支持向量机的深入理解

大数据机器学习课程第五次作业

叶璨铭 (2024214500) ycm24@mails.tsinghua.edu.cn

2024年11月9日星期六

目录

|  |
| --- |
| 重要 |
| 本文档具有一定的交互性，建议使用浏览器打开html文件，这样比pdf文件阅读体验更佳。 |

## 1 第一题——一个例子理解支持向量机

题目如下

已知正例点 , , , 负例点, , 试求最大间隔分离超平面和分类决策函数，并在图上画出分离超平面、间隔边界及支持向量。

### 1.1 审题

我们首先复习一下李航书上的内容。

* 什么是分离超平面？什么是超平面？
  + 首先我认为为了学好这一章以及感知机的内容，**有必要详细了解超平面到底是什么**。
  + 中的[超平面(hyperplane)](https://en.wikipedia.org/wiki/Hyperplane)是指其中一个 维 (需要多少个向量来张成)的子空间（不一定通过原点，所以不一定是向量空间）。超平面将原本的空间一分为二，分为两个半平面。
  + [超平面的解析式有很多种](https://baike.baidu.com/item/%E5%B9%B3%E9%9D%A2%E6%96%B9%E7%A8%8B/9949549)，高中和高数就学过平面有一般式、点法式、截距式、参数式等，我们机器学习中关注是一般式，而一般式和点法式是可以互相转换的，具有一定的关联。
  + 高数中学的三维空间中的平面一般式是 ，A,B,C不同时为零。扩展一下那就是，其中 是超平面的法向量，要求不是零向量，也就是至少其中一个分量不为零。
    - 李航书上随口一说 是超平面的“截距”，但是这个说法我找不到依据（不要想当然以为直线里面的 y=kx+b 的b叫截距这里就也是截距）。实际上一般式里面的D没有任何几何意义！
    - 所谓截距（intercept），是平面和坐标轴的交点的位置，也就是其他维度的值取0的时候，剩下这个坐标轴这个维度上，取哪个值才能正好也落在平面上。
      * 所以平面的截距式是 ;
      * 这里的 才是真正的 “截距”。
    - 三维空间的二维平面这里有四个参数，但是实际上平面的维度是2，这是为什么呢？首先这是个方程决定的点集，而不是通过函数决定的点集，方程左右同时乘k仍然成立，所以少一个自由度。那还有一个参数呢？平面维度是2的意思是，你确定点在这个平面上的时候，两个坐标可以描述上面的点，这个2和需要多少个参数来描述这个平面自身在三维空间的位置是两回事。那为什么n维空间的超平面需要n-1个参数来描述呢？这里其实还有些数学细节可以深入探讨，TODO。
  + 点法式是, 和一般式的一样都是法向量(可长可短，方向是那个方向)。
    - 注意李航书的符号规范上不用。
    - 这个很好理解，平面上一个点 与 真的在直线上的点 的差的向量，必然还是在平面上的，所以 这个差向量 和 法向量垂直。
    - 稍微变一下就是 ,
    - ，如果 , 在这个情况下， 是有意义的，表示 某个平面上的点投影到法向量方向上的长度，那实际上就是从坐标原点0向量到达平面的垂直距离（最短距离），也就是说李航就算要说是什么，也应该说它是“到原点距离”，而不是“截距”，何况这是有条件的。
* 什么是最大间隔？函数间隔与几何间隔是什么？
* 什么是间隔边界？

复习之后，我们清晰看到题目询问的问题。

首先对正例点 , , , 负例点, 可视化

import numpy as np

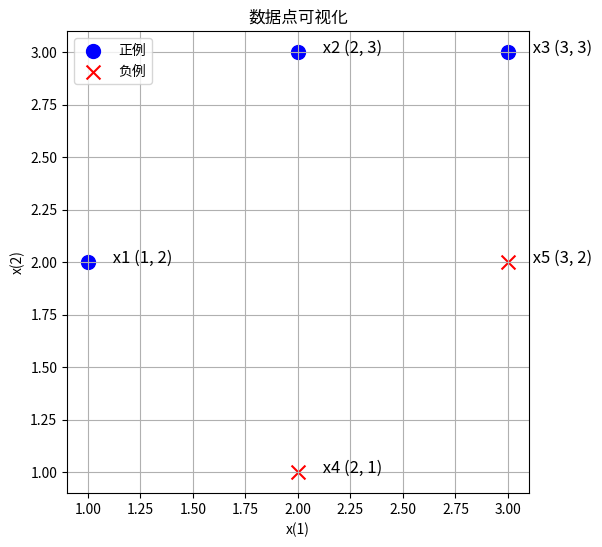
X = np.array([  
 [1, 2],   
 [2, 3],  
 [3, 3],  
 [2, 1],   
 [3, 2]  
])  
y = np.array([1]\*3 + [-1]\*2)

[source](https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data/blob/main/thu_big_data_ml/svm/vis.py#L13)

plot\_binary\_classification\_2d

plot\_binary\_classification\_2d (X, y, labels=[1, -1], label\_names=['正例',  
 '负例'], label\_colors=['blue', 'red'],  
 label\_markers=['o', 'x'], x\_names=['x(1)',  
 'x(2)'], point\_size=100, text\_offset=(0.1,  
 0), point\_names=None, title='数据点可视化',  
 size=(9, 6))

# 显示图表  
fig, ax = plot\_binary\_classification\_2d(X, y)



要求找出上图的最大间隔分离超平面，以及对应的分类决策函数；

然后再图上画出分离超平面、间隔边界及支持向量。

### 1.2 解题

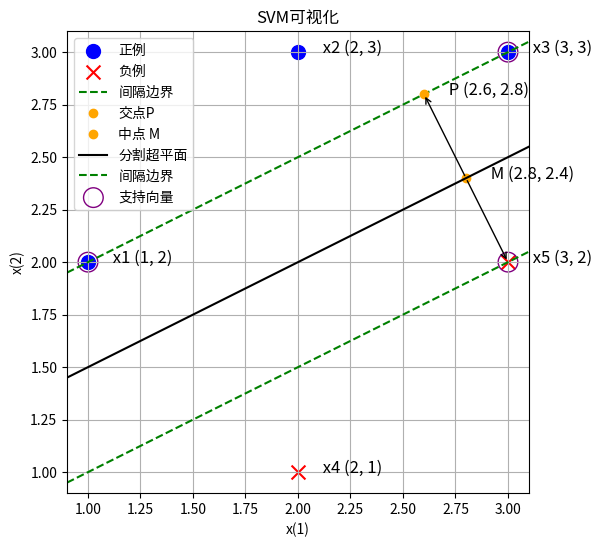
#### 1.2.1 解法一：几何观察法（考试写填空题）

由图像可知，分离超平面应当在x4和x5构成的直线的上方，x1和x3构成直线的下方。

注意到这两条直线的斜率不同。如果是x1和x4的中线作为分割超平面，x3无法分对。

所以应当是x1和x3联合起来一起作为正例的支持向量，然后x5对他们构成的直线做垂直线，然后垂直线的中线就是最大间隔的分割超平面。

fig, ax = plot\_binary\_classification\_2d(X, y, title="SVM可视化")  
# 绘制x1x3构成的直线  
x1, x3 = X[0], X[2]  
line\_x = np.array([x1[0], x3[0]])  
line\_y = np.array([x1[1], x3[1]])  
# ax.plot(line\_x, line\_y, color='green', linestyle='--', label='间隔边界')  
ax.axline((x1[0], x1[1]), (x3[0], x3[1]), color='green', linestyle='--', label='间隔边界')  
  
# 计算x1x3直线的斜率和截距  
slope = (line\_y[1] - line\_y[0]) / (line\_x[1] - line\_x[0])  
intercept = line\_y[0] - slope \* line\_x[0]  
  
# 计算x5到x1x3直线的垂线交点P  
x5 = X[4]  
perpendicular\_slope = -1 / slope  
perpendicular\_intercept = x5[1] - perpendicular\_slope \* x5[0]  
p\_x = (intercept - perpendicular\_intercept) / (perpendicular\_slope - slope)  
p\_y = slope \* p\_x + intercept  
  
# 绘制垂线交点P  
ax.plot(p\_x, p\_y, 'o', color='orange', label='交点P')  
ax.text(p\_x + 0.1, p\_y, f' P ({p\_x}, {p\_y})', fontsize=12, ha='left')  
  
# 计算垂线中点  
mid\_x = (p\_x + x5[0]) / 2  
mid\_y = (p\_y + x5[1]) / 2  
ax.plot(mid\_x, mid\_y, 'o', color='orange', label='中点 M')  
ax.text(mid\_x + 0.1, mid\_y, f' M ({mid\_x}, {mid\_y})', fontsize=12, ha='left')  
  
  
# 绘制箭头：中点到交点  
ax.annotate('', xy=(p\_x, p\_y), xytext=(mid\_x, mid\_y),  
 arrowprops=dict(facecolor='orange', arrowstyle="->"))  
  
# 绘制箭头：中点到x5  
ax.annotate('', xy=(x5[0], x5[1]), xytext=(mid\_x, mid\_y),  
 arrowprops=dict(facecolor='orange', arrowstyle="->"))  
  
# 绘制Px5的中垂线  
mid\_x = (p\_x + x5[0]) / 2  
mid\_y = (p\_y + x5[1]) / 2  
mid\_slope = -(x5[0] - p\_x) / (x5[1] - p\_y)  
mid\_intercept = mid\_y - mid\_slope \* mid\_x  
x\_values = np.linspace(mid\_x - 1, mid\_x + 1, 100)  
y\_values = mid\_slope \* x\_values + mid\_intercept  
# ax.plot(x\_values, y\_values, color='black', linestyle='-', label='分割超平面')  
ax.axline((mid\_x, mid\_y), slope=mid\_slope, color='black', linestyle='-', label='分割超平面')  
  
  
  
# 计算过x5点且斜率与x1x3相同的直线方程  
x5\_line\_x = np.array([x5[0], x5[0] + 1]) # 创建一个x值数组，第二个点x值比x5大1  
x5\_line\_y = slope \* x5\_line\_x + (x5[1] - slope \* x5[0]) # 计算对应的y值  
  
# 绘制过x5点的直线  
# ax.plot(x5\_line\_x, x5\_line\_y, color='green', linestyle='--', label='间隔边界')  
# 绘制过x5点且斜率与x1x3相同的直线  
ax.axline((x5[0], x5[1]), slope=slope, color='green', linestyle='--', label='间隔边界')  
  
# 标注支持向量  
  
support\_vector\_indices = [1, 3, 5] # 假设x1, x3, x5的索引分别是0, 2, 4  
for idx in support\_vector\_indices:  
 i = idx-1  
 ax.scatter(X[i, 0], X[i, 1], marker='o', facecolors='none',   
 edgecolors='purple', s=100\*2, # color='purple',  
 label='支持向量' if i == 0 else "")  
  
# 添加图例  
ax.legend()  
# ax.set\_xlim([0, 6])  
# ax.set\_ylim([0, 4])  
fig.set\_size\_inches(9, 6)  
  
# 这一句非常重要，要不然图中的垂直线都不垂直  
ax.set\_aspect('equal', adjustable='box') # https://stackoverflow.com/questions/17990845/how-do-i-equalize-the-scales-of-the-x-axis-and-y-axis  
  
ax



|  |
| --- |
| 注记 |
| 本解法不太好写在考试的解答题里面，可以写在选择题填空题里面加快速度。不对于AI研究而言，关键是无法适用于高维的情况自动求解。 |

#### 1.2.2 解法二：手算推导法（考试写解答题）

我们不用对偶问题，直接用原始问题来求。也就是

$$
\min\_{w, b} \frac{1}{2} ||w||^2 \\
s.t. y\_i (w \cdot x\_i + b) -1 \ge 0 , \forall i = 1,2,...,N\\
$$

记 w = (A, B), b=C

$$
A+2B+C \ge 1 \\
2A+3B+C \ge 1 \\
3A+3B+C \ge 1 \\
2A+B+C \le -1 \\
3A+2B+C \le -1 \\
$$

from sympy import symbols, solve, And

# 定义变量  
A, B, C = symbols('A B C')  
l = [A, B, C]  
  
# 定义不等式组  
inequalities = [  
 A + 2\*B + C >= 1,  
 2\*A + 3\*B + C >= 1,  
 3\*A + 3\*B + C >= 1,  
 2\*A + B + C <= -1,  
 3\*A + 2\*B + C <= -1  
]  
  
# 解不等式组  
solutions = [solve(inequalities, (i)) for i in l]

solutions[0]

solutions[1]

solutions[2]

可见这个优化问题对于考场上来说其实是不现实的，太多可能性了，没法推导出具体的东西。我们可以用凸二次规划来求解。

import numpy as np  
from scipy.optimize import minimize

# 定义目标函数  
def objective(x):  
 A, B = x  
 return 0.5 \* (A\*\*2 + B\*\*2)  
  
# 定义线性不等式约束  
def constraint(x):  
 A, B = x  
 return np.array([  
 A + 2\*B + 1, # A + 2B + C >= 1，这里C默认为1  
 2\*A + 3\*B + 1, # 2A + 3B + C >= 1  
 3\*A + 3\*B + 1, # 3A + 3B + C >= 1  
 -2\*A - B - 1, # 2A + B + C <= -1  
 -3\*A - 2\*B - 1 # 3A + 2B + C <= -1  
 ])  
  
# 约束条件，类型为'ineq'表示不等式约束  
con = {'type': 'ineq', 'fun': constraint}  
  
# 初始猜测值  
x0 = [0, 0]  
  
# 调用 minimize 函数求解  
sol = minimize(objective, x0, constraints=con, method='SLSQP')  
  
# 输出结果  
if sol.success:  
 print('最优解：', sol.x)  
 print('目标函数的最小值：', sol.fun)  
else:  
 print('求解失败')

最优解： [-0.66666667 0.33333333]  
目标函数的最小值： 0.2777777777777774

由于解法1已经进行了详细可视化，这里我们只是计算确认了一下，可视化的结果与解法1一样，因而不再赘述重画。

我们还可以使用对偶问题，然后观察 是否严格大于0，来发现支持向量，然后再用解法1中作中垂线的方法来求分离超平面，这里篇幅有限，就不再详细介绍了。

#### 1.2.3 解法三：使用Python和机器学习库（科研使用）

参考 https://scikit-learn.org/0.19/auto\_examples/svm/plot\_iris.html

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import svm, datasets

[source](https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data/blob/main/thu_big_data_ml/svm/vis.py#L72)

try\_svm\_and\_plot\_for\_binary\_2d

try\_svm\_and\_plot\_for\_binary\_2d (X, y, C=1.0, x\_names=['x(1)', 'x(2)'])

|  | **Type** | **Default** | **Details** |
| --- | --- | --- | --- |
| X |  |  |  |
| y |  |  |  |
| C | float | 1.0 | 软间隔的惩罚项, >0 |
| x\_names | list | [‘x(1)’, ‘x(2)’] |  |

[source](https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data/blob/main/thu_big_data_ml/svm/vis.py#L66)

plot\_contours

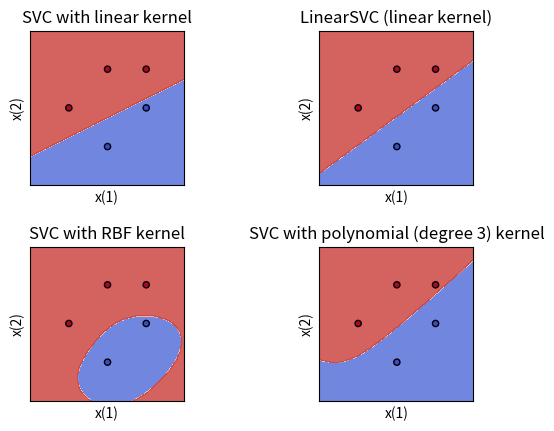
plot\_contours (ax, clf, xx, yy, \*\*params)

[source](https://github.com/Open-Book-Studio/THU-Coursework-Machine-Learning-for-Big-Data/blob/main/thu_big_data_ml/svm/vis.py#L59)

make\_meshgrid

make\_meshgrid (x, y, h=0.02)

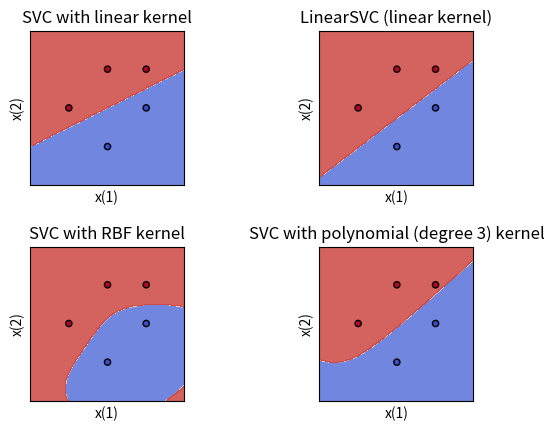
try\_svm\_and\_plot\_for\_binary\_2d(X, y)  
pass



上面的两个图是是线性支持向量机，理论上应该和我们上面的结果一样，然而看起来却有点不同。

这是因为我们使用的是软间隔支持向量机，C参数设置为1太小了，C趋近于无穷的时候，才和硬间隔情况一样。我们修改C重试一下。

try\_svm\_and\_plot\_for\_binary\_2d(X, y, C=1000000)  
pass



这一次图1和我们的结果是一样的了。 由于解法1中可视化很全面，我们对于支持向量、分割平面不再赘述重画。

## 2 第二题——理解软间隔支持向量机以及其对偶形式

题目如下 > 线性支持向量机还可以定义为以下形式: >

>

$$s.t.\ y\_i(w\cdot x\_i+b)\geq1-\xi\_i,\ i=1,2,\cdots,N \\ \xi\_i\geq0,\ i=1,2,\cdots,N$$

> > 试求其对偶形式。

### 2.1 审题

* 题目中“还可以定义为以下形式”是什么意思？题目中给的形式是李航书上的哪个形式的变种吗？
  + 实际上，题目给的形式是李航书上软间隔线性支持向量机的凸二次规划问题（原始问题）的形式的变种。
  + 统计学习书126页给出的优化问题是
  + 约束条件与本题一样，但是优化目标本题变种形式是，原本的形式是
* 变种问题有什么意义？
  + 引入软间隔的时候，我们约束条件都是说，用每个样本自己的松弛变量（slack variable）把它拉回正轨，让函数间隔大于1。
  + 为了防止学习器摆烂，直接用松弛变量结束游戏，最小化的目标问题中加入惩罚项，惩罚松弛变量。
  + 所以具体怎么惩罚是可以人为规定的。
  + 函数间隔正是分对了，负是分错了，负的多的分错的多，需要拉回1所需要的松弛变量大。松弛变量是正数，因为不能反着拉。
  + 如果分对了，松弛变量
  + 李航的书缺乏proper reference，所以读者只好自己找找是谁提出了这个变种。
    - 论文指出，https://orsj.org/wp-content/or-archives50/pdf/e\_mag/Vol.60\_03\_262.pdf
    - slack varible的惩罚项平方对于非线性问题的优化更好。
    - 但是这个这里不是针对SVM讲的，我不确定SVM里面的slack variable与论文中说的一般的优化问题的是否是一个含义。
* 在目前常用的SVM里面用到的是哪个形式？
  + 根据sklearn，https://scikit-learn.org/stable/modules/svm.html#svc
  + SVC使用的还是书上的那个普通形式，不是本题的形式。

### 2.2 解题

对于原问题 >

>

$$s.t.\ y\_i(w\cdot x\_i+b)\geq1-\xi\_i,\ i=1,2,\cdots,N \\ \xi\_i\geq0,\ i=1,2,\cdots,N$$

构建拉格朗日函数。为此，为不等式约束引入一组拉格朗日乘子 。

那么原问题就等价于求解让

最小的那个 值， 因为这相当于遍历了所有的 组合，如果不符合约束 一下子就可以飞到正无穷，然后把整个值变的很高，这样的 自然就被排除了，剩下的满足约束条件的, 不敢轻举妄动，只好点出最优值0, 这个情况下搜索就和原本的一模一样，同时我们还去掉了约束（引入了拉格朗日乘子 的新约束，当然也别忘了）。

而原问题的对偶问题就是翻转一下极小极大为极大极小，即：

假如外围问题遍历过程中给定了 的具体值，我们就可以求解里面的问题，这个时候暂时把当做常数，而里面的问题可以求导来求解。

令下面式子为0

其中第三个式子和统计学习127页的普通形式不同，我们的是平方过的，所以求导后还和 有关。 还要注意第三个式子是很容易犯错的，最好对每一个具体的 求导，而不是对一个整体向量 求导。

得

由于求导数为0是取得极值的必要条件，所以在取得min的情况下，可以代入上面的式子回到L，所以可以说

由于 是常数，很多项对于优化无用, 所以

$$\argmin\_{w,b, \xi} L(w, b, \xi, \alpha, \beta) = \argmin\_{w,b} M(w, b)$$

我们定义

平常确实是这样，但是这里我们其实不能消去，因为 我们下一步要最大化函数，所以我们只能老老实实代入原本的L。

经过一番激烈化简，我们得到 在min的情况下，

注意和统计学习书127页比，多了

接下来 求解

等价于求解

完整地来说还有乘子以及刚才求导带来的约束条件

$$
\sum\_{i=1}^N \alpha\_i y\_i = 0\\
\alpha\_i \geq 0\\
\beta\_i \geq 0\\
\xi\_i = \frac{1}{2\cdot C} (\alpha\_i + \beta\_i) \ge 0
$$

注意之所有有最后一个式子，是因为 也是一个没有被处理掉的约束条件，所以我们需要这样来重新表达为对 的约束条件，如果这个忘记了式子就不对了。不过我们马上就发现, 惩罚代价严格大于0，所以这个约束自然是满足的，当场就可以去掉。

李航书上最后把 去掉了，简化了问题为的最大化问题，这里我们能不能一样操作呢？答案是肯定的。

我们注意到优化问题 其实对于 来说很简单，因为最后一项是正数，所以 其他东西给定时，就可以让问题取到最大，同时这永远不会违反约束条件。所以我们的问题又可以进一步简化到只和 有关！

$$
\max\_{\alpha} -\frac{1}{2}\sum\_{i=1}^{N}\sum\_{j=1}^{N}\alpha\_i\alpha\_jy\_iy\_j(x\_i\cdot x\_j)+\sum\_{i=1}^{N}(\alpha\_i-\frac{1}{4C}\alpha\_i^2) \\
s.t. \sum\_{i=1}^N \alpha\_i y\_i = 0\\
\alpha\_i \geq 0\\
$$

上面这个就是软间隔支持向量机变种形式的原始问题的对偶问题化简后的形式！

李航书上进一步推导了一下此时对应的分割超平面的形式，这个超平面的分类决策函数的式子才叫做SVM的对偶形式，所以我们还要再推导一下。

由于问题仍然是凸二次规划问题，所以根据书上定理C.2， 对偶问题和原问题也是等价的，根据C.3还额外可以满足KKT条件(约束满足、刚才的求导=0满足，而且额外地，要不乘子为0，要不约束为0)，所以类似于 书上定理7.3， $$ w^\* = \_{i=1}^N \_i^\* y\_i x\_i \

\_i[y\_i(wx\_i+b)-1+\_i] = 0 \

\_i\_i = 0 $$ 其中后面两个式子是KKT条件特别的部分，告诉我们要不约束是0，要不乘子为0。

这里和书上有所不同， 不一定是0,

如果存在一个 (即支持向量), 那么$y\_i(wx\_i+b)-1+\_i = y\_i(wx\_i+b)-1+\_i =0 $ 我们可以根据支持向量去反推出 。

所以分类决策函数为

为题目所求对偶形式。

### 2.3 题目扩展问题