班级:1613012¶

姓名：安炳旭¶

学号： 16130120187¶

作业：用梯度下降法实现对率回归（Logisitc Regression）(《机器学习》教材第三章课后习题 3.3), 并给出算法主要步骤和程序以及在西瓜数据集 3.0α 上的结果.¶

1.导入所需的包¶

In [1]:

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

2.设置西瓜书的数据集¶

密度 含糖率 好瓜

In [2]:

```
data = np.array([
[0.697, 0.460, 1],
[0.774, 0.376, 1],
[0.634, 0.264, 1],
[0.608, 0.318, 1],
[0.556, 0.215, 1],
[0.403, 0.237, 1],
[0.481, 0.149, 1],
[0.437, 0.211, 1],
[0.666, 0.091, 0],
[0.243, 0.267, 0],
[0.245, 0.057, 0],
[0.343, 0.099, 0],
```

```
[0.639, 0.161, 0],
[0.657, 0.198, 0],
[0.360, 0.370, 0],
[0.593, 0.042, 0],
[0.719, 0.103, 0],
])
```

3.可视化数据¶

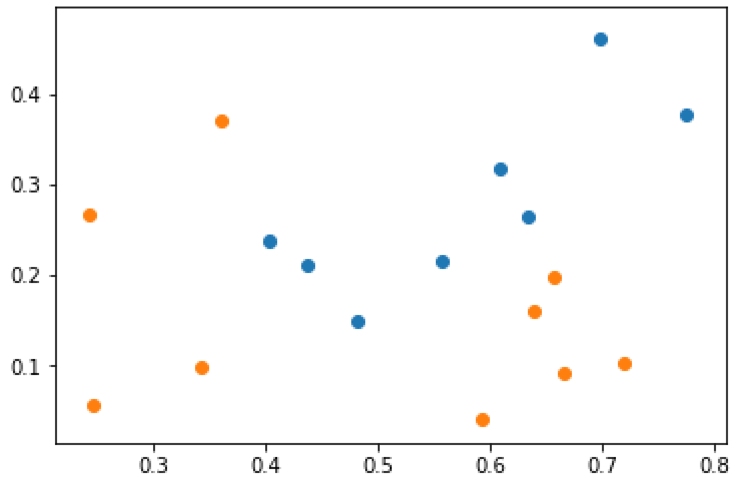
数据较少 而且主要是为了理解梯度下降算法故不进行样本集划分了¶

In [3]:

```
X_train = data[:,0:-1]
y_train = data[:,-1]
```

In [4]:

```
plt.scatter(X_train[y_train==1,0],X_train[y_train==1,1])
plt.scatter(X_train[y_train==0,0],X_train[y_train==0,1])
plt.show()
```



4.定义损失函数¶

In [5]:

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

In [6]:

```
def J(theta, X_b, y):
    y_hat = sigmoid(X_b.dot(theta))
    try:
```

```
        return -np.sum((y * np.log(y_hat)+(1-y)*np.log(1-y_hat))) / len(y)
    except:
        return float('inf')
```

5.损失函数的导数¶

In [7]:

```
def dJ(theta, X_b, y):
    return X_b.T.dot(sigmoid(X_b.dot(theta)) - y) * 2. / len(y)
```

6.梯度下降算法(并且绘制损失函数的曲线)¶

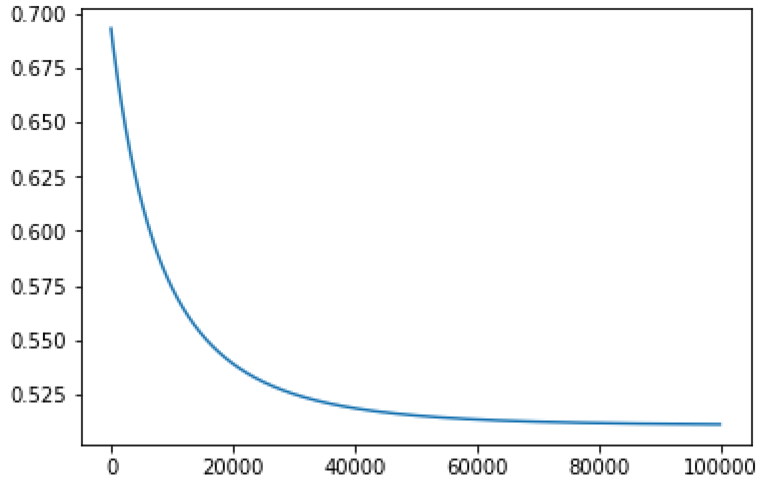
In [8]:

```
def gradient_descent(X_b, y, initial_theta, eta, n_iters=1e4, epsilon=1e-8):
    #X_b 为增广矩阵  第一列全是一  y = theta *X_b
    theta = initial_theta
    cur_iter = 0
    list_iter = []
    list_x_100 =[]
    while cur_iter < n_iters:
        if cur_iter%100==0:
            list_iter.append(J(theta, X_b, y))
            list_x_100.append(cur_iter)
        gradient = dJ(theta, X_b, y)
        last_theta = theta
        theta = theta - eta * gradient
        if (abs(J(theta, X_b, y) - J(last_theta, X_b, y)) < epsilon):
            break
        cur_iter += 1
    #绘制 loss 曲线
    plt.plot(list_x_100,list_iter)
    plt.show()
    return theta
```

In [9]:

```
# 将 X 矩阵的增加一列全是一的同维度矩阵 这样方便算法的向量化
X_b = np.hstack([np.ones((len(X_train), 1)), X_train])
# 将 theta 全都初始化为 theta 进行梯度下降
initial_theta = np.zeros(X_b.shape[1])
# eta 代表学习速率  n_iters  代表最大迭代次数     eta 过大 会导致发散！！！

final_theta = gradient_descent(X_b, y_train, initial_theta=initial_theta,
eta=0.01,n_iters=100000)
```



w 为系数矩阵 b 为截距

In [10]:

```
w = final_theta[1:]
```

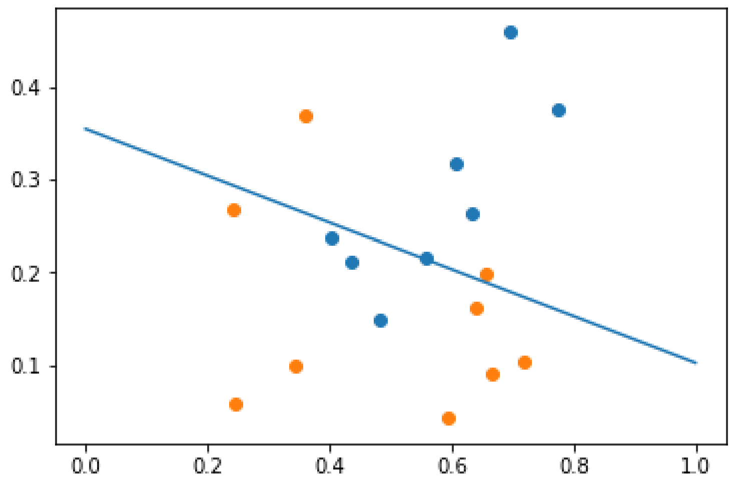
In [11]:

```
b = final_theta[:1]
```

7.绘制决策边界函数实现¶

In [33]:

```
x_plot1 = np.linspace(0,1,100)
x_plot2 = (-b-w[0]*x_plot1)/w[1]
plt.plot(x_plot1,x_plot2)
plt.scatter(X_train[y_train==1,0],X_train[y_train==1,1])
plt.scatter(X_train[y_train==0,0],X_train[y_train==0,1])
plt.show()
```



8.结果¶

In [24]:

```
predict_y = np.array(sigmoid(X_train.dot(w)+b)> 0.5,dtype="float32")
```

In [36]:

```
print(predict_y)
```

```
[ 1.  1.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  1.  1.  0.  0.]
```

In [37]:

```
print(y_train)
```

```
[ 1.  1.  1.  1.  1.  1.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

准确率¶

In [35]:

```
accuracy = np.sum(predict_y ==y_train)/len(y_train)
print(accuracy)
```

```
0.705882352941
```