链滴

支持向量机 (SVM)，序列最小优化算法 (SMO)

作者：Lonery

原文链接：https://ld246.com/article/1615275624030

来源网站：链滴

支持向量机(Support Vector Machine)由V.N. Vapnik，A.Y. Chervonenkis，C. Cortes 等在1964年出。序列最小优化算法（Sequential minimal optimization）是一种用于解决支持向量机训练过程所产生优化问题的算法。由John C. Platt于1998年提出。

<p1 style="text-indent:2em">

支持向量机的推导在西瓜书，各大网站已经有详细的介绍。本文主要依据John C. Platt发表的文章《Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines》来现SVM与SMO算法。

</p1>

算法的流程：

SMO算法

- 输入：训练数据集$T=\{(x_1,y_1),(x_2,y_2),...,(x_N,y_N)\}$
  $x_i\in\mathcal{X}=R^n, y_i\in\mathcal{Y}=\{+1,-1\}, i=1,2,...,N$ ，精度ε
- 输出：近似解 α

(1)取初值$\alpha^{(0)}=0$ ，令 $k=0$

(2)选取优化变量α_1^k,α_2^k，解析求解两个变量的最优化问题，求得最优解$\alpha_1^{(k+1)},\alpha_2^{(k+1)}$，更新 α 为$\alpha^{(k+1)}$;

(3)若在精度ε范围内满足停机条件

$$\sum_{i=1}^{N}\alpha_i y_i=0$$

$$0\le\alpha_i\le C,\quad i=1,2,...,N$$

$$y_i\cdot g(x_i)=\begin{cases}\ge 1, & \{x_i|\alpha_i=0\}\\ =1, & \{x_i|0<\alpha_i<C\}\\ \le 1, & \{x_i|\alpha_i=C\}\end{cases}$$

$$g(x_i)=\sum_{j=1}^{N}\alpha_j y_j K(x_j,x_i)+b$$

则转(4);否则令k=k+1,转(2);

(4)取$\hat{\alpha}=\alpha^{(k+1)}$

</br>

```
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
```

定义需要的数据，包含数据样本，数据标签，偏置b，拉格朗日乘子α，容忍系数C等。

```
class Par:
```

```
    def __init__(self,n,D,C,eps,tol):
        self.X=datasets.make_blobs(n_samples=n,n_features=D,centers=2,cluster_std=1.0,shuffl
=True,random_state=None)
        self.point=self.X[0]
        self.target=self.X[1]
        self.target[np.nonzero(self.target==0)[0]]=-1
        self.w=np.zeros((1,D))[0]
        self.b=0
        self.E=-self.target
        self.alpha=np.zeros((1,n))[0]
        self.n=n
        self.C=C
        self.eps=eps
        self.tol=tol
```

定义核函数，预测公式。

```
def kernel(x,y):
    return np.dot(x,y.T)

def f(x):
    s=0
    for i in range(n):
        s+=P.alpha[i]*P.target[i]*kernel(P.point[i],x)
    return s-P.b
```

被选中的一对α更新细节：

```
def takeStep(i1,i2):
    if i1==i2:
        return 0
    alph2=P.alpha[i2]
    alph1=P.alpha[i1]
    y1=P.target[i1]
    y2=P.target[i2]
    s=y1*y2
    #Compute L,H via equations (13) and (14)
    if y1!=y2:
        L=max(0,alph2-alph1)
        H=min(P.C,P.C+alph2-alph1)
    else:
        L=max(0,alph2+alph1-P.C)
        H=min(P.C,alph2+alph1)
    if L==H:
        return 0
    k11=kernel(P.point[i1],P.point[i1])
    k12=kernel(P.point[i1],P.point[i2])
    k22=kernel(P.point[i2],P.point[i2])
    eta=k11+k22-2*k12
    if eta>0:
        a2=alph2+y2*(P.E[i1]-P.E[i2])/eta
    if a2<L:
        a2=L
    elif a2>H:
```

```
        a2=H
    else:
        f1=y1*(P.E[i1]+b)-alph1*k11-s*alph2*k12
        f2=y2*(P.E[i2]+b)-s*alph1*k12-alph2*k22
        L1=alph1+s*(alph2-L)
        H1=alph1+s*(alph2+H)
        psiL=L1*f1+L*f2+0.5*L1**2*k11+0.5*L**2*k22+s*L*L1*k12
        psiH=H1*f1+H*f2+0.5*H1**2*k11+0.5*H**2*k22+s*H*H1*k12
        Lobj = psiL
        Hobj = psiH
    if Lobj<Hobj-eps:
        a2=L
    elif Lobj>Hobj+eps:
        a2=H
    else:
        a2=alph2
    if abs(a2-alph2)<P.eps*(a2+alph2+P.eps):
        return 0
    a1=alph1+s*(alph2-a2)
    #Update threshold to reflect change in Lagrange multipliers
    b1=P.E[i1]+y1*(a1-alph1)*k11+y2*(a2-alph2)*k12+P.b
    b2=P.E[i2]+y1*(a1-alph1)*k12+y2*(a2-alph2)*k22+P.b
    if a1>0 and a1<P.C:
        P.b=b1
    elif a2>0 and a2<P.C:
        P.b=b2
    else:
        P.b=(b1+b2)/2
    #Update weight vector to reflect change in a1 & a2, if SVM is linear
    P.w=P.w+y1*(a1-alph1)*P.point[i1]+y2*(a2-alph2)*P.point[i2]
    #Store a1 in the alpha array
    P.alpha[i1]=a1
    #Store a2 in the alpha array
    P.alpha[i2]=a2
    #Update error cache using new Lagrange multipliers
    P.E[i1]=f(P.point[i1])-P.target[i1]
    P.E[i2]=f(P.point[i2])-P.target[i2]
    return 1
```

内循环选择第二个α：

```
def examineExample(i2):
    global valid
    alph2=P.alpha[i2]
    y2=P.target[i2]
    r2=P.E[i2]*y2
    if (r2<-P.tol and alph2<P.C) or (r2>P.tol and alph2>0):
        valid=np.where((P.alpha!=0) & (P.alpha!=C))[0]
        Long=len(valid)
        if Long > 1:
            #i1 = result of second choice heuristic (section 2.2)
            best=-1
            if len(valid)>1:
                for k in valid:
```

```
                deltaE=abs(P.E[i2]-P.E[k])
                if deltaE>best:
                    best=deltaE
                    i1=k
                if takeStep(i1,i2):
                    return 1
        #loop over all non-zero and non-C alpha, starting at a random point
        if Long>0:
            random_index=np.random.randint(0,Long)
            for i in np.hstack((valid[random_index:Long],valid[0:random_index])):
                i1=i
                if takeStep(i1,i2):
                    return 1
        #loop over all possible i1, starting at a random point
        random_index=np.random.randint(0,n)
        for i in np.hstack((np.arange(random_index,n),np.arange(0,random_index))):
            #i1=loop variable
            i1=i
            if takeStep(i1,i2):
                return 1
    return 0
```

外循环选择第一个α：

```
def SMO():
    global valid
    numChanged=0
    examineAll=1
    while numChanged>0 or examineAll:
        numChanged=0
        if examineAll:
            for i in range(n):
                numChanged+=examineExample(i)
        else:
            #loop I over examples where alpha is not 0 & not C
            for i in valid:
                numChanged+=examineExample(i)
        if examineAll==1:
            examineAll=0
        elif numChanged==0:
            examineAll=1
```

主函数入口：

```
if __name__ == '__main__':
    n=100       #样本个数
    C=10
    eps=0.001  #停止精度
    tol=0.001   #分类容错率
    D=2           #样本维度
    P=Par(n,D,C,eps,tol)
    SMO()
    #绘制图像
    plt.scatter(P.point[:,0],P.point[:,1],c=P.target)
```

```
x=np.arange(-10,10,0.1)
y=(P.b-P.w[0]*x)/P.w[1]
plt.plot(x,y)
plt.show()
Y=kernel(P.point,P.w)-P.b
count=0
for i in range(n):
if Y[i]*P.target[i]<0:
count+=1
print('Error Point num:',count)
```

单次测试结果：