支持向量机

硕 1810 M201873350

王晶

November 30, 2018

1 支持向量机介绍

支持向量机 (Support Vector Machine，SVM) 被许多人认为是现有最好的监督学习算法之一，本文将从二分类支持向量机及扩展，SVM 处理多分类问题，二分类 SVM 实现等方面介绍支持向量机。

2 二分类支持向量机及扩展

2.1 符号和约定

我们约定单个数据的特征 (**feature**) 为一 n 维向量，单个数据的类别标签 (**label**) 为一个值，假定训练集 S 共有 m 个数据。数据集特征的矩阵表示为 X，数据集的标签向量表示为 Y，特征矩阵中第 i 个特征表示为 $x^{(i)}$，标签向量中第 i 个标签类别表示为 $y^{(i)}$，下标均从 1 开始。

2.2 函数间隔与几何间隔

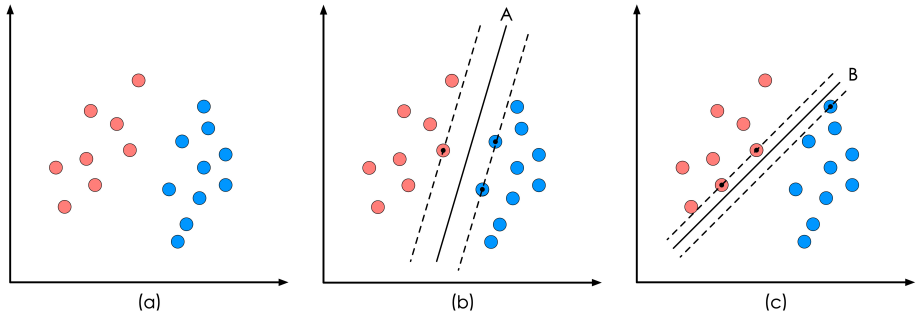

Figure 1: 经典的二分类

SVM 主要用于解决数据的最优分类问题，在经典的二分类中，红色和蓝色两类数据点可以被多条直线所分隔开，那么如何表达这些分隔直线的分隔效果呢？或者说选哪条直线分类的效果最优呢？直觉上可以感知到如果两类点离直线越远，分类的效果越明显，我们将分类的效果使用间隔 (**margin**) 表示，同时我们将 n 维空间中 $n-1$ 维度的线性子空间 ($w^Tx+b=0, x\in\mathbb{R}^n$) 称为分离超平面[1](**separating hyperplane**)，在图将两类点分开的直线是一个超平面，在三维空间中，二维平面是一个超平面。这里，数据被分为正类和父类，分离超平面是一个线性的二分类器，写为:

$$h_{w,b}(x)=g(w^Tx+b)$$

其中，若 $z\geq 0$ 则 $g(z)=1$ 否则 $g(z)=-1$，前者称为正类，后者称为负类。

[1]实际上 n 大于 3 才被称为“超”平面

对于样本点 $(x^{(i)}, y^{(i)})$ 到分离超平面 (w, b) 的函数间隔 (**functional margin**) 定义为

$$\gamma^{(i)} = y^{(i)}(w^T x^{(i)} + b)$$

通过样本点的函数间隔定义我们可以得到，如果 $y^{(i)} = 1$ 而 $w^T x^{(i)} + b < 0$ 从而 $\gamma^{(i)} < 0$，$y^{(i)} = -1$ 也有类似结果，这说明当超平面误分类时，样本点的函数间隔为负，所以当函数间隔 $\gamma^{(i)} > 0$ 时分类器分类正确。同时定义对于训练集 S 到分离超平面的函数间隔为

$$\gamma = \min_{i=1,\cdots,m} \gamma^{(i)}$$

也就是训练集的对分离超平面 (w, b) 函数间隔为训练集中所有样本点中最小的函数间隔同时也可以观察到，将 w, b 同时放缩相同的倍数时，分离超平面的位置并没有改变，但函数间隔却放缩相同的倍数。

以上结果表明，函数间隔仅仅能够表达分类的正确与否，却并不能分类的效果。因为对 w, b 同时乘以一个非零放缩因子能够改变函数间隔，为了函数间隔不被放缩因子所改变，我们引入几何间隔 (**geometric margin**) 的概念。与函数间隔类似

$$d^{(i)} = \frac{y^{(i)}(w^T x^{(i)} + b)}{||w||}$$

容易看出几何间隔是函数间隔在 $||w|| = 1$ 时的单位化。对于样本点，在二维平面中，几何间隔是点到直线的距离；在三维空间中，几何间隔是点到平面的距离。。

$$d^{(i)} = \frac{\gamma^{(i)}}{||w||}$$

$$d = \min_{i=1,\cdots,m} d^{(i)}$$

2.3 最优分类器

前面提到 SVM 不是普通的分类器，它是一个最优分类器，直觉上，训练集几何间隔越大，分类越有把握。针对拥有 m 个样本的训练集 S，最优化问题为

$$\begin{aligned} &\max_{\gamma,w,b} \quad d \\ &s.t. \quad \frac{y^{(i)}(w^T x^{(i)} + b)}{||w||} \geq d, i = 1, \cdots, m \end{aligned}$$

可化简为

$$\begin{aligned} &\max_{\gamma,w,b} \quad \frac{1}{||w||} \\ &s.t. \quad y^{(i)}(w^T x^{(i)} + b) \geq 1, i = 1, \cdots, m \end{aligned}$$

由于最大化 $\frac{1}{||w||}$ 与最小化 $\frac{1}{2}||w||^2$ 取得最值时的 w 值相同，

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

上述凸二次优化且仅有线性约束问题的解就是最优分类器。

2.4 拉格朗日函数及分析

高数中，对于某个连续可微的函数 $f(x), x \in \mathbb{R}^n$ 求最值，通常直接 $\nabla_x f(x) = 0$ 令偏导等于零，然后从中筛选即可，对于有等式约束 $g(x) = 0$，可以应用**拉格朗日乘数法**，对于还含有不等式约束的最优化问题，这里定义广义拉格朗日 (**generalized lagrangian**) 问题的基本形式

有原始 (**primal**) 的最优问题

$$\begin{aligned} \min_{w} \quad & f(w) \\ s.t. \quad & g_i(w) \leq 0, i = 1, \cdots, k \\ & h_i(w) = 0, i = 1, \cdots, l \end{aligned} \tag{2}$$

定义广义拉格朗日函数为

$$\mathcal{L}(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

对于如上定义的简化的数学问题 $\mathcal{L} = f + \beta h$ ，如 Figure 2中约束的可行域 因为梯度代表函数增长最快的方向，所以对于在边界上的 x，$g(x)$ 的梯度指

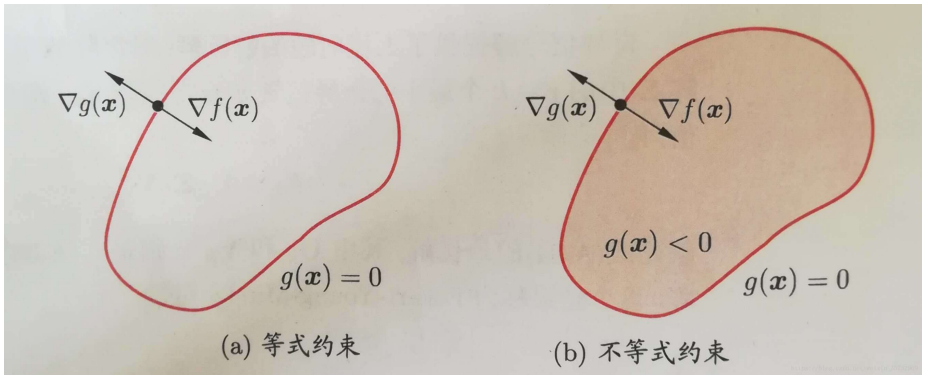

Figure 2: $g(x)$ 可行域

向可行域外，由于我们求的是 $f(x)$ 的最小值，所以 $f(x)$ 的梯度指向可行域内 (若指向可行域外，则可行域内可取到最小值)，$f(x)$ 与 $g(x)$ 梯度相反。现在讨论最优解的位置:

在可行域边界: 由 $\nabla \mathcal{L} = 0$ 可以推出 $\beta > 0$

在可行域内: 这时相当于没有约束条件，可以推出 $\beta = 0$

在原始问题 $\mathcal{L}(w, \alpha, \beta)$ 中可以类似得到 $\beta_i >= 0$ 且 $\beta_i h_i = 0$，其中

2.5 Karush-Kuhn-Tucker(KKT) 条件

经过前一节的分析，我们可以推出 $\beta_i h_i = 0$ 这一条件。恰好这一条件被称为 KKT**对偶互补条件。**

什么是 KKT 条件? 在优化理论中，KKT 条件是非线性问题 (**nonlinear programming**) 最优解的必要条件，可以看出 KKT 条件是拉格朗日乘数法在不等式约束条件下的推广，满足 KKT 条件的不一定是最优解，但最优解一定满足 KKT 条件。

如在2中，约束称为原始可行性 (**primal feasibility**)，$\beta_i \geq 0$ 称为对偶可行性 (**dual feasibility**)，$\beta_i h_i = 0$ 称为松弛互补条件也就是 KKT 对偶互补条件。

KKT 条件与最优化问题又有什么关系呢?

在2中，我们定义

$$\theta_P(w) = \max_{\alpha,\beta:\beta_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

在可行域内或边界: 由 $\beta_i >= 0, g(x) \leq 0$ 且 $\beta_i h_i = 0$ 推出 $\theta_P(w) = f(w)$
在可行域外: $g(x) \geq 0$，可以推出 $\theta_P(w) = \infty$
原始问题可以转化

$$p^* = \min_w \theta_P(w) = \min_w \max_{\alpha,\beta:\beta_i \geq 0} \mathcal{L}(w, \alpha, \beta)$$

其对偶 (**dual**) 优化问题

$$\theta_D(w) = \min_w \mathcal{L}(w, \alpha, \beta)$$

$$d^* = \max_{\alpha,\beta:\beta_i \geq 0} \min_w \mathcal{L}(w, \alpha, \beta)$$

定理一: $d^* \leq p^*$
证明:

$$\begin{aligned} \min_w \mathcal{L}(w, \alpha, \beta) \leq \mathcal{L} \leq \max_{\alpha,\beta:\beta_i \geq 0} \mathcal{L}(w, \alpha, \beta) \quad \forall \alpha, \beta, w \\ \theta_D(w) \leq \theta_P(w) \quad \forall \alpha, \beta, w \\ d^* \leq p^* \end{aligned} \tag{3}$$

$d^* \leq p^*$ 称为弱对偶性，如果存在一组最优解 (w^*, α^*, β^*) 使 $d^* = p^*$，则强对偶性成立

定理二: 对于原始函数和对偶函数，目标函数 $f(x)$ 和不等式约束条件 $g_i(x), \forall i$ 均为**凸函数**，等式约束 $h_i(x)$ 为仿射函数并且至少存在一个 x 使所有不等式约束严格成立 ($\exists x, \forall i, g_i(x) < 0$ 成立)，则存在 (w^*, α^*, β^*) 使

$d^* = p^* = \mathcal{L}(w^*, \alpha^*, \beta^*)$。并且其充分必要条件为

$$\begin{aligned}
&\nabla_w \mathcal{L}(w^*, \alpha^*, \beta^*) = 0 \\
&\nabla_\alpha \mathcal{L}(w^*, \alpha^*, \beta^*) = 0 \\
&\nabla_\beta \mathcal{L}(w^*, \alpha^*, \beta^*) = 0 \\
&\forall i, \quad \beta_i g_i(w^*) \geq 0 \\
&\forall i, \quad g_i(w^*) \leq 0 \\
&\forall i, \quad \beta_i \geq 0 \\
&\forall j, \quad h_j(w^*) = 0
\end{aligned} \tag{4}$$

其中后**四**个公式为 KKT 条件。

至此训练集 S 上的最优化问题可以写成

$$\begin{aligned}
&\min_{\gamma, w, b} \quad \frac{1}{2}||w||^2 \\
&s.t. \quad 1 - y^{(i)}(w^T x^{(i)} + b) \leq 0, i = 1, \cdots, m
\end{aligned} \tag{5}$$

约束为 LCQ(**linearity constraint qualification**) 满足 KKT 条件，对应的拉格朗日函数为

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}||w||^2 + \sum_{i=1}^{m} \alpha_i \left[1 - y^{(i)}(w^T x^{(i)} + b)\right] \tag{6}$$

使用 KKT 条件

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0$$

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

$$\nabla_b \mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

这里对 b 求偏导是将 b 作为一个变量，进行多元函数求偏导，代入6

$$\mathcal{L}(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x_i, x_j \rangle \tag{7}$$

转化对偶问题最优化为

$$\begin{aligned}
&\max_{\alpha} \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x_i, x_j \rangle \\
&s.t. \quad \alpha_i \geq 0, i = 1, \cdots, m \\
&\qquad \sum_{i=1}^{m} \alpha_i y^{(i)} = 0
\end{aligned} \tag{8}$$

当找到最优解 w^* 时，由6来确定，即

$$b^* = -\frac{\max_{y^{(i)}=-1} w^{*T}x^{(i)} + \min_{y^{(i)}=1} w^{*T}x^{(i)}}{2} \tag{9}$$

对于形如 $y^{(i)}(w^{*T}x^{(i)}) < 1$ 的约束，由2.4可以推出对应的 $\alpha_i = 0$，实际上是 $y^{(i)}(w^{*T}x^{(i)}) = 1$ 的向量构成了分离超平面，这些向量被称为支持向量，这也是支持向量机名称的由来。

2.6 核函数

在现实生活中，可能与分类有关的输入 x 不一定是线性的，也可能与 x^2, x^3 有关，实际上可能存在特征映射 (**feature mapping**)，如

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

因为8中完全表述为 $\langle x_i, x_j \rangle$ 内积的形式，因此我们使用核 (**kernel**) 来代替内积

$$K(x,z) = \langle \phi(x), \phi(z) \rangle = \phi(x)^T \phi(z)$$

核的问题之一是计算相应的特征映射 ϕ 可能极其费力，给定 ϕ 通过计算 $\phi(x), \phi(z)$ 然后做内积的计算方式很简单，如计算 $K(x,z) = (x^Tz)^2$ 中计算 $\phi(x)$ 就需要 $O(n^2)$ 时间，更一般的 $K(x,z) = (x^Tz + c)^d$ 需要 $O(n^d)$ 特征映射，计算但是 $K(x,z)$ 仍只需要 $O(n)$ 时间。

从另一个角度看，给定一个矩阵 K，如何判断它是一个有效的核函数？将核函数在训练集运行我们可以得到核矩阵，即 $K_{ij} = K(x^{(i)}, y^{(j)}) = K_{ji}$ 由此核矩阵必定对称。**Mercer**定理索性给出了有限维核矩阵有效的充要条件: 对称且半正定!

2.7 线性不可分及正则化

对于线性不可分数据集，我们在原来1基础上 l_1 正则项。

$$\begin{aligned} \min_{\gamma,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ s.t. \quad & y^{(i)}(w^Tx^{(i)} + b) \geq 1 - \xi_i, i = 1, \cdots, m \\ & \xi_i \geq 0, i = 1, \cdots, m \end{aligned} \tag{10}$$

其中 ξ 的存在允许了样本的函数间隔小于 1，对应目标函数增加的代价 $C\xi$，C 平衡了 $||w||$ 更大和确保大多数样本函数间隔至少为 1 两者之间的关系。对应的拉格朗日函数为

$$\mathcal{L}(w,b,\xi,\alpha,r) = \frac{1}{2}w^Tw + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i\left[y^{(i)}(x^{(i)T}w + b) - 1 + \xi_i\right] - \sum_{i=1}^{m}r_i\xi_i$$

其中 α 和 r 为拉格朗日乘子，由此我们可以推出与之对应的对偶问题为

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x_i, x_j \rangle \\ s.t. \quad & 0 \le \alpha_i \le C, i = 1, \cdots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{11}$$

同样地，由 KKT 对偶互补条件推出

$$\begin{aligned} \alpha_i = 0 &\Rightarrow y^{(i)}(w^T x^{(i)} + b) \ge 1 \\ \alpha_i = C &\Rightarrow y^{(i)}(w^T x^{(i)} + b) \le 1 \\ 0 < \alpha_i < C &\Rightarrow y^{(i)}(w^T x^{(i)} + b) = 1 \end{aligned} \tag{12}$$

这也对测试 SMO 算法是否收敛非常有用。

2.8 SMO 算法

理论上在6中就可以使用传统的二次规划算法求解，但当训练集非常庞大时，求解时间过长，因此我们可以使用迭代的 SMO**(sequential minimal optimization)**算法来进行计算。SMO 算法与坐标上升 (**coordinate ascent**) 算法类似。

考虑下面无约束的最优化问题

$$\max_{\alpha} W(\alpha_1, \alpha_2, \cdots, \alpha_m)$$

坐标上升算法是先固定一个 α_i，计算满足目标的新 α_i，直至算法收敛。

Algorithm 1 坐标上升算法

repeat
  select α_i and calculate new α_i;
  reoptimize $W(\alpha)$ with new α_i and holding others fixed;
until convergence

由此来看，在11中可以选取一个 α_i 来进行坐标上升是可行的吗？很遗憾，不行。因为约束的存在，使得更新 α_i 必然会违反约束条件，所以我们改进为同时选取两个 α_i, α_j 来尝试类似坐标上升的算法，从而不断向全局最大值逼近直至收敛。

方便起见，我们使用 α_1, α_2 进行演示算法

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta \tag{13}$$

$$\max_{\alpha} W(\alpha_1, \alpha_2, \cdots, \alpha_m) = \max_{\alpha} W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \cdots, \alpha_m) \tag{14}$$

Algorithm 2 SMO 算法

repeat
  select α_i, α_j and calculate new α_i, α_j;
  reoptimize $W(\alpha)$ with new α_i, α_j and holding others fixed;
until convergence

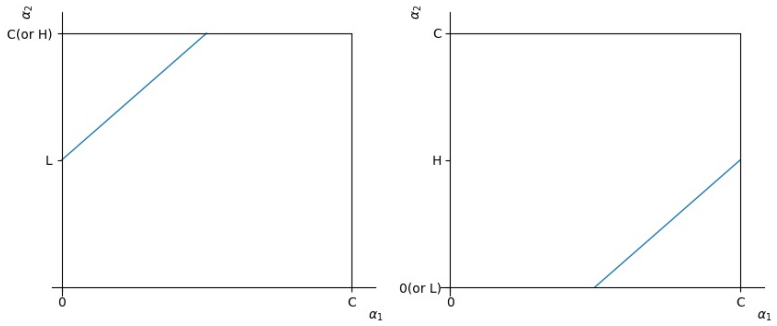

Figure 3: $y^{(1)} \neq y^{(2)}$

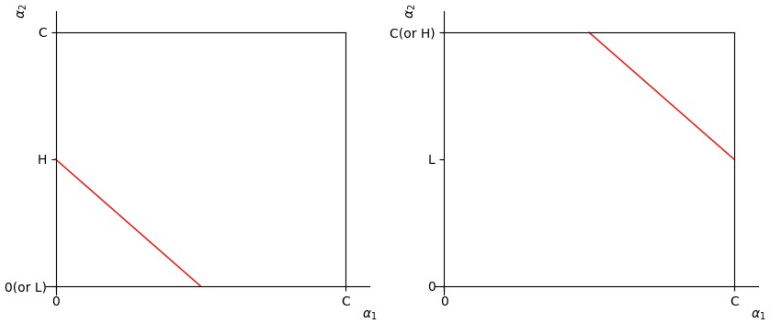

Figure 4: $y^{(1)} = y^{(2)}$

$$\alpha_1^{old} y^{(1)} + \alpha_2^{old} y^{(2)} = \alpha_1^{new} y^{(1)} + \alpha_2^{new} y^{(2)} \tag{15}$$

从11来看，求 $W(\alpha)$ 最大值其实是求二次函数在给定区间 $[L, H]$ 的最大值其中 L, H 如图3与4

计算 L, H

$$\begin{aligned} &y^{(1)} \neq y^{(2)} \Rightarrow \\ &\qquad L = \max(0, \alpha_2 - \alpha_1) \\ &\qquad H = \min(C, C + \alpha_2 - \alpha_1) \\ &y^{(1)} = y^{(2)} \Rightarrow \\ &\qquad L = \max(0, \alpha_2 + \alpha_1 - C) \\ &\qquad H = \min(C, \alpha_2 + \alpha_1) \end{aligned} \tag{16}$$

计算 α_2^{new}

$$\alpha_2^{new} = \begin{cases} H & \alpha_2^{new,uncliped} > H \\ \alpha_2^{new,uncliped} & L \leq \alpha_2^{new,uncliped} \leq H \\ L & \alpha_2^{new,uncliped} < L \end{cases} \tag{17}$$

同时根据15计算 $\alpha_1^{new} = \alpha_1^{old} + y^{(1)} y^{(2)} (\alpha_2^{old} - \alpha_2^{new})$
下面介绍具体计算 α_i^{new}，由

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j K_{ij} \\ s.t. \quad & 0 \leq \alpha_i \leq C, i = 1, \cdots, m \\ & \sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \end{aligned} \tag{18}$$

更新 α_i, α_j

$$\begin{aligned} f(x) = w^T + b &= \sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) + b \\ E_k &= f(x^{(k)}) - y^{(k)} \\ \eta &= 2K_{ij} - K_{ii} - K_{jj} \\ \alpha_j &:= \alpha_j - \frac{y^{(i)}(E_i - E_j)}{\eta} \end{aligned} \tag{19}$$

其中当 $\eta = 0$ 时，不用更新。然后更新 b 当 $0 < \alpha_i < C$ 时，根据12此时对应的样本点为支持向量，更新结果为

$$b_1 = b - E_i - y^{(i)}(\alpha_i - \alpha_i^{old}) K_{ii} - y^{(j)}(\alpha_j - \alpha_j^{old}) K_{ij}$$

同理对于 $0 < \alpha_j < C$ 有

$$b_2 = b - E_j - y^{(i)}(\alpha_i - \alpha_i^{old}) K_{ij} - y^{(j)}(\alpha_j - \alpha_j^{old}) K_{jj}$$

当上述条件同时成立时 $b1 = b2$ 显然成立当 $\alpha_i = C$ 或者 $\alpha_i = 0$(α_j 类似)，b_1, b_2 之间的值均符合 KKT 条件，我们取 $b = (b_1 + b_2)/2$

$$b = \begin{cases} b_1 & 0 < \alpha_i < C \\ b_2 & 0 < \alpha_j < C \\ (b_1 + b_2)/2 & otherwise \end{cases} \tag{20}$$

3 SVM 处理多分类问题

SVM 是一个二分类器，如何处理多分类问题呢？常用有三种方法

3.1 一对多 (one-versus-rest)

将某个类别的样本归为一类，将其他样本归为一类，总共有 k 类，这样就可以构造 k 个 SVM，根据 k 个 SVM 分类的结果，决定最终的分类结果。

3.2 一对一 (one-versus-one)

因为任意两个类别的样本可以设计出一个 SVM，这样 k 个类别共有 $k(k-1)/2$ 个 SVM，根据所有 SVM 的结果，判断位置样本的类别。

3.3 多分类 SVM

二分类中，样本 $(x^{(i)}, y^{(i)})$ 的损失函数可以看成

$$L_i = C\,\mathsf{max}(0, 1 - y^{(i)} w^T x^{(i)}) + R(W)$$

于是在多分类中我们定义 $f(x^{(i)}, W)_j$ 为 $x^{(i)}$ 分到 j 类的得分，则样本损失函数定义误分类的代价为

$$L_i = \sum_{y^{(i)} \neq j} \mathsf{max}(0, f(x^{(i)}, W)_j - f(x^{(i)}, W)_{y^{(i)}} + \Delta) \tag{21}$$

例如

$$f(x^{(i)}, W) = [-13, -7, 11] \quad and \quad y_{(i)} = 0, \Delta = 10$$

$$L_i = \mathsf{max}(0, -7 - 13 + 10) + \mathsf{max}(0, 11 - 13 + 10)$$

这样我们的到全局的损失函数为

$$L = \underbrace{\frac{1}{N} \sum_i \sum_{j \neq y_i} \mathsf{max}(0, f_j - f_{y_i} + 1)}_{\text{data loss}} + \underbrace{\lambda \sum_k \sum_l W_{k,l}^2}_{\text{regularization loss}} \tag{22}$$

然后再进行最优化求解。

4 二分类 SVM 实现

数据集生成来源于https://github.com/WenDesi/lihang_book_algorithm/blob/master/svm/generate_dataset.py

实验环境:python 3.6，并安装 numpy，sklearn 包，运行时去掉注释

```
# -*- coding: utf-8 -*-
import numpy as np
import random
from generate_dataset import *
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import time


class SVM:

        def __init__(self, tolerance=0.001, max_passes=40, kernel='linear',
        regularization_parameter=0.6):
                self.tolerance = tolerance
                self.C = regularization_parameter
                self.kernel = kernel
                self.max_passes = max_passes

        def _init_parameters(self, features, labels):
                self.X = features
                self.Y = labels
                self.b = 0.0
                self.n = len(features[0])
                self.m = len(features)
                self.alpha = [0.0] * self.m  # m-dimension zero vector
                self.E = np.zeros(self.m)

        def fit(self, features, labels):
                self._init_parameters(features, labels)
                passes = 0
                while passes < self.max_passes:
                        nums_changed_alphas = 0
                        for i in range(self.m):
                        self.E[i] = self.f(self.X[i]) - self.Y[i]
                        #计算E[i]
                        if self.Y[i] * self.E[i] < -self.tolerance
                                and self.alpha[i] < self.C or\
                                self.Y[i] * self.E[i] > self.tolerance
                                and self.alpha[i] > 0:
                                j = i
                                while j == i:
                                j = random.randint(0, self.m - 1)
                                #选取另一个j
                                self.E[j] = self.f(self.X[j]) - self.Y[j]
                                a_i = self.alpha[i]
                                a_j = self.alpha[j]
                                #保存ai,aj
                                if self.Y[i] != self.Y[j]:
                                        L = max(0.0,
                                        self.alpha[j] - self.alpha[i])
```

```
                                        H = min(self.C,
                                        self.C + self.alpha[j] - self.alpha[i])
                                else:
                                        L = max(0.0,
                                        self.alpha[i] + self.alpha[j] - self.C)
                                        H = min(self.C,
                                        self.alpha[i] + self.alpha[j])
                                if L == H:
                                        continue
                                eta = 2 * np.dot(self.X[i], self.X[j]) -
                                np.dot(self.X[i], self.X[i]) -\
                                np.dot(self.X[j], self.X[j])
                                if eta >= 0:
                                        continue
                                self.alpha[j] = self.alpha[j] - self.Y[j] *
                                 (self.E[i] - self.E[j]) / eta
                                if self.alpha[j] > H:
                                        self.alpha[j] = H
                                elif self.alpha[j] < L:
                                        self.alpha[j] = L
                                if abs(self.alpha[j] - a_j) < 1e-5:
                                        continue
                                self.alpha[i] = self.alpha[i] + self.Y[i] *
                                self.Y[j] * (a_j - self.alpha[j])
                                #根据公式更新b
                                b1 = self.b - self.E[i] - self.Y[i] *
                                 (self.alpha[i] - a_i) *
                                 np.dot(self.X[i], self.X[i]) - \
                                self.Y[i] * (self.alpha[j] - a_j) *
                                 np.dot(self.X[i], self.X[j])
                                b2 = self.b - self.E[j] - self.Y[i] *
                                (self.alpha[i] - a_i) * np.dot(self.X[i], self.X[j]) - \
                                self.Y[i] * (self.alpha[j] - a_j) *
                                 np.dot(self.X[j], self.X[j])
                                if 0 < self.alpha[i] < self.C:
                                        self.b = b1
                                elif 0 < self.alpha[j] < self.C:
                                        self.b = b2
                                else:
                                        self.b = (b1 + b2) / 2
                                nums_changed_alphas += 1
                        if nums_changed_alphas == 0:
                                passes += 1
                        else:
                                passes = 0
                # print("passes = ", passes, "\n")
                def _predict(self, feature):
                        return (1 if np.sum([self.alpha[i] * self.Y[i] *
                        np.dot(feature, self.X[i]) for i in range(self.m)]) + self.b >
                        0 else -1)
```

```
101
102         def predict(self, features):
103                 return [self._predict(feature) for feature in features]
104
105         def f(self, x):
106                 return np.sum([self.alpha[i] * self.Y[i] *
107                 np.dot(self.X[i], x) for i in range(self.m)]) + self.b
108
109
110 if __name__ == "__main__":
111         start = time.time()
112         train_features, train_labels, test_features, test_labels =
113         generate_dataset(200, visualization=False)
114         end = time.time()
115         #print("read data use", end-start, "s\n")
116
117         #print("start training")
118         start = time.time()
119         svm = SVM()
120         svm.fit(train_features, train_labels)
121         end = time.time()
122         #print("training use", end - start, "s\n")
123
124         #print("start predicting")
125         start = time.time()
126         test_predict = svm.predict(test_features)
127         end = time.time()
128         #print("predicting use", end - start, "s\n")
129
130         print("alpha", svm.alpha)
131         score = accuracy_score(test_labels, test_predict)
132         #print("the accuracy score is", score)
```

5 参考文献

1. Andrew Ng, CS229 Lecture notes:Support Vector Machines,http://cs229.stanford.edu/notes/cs229-notes3.pdf/
2. Andrew Ng, CS229 materials:The Simplified SMO Algorithm,http://cs229.stanford.edu/materials/smo.pdf
3. Fei-Fei Li, CS231n Lecture notes:Loss Functions and Optimization, http://vision.stanford.edu/teaching/cs231n/slides/2018/cs231n_2018_lecture03.pdf