机器学习——支持向量机（下）

前言

上一次讲到线性支持向量机的推导以及SMO算法的实现，这一节将讲到SMO的优化算法以及非线性支持向量机。

SMO算法优化：

简单的SMO是利用两个随机选择的α进行迭代，但是当数据集太大时运行速度会很慢，我们可以采用启发式的选择方式来选择第二个α，以此达到优化效果。

启发式选择方式：

对于下面的式子更新：

$$\eta = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$$

$$\alpha_j^{new} = \alpha_j^{old} + \frac{y_i(E_i - E_j)}{\eta}$$

在实现SMO算法时，先计算η，再更新α_j，为了加快第二个乘子的迭代速度，需要让斜率增大，因此只能：

$$max|E_i - E_j|$$

因此有以下步骤：

最外层的循环中，首先在样本中选择违反KKT的条件的一个乘子作为最外层循环，然后启发式选择另外的乘子进行优化，同时在非边界乘子中寻找使得上式子最大的样本，如果没有找到，就从整个样本中随机选择一个样本。也就是说：我们通过一个外循环来选择违反KKT条件的一个乘子，并且在以下两种方式进行交替：（1）在所有数据集上进行单边扫描，（2）在非边界中实现单边扫描。非边界alpha值是那些不等于边界C或0的alpha值，并且跳过那些已知的不会改变的alpha值，所以要建立列表，用于更新状态。在选择一个alpha值后，算法会通过启发式选择第二个alpha值。

代码实现：

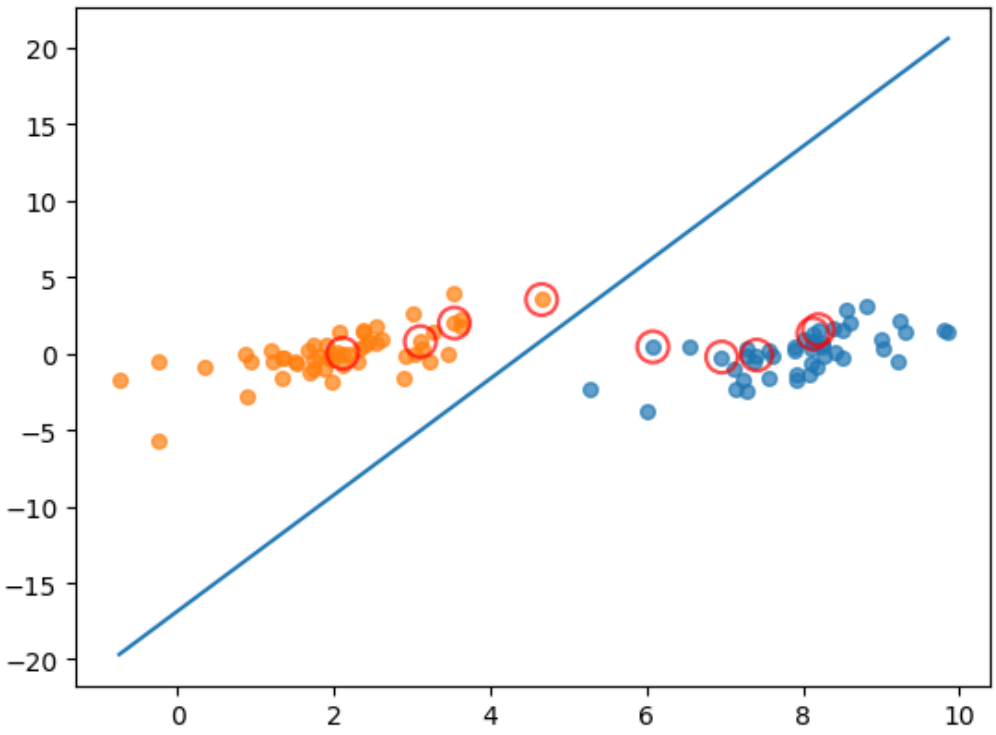
```
import matplotlib.pyplot as plt
import numpy as np
import random

class optStruct:

    def __init__(self,dataMatIn,classLabels,C,toler):
        self.X=dataMatIn
        self.labelMat=classLabels
        self.C=C
        self.tol=toler
        self.m=np.shape(dataMatIn)[0]
        self.alphas=np.mat(np.zeros((self.m,1)))
        self.b=0
        self.eCache=np.mat(np.zeros((self.m,2)))

def loadDataSet(filename):
```

```
    dataMat=[]
    lableMat=[]
    fr=open(filename)
    for line in fr.readlines():
        lineArr=line.strip().split('\t')
        dataMat.append([float(lineArr[0]),float(lineArr[1])])
        lableMat.append(float(lineArr[2]))
    return dataMat,lableMat

def calcEk(oS,k):
    fXk=float(np.multiply(oS.alphas,oS.labelMat).T*(oS.X*oS.X[k,:].T)+oS.b)
    Ek=fXk-float(oS.labelMat[k])
    return Ek

def selectJrand(i,m):
    j=i
    while(j==i) :
        j=int(random.uniform(0,m))
    return j

def selectJ(i,oS,Ei):
    maxK=-1
    maxDeltaE=0
    Ej=0
    oS.eCache[i]=[1,Ei]
    validEcachelist=np.nonzero(oS.eCache[:,0].A)[0]
    if(len(validEcachelist))>1:
        for k in validEcachelist:
            if k==i :
                continue
            Ek=calcEk(oS,k)
            deltaE=abs(Ei-Ek)
            if(deltaE>maxDeltaE):
                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):
    Ek=calcEk(oS,k)
    oS.eCache[k]=[1,Ek]

def clipAlpha(aj,H,L):
    if aj>H:
        aj=H
    if L>aj:
        aj=L
    return aj

def innerL(i,oS):
    Ei=calcEk(oS,i)
    #优化alpha
    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)):
```

```
        #使用内循环
        j,Ej=selectJ(i,oS,Ei)
        #保存
        #alphaIold=oS.alphas[i].copy()
        #alphaJold=oS.alphas[j].copy()
        alphaIold = oS.alphas[i].copy(); alphaJold = oS.alphas[j].copy();
        #计算上下界
        if (oS.labelMat[i] != oS.labelMat[j]):
            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
        #步骤三：计算eta
        eta = 2.0 * oS.X[i,:] * oS.X[j,:].T - oS.X[i,:] * oS.X[i,:].T -
oS.X[j,:] * oS.X[j,:].T
        if eta>=0:
            print("eta>=0")
            return 0
        oS.alphas[j]-=oS.labelMat[j]*(Ei-Ej)/eta
        #修建
        oS.alphas[j]=clipAlpha(oS.alphas[j],H,L)
        #更新误差
        updateEk(oS,j)
        if(abs(oS.alphas[j]-alphaJold)<0.00001):
            print("alpha_j变化太小")
            return 0
        oS.alphas[i]+=oS.labelMat[j]*oS.labelMat[i]*(alphaJold-oS.alphas[j])
        updateEk(oS,i)
        #更新b1，b2
        b1=oS.b-Ei-oS.labelMat[i]*(oS.alphas[i]-
alphaIold)*oS.X[i,:]*oS.X[i,:].T-oS.labelMat[j]*(oS.alphas[j]-
alphaJold)*oS.X[i,:]*oS.X[j,:].T
        b2=oS.b-Ej-oS.labelMat[i]*(oS.alphas[i]-
alphaIold)*oS.X[i,:]*oS.X[j,:].T-oS.labelMat[j]*(oS.alphas[j]-
alphaJold)*oS.X[j,:]*oS.X[j,:].T
        #更新b
        if (0<oS.alphas[i]) and (oS.C>oS.alphas[i]):
            oS.b=b1
        elif (0<oS.alphas[j]) and (oS.C>oS.alphas[j]):
            oS.b=b2
        else:
            oS.b=(b1+b2)/2.0
        return 1
    else:
        return 0


def smoP(dataMatIn,classLabels,C,toler,maxIter):
    oS=optStruct(np.mat(dataMatIn),np.mat(classLabels).transpose(),C,toler)
    iter=0
    entireSet=True
    alphaPairsChanged=0
    while(iter<maxIter) and ((alphaPairsChanged>0) or (entireSet)):
        alphaPairsChanged=0
```

```
        if entireSet:
            for i in range(oS.m):
                alphaPairsChanged+=innerL(i,oS)
                print("全样本遍历：第%d次迭代 样本：%d，alpha优化次数：%d" %
(iter,i,alphaPairsChanged))
            iter+=1
        else:
            nonBoundIs = np.nonzero((oS.alphas.A>0)*(oS.alphas.A<C))[0]
            for i in nonBoundIs:
                alphaPairsChanged+=innerL(i,oS)
                print("非边界遍历：第%d次迭代 样本：%d，alpha优化次数：%d" %
(iter,i,alphaPairsChanged))
            iter +=1
        if entireSet:
            entireSet=False
        elif(alphaPairsChanged==0):
            entireSet=True
        print("迭代次数：%d" % iter)
    return oS.b,oS.alphas

def showClassifer(dataMat,classLabels,w,b):
    data_plus=[]
    data_minus=[]
    for i in range(len(dataMat)):
        if classLabels[i]>0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np=np.array(data_plus)
    data_minus_np=np.array(data_minus)
    plt.scatter(np.transpose(data_plus_np)[0],np.transpose(data_plus_np)
[1],s=30,alpha=0.7)
    plt.scatter(np.transpose(data_minus_np)[0],np.transpose(data_minus_np)
[1],s=30,alpha=0.7)
    x1=max(dataMat)[0]
    x2=min(dataMat)[0]
    a1,a2=w
    b=float(b)
    a1=float(a1[0])
    a2=float(a2[0])
    y1=(-b-a1*x1)/a2
    y2= (-b-a1*x2)/a2
    plt.plot([x1,x2],[y1,y2])
    for i,alpha in enumerate(alphas):
        if alpha>0:
            x,y=dataMat[i]
            plt.scatter([x],
[y],s=150,c='none',alpha=0.7,linewidth=1.5,edgecolor='red')
    plt.show()

def calcWs(alphas,dataArr,classLabels):
    X=np.mat(dataArr)
    labelMat=np.mat(classLabels).transpose()
    m,n=np.shape(X)
    w=np.zeros((n,1))
    for i in range(m):
        w+=np.multiply(alphas[i]*labelMat[i],X[i,:].T)
    return w
```

```
dataArr,classLabels=loadDataSet('testSet.txt')
b,alphas=smoP(dataArr,classLabels,0.6,0.001,40)

w=calcWs(alphas,dataArr,classLabels)

showClassifer(dataArr,classLabels,w,b)
```



可以尝试运行一下，发现运行速度比前一个版本快很多，相比简单版的SMO算法只是随机选点，完整的版本覆盖整个数据集，也得到更好的结果。