Module II: Medical Data & Machine Learning

Lecture 9 - Implementation of classifier and regression model with Python

By Cheng Peng & Yong Yang

写在前面：

（1）本课程《智医工实验—第二部分》的教学是一门独立课程。但是在教学内容上，本课程仍然不可避免地与生物医学工程系刘泉影老师的《机器学习》课程的教学内容发生联系。

（2）尽管绝大部分同学可能都参加了刘泉影老师的《机器学习》课程，并且都具备了相当的Python语言编程知识，但是本课程仍然需要考虑有不具备Python语言相关应用知识以及编程技巧的同学。

（3）基于以上原因，作为本课程第二阶段的第一节课，我们简单梳理一下本课程的教学内容，并对其分类，总结一下有哪些常用操作，如果同学们对这些操作不太熟练，可以在课后有针对性的进行练习。

问卷：https://www.wjx.cn/report/192094525.aspx?sat=1 (https://www.wjx.cn/report/192094525.aspx?sat=1)

本课程的核心目的有以下几个：

（1）帮助同学们熟悉Python语言的特点以及编程技巧，能够比较熟练的应用Python语言实现一些具有简单功能的程序；

（2）帮助同学们掌握各类与机器学习以及神经网络相关的数学模型与数学思想，包括相关的数学工具，以及其应用场景，适用条件，思考方式等，此外还包括程序化实现手段；

（3）帮助同学们掌握神经网络的应用方法与相关概念，并能够用Python语言实现具备简单功能的神经网络。

关于编程环境与常用工具

关于课程中需要用到的工具，主要有Python编译器和Jupyter notebook两个。

Python编译器有很多种，各位同学可以根据自己的习惯选择，我这里用的是Pycharm。

关于Python环境的配置，这里很难用三言两语说清楚，建议各位同学参考B站视频：

https://www.bilibili.com/video/BV1hE411t7RN/?spm_id_from=333.999.0.0&vd_source=91123ea2d437a2497e09b2dfc55b13be (https://www.bilibili.com/video/BV1hE411t7RN/?spm_id_from=333.999.0.0&vd_source=91123ea2d437a2497e09b2dfc55b13be)

只需要看前两p即可。

在机房中，Pycharm编程环境应改是已经配置好的，但是还是建议各位同学需要回去在自己的电脑上也配置一个Python编程环境，有可能有些部分需要各位同学回去完成。

Jupyter notebook主要用于提交作业以及进行课程演示，课堂上，第一次课的时候各位同学可以不用，教程参见知乎上的这篇帖子： https://zhuanlan.zhihu.com/p/33105153?utm_medium=social (https://zhuanlan.zhihu.com/p/33105153?utm_medium=social)

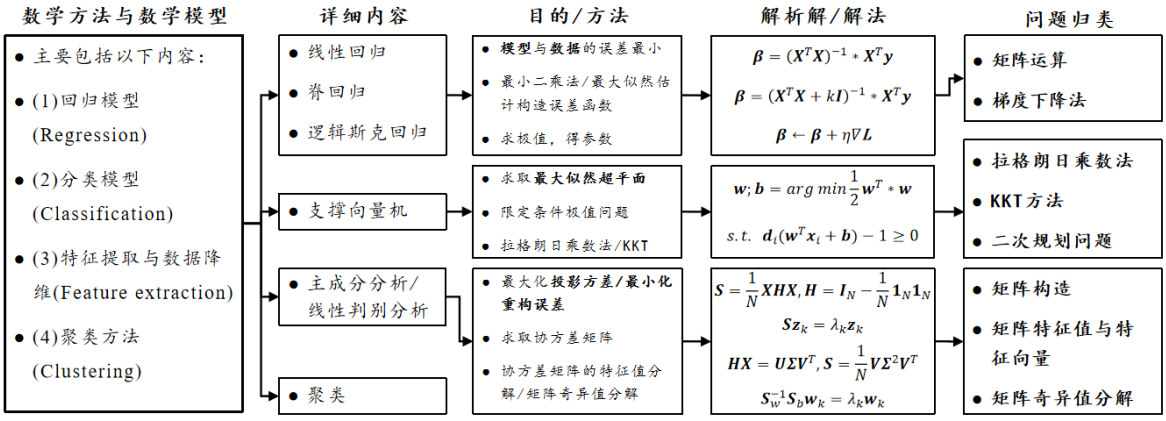
下面我们正式开始课程内容：

我们首先回顾一下刘泉影老师《机器学习》课程中的教学内容，并对这一部分内容进行简单的分类与归纳，从中总结一下我们需要了解或者掌握的基本操作以及数学方法，然后从这些最基本的地方入手开始我们的python编程学习。

在这里，我们粗略的将《机器学习》的内容分成如下两类：

（1）数学方法与数学模型：回归模型(Regression)；分类模型(Classification)；特征提取与数据降维(Feature extraction)；聚类方法(Clustering)。

（2）神经网络：神经网络(Neural Network)；卷积神经网络(CNN)；循环神经网络(RNN)。 我们前几次课程主要集中讨论第一部分的内容，第二部分等我们开始神经网络部分的实验课程时再回顾，不然内容太多，也不方便处理。



从上面的内容，我们可以看到，除了用朴素的数学思想以及数学工具与推导，将一个实际问题转化为数学语言表述并解决的核心过程之外，一些更基本的操作包括：

（1）矩阵四则运算以及矩阵求逆；

（2）梯度下降法；

（3）拉格朗日乘数法；

（4）最优解与二次规划问题；

（5）矩阵特征值与特征向量；

（6）矩阵的奇异值分解；

（7）……

当然，还有更基本的一些操作，比如：

（1）导入库；

（2）数据读取；

（3）循环/迭代；

（4）画图；

（5）创建数组(矩阵)；

（6）矩阵重组；

（7）……

如果各位同学对这些操作很熟悉，那么本次实验课将会较为轻松，如果不是那么熟悉，我们课上会进行一部分练习，当然同学们也可以下去之后进行有针对性的练习。

Python基本操作练习——导入

In [2]:
```
#导入库1
import numpy
A = numpy.zeros((3, 5))
print(A)
```

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

In [4]:
```
#导入库2
import numpy as np
A = np.zeros((2, 4))
print(A)
```

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

In [5]:
```
#导入库3
from numpy import *
A = zeros((4, 3))
print(A)
```

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

Python基本操作练习——数据读取

In [6]:
```
#数据读取
file_name = 'test.txt'
data = np.loadtxt(file_name, dtype='float32', delimiter=' ')
print(data)
```

```
[9. 8. 5. 2. 1. 1.]
```

读取文字

In [2]:
```
f = open("test_文本.txt", mode='r', encoding='utf-8')

result = f.read(6)

f.close()

print(result)
```

南方科技大学

Python基本操作练习——for循环/while循环

In [10]:
```
#for循环1
for x in range(1,10):
    print(x)
```

1
2
3
4
5
6
7
8
9

In [11]:
```
#for循环2
for num in range(1,100):
    pass
print(num)
```

99

In [12]:
```
#while循环
i = 0
while i < 20:
    print(i)
    i=i+2
```

0
2
4
6
8
10
12
14
16
18

Python基本操作练习——判断/分支语句

In [13]:
```
#简单判断
for a in range(1, 10):
    if a % 2 == 0:
        print(str(a) + '   是偶数')
    else:
        pass
```

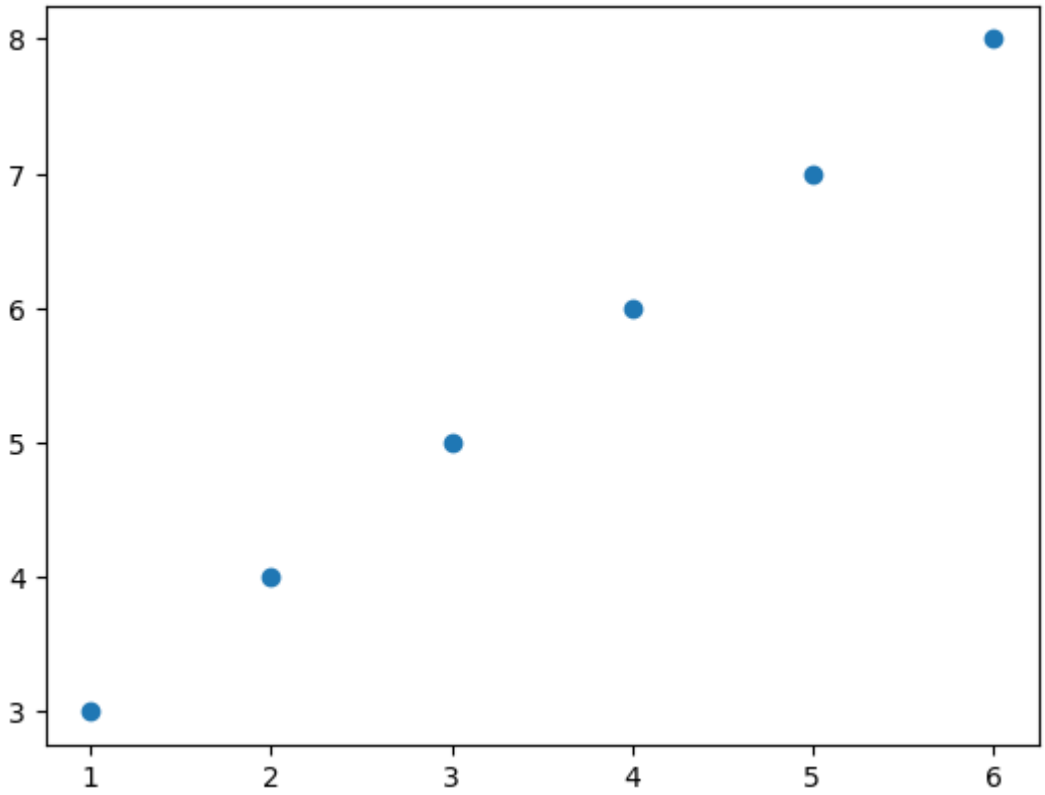
```
2   是偶数
4   是偶数
6   是偶数
8   是偶数
```

In [14]:
```
#多重判断
a = 10
b = 20

if a < b:
    print(str(a) + '<' + str(b))
elif a == b:
    print(str(a) + '=' + str(b))
elif a > b:
    print(str(a) + '>' + str(b))
else:
    pass
```

```
10<20
```

Python基本操作练习一画图 更详细的画图命令介绍:
https://blog.csdn.net/Gou_Hailong/article/details/120089602
(https://blog.csdn.net/Gou_Hailong/article/details/120089602)

In [15]:
```
#简单画图
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5, 6]
y = [3, 4, 5, 6, 7, 8]
plt.figure()
plt.scatter(x, y)
plt.show()

#其实这里是有一点问题的，不知道大家看出来没有
```



基本操作统一练习1： 编写一个程序，判断一个正整数是不是素数。

In [16]:
```
# 参考程序
# 输入
num = 42

# 判断是不是素数
n = 2
for i in range(2, num):
    s = num % i
    n = n + 1
    if s == 0:
        break
# 输出
if n < num:
    print(str(num) + "不是素数")
elif n == num:
    print(str(num) + "  是素数")
else:
    pass
# 当然这个程序并不完善，比如当输入的数是负数或者0，1的时候，同学们可以加入几个判断把这
```

```
42不是素数
```

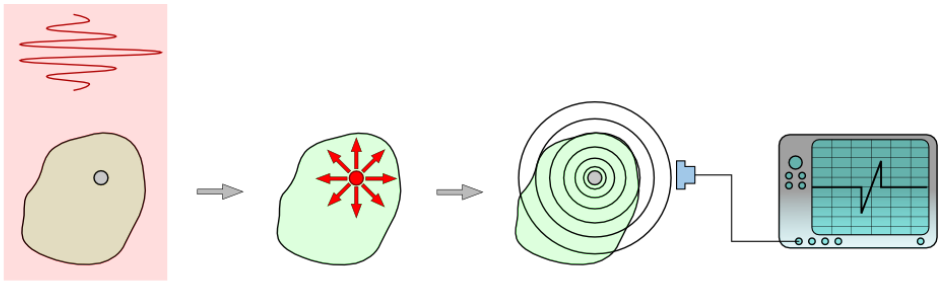
基本操作统一练习2： 编写一个程序，找出1到100之间的所有素数。

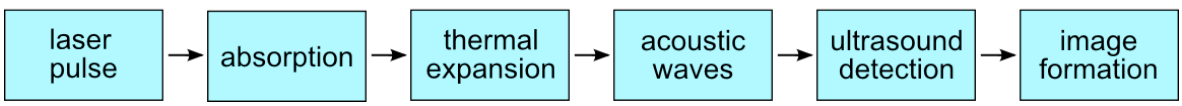
In [17]:
```
# 参考程序
# 输入
for num in range(2, 101):

    # 判断是不是素数
    n = 2
    for i in range(2, num):
        s = num % i
        n = n + 1
        if s == 0:
            break
    # 输出
    if n < num:
        print(str(num) + " 不是素数")
    elif n == num:
        print(str(num) + "   是素数")
    else:
        pass
```

```
2    是素数
3    是素数
4 不是素数
5    是素数
6 不是素数
7    是素数
8 不是素数
9 不是素数
10 不是素数
11    是素数
12 不是素数
13    是素数
14 不是素数
15 不是素数
16 不是素数
17    是素数
18 不是素数
19    是素数
20 不是素数
21 不是素数
22 不是素数
23    是素数
24 不是素数
25 不是素数
26 不是素数
27 不是素数
28 不是素数
29    是素数
30 不是素数
31    是素数
32 不是素数
33 不是素数
34 不是素数
35 不是素数
36 不是素数
37    是素数
38 不是素数
39 不是素数
40 不是素数
41    是素数
42 不是素数
43    是素数
44 不是素数
45 不是素数
46 不是素数
47    是素数
48 不是素数
49 不是素数
50 不是素数
51 不是素数
52 不是素数
53    是素数
54 不是素数
55 不是素数
56 不是素数
57 不是素数
58 不是素数
59    是素数
60 不是素数
61    是素数
62 不是素数
```

63 不是素数
64 不是素数
65 不是素数
66 不是素数
67 是素数
68 不是素数
69 不是素数
70 不是素数
71 是素数
72 不是素数
73 是素数
74 不是素数
75 不是素数
76 不是素数
77 不是素数
78 不是素数
79 是素数
80 不是素数
81 不是素数
82 不是素数
83 是素数
84 不是素数
85 不是素数
86 不是素数
87 不是素数
88 不是素数
89 是素数
90 不是素数
91 不是素数
92 不是素数
93 不是素数
94 不是素数
95 不是素数
96 不是素数
97 是素数
98 不是素数
99 不是素数
100 不是素数

基本操作统一练习3： 进行一次PAT环形扫描中的图像重建。

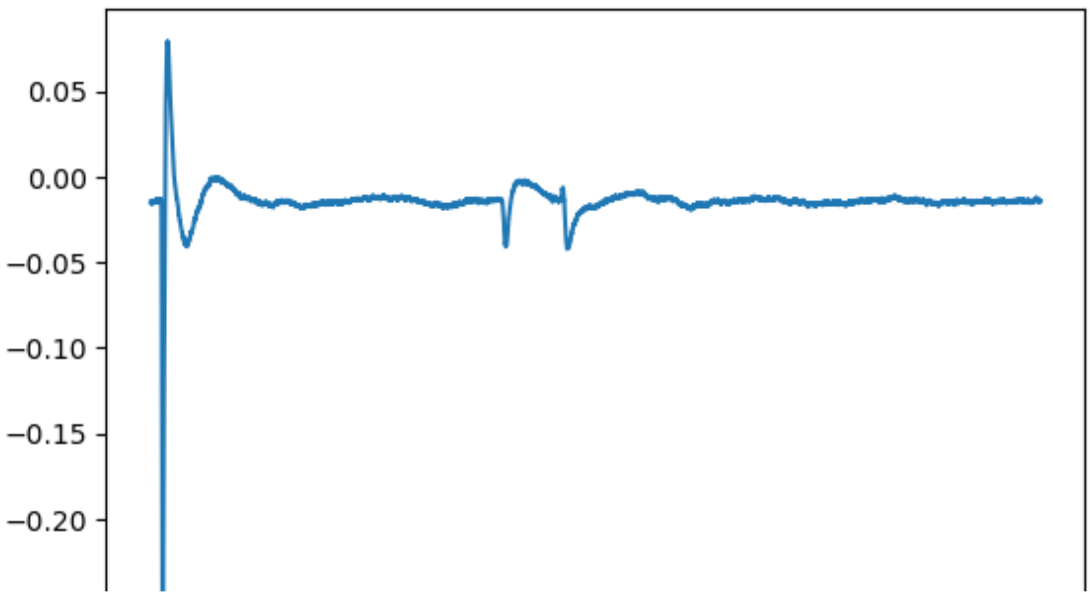




In [18]:
```
# This is a python program to conduct Delay & Sum Reconstruction Algorithm
# The purpose of this program is to learn and practise python programming

# Program Start

# Import Related Libraries
import math as math
import numpy as np
import mpmath as mp
import matplotlib.pyplot as plt

# Data Reading
file_name = '1.lvm'
data = np.loadtxt(file_name, dtype='float32', delimiter=' ')  # 获取数据

# Parameter setting
aline_len = 8000
aline_num = 180

x_min = -10.0
y_min = -10.0
x_max = 10.0
y_max = 10.0
dx = dy = 0.1

v = 1.5
dt = 1/100

# Date Re-shape
arr_aline = np.zeros((aline_len, aline_num))
for aline_index in range(aline_num):
    arr_aline[:, aline_index] = data[aline_index*aline_len:(aline_index+1)*aline_len]

t = np.linspace(0, aline_len, aline_len)
print(t.shape)
plt.plot(t, arr_aline[:, 1])
plt.show()


# Reconstruction
x_num = (x_max - x_min)/dx
y_num = (y_max - y_min)/dy
x_num = int(x_num) + 1
y_num = int(y_num) + 1
det_num = aline_num
im = np.zeros((y_num, x_num), dtype=float)

det_radius = 51.9

for n in range(det_num):
    for x in range(x_num):
        for y in range(y_num):
            x_det = det_radius*mp.cos(math.pi*n*2/det_num)
            y_det = det_radius*mp.sin(math.pi*n*2/det_num)
            x_im = x_min + x*dx
            y_im = y_min + y*dy
            dis2 = (x_det - x_im)**2 + (y_det - y_im)**2
            dis = dis2**0.5
            aline_index = (dis/v)/dt
            aline_index = int(aline_index)+1
            im[y, x] = im[y, x] + arr_aline[aline_index, n]
```

```
    print(n)
# Show Reconstructed Image
plt.imshow(im)
plt.show()
```

```
(8000,)
```



经过上面的练习以及演示，我们应该已经知道python的基本操作需要如何进行了，下面我们就正式开始本次课的实验内容 也就是基于Python实现回归模型与分类器

首先我们尝试一下回归模型。 关于回归模型的详细描述以及数学推导我们在这里就不进行了，只作简单介绍。

什么是回归（regression）？ 回归分析：根据数据建立数学模型，或者已知模型求取参数。因此，模型/参数的准确性十分重要。 怎么度量准确性？ 准确性高 = 误差小（最小二乘法/最大似然估计），当然也有很多其他类型的方法，我们这里暂时只专注于最小二乘法/最大似然估计。

研究问题先从最简单的情况入手，回归模型中最简单的是线性回归。 所谓线性回归，用下面这张图就能说清：

那么接下来就是，我们知道了数据，知道了数学模型的类型，现在怎么求取参数？ （1）根据最小二乘法构造误差函数； （2）求误差函数的极小值； （3）根据极值条件反解出参数。

在有些情况下，上面的过程最终能够得到一个解析解，这个时候可以直接用数据进行计算，这没什么好说的。 但是也有解析解不存在或者很难表示的情况，这个时候我们就不能依赖解析解，而要用其他方法求取极值。 在我们这个实验里面，我们主要采用的是梯度下降法。

梯度下降法也是一种很朴素的数值计算方法。 梯度表示函数增长最快的方向，如果我们要求极小值，那么应该反着梯度的方向去找。 公式： x(j+1) = x(j)+λ*(-grad(x(j)))

其中：x是自变量，λ是学习率，也就是沿着梯度的反方向走多远。 学习率一般会设置的比较小，因为太大的话容易出现异常情况。 我们这里还是进行一下简单的练习。

进阶练习1——梯度下降法： 各位同学可以自定义一个二次型函数，然后用梯度下降法求取其极值点。 比如z=(x-2)^2+(y-3)^2这个函数的极值点。 当然明眼人一眼就能看出来，但是作为练习，我们没必要一下就挑战特别复杂的情况，要先保证方法是正确的。

In [19]:
```
# 梯度下降法范例程序1
# 导入库
import math as math


# 定义函数
def fx(x):
    return (x - 1) ** 2


# 定义导数
def dfx(x):
    return 2 * (x - 1)


# 定义学习率与初始值
gama = 0.1
x1 = 10
# 开始计算-第一次循环
x2 = x1 - gama * dfx(x1)
temp = x2 - x1
n = 0
# 开始计算-后续
while abs(temp) > math.pow(10, -9):
    x2 = x1 - gama * dfx(x1)
    temp = x2 - x1
    x1 = x2
    n = n + 1
print(n)
print(x2)
# 当然，虽然原则上说在不确定循环次数的情况下用while循环，
# 但是这里可以看到while循环有时候还挺麻烦的，
# 所以也可以用for循环代替，这就交给各位同学们来实现了。
```

```
97
1.0000000035807273
```

In [20]:
```
# 梯度下降法范例程序2
# 导入库
import math as math


# 定义函数
def f(x, y):
    return (x - 1) ** 2 + y ** 2


# 定义梯度
def grad_f(x, y):
    return [2 * (x - 1), 2 * y]


# 定义学习率与初始值
gama = 0.1
x1 = 10
y1 = 10
# 开始计算-第一次循环
x2 = x1 - gama * grad_f(x1, y1)[0]
y2 = y1 - gama * grad_f(x1, y1)[1]
temp = (x2 - x1) ** 2 + (y2 - y1) ** 2
n = 0
# 开始计算-后续
while abs(temp) > math.pow(10, -9):
    x2 = x1 - gama * grad_f(x1, y1)[0]
    y2 = y1 - gama * grad_f(x1, y1)[1]

    temp = (x2 - x1) ** 2 + (y2 - y1) ** 2

    x1 = x2
    y1 = y2

    n = n + 1

print(n)
print(x2)
print(y2)
# 当然，虽然原则上说在不确定循环次数的情况下用while循环，
# 但是这里可以看到while循环有时候还挺麻烦的，
# 所以也可以用for循环代替，这就交给各位同学们来实现了。
```

```
52
1.0000822094671
9.134385233318146e-05
```

通过以上的练习，我们应该已经知道梯度下降法怎么使用了，下面我们就来练习线性回归吧。首先我们还是练习简单的线性回归，也就是一维线性回归。

具体做法：

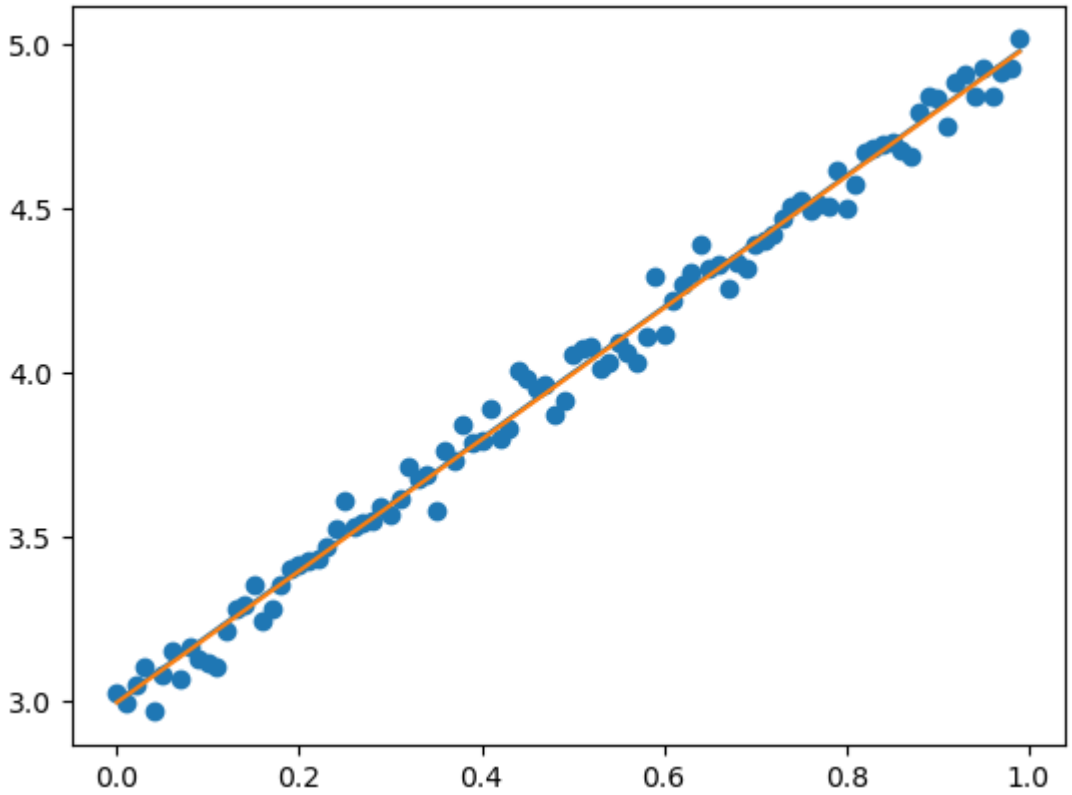
（1）我们自己建立一个线性模型，然后自己产生一组随机观测数据；

（2）然后用我们自己产生的数据来做线性回归，看看我们能不能得到和我们自己设置的模型接近的参数结果；

（3）搞定之后，我们再从数据库里面寻找数据，进行进一步的练习。

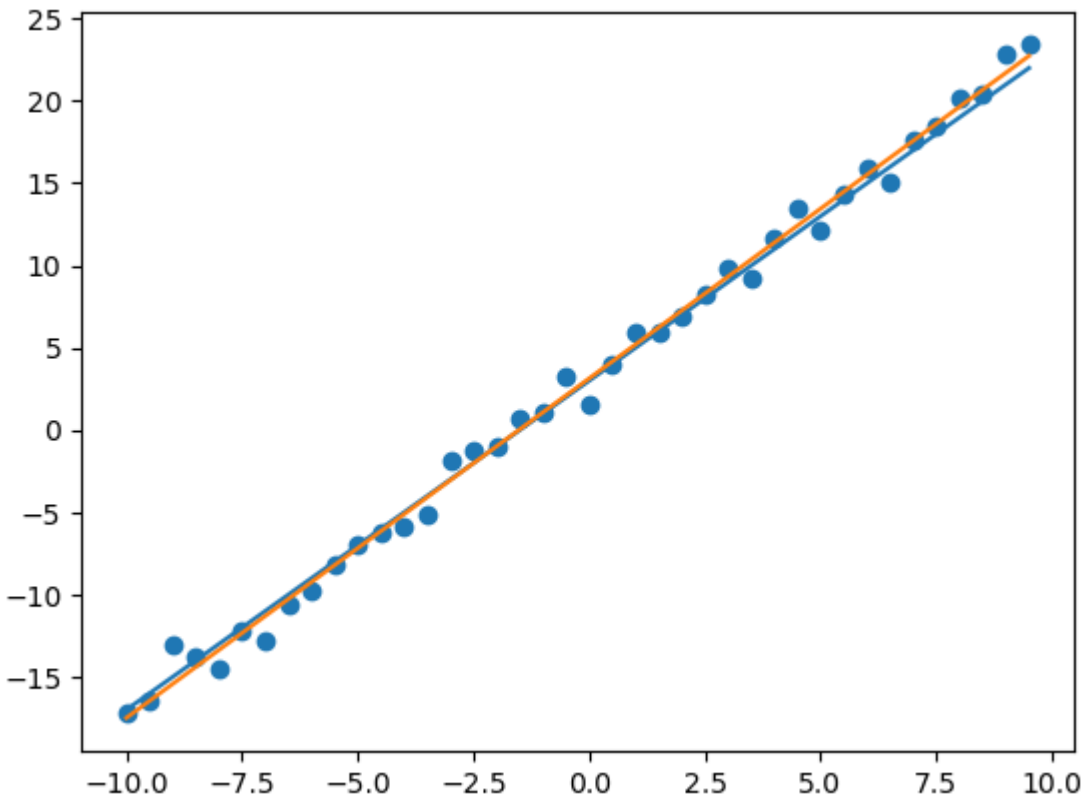
In [21]:
```
# 线性回归练习1——解析法
# 导入库
import numpy as np
import matplotlib.pyplot as plt
# 生成数据
x = np.zeros(100)
eps = np.zeros(100)
for i in range(0, np.shape(x)[0]):
    x[i] = i/100
    eps[i] = np.random.normal(loc=0, scale=0.05, size=None)
y1 = 3 + 2 * x
y2 = y1 + eps
# 构建矩阵
XX = np.zeros((100, 2))
XX[:, 0] = np.ones(100)
XX[:, 1] = x
# 计算
A = np.linalg.inv(np.dot(XX.T, XX))
B = np.dot(XX.T, y2)
beta = np.dot(A, B)
print(beta)
# 查看数据
plt.figure()
plt.plot(x, y1)
plt.plot(x, np.dot(XX, beta.T))
plt.scatter(x, y2)
plt.show()
```

```
[2.99557324 2.00143955]
```



In [22]:
```
# 线性回归练习2——梯度下降法
# 导入库
import numpy as np
import matplotlib.pyplot as plt

# 生成数据
x0 = np.arange(-10, 10, 0.5)
y0 = 3 + 2 * x0 + np.random.normal(loc=0, scale=1, size=np.shape(x0)[0])
y0 = y0.T

# 构建矩阵
N = np.shape(x0)[0]
A = np.ones(N)
X = np.vstack((A, x0))
X = X.T
print(np.dot(np.linalg.inv(np.dot(X.T, X)), np.dot(X.T, y0)))
# 初始值与学习率
gama = 0.01
beta = np.array([-10, -10])
beta = beta.T
# 循环
for i in range(1,1000):
    L = (- 2 / N) * np.dot(X.T, y0 - np.dot(X, beta))
    beta = beta - gama * L
print(beta)
# 画图
plt.figure()
plt.scatter(x0, y0)
plt.plot(x0, 3 + 2 * x0)
plt.plot(x0, beta[0] + beta[1] * x0)
plt.show()
```

```
[3.1412361  2.06298925]
[3.14123608 2.06298925]
```



对于上面的范例，各位同学可以试试看去改变生成数据时的方差（也就是观测误差）的大小。然后重复运行这个程序，看看结果会有什么变化。

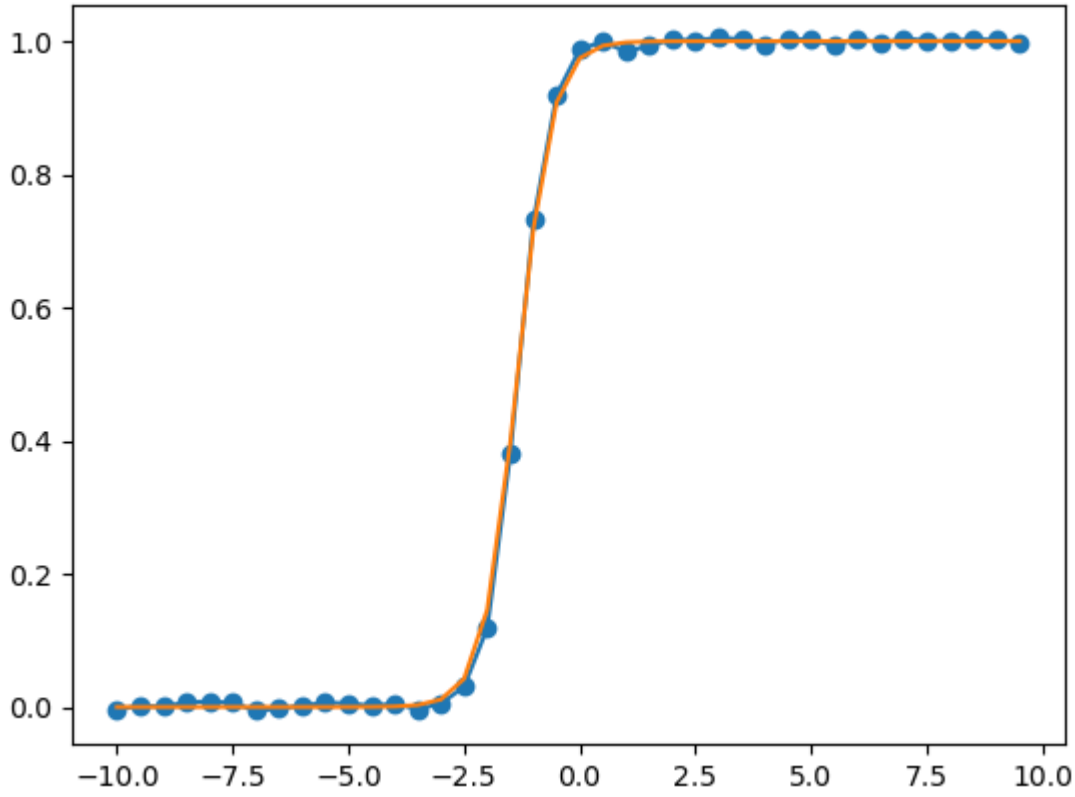
对于脊回归，二维情况下不太好演示。

我们这里先省略一下，后面有是时间了再提，他的思路和方法都是一样的。

同样的，对于逻辑斯克回归，也是一样的方法，当然逻辑斯克回归只能用梯度下降法，因为它的解析解很难表示出来。

In [23]:
```
# 逻辑斯克回归练习——梯度下降法
# 导入库
import math as math
import numpy as np
import matplotlib.pyplot as plt


# 定义函数
def sigma(beta0, beta1, x):
    u = beta0 + beta1 * x
    y = 1 / (1 + math.e ** (-u))
    return y


# 产生数据
x0 = np.arange(-10, 10, 0.5)
y0 = sigma(4, 3, x0) + np.random.normal(loc=0, scale=0.005, size=np.shape(x0)[0])

# 定义学习率与初始值
gama = 0.1
beta0_1 = 10
beta1_1 = 10
grad0 = np.zeros(np.shape(x0)[0])
grad1 = np.zeros(np.shape(x0)[0])
for n in range(0, 10000):
    for i in range(np.shape(x0)[0]):
        grad0[i] = -(y0[i]-sigma(beta0_1, beta1_1, x0[i]))
        grad1[i] = -(y0[i]-sigma(beta0_1, beta1_1, x0[i]))*x0[i]
    grad0sum = sum(grad0)
    grad1sum = sum(grad1)
    beta0_1 = beta0_1 - gama * grad0sum
    beta1_1 = beta1_1 - gama * grad1sum

print(beta0_1)
print(beta1_1)

# 显示数据
plt.figure()
plt.scatter(x0, y0)
plt.plot(x0, y0)
plt.plot(x0, sigma(beta0_1, beta1_1, x0))
plt.show()
```

```
3.6479015833761905
2.7071871076404537
```



以上练习都是我们在可控范围内进行的，我们预先知道真实答案，因此能够判断结果的准确性。

但是实际应用中不可能把真实结果预先通知你，所以我们这里需要一些更接近实际的练习。

如果各位同学进了课题组，在征得课题组PI同意的情况下也可以用自己做实验时的数据。

但是要注意，必须是PI同意的情况下，不然会有麻烦。

此外网络上也有一些数据集，但是比较凌乱

线性回归数据集：https://www.datafountain.cn/datasets/4473 (https://www.datafountain.cn/datasets/4473)

https://www.heywhale.com/mw/dataset/5fb8ddcd15a3c30030604aae/file (https://www.heywhale.com/mw/dataset/5fb8ddcd15a3c30030604aae/file)

鸢尾花分类：iris数据集包含在sklearn库当中，具体在sklearn\datasets\data文件夹下，文件名为iris.csv

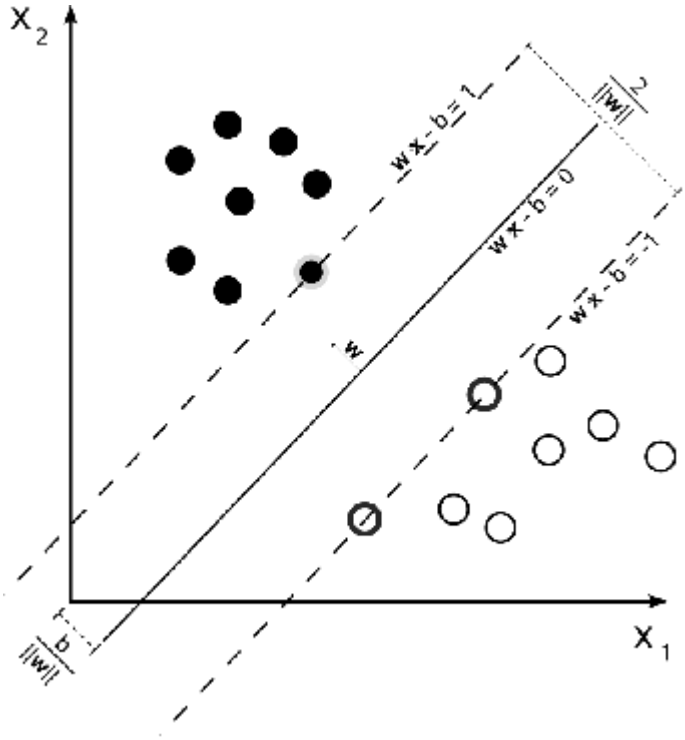
---

上面我们结束了各类回归模型的学习和练习

下面我们接着进行分类器（支撑向量机）的学习

支撑向量机是一类监督学习模型，也就是我们事先知道哪些样本点属于{-1，+1}中的哪一类。

在知道分类情况的前提下，求取最大似然超平面。



在《机器学习》那门课中是有对这个问题的详细推导的，包括怎么把实际问题转化为数学表达，然后再利用数学方法求解最优参数，最后从参数构造出最大似然超平面的过程。

我们这里不做详细的推导了，上面的过程我们在这里简要概括一下，求取最大似然超平面的过程，就是构造一个二次规划问题，然后求最优解的过程，也就是这三步：

（1）构造二次型

（2）求解参数

（3）从参数构造最大似然超平面

首先是二次型的构造问题，构造Q(α)的公式是：

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T \begin{bmatrix} d_1 d_1 \mathbf{x}_1^T \mathbf{x}_1 & d_1 d_2 \mathbf{x}_1^T \mathbf{x}_2 & \cdots & d_1 d_n \mathbf{x}_1^T \mathbf{x}_n \\ d_2 d_1 \mathbf{x}_2^T \mathbf{x}_1 & d_2 d_2 \mathbf{x}_2^T \mathbf{x}_2 & \cdots & d_2 d_n \mathbf{x}_2^T \mathbf{x}_n \\ \vdots & \vdots & \ddots & \vdots \\ d_n d_1 \mathbf{x}_n^T \mathbf{x}_1 & d_n d_2 \mathbf{x}_n^T \mathbf{x}_2 & \cdots & d_n d_n \mathbf{x}_n^T \mathbf{x}_n \end{bmatrix} \boldsymbol{\alpha} + (-\mathbf{1}^T)\boldsymbol{\alpha}$$

$$\text{s.t.} \quad \mathbf{d}^T\boldsymbol{\alpha} = 0$$

$$\mathbf{0} \le \boldsymbol{\alpha} \le \infty$$

里面的α是拉格朗日乘子，也就是拉格朗日乘数法里面会用到的一种参数。

α的维数，也就是具体需要多少个α和限定条件有关(也就是有多少个采样点需要分类)。

The **Lagrangian multipliers**: $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$
The **generalized Lagrangian function** is:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{n} \alpha_i \left(d_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) - 1 \right)$$

拉格朗日乘数法一种寻找变量受一个或多个条件所限制的多元函数的极值的方法。

这种方法将一个有n个变量与k个约束条件的最优化问题转换为一个有n + k个变量的方程组的极值问题，其变量不受任何约束。

设给定二元函数z=*f*(x,y)和附加条件φ(x,y)=0，为寻找z=*f*(x,y)在附加条件下的极值点，先做拉格朗日函数

F(x,y,λ)=*f*(x,y) + λ* φ(x,y)

然后分别令F(x,y,λ)对x，y，λ求偏导数，令其为0，得到三个方程。

由上述方程组解出x,y及λ，如此求得的(x,y)，就是函数z=*f*(x,y)在附加条件φ(x,y)=0下的可能极值点。

下面我们编个程序练习一下

同样的，先从最简单的情况开始，我们搞个一元函数。

目标函数：f（x）=（x - 2）^2

限制条件：φ（x）= x - 3 = 0

当然这个是一目了然的，但是我们要的是用程序来解决问题，而不是我们自己。

编之前大家可以想想这个问题要用什么类型的算法来解决（不是命令），能不能用前面用过的梯度下降法。

参考范例：https://www.easck.com/cos/2020/0805/568178.shtml (https://www.easck.com/cos/2020/0805/568178.shtml)

In [24]:
```
# 导入python的sympy库
import sympy as sy

# 定义自变量
x = sy.symbols('x')
lam = sy.symbols('lambda')

# 定义函数，目标函数（x - 2）^2，限定条件 x - 3 = 0
L = (x - 2)**2 + lam * (x - 3)

# 求偏导数
diffL_x = sy.diff(L, x)
diffL_lam = sy.diff(L, lam)

# 求解
ans = sy.solve([diffL_x, diffL_lam], [x, lam])

print(ans)
```

```
{x: 3, lambda: -2}
```

当然上面这个题目比较白给，我们再练习一下稍微复杂一点的。

比如目标函数f（x，y）= 9xy

然后限制条件：φ（x，y）= x^2 + y^2 = 25

不过方法都是一样的，利用symbols函数构造二次型。

利用solve函数求解。

In [25]:
```
# 导入python的sympy库
import sympy as sy

# 定义自变量
x = sy.symbols('x')
y = sy.symbols('y')
lam = sy.symbols('lambda')

# 定义函数，目标函数9*x*y，限定条件 x^2 + y^2 = 25
L = 9*x*y + lam * (x**2 + y**2 - 25)

# 求偏导数
diffL_x = sy.diff(L, x)
diffL_y = sy.diff(L, y)
diffL_lam = sy.diff(L, lam)

# 求解
ans = sy.solve([diffL_x, diffL_y, diffL_lam], [x, y, lam])

print(ans)
```

```
[(-5*sqrt(2)/2, -5*sqrt(2)/2, -9/2), (-5*sqrt(2)/2, 5*sqrt(2)/2, 9/2), (5*sqrt(2)/
2, -5*sqrt(2)/2, 9/2), (5*sqrt(2)/2, 5*sqrt(2)/2, -9/2)]
```

这个时候就有多个极值点了。

当然，求解上面的类似二次规划的问题的方法不止一种，这里我们也提及其他方法给大家看看。

Python中有专门负责求解这种二次规划问题的函数库。

比如我们下面要用到的scipy.optimize._minimize的这个库

需要注意的是，拉格朗日乘数法只能针对等式类型的限定条件，如果限定条件是不等式，那么就需要对拉格朗日法进行改进才能用。

In [26]:
```
# 导入库
from scipy.optimize import minimize
import numpy as np


# 构造目标函数9*x*y
def func(x):
    fun = 9 * x[0] * x[1]
    return fun


# 创建限定条件 x^2 + y^2 = 25
cons = ({'type': 'eq', 'fun': lambda x: x[0] ** 2 + x[1] ** 2 - 25})

x0 = np.array((0.0, -1.0))  # 设置初始值，初始值的设置很重要，很容易收敛到另外的极值点

# 求解#
ans = minimize(func, x0, method='SLSQP', constraints=cons)
print(ans.success)
print("x1=", ans.x[0], ";  x2=", ans.x[1])
print("最优解为：", ans.fun)
```

```
True
x1= 3.5355319518756985 ;  x2= -3.535535861902913
最优解为：  -112.50000006084122
```

In [ ]:
```
!pip install qpsolvers --index-url http://mirrors.sustech.edu.cn/pypi/simple
```

除了scipy.optimize._minimize

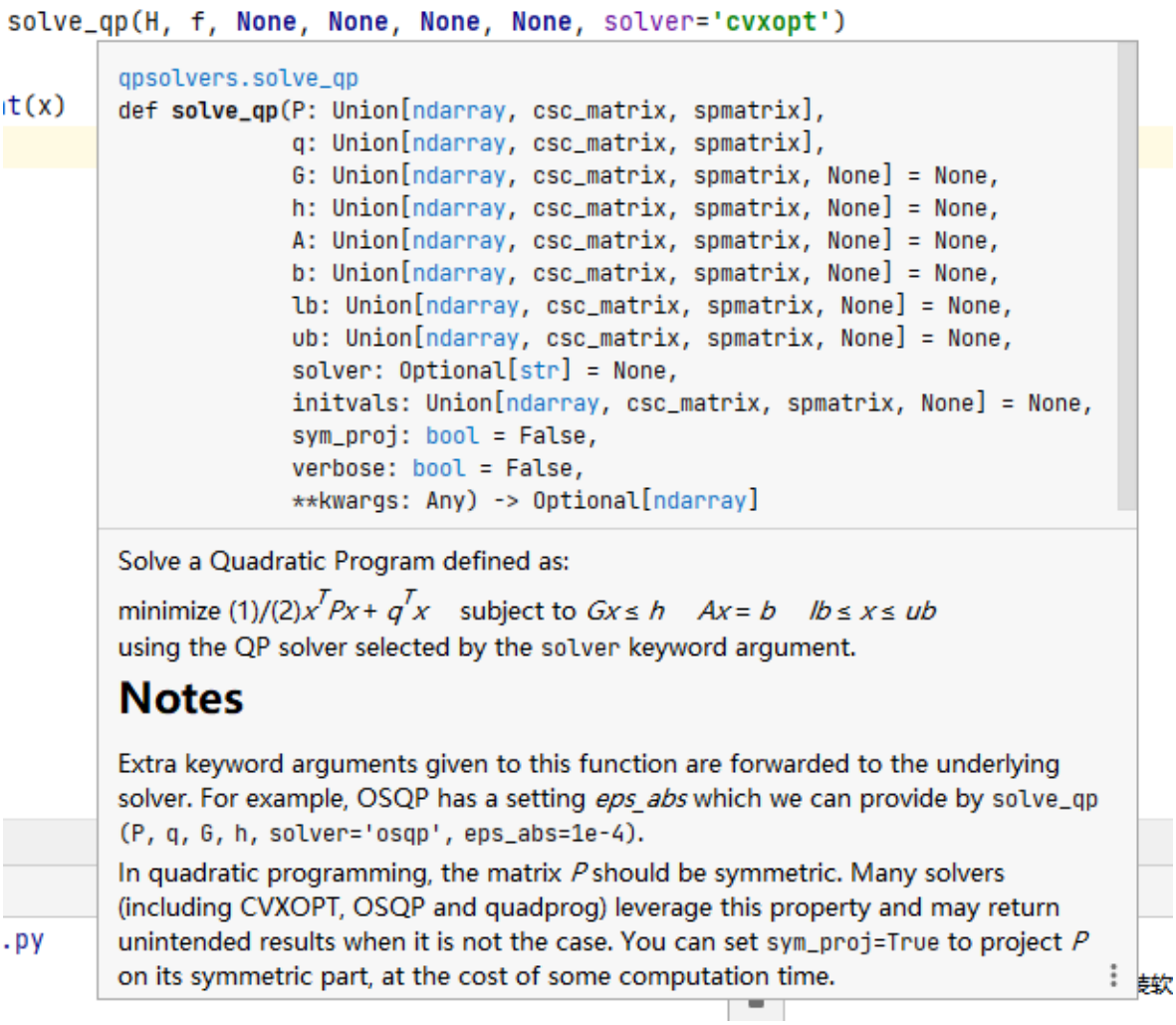
还有qpsolver也能求解这一类最优化问题，这里也演示一下。

In [27]:
```
# 导入库
import numpy as np
from qpsolvers import solve_qp

# 矩阵
H = np.array([[1.0, -1.0], [-1.0, 2.0]])
f = np.array([[-2.0], [-6.0]]).reshape((2,))
L = np.array([[1.0, 1.0], [-1.0, 2.0], [2.0, 1.0]])
k = np.array([[2.0], [2.0], [3.0]]).reshape((3,))
x = solve_qp(H, f, None, None, None, None, solver='cvxopt')

print(x)
```

```
[10.  8.]
```

用solve_qp的时候还要记住，需要安装cvxopt求解器：pip install cvxopt

同时如果是不存在的约束条件，需要用None来补位。



我们再回到SVM里面，怎么构造二次型的问题。

α的维数和限定条件有关，前面的范例中，限制条件只有一个。

但是在SVM中，限制条件和我们要分类的数据点个数相同。

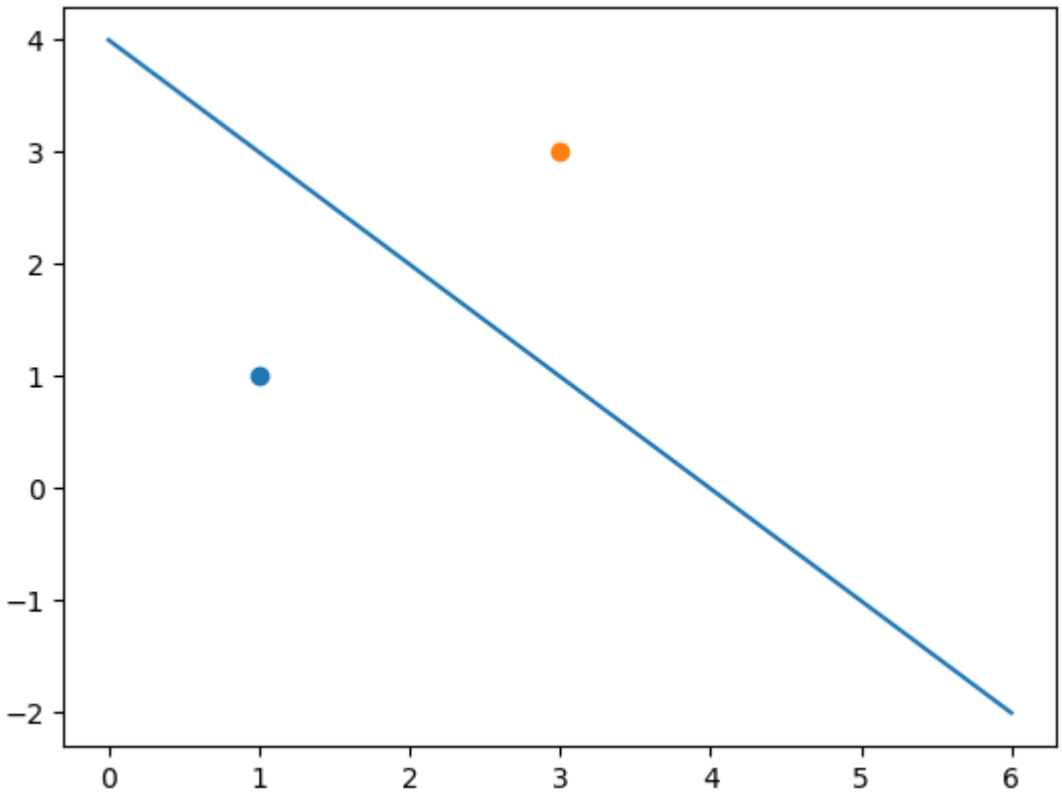
记得我们前面说过的，研究问题从最简单的情况开始。

当我们只有两个点需要分类的时候显然是最简单的情况，可以预见我们要求的最大似然超平面（直线）是两点连线的中垂线。

这个时候我们编写一下程序来测试一下看看。

我们来计算一下点（1，1）和点（3，3）的最大似然超平面，可以预见这个结果是x + y = 4

In [28]:
```
# 导入库
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# 生成数据点1
x1 = np.array([[1.0], [1.0]])
d1 = 1.0
# 生成数据点2
x2 = np.array([[3.0], [3.0]])
d2 = -1.0

# d矩阵
d = np.array([[d1], [d2]])
dd = np.dot(d, d.T)

# x矩阵
xx = np.zeros([2, 2])
xx[0, 0] = np.dot(x1.T, x1)
xx[0, 1] = np.dot(x1.T, x2)
xx[1, 0] = np.dot(x2.T, x1)
xx[1, 1] = np.dot(x2.T, x2)


# 构造二次型
def fun_q(alpha):
    fun = np.dot(np.dot([alpha[0], alpha[1]], dd * xx), [[alpha[0]], [alpha[1]]]) + r
    return fun


# 构造约束条件
cons = ({'type': 'eq', 'fun': lambda alpha: alpha[0] - alpha[1]},
        {'type': 'ineq', 'fun': lambda alpha: alpha[0]},
        {'type': 'ineq', 'fun': lambda alpha: alpha[1]}
        )

# 设置初始值，初始值的设置很重要，很容易收敛到另外的极值点中，建议多试几个值
alpha0 = np.array((10.0, 4.0))

# 求解alpha
ans = minimize(fun_q, alpha0, method='SLSQP', constraints = cons)
alpha_p = np.array([ans.x[0], ans.x[1]])

# 求解斜率
w = alpha_p[0]*d1*x1 + alpha_p[1]*d2*x2

# 求解截距（每个采样点）
b1 = 1/d1-np.dot(w.T, x1)
b2 = 1/d2-np.dot(w.T, x2)

# 求解截距（所有采样点平均）
b = (b1 + b2)/2

plt.figure()
plt.scatter(x1[0], x1[1])
plt.scatter(x2[0], x2[1])
plt.plot(np.array([0, 1, 2, 3, 4, 5, 6]), 4-np.array([0, 1, 2, 3, 4, 5, 6]))
plt.show()
```



上面的结果符合预期，说明我们的程序是对的。

但是这个程序显然也有非常多的问题。

首先它满足了正确性，这一点没有问题，对吧。

然后规范性上，问题也不大，起码各个部分的功能我都标清楚了。

但是除此之外，它就没什么可以说的地方了。

最明显的问题，它只能针对两个采样点的情况，多一个采样点就不行了。

里面的表达式非常生硬，几乎就是把数学公式直接带进去。

点数少的时候还可以，数目多了，就会非常麻烦。

而且这些采样点还只能停留在二维平面上，实际应用中经常会有高维情况出现，这个时候，这个程序也没法处理这种问题。

这个时候我们会说这个程序缺乏泛用性。

但是没关系，我们这个程序是对的，说明我们的理解我们的思路都是没问题的，在这个基础上我们一点一点的修正它。

首先解决采样点数目的问题

但是在正式开始之前，我们先插入一些其他的信息，方便我们后面操作的。

我们再看一下我们前面的范例，包括SVM的理论分析，拉格朗日乘数法，scipy.optimize._minimize库函数以及qpsolvers库的用法。

可以发现，类似SVM这样的二次规划问题，他的关键其实是构造矩阵，也就是下面这个很复杂的矩阵。

$$\arg\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T \begin{bmatrix} d_1 d_1 \mathbf{x}_1^T \mathbf{x}_1 & d_1 d_2 \mathbf{x}_1^T \mathbf{x}_2 & \cdots & d_1 d_n \mathbf{x}_1^T \mathbf{x}_n \\ d_2 d_1 \mathbf{x}_2^T \mathbf{x}_1 & d_2 d_2 \mathbf{x}_2^T \mathbf{x}_2 & \cdots & d_2 d_n \mathbf{x}_2^T \mathbf{x}_n \\ \vdots & \vdots & \ddots & \vdots \\ d_n d_1 \mathbf{x}_n^T \mathbf{x}_1 & d_n d_2 \mathbf{x}_n^T \mathbf{x}_2 & \cdots & d_n d_n \mathbf{x}_n^T \mathbf{x}_n \end{bmatrix} \boldsymbol{\alpha} + (-\mathbf{1}^T)\boldsymbol{\alpha}$$

$$\text{s.t.} \quad \mathbf{d}^T \boldsymbol{\alpha} = 0$$

所以为了后面操作的方便，我们在这里先插入一些在python中的常见矩阵操作。

正好，我们要拓展采样点数目的问题，也是拓展矩阵的维数，那么我们两个问题放在一起，先看看python里面的矩阵操作，然后我们再来拓展矩阵维数，解决我们前面的程序中，采样点数不足的问题。

练习，python中的矩阵操作

Python中有关矩阵操作的库是numpy，我们涉及矩阵操作基本都需要导入这个库。

我们首先看看矩阵的创建。

In [29]:
```
import numpy as np

# 单位矩阵
A = np.eye(3)
print("A")
print(A)

# 对角矩阵
B = np.diag([1.0, 2.0, 3.0, 4.0])
print("B")
print(B)

# 全0矩阵
C = np.zeros([2, 3])
print("C")
print(C)

# 全1矩阵
D = np.ones([3, 2])
print("D")
print(D)
```

```
A
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
B
[[1. 0. 0. 0.]
 [0. 2. 0. 0.]
 [0. 0. 3. 0.]
 [0. 0. 0. 4.]]
C
[[0. 0. 0.]
 [0. 0. 0.]]
D
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

还有矩阵的基本运算

In [30]:
```
import numpy as np

A = np.array([[1, 2, 3],[4, 5, 6]])
B = np.array([[4, 5, 6],[7, 8, 9]])

# 矩阵相加
print("A + B")
print(A + B)

# 矩阵相减
print("A - B")
print(A - B)
print("B - A")
print(B - A)

# 矩阵转置
print("A转置")
print(A.T)
print("B转置")
print(np.transpose(B))

# 矩阵相乘1
print("A*B")
print(A*B)
print("B*A")
print(B*A)

# 矩阵相乘2
print("A*B.T")
print(np.dot(A,B.T))
print("B.T*A")
print(np.dot(B.T,A))
#
```

```
A + B
[[ 5  7  9]
 [11 13 15]]
A - B
[[-3 -3 -3]
 [-3 -3 -3]]
B - A
[[3 3 3]
 [3 3 3]]
A转置
[[1 4]
 [2 5]
 [3 6]]
B转置
[[4 7]
 [5 8]
 [6 9]]
A*B
[[ 4 10 18]
 [28 40 54]]
B*A
[[ 4 10 18]
 [28 40 54]]
A*B.T
[[ 32  50]
 [ 77 122]]
B.T*A
[[32 43 54]
 [37 50 63]
 [42 57 72]]
```

一个需要注意的特殊情况，python中的一维数组

在python里面，一个一维数组，和一个[n, 1]维的二维数组是不太一样的。

主要体现在转置操作的时候。

在刘泉影老师的《机器学习》课程中，经常需要用到一维列向量，以及列向量的转置，所以这里额外提一下。

In [31]:
```
import numpy as np

# 行向量A
A = np.array([1.0, 1.0, 4.0, 5.0, 1.0, 4.0])
print("A")
print(A)

# 试图将A转置
print("A.T")
print(A.T)

# 转置一维向量的正确操作
print("一维矩阵A的shape属性")
print(A.shape)
A.shape = (1,A.shape[0])
print("A.T")
print(A.T)
```

```
A
[1. 1. 4. 5. 1. 4.]
A.T
[1. 1. 4. 5. 1. 4.]
一维矩阵A的shape属性
(6,)
A.T
[[1.]
 [1.]
 [4.]
 [5.]
 [1.]
 [4.]]
```

如果A是一个二维数组，问题就少很多了

In [32]:
```
import numpy as np

# 行向量A
A = np.array([[1.0, 1.0, 4.0, 5.0, 1.0, 4.0]])
print("A")
print(A)

# A转置
print("A.T")
print(A.T)
```

```
A
[[1. 1. 4. 5. 1. 4.]]
A.T
[[1.]
 [1.]
 [4.]
 [5.]
 [1.]
 [4.]]
```

除此之外还有一些矩阵参数有关的函数命令

In [33]:
```
import numpy as np

# 矩阵A
A = np.array([[1.0, 2.0 ,3.0 ],
              [4.0, 5.0 ,6.0 ],
              [7.0 ,8.0 ,9.0 ]])
print("A")
print(A)

# 矩阵的迹trace
print("trace of A")
print(np.trace(A))

# 矩阵的秩rank
print("rank of A")
print(np.linalg.matrix_rank(A))

# 矩阵的维数
print("shape of A")
print(A.shape)

# 矩阵的子集
print("submatrix of A")
print(A[0:2, 0:2])
```

```
A
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
trace of A
15.0
rank of A
2
shape of A
(3, 3)
submatrix of A
[[1. 2.]
 [4. 5.]]
```

维数太大的时候我们可以用循环来解决。

通过以上操作，我们知道了矩阵的常用操作。

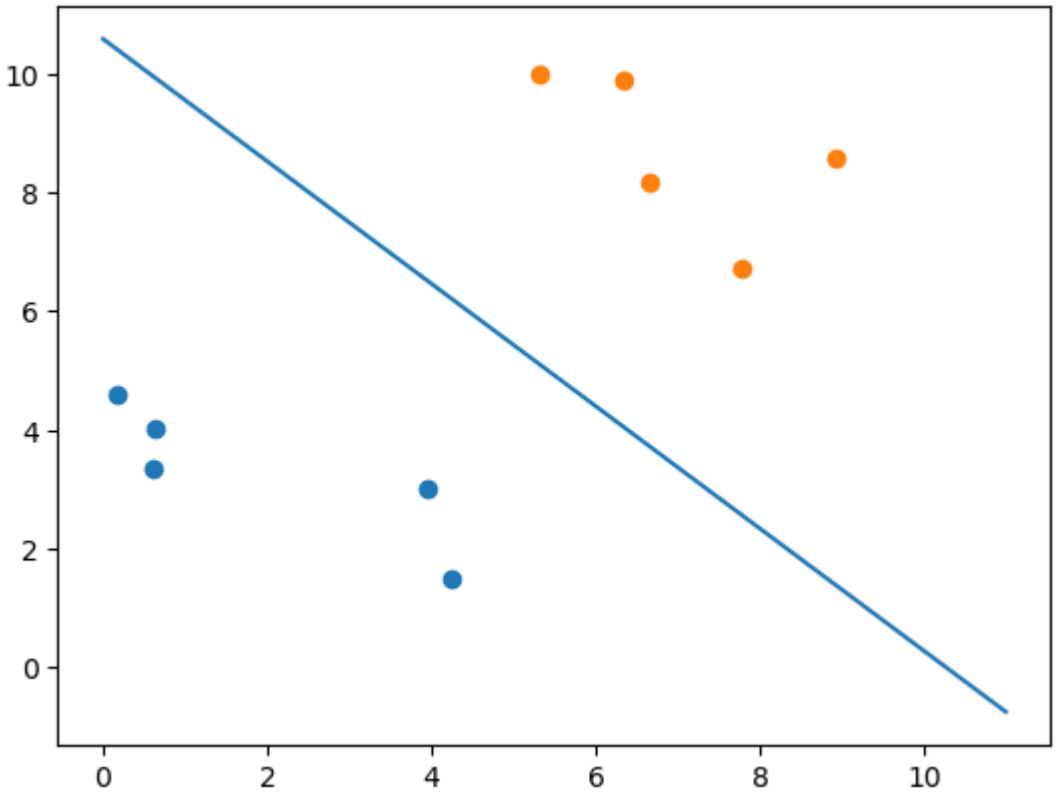
下面我们试着对这个程序进行一下改进

首先是采样点数多于2的时候

In [34]:
```
# 导入库
import numpy as np
import matplotlib.pyplot as plt
from qpsolvers import solve_qp
from cvxopt import matrix, solvers

# 生成数据
n1 = 5
n2 = 5
x1 = np.zeros([2, n1])
x2 = np.zeros([2, n2])
d = np.zeros([1, n1 + n2])
for i in range(0, np.shape(x1)[1]):
    x1[0, i] = np.random.uniform(0, 5, size=None)
    x1[1, i] = np.random.uniform(0, 5, size=None)
for j in range(0, np.shape(x2)[1]):
    x2[0, j] = np.random.uniform(5, 10, size=None)
    x2[1, j] = np.random.uniform(5, 10, size=None)
for k in range(0, n1 + n2):
    if k <= np.shape(x1)[1] - 1:
        d[0, k] = -1
    else:
        d[0, k] = 1
x = np.hstack((x1, x2))

# 创建矩阵
dd = np.dot(d.T, d)
xx = np.zeros([n1 + n2, n1 + n2])
for i in range(0, n1 + n2):
    for j in range(0, n1 + n2):
        xx[i, j] = np.dot(x[:, i].T, x[:, j])
# 求解
P = matrix(dd*xx)
q = matrix(-np.ones([n1 + n2, 1]))
G = matrix(-np.eye(n1+n2))
h = matrix(np.zeros([n1+n2,1]))
A = matrix(d)
b = matrix(np.zeros([1, 1]))

solution = solvers.qp(P, q, G, h, A, b)
alphas = np.ravel(solution['x'])

# 计算参数
w = np.zeros([2, 1])
b = np.zeros([1, n1+n2])
for j in range(0, n1 + n2):
    w[0, 0] = w[0, 0] + alphas[j]*d[0, j]*x[0, j]
    w[1, 0] = w[1, 0] + alphas[j]*d[0, j]*x[1, j]
for j in range(0, n1 + n2):
    b[0, j] = 1/d[0, j]-np.dot(x[:, j],w)

b = np.sum(b)/(n1+n2)

c1 = -w[0]/w[1]
c2 = -b/w[1]
print(w)
print(b)

t = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

# 显示数据
```

```
plt.figure()
plt.scatter(x1[0, :], x1[1, :])
plt.scatter(x2[0, :], x2[1, :])
plt.plot(t, c1*t+c2)
plt.show()
```

```
[[0.26981704]
 [0.26154908]]
-2.7706431896876618
```



这里需要注意的事情是solvers.qp求解器的用法，必须把所有限制条件都加上，不然可能会报错

当然这里除了cvxopt求解器，还可以用qpsolvers求解器来求解

In [35]:
```
# 导入库
import numpy as np
import matplotlib.pyplot as plt
from qpsolvers import solve_qp

# 生成数据
n1 = 5
n2 = 5
x1 = np.zeros([2, n1])
x2 = np.zeros([2, n2])
d = np.zeros([1, n1 + n2])
for i in range(0, np.shape(x1)[1]):
    x1[0, i] = np.random.uniform(0, 5, size=None)
    x1[1, i] = np.random.uniform(0, 5, size=None)
for j in range(0, np.shape(x2)[1]):
    x2[0, j] = np.random.uniform(5, 10, size=None)
    x2[1, j] = np.random.uniform(5, 10, size=None)
for k in range(0, n1 + n2):
    if k <= np.shape(x1)[1] - 1:
        d[0, k] = -1
    else:
        d[0, k] = 1
x = np.hstack((x1, x2))

# 创建矩阵
dd = np.dot(d.T, d)
xx = np.zeros([n1 + n2, n1 + n2])
for i in range(0, n1 + n2):
    for j in range(0, n1 + n2):
        xx[i, j] = np.dot(x[:, i].T, x[:, j])
```

使用qpsolvers和前面的程序也没什么不一样，就是求解这里发生了变更

In [36]:
```
# 求解
P = dd*xx
q = -np.ones([n1 + n2, 1])
G = -np.eye(n1+n2)
h = np.zeros([n1+n2,1])
A = d
b = np.zeros([1, 1])

alphas = solve_qp(P, q, G, h, A, b, solver='cvxopt')
```

```
# 计算参数
w = np.zeros([2, 1])
b = np.zeros([1, n1+n2])
for j in range(0, n1 + n2):
     w[0, 0] = w[0, 0] + alphas[j]*d[0, j]*x[0, j]
     w[1, 0] = w[1, 0] + alphas[j]*d[0, j]*x[1, j]
for j in range(0, n1 + n2):
     b[0, j] = 1/d[0, j]-np.dot(x[:, j],w)

b = np.sum(b)/(n1+n2)

c1 = -w[0]/w[1]
c2 = -b/w[1]
print(w)
print(b)
```

```
print(c1)
print(c2)
t = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

# 显示数据
plt.figure()
plt.scatter(x1[0, :], x1[1, :])
plt.scatter(x2[0, :], x2[1, :])
plt.plot(t, c1*t+c2)
plt.show()
```

以上就是我们这第一次课的全部内容了，谢谢大家。

SVM部分还有核方法和软边界问题，但是核心内容就是上面的求解二次约束问题。

剩下的部分我们下节课再讲解。

关于考核：

考核会以报告的形式提交，希望各位同学都提供给我Jupyter Notebook形式的作业，这样很方便，比较一目了然，也能最大程度上的避免抄袭。

考核标准呢，我们目前定了四个方面，分别是：

（1）准确性：就是你的程序得是对的，能正常运行，而且运行的结果也是可接受的；

（2）规范性：Python编程的一些规范，比如注释，缩进这些，简单来说就是可读性要好，别人拿到你的程序，简单看一看就能知道你的程序各部分的功能；

（3）泛用性：就是你的程序能够应对尽可能多的情况，比如我们那个判断素数的程序，你输入了一个负数或者小数，那这个时候怎么办；再比如我们的回归模型，如果数据维数出现变化，你的程序能不能应对，等等。

最后这一条的要求呢，可能不够明确，因为每个人对泛用性的理解都不太一样，所以各位同学如果觉得最后一条很难实现，那么尽量满足前两条就好了。

当然也欢迎同学们提出一些宝贵意见。

做不出来不要紧，我这边第一次上课，难免讲得不好，所以考核上会适当放松一些。

当然教学任务目标还是要达到的，得保证让同学们掌握一门技能。

如果有讲得不清楚的，或者觉得有困难的地方，欢迎同学们来问。

总结，让我们回顾一下本次课的教学内容：

(1)对本门课程的实验内容、涉及的数学方法以及相关的数学操作进行了简单的归类。

(2)细化了不同类别实验内容涉及的关键编程操作以及常用Python库与命令，包括：导入库，数据读取，循环操作，判断，画图等等。

(3)开展了基于Python语言的回归模型与分类器的实现，并从准确性，规范性，泛用性等方面对Python程序进行了评估。

作业：

回归模型以及分类器模型的可控练习，就是我们在课堂上练习的，最理想情况下的回归模型和分类器练习，这个要写到报告里面去。

能够实现这一点，说明各位同学起码掌握了最基本的关于回归模型以及分类器的使用方法与编程技巧。

当然也请注意我们的编程要求。

之后就是实际练习，从网上找的结果，进行一下回归模型以及分类器的编程