# 支持向量机（线性核）

1. 模型概述
2. 概念描述

支持向量机（support vector machines, 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；SVM还包括核技巧，这使它成为实质上的非线性分类器。SVM的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。SVM的的学习算法就是求解凸二次规划的最优化算法。

1. 公式描述

|  |  |  |  |
| --- | --- | --- | --- |
| **数据集** | x= | y= | =[] |
| **决策函 数** |  | | |
| **Hinge loss** |  | | |
| **损失函 数** |  | | |
| **梯度下降法** | If()  = -  = -  = - | | If()  = -  = - |
| **梯度详 解** | =2+C(-)  =2+C(-)  = C(-) | | |

1. 重要代码详解

决策函数：返回该点的决策，这里返回的是矩阵。

def h(x,w):

    return x\*w.T

损失函数：left为; right为;其中使用temp实现**Hinge loss**将其中小于0的元素置为0。

def cost(x,y,w,C):

    left=np.sum(np.power(w[:,1:],2))/2

    temp=1-np.multiply(y,h(x,w))

    temp[temp<0]=0

    right=C\*np.sum(temp)

    return left+right

梯度下降函数：按照上表中的公式进行梯度下降，其中修改了一点为2我在这里使其乘以目的是为了使迭代次数增加，使迭代次数对于正则化的影响越来越强。

def graddecent(x,y,w,C,alpha,count):

    Loss=np.zeros(count)

    for i in range(1,count+1):

        for j in range(len(y)):

            if 1-y[j]\*h(x[j,:],w)<0:

                w[:,1:]-=alpha\*(2\*w[:,1:]\*(1/i))

            else:

                w[:,0]-=alpha\*(C\*(-y[j,:]\*x[j,0]))

                w[:,1:]-=alpha\*(2\*w[:,1:]\*(1/i)+C\*(-y[j,:]\*x[j,1:]))

        Loss[i-1]=cost(x,y,w,C)

        if i%100==0:

            print("第%d次,Loss为%.6f"%(i,Loss[i-1]))

    return w,Loss

绘图：分别对C=1和C=100两种情况绘图，绘制为决策边界。

pos=data[data['y'].isin([1])]

neg=data[data['y'].isin([-1])]

plt.plot(pos['x1'],pos['x2'],'o',c='b')

plt.plot(neg['x1'],neg['x2'],'x',c='r')

x\_=np.linspace(x[:,1:].min(),x[:,1:].max(),51)

y\_= -(w1[0,0] + x\_\*w1[0,1]) / w1[0,2]

plt.plot(x\_,y\_,'r')

plt.title("C=1")

plt.show()

plt.plot(pos['x1'],pos['x2'],'o',c='b')

plt.plot(neg['x1'],neg['x2'],'x',c='r')

x\_=np.linspace(x[:,1:].min(),x[:,1:].max(),51)

y\_= -(w2[0,0] + x\_\*w2[0,1]) / w2[0,2]

plt.plot(x\_,y\_,'r')

plt.title("C=100")

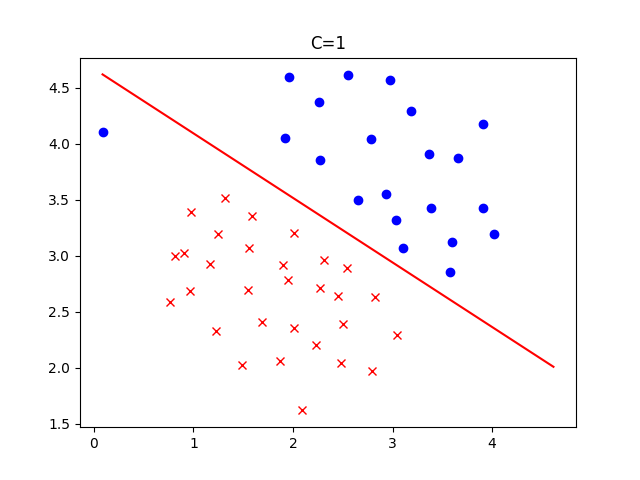
plt.show()

1. 结果分析e

根据C=1和C=100两种情况给出结果

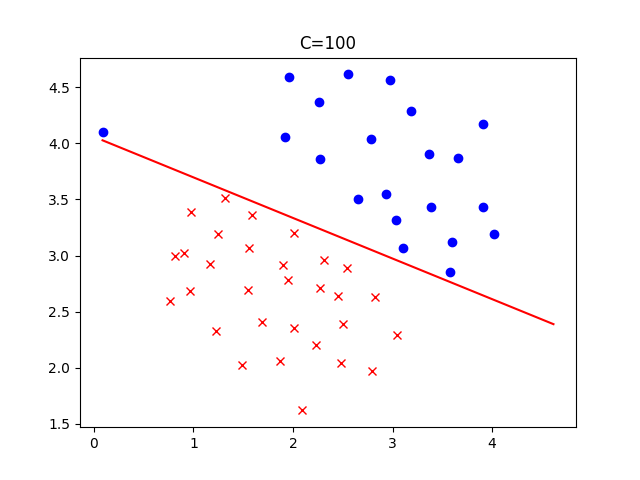
C=1时求得w=[[-12.744 1.57384444 2.72946453]]

此时决策边界



C=100时求得w= [[-73.8 6.58473743 18.18035156]]

此时决策边界



此时可以看出正则化对于决策边界的影响，当C较小时允许决策犯较多错误，不会过拟合

当C较大时则允许决策犯较少错误，容易过拟合。

四.源码

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.io import loadmat

row\_data = loadmat('支持向量机/data1.mat')

data = pd.DataFrame(row\_data['X'], columns=['x1', 'x2'])

data['y']=row\_data['y']

data['y'] = data['y'].map(lambda x: -1 if x == 0 else 1)

data.insert(0, 'ones', 1)

print(data)

cols = data.shape[1]

x=np.matrix(data.iloc[:,0:cols-1])

y=np.matrix(data.iloc[:,cols-1:cols])

w\_1=np.matrix(np.zeros(3))

w\_2=np.matrix(np.zeros(3))

def h(x,w):

    return x\*w.T

def cost(x,y,w,C):

    left=np.sum(np.power(w[:,1:],2))/2

    temp=1-np.multiply(y,h(x,w))

    temp[temp<0]=0

    right=C\*np.sum(temp)

    return left+right

def graddecent(x,y,w,C,alpha,count):

    Loss=np.zeros(count)

    for i in range(1,count+1):

        for j in range(len(y)):

            if 1-y[j]\*h(x[j,:],w)<0:

                w[:,1:]-=alpha\*(2\*w[:,1:]\*(1/i))

            else:

                w[:,0]-=alpha\*(C\*(-y[j,:]\*x[j,0]))

                w[:,1:]-=alpha\*(2\*w[:,1:]\*(1/i)+C\*(-y[j,:]\*x[j,1:]))

        Loss[i-1]=cost(x,y,w,C)

        if i%100==0:

            print("第%d次,Loss为%.6f"%(i,Loss[i-1]))

    return w,Loss

alpha=0.003

count=5000

C1=1

C2=100

w1,loss1=graddecent(x,y,w\_1,C1,alpha,count)

w2,loss2=graddecent(x,y,w\_2,C2,alpha,count)

#查看损失函数

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(np.arange(1,count+1), loss1, 'r')

ax.set\_xlabel('count')

ax.set\_ylabel('loss')

ax.set\_title('C1')

plt.show()

fig, ax = plt.subplots(figsize=(8,6))

ax.plot(np.arange(1,count+1), loss2, 'r')

ax.set\_xlabel('count')

ax.set\_ylabel('loss')

ax.set\_title('C2')

plt.show()

print(w1,'\n',w2)

pos=data[data['y'].isin([1])]

neg=data[data['y'].isin([-1])]

plt.plot(pos['x1'],pos['x2'],'o',c='b')

plt.plot(neg['x1'],neg['x2'],'x',c='r')

x\_=np.linspace(x[:,1:].min(),x[:,1:].max(),51)

y\_= -(w1[0,0] + x\_\*w1[0,1]) / w1[0,2]

plt.plot(x\_,y\_,'r')

plt.title("C=1")

plt.show()

plt.plot(pos['x1'],pos['x2'],'o',c='b')

plt.plot(neg['x1'],neg['x2'],'x',c='r')

x\_=np.linspace(x[:,1:].min(),x[:,1:].max(),51)

y\_= -(w2[0,0] + x\_\*w2[0,1]) / w2[0,2]

plt.plot(x\_,y\_,'r')

plt.title("C=100")

plt.show()