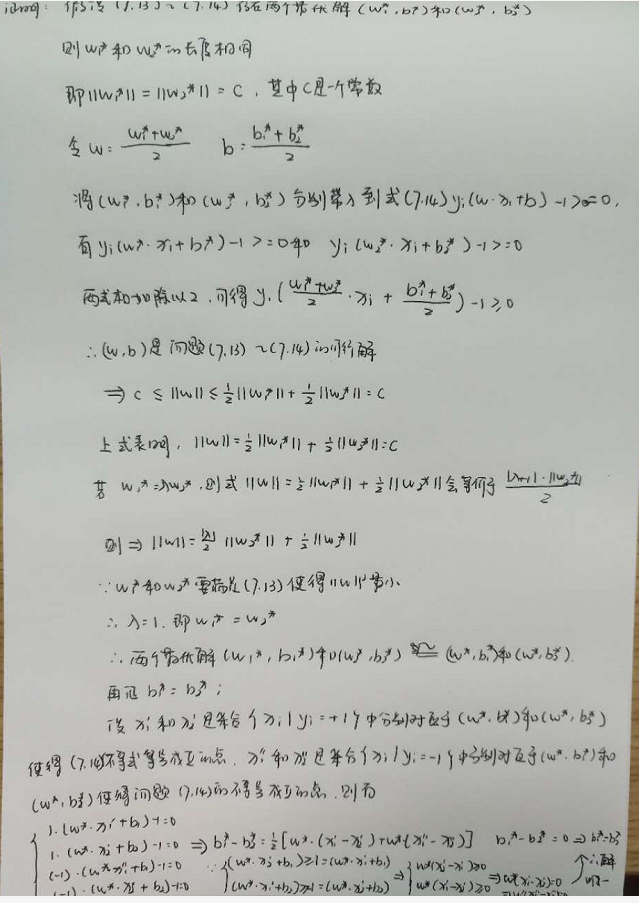
1. （必填）自己提出的问题的理解（罗列全部）：
2. **提出的问题1：**

P126，如何证明w的解是唯一的？

讨论后的理解：



1. **提出的问题2：**

如何找软间隔最大化时的支持向量？

讨论后的理解：

在软间隔最大化时，因为对每个样本(xi,yi)引入了松弛变量ξi。从下图来研究软间隔最大化时支持向量的情况，第i个点到对应类别支持向量的距离为ξi||w||2。根据软间隔最大化时KKT条件中的对偶互补条件α∗i(yi(wTxi+b)−1+ξ∗i)=0有：

a) 如果α=0,那么yi(wTxi+b)−1≥0,即样本在间隔边界上或者已经被正确分类。如图中所有远离间隔边界的点。

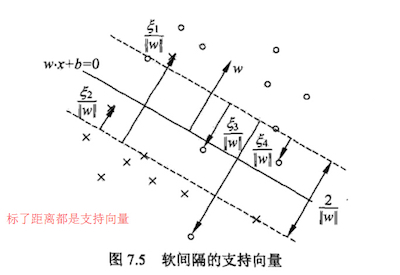
b) 如果0<α<C,那么ξi=0,yi(wTxi+b)−1=0,即点在间隔边界上。

c) 如果α=C，说明这是一个可能比较异常的点，需要检查此时ξi

　　i)如果0≤ξi≤1,那么点被正确分类，但是却在超平面和自己类别的间隔边界之间。如图中的样本2和4.

　　ii)如果ξi=1,那么点在分离超平面上，无法被正确分类。

　　　　iii)如果ξi>1,那么点在超平面的另一侧，也就是说，这个点不能被正常分类。如图中的样本1和3.



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**：

p132，图7.6中0-1损失函数在间隔大于0的部分是什么样的。

自己的理解：

0-1损失函数即为“判断正确”的话损失函数的值为0，“判断错误”的话损失函数的值为1，所以在图中左边的部分为一条恒等于1 的直线，右边的部分则是和坐标轴重合的直线。

2. **问题4：**

p131，（7.58）为什么式中要有一个正则化项？

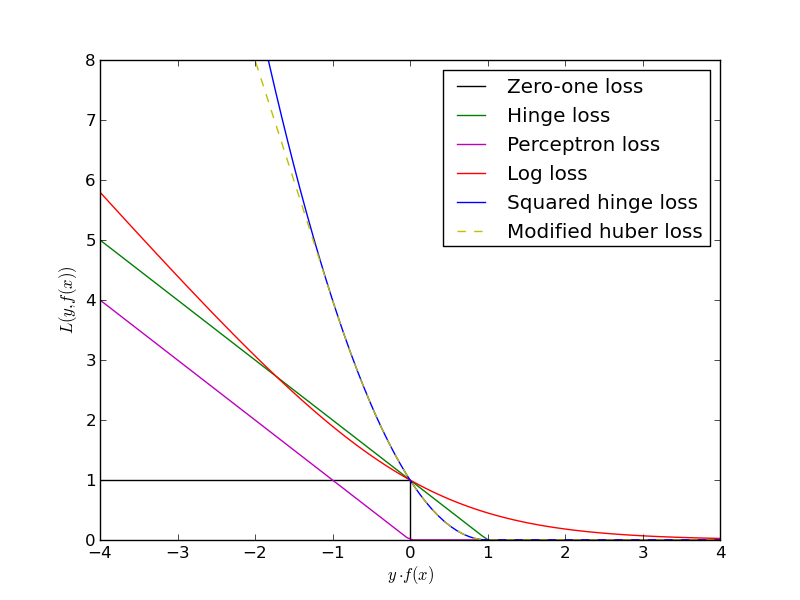
自己的理解：

和本章开始的时候的线性支持向量机相同，因为函数距离在求分类超平面的时候会出现只要等比例增加w和b就能在增大函数距离的同时而不改变超平面，这里也是类似，如果仅仅使用前面的损失函数，也会出现同样的问题，所以可以在后面增加一个w的l2范数来避免这种情况。

1. **问题5：**

多种损失函数的比较与选择？

讨论后的理解：



从上图可以看出上面介绍的这些损失函数都可以看作是0-1损失的单调连续近似函数，而因为这些损失函数通常是凸的连续函数，因此常用来代替0-1损失进行优化。它们的相同点是都随着margin→−∞而加大惩罚；不同点在于，logistic loss和hinge loss都是线性增长，而exponential loss是以指数增长。

值得注意的是上图中modified huber loss的走向和exponential loss差不多，并不能看出其robust的属性，其实这和算法时间复杂度一样。

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7.1-7.2：支持向量机）

2、下周计划：统计学习方法（第七章：支持向量机剩余部分）

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）

支持向量机（SVM）是一种二分类模型。它的基本模型是定义在特征空间上的间隔最大的线性分类器，间隔最大使它有别于感知机；支持向量机还包括和技巧，这使它陈伟实质上的非线性分类器。支持向量机的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题，也等价于正则化的合页损失函数的最小化问题。支持向量机的学习算法是求解凸二次规划的最优化算法。

支持向量机学习方法包含构建由简至繁的模型：线性可分支持向量机（linearsupport vector machine in linearly separable case）、线性支持向量机及非线性支持向量机。简单模型是复杂模型的基础，也是复杂模型的特殊情况。当训练数据线性可分时，通过硬间隔最大化，学习一个线性的分类器，即线性可分支持向量机，又称为硬间隔支持向量机；当训练数据近似线性可分时，通过软间隔最大化，也学习一个线性的分类器，即线性支持向量机，又称为软间隔支持向量机；当训练数据线性不可分时，通过使用和技巧及软间隔最大化，学习非线性支持向量机。

当输入空间为欧式空间或离散集合、特征空间为希尔伯特空间时，核函数表示将输入从输入空间映射到特征空间得到的特征向量之间的内积。通过使用核函数可以学习非线性支持向量机，等价于隐式地在高维的特征空间中学习线性支持向量机。这样的方法称为核技巧。核方法是比支持向量机更为一般的机器学习方法。

2、伪代码的具体实现(选做)

SVM算法实现

from numpy import \*

def loadDataSet(filename): #读取数据

dataMat=[]

labelMat=[]

fr=open(filename)

for line in fr.readlines():

lineArr=line.strip().split('\t')

dataMat.append([float(lineArr[0]),float(lineArr[1])])

labelMat.append(float(lineArr[2]))

return dataMat,labelMat #返回数据特征和数据类别

def selectJrand(i,m): #在0-m中随机选择一个不是i的整数

j=i

while (j==i):

j=int(random.uniform(0,m))

return j

def clipAlpha(aj,H,L): #保证a在L和H范围内（L <= a <= H）

if aj>H:

aj=H

if L>aj:

aj=L

return aj

def kernelTrans(X, A, kTup): #核函数，输入参数,X:支持向量的特征树；A：某一行特征数据；kTup：('lin',k1)核函数的类型和参数

m,n = shape(X)

K = mat(zeros((m,1)))

if kTup[0]=='lin': #线性函数

K = X \* A.T

elif kTup[0]=='rbf': # 径向基函数(radial bias function)

for j in range(m):

deltaRow = X[j,:] - A

K[j] = deltaRow\*deltaRow.T

K = exp(K/(-1\*kTup[1]\*\*2)) #返回生成的结果

else:

raise NameError('Houston We Have a Problem -- That Kernel is not recognized')

return K

#定义类，方便存储数据

class optStruct:

def \_\_init\_\_(self,dataMatIn, classLabels, C, toler, kTup): # 存储各类参数

self.X = dataMatIn #数据特征

self.labelMat = classLabels #数据类别

self.C = C #软间隔参数C，参数越大，非线性拟合能力越强

self.tol = toler #停止阀值

self.m = shape(dataMatIn)[0] #数据行数

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

self.b = 0 #初始设为0

self.eCache = mat(zeros((self.m,2))) #缓存

self.K = mat(zeros((self.m,self.m))) #核函数的计算结果

for i in range(self.m):

self.K[:,i] = kernelTrans(self.X, self.X[i,:], kTup)

def calcEk(oS, k): #计算Ek（参考《统计学习方法》p127公式7.105）

fXk = float(multiply(oS.alphas,oS.labelMat).T\*oS.K[:,k] + oS.b)

Ek = fXk - float(oS.labelMat[k])

return Ek

#随机选取aj，并返回其E值

def selectJ(i, oS, Ei):

maxK = -1

maxDeltaE = 0

Ej = 0

oS.eCache[i] = [1,Ei]

validEcacheList = nonzero(oS.eCache[:,0].A)[0] #返回矩阵中的非零位置的行数

if (len(validEcacheList)) > 1:

for k in validEcacheList:

if k == i:

continue

Ek = calcEk(oS, k)

deltaE = abs(Ei - Ek)

if (deltaE > maxDeltaE): #返回步长最大的aj

maxK = k

maxDeltaE = deltaE

Ej = Ek

return maxK, Ej

else:

j = selectJrand(i, oS.m)

Ej = calcEk(oS, j)

return j, Ej

def updateEk(oS, k): #更新os数据

Ek = calcEk(oS, k)

oS.eCache[k] = [1,Ek]

#首先检验ai是否满足KKT条件，如果不满足，随机选择aj进行优化，更新ai,aj,b值

def innerL(i, oS): #输入参数i和所有参数数据

Ei = calcEk(oS, i) #计算E值

if ((oS.labelMat[i]\*Ei < -oS.tol) and (oS.alphas[i] < oS.C)) or ((oS.labelMat[i]\*Ei > oS.tol) and (oS.alphas[i] > 0)): #检验这行数据是否符合KKT条件 参考《统计学习方法》p128公式7.111-113

j,Ej = selectJ(i, oS, Ei) #随机选取aj，并返回其E值

alphaIold = oS.alphas[i].copy()

alphaJold = oS.alphas[j].copy()

if (oS.labelMat[i] != oS.labelMat[j]): #以下代码的公式参考《统计学习方法》p126

L = max(0, oS.alphas[j] - oS.alphas[i])

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

else:

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

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

if L==H:

print("L==H")

return 0

eta = 2.0 \* oS.K[i,j] - oS.K[i,i] - oS.K[j,j] #参考《统计学习方法》p127公式7.107

if eta >= 0:

print("eta>=0")

return 0

oS.alphas[j] -= oS.labelMat[j]\*(Ei - Ej)/eta #参考《统计学习方法》p127公式7.106

oS.alphas[j] = clipAlpha(oS.alphas[j],H,L) #参考《统计学习方法》p127公式7.108

updateEk(oS, j)

if (abs(oS.alphas[j] - alphaJold) < oS.tol): #alpha变化大小阀值（自己设定）

print("j not moving enough")

return 0

oS.alphas[i] += oS.labelMat[j]\*oS.labelMat[i]\*(alphaJold - oS.alphas[j])#参考《统计学习方法》p127公式7.109

updateEk(oS, i) #更新数据

#以下求解b的过程，参考《统计学习方法》p129公式7.114-7.116

b1 = oS.b - Ei- oS.labelMat[i]\*(oS.alphas[i]-alphaIold)\*oS.K[i,i] - oS.labelMat[j]\*(oS.alphas[j]-alphaJold)\*oS.K[i,j]

b2 = oS.b - Ej- oS.labelMat[i]\*(oS.alphas[i]-alphaIold)\*oS.K[i,j]- oS.labelMat[j]\*(oS.alphas[j]-alphaJold)\*oS.K[j,j]

if (0 < oS.alphas[i]<oS.C):

oS.b = b1

elif (0 < oS.alphas[j]<oS.C):

oS.b = b2

else:

oS.b = (b1 + b2)/2.0

return 1

else:

return 0

#SMO函数，用于快速求解出alpha

def smoP(dataMatIn, classLabels, C, toler, maxIter,kTup=('lin', 0)): #输入参数：数据特征，数据类别，参数C，阀值toler，最大迭代次数，核函数（默认线性核）

oS = optStruct(mat(dataMatIn),mat(classLabels).transpose(),C,toler, kTup)

iter = 0

entireSet = True

alphaPairsChanged = 0

while (iter < maxIter) and ((alphaPairsChanged > 0) or (entireSet)):

alphaPairsChanged = 0

if entireSet:

for i in range(oS.m): #遍历所有数据

alphaPairsChanged += innerL(i,oS)

print("fullSet, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged)) #显示第多少次迭代，那行特征数据使alpha发生了改变，这次改变了多少次alpha

iter += 1

else:

nonBoundIs = nonzero((oS.alphas.A > 0) \* (oS.alphas.A < C))[0]

for i in nonBoundIs: #遍历非边界的数据

alphaPairsChanged += innerL(i,oS)

print("non-bound, iter: %d i:%d, pairs changed %d" % (iter,i,alphaPairsChanged))

iter += 1

if entireSet:

entireSet = False

elif (alphaPairsChanged == 0):

entireSet = True

print("iteration number: %d" % iter)

return oS.b,oS.alphas

def testRbf(data\_train,data\_test):

dataArr,labelArr = loadDataSet(data\_train) #读取训练数据

b,alphas = smoP(dataArr, labelArr, 200, 0.0001, 10000, ('rbf', 1.3)) #通过SMO算法得到b和alpha

datMat=mat(dataArr)

labelMat = mat(labelArr).transpose()

svInd=nonzero(alphas)[0] #选取不为0数据的行数（也就是支持向量）

sVs=datMat[svInd] #支持向量的特征数据

labelSV = labelMat[svInd] #支持向量的类别（1或-1）

print("there are %d Support Vectors" % shape(sVs)[0]) #打印出共有多少的支持向量

m,n = shape(datMat) #训练数据的行列数

errorCount = 0

for i in range(m):

kernelEval = kernelTrans(sVs,datMat[i,:],('rbf', 1.3)) #将支持向量转化为核函数

predict=kernelEval.T \* multiply(labelSV,alphas[svInd]) + b #这一行的预测结果（代码来源于《统计学习方法》p133里面最后用于预测的公式）注意最后确定的分离平面只有那些支持向量决定。

if sign(predict)!=sign(labelArr[i]): #sign函数 -1 if x < 0, 0 if x==0, 1 if x > 0

errorCount += 1

print("the training error rate is: %f" % (float(errorCount)/m)) #打印出错误率

dataArr\_test,labelArr\_test = loadDataSet(data\_test) #读取测试数据

errorCount\_test = 0

datMat\_test=mat(dataArr\_test)

labelMat = mat(labelArr\_test).transpose()

m,n = shape(datMat\_test)

for i in range(m): #在测试数据上检验错误率

kernelEval = kernelTrans(sVs,datMat\_test[i,:],('rbf', 1.3))

predict=kernelEval.T \* multiply(labelSV,alphas[svInd]) + b

if sign(predict)!=sign(labelArr\_test[i]):

errorCount\_test += 1

print("the test error rate is: %f" % (float(errorCount\_test)/m))

#主程序

def main():

filename\_traindata='C:\\Users\\Administrator\\Desktop\\data\\traindata.txt'

filename\_testdata='C:\\Users\\Administrator\\Desktop\\data\\testdata.txt'

testRbf(filename\_traindata,filename\_testdata)

if \_\_name\_\_=='\_\_main\_\_':

main()