1 线性回归 Linear Regression

1.1 输入数据集

In [3]:
```
# loading data
data = np.loadtxt(r'E:\docs\机器学习\MachineLearning_HW_CQUT\HW1 linear model\data1.txt',delimiter=',')
print(data.shape)
num_feature = data.shape[1] - 1
data = data.astype('float32')
```

```
(47, 3)
```

In [4]:
```
#data
```

Out[4]:
```
array([[2.10400e+03, 3.00000e+00, 3.99900e+05],
       [1.60000e+03, 3.00000e+00, 3.29900e+05],
       [2.40000e+03, 3.00000e+00, 3.69000e+05],
       [1.41600e+03, 2.00000e+00, 2.32000e+05],
       [3.00000e+03, 4.00000e+00, 5.39900e+05],
       [1.98500e+03, 4.00000e+00, 2.99900e+05],
       [1.53400e+03, 3.00000e+00, 3.14900e+05],
       [1.42700e+03, 3.00000e+00, 1.98999e+05],
       [1.38000e+03, 3.00000e+00, 2.12000e+05],
       [1.49400e+03, 3.00000e+00, 2.42500e+05],
       [1.94000e+03, 4.00000e+00, 2.39999e+05],
       [2.00000e+03, 3.00000e+00, 3.47000e+05],
       [1.89000e+03, 3.00000e+00, 3.29999e+05],
       [4.47800e+03, 5.00000e+00, 6.99900e+05],
       [1.26800e+03, 3.00000e+00, 2.59900e+05],
       [2.30000e+03, 4.00000e+00, 4.49900e+05],
       [1.32000e+03, 2.00000e+00, 2.99900e+05],
       [1.23600e+03, 3.00000e+00, 1.99900e+05],
       [2.60900e+03, 4.00000e+00, 4.99998e+05],
```

数据集标准化，归一化：

In [5]:
```
data_norm = data.copy()
```

In [6]:
```
maximum = np.max(data_norm,axis=0,keepdims=True)
```

In [7]:
```
print(maximum)
```

```
[[4.478e+03 5.000e+00 6.999e+05]]
```

In [8]:
```
minimun = np.min(data_norm,axis=0,keepdims=True)
```

In [9]:
```
data_norm = (data_norm - minimun)/(maximum - minimun)
```

数据集 7:3 划分：

In [11]:
```
data_train, data_test = train_test_split(data_norm, train_size=0.7, random_state=42)
```

In [12]:
```
#data_train
```

Out[12]:
```
array([[1.        , 1.        , 1.        ],
       [0.10590182, 0.5       , 0.05660377],
       [0.        , 0.25      , 0.01886792],
       [0.39933813, 0.75      , 0.5283019 ],
       [0.1770546 , 0.5       , 0.13698113],
       [0.12906784, 0.25      , 0.24528302],
       [0.49227798, 0.5       , 0.24528302],
       [0.32763377, 0.75      , 0.2735849 ],
       [0.27578598, 0.75      , 0.24528302],
       [0.34528404, 0.5       , 0.43396226],
       [0.04081633, 0.        , 0.        ],
       [0.27220076, 0.25      , 0.33962265],
       [0.31246552, 0.75      , 0.24528302],
       [0.31660232, 0.5       , 0.33415094],
       [0.26447877, 0.75      , 0.21886793],
       [0.20628792, 0.5       , 0.3018868 ],
       [0.38223937, 0.5       , 0.3018868 ],
       [0.2857143 , 0.25      , 0.16056603],
       [0.42691672, 0.5       , 0.3756604 ],
       [0.1613348 , 0.5       , 0.1509434 ],
       [0.30612245, 0.75      , 0.16981132],
       [0.35300606, 0.75      , 0.33037737],
       [0.30005515, 0.75      , 0.13226226],
       [0.20739107, 0.5       , 0.13773584],
       [0.484556  , 0.75      , 0.6228264 ],
       [0.09680089, 0.5       , 0.13132076],
       [0.2523442 , 0.5       , 0.15660377],
       [0.15857695, 0.5       , 0.05490377],
       [0.47297296, 0.75      , 0.2718868 ],
       [0.11472698, 0.5       , 0.16981132],
       [0.3717595 , 0.5       , 0.57566035],
       [0.9274683 , 0.75      , 0.71528304]], dtype=float32)
```

1.2 线性回归

正规方程：

In [39]:
```
temp=np.matmul(X_train.T,X_train)
```

In [40]:
```
#temp.shape
```

Out[40]: (3, 3)

In [41]:
```
temp = np.linalg.inv(temp)
```

In [42]:
```
temp1=np.matmul(temp,X_train.T)
```

In [43]:
```
temp2=np.matmul(temp1,y_train)
```

In [44]:
```
w=temp2
```

In [45]:
```
#w
```

Out[45]:
```
array([[ 0.89707607],
       [-0.02132246],
       [ 0.01620799]])
```

梯度下降:

In [50]:
```
#梯度下降
w = np.random.rand(num_feature+1,1)
```

In [53]:
```
def losss(y_pred,y):
    return np.mean(np.square(y_pred-y))
iteration=10000
lr=0.1
```

In [54]:
```
log=[]
for i in range(iteration):
    y_pred=np.matmul(X_train,w)
    loss=losss(y_pred,y_train)
    print('iter:{},loss:{}'.format(i,loss))
    log.append([i,loss])
    term=lr*np.mean((y_pred-y_train)*X_train,axis=0).reshape(-1,1)
    w=w-term
```

...

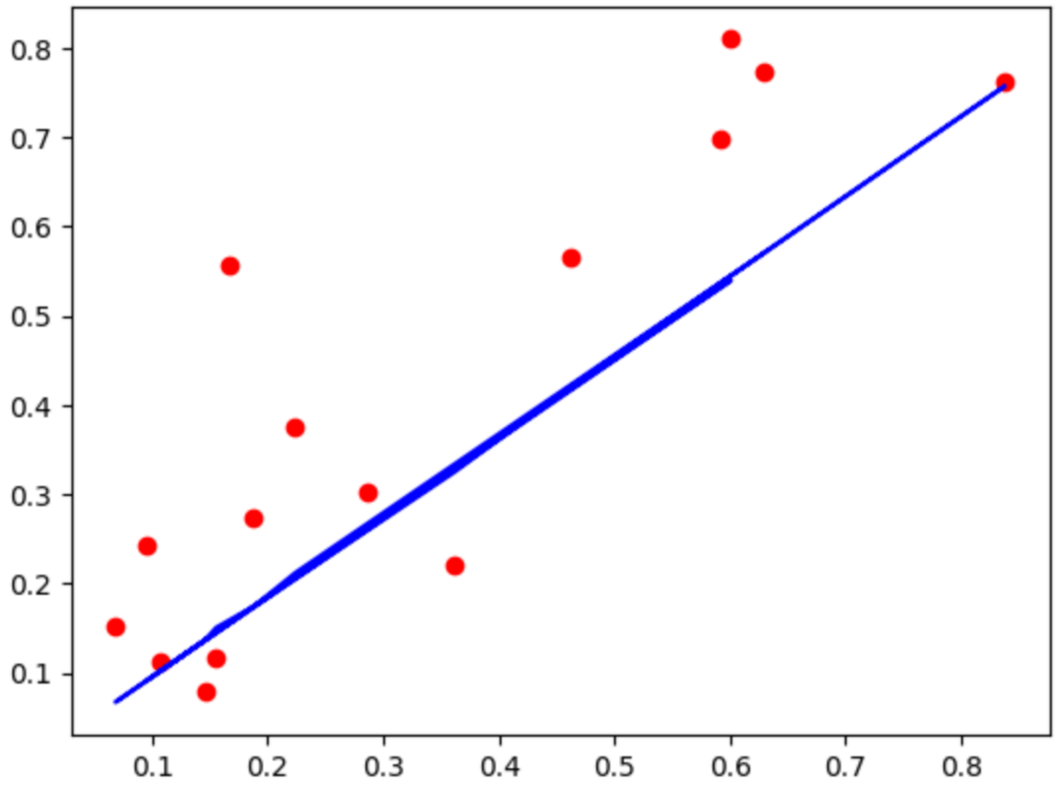
In [57]:
```
w

#loss curve
log=np.array(log)
plt.plot(log[:,0],log[:,1],'r',label='loss')
```

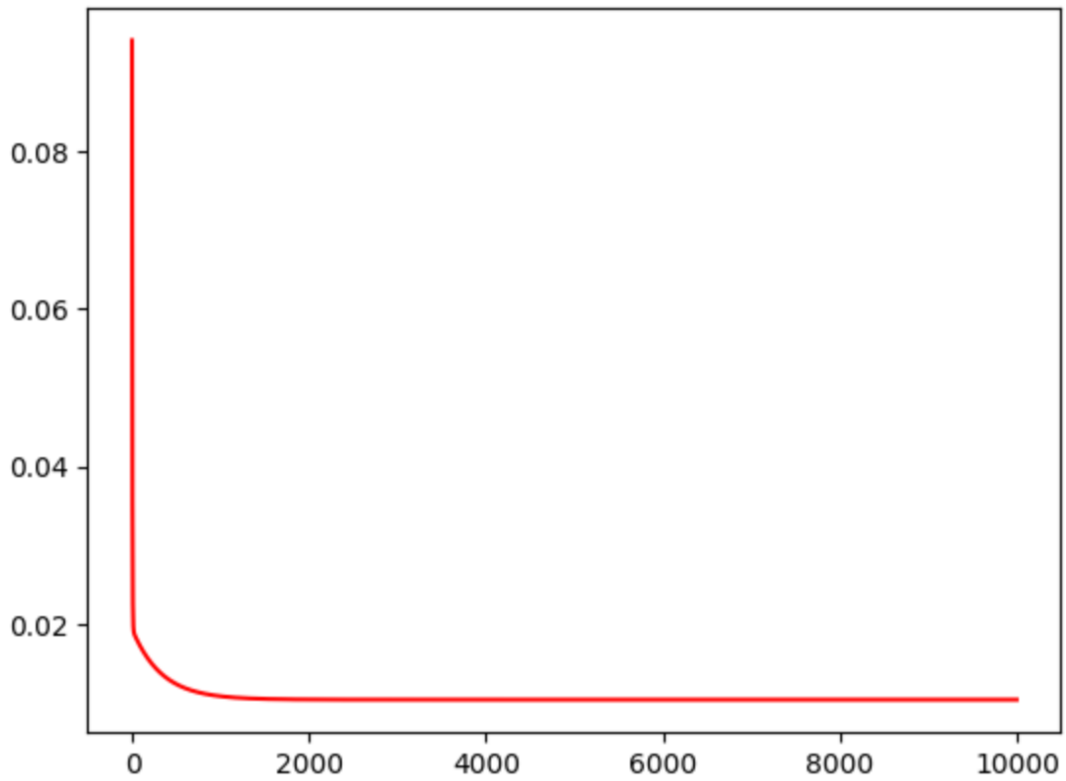
```
iter:0, loss:0.09407709950498533
iter:1, loss:0.07428947680862805
iter:2, loss:0.059729929874943424
iter:3, loss:0.04901529638916592
iter:4, loss:0.04112835506401485
iter:5, loss:0.035321009193869365
iter:6, loss:0.031043084784783896
iter:7, loss:0.027889966452024913
iter:8, loss:0.02556408720640114
iter:9, loss:0.023846606827669377
iter:10, loss:0.022576583246448293
iter:11, loss:0.021635654521083528
iter:12, loss:0.020936773479965802
iter:13, loss:0.020415922821933156
iter:14, loss:0.020026022139618006
iter:15, loss:0.01973244695210906
iter:16, loss:0.019509733260141823
iter:17, loss:0.01933915397197412
iter:18, loss:0.019206936530481375
iter:19, loss:0.019102952099866493
iter:20, loss:0.019019751552193778
iter:21, loss:0.018951856501459096
iter:22, loss:0.018895237907668033
iter:23, loss:0.018846932625818026
iter:24, loss:0.018804761403916688
iter:25, loss:0.018767121489823346
iter:26, loss:0.018732834107773884
iter:27, loss:0.018701032287802573
iter:28, loss:0.018671078371957927
iter:29, loss:0.018642503345769557
iter:30, loss:0.018614962220693462
iter:31, loss:0.01858820122095345
iter:32, loss:0.01856203365170809
```

1.3 可视化

正规方程：



梯度下降结果 loss：



2 逻辑回归 Logitstic Regression/Percetron

2.1 输入数据集

In [2]:
```
# loading data
data = np.loadtxt(r'E:\docs\机器学习\MachineLearning_HW_CQUT\HW1 linear model\data2.txt',delimiter=',')
print(data.shape)
num_feature = data.shape[1] - 1
data = data.astype('float32')
```

```
(100, 3)
```

先划分，再标准化：

```
train_data,test_data=train_test_split(data,train_size=0.7,random_state=42)

train_data_norm=train_data.copy()
test_data_norm=test_data.copy()

train_max=np.max(train_data_norm,axis=0,keepdims=True)
train_min=np.min(train_data_norm,axis=0,keepdims=True)

test_max=np.max(test_data_norm,axis=0,keepdims=True)
test_min=np.min(test_data_norm,axis=0,keepdims=True)

train_data_norm=(train_data_norm-train_min)/(train_max-train_min)
test_data_norm=(test_data_norm-test_min)/(test_max-test_min)

x_train=train_data_norm[:,0:num_feature]
x_train=np.concatenate((x_train,np.ones((x_train.shape[0],1))),axis=1)

y_train=train_data_norm[:,num_feature]
y_train=y_train.reshape((-1,1))

x_test=test_data_norm[:,0:num_feature]
x_test=np.concatenate((x_test,np.ones((x_test.shape[0],1))),axis=1)

y_test=test_data_norm[:,num_feature]
y_test=y_test.reshape((-1,1))
```

2.2 逻辑回归

梯度计算：

```
In [9]: #梯度下降
w = np.random.rand(num_feature+1,1)
iteration=10000
lr=0.1

def sigmoid(x):
    return (math.e**x)/(1+math.e**x)
def cal_loss(y_pred,y):
    return (-1)*np.mean(y*np.log(y_pred)+(1-y)*np.log(1-y_pred))
```

```
In [11]: log=[]
for i in range(iteration):
    y_pred=sigmoid(np.matmul(x_train,w))
    loss=cal_loss(y_pred,y_train)
    log.append([i,loss])
    print('iter:{},loss:{}'.format(i,loss))
    term=lr*np.mean((y_pred-y_train)*x_train,axis=0).reshape(-1,1)
    w=w-term
```

```
iter:0,loss:0.6411275233181749
iter:1,loss:0.6404170833313101
iter:2,loss:0.6397084538437308
iter:3,loss:0.6390016237718079
iter:4,loss:0.638296582519345
iter:5,loss:0.6375933199433236
iter:6,loss:0.6368918263220776
iter:7,loss:0.6361920923257313
iter:8,loss:0.6354941089887407
iter:9,loss:0.6347978676843865
iter:10,loss:0.6341033601010828
iter:11,loss:0.6334105782203725
iter:12,loss:0.6327195142964903
iter:13,loss:0.6320301608373811
iter:14,loss:0.6313425105870706
iter:15,loss:0.6306565565092956
iter:16,loss:0.629972291772298
iter:17,loss:0.6292897097347055
iter:18,loss:0.6286088039324202
```

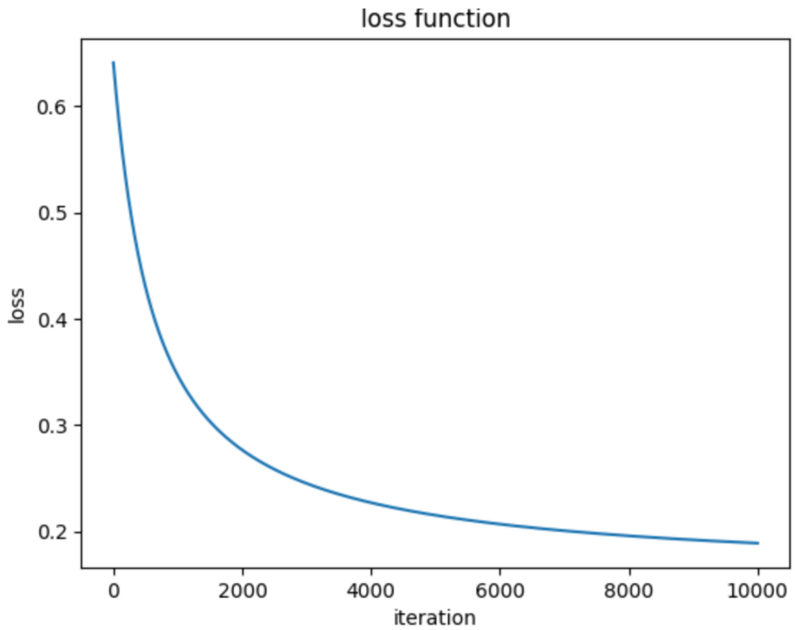
加入正则项：

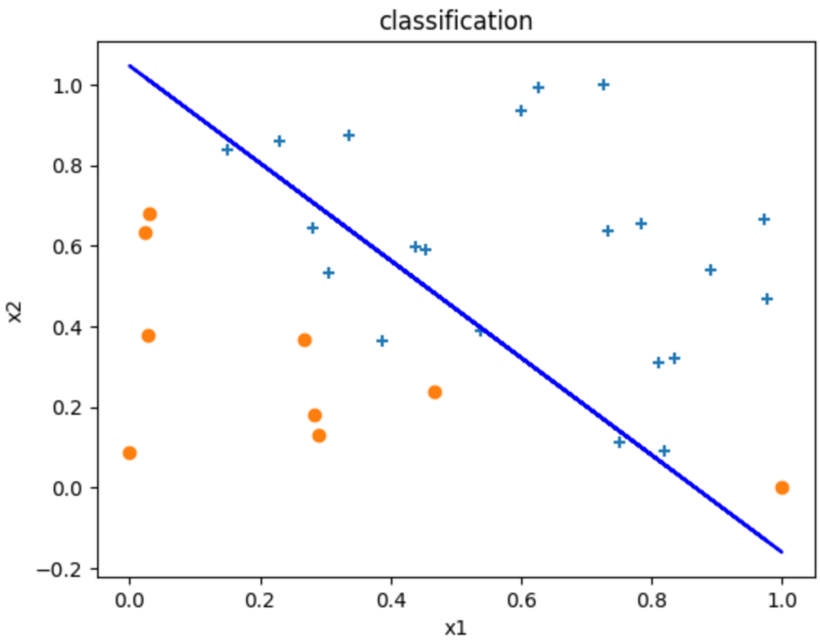
In [41]:
```python
log=[]
for i in range(iteration):
    y_pred=sigmoid(np.matmul(x_train,w))
    loss=cal_loss(y_pred,y_train)
    log.append([i,loss])
    print('iter:{},loss:{}'.format(i,loss))
    #lr=0.1  lambdaw=0.01
    grad=np.mean((y_pred-y_train)*x_train,axis=0).reshape(-1,1)+lambdaw*2*w
    term=lr*grad
    w=w-term
```

```
iter:0,loss:0.690152819380783
iter:1,loss:0.6853588479547684
iter:2,loss:0.6808115959014889
iter:3,loss:0.6765000561613892
iter:4,loss:0.6724133981681859
iter:5,loss:0.6685409997592993
iter:6,loss:0.6648724752500623
iter:7,loss:0.6613976997394363
iter:8,loss:0.6581068297588347
iter:9,loss:0.6549903204123709
iter:10,loss:0.6520389391865157
iter:11,loss:0.649243776630125
iter:12,loss:0.6465962541224837
iter:13,loss:0.6440881289579777
iter:14,loss:0.6417114969818565
iter:15,loss:0.6394587930128706
iter:16,loss:0.6373227892860908
iter:17,loss:0.6352965921434706
iter:18,loss:0.6333736371913888
```

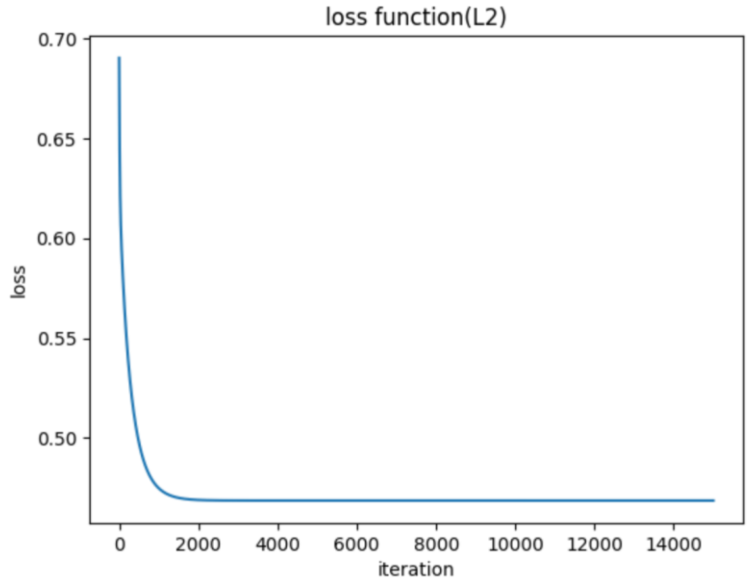
2.3 可视化

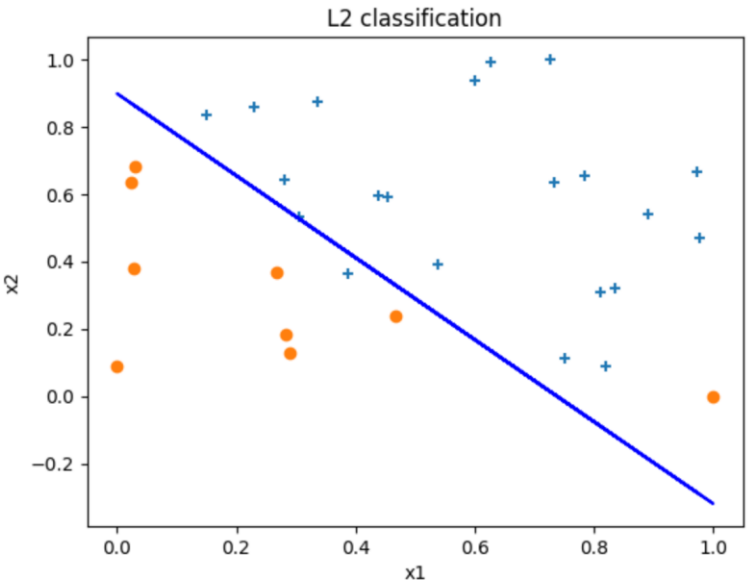
不用 L2 正则化：


loss function
loss
iteration
0.6
0.5
0.4
0.3
0.2
0
2000
4000
6000
8000
10000


classification
x2
x1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

用 L2 正则化：





3 Bonus：分析

1.正规方程要求矩阵是满秩矩阵，否则可能会存在多个解。

梯度下降法是一种迭代的方式，利用计算机的强大算力，能在复杂情况下获得比正规方程更好的结果。

2.用了正则化后的划分结果比不用正则化的结果 loss 反而更高，可能是正则项系数没有设置好的原因。

3.如果不做归一化，会造成尺度不均衡。那么进行梯度下降的时候会出现大尺度

方向参数更新很小，沿着小尺度方向参数更新很大。造成 loss 曲线收敛很慢。