

Support Vector Machine

Wenfeng Luo, 18214551

本文讨论支持向量机 (Support Vector Machine) 分类器中涉及的二次规划问题。章节1列出本文用到的符号表示；章节2形式化定义支持向量机分类器涉及的优化问题，包括线性可分以及不可分两种情形；章节3讨论如何通过拉格朗日对偶性得到等价的仅关于拉格朗日乘子的对偶优化问题；章节4介绍两种优化算法用于求解相关的对偶问题，分别是简化版的序列最小化算法 (Sequential Minimal Optimization) 和基于二阶梯度信息的改进版本；章节5简单介绍实现细节；章节6给出了上述优化算法的实验结果，并比较两种优化算法的运行效率。

1 Notation

在深入讨论之前，先声明本文用到的一些符号规定：

- 训练数据： $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}, x^{(i)} \in R^n, y^{(i)} \in \{1, -1\}$
- 模型参数： w, b
- 分类器： $h_{w,b}(x) = g(w^T x + b)$ 。如果 $g(z) > 0$ ，预测为 1，否则预测为-1。

2 Problem Formulation

2.1 Linearly Separable

当我们的数据集线性可分时 (如图1)，我们希望得到这样一个决策边界 P_0 ，它到最近的正样本和负样本的距离是最远的。我们可以形式化为对应边界的两条直线 P_1 和 P_2 之间的距离最大。容易知道两直线的距离为：

$$\text{margin} = \frac{k - (-k)}{\sqrt{w_1^2 + \dots + w_n^2}} = \frac{2k}{\|w\|_2} \quad (1)$$

于是我们可以得到第一个优化问题

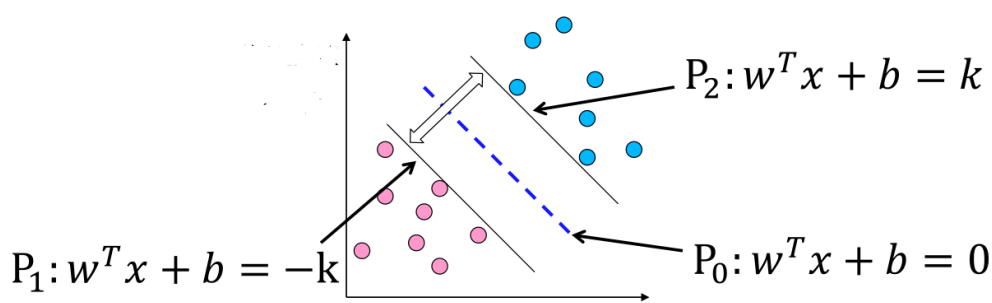


Figure 1: 最大间隔分类器

$$\begin{aligned}
 \max_{w,b,k} \quad & \frac{2k}{\|w\|_2} \\
 \text{s.t.} \quad & w^T x^{(i)} + b \geq k \text{ for } y^{(i)} = 1 \\
 & w^T x^{(i)} + b \leq -k \text{ for } y^{(i)} = -1 \\
 & k > 0
 \end{aligned} \tag{2}$$

简化一下，可以把正负训练样本的约束条件写成统一的式子

$$\begin{aligned}
 \max_{w,b,k} \quad & \frac{2k}{\|w\|_2} \\
 \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq k \quad i = 1, \dots, m \\
 & k > 0
 \end{aligned} \tag{3}$$

通过简单的变量替换 (取 $w' = \frac{w}{k}, b' = \frac{b}{k}$)，我们可以发现 k 是无关紧要的。另外，我们还可再变换一下形式，因为优化某个函数的最大值等价优化该函数的倒数的最小值，并且这样做以后方便我们对参数 w 的求导。于是可以得到如下的等价优化问题

$$\begin{aligned}
 \min_{w,b} \quad & \frac{1}{2} w^T w \\
 \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad i = 1, \dots, m
 \end{aligned} \tag{4}$$

2.2 Linearly Non-separable and Regularization

上述的讨论只适用于数据集线性可分的情况，但很多时候我们不能找到一条直线将数据集完美地划分正负样本，如图2(a)。此外，有些时候我们也不期望分分类器完美地划分训练集，比如训练集有部分离群点的情形，这些离群

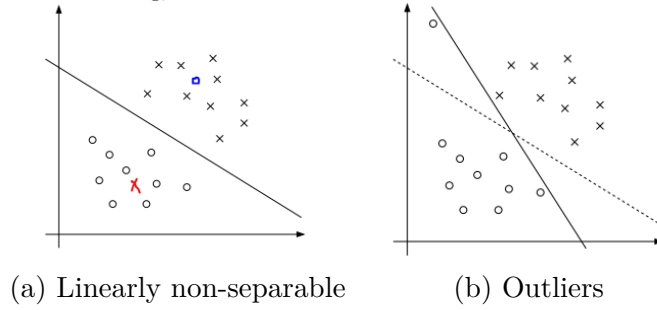


Figure 2: 线性不可分和离群点的情形

点对决策边界起极大的扰动作用，如图2(b)。所以，我们可以考虑让某些点的约束条件不严格成立，同时对不成立的约束条件引入某种惩罚项，比如 l_1 正则，于是可以在原来的基础上得到新的优化问题

$$\begin{aligned}
 \min_{w, b, \xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^m \xi_i \\
 \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geq 1 - \xi_i \quad i = 1, \dots, m \\
 & \xi_i \geq 0 \quad i = 1, \dots, m
 \end{aligned} \tag{5}$$

其中， C 是一个预定义好的大于 0 的常数，它决定了前后两项损失的权重，类似于线性规划的正则参数。很明显，优化问题 (5) 是一个典型的二次规划问题，我们已经可以用一般的商用包求解该问题，但是在训练样本比较大的时候求解效率是比较低下的。接下来我们考虑如何得到该问题的对偶问题，然后再讨论一个求解该对偶问题的高效迭代算法。

3 Lagrange Duality

这一章节我们考虑使用拉格朗日对偶得到优化问题 (5) 的对偶问题。先写出原问题的拉格朗日函数

$$\begin{aligned}
 L(w, b, \xi, \alpha, \beta) = & \frac{1}{2} w^T w + C \sum_{i=1}^m \xi_i \\
 & - \sum_{i=1}^m \alpha_i \left(y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i \right) - \sum_{i=1}^m \beta_i \xi_i
 \end{aligned} \tag{6}$$

其中，我们要求不等式约束对应的乘子 $\alpha_i \geq 0, \beta_i \geq 0, i = 1, \dots, m$ 。分别对原问题的决策变量求导并置零，可以得到：

$$\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \quad (7)$$

$$\frac{\partial L}{\partial b} = 0 \rightarrow \sum_{i=1}^m \alpha_i y^{(i)} = 0 \quad (8)$$

$$\frac{\partial L}{\partial \xi_i} = 0 \rightarrow \beta_i = C - \alpha_i \quad (9)$$

从最后一个等式我们还可以得到 α_i 的另外一个约束，因为 $\beta_i \geq 0$ ，因此 $0 \leq \alpha_i \leq C$ 。利用上面的等式关系回代回拉格朗日函数，可以得到只与乘子 α 有关的对偶函数：

$$g(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)} \rangle \quad (10)$$

根据对偶性，原问题等价于求解如下的对偶问题

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \quad i = 1, \dots, m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0 \end{aligned} \quad (11)$$

为了判别算法的收敛性，我们可以列出上述问题的 KKT 条件

- $\alpha_i = 0, y^{(i)}(w^T x^{(i)} + b) \geq 1$: $x^{(i)}$ 在边界以外；
- $0 < \alpha < C, y^{(i)}(w^T x^{(i)} + b) = 1$: $x^{(i)}$ 刚好好在边界上面；
- $\alpha_i = C, y^{(i)}(w^T x^{(i)} + b) \leq 1$: $x^{(i)}$ 在边界以内。

这里说的边界就是图1中的直线 P_1 和 P_2 。上述的条件非常重要，因为在后续介绍的 SMO 算法就是选取那些没有满足上述条件的乘子 α_i 来进行优化。

4 SMO algorithm

优化问题 (4, 5, 11) 都是二次规划问题，可以用通用的程序求解，但时间效率比较低。下面讨论如何在有约束条件下使用坐标下降方法高效快速求解

关于拉格朗日乘子的优化问题 (11)，求解到最优的 α^* 以后，我们可以通过如下方式恢复原来的参数

$$\begin{aligned} w &= \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \\ b &= -\frac{1}{2} \left(\min_{y^{(i)}=1} w^T x^{(i)} + \max_{y^{(i)}=-1} w^T x^{(i)} \right) \end{aligned} \quad (12)$$

4.1 章节简单介绍坐标下降方法的相关知识；4.2 章节讲解序列最小化算法 (Sequential Minimal Optimization) 的一个简单实现版本；4.3 章节介绍 SMO 的一种改进版本，该版本使用了二阶信息去选取优化的拉格朗日乘子，比原有的简单实现更高效快速，而且能确保收敛到最优值。

4.1 Coordinate Descent

首先我们考虑一个无约束的优化问题

$$\max_{\alpha} f(\alpha_1, \alpha_2, \dots, \alpha_n) \quad (13)$$

坐标下降方法每次只选取其中一个决策变量而固定其他所有的变量进行优化，这样以来我们每一次迭代需要求解的子问题就变得很简单，只是关于一个一元函数的最小化。另外，如果这个一元函数有特殊的结构，比如是关于 α_i 的二次函数，那么子问题的优化是有解析解的，因此可以很容易求解。我们可以写出坐标下降方法的简单过程

Algorithm 1 Coordinate Descent

```

1: Initialize all  $\alpha$ 's
2: while Not convergent do
3:   for  $i \leftarrow 1$  to  $n$  do
4:      $\alpha_i = \operatorname{argmax}_{\hat{\alpha}_i} f(\alpha_1, \dots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_n)$ 
5:   end for
6: end while
```

当然，我们还可以借助一些启发式的策略来决定决策变量的优化顺序，以加快算法收敛速率。

4.2 Slow Version

要将上述的坐标下降方法用于我们的优化问题，我们还得考虑问题 (11) 中的等式约束。由于问题中的等式约束，我们每次优化的时候如果只选取一个

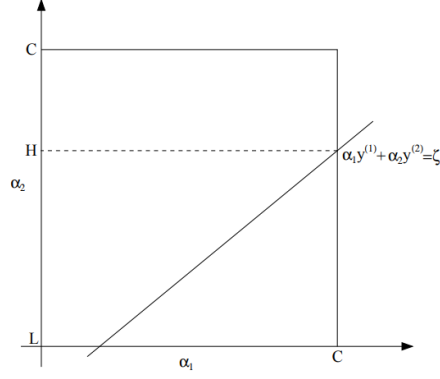


Figure 3: 乘子 α_1, α_2 的约束条件

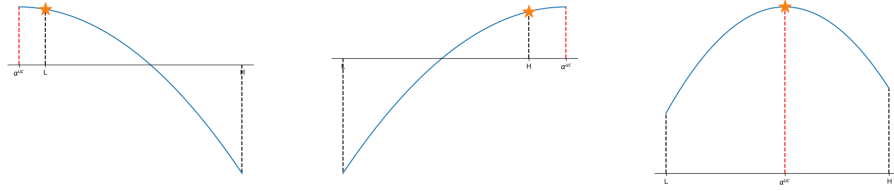
乘子，那么我们没有优化的余地了。SMO 算法的思想就是每次选取两个乘子进行优化，这时候我们要解决的子问题就是简单的二次函数的优化。举个例子，如果我们选取 α_1, α_2 为优化变量而固定其他的乘子，利用问题 (11) 的等式约束可以得到这两个乘子之间的关系为

$$\begin{aligned} \alpha_1 y^{(1)} + \alpha_2 y^{(2)} &= \zeta \\ \zeta &= - \sum_{i=3}^m \alpha_i y^{(i)} \end{aligned} \quad (14)$$

ζ 在优化 α_1, α_2 时保持不变。另外，我们还要考虑拉格朗日乘子的框式约束，图3描述了两个乘子在框式约束下的有效区域。利用等式关系替换 $\alpha_1 = y^{(1)}(\zeta - \alpha_2 y^{(2)})$ ，因此我们只需要考虑 α_2 的约束条件以使得最优解有效。考虑不同的情形， α_2 的上下界分别为

1. $y^{(1)} = y^{(2)}$ 时， $L = \max(0, \alpha_1 + \alpha_2 - C), H = \min(C, \alpha_1 + \alpha_2)$;
2. $y^{(1)} \neq y^{(2)}$ 时， $L = \max(0, \alpha_2 - \alpha_1), H = \min(C, C + \alpha_2 - \alpha_1)$ 。

综上所述，我们需要在 $[L, H]$ 区间内最优化一个关于 α_2 的二次函数。另外，由点积的性质我们知道 $2\langle x, y \rangle - \langle x, x \rangle - \langle y, y \rangle \leq 0$ ，因此这个二次函数的二次项系数是小于等于 0 的。记 α_2^{uc} 为对称轴的位置，有如下三种情况



$$(1) \alpha_2^{uc} < 0$$

$$(2) \alpha_2^{uc} > C$$

$$(3) 0 \leq \alpha_2^{uc} \leq C$$

上述三种情况可以用一个式子表示

$$\alpha_2^* = \min(\max(L, \alpha_2^{uc}), H) \quad (15)$$

下面给出 SMO 的简单实现版本 算法2中需要指定一个超参数 tol ，该超参

Algorithm 2 Simple Implementation of SMO

Input: $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Output: w, b

```

1: Initialize all  $\alpha$ 's to 0
2:  $\text{tol}=10, \text{iter}=0$ 
3: while  $\text{iter} < \text{tol}$  do
4:    $\text{iter} = \text{iter} + 1$ 
5:   for  $i \leftarrow 1$  to  $m$  do
6:     if KKT condition not satisfied for  $\alpha_i$  then
7:       Randomly choose another  $\alpha_j$ 
8:       Compute the lower bound  $L$  and upper bound  $H$  for  $\alpha_j$ 
9:       if  $L = H$  then
10:        break
11:      end if
12:       $\text{iter} = 0$ 
13:      Compute the unclipping value  $\alpha_j^{uc}$  for  $\alpha_j$ 
14:       $\zeta = \alpha_i^{\text{old}} y^{(i)} + \alpha_j^{\text{old}} y^{(j)}$ 
15:       $\alpha_j^{\text{new}} = \min(\max(L, \alpha_j^{uc}), H)$ 
16:       $\alpha_i^{\text{new}} = y^{(i)}(\zeta - \alpha_j^{\text{new}} y^{(j)})$ 
17:    end if
18:  end for
19: end while
20: Use equation (12) to Compute  $w, b$  from  $\alpha$ 's

```

数表明当连续遍历整个数据集 tol 次的时候，都没有发现可以更新的乘子了，

那么算法终止。当完成拉格朗日乘子的优化以后，我们通过公式12计算模型原来的参数 w, b 。得到模型的参数以后，就可以对新的样本进行预测了。

4.3 Fast Version

算法2由于只是简单地随机选取第二个乘子，因此效率比较低。这一章节介绍 libsvm 库中基于二阶导数信息的乘子选取方式。在讨论新的选取乘子方法之前，我们先定义一些符号：

$$\begin{aligned} a_{ij} &= K_{ii} + K_{jj} - 2K_{ij} \\ b_{ij} &= -y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j \\ I_{up}(\alpha) &= \{t | \alpha_t < C, y_t = 1 \text{ or } \alpha_t > 0, y_t = -1\} \\ I_{low}(\alpha) &= \{t | \alpha_t < C, y_t = -1 \text{ or } \alpha_t > 0, y_t = 1\} \end{aligned} \quad (16)$$

上面的符号中， K 即核矩阵，如果是简单的线性和函数，其中的元素 K_{ij} 即为 $\langle x^{(i)}, x^{(j)} \rangle$ 。 I_{up} 和 I_{low} 表示两类不同的违背了 KKT 条件的乘子集合。除此之外，由于某些核函数会导致非正定的核矩阵，libsvm 的实现中还对 a_{ij} 的值做了如下调整

$$\hat{a}_{ij} = \begin{cases} a_{ij} & a_{ij} > 0 \\ \tau & otherwise \end{cases} \quad (17)$$

τ 为一个非常接近于 0 的常量，一般取 $1e-12$ 。

借鉴 libsvm 库的实现，在每次迭代的时候，按照如下方式选取两个乘子：

$$\begin{aligned} i &= \operatorname{argmin}_t \{y_t \nabla f(\alpha)_t | t \in I_{up}(\alpha)\} \\ j &= \operatorname{argmax}_t \{-\frac{b_{ii}}{\hat{a}_{it}} | t \in I_{low}, -y_t \nabla f(\alpha)_t < -y_i \nabla f(\alpha)_i\} \end{aligned} \quad (18)$$

基于上面的讨论，我们可以给出如下新的优化算法

5 Implementation

代码实现在附件中，也可以在 [github](https://github.com/irlyue/MLAlg)¹上查看。主要是两个文件：

- svm.py: 该模块实现了常见的核函数，包括线性核，多项式核以及高斯核；以及一个 SVM 类提供训练和测试接口。
- svm_solver.py: 这个模块主要实现了本文提到的两种优化算法，WSS1Solver 和 WSS3Solver。
- SVM Solver Comparison.ipynb: 涉及实验的代码。

¹<https://github.com/irlyue/MLAlg>

Algorithm 3 Improved Implementation of SMO

Input: $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ **Output:** w, b

```
1: Initialize all  $\alpha$ 's to 0
2: tol=10, iter=0
3: while True do
4:   Choose two multipliers  $\alpha_i, \alpha_j$  using equation 18
5:   if no such  $\alpha_i, \alpha_j$  exist then
6:     break
7:   end if
8:   Compute the lower bound  $L$  and upper bound  $H$  for  $\alpha_j$ 
9:   Compute the unclipping value  $\alpha_j^{uc}$  for  $\alpha_j$ 
10:   $\zeta = \alpha_i^{old} y^{(i)} + \alpha_j^{old} y^{(j)}$ 
11:   $\alpha_j^{new} = \min(\max(L, \alpha_j^{uc}), H)$ 
12:   $\alpha_i^{new} = y^{(i)}(\zeta - \alpha_j^{new} y^{(j)})$ 
13: end while
14: Use equation (12) to Compute  $w, b$  from  $\alpha$ 's
```

6 Experimental Results

数据集 实验使用了两个人造数据集，他们的详细描述如下：

1. 数据集 A: 线性可分的二分类数据，训练样本个数 100，实验时统一使用线性核函数；
2. 数据集 B: 线性不可分的二分类数据，训练样本个数 500，实验时统一使用高斯核函数。

图 (4) 给出了两个数据的可视化结果。

简单的二分类问题 图 (5) 给出了训练线性核函数的 SVM 在一个二分类数据集上的结果，正则参数 $C = 1$ 。

两种优化算法的运行效率比较 在这次实验中，我们比较两种算法在不同规模数据集下的性能。实验结果如表格 (1) 所示。可以看到，算法 3 的运行效率要远远优于算法 2。

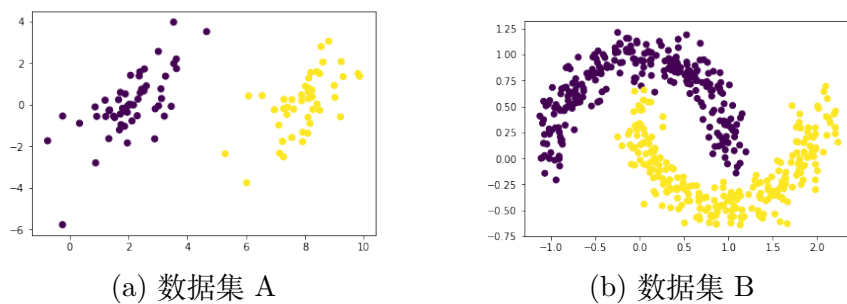


Figure 4: 二分类数据集

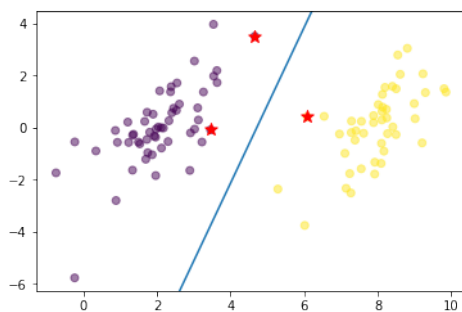


Figure 5: 线性 SVM 的决策边界 (蓝线) 以及支持向量 (红)

Table 1: 两种算法在不同规模数据集下的性能

优化算法	数据集 A		数据集 B	
	运行时间	支持向量的个数	运行时间	支持向量的个数
算法 2	103ms	5	5.24s	207
算法 3	5.9ms	3	21.3ms	207

7 References

1. CS229 Lecture notes: Support Vector Machines
2. Fast Training of Support Vector Machines using Sequential Minimal Optimization
3. Working Set Selection Using Second Order Information for Training Support Vector Machines
4. Machine Learning in Action, chapter 6