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： 专业选修

实验题目： 逻辑回归

学号：120L030501
姓名： 张明远

一、实验目的

目标：

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

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。

环境：

python3.10

三、设计思想（本程序中的用到的主要算法及数据结构）

3.1 生成数据

本实验要求对二分类数据进行逻辑回归。利用多维高斯分布函数来生成数据，为便于画图展示，主要使用二维数据。首先生成朴素贝叶斯数据，在这里选取中心点为(0,0)、(3,3)。

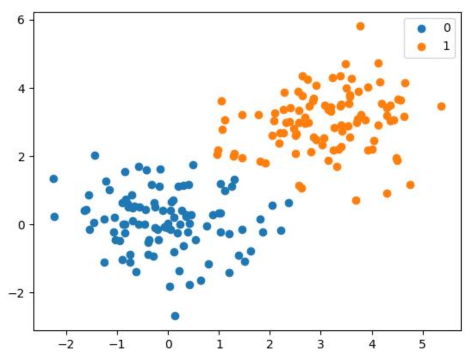
若满足朴素贝叶斯，则认为 X 的各维度数据关于 Y 条件独立，则协方差矩阵为

Cov = [[cov11,cov12][cov21,cov22]]

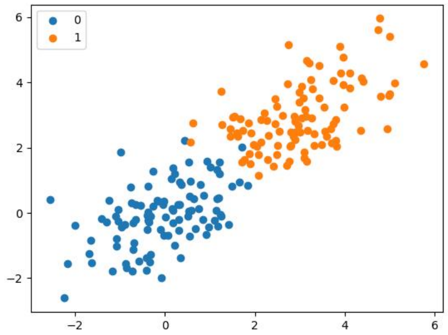
其中 cov12 = cov21 = 0

cov11 就是 X1 的方差， cov22 就是 X2 的方差，

根据协方差矩阵添加高斯噪声。先生成样本点两类生成数据如下：



然后生成 X 的各个维度有些相关的数据，即令 cov12=cov21!=0,生成结果如下，可见数据明显变形



可见通过噪声处理样本点使得数据相对中心点有了不同程度的偏移，较为符合现实中的分布情况，接下来以这两份数据进行实验。

代码核心部分如下：

```
def generatePoints(center, rand_normal_x, rand_normal_y, nums, naive_rate):
    x = center[0]
    y = center[1]
    cov = [[rand_normal_x, naive_rate], [naive_rate, rand_normal_y]]
    xs = np.random.multivariate_normal((x, y), cov, nums)
    return xs
```

3.2 逻辑回归解决二分类问题

逻辑回归的主要任务是从训练集中的各个训练数据的各个维度 X=(x1,x2,x3,...,xn)中学习到一个分类器 f:X->Y,以便于预测一个新的样本的 label。

为了建立一个相对简单的模型，我们做了几个简化问题的假设：

①X 的每一维属性 X_i 都是实数，故 X 可视为形如＜X1，X2...Xn＞的 n 维 vector

②Y 是 boolean 值，取值为 1 或 0

③X_i 关于 Y 条件独立

④P（Xi ｜ Y=yk）~ N（μ，σ）

⑤P(Y)~ B(π)
由这个假设，我们可以推得：

$$P(Y=0 \mid X)=\frac{1}{1+\exp\left(\ln \frac{\pi}{1-\pi}+\sum_i \ln \frac{P(X||Y=1)}{P(X|Y=0)}\right)}$$

$$P(Y=1|X)=\frac{1}{1+\exp\left(\ln \frac{1-\pi}{\pi}+\sum_i \ln \frac{P\left(X_{|}|Y=0\right)}{P\left(X_{|}|Y=1\right)}\right)}$$

代换相关参数可得

$$\mathrm{P}(\mathrm{Y}=1 \mid \mathrm{X})=\frac{1}{1+\exp(\mathrm{w}^{\mathrm{T}}\mathrm{X})}=\operatorname{sigmoid}\left(-\mathrm{w}^{\mathrm{T}}\mathrm{X}\right)$$

$$\mathrm{P}(\mathrm{Y}=0 \mid \mathrm{X})=\frac{\exp\left(\mathrm{w}^{\mathrm{T}}\mathrm{X}\right)}{1+\exp(\mathrm{w}^{\mathrm{T}}\mathrm{X})}=\frac{1}{1+\exp(-\mathrm{w}^{\mathrm{T}}\mathrm{X})}=\operatorname{sigmoid}\left(\mathrm{w}^{\mathrm{T}}\mathrm{X}\right)$$

即逻辑回归的形式。利用 odds 的概念，我们还可以得出两类数据的分界为 W.T*X = 0 这个超平面。接下来就是利用一定的方法优化方程的值，从而得出最佳的 w 参数。

在这里有两种方法：最大似然估计 MLE 和贝叶斯估计 MAP，两者的 loss 函数里面就分别代表了无正则项的 loss 函数和有正则项的 loss 函数。

MLE 的核心思想就是：将参数 w 看作唯一真值，我们的任务就是找到这个 w，使得在这组参数下，我们的数据的似然度（概率）最大。

根据这一思想，我们得到 MLE 的 loss 函数为：

$$\operatorname{loss}(\mathrm{w})=\sum_1\left(-\mathrm{Y}^1\mathrm{w}^{\mathrm{T}}\mathrm{X}^1+\ln\left(1+\exp\left(\mathrm{w}^{\mathrm{T}}\mathrm{X}^1\right)\right)\right)$$

MAP 的核心思想是： w w w 是一个随机变量，符合一定的概率分布。

所以我们的任务就是给 w w w 添加一个先验 P (w) P(w) P(w)，然后使得 P (w) P (Y | X , w) P(w)P(Y|X,w) P(w)P(Y|X,w)最大。

而 MAP 的结果等价于在 MLE 的结果上加上正则项。

$$\mathrm{loss}(\mathrm{w})=\sum_{1}\left(-\mathrm{Y}^{1}\mathrm{w}^{\mathrm{T}}\mathrm{X}^{1}+\ln\left(1+\exp\left(\mathrm{w}^{\mathrm{T}}\mathrm{X}^{\mathrm{l}}\right)\right)\right)+\frac{\lambda}{2}\mathrm{w}^{\mathrm{T}}\mathrm{w}$$

loss 函数的计算代码如下：

```
def getLoss(x, y, w):
 loss = 0
 for i in range(x.shape[1]):
     wx = w @ x[i]
     loss_single                                                     =
y[i]*np.log(sigmoid(wx))+(1-y[i])*np.log(1-sigmoid(wx))
     loss -= loss_single
 return loss
```

有了 loss 函数，我们只需要使用优化方法求解 loss 函数最小时的 w 即可。在这里我们使用梯度下降法求解。loss 函数对 w 求梯度如下：

$$\frac{\partial 1(\ \mathrm{W})}{\partial \mathrm{w_i}}=\sum\nolimits_{1}\mathrm{X}_{\mathrm{i}}^{\mathrm{l}}\left(\mathrm{Y}^{1}-\frac{1}{1+\exp\left(\mathrm{w}_0+\sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{w_i}\mathrm{X}_{\mathrm{i}}^{1}\right)}\right)$$

$$\mathrm{w_i}=\mathrm{w_i}-\eta\sum\nolimits_{1}\mathrm{X}_{\mathrm{i}}^{1}\left(\mathrm{Y}^{1}-\frac{1}{1+\exp\left(\mathrm{w}_0+\sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{w_i}\mathrm{X}_{\mathrm{i}}^{1}\right)}\right)$$

在加入了正则项修正后则为：

$$\frac{\partial 1(\ \mathrm{W})}{\partial \mathrm{w_i}}=\sum_{1}\mathrm{X}_{\mathrm{i}}^{\mathrm{l}}\left(\mathrm{Y}^{1}-\frac{1}{1+\exp\left(\mathrm{w}_0+\sum_{\mathrm{i}=1}^{\mathrm{n}}\mathrm{w_i}\mathrm{X}_{\mathrm{i}}^{1}\right)}\right)+\lambda I$$

梯度下降的核心代码如下：

```
def getW():
    x, y = getXY()
    # 随机生成初始矩阵
    w = np.array([1.0 for i in range(3)])
    # 计算初始 loss
    old_loss = loss = getLoss(x, y, w)
    times = 0  # 迭代次数
    # 正则化参数
    lambda_analytical = 0
    while True:
        delta_w = np.array([0.0 for i in range(3)])
        for i in range(x.shape[0]):
            wx = w @ x[i]
            d = x[i] * (y[i] - sigmoid(wx))
            delta_w += d
        w += learning_rate * delta_w / x.shape[1] -lambda_analytical
* w
        loss = getLoss(x, y, w)
        times += 1
        print('epoch {}: Loss = {}'.format(times, loss))
        if abs(old_loss - loss) < 10 ** -8:
            break
        old_loss = loss
    print('Training    completed.\nFinal    Loss    =    {}\nw    =
{}'.format(loss, w))
    return w
```

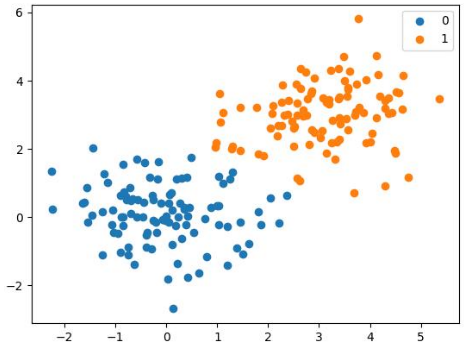
四、实验结果与分析

4.1 使用数据说明

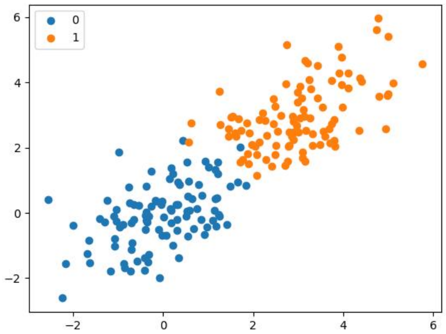
朴素贝叶斯数据：

第一类：中心点（0，0），噪声均值皆为 0，标准差 cov11=cov22=1 的 100 个点

第二类：中心点（3，3），噪声均值皆为 0，标准差 cov11=cov22=1 的 100 个点



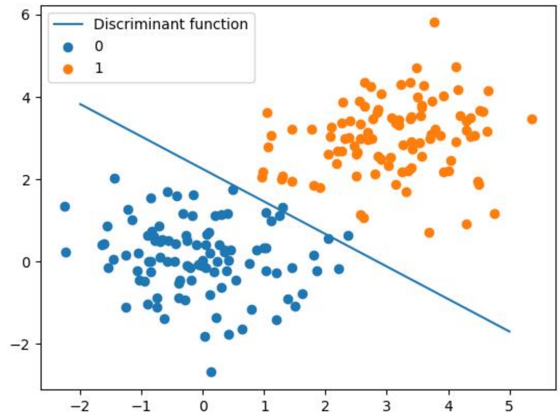
非朴素贝叶斯数据：
在朴素贝叶斯数据上令 cov12=cov21=0.2



4.2 满足朴素贝叶斯的数据分类

4.2.1 无正则项

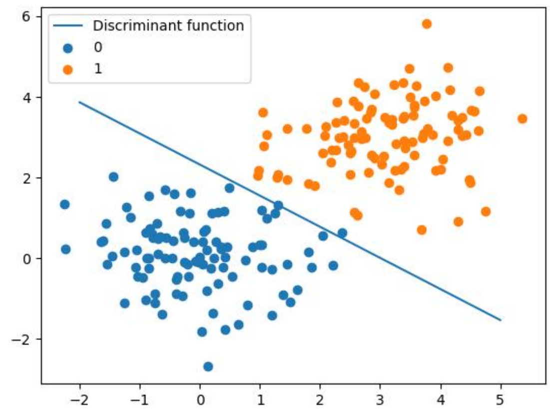
```
Discriminant function: 8.71x1+11.04x2-24.81=0
Final Loss = 7.178687114977454e-09
precision = 0.99
```



4.2.2 有正则项

lambda = 0.1 时

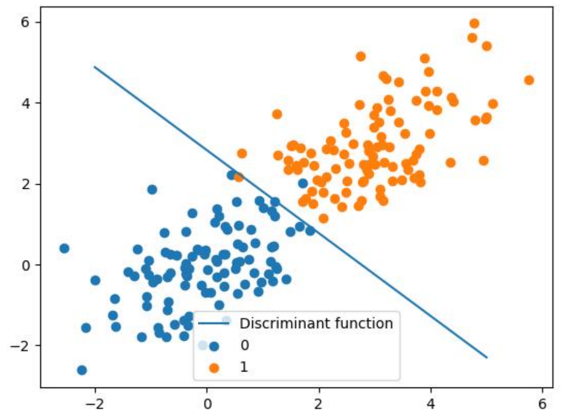
```
Discriminant function：1.23x1+1.59x2-3.70=0
Final Loss = 0.08012525027932554
precision = 0.99
```



4.3 不满足朴素贝叶斯的数据分类

4.3.1 无正则项

```
Discriminant function：8.13x1+7.93x2-22.43=0
Final Loss =  3.117024531064788e-09
precision = 0.99
```



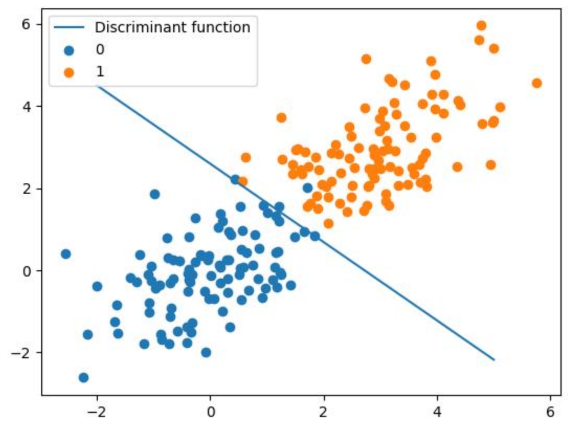
4.3.2 有正则项

lambda = 0.1 时

Discriminant function：1.37x1+1.43x2-3.72=0

Final Loss = 0.07301754719468272

precision = 0.98



4.4 分析

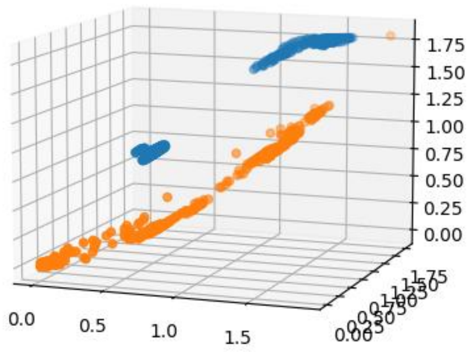
通过以上 4 种不同条件下的测试可以看出，逻辑回归分类器在在满足朴素贝叶斯假设时分类良好，在不满足朴素贝叶斯假设时分类效果也与满足时相差并不大，可能是由于数据维度只有二维，不会发生较大的偏差。并且在二维条件下，是否有惩罚项对其影响也不大，但当减小训练集时，所得到的判别函数确实存在过拟合现象，加入正则项可以预防此现象的发生。

4.5UCI 数据集测试

UCI 数据集是一个常用的机器学习标准测试数据集，是加州大学欧文分校(University of CaliforniaIrvine)提出的用于机器学习的数据库。机器学习算法的测试大多采用的便是 UCI 数据集了,其重要之处在于“标准”二字,新编的机器学习程序可以采用 UCI 数据集进行测试,类似的机器学习算法也可以一较高下。

本次实验选用 UCI 中的 Skin Segmentation Data Set，它是一个二分类数据集，且 x 只含有三个维度，适合可视化处理。数据集中一共有 245057 条数据，从中随机选取两种类各 1000 条，共 2000 个数据进行试验。

数据如图所示：

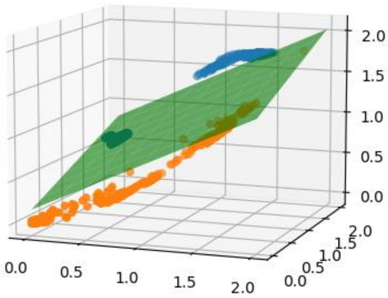


改造一下先前用于平面数据的程序即可进行分类。
得到结果如下：

Discriminant function：33.36x1+17.94x2-54.86x3+8.77=0

Final Loss = 2.5197884101640468e-05

precision = 1.0



可见逻辑回归准确地分类了数据集。

五、结论

逻辑回归可以很好地解决简单的线性分类问题，其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，而是直接根据已有的数据求解分类超平面。

从结果中可以看出，类条件分布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

1）Pattern Recognition and Machine Learning.
2）Gradient descent wiki
3）Conjugate gradient method wiki
4）Shewchuk J R. An introduction to the conjugate gradient method without the agonizing pain[J]. 1994.

七、附录：源代码（带注释）

文件名	说明
config.py	参数设置
dataGenerator.py	数据生成器
logistics_regression.py	逻辑回归主程序
UCI_lr.py	逻辑回归程序 uci 改造

config.py

```
# 第一类数据中心
center_x = 0
center_y = 0
# 第二类数据中心
```

```
center_a = 3
center_b = 3
# 单类数据量
nums = 100
# 第一类数据标准差
rand_normal_x = 1
rand_normal_y = 1
# 第二类数据标准差
rand_normal_a = 1
rand_normal_b = 1
# 非朴素贝叶斯参数
naive_rate_xy = 0
naive_rate_ab = 0
# 学习率
learning_rate = 0.3
# 文件路径
data_file = r"data/data_anti_naive.csv"
uci_file = r'data/skin.txt'
```

dataGenerator.py

```
import numpy as np
import matplotlib.pyplot as plt
from config import *


# 单类点生成器
def generatePoints(center, rand_normal_x, rand_normal_y, nums, naive_rate):
    # 中心
    x = center[0]
    y = center[1]
    # 协方差矩阵
    cov = [[rand_normal_x, naive_rate], [naive_rate, rand_normal_y]]
    # x 各维生成
    xs = np.random.multivariate_normal((x, y), cov, nums)
    return xs

# 生成点并画出点图
xs = generatePoints([center_x, center_y], rand_normal_x, rand_normal_y, nums, naive_rate_xy)
x = [xs[i][0] for i in range(xs.shape[0])]
y = [xs[i][1] for i in range(xs.shape[0])]
```

```
xs = generatePoints([center_a, center_b], rand_normal_a, rand_normal_b,
nums, naive_rate_ab)
a = [xs[i][0] for i in range(xs.shape[0])]
b = [xs[i][1] for i in range(xs.shape[0])]
plt.scatter(x, y)
plt.scatter(a, b)
plt.legend(['0', '1'])
plt.show()
# 保存点
matrix = np.append(x, a)
matrix = np.append(matrix, y)
matrix = np.append(matrix, b)
matrix = np.append(matrix, [0 for i in range(len(x))])
matrix = np.append(matrix, [1 for j in range(len(a))])
matrix.resize(3, 2 * nums)
np.savetxt(fname=data_file, X=matrix)
```

logistics_regression.py

```
import numpy as np
import matplotlib.pyplot as plt
from config import *


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


# 读取 x, y
def getXY():
    xy = np.loadtxt(data_file)
    x = xy[:2]
    y = xy[2]
    x = np.concatenate([x, np.array([[1 for i in range(2 * nums)]])],
axis=0)
    return x.T, y.T


# 计算 loss 函数
def getLoss(x, y, w):
    loss = 0
    for i in range(x.shape[1]):
```

```
        wx = w @ x[i]
        loss_single = y[i] * np.log(sigmoid(wx)) + (1 - y[i]) * np.log(1
- sigmoid(wx))
        loss -= loss_single
    return loss


# 梯度下降计算
def getW():
    x, y = getXY()
    # 随机生成初始矩阵
    w = np.array([1.0 for i in range(3)])
    # 计算初始 loss
    old_loss = loss = getLoss(x, y, w)
    times = 0  # 迭代次数
    # 正则化参数
    lambda_analytical = 0.1
    delta = 1
    # 梯度下降主循环
    while True:
        delta_w = np.array([0.0 for i in range(3)])
        for i in range(x.shape[0]):
            wx = w @ x[i]
            d = x[i] * (y[i] - sigmoid(wx))
            delta_w += d
        w += (learning_rate * delta_w / x.shape[1] - lambda_analytical
* w) * delta
        loss = getLoss(x, y, w)
        times += 1
        # 递减学习率
        if times % 1000 == 0:
            delta *= 0.96
        print('epoch {}: Loss = {}'.format(times, loss))
        if abs(old_loss - loss) < 10 ** -8:
            break
        old_loss = loss
    print('Training completed.\nFinal Loss = {}\nw = {}'.format(loss, w))
    return w


# 计算准确率
def getAccuracy(w, x, y):
    total = y.shape[0]
    correct = 0
```

```
    for i in range(x.shape[0]):
        if w @ x[i] < 0:
            correct += 1 - y[i]
        else:
            correct += y[i]
    return correct / total


# 画图
w = getW()
xx, yy = getXY()
print(getAccuracy(w, xx, yy))
points = getXY()
points_xs, _ = getXY()
points_xs = points_xs[:nums]
x = [points_xs[i][0] for i in range(nums)]
y = [points_xs[i][1] for i in range(nums)]
points_as, _ = getXY()
points_as = points_as[nums:]
a = [points_as[i][0] for i in range(nums)]
b = [points_as[i][1] for i in range(nums)]
line_x = np.linspace(-2, 5, 300)
line_y = (-w[2] - w[0] * line_x) / w[1]
plt.plot(line_x, line_y)
plt.scatter(x, y)
plt.scatter(a, b)
plt.legend(['Discriminant function', '0', '1'])
plt.show()
```

UCI_lr.py

```
import numpy as np
import matplotlib.pyplot as plt
from config import *


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


# 读取x，y
def getXY():
```

```
    x = [[], [], [], []]
    y = []
    with open(uci_file, 'r') as uci:
        for line in uci:
            if line == '\n':
                continue
            line_split = line.rstrip().split('\t')
            for i in range(3):
                x[i].append(float(line_split[i]))
            x[3].append(1)
            y.append(float(line_split[3]) - 1)
    x = np.array(x)
    for i in range(3):
        mean = x[i].mean()
        for j in range(len(x[i])):
            x[i][j] /= mean
    y = np.array(y)
    return x.T, y.T


# 计算 loss 函数
def getLoss(x, y, w):
    loss = 0
    for i in range(x.shape[0]):
        wx = w @ x[i]
        loss_single = y[i] * np.log(sigmoid(wx)) + (1 - y[i]) * np.log(1
- sigmoid(wx))
        loss -= loss_single
    loss /= x.shape[0]
    return loss


# 梯度下降计算
def getW():
    x, y = getXY()
    # 随机生成初始矩阵
    w = np.array([1.0 for i in range(4)])
    # 计算初始 loss
    old_loss = loss = getLoss(x, y, w)
    times = 0  # 迭代次数
    # 正则化参数
    lambda_analytical = 0
    # 梯度下降主循环
    while True:
```

```
        delta_w = np.array([0.0 for i in range(4)])
        for i in range(x.shape[0]):
            wx = w @ x[i]
            d = x[i] * (y[i] - sigmoid(wx))
            delta_w += d
        delta_w /= np.linalg.norm(delta_w)
        w += learning_rate * delta_w / x.shape[1] - lambda_analytical *
w
        loss = getLoss(x, y, w)
        times += 1
        print('epoch {}: Loss = {}'.format(times, loss))
        if abs(old_loss - loss) < 10 ** -7:
            break
        old_loss = loss
    print('Training completed.\nFinal Loss = {}\nw = {}'.format(loss, w))
    return w


# 计算准确率
def getAccuracy(w, x, y):
    total = y.shape[0]
    correct = 0
    for i in range(x.shape[0]):
        if w @ x[i] < 0:
            correct += 1 - y[i]
        else:
            correct += y[i]
    return correct / total


# 画图
w = getW()
xx, yy = getXY()
print(getAccuracy(w, xx, yy))
x, y = getXY()
xs = [[], [], [], []]
ys = [[], [], [], []]
for i in range(y.shape[0]):
    if y[i] == 0:
        for j in range(4):
            xs[j].append(x[i][j])
    else:
        for j in range(4):
            ys[j].append(x[i][j])
```

```
ax = plt.axes(projection='3d')
ax.view_init(10, -70)
x = np.linspace(0, 2, 9)
y = np.linspace(0, 2, 9)
X, Y = np.meshgrid(x, y)
ax.plot_surface(X, Y, Z=-(w[0] * X + w[1] * Y + w[3]) / w[2], color='g',
alpha=0.6)
ax.scatter3D(xs[0], xs[1], xs[2], 'gray')
ax.scatter3D(ys[0], ys[1], ys[2], 'gray')
plt.show()
```