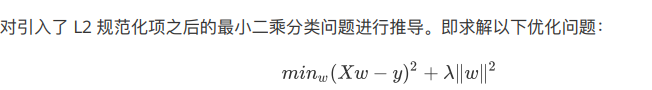
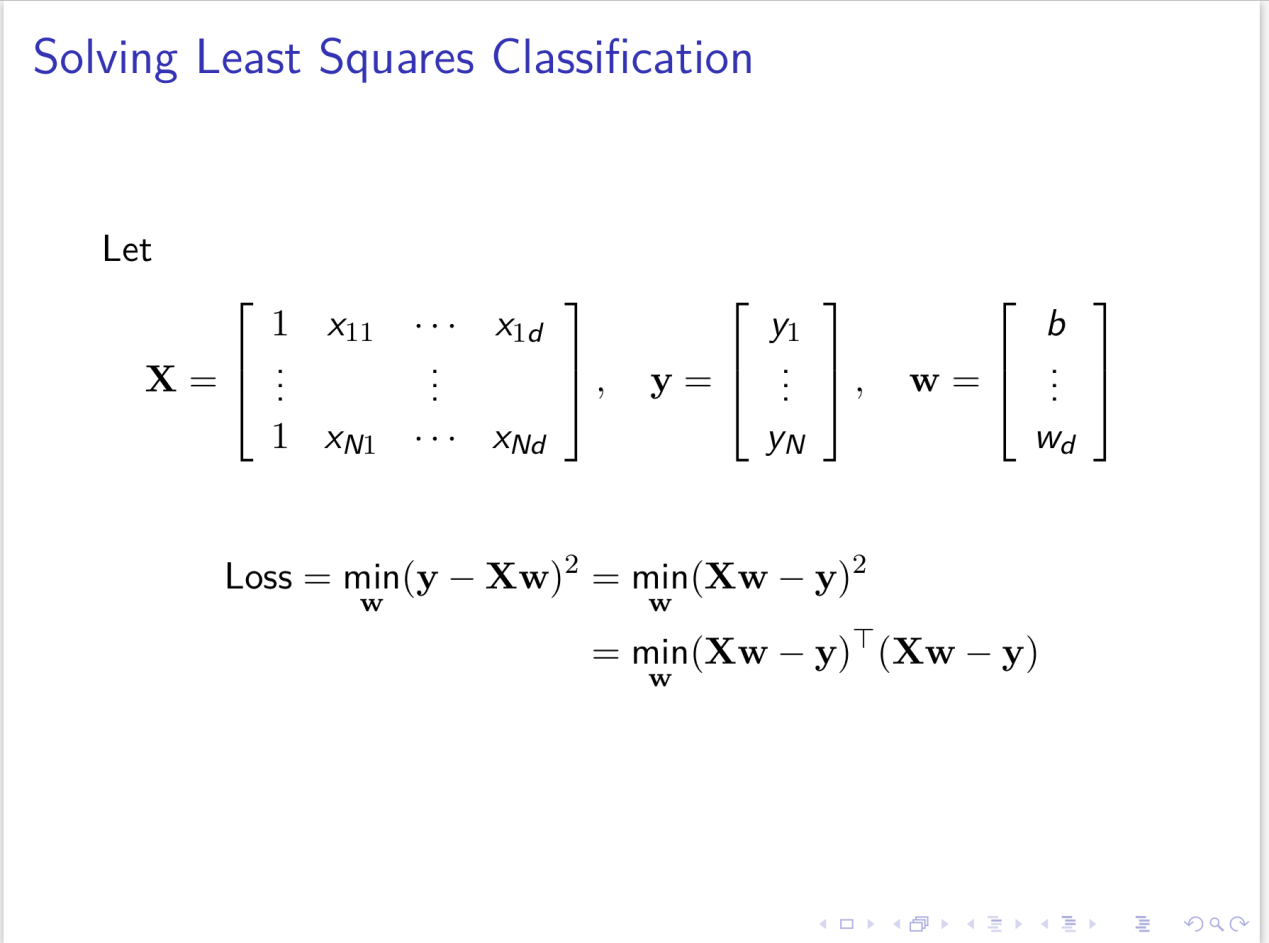
人工智能导论 第二次实验 实验报告

**1：linearclassification**

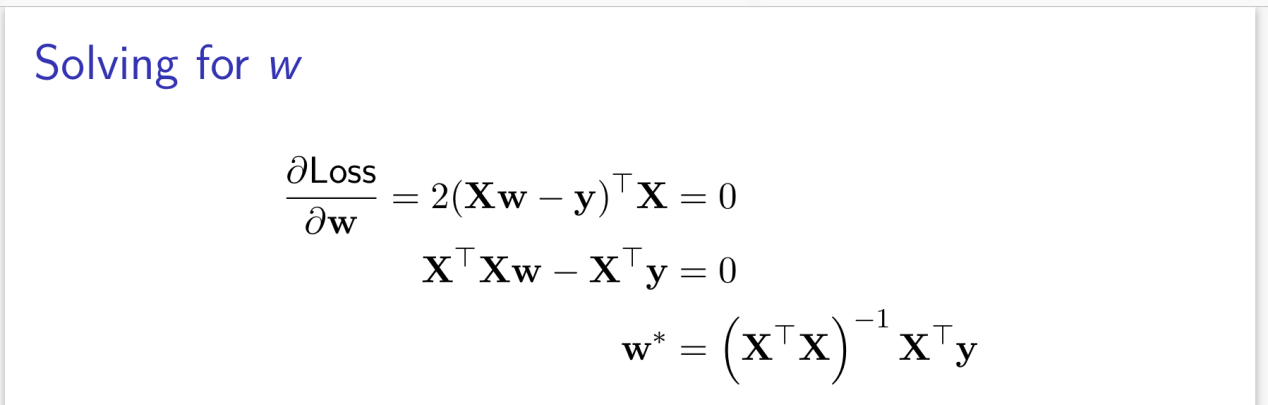
训练方法：采用线性回归模型学习：



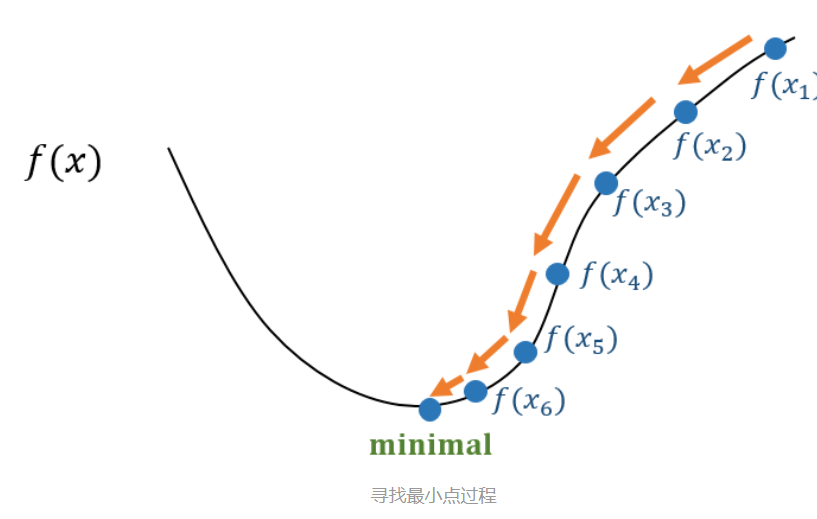
迭代更新方法：梯度下降法（来自吉建民老师的PPT），下图为目标函数，（缺少了正则项）



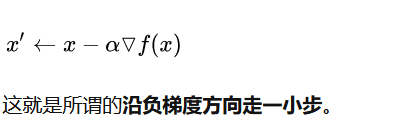
梯度表达式：同样来自吉老师PPT，同样缺少了正则项（2\*lamda\*omega）



梯度下降法的原理：下图来自百度-知乎-梯度下降法



因此，根据梯度，我们可以写出参数迭代的表达式：



其中x即为我们本题中的omega（w）参数。

将以上分析过程，用代码实现，即为linearclassification的train（fit）部分：

def loss(self,features,labels,w):

        attachment = np.ones(features.shape[0])

        X = np.c\_[attachment,features]

        # X为原features首列改变成1的结果

        temp = np.dot(X,w)

        # print((temp-labels).shape)

        return int(np.dot((temp-labels).reshape(1,-1),temp-labels) + self.Lambda\*np.dot(w.reshape(1,-1),w))

    '''根据训练数据train\_features,train\_labels计算梯度更新参数W'''

    def fit(self,train\_features,train\_labels):

        ''''

        需要你实现的部分

        '''

        attachment = np.ones(train\_features.shape[0])

        print(attachment.shape,train\_features.shape)

        X = np.c\_[attachment,train\_features]

        w = np.zeros(train\_features.shape[1] + 1)

        w = w.reshape(-1,1)

        # w的初值为0

        fit\_epochs = self.epochs

        while fit\_epochs > 0 :

            # print(w)

            # print(self.loss(train\_features,train\_labels,w))

            fit\_epochs = fit\_epochs - 1

            # temp = X\*w-train\_labels

            # print(X)

            temp = np.dot(X,w)

            temp = temp - train\_labels

            # print(temp)

            temp = np.dot(temp.reshape(temp.shape[1],temp.shape[0]),X)

            # print(temp)

            grad = 2\*temp + 2\*self.Lambda\*w.reshape(1,-1)

            # print(grad)

            # 计算梯度

            w = w - self.lr\*grad.reshape(-1,1)

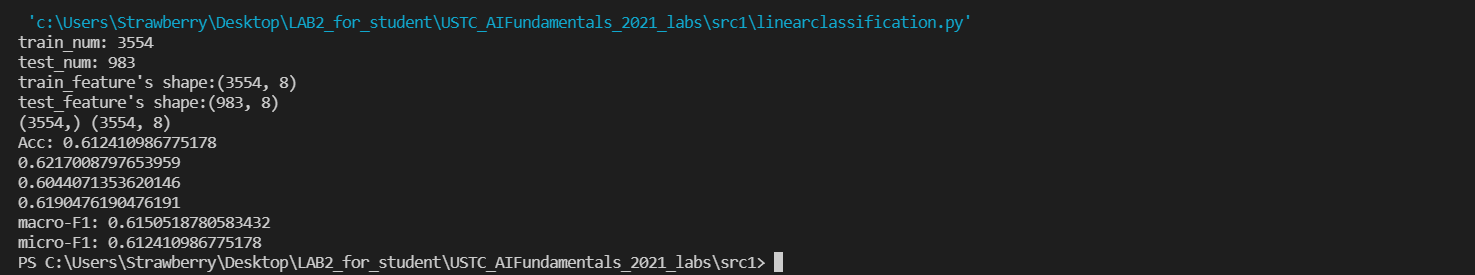
        # print(w)

        self.w = w

至于预测部份，即按照我们训练出来的参数w，根据test\_data，计算其相应的线性函数值（注意：这里我们计算的值要转化为离散的labels：1，2，3；转化方法非常自然：f（x）>2.5时，标记为label-3；f（x）<1.5时，标记为label-1；f（x）介于1.5与2.5之间时，标记为label-2；即可）

根据predict\_labels与test\_labels的差距，统计acc（准确率）等指标输出。

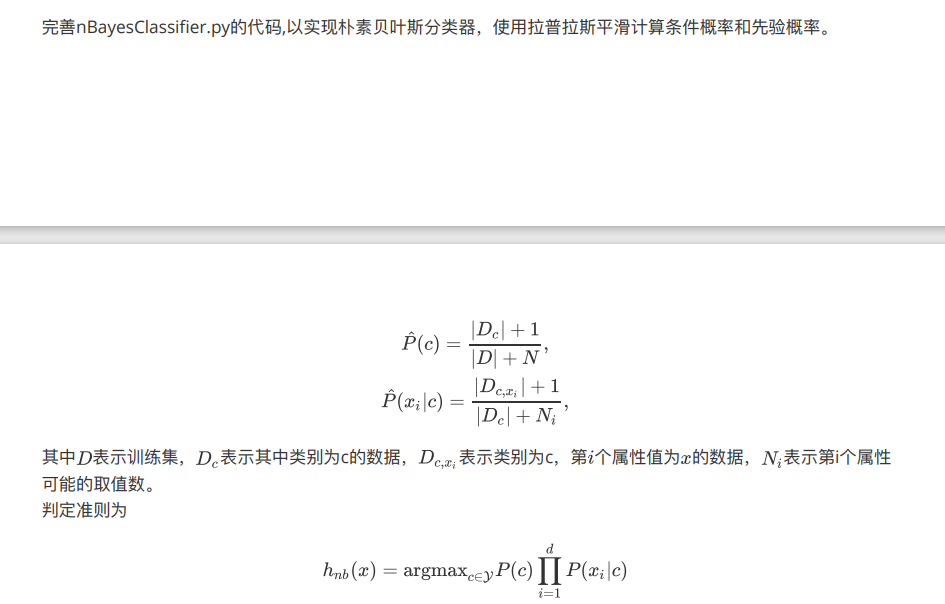
预测结果如下图所示：

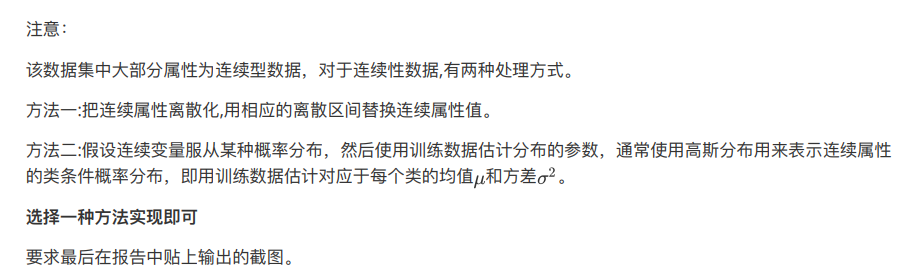


可见预测成功率达到了61.2%左右。

**2：NaiveBayesian**

实验原理如下：

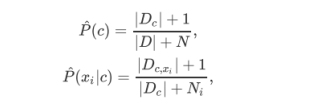




本题我们采用对feature[0]使用离散计算频率（拉普拉斯平滑处理）的方法，对feature[1..7]采用使用高斯分布（正态分布）拟合（此时不需要考虑拉普拉斯平滑处理）的方法。

本质上，我们的训练过程即为：

1：统计相关信息阶段，遍历整个训练数据集（train\_data)，统计各个categories的数量，统计feature[0]取值为1，2，3时各个categories的数量，统计feature[1..7]在不同categories分类的子集下的数据集（subset\_array）；

2：拟合阶段，Pc和对feature[0]的条件概率都还是按照

来进行计算。其中Pxc[(i,0,j)]表示在第i个categories（i = 1，2，3）中，feature[0] = j（j = 1，2，3）出现的概率；Pc[c]表示categories c 出现的概率。

针对连续性特征feature[1..7]，我们统计各个subset\_array的均值（mean）和标准差（s\_d);

然后把这一组参数特征（唯一地决定了高斯分布的表达式）赋值给Pxc[(i,j)]表示第i个categories中第j（j = 1，2，3...7）个属性的分布参数情况。

然后，训练阶段（fit）完成。

代码如下：

 def fit(self,traindata,trainlabel,featuretype):

        '''

        需要你实现的部分

        '''

        # 对于连续的数据，我们采用第二种方式（使用高斯分布拟合的方法）来实现

        # 先统计c1，c2，c3的概率

        # 遍历所有训练数据

        # 我们需要统计

        '''

            1:number of category

            2:建立subset array with a certain category and a certain feature

            3:for discrete feature[0],我们应该统计pxc

        '''

        num\_c = {1:0,2:0,3:0}

        num\_c\_feature = {(1,1):0,(1,2):0,(1,3):0,(2,1):0,(2,2):0,(2,3):0,(3,1):0,(3,2):0,(3,3):0}

        subset\_array\_dict = {}

        subset\_array\_cnt = {}

        for i in range(1,4):

              for j in range(1,8):

                  subset\_array\_cnt[(i,j)] = 0

        for i in range(traindata.shape[0]):

            num\_c[int(trainlabel[i])] += 1

            #更新相应种类训练数据数量数

            num\_c\_feature[(int(trainlabel[i]),int(traindata[i][0]))] += 1

            #为离散特征feature0和不同种类统计训练数据数量

            for j in range(1,8):

                if subset\_array\_cnt[(int(trainlabel[i]),j)] == 0:

                    #建立数组sub\_array

                    subset\_array\_dict[(int(trainlabel[i]),j)] = np.array(float(traindata[i][j]))

                    subset\_array\_cnt[(int(trainlabel[i]),j)] += 1

                else:

                    subset\_array\_dict[(int(trainlabel[i]),j)] = np.append(subset\_array\_dict[(int(trainlabel[i]),j)],float(traindata[i][j]))

        # debug

        # print(subset\_array\_dict)

        # 计算PC

        for i in range(1,4):

            self.Pc[i] = (num\_c[i]+1)/(num\_c[1]+num\_c[2]+num\_c[3]+3)

        # 计算Px[0]c

        for i in range(1,4):

            for j in range(0,8):

                if j == 0:

                    # 离散条件概率

                    for k in range(1,4):

                        self.Pxc[(i,j,k)] = (num\_c\_feature[i,k]+1)/(num\_c[i]+3)

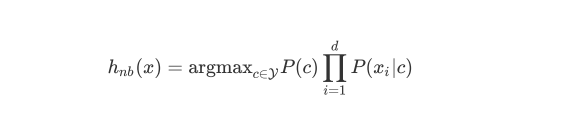
                else:

                    # 连续条件概率

                    self.Pxc[(i,j)] = self.mean\_and\_standard\_deviation(subset\_array\_dict[(i,j)])

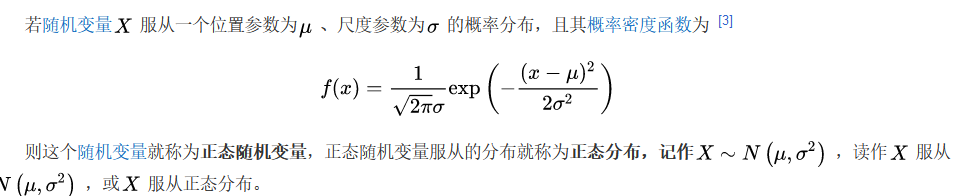
我们进入预测阶段：（predict）

我们即按照公式：



去计算等式右侧的各个取值，取其最大值对应的参数，即为应该预测的分类c。

（这里注意，针对P（xi|c）中连续变量的情形，我们使用高斯分布的概率密度来代替概率取值，由于我们只是比较各个c取值时右侧的相对大小，所以使用概率密度并无影响）



代码如下:

def predict(self,features,featuretype):

        '''

        需要你实现的部分

        '''

        pred = []

        test\_num = features.shape[0]

        for k in range(test\_num):

            max = 0

            c\_predict = 0

            probabilities = []

            for c in range(1,4):

                temp = self.Pc[c]

                temp \*= self.Pxc[(c,0,int(features[k][0]))]

                for i in range(1,8):

                    (mean,s\_d) = self.Pxc[(c,i)]

                    p = self.norm\_distribution\_function(mean,s\_d,features[k][i])

                    temp \*= p

                '''

                probabilities.append(temp)

            c\_predict = np.argmax(probabilities)

                '''

                if temp > max:

                    max = temp

                    c\_predict = c

            pred.append(c\_predict)

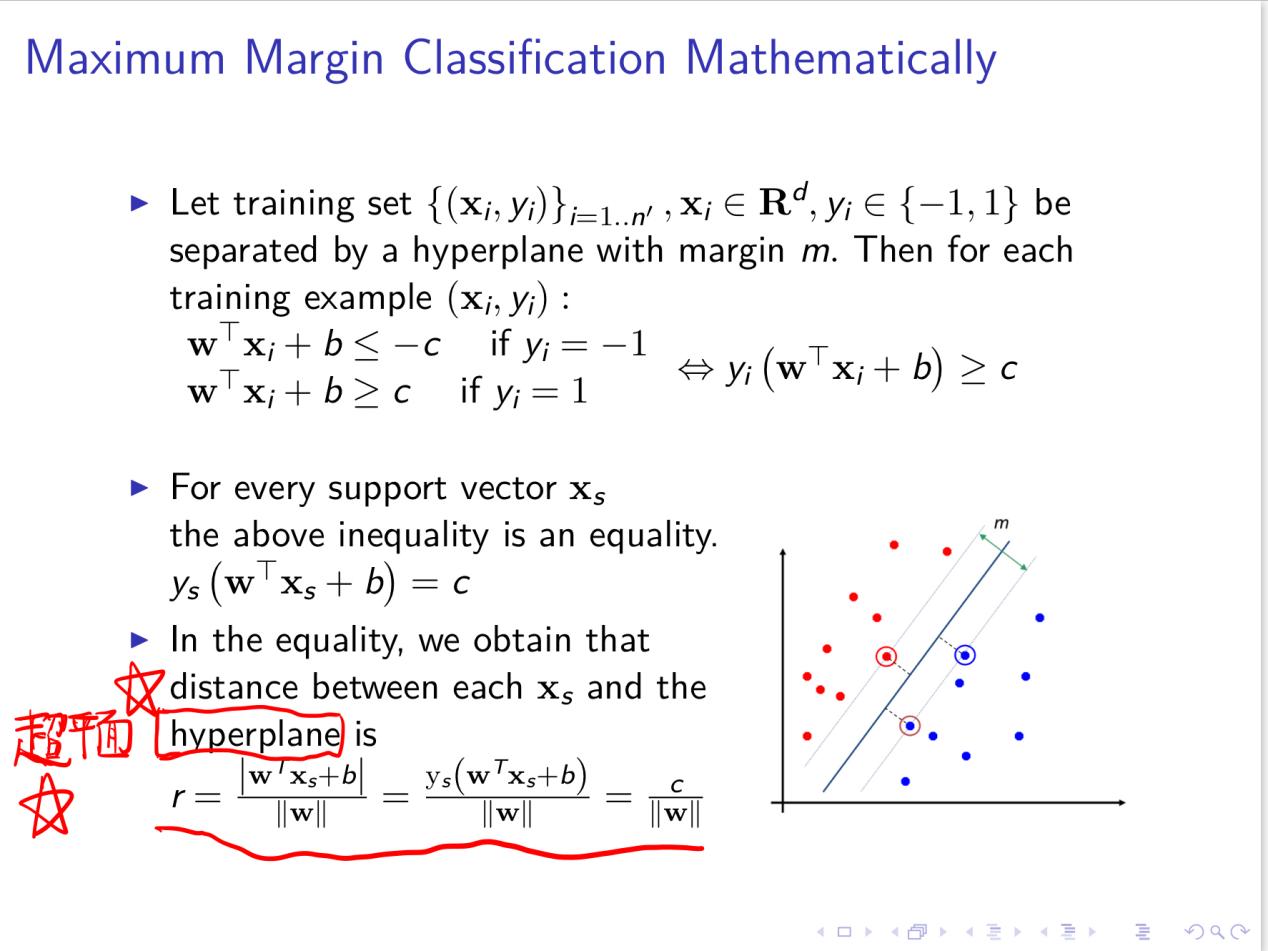
        pred = np.array(pred).reshape(test\_num,1)

        return pred

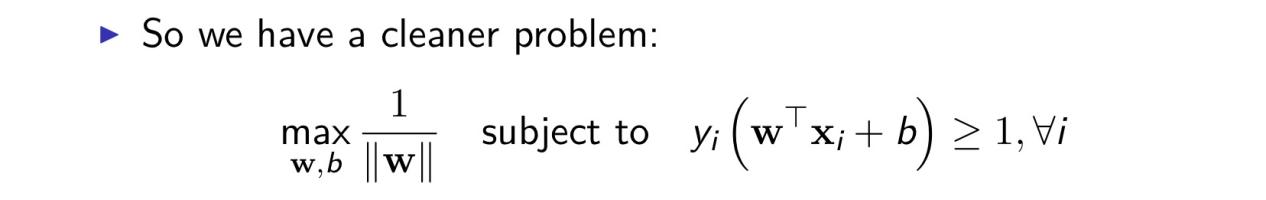
**3：SVM的实现**

SVM的实现原理：

1：动机：最大间距分类：

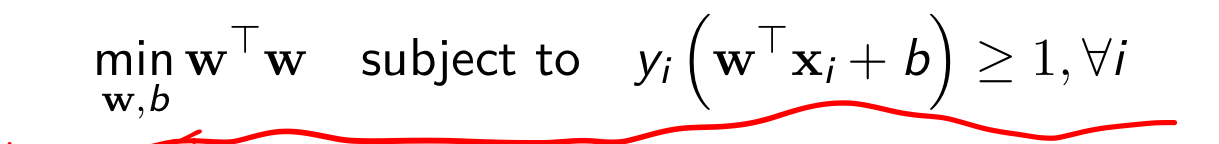


我们通过将条件两侧都除以常数c，得到新的条件和新的目标函数：

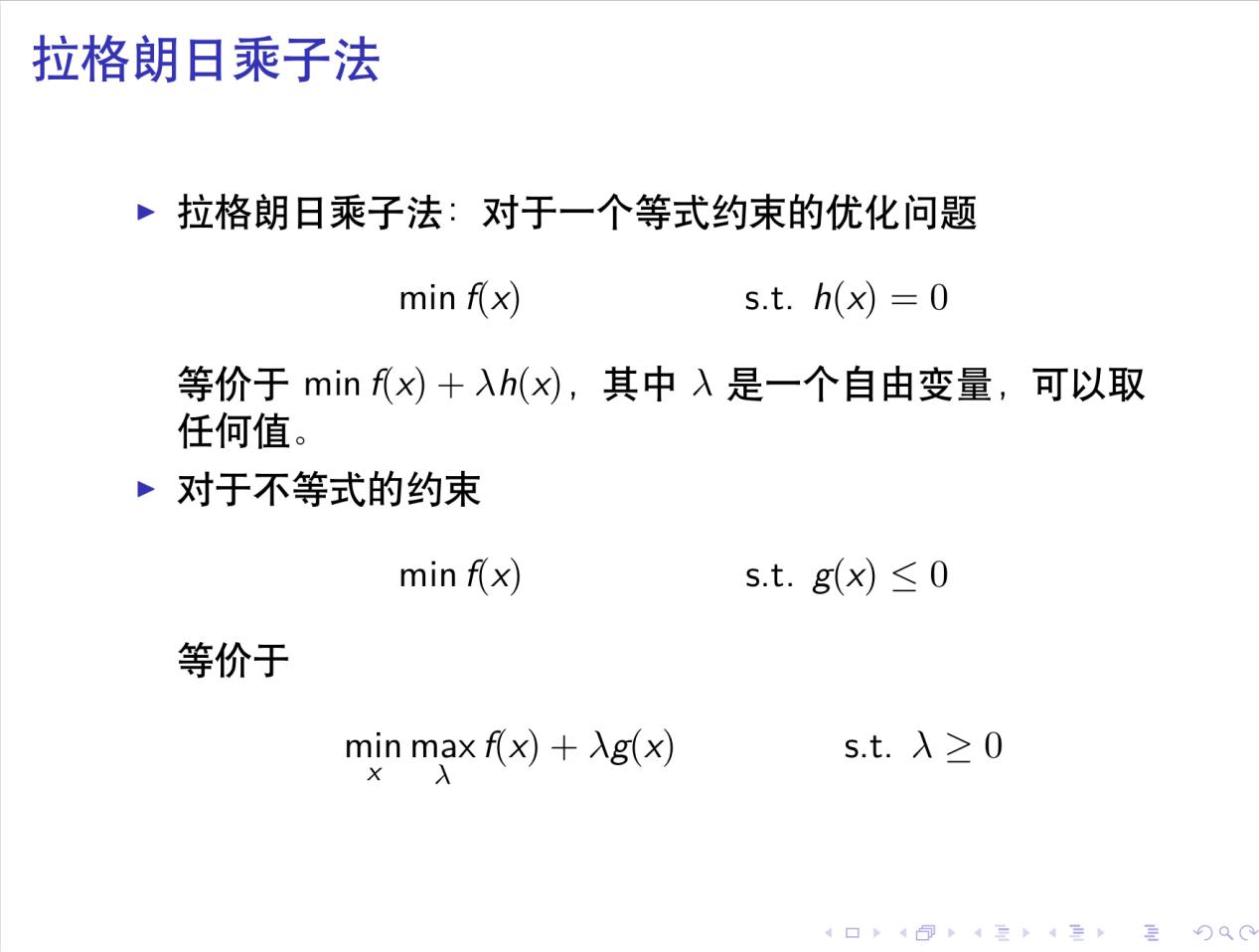


据此，我们考虑使用QP（二次规划）解决这个问题；

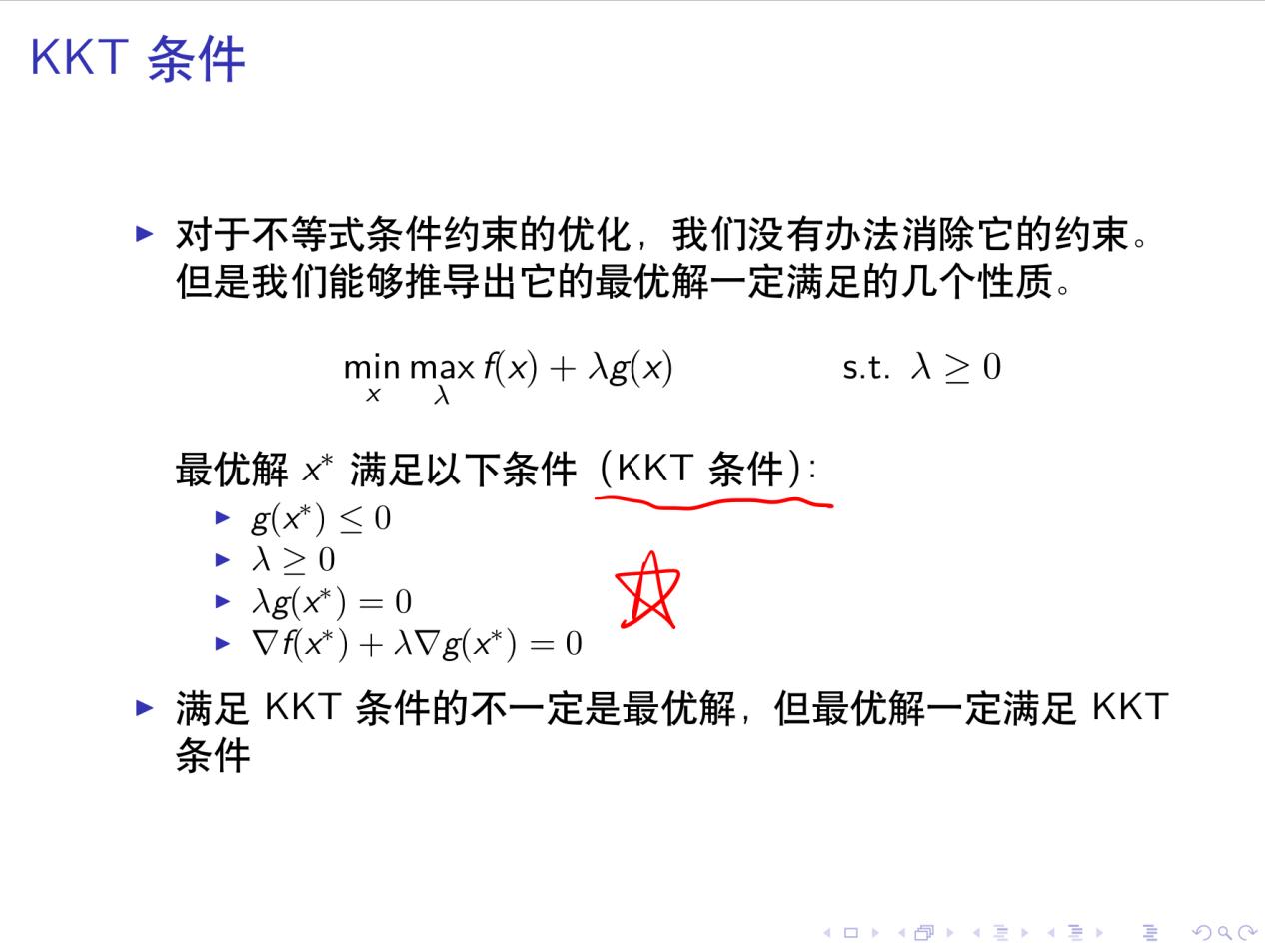
变换目标函数为min形式：



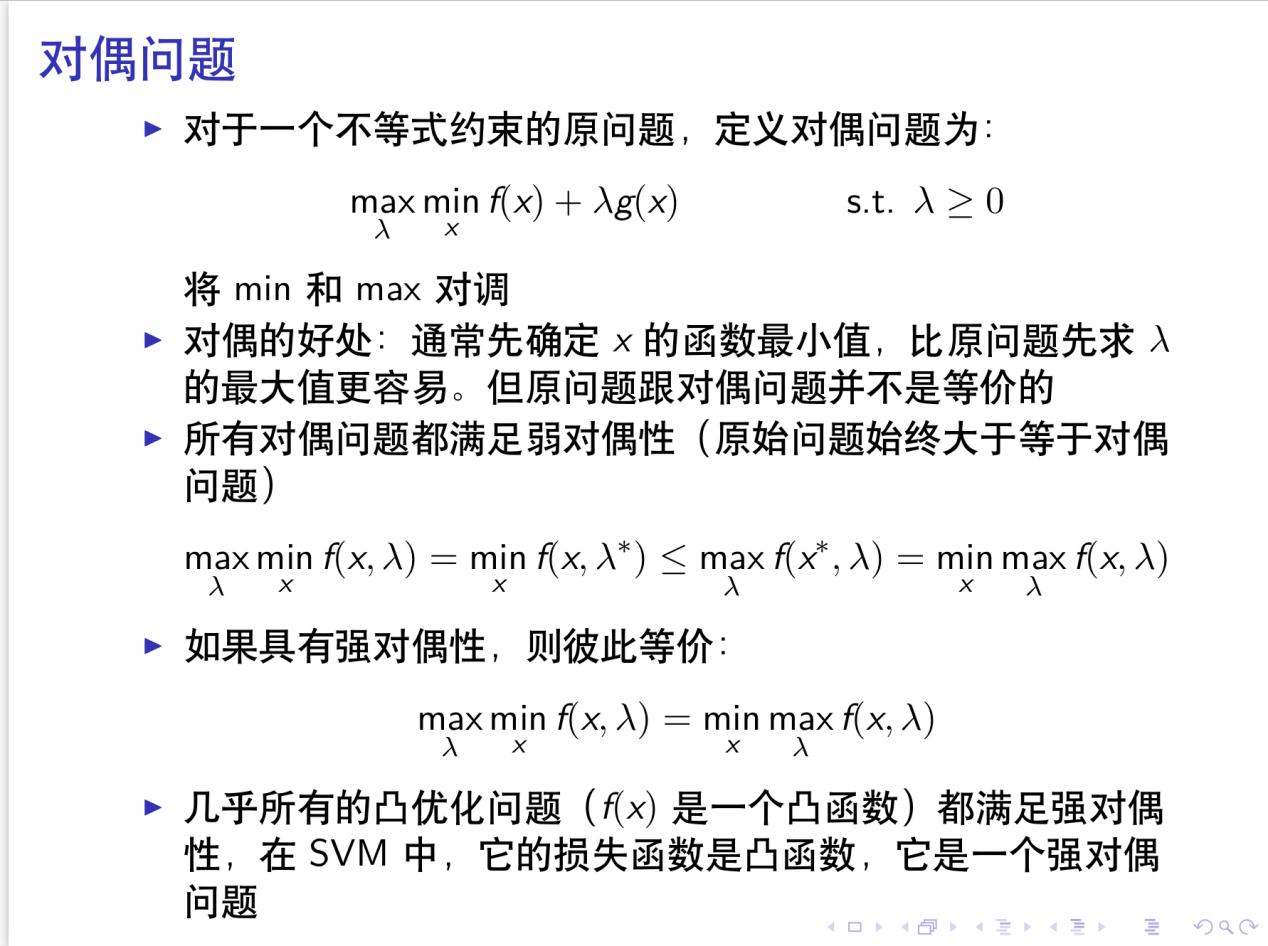
这里我们需要按照拉格朗日乘子法解决问题：



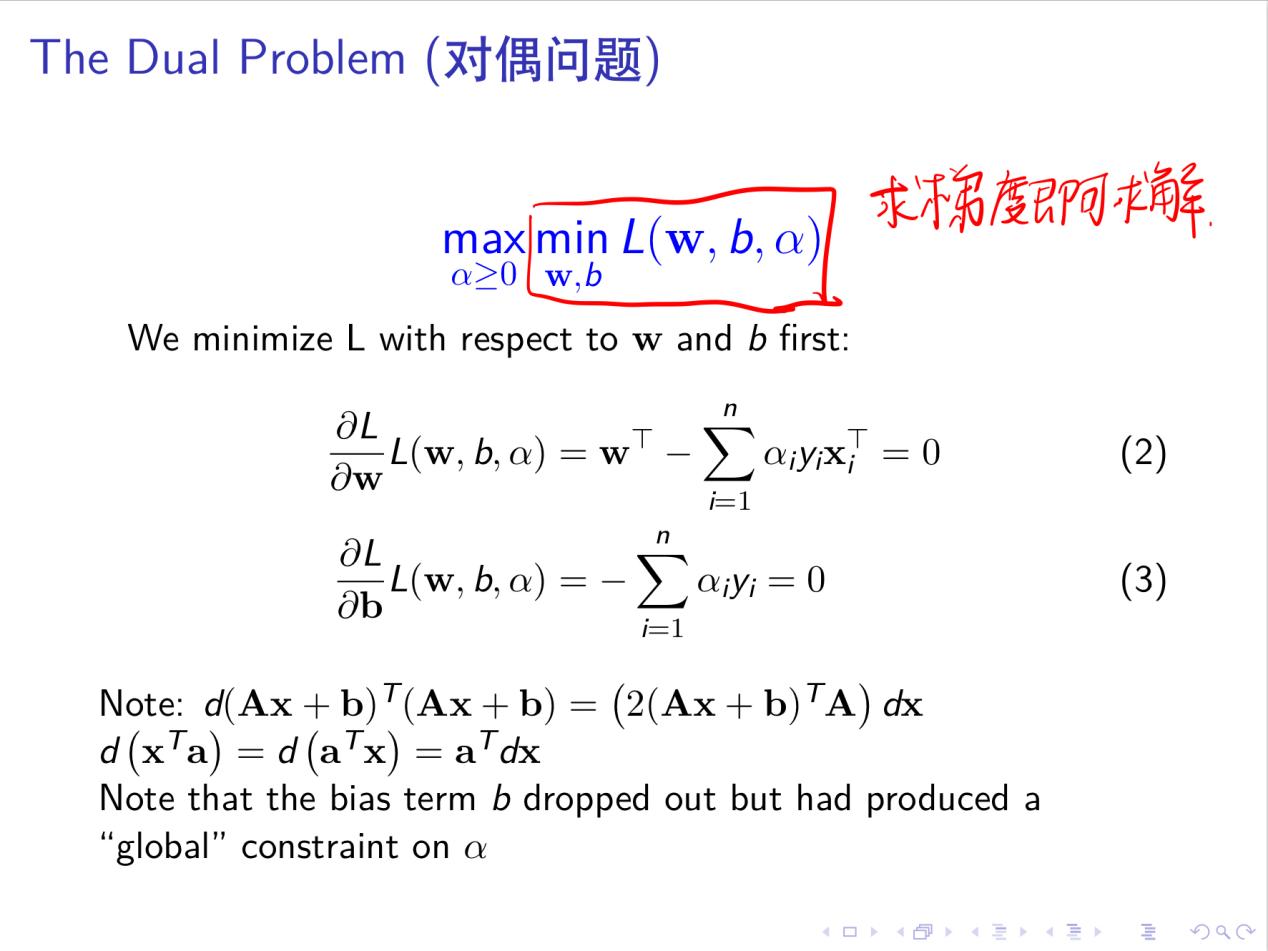
最优解一定需要满足KKT条件：



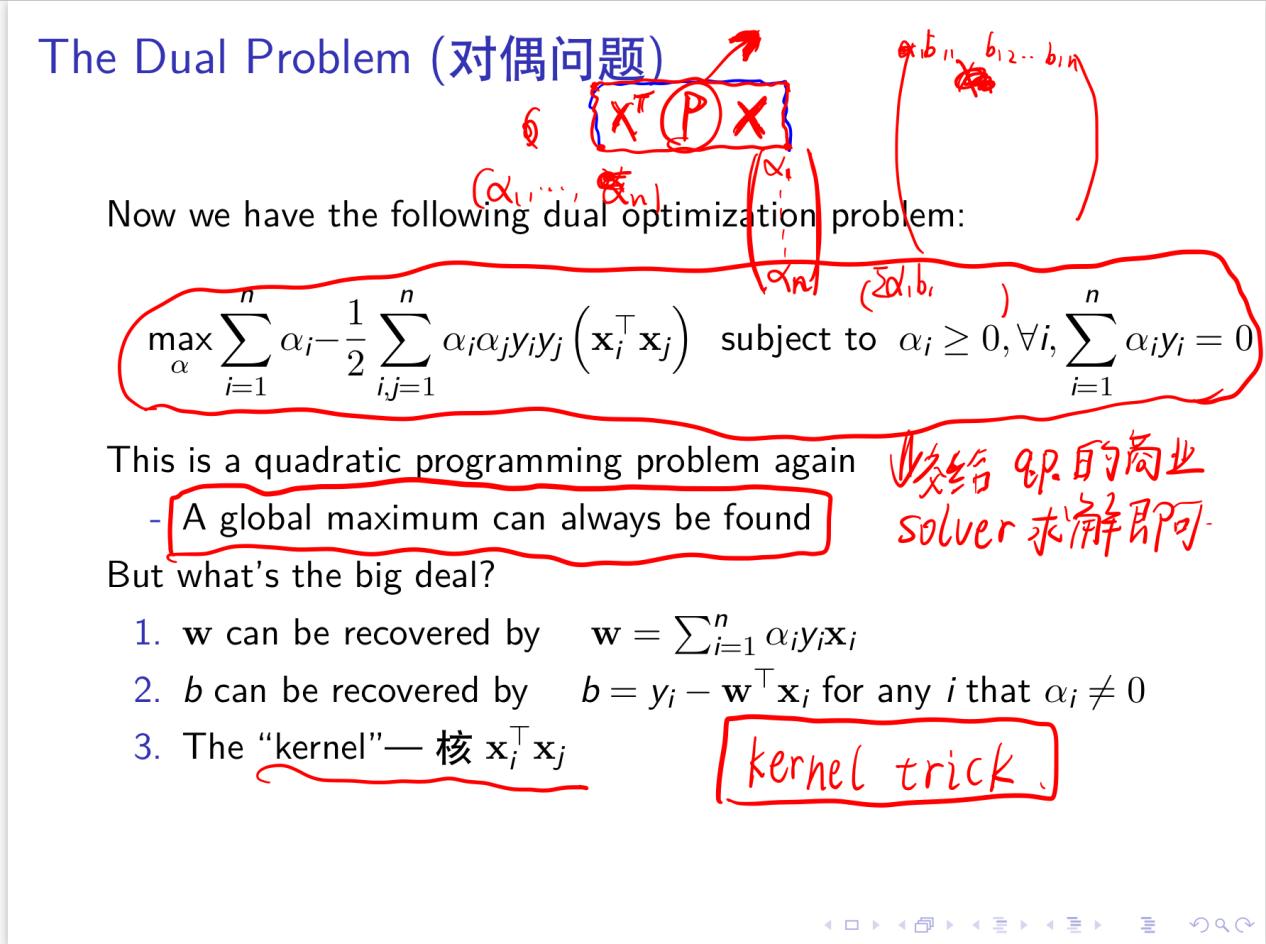
接下来我们根据凸优化问题满足对偶性，将问题转化为：



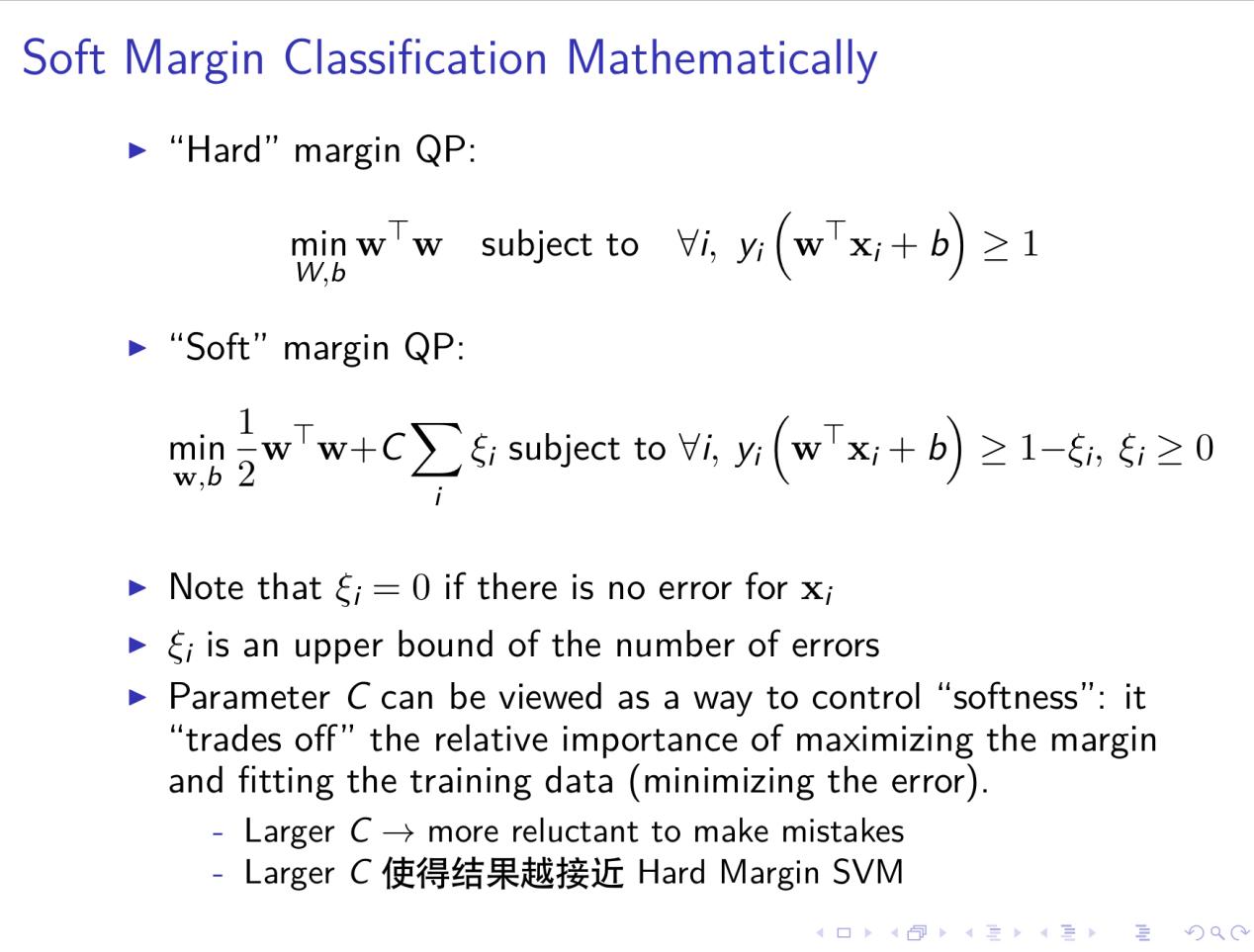
使用求导，求出MAX（MIN（L（w，b, alpha）））中内部的最小值；



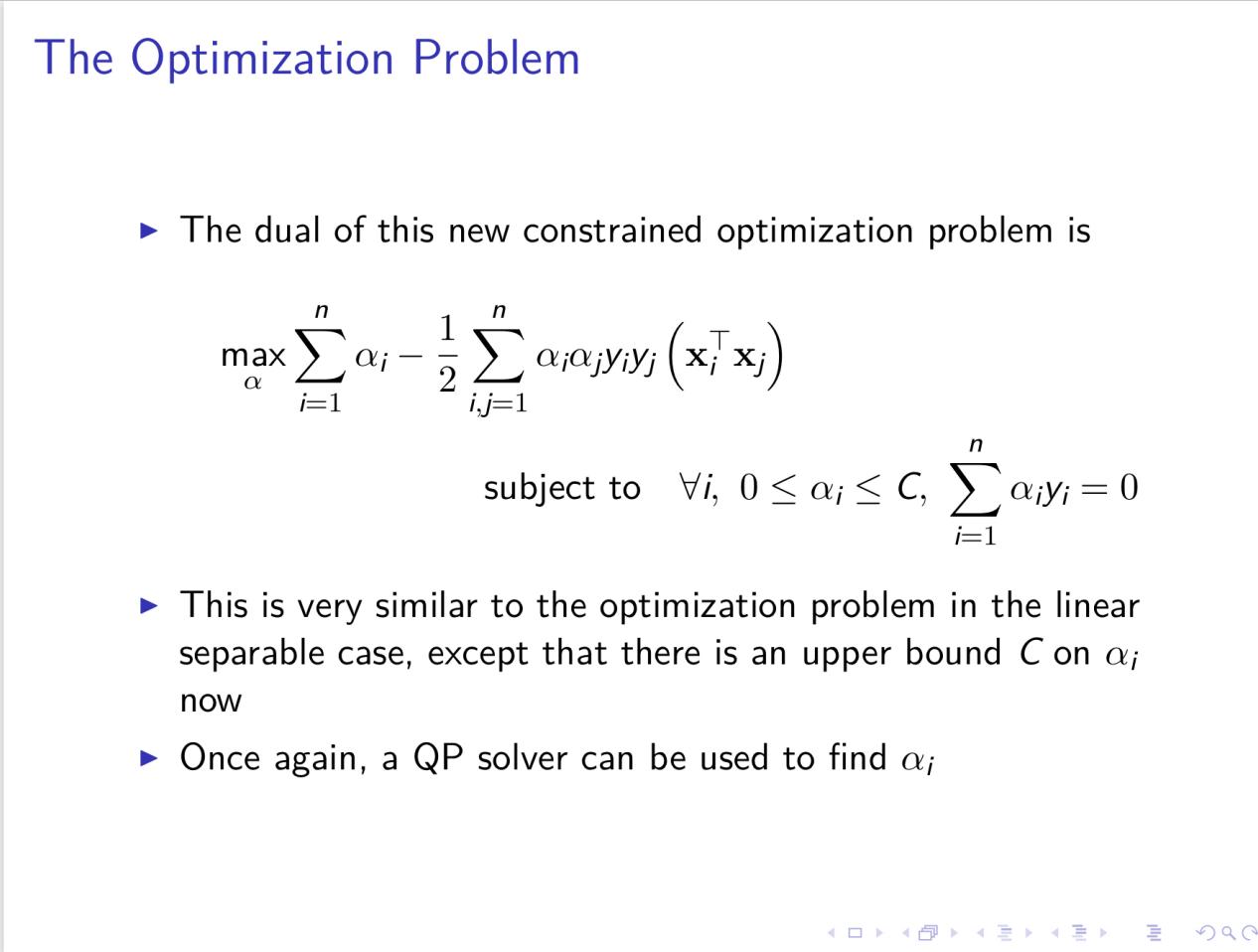
因此，问题转化为：



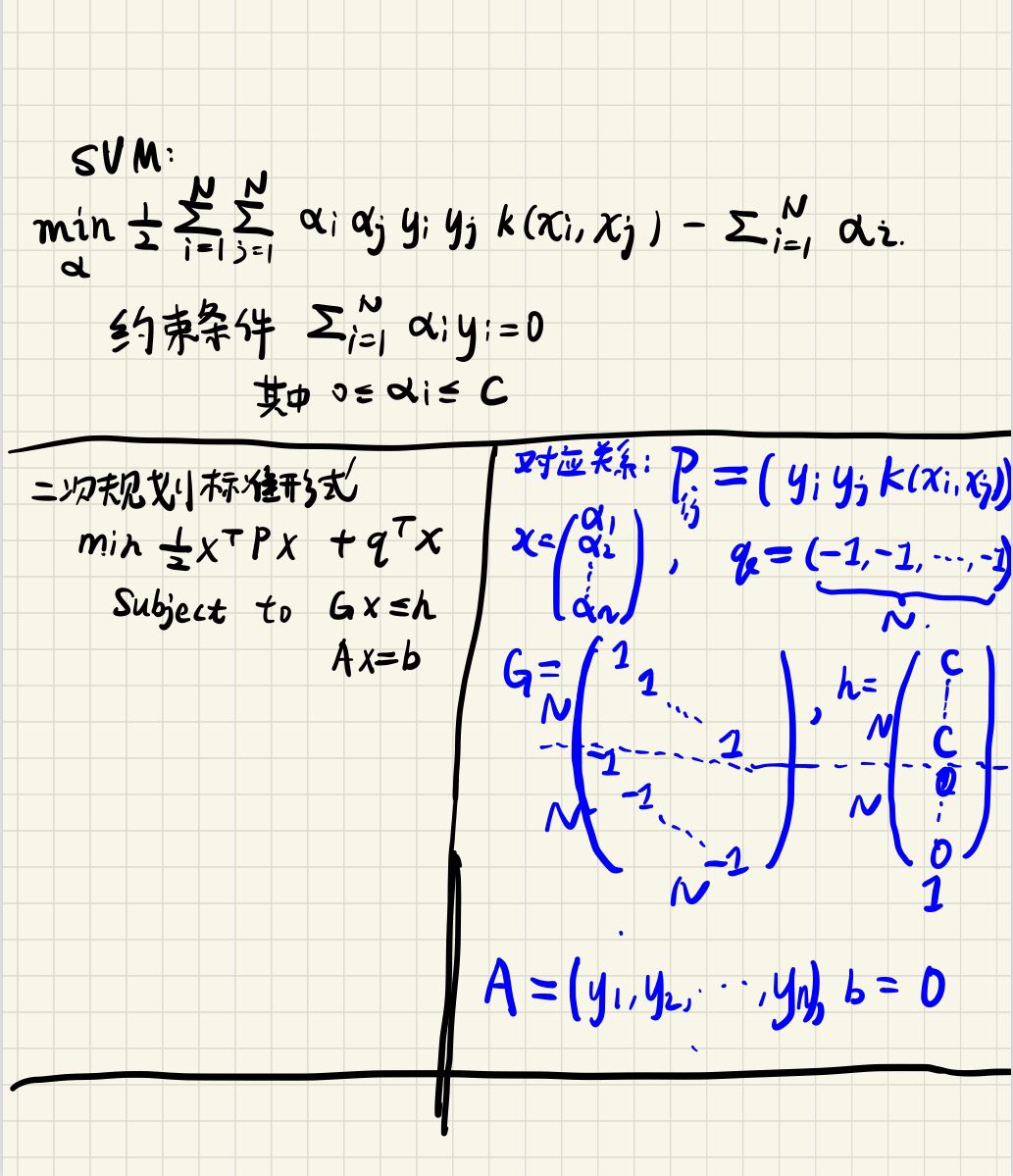
考虑软边界情形：



最终问题转化为对偶形式：



此时我们构造该问题为标准形式：（如下图）



本质上，代码即为上图的翻译：

class SupportVectorMachine:

    '''参数初始化

    lr: 梯度更新的学习率

    Lambda: L2范数的系数

    epochs: 更新迭代的次数

    '''

    def \_\_init\_\_(self,kernel,C,Epsilon):

        self.kernel=kernel

        self.C = C

        self.Epsilon=Epsilon

    '''KERNEL用于计算两个样本x1,x2的核函数'''

    def KERNEL(self, x1, x2, kernel='Gauss', d=2, sigma=1):

        #d是多项式核的次数,sigma为Gauss核的参数

        K = 0

        if kernel == 'Gauss':

            K = np.exp(-(np.sum((x1 - x2) \*\* 2)) / (2 \* sigma \*\* 2))

        elif kernel == 'Linear':

            K = np.dot(x1,x2)

        elif kernel == 'Poly':

            K = np.dot(x1,x2) \*\* d

        else:

            print('No support for this kernel')

        return K

    '''

    根据训练数据train\_data,train\_label（均为np数组）求解svm,并对test\_data进行预测,返回预测分数，即svm使用符号函数sign之前的值

    train\_data的shape=(train\_num,train\_dim),train\_label的shape=(train\_num,) train\_num为训练数据的数目，train\_dim为样本维度

    预测结果的数据类型应为np数组，shape=(test\_num,1) test\_num为测试数据的数目

    '''

    def fit(self,train\_data,train\_label,test\_data):

        '''

        需要你实现的部分

        '''

        # 首先构造系数矩阵P，Pij = <yixi,yjxj>

        train\_num = train\_data.shape[0]

        P = np.zeros((train\_num,train\_num))

        temp = train\_data

        for i in range(train\_num):

            for j in range(train\_num):

                P[i][j] = self.KERNEL(temp[i],temp[j],self.kernel)\*train\_label[i]\*train\_label[j]

        # 构造q

        q = np.ones((train\_num,1))

        q = -1\*q

        # 构造G

        G1 = np.eye(train\_num,dtype = int)

        G2 = np.eye(train\_num,dtype = int)

        G2 = -1\*G2

        G = np.r\_[G1,G2]

        # 构造h

        h1 = np.zeros((train\_num,1))

        for i in range(train\_num):

            h1[i] = self.C

        h2 = np.zeros((train\_num,1))

        h = np.r\_[h1,h2]

        # 构造A

        A = train\_label.reshape(1,train\_num)

        # 构造b

        b = np.zeros((1,1))

        P = P.astype(np.double)

        q = q.astype(np.double)

        G = G.astype(np.double)

        h = h.astype(np.double)

        A = A.astype(np.double)

        b = b.astype(np.double)

        P\_1 = cvxopt.matrix(P)

        q\_1 = cvxopt.matrix(q)

        G\_1 = cvxopt.matrix(G)

        h\_1 = cvxopt.matrix(h)

        A\_1 = cvxopt.matrix(A)

        b\_1 = cvxopt.matrix(b)

        sol = cvxopt.solvers.qp(P\_1,q\_1,G\_1,h\_1,A\_1,b\_1)

        sol\_x = sol['x']

        alpha = np.array(sol\_x)

        indices = np.where(alpha > self.Epsilon)[0]

        bias = np.mean(

            [train\_label[i] - sum([train\_label[i] \* alpha[i] \* self.KERNEL(x, train\_data[i],self.kernel) for x in train\_data[indices]]) for i in indices])

        test\_num = test\_data.shape[0]

        predictions = []

        for j in range(test\_num):

            prediction = bias +  sum([train\_label[i] \* alpha[i] \* self.KERNEL(test\_data[j], train\_data[i],self.kernel) for i in indices])

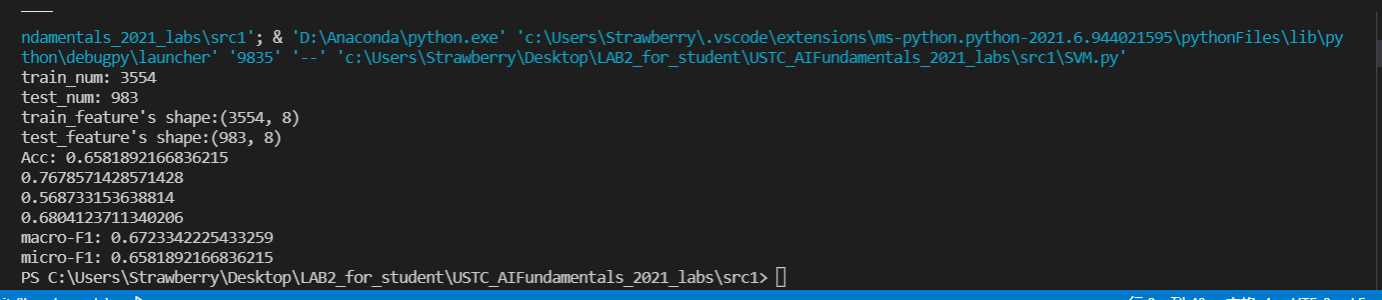
            predictions.append(prediction)

        prediction = np.array(predictions).reshape(test\_num,1)

        # print(prediction)

        return prediction

代码运行结果：



其中预测正确率达到了65%左右。