Support Vector Machine

January 15, 2019

1 Notation

- 训练数据：$\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}, x^{(i)} \in R^n, y^{(i)} \in \{1, -1\}$
- 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

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：

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

于是我们可以得到第一个优化问题

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

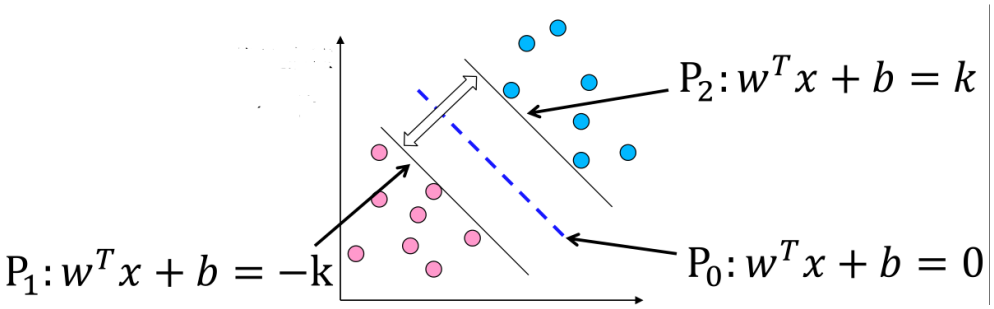


Figure 1: 最大间隔分类器

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

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

现在优化变量多了一个 k，我们可以稍微做一下变量替换就可以发现 k 是无关紧要的。取 $w' = \frac{w}{k}, b' = \frac{b}{k}$，我们可以得到如下的等价优化问题

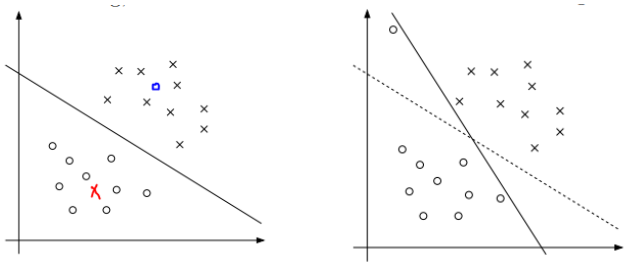
$$\begin{aligned} \max_{w',b'} \quad & \frac{2}{||w'||_2} \\ \text{s.t.} \quad & y^{(i)}(w'^T x^{(i)} + b') >= 1 \quad i = 1, ..., m \end{aligned} \tag{4}$$

可以看到，经过变量的替换，原有的 k 不再起作用。为了书写的方便，我们仍然使用 w, b 作为我们的优化变量，即

$$\begin{aligned} \max_{w,b} \quad & \frac{2}{||w||_2} \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) >= 1 \quad i = 1, ..., m \end{aligned} \tag{5}$$

另外，我们还再变换一下形式，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

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} w^T w \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) >= 1 \quad i = 1, ..., m \end{aligned} \tag{6}$$



(a) Linearly non-separable (b) Outliers

Figure 2: 线性不可分和离群点的情形

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

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) >= 1 - \xi_i \quad i = 1, ..., m \\ & \xi_i \geq 0 \quad i = 1, ..., m \end{aligned} \tag{7}$$

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 (7) 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。

2.3 Lagrange Duality

接着，我们讨论如何使用拉格朗日对偶性求解上述优化问题，先写出上述问题的拉格朗日函数并求偏导

$$
\begin{aligned}
L(w,b,\xi) &= \frac{1}{2}w^T w + C\sum_{i=1}^{m}\xi_i \\
&\quad - \sum_{i=1}^{m}\alpha_i\left(y^{(i)}(w^T x^{(i)}+b) - 1 + \xi_i\right) - \sum_{i=1}^{m}\beta_i\xi_i \\
\frac{\partial L}{\partial w} &= w - \sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)} \qquad (8) \\
\frac{\partial L}{\partial b} &= -\sum_{i=1}^{m}\alpha_i y^{(i)} \\
\frac{\partial L}{\partial \xi_i} &= C - \alpha_i - \beta_i
\end{aligned}
$$

这时我们可以将所有的导数置为 0:

$$\frac{\partial L}{\partial w} = 0 \rightarrow w = \sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)} \qquad (9)$$

$$\frac{\partial L}{\partial b} = 0 \rightarrow \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \qquad (10)$$

$$\frac{\partial L}{\partial \xi_i} = 0 \rightarrow \beta_i = C - \alpha_i \qquad (11)$$

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 $0 \leq \alpha_i \leq C$。接着利用相应的等式关系回代回去

$$\begin{aligned} L(w,b,\xi) =& \frac{1}{2}w^T w - \sum_{i=1}^{m}\alpha_i y^{(i)} w^T x^{(i)} - b\sum_{i=1}^{m}\alpha_i y^{(i)} + \\ & (C-\alpha_i-\beta_i)\sum_{i=1}^{m}\xi_i + \sum_{i=1}^{m}\alpha_i \\ =& \left(\sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)}\right)^T \left(\sum_{i=1}^{m}\alpha_i y^{(i)} x^{(i)}\right) - \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} \left(\sum_{j=1}^{m}\alpha_i y^{(j)} x^{(j)}\right)^T x^{(i)} + \sum_{i=1}^{m}\alpha_i \\ =& \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)}\rangle \end{aligned} \tag{12}$$

于是原始的拉格朗日函数转化为仅关于拉格朗日乘子 α 的函数。根据对偶性，原问题等价于求解如下的对偶问题

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)}\rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \quad i = 1, ..., m \\ & \sum_{i=1}^{m}\alpha_i y^{(i)} = 0 \end{aligned} \tag{13}$$

为了判别算法的收敛性，我们可以列出上述问题的 KKT 条件

$$\begin{aligned} (C-\alpha_i^*)\xi_i^* &= 0 \\ \xi_i^* &\geq 0 \\ \alpha_i^*\left(y^{(i)}(w^T x^{(i)} + b) - 1 + \xi_i^*\right) &= 0 \\ y^{(i)}(w^T x^{(i)} + b) &\geq 1 - \xi_i^* \end{aligned} \tag{14}$$

一个显然的问题是那么多变量在里面，怎么来进行收敛性判断？我们其实可以对 KKT 条件做进一步简化。实质上，ξ_i 的选取是比较自由的，在优化过程我们只要保证存在 ξ_i 满足条件即可，而不需要去明确知道具体的值。于是我们可以考虑在哪些情况下，KKT 条件是成立的：

1. 考虑 $\alpha_i = 0$ 的情形，那么第一个式子要成立，则需要对应的 $\xi_i = 0$。此时前面 3 个条件均成立了，但最后一个条件不能进一步简化了。于是第一种情况为：$\alpha_i = 0, \xi_i = 0, y^{(i)}(w^T x^{(i)} + b) \geq 1$；

2. 考虑 $0 < \alpha < C$ 的情形，那么很显然两个等式要成立，必然有 $\xi_i = 0, y^{(i)}(w^T x^{(i)} + b) = 1$，这样一来其实另外两个不等式也成立了。所以第二种情况为：$0 < \alpha < C, \xi_i = 0, y^{(i)}(w^T x^{(i)} + b) = 1$；

3. 最后是 $\alpha_i = C$ 的情况，那么对于第三个式子必然要有 $y^{(i)}(w^T x^{(i)} + b) - 1 = -\xi_i$。刚才讨论了，$\xi_i$ 的取值是比较自由的，只要存在 ξ_i 满足条件就可以了，故我们只要限制 $y^{(i)}(w^T x^{(i)} + b) - 1 \leq 0$ 即 $y^{(i)}(w^T x^{(i)} + b) \leq 1$ 就可以找到对应的 ξ_i。所以第三种情况为：$\alpha_i = C, \xi_i > 0, y^{(i)}(w^T x^{(i)} + b) \leq 1$。

综上所述，优化问题的 KKT 条件可以描述为

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

3 SMO algorithm

优化问题 (6, 7, 13) 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。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 (13)，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

$$
\begin{aligned}
w &= \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} \\
b &= -\frac{1}{2}\left(\min_{y^{(i)}=1} w^T x^{(i)} + \max_{y^{(i)}=-1} w^T x^{(i)} \right)
\end{aligned}
\tag{15}
$$

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，这个版本不保证算法结束以后得到最优解；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橙子，比原有的简单实现更高效快速，而且能确保收敛到最优值。

3.1 Coordinate Descent

首先我们考虑一个无约束的优化问题

$$\max_{\alpha} \quad f(\alpha_1, \alpha_2, ..., \alpha_n) \tag{16}$$

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 α's
2: **while** Not convergent **do**
3: **for** $i \leftarrow 1$ to n **do**
4: $\alpha_i = \text{argmax}_{\hat{\alpha_i}} \quad f(\alpha_1, ..., \alpha_{i-1}, \hat{\alpha_i}, \alpha_{i+1}, ..., \alpha_n)$
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。

3.2 Simple 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 (13) 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，那么这两个乘子之间的关系如下

$$\begin{aligned} \alpha_1 y^{(1)} + \alpha_2 y^{(2)} &= \zeta \\ \zeta &= -\sum_{i=3}^{m} \alpha_i y^{(i)} \end{aligned} \tag{17}$$

ζ 在优化 α_1, α_2 是保持不变。另外，我们还要考虑拉格朗日乘子的框式约束，可以使用图3 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

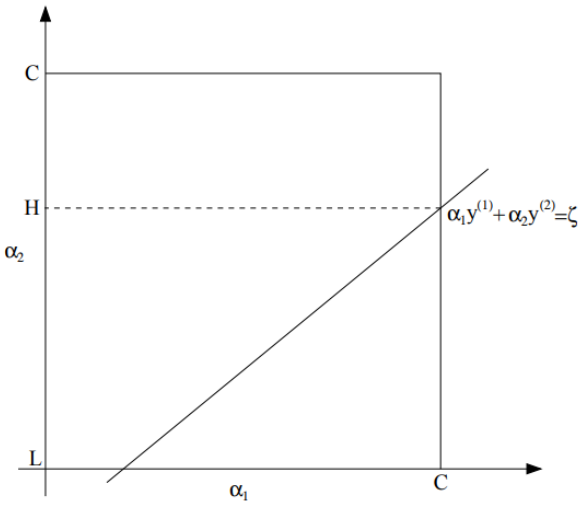
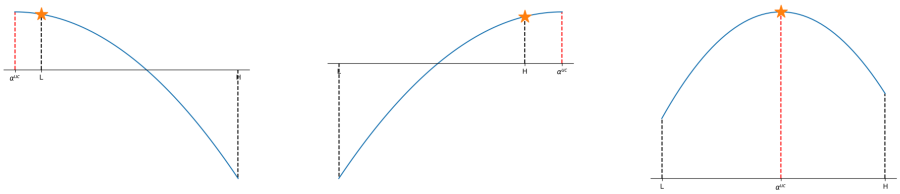


Figure 3: 乘子 α_1, α_2 的约束条件

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。考虑对称轴 α_2^{uc} 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

$$\alpha_2^* = \min(\max(L, \alpha_2^{uc}), H) \tag{18}$$

下面给出 SMO 的简单实现版本

Algorithm 2 Simple Implementation of SMO

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

Initialize all α's to 0
tol=10, iter=0
while iter $<$ tol **do**
    iter = iter + 1
    for $i \leftarrow 1$ to m **do**
        if KKT condition not satisfied for α_i **then**
            Randomly choose another α_j
            Compute the lower bound L and upper bound H for α_j
            if $L = H$ **then**
                break
            end if
            iter = 0
            Compute the unclipping value α_j^{uc} for α_j
            $\zeta = \alpha_i^{old} y^{(i)} + \alpha_j^{old} y^{(j)}$
            $\alpha_j^{new} = \min(\max(L, \alpha_j^{uc}), H)$
            $\alpha_i^{new} = y^{(i)}(\zeta - \alpha_j^{new} y^{(j)})$
        end if
    end for
end while
Use equation (15) to Compute w, b from α's

算法2中需要指定一个超参数 tol，该超参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15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。

3.3 Complete Version

算法2由于只是简单地随机选取第二个乘子，因此效率比较低下。在讨论新的选取乘子方法之前，我们先定义一些符号。

$$\begin{aligned} a_{ij} &= K_{ii} + K_{jj} - 2K_{ij} \\ b_{ij} &= -y_i \nabla f(\alpha)_i + y_j \nabla f(\alpha)_j \\ I_{up}(\alpha) &= \{t | \alpha_t < C, y_t = 1 \text{ or } \alpha_t > 0, y_t = -1\} \\ I_{low}(\alpha) &= \{t | \alpha_t < C, y_t = -1 \text{ or } \alpha_t > 0, y_t = 1\} \end{aligned} \tag{19}$$

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

$$\hat{a_{ij}} = \begin{cases} a_{ij} & a_{ij} > 0 \\ \tau & otherwise \end{cases} \tag{20}$$

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，我们可以使用如下的方式选取

$$\begin{aligned} i &= \text{argmin}_t \{ y_t \nabla f(\alpha) | t \in I_{up}(\alpha) \} \\ j &= \text{argmax}_t \{ -\frac{b_{ii}}{\hat{a_{it}}} | t \in I_{low}, -y_t \nabla f(\alpha)_t < -y_i \nabla f(\alpha)_i \} \end{aligned} \tag{21}$$

基于上面的讨论，我们可以给出如下新的优化算法

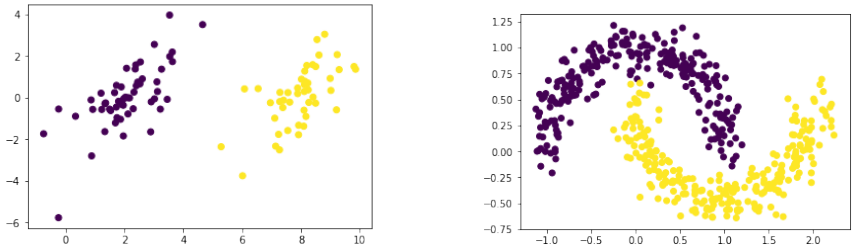
Algorithm 3 Improved Implementation of SMO

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

Initialize all α's to 0
tol=10, iter=0
while True **do**
    Choose two multipliers α_i, α_j using equation 21
    if no such α_i, α_j exist **then**
        break
    end if
    Compute the lower bound L and upper bound H for α_j
    Compute the unclipping value α_j^{uc} for α_j
    $\zeta = \alpha_i^{old} y^{(i)} + \alpha_j^{old} y^{(j)}$
    $\alpha_j^{new} = \min(\max(L, \alpha_j^{uc}), H)$
    $\alpha_i^{new} = y^{(i)}(\zeta - \alpha_j^{new} y^{(j)})$
end while
Use equation (15) to Compute w, b from α's

4 Implementation

代码实现在附件中，也可以在github上查看。主要是两个文件：



(a) 数据集 A　　(b) 数据集 B

Figure 4: 二分类数据集

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

- svm.py: 主要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。

5 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。

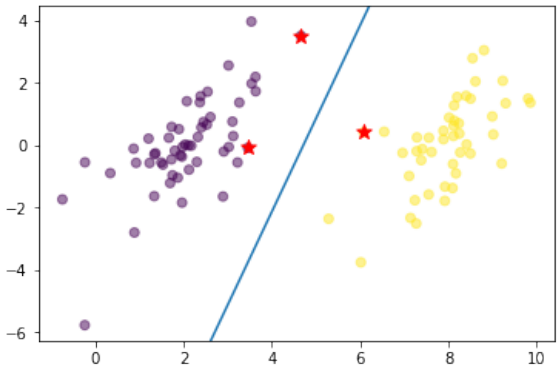


Figure 5: 线性 SVM 的决策边界 (蓝线) 以及支持向量 (红)

6 References

1. CS229 Lecture notes: Support Vector Machines
2. Working Set Selection Using Second Order Information for Training Support Vector Machines
3. Machine learning in action-chapter 6