机器学习实验 6. 支持向量机 SVM

支持向量机（SVM）概述

支持向量机（support vector machines，SVM）是一种二类分类模型。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；支持向量机还包括核技巧，这使它成为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（convex quadratic programming）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。

支持向量机的原理

给定训练样本集 $D=\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}, y_i \in \{-1,+1\}$，分类学习最基本的想法就是基于训练集 D 在样本空间中找到一个划分超平面，将不同类别的样本分开。但能将训练样本分开的划分超平面可能有很多，如图 1 所示，我们应该努力去找到哪一个呢?

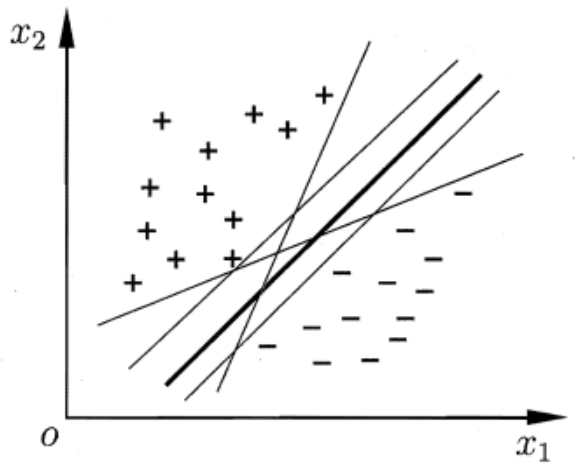


图 1 存在多个划分超平面将两类训练样本分开

直观上看，应该去找位于两类训练样本"正中间"的划分超平面，即图 1 中“-”

符号，因为该划分超平面对训练样本局部扰动的“容忍”性最好。例如，由于训练集的局限性或噪声的因素，训练集外的样本可能比图 1 中的训练样本更接近两个类的分隔界，这将使许多划分超平面出现错误，而“-”的超平面受影响最小。换言之，这个划分超平面所产生的分类结果是最鲁棒的，对未见示例的泛化能力最强。

在样本空间中，划分超平面可通过如下线性方程来描述：

$$w^T + b = 0 \tag{1}$$

其中 $w = (w_1; w_2; ,...; w_d)$ 为法向量，决定了超平面的方向；b 为位移项，决定了超平面与原点之间的距离。显然，划分超平面可被法向量 w 和位移 b 确定，下面我们将其记为 (w,b)。样本空间中任意点 x 到超平面 (w,b) 的距离可写为：

$$\mathbf{r} = \frac{\left|w^T x + b\right|}{\|w\|} \tag{2}$$

假设超平面 (w,b) 能将训练样本正确分类，即对于 $(x_i, y_i) \in D$，若 $y_i = +1$，则有;若，则有 $w^T w_i + b > 0$ $w^T w_i + b < 0$.令

$$\begin{cases} w^T x_i + b \geq +1, y_i = +1; \\ w^T x_i + b \leq -1, y_i = -1; \end{cases} \tag{3}$$

如图 2 所示，距离超平面最近的这几个训练样本点使式(3)的等号成立，它们被称为“支持向量” (support vector)，两个异类支持向量到超平面的距离之和为

$$\gamma = \frac{\mathbf{2}}{\|\mathbf{w}\|} \tag{4}$$

它被称为“间隔” (margin)。

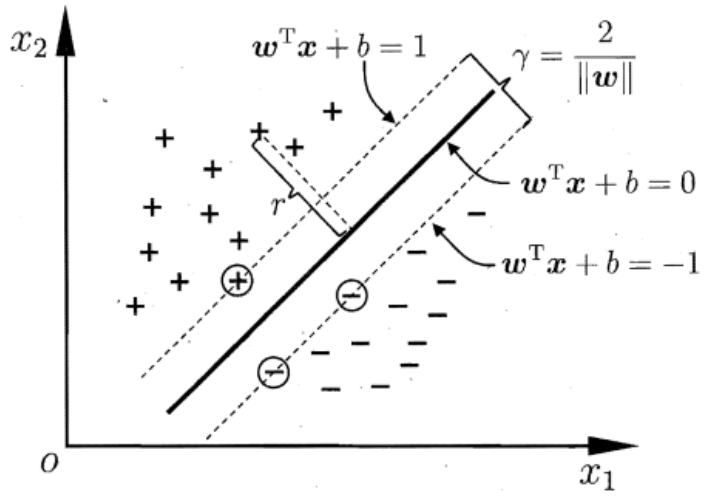


图 2 支持向量与间隔

欲找到具有“最大间隔” (maximum margin)的划分超平面，也就是要找到能满足式(3)中约束的参数 w 和 b，使得γ 最大，即

$$\begin{aligned}&\max_{w,b}\frac{2}{\|w\|}\\&s.t.y_{i.}(w^{T}x_{i}+b)\geq 1,i=1,2,...,m\end{aligned}\qquad(5)$$

显然，为了最大化间隔，仅需最大化$\|w\|^{-1}$，这等价于最小化$\|w\|^{2}$。于是，式(5)可重写为

$$\begin{aligned}&\min_{w,b}\frac{1}{2}\|w\|^{2}\\&s.t.y_{i.}(w^{T}x_{i}+b)\geq 1,i=1,2,...,m\end{aligned}\qquad(5)$$

这就是支持向量机的基本型。

SVM 在 SMO 算法中的一般实现

1）生成已标记的数据集:

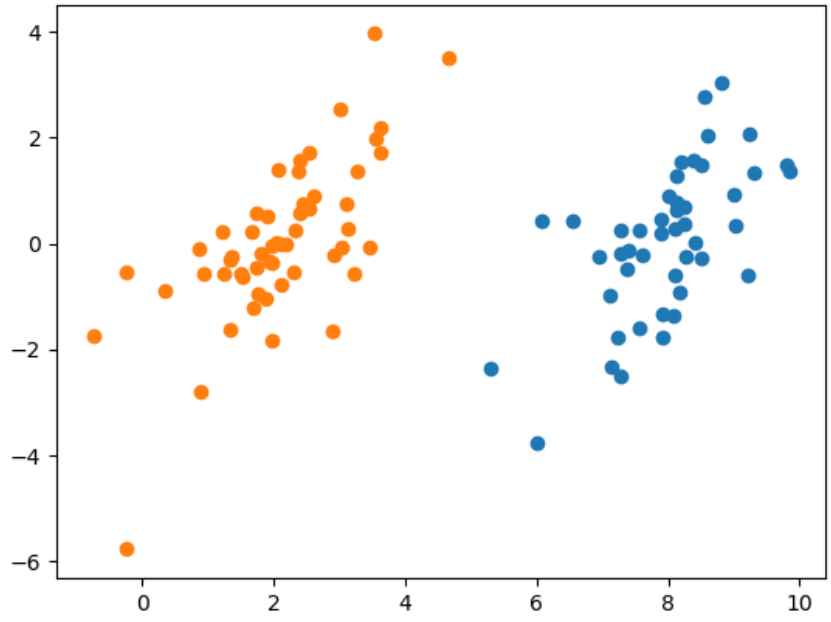
数据集的下载地址为：https://github.com/ahtchxw/leetcode/blob/master/testSet.txt。将数据集文件放在这个 python 文件目录下。然在再 python 文件输入:

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

def loadDataSet(fileName):
    dataMat = []; labelMat = []
    fr = open(fileName)
    for line in fr.readlines():                                    #逐行读取，滤除空格等
        lineArr = line.strip().split('\t')
        dataMat.append([float(lineArr[0]), float(lineArr[1])])     #添加数据
        labelMat.append(float(lineArr[2]))                         #添加标签
    return dataMat,labelMat
```

这一步主要是读取并处理数据集，创造两个矩阵，一个是数据矩阵一个是标签矩阵，并将处理过的数据分别存入这两个矩阵中。

```
def showDataSet(dataMat, labelMat):
    data_plus = []                                  #正样本
    data_minus = []                                 #负样本
    for i in range(len(dataMat)):
        if labelMat[i] > 0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np = np.array(data_plus)              #转换为numpy矩阵
    data_minus_np = np.array(data_minus)            #转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0], np.transpose(data_plus_np)[1])    #正样本散点图
    plt.scatter(np.transpose(data_minus_np)[0], np.transpose(data_minus_np)[1]) #负样本散点图
    plt.show()

dataMat, labelMat = loadDataSet('testSet.txt')
showDataSet(dataMat, labelMat)
```

这一步就是对数据进行可视化操作，看一下输出结果：



这就是我们使用的二维数据集，显然线性可分。现在我们使用简化版的 SMO 算法进行求解线性 svm。

2）简化版 SMO 算法：

```
from time import sleep
import random
import types

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

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

这里先简单介绍一下 SMO 算法。SMO 算法的目标是求出一系列 alpha 和 b，一旦求出了这些 alpha，就很容易计算出权重向量 w 并得到分隔超平面。SMO 算法的工作原理是：每次循环中选择两个 alpha 进行优化处理。一旦找到了一对合适的 alpha，那么就增大其中一个同时减小另一个。这里所谓的“合适”就是指两个 alpha 必须符合以下两个条件，条件之一就是两个 alpha 必须要在间隔边界之外，而且第二个条件则是这两个 alpha 还没有进进行过区间化处理或者不在边界上。而上面两个函数就是用来随机选择和修剪 alpha 值的。接下来就是 SMO 的主函数：

```
def smoSimple(dataMatIn, classLabels, C, toler, maxIter):
    dataMatrix = np.mat(dataMatIn); labelMat = np.mat(classLabels).transpose() #转换为numpy的mat存储
    b = 0; m,n = np.shape(dataMatrix)    #初始化b参数，统计dataMatrix的维度
    alphas = np.mat(np.zeros((m,1)))     #初始化alpha参数，设为0
    iter_num = 0    #最多迭代matIter次
    while (iter_num < maxIter):
        alphaPairsChanged = 0
        for i in range(m):
            #步骤1：计算误差Ei
            fXi = float(np.multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[i,:].T)) + b
            Ei = fXi - float(labelMat[i])
            #优化alpha，更设定一定的容错率。
            if ((labelMat[i]*Ei < -toler) and (alphas[i] < C)) or ((labelMat[i]*Ei > toler) and (alphas[i] > 0)):
                #随机选择另一个与alpha_i成对优化的alpha_j
                j = selectJrand(i,m)
                #步骤1：计算误差Ej
                fXj = float(np.multiply(alphas,labelMat).T*(dataMatrix*dataMatrix[j,:].T)) + b
                Ej = fXj - float(labelMat[j])
                #保存更新前的aplpha值，使用深拷贝
                alphaIold = alphas[i].copy(); alphaJold = alphas[j].copy();
                #步骤2：计算上下界L和H
                if (labelMat[i] != labelMat[j]):
                    L = max(0, alphas[j] - alphas[i])
                    H = min(C, C + alphas[j] - alphas[i])
                else:
                    L = max(0, alphas[j] + alphas[i] - C)
                    H = min(C, alphas[j] + alphas[i])
                if L==H: print("L==H"); continue
                #步骤3：计算eta
                eta = 2.0 * dataMatrix[i,:]*dataMatrix[j,:].T - dataMatrix[i,:]*dataMatrix[i,:].T
                - dataMatrix[j,:]*dataMatrix[j,:].T
                if eta >= 0: print("eta>=0"); continue
                #步骤4：更新alpha_j
                alphas[j] -= labelMat[j]*(Ei - Ej)/eta
                #步骤5：修剪alpha_j
                alphas[j] = clipAlpha(alphas[j],H,L)
                if (abs(alphas[j] - alphaJold) < 0.00001): print("alpha_j变化太小"); continue
                #步骤6：更新alpha_i
                alphas[i] += labelMat[j]*labelMat[i]*(alphaJold - alphas[j])
                #步骤7：更新b_1和b_2
                b1 = b - Ei- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[i,:].T
                - labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[i,:]*dataMatrix[j,:].T
                b2 = b - Ej- labelMat[i]*(alphas[i]-alphaIold)*dataMatrix[i,:]*dataMatrix[j,:].T
                - labelMat[j]*(alphas[j]-alphaJold)*dataMatrix[j,:]*dataMatrix[j,:].T
                #步骤8：根据b_1和b_2更新b
                if (0 < alphas[i]) and (C > alphas[i]): b = b1
                elif (0 < alphas[j]) and (C > alphas[j]): b = b2
                else: b = (b1 + b2)/2.0
                alphaPairsChanged += 1 #统计优化次数
                print("第%d次迭代 样本:%d, alpha优化次数:%d" % (iter_num,i,alphaPairsChanged)) #打印统计信息
        if (alphaPairsChanged == 0): iter_num += 1
        else: iter_num = 0
        print("迭代次数: %d" % iter_num)
    return b,alphas
```

完成上述函数后我们得到了想要的 b 的值和 alpha 的值，然后根据数据矩阵和标签矩阵就可以求得 w 的值：

```
def get_w(dataMat, labelMat, alphas):
    alphas, dataMat, labelMat = np.array(alphas), np.array(dataMat), np.array(labelMat)
    w = np.dot((np.tile(labelMat.reshape(1, -1).T, (1, 2)) * dataMat).T, alphas)
    return w.tolist()
```

至此 SMO 的模型基本构成，最后对其进行可视化操作：

```
def showClassifer(dataMat, w, b):
    #绘制样本点
    data_plus = []                                  #正样本
    data_minus = []                                 #负样本
    for i in range(len(dataMat)):
        if labelMat[i] > 0:
            data_plus.append(dataMat[i])
        else:
            data_minus.append(dataMat[i])
    data_plus_np = np.array(data_plus)              #转换为numpy矩阵
    data_minus_np = np.array(data_minus)            #转换为numpy矩阵
    plt.scatter(np.transpose(data_plus_np)[0], np.transpose(data_plus_np)[1], s=30, alpha=0.7)   #正样本散点图
    plt.scatter(np.transpose(data_minus_np)[0], np.transpose(data_minus_np)[1], s=30, alpha=0.7) #负样本散点图
    #绘制直线
    x1 = max(dataMat)[0]
    x2 = min(dataMat)[0]
    a1, a2 = w
    b = float(b)
    a1 = float(a1[0])
    a2 = float(a2[0])
    y1, y2 = (-b- a1*x1)/a2, (-b - a1*x2)/a2
    plt.plot([x1, x2], [y1, y2])
    #找出支持向量点
    for i, alpha in enumerate(alphas):
        if abs(alpha) > 0:
            x, y = dataMat[i]
            plt.scatter([x], [y], s=150, c='none', alpha=0.7, linewidth=1.5, edgecolor='red')
    plt.show()
```

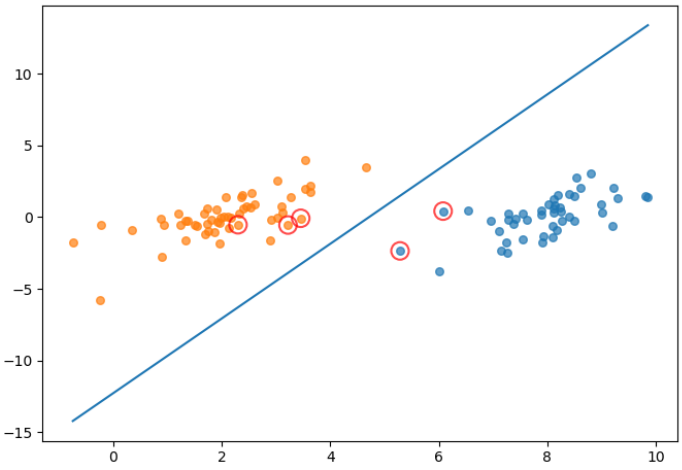
添加完上述函数之后，这里要把之前对数据集可视化的代码删除，即删除这一行代码：

```
showDataSet(dataMat,labelMat)
```

然后再输入：

```
b,alphas = smoSimple(dataMat, labelMat, 0.6, 0.001, 40)
w = get_w(dataMat, labelMat, alphas)
showClassifer(dataMat, w, b)
```

运行结果如下：



其中，中间的蓝线为求出来的分类器，用红圈圈出的点为支持向量点。