)
    lamda = 1e-2
    steps = 10
    regex = 1e-3
    # test.de_gradient2(lamda, steps, regex)
    # test.de_gradient1(lamda, steps)
    test.newton(steps)
    test.read_test_data(test_set)
    correct = test.pre_test()
    print(correct)
    x0 = test.test_matrix[:500, 0]
    y0 = test.test_matrix[:500, 1]
    x1 = test.test_matrix[500:, 0]
    y1 = test.test_matrix[500:, 1]
    plt.scatter(x0, y0, marker='.', color='lightgreen')
```

```python
    plt.scatter(x1, y1, marker='+', color='lightskyblue')
    dx = np.linspace(0, 10, 100)
    dy = (-test.w[2][0] - test.w[0][0] * dx) / test.w[1][0]
    # plt.title("lamda=" + str(lamda) + ",steps=" + str(steps)+",regex ="+str(regex))
    # plt.title("lamda=" + str(lamda) + ",steps=" + str(steps))
    plt.plot(dx, dy, color='y')
    ans = "shot rate= " + str(correct)
    plt.text(0, 1, ans, color='hotpink', fontsize=15)
    plt.show()



def generate_data():
    # 生成高斯数据
    f = open('test_gauss_not_bayes.csv', 'w')
    mean0 = [2, 3]
    cov = np.mat([[2, 1], [1, 2]])
    x0 = np.random.multivariate_normal(mean0, cov, 500).T

    mean1 = [7, 8]
    x1 = np.random.multivariate_normal(mean1, cov, 500).T

    for i in range(len(x0.T)):
        line = []
        line.append(x0[0][i])
        line.append(x0[1][i])
        line.append(1)
        line = ",".join(str(i) for i in line)
        line = line + "\n"
        f.write(line)

    for i in range(len(x0.T)):
        line = []
        line.append(x1[0][i])
        line.append(x1[1][i])
        line.append(0)
        line = ",".join(str(i) for i in line)
        line += "\n"
        f.write(line)
    f.close()


test_model()
```