1. （必填）自己提出的问题的理解（罗列全部）：
2. **提出的问题1：**

希伯特空间与普通的线性空间的区别是什么？

讨论后的理解：

在数学中，希尔伯特空间是欧几里德空间的一个推广，其不再局限于有限维的情形。与欧几里德空间相仿，希尔伯特空间也是一个内积空间，其上有距离和角的概念（及由此引申而来的正交性与垂直性的概念）。此外，希尔伯特空间还是一个完备的空间，其上所有的柯西序列等价于收敛序列，从而微积分中的大部分概念都可以无障碍地推广到希尔伯特空间中。希尔伯特空间为基于任意正交系上的多项式表示的傅立叶级数和傅立叶变换提供了一种有效的表述方式，而这也是泛函分析的核心概念之一。希尔伯特空间是公式化数学和量子力学的关键性概念之一。内积空间+完备性⟶ 希尔伯特空间，其中完备性的意思就是空间中的极限运算不能跑出该空间，如有理数空间中的2–√的小数表示，其极限随着小数位数的增加收敛到2–√，但2–√属于无理数，并不在有理数空间，故不满足完备性。一个通俗的理解是把学校理解为一个空间，从学校内的宿舍中开始一直往外走，当走不动停下来时（极限收敛），发现已经走出学校了（超出空间），不在学校范围内了（不完备了）。希尔伯特就相当于地球，无论你怎么走，都还在地球内（飞出太空除外）。

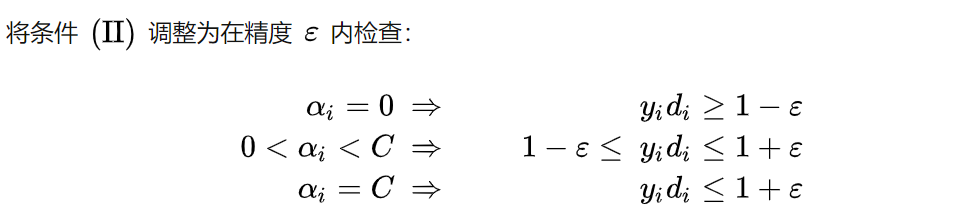
2. **提出的问题2：**

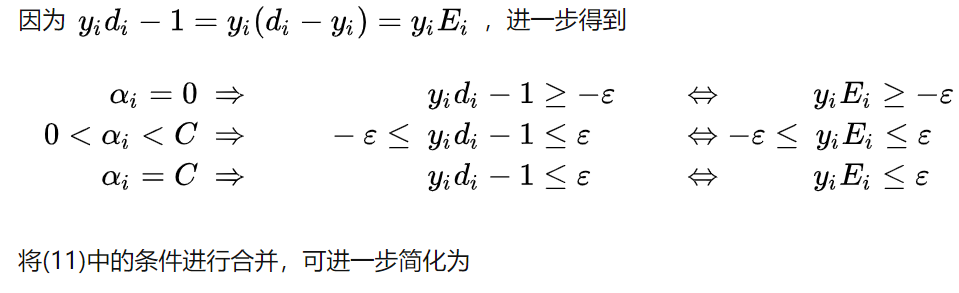
在精度内检查KTT条件的方法是什么？

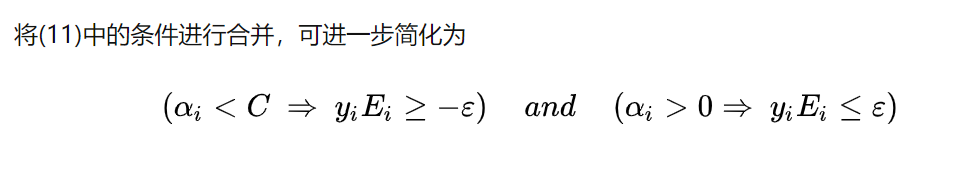
讨论后的理解：

采用在精度 [公式] 内检查KKT条件，一方面的原因是可以放宽精度的要求，加快收敛；另一方面的原因是浮点运算存在舍入误差(个人理解)。

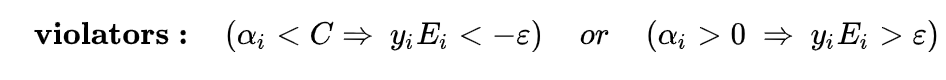
关于在精度 [公式] 内检查KKT条件的具体做法，文献[1-4]均没有具体描述。但是Platt1998和1999给出了SMO的伪代码(pseudo-code)，从其'主程序'和examineExample()函数可以知道其检查KKT条件所使用的语句。下面进行推导。







相反，对违反KKT条件的样本有：

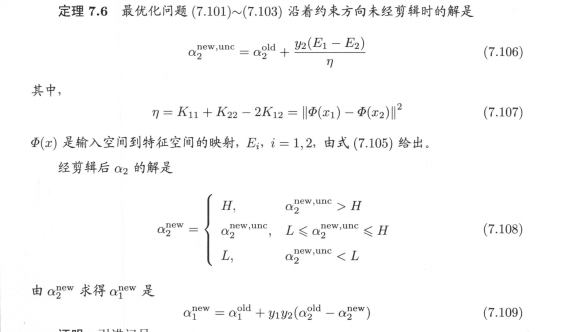


1. （必填）别人提出的问题的理解（选择几个问题罗列，并给出理解）：

1. **问题3**：

为什么α2选择的标准是希望有足够大的变法，以及为什么可以根据|E1-E2|来判断？？

自己的理解：



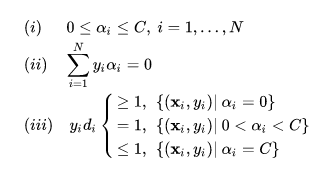
因为根据定理7.6。

a变化的步长由|E1-E2|决定，这个值越大，变化得越快，迭代得就越快。

2. **问题4：**

P49页，SMO算法步骤3,若在精度 范围内....为什么满足这个条件就停机？

自己的理解：



3**. 问题5：**

p144页的y1不等于y2为什么能推出来a1-a2=k？

自己的理解：

因为y1，y2属于{-1，1}，而有y1不等于y2时，y1与y2异号，所以消去可得a1-a2=k

三、（必填）读书计划

1、本周完成的内容章节：统计学习方法（第七章：7.3-7.4）

2、下周计划：统计学习方法（第八章：提升算法）

四、（选做）读书摘要及理解或伪代码的具体实现（读书摘要、伪代码的具体实现代码等可以写到这个部分）

1、读书摘要及理解（选做）

**核函数**

我们可以自己构造一个核函数，但是往往比较复杂，因为我们要证明k是对称的、K是半正定的这两个充分必要条件。所以，我们往往是用已经有的有效的核函数去解决问题。

线性核函数，最基本的核函数，平面上的直线，三维的面，多维的超平面，可以使用对偶svm利用二次规划库直接计算。这是我们第一个需要考虑的，简单有效，但是如果是线性不可分的训练集则无法使用。

多项式核函数，优点是阶数Q可以灵活设置；缺点是当Q很大时，K的数值范围波动很大，而且参数个数较多，难以选择合适的值，超平面是多项式的曲面

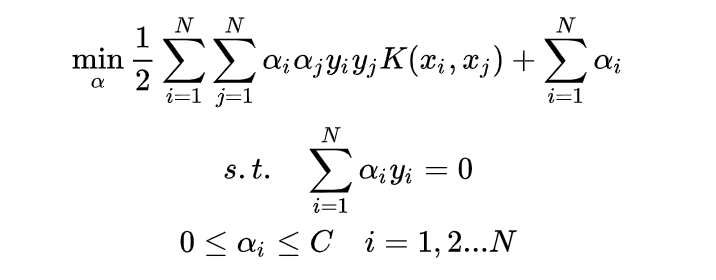
高斯核函数，优点是边界更加复杂多样，能最准确地区分数据样本，数值计算K值波动较小，而且只有一个参数，容易选择；缺点是由于特征转换到无限维度中，w没有求解出来，而且可能会发生过拟合。

实际上核函数代表的是两个样本x和x’，特征变换后的相似性即内积。

**序列最小最优化算法**

支持向量机的学习问题可以形式化为求解凸二次规划问题，这样的凸二次规划问题具有全局最优解。在此介绍序列最小最优化算法（SMO）。

对偶问题



2、代码实现（选做）

def SMOsimple(data,label,C,toler,maxIter):

"""

data:样本各属性值

label：各样本对应标签

C：软间隔最大化的松弛系数对应的惩罚因子，也是约束条件中alpha的上界(对于线性可分数据集，C作用不大；对于线性不可分数据集，结果对C敏感)

toler：容错率，偏离KKT条件的容错率

maxIter：外层循环迭代次数

"""

#初始化alpha=0,b=0,alpha个数为样本数，一个样本对应一个alpha

dataMatrix=np.mat(data);labelMatrix=np.mat(label).transpose() #这里labelMatrix形状为m\*1

b=0;m,n=dataMatrix.shape

alphas=np.mat(np.zeros((m,1)))

iters=0

while iters<maxIter:

alphaPairsChanged=0 #存储每次内循环改变的aplha对数量，每次外循环应该重新置零

for i in range(m): #内循环遍历所有样本点

#计算第i个样本点的预测值gxi和预测误差Ei

gxi=float(np.multiply(alphas,labelMatrix).transpose()\*(dataMatrix\*dataMatrix[i,:].transpose()))+b

Ei=gxi-labelMatrix[i]

"""检验第i个样本点是否满足KKT条件，若满足则会跳出本次内循环(不更新这个alphai)，进行下一次内循环；

若不满足，看它是否是违反KKT条件超过容错率toler的点,若不是，则跳出本次内循环(不更新这个alphai)，进行下一次内循环；

若是，则继续选择alphaj，计算gx,E,eta,进而求得aj解析解，进而求得ai解析解，进而更新b值"""

if (labelMatrix[i]\*Ei<-toler and alphas[i]<C) or (labelMatrix[i]\*Ei>toler and alphas[i]>0):

j=randPickj(i,m)

gxj=float(np.multiply(alphas,labelMatrix).transpose()\*(dataMatrix\*dataMatrix[j,:].transpose()))+b

Ej=gxj-labelMatrix[j]

#存储alpha初始值，用于后续计算

alphaIold=alphas[i].copy()

alphaJold=alphas[j].copy()

#计算剪切边界(很简单的几何计算，见统计学习方法)

if labelMatrix[i]!=labelMatrix[j]:

L=max(0,alphas[j]-alphas[i]) #这里alpha[i]仍然等于alphaIold

H=min(C,C+alphas[j]-alphas[i])

else:

L=max(0,alphas[j]+alphas[i]-C)

H=min(C,alphas[j]+alphas[i])

if L==H:

print ("L==H")

continue #第一个跳出条件(跳出本次内循环，遍历下一个alpha进行更新)

#计算eta

eta=dataMatrix[i,:]\*dataMatrix[i,:].transpose()+dataMatrix[j,:]\*dataMatrix[j,:].transpose()\

-2.0\*dataMatrix[i,:]\*dataMatrix[j,:].transpose()

if eta==0:

print ("eta=0")

continue #第二个跳出条件(因为eta=0不好处理，且出现情况较少，因此这里咱不处理，直接跳出)

#根据统计学习方法中的结果公式得到alphaj的解析解

alphas[j]=alphas[j]+labelMatrix[j]\*(Ei-Ej)/eta

alphas[j]=clipAlpha(alphas[j],H,L)

#检验alphaj与alphaJold是否有足够大的改变，若改变不够大，说明与alpha旧值没有什么差异，跳出本次内循环

if alphas[j]-alphaJold<0.00001:

print ("j not moving enough")

continue #第三个跳出条件

#约束条件让我们可以根据alphaJ求出alphaI

alphas[i]=alphas[i]+labelMatrix[i]\*labelMatrix[j]\*(alphaJold-alphas[j])

#更新b值,根据alpha是否在0～C决定更新的b值

b1=-Ei-labelMatrix[i]\*(alphas[i]-alphaIold)\*dataMatrix[i,:]\*dataMatrix[i,:].transpose()\

-labelMatrix[j]\*(alphas[j]-alphaJold)\*dataMatrix[j,:]\*dataMatrix[i,:].transpose()+b

b2=-Ej-labelMatrix[i]\*(alphas[i]-alphaIold)\*dataMatrix[i,:]\*dataMatrix[j,:].transpose()\

-labelMatrix[j]\*(alphas[j]-alphaJold)\*dataMatrix[j,:]\*dataMatrix[j,:].transpose()+b

#若ai或aj在(0,C)之间，则取b=bi或b=bj，若ai aj都不在(0,C)之间，取均值

if alphas[i]>0 and alphas[i]<C:

b=b1

elif alphas[j]>0 and alphas[j]<C:

b=b2

else:

b=(b1+b2)/2.0

alphaPairsChanged+=1 #若进行到这里，说明ai aj经过了层层筛选(continue)，已经被更新，于是内循环中alpha对更新次数+1

print ("iter:{0}; i:{1}; alpha pair changed:{2}".format(iters,i,alphaPairsChanged))

"""只有在内循环未对任何一对alpha做修改时，iters+1；否则我们让iters回到0，继续内循环；

只有当内循环未修改任一alpha对，且连续maxIter次迭代，才会结束(以保证所有alpha得到了充分的修改)

(这里其实有个改进点：只要alpha被修改，iter就+1，然后在引入一个停止条件(整个数据集没有可以再更新的alpha值)同时判断即可)

注意缩进"""

if alphaPairsChanged==0:

iters+=1

else:

iters=0

print ("iteration numer:%d" %iters)

return b,alphas