机器学习——支持向量机（非线性与sklearn的应用）

对于线性可分的分类模型，我们前面的分析已经讲得很多了，但是如果是分类非线性的情况呢？我们需要的就是选择一个核函数。通过这个我们选定的核函数（非线性映射）将输入映射到一个高维的特征空间，将其在高维空间线性可分，在这个空间里进行最优分类。

超平面方程

对于上一次讲到的超平面方程：

$$f(x)=\sum_{i=1}^{n}\alpha_i y_i x_i^T x+b$$

将其使用非线性映射，映射到特征空间，分类函数变形为：

$$f(x)=\sum_{i=1}^{n}\alpha_i y_i <\emptyset(x_i),\emptyset(x)>+b$$

也就是说我们将非线性分类映射到另外一个空间，在这个新空间进行线性学习分类。

核函数

如果有一种方法可以在特征空间中直接计算内积，与原函数一样，就可以直接进行计算。具体地讲，如果没有核技术，我们就需要先计算线性映射$\emptyset(x_1)$和$\emptyset(x_2)$，然后计算它们的内积，而是用了核技术，可以有：

$$<\emptyset(x_1),\emptyset(x_2)>=k(<\emptyset(x_1),\emptyset(x_2)>)$$

这个表达式非常简单，计算很方便。

更进一步--非线性数据处理：

对于非线性的数据处理，一般我们可以找到合理的低维转高维的方法，使它从非线性的分类数据变成一个线性可分的数据，但是我们注意到如果这是一个多维转换的映射，如果维数太多，那么我们需要进行的计算量会很大，为了减少这个计算量，我们有一个核运算来解决这个问题。

例如对于以下2维到5维的变换：

$$\emptyset((x_1,x_2))=(\sqrt{2}x_1,x_1^2,\sqrt{2}x_2,x_2^2,\sqrt{2}x_1x_2,1)$$

我们采用单纯地对

$$a_1=(x_1,x_2),a_2=(y_1,y_2)$$

进行处理，它等价于下面直接的核函数处理：

$$(<a_1,a_2>+1)^2$$

但是对于任意映射的核函数是不可能去一一找的，所以我们有流行的核函数，例如径向基核函数，如下：

$$k(x_1,x_2)=exp\{-\frac{||x_1-x_2||^2}{2\sigma^2}\}$$

alpha是可以调节高次特征的权重衰减的。

下面我们做一个简单地例子实现，其实就是在之前的实现代码上将所有我们需要求到内积的地方改成核函数的算法就可以了：

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

class optStruct:

    def __init__(self,dataMatIn,classLabels,C,toler,kTup):
        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)))
        self.K=np.mat(np.zeros((self.m,self.m)))
        for i in range(self.m):
            self.K[:,i]=kernelTrans(self.X,self.X[i,:],kTup)

def kernelTrans(X,A,kTup):
    m,n=np.shape(X)
    K=np.mat(np.zeros((m,1)))
    if kTup[0]=='lin':
        K=X*A.T
    elif kTup[0]=='rbf':
        for j in range(m):
            deltaRow=X[j,:]-A
            K[j]=deltaRow*deltaRow.T
        K = np.exp(K/(-1*kTup[1]**2))
    else:
        raise NameError('核函数无法识别')
    return K

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.K[:,k])+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
```

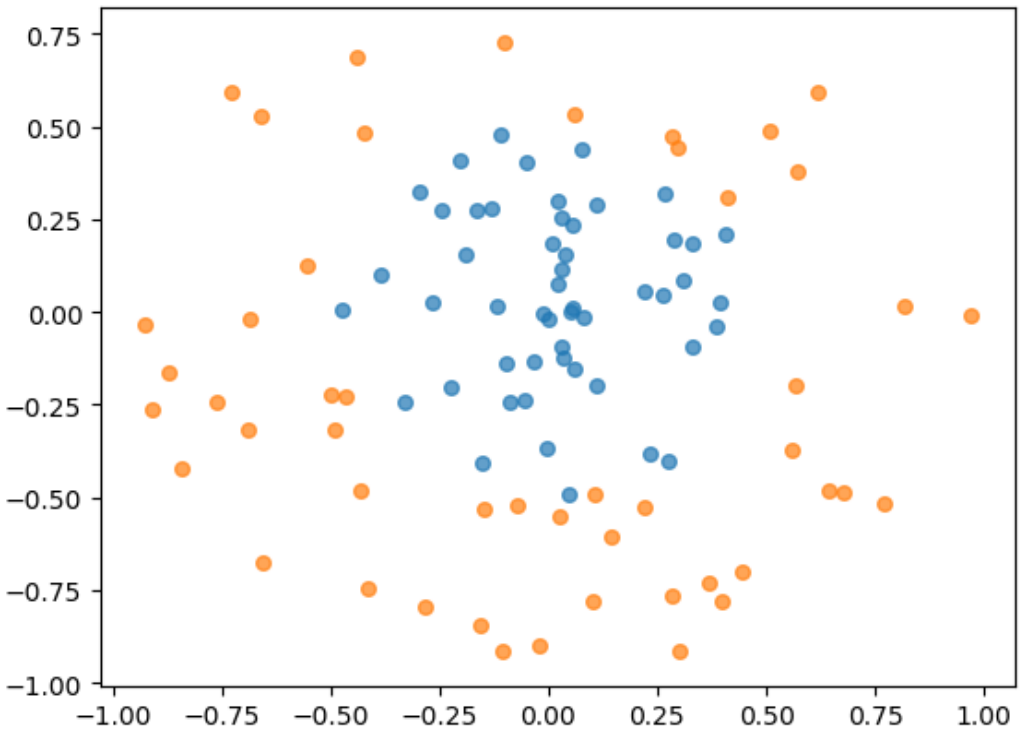
```
        eta = 2.0 * oS.K[i,j] - oS.K[i,i] - oS.K[j,j]
        if eta>=0:
            print("eta>=0")
            return 0
        oS.alphas[j]-=oS.labelMat[j]*(Ei-Ej)/eta
        #修建
        oS.alphas[j]=clipAlpha(oS.alphas[j],H,L)
        #更新误差
        updateEk(oS,j)
        if(abs(oS.alphas[j]-alphaJold)<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.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]
        #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,kTup=('lin',0)):
    oS=optStruct(np.mat(dataMatIn),np.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):
                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 testRbf(k1=1.3):
```

```
    dataArr,labelArr=loadDataSet('testSetRBF.txt')
    b,alphas=smoP(dataArr,labelArr,200,0.0001,100,('rbf',k1))
    dataMat = np.mat(dataArr)
    labelMat=np.mat(labelArr).transpose()
    svInd = np.nonzero(alphas.A>0)[0]
    sVs=dataMat[svInd]
    labelSV=labelMat[svInd]
    print("支持向量个数：%d" % np.shape(sVs)[0])
    m,n=np.shape(dataMat)
    errorCount=0
    for i in range(m):
        kernelEval =kernelTrans(sVs,dataMat[i,:],('rbf',k1))
        predict=kernelEval.T*np.multiply(labelSV,alphas[svInd])+b
        if np.sign(predict)!=np.sign(labelArr[i]):
            errorCount+=1
    print("训练集错误率： %.2f%%" %((float(errorCount)/m)*100))
    dataArr,labelArr=loadDataSet('testSetRBF2.txt')
    errorCount=0
    dataMat=np.mat(dataArr)
    labelMat=np.mat(labelArr)
    m,n=np.shape(dataMat)
    for i in range(m):
        kernelEval=kernelTrans(sVs,dataMat[i,:],('rbf',k1))
        predict=kernelEval.T*np.multiply(labelSV,alphas[svInd])+b
        if np.sign(predict)!=np.sign(labelArr[i]):
            errorCount+=1
    print("训练集错误率： %.2f%%" %((float(errorCount)/m)*100))
    showClassifer(dataArr,labelArr)

def showClassifer(dataMat,classLabels):
    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)
    plt.show()

testRbf()
```

我们输出训练集得到的错误率以及测试集的错误率：

> 迭代次数：4
> 支持向量个数：22
> 训练集错误率： 0.00%
> 训练集错误率： 8.00%

数据分类如下：



当然我们自己写出这样的函数主要是为了我们更好地理解支持向量机，而且我们的函数单线程、二分类，它的运行时间以及功能都很有限，我们如果掌握sklearn的使用会更加方便，我们不用理解它怎么实现也可以实现多分类，下面是对于数字识别的应用：

```
import numpy as np
import operator
from os import listdir
from sklearn.svm import SVC

def img2vector(filename):
    returnVect=np.zeros((1,1024))
    fr=open(filename)
    for i in range(32):
        linestr=fr.readline()
        for j in range(32):
            returnVect[0,32*i+j]=int(linestr[j])
    return returnVect

def handwritingClassTest():
    hwLabels=[]
    trainingFileList=listdir('trainingDigits')
    m=len(trainingFileList)
    trainingMat=np.zeros((m,1024))
    for i in range(m):
        filenameStr=trainingFileList[i]
        classNumber=int(filenameStr.split('_')[0])
        hwLabels.append(classNumber)
        trainingMat[i,:]=img2vector('trainingDigits/%s' %(filenameStr))
    clf=SVC(C=200,kernel='rbf')
    clf.fit(trainingMat,hwLabels)
    testFilelist=listdir('testDigits')
```

```
    errorCount=0.0
    mTest=len(testFilelist)
    for i in range(mTest):
        filenameStr=testFilelist[i]
        classNumber=int(filenameStr.split('_')[0])
        vectorUnderTest=img2vector('testDigits/%s' %(filenameStr))
        classifierResult=clf.predict(vectorUnderTest)
        print("分类结果是%d\t真实结果是%d" %(classifierResult,classNumber))
        if(classifierResult!=classNumber):
            errorCount+=1.0
    print("总共错了%d个数据\n错误率是%f%%" %(errorCount,errorCount/mTest*100))

handwritingClassTest()
```

得到的多分类结果（部分）：

> 分类结果是9 真实结果是9
> 分类结果是9 真实结果是9
> 分类结果是9 真实结果是9
> 分类结果是9 真实结果是9
> 分类结果是9 真实结果是9
> 总共错了13个数据
> 错误率是1.374207%

写到终于把支持向量机的内容全写完了！

总结：

支持向量机可以用于线性以及非线性的分类，而且它的模型学习能力很强，对于高维、小样本问题解决能力也很强，总的来说，支持向量机是最好的现成分类器但是对于调节核函数的选择很敏感。