SVM 的数学推导和 Python 实现

赵新锋

2021 年 8 月 5 日

摘要

支持向量机（support vector machines, SVM）是一种分类模型，该模型在特征空间中求解间隔最大的分类超平面。当训练数据近似线性可分时，可以通过增加软间隔学习一个线性分类器。当线性不可分时，利用核技巧，隐式的将特征空间映射到高维特征空间，从而达到线性可分。使用序列最小最优化算法（SMO），可以快速求解模型的参数。

关键词： 支持向量机; SVM; SMO; 矩阵运算; 矩阵求导;numpy;sklearn.

目录

1 数学推导与 python 实现

1.1 SVM 简介

当训练数据线性可分，可以得到一个线性超平面 $x \cdot w + b = 0$, 将在超平面上方的归为正类，将在超平面下方的归为负类。当数据点与超平面距离越远时，表示分类的确定性越高，这样虽然线性分类的超平面可能有无数多个，但是我们可以找到一个所有点距离超平面最大的一个超平面。相应的决策函数为:

$$f(x) = sign(x \cdot w^* + b^*)$$

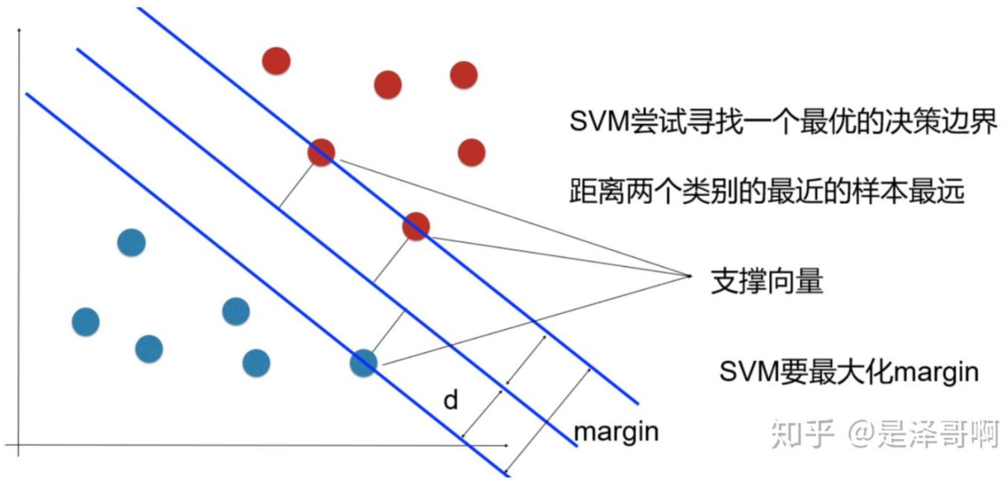

图 1: SVM 线性可分

1.2 最大间隔分类超平面

$x_i^T \cdot w_0 + b_0$ 为正时, y_i 为正, 当 $x_i^T \cdot w_0 + b_0$ 为负时, y_i 为负, 则可以定义 $\frac{y_i \cdot (x_i^T \cdot w_0 + b_0)}{\|w_0\|}$ 为几何间隔，表示数据点距离超平面的距离。模型最终会找到一个参数为 w_0 和 b_0 的分离超平面，所有点距离超平面的距离都大于等于 d，将距离正好等于 d 的数据点称之为支持向量。

$$\begin{aligned} \arg\max_{w_0, b_0} \; d = \frac{y_0 \cdot (x_0^T \cdot w_0 + b_0)}{\|w_0\|} \\ \text{s.t.} \quad \frac{y_i \cdot (x_i^T \cdot w_0 + b_0)}{\|w_0\|} \geq d \end{aligned} \tag{1}$$

将 w_0 和 b_0 进行一定比例的缩放

$$w = \frac{w_0}{y_0 \cdot (x_0^T \cdot w_0 + b_0)}$$
$$b = \frac{b_0}{y_0 \cdot (x_0^T \cdot w_0 + b_0)}$$

可以将 (1) 式化简为:

$$\begin{aligned}
&\arg\max_{w}\ d = \frac{1}{\|w\|} \\
\Leftrightarrow\ &\arg\min_{w}\ d = \|w\| \\
\Leftrightarrow\ &\arg\min_{w}\ d = \frac{1}{2}w^T \cdot w \qquad (2)\\
&\text{s.t.}\ \ y_i \cdot (x_i^T \cdot w + b) \geq 1 \qquad (3)
\end{aligned}$$

将(2) 和 (3) 利用拉格朗日乘数法，获得拉格朗日原始问题形式:

$$\begin{aligned}
\arg\min_{w,b}\max_{\alpha}\ L(w,b,\alpha) &= \frac{1}{2}w^T \cdot w - \alpha^T \cdot (y \odot (X \cdot w + b) - 1) \\
&= \frac{1}{2}w^T \cdot w - (\alpha \odot y)^T \cdot (X \cdot w + b) + \alpha^T \cdot \mathbf{1}^m \\
&= \frac{1}{2}w^T \cdot w - (\alpha \odot y)^T \cdot (X \cdot w + b) + \mathbf{1}^T \cdot \alpha \qquad (4)
\end{aligned}$$

当满足 KTT 条件时，拉格朗日对偶问题的解等价于(4) 的解:

$$\arg\max_{\alpha}\min_{w,b}\ L(w,b,\alpha) = \frac{1}{2}w^T \cdot w - (\alpha \odot y)^T \cdot (X \cdot w + b) + \mathbf{1}^T \cdot \alpha \qquad (5)$$

1.3 求解拉格朗日对偶问题

首先求 L 以 w、b 为参数的极小值，通过求微分得到偏导数形式。

$$\begin{aligned}
\mathrm{d}L = &\frac{1}{2}tr[(\mathrm{d}w)^T \cdot w + w^T \cdot \mathrm{d}w) - (\alpha \odot y)^T \cdot (X\mathrm{d}w) \\
&- (1^T \cdot (\alpha \odot y))^T \cdot \mathrm{d}b \\
&- (\mathrm{d}\alpha)^T \cdot (y \odot (X \cdot w + b) - 1)] \\
= &\, tr[w^T\mathrm{d}w - (X^T \cdot (\alpha \odot y))^T\mathrm{d}w - (\alpha \odot y)^T \cdot \mathrm{d}b - (y \odot (X \cdot w + b) - 1)^T\mathrm{d}\alpha]
\end{aligned}$$

从而得到 w、b 偏导数，并令偏导数为 0。

$$\begin{aligned}
\frac{\partial L}{\partial w} &= w - X^T \cdot (\alpha \odot y) = \mathbf{0} \\
\frac{\partial L}{\partial b} &= -1^T \cdot (\alpha \odot y) = -y^T \cdot \alpha = 0
\end{aligned}$$

推导出如下关系:

$$w = X^T \cdot (\alpha \odot y) \qquad (6)$$

$$y^T \cdot \alpha = 0 \qquad (7)$$

将(6)式和 (7)式代入(5)式中，

$$\begin{aligned}\arg\max_{\alpha} L(w,b,\alpha) &= \frac{1}{2}(\alpha\odot y)^T\cdot X\cdot X^T\cdot(\alpha\odot y)\\&-(\alpha\odot y)^T\cdot X\cdot X^T\cdot(\alpha\odot y)\\&-(\alpha\odot y)^T\cdot b^m\\&+\mathbf{1}^T\cdot\alpha\\&=-\frac{1}{2}(\alpha\odot y)^T\cdot X\cdot X^T\cdot(\alpha\odot y)+(\alpha\odot y)^T\cdot b+\mathbf{1}^T\cdot\alpha\\&=-\frac{1}{2}(\alpha\odot y)^T\cdot X\cdot X^T\cdot(\alpha\odot y)+\mathbf{1}^T\cdot\alpha\end{aligned}$$

去除负号，将极大转换成极小形式：

$$\begin{aligned}\arg\min_{\alpha} L(w,b,\alpha) &= \frac{1}{2}(\alpha\odot y)^T\cdot X\cdot X^T\cdot(\alpha\odot y)-\mathbf{1}^T\cdot\alpha\\\text{s.t.}\ \ y^T\cdot\alpha=0\end{aligned}\tag{8}$$

为了使拉格朗日对偶问题的解与原始问题解相同，需要同时满足 KTT 条件：

$$\begin{aligned}\frac{\partial L}{\partial w}&=0\\\frac{\partial L}{\partial b}&=0\\\alpha\odot(y\odot(X\cdot w+b)-1)&=\mathbf{0}^m\\y\odot(X\cdot w+b)-1&\geq\mathbf{0}^m\\\alpha&\geq\mathbf{0}^m\end{aligned}$$

1.4 软间隔

当训练数据近似线性可分，有些异常点或噪声点导致无法找到分离超平面，可以对每个数据点加一个松弛变量 ξ_i，从而让所有数据点均满足约束。

加入软间隔参数, 每个向量点距离分类超平面距离增加 ξ_i，同时增加一个惩罚系数 C, 代入(5)新形式如下：

$$\begin{aligned}\arg\min_{w}\ \ d=\frac{1}{2}w^T\cdot w+C\cdot\mathbf{1}^m\cdot\xi\\\text{s.t.}\ \ y_i\cdot(x_i^T\cdot w+b)\geq 1-\xi_i\\\xi_i\geq 0\end{aligned}$$

将其转换为拉格朗日对偶形式：

$$L(w,b,\xi,\alpha,\mu)=\frac{1}{2}w^T\cdot w+C\cdot\mathbf{1}^T\cdot\xi-\alpha^T\cdot(y\odot(X\cdot w+b)-1+\xi)-\mu^T\cdot\xi\tag{9}$$

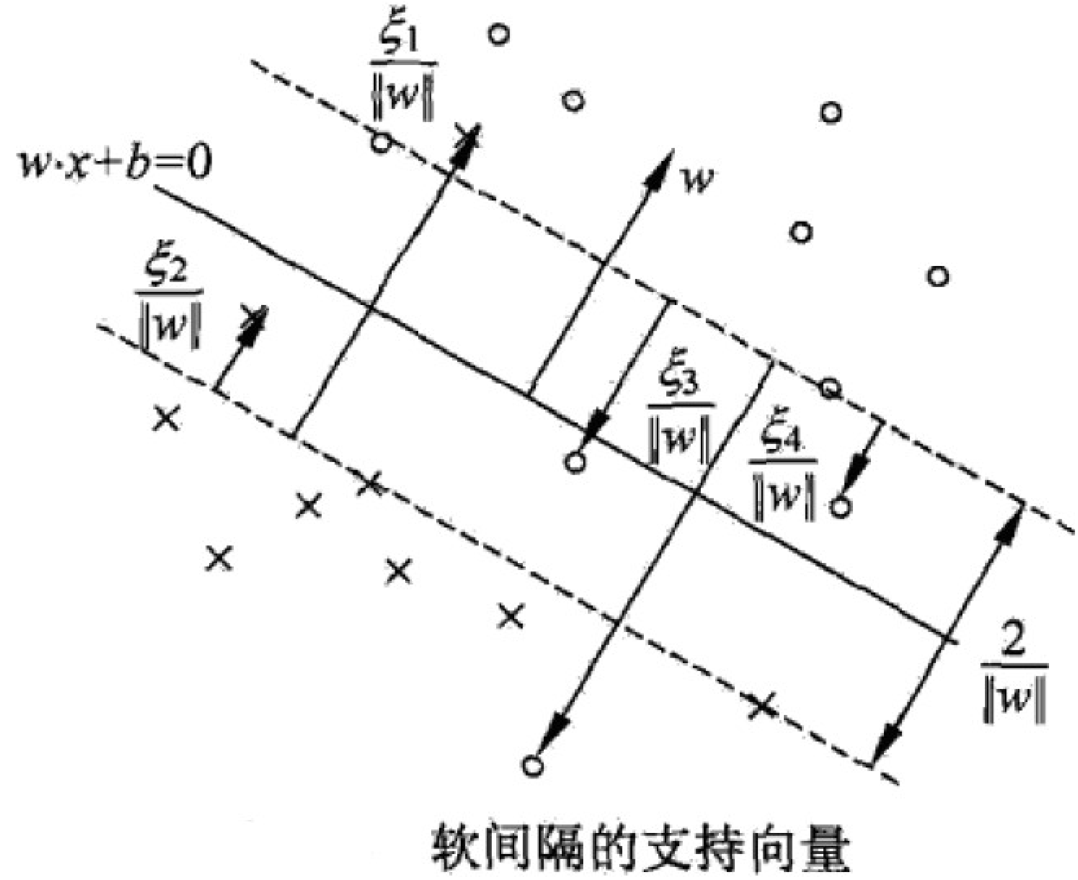
$\frac{\xi_1}{\|w\|}$
$w \cdot x+b=0$
w
$\frac{\xi_2}{\|w\|}$
$\frac{\xi_3}{\|w\|}$
$\frac{\xi_4}{\|w\|}$
$\frac{2}{\|w\|}$
软间隔的支持向量

图 2: SVM 软间隔

求得对于 w、b、ξ 的偏导并令其为 0：

$$\frac{\partial L}{\partial w} = w - X^T \cdot (\alpha \odot y) = \mathbf{0}$$
$$\frac{\partial L}{\partial b} = -1^T \cdot (\alpha \odot y) = -y^T \cdot \alpha = 0$$
$$\frac{\partial L}{\partial \xi} = C - \alpha - u = \mathbf{0}$$

代入 (9) 式中：

$$\begin{aligned}\arg\max_{\alpha} L(\alpha) = -\frac{1}{2}(\alpha \odot y)^T \cdot X \cdot X^T \cdot (\alpha \odot y) + \mathbf{1}^m \cdot \alpha \\ \text{s.t.}\quad & y^T \cdot \alpha = 0 \\ & C - \alpha - \mu = \mathbf{0} \\ & \alpha_i \geq 0 \\ & \mu_i \geq 0\end{aligned}$$

$C - \alpha - u = \mathbf{0}$、$\alpha_i \geq 0$ 、$u_i \geq 0$ 约束可以化简为：

$$\begin{aligned}\arg\min_{\alpha} L(\alpha) = \frac{1}{2}(\alpha \odot y)^T \cdot X \cdot X^T \cdot (\alpha \odot y) - \mathbf{1}^m \cdot \alpha \\ \text{s.t.}\quad & y^T \cdot \alpha = 0 \\ & 0 \leq \alpha_i \leq C \Leftrightarrow \mathbf{0}^m \leq \alpha \leq \boldsymbol{C}^m\end{aligned} \tag{10}$$

为了使拉格朗日对偶问题的解与原始问题解相同，需要同时满足 KTT 条件：

$$\begin{aligned}\frac{\partial L}{\partial w} &= 0 \\ \frac{\partial L}{\partial b} &= 0 \\ \frac{\partial L}{\partial \xi} &= 0 \\ \alpha \odot (y \odot (X \cdot w + b) - 1 + \xi) &= \mathbf{0}^m \\ y \odot (X \cdot w + b) - 1 + \xi &\geq \mathbf{0}^m \\ \alpha &\geq \mathbf{0}^m \\ \mu \odot \xi &= \mathbf{0}^m \\ \xi &\geq \mathbf{0}^m \\ \mu &\geq \mathbf{0}^m\end{aligned}$$

1.5 核函数

近似线性可分用软间隔方式解决，然而当训练数据是非线性数据，会出现无法在原特征空间找到分离超平面。可以使用一个非线性变换，将数据从原特征空间映射到更高

维的新空间，然后在新空间中寻找线性分类超平面，这种方法被称为核技巧。观察 (10) 式中计算 $X \cdot X^t$，即需要计算 $x_i \cdot x_i$ 内积。将核技巧应用到 SVM，定义核函数为：

$$K(x, z) = \phi(x) \cdot \phi(z)$$

即将原来的向量内积，改成先让向量映射到新空间，然后再求内积。核技巧的另外一个优点是，不需要显示的定义 ϕ 而是直接计算出 $\phi(x) \cdot \phi(z)$ 的结果，以高斯核函数为例：

$$K(x, z) = \exp(-\frac{\|x - z\|^2}{2\sigma^2})$$

则 (10) 式利用核技巧转化为：

$$\begin{aligned}
\arg\min_{\alpha} L(\alpha) = &\frac{1}{2}(\alpha \odot y)^T \cdot K(X, X) \cdot (\alpha \odot y) - \mathbf{1}^m \cdot \alpha \\
\text{s.t.}\ \ &y^T \cdot \alpha = 0 \\
&\mathbf{0}^m \leq \alpha \leq \boldsymbol{C}^m \\
&\alpha \odot (y \odot (X \cdot w + b) - 1 + \xi) = \mathbf{0}^m \\
&y \odot (X \cdot w + b) - 1 + \xi \geq \mathbf{0}^m \\
&\alpha \geq \mathbf{0}^m \\
&\mu \odot \xi = \mathbf{0}^m \\
&\xi \geq \mathbf{0}^m \\
&\mu \geq \mathbf{0}^m
\end{aligned} \tag{11}$$

1.6 序列最小最优化算法 SMO

支持向量机的拉格朗日对偶问题是一个凸二次规划问题，具有全局最优解，序列最小最优化算法即 SMO 算法，是高效求解支持向量机解的一种算法。其基本思路是，如果所有变量都满足了 KTT 条件，那么就求得了问题的解。SMO 算法过程如下：

- 选择两个变量，如 α_1, α_2，固定其他变量，那么原问题就变成了两个变量的二次优化问题。
- 由于有约束 $y^T \cdot \alpha = 0$ 的存在，选择了两个变量，实际上自由变量只有一个。
- 求解两个变量的最优解，迭代直到所有变量满足 KTT 条件。
- 迭代求解使用的解析方法，效率高

当选择两个变量，如 α_1, α_2 时，将其他变量看做常数：

$$\begin{aligned}
\arg\min_{\alpha} L(\alpha) &= \frac{1}{2}(\alpha \odot y)^T \cdot K(X, X) \cdot (\alpha \odot y) - \mathbf{1}^m \cdot \alpha \\
&= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{m}\alpha_i
\end{aligned}$$

则上式子去除不包含 α_1, α_2 的项后化简为:

$$
\begin{aligned}
\arg\min_{\alpha_1,\alpha_2} W(\alpha_1,\alpha_2) &= \frac{1}{2}K_{11}a_1^2 + y_1y_2K_{12}a_1a_2 + \frac{1}{2}K_{22}a_2^2 \\
&\quad + y_1a_1\sum_{i=3}^{m} y_ia_iK_{i1} \\
&\quad + y_2a_2\sum_{i=3}^{m} y_ia_iK_{i2} \\
&\quad - (a_1+a_2) \\
&= \frac{1}{2}K_{11}a_1^2 + y_1y_2K_{12}a_1a_2 + \frac{1}{2}K_{22}a_2^2 \\
&\quad + y_1a_1(\sum_{i=1}^{m} y_ia_i^{old}K_{i1} - y_1a_1^{old}K_{11} - y_2a_2^{old}K_{12}) \\
&\quad + y_2a_2(\sum_{i=1}^{m} y_ia_i^{old}K_{i2} - y_1a_1^{old}K_{12} - y_2a_2^{old}K_{22}) \\
&\quad - (a_1+a_2) \\
&= \frac{1}{2}K_{11}a_1^2 + y_1y_2K_{12}a_1a_2 + \frac{1}{2}K_{22}a_2^2 \\
&\quad + y_1a_1((y\odot a^{old})^T\cdot K(X,x_1) - y_1a_1^{old}K_{11} - y_2a_2^{old}K_{12}) \\
&\quad + y_2a_2((y\odot a^{old})^T\cdot K(X,x_2) - y_1a_1^{old}K_{12} - y_2a_2^{old}K_{22}) \\
&\quad - (a_1+a_2)
\end{aligned}
$$

利用约束 s.t. $y^T\cdot\alpha = 0$ 得到:

$$
\begin{aligned}
a_1y_1 + a_2y_2 = a_1^{old}y_1 + a_2^{old}y_2 = \zeta \\
a_1 = y_1(\zeta - a_2y_2) = a_1^{old} + a_2^{old}y_1y_2 - a_2y_1y_2
\end{aligned}
$$

将 a_1 代入:

$$
\begin{aligned}
\arg\min_{\alpha_2} W(\alpha_2) &= \frac{1}{2}K_{11}(y_1(\zeta - a_2y_2))^2 + y_1y_2K_{12}y_1(\zeta - a_2y_2)a_2 + \frac{1}{2}K_{22}a_2^2 \\
&\quad + y_1y_1(\zeta - a_2y_2)((y\odot a^{old})^T\cdot K(X,x_1) - y_1a_1^{old}K_{11} - y_2a_2^{old}K_{12}) \\
&\quad + y_2a_2((y\odot a^{old})^T\cdot K(X,x_2) - y_1a_1^{old}K_{12} - y_2a_2^{old}K_{22}) \\
&\quad - (y_1(\zeta - a_2y_2) + a_2)
\end{aligned}
$$

针对 a_2 求导，并令导数为 0:

$$
\begin{aligned}
\frac{\partial W}{\partial a_2} &= K_{11}y_2(a_2y_2-\zeta)+K_{12}(y_2\zeta-2a_2)+K_{22}a_2 \\
&\quad - y_2((y\odot a^{old})^T\cdot K(X,x_1)-y_1a_1^{old}K_{11}-y_2a_2^{old}K_{12}) \\
&\quad + y_2((y\odot a^{old})^T\cdot K(X,x_2)-y_1a_1^{old}K_{12}-y_2a_2^{old}K_{22}) \\
&\quad + y_1y_2-1 \\
&= a_2(K_{11}-2K_{12}+K_{22}) \\
&\quad - K_{11}y_2(y_1a_1^{old}+y_2a_2^{old})+K_{12}y_2(y_1a_1^{old}+y_2a_2^{old}) \\
&\quad + y_2((y\odot a^{old})^T\cdot K(X,x_2)-(y\odot a^{old})^T\cdot K(X,x_1)) \\
&\quad + y_1y_2a_1^{old}K_{11}+a_2^{old}K_{12}-y_1y_2a_1^{old}K_{12}-a_2^{old}K_{22} \\
&\quad + y_1y_2-1 \\
&= a_2(K_{11}-2K_{12}+K_{22}) \\
&\quad + y_2((y\odot a^{old})^T\cdot K(X,x_2)-(y\odot a^{old})^T\cdot K(X,x_1)) \\
&\quad - a_2^{old}(K_{11}-2K_{12}+K_{22}) \\
&\quad + y_1y_2-1 \\
&= a_2(K_{11}-2K_{12}+K_{22}) \\
&\quad + y_2((y\odot a^{old})^T\cdot K(X,x_2)-y_2-((y\odot a^{old})^T\cdot K(X,x_1)-y_1)) \\
&\quad - a_2^{old}(K_{11}-2K_{12}+K_{22}) \\
&= 0
\end{aligned}
$$

则可以推得:

$$
a_2^{new,unclip}=a_2^{old}+\frac{y_2(((\alpha\odot y)^T\cdot K(X,x_1)-y_1)-((\alpha\odot y)^T\cdot K(X,x_2)-y_2))}{(K_{11}-2K_{12}+K_{22})} \tag{12}
$$

考虑到 $w=X^T\cdot(a\odot y)$ 令 $g(x)=w^T\cdot x+b=(a\odot y)^T\cdot X\cdot x+b$，将将内积转换成核函数形式，并且定义函数 E_i 为函数 $g(x)$ 与 y_i 的误差值:

$$
\begin{aligned}
g(x_i) &= (a\odot y)^T\cdot K(X,x_i)+b \\
E_i &= g(x_i)-y_i=(a\odot y)^T\cdot K(X,x_i)+b-y_i \\
\eta &= K_{11}-2K_{12}+K_{22}
\end{aligned}
$$

则 (12) 式得到未经剪辑的 a_2 解为:

$$
a_2^{new,unclip}=a_2^{old}+\frac{y_2(E_1-E_2)}{\eta} \tag{13}
$$

因为有约束 $a_1y_1+a_2y_2=\zeta=a_1^{old}y_1+a_2^{old}y_2$ 并且 $0\le a_i\le C$，下面讨论 a_2 的上限下限 $L\le a_2\le H$, 当 $y_1\ne y_2$ 时:

$$
\begin{aligned}
a_2-a_1 &= a_2^{old}-a_1^{old} \\
L &= max(0,a_2^{old}-a_1^{old}) \\
H &= min(C,C+a_2^{old}-a_1^{old})
\end{aligned}
$$

当 $y_1 = y_2$ 时:

$$
\begin{aligned}
a_2 + a_1 &= a_2^{old} + a_1^{old} \\
L &= max(0, a_2^{old} + a_1^{old} - C) \\
H &= min(C, a_2^{old} + a_1^{old})
\end{aligned}
$$

则 a_2 经剪辑后解为:

$$
a_2^{new} = \begin{cases} H, & a_2^{new,unclip} > H \\ a_2^{new,unclip}, & L \le a_2^{new,unclip} \le H \\ L, & a_2^{new,unclip} < L \end{cases}
$$

对于偏置 b, 由于 KTT 条件约束有:

$$
\alpha \odot (y \odot (X \cdot w + b) - 1) = \mathbf{0}^m
$$

对于任一 $a_i > 0$ 有 $(a^{old} \odot y))^T \cdot K(X, x_i) + b = y_i$ 整理后得到, 当 $0 \le a_1 \le C$ 时:

$$
\begin{aligned}
b_1^{new} &= y_1 - (a^{new} \odot y))^T \cdot K(X, x_1) \\
&= y_1 - (a^{old} \odot y))^T \cdot K(X, x_1) + a_1^{old} y_1 K_{11} + a_2^{old} y_2 K_{12} - a_1^{new} y_1 K_{11} - a_2^{new} y_2 K_{12}
\end{aligned}
$$

当 $0 \le a_2 \le C$ 时:

$$
\begin{aligned}
b_2^{new} &= y_2 - (a^{new} \odot y))^T \cdot K(X, x_2) \\
&= y_2 - (a^{old} \odot y))^T \cdot K(X, x_2) + a_1^{old} y_1 K_{12} + a_2^{old} y_2 K_{22} - a_1^{new} y_1 K_{12} - a_2^{new} y_2 K_{22}
\end{aligned}
$$

若 a_1 、a_2 都满足条件，那么 $b_1^{new} = a_2^{new}$。
当所有 a_i 满足 KTT 条件时，模型得解。

2 总结

2.1 线性可分支持向量机

当训练数据线性可分时，支持向量机可以得到一个最大间隔的分离超平面，若近似可分，则可以添加软间隔的方式，若线性不可分，则可以利用核技巧确保可以找到最大间隔分离超平面。由于得到的分离间隔最大，支持向量机有较好的鲁棒性。

2.2 模型训练

通过拉格朗日对偶变换，通过 KTT 条件，可以将原始问题转换为拉格朗日对偶问题，从而自然的引入核函数。SMO 是常用的模型训练算法，其本质上是凸二次规划问题，当所有变量都满足 KTT 条件，可以得到最佳解。SMO 的计算过程直接利用解析解，求解速度较快。

参考文献

[1] Ian Goodfellow / Yoshua Bengio / Aaron Courville . *Deep Learning*[M]. 北京: 清华大学出版社,2017-08-01.

[2] 张贤达. 矩阵分析与应用 [M]. 北京: 人民邮电出版社,2013-11-01.

[3] 李航. 统计学习方法 [M]. 北京: 清华大学出版社,2012-03.

[4] David C. Lay / Steven R. Lay / Judi J. McDonald . 线性代数及其应用 [M]. 北京: 机械工业出版社,2018-07.

[5] 史蒂文 · J. 米勒. 普林斯顿概率论读本 [M]. 北京: 人民邮电出版社,2020-08.

A 支持向量机的 python 源码

支持向量机的 python 源码实现，只使用 numpy 库：

```
# -*- coding: utf8 -*-
import math
import time
import numpy as np
import sklearn
from sklearn import datasets
from sklearn.datasets import load_breast_cancer
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
dataset = sklearn.datasets.load_breast_cancer()
dataset.target[dataset.target == 0] = -1 #0/1 标记替换成1/-1标记

class GSKernal:
    def __init__(self):
        self.sigma = 10
    def cal(self, x, z):
        result = None
        if len(x.shape) == 2:
            result = np.linalg.norm(x-z, axis=1) ** 2
        else:
            result = np.linalg.norm(x-z) ** 2
        return np.exp(-1 * result / (2 * self.sigma**2))

class SVM:
    def __init__(self, kn = None, C = 1, toler = 0.001):
        self.w = None #w
        self.b = None #b
        if not kn:
            kn = GSKernal()
        self.kn= kn
        self.C = C
        self.toler = toler
        self.alpha = None
        self.X = None
        self.Y = None
        self.kCache = {}
        return
    def calK(self, i, j):
        key = ''
        if i <= j:
            key = '%d_%d'%(i, j)
        else:
            key = '%d_%d'%(j, i)
        ret = self.kCache.get(key)
```

```
        if ret:
            return ret
        x1 = self.X[i]
        x2 = self.X[j]
        ret = self.kn.cal(x1, x2)
        self.kCache[key] = ret
        return ret
    def calK2(self, j):
        key = 'All_%d'%(j)
        ret = self.kCache.get(key)
        if ret:
            return ret['data']
        ret = np.zeros(self.X.shape[0])
        x2 = self.X[j]
        for i in range(self.X.shape[0]):
            x1 = self.X[i]
            ret[i] = self.kn.cal(x1, x2)
        self.kCache[key] = {'data':ret}
        return ret

    def isZero(self, v):
        return math.fabs(v) < self.toler
    def calGxi(self, i):
        xi = self.X[i]
        ay = np.multiply(self.alpha, self.Y)
        gxi = np.dot(ay, self.calK2(i)) + self.b
        return gxi
    def calcEi(self, i):
        gxi = self.calGxi(i)
        return gxi - self.Y[i]
    def isSatisfyKKT(self, i):
        gxi = self.calGxi(i)
        yi = self.Y[i]
        if self.isZero(self.alpha[i]) and (yi * gxi >= 1):
            return True
        elif self.isZero(self.alpha[i] - self.C) and (yi * gxi <= 1):
            return True
        elif (self.alpha[i] > -self.toler) and (self.alpha[i] < (self.C +
            self.toler)) \
                and self.isZero(yi * gxi - 1):
            return True

        return False
    def getAlphaJ(self, E1, i):
        E2 = 0
        maxE1_E2 = -1
        maxIndex = -1
```

```
        for j in range(self.X.shape[0]):
            if j == i:
                continue
            E2_tmp = self.calcEi(j)
            if E2_tmp == 0:
                continue
            if math.fabs(E1 - E2_tmp) > maxE1_E2:
                maxE1_E2 = math.fabs(E1 - E2_tmp)
                E2 = E2_tmp
                maxIndex = j
        if maxIndex == -1:
            maxIndex = i
            while maxIndex == i:
                maxIndex = int(random.uniform(0, self.X.shape[0]))
            E2 = self.calcEi(maxIndex)

        return E2, maxIndex

    def fit(self, X, Y, iterMax = 10):
        self.kCache.clear()
        scalerX = sklearn.preprocessing.StandardScaler().fit(X)#
            StandardScaler
        #Y = Y.reshape(-1, 1)
        X = scalerX.transform(X)

        w = np.zeros(X.shape[1])
        self.b = 0
        a = np.zeros(X.shape[0])
        self.alpha = a
        self.X = X
        self.Y = Y

        iterStep = 0; parameterChanged = 1

        calTims = 0
        while (iterStep < iterMax) and (parameterChanged > 0):
            iterStep += 1
            parameterChanged = 0

            for i in range(self.X.shape[0]):
                if self.isSatisfyKKT(i):
                    continue
                E1 = self.calcEi(i)
                E2, j = self.getAlphaJ(E1, i)

                y1 = self.Y[i]
                y2 = self.Y[j]
                x1 = self.X[i]
```

```
x2 = self.X[j]
a1_old = a[i]
a2_old = a[j]

if y1 != y2:
    L = max(0, a2_old - a1_old)
    H = min(self.C, self.C + a2_old - a1_old)
else:
    L = max(0, a2_old + a1_old - self.C)
    H = min(self.C, a2_old + a1_old)
if L == H:
    continue

k11 = self.calK(i, i)
k12 = self.calK(i, j)
k21 = k12
k22 = self.calK(j, j)

ay = np.multiply(a, self.Y)
ayk1 = np.dot(ay, self.calK2(i))
ayk2 = np.dot(ay, self.calK2(j))
a2_new = a2_old + (y2 * (ayk1 - y1 - (ayk2 - y2))) / (k11 -
    2*k12 + k22)
a1_new = a1_old + y1 * y2 * (a2_old - a2_new)

b1New = -1 * E1 - y1 * k11 * (a1_new - a1_old) \
            - y2 * k21 * (a2_new - a2_old) + self.b
b2New = -1 * E2 - y1 * k12 * (a1_new - a1_old) \
        - y2 * k22 * (a2_new - a2_old) + self.b
bNew = 0
if (a1_new > 0) and (a1_new < self.C):
    bNew = b1New
elif (a2_new > 0) and (a2_new < self.C):
    bNew = b2New
else:
    bNew = (b1New + b2New) / 2

self.alpha[i] = a1_new
self.alpha[j] = a2_new
self.b = bNew
if math.fabs(a2_new - a2_old) >= 0.00001:
    parameterChanged += 1
calTims += 1

if calTims % 50 == 0:
    print('train iter:%d SMO times:%d'%(iterStep, calTims))
#time.sleep(1)
#return
```

```
        Yo = self.predict(self.X)
        #print(Yo)
        score = r2_score(Yo, self.Y)
        print('score', score)
        return
    def predict(self, X):
        if len(X.shape) == 1:
            return self.predictOne(X)
        ret = np.zeros(X.shape[0])
        for i in range(X.shape[0]):
            ret[i] = self.predictOne(X[i])
        return ret
    def predictOne(self, x):
        ret = np.zeros(self.X.shape[0])
        x2 = x
        for i in range(self.X.shape[0]):
            x1 = self.X[i]
            ret[i] = self.kn.cal(x1, x2)
        ay = np.multiply(self.alpha, self.Y)
        gxi = np.dot(ay, ret) + self.b
        if gxi > 0:
            return 1
        return -1

if __name__ == '__main__':
    svm = SVM()
    print('data', dataset.data.shape)
    svm.fit(dataset.data, dataset.target)
```