SVM 学习笔记

李向阳　d1142845997@gmail.com

目录

1 引入

支撑向量机 (Support Vector Machine) 是一种的分类方法, 当然它也可以用于回归. 不过我们这里主要讨论分类, 仍然主要针对二分类.

在 Logistic 回归模型中, 我们知道决策边界是 $\boldsymbol{\theta}^T\boldsymbol{x} = 0$, 一个样本属于正类的概率为

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T\boldsymbol{x})$$

一个样本 $\boldsymbol{x}$ 归为正类当且仅当 $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \geqslant 0.5$, 或者等价地, 当且仅当 $\boldsymbol{\theta}^T\boldsymbol{x} \geqslant 0$, 显然, 给定一个样本 $\boldsymbol{x}$, $\boldsymbol{\theta}^T\boldsymbol{x}$ 的值越大, 那么我们就越有信心将其归为正类, 反之, $\boldsymbol{\theta}^T\boldsymbol{x}$ 的值越小, 那么我们就越有信心将其归为负类, 换句话说, 我们希望所有的样本计算出的 $\boldsymbol{\theta}^T\boldsymbol{x}$ 的值与 0 相差都尽量的大.

SVM 的思想与此类似, 我们希望找到一个超平面, 使得样本点到超平面的距离尽量的大. 下面我们就来介绍 SVM 模型.

为了讨论的方便, 我们这次稍微改换一下记号. 之前在 Logistic 回归中, 我们用类标签 $y = 1$ 表示正类, 用 $y = 0$ 表示负类, 在 SVM 中, 我们改用 $y = 1$ 表示正类, $y = -1$ 表示负类 (只是为了推导的方便). 所谓 SVM 模型, 是用如下公式来刻画:

$$y = \text{sign}(f(\boldsymbol{x})) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$$

即令 $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$, 如果 $f(\boldsymbol{x}) > 0$, 就令 $y = 1$(把样本归为正类), 如果 $f(\boldsymbol{x}) < 0$, 就令 $y = -1$(把样本归为负类).

注意这里与 Logistic 回归中记号的不同, 样本 $\boldsymbol{x}$ 有 n 个特征, 即 $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)^T$, 参数不再用 $\boldsymbol{\theta} = (\theta_0, \theta_1, \cdots, \theta_n)$ 来表示, 而是把常数项 θ_0 单独拿出来用 b 表示, 其它参数用 $\boldsymbol{w} = (w_1, w_2, \cdots, w_n)^T$ 表示, 因此我们不必再引入特征 $x_0 = 1$. 此外, 我们这里采用机器学习大纲中的第 2 套记号, 即用 $\boldsymbol{x}_i$ 表示第 i 个具体的样本, 直接理解为具有 n 个分量的向量, 而不是用 $\boldsymbol{x}^{(i)}$ 表示第 i 个样本, 然后设样本个数为 N.

有了模型, 我们就需要估计出模型参数 $\boldsymbol{w}$ 和 b. 这里我们先不采用统计中的观点 (前面的线性回归也好、逻辑回归都可以用最大似然估计), 那么采用什么标准来估计参数呢? 下面就正式讨论 SVM 模型.

2 基本 SVM 模型

2.1 函数间隔和几何间隔

我们希望样本点到超平面的距离越远越好, 实际上, 结合上面的分析, 我们这里有 2 个标准, 分别是函数间隔 (Functional Margin) 和几何间隔

(Geometric Margin).

Functional Margin 用 $\hat{\gamma}$ 表示, 其定义为

$$\hat{\gamma} = y(\boldsymbol{w}^T\boldsymbol{x} + b) = yf(\boldsymbol{x})$$

注意到 $\boldsymbol{w}^T\boldsymbol{x} + b > 0$ 时, $y = 1$, 而 $\boldsymbol{w}^T\boldsymbol{x} + b < 0$ 时, $y = -1$, 因此前面乘上 y 可以保证 Functional Margin 的非负性. 显然, 我们希望对于所有的样本点, 这个值越大越好, 这个可以成为我们的优化目标.

Geometric Margin 用 $\tilde{\gamma}$ 表示, 其定义就直接是样本点 $\boldsymbol{x}$ 到超平面 $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ 的距离. 当然, 我们也希望这个值越大越好, 这个也可以作为我们的优化目标.

二者之间有什么关系呢?

对于一个点 $\boldsymbol{x}$(实际上是用 n 维向量表示), 设其到超平面的投影点为 $\boldsymbol{x}_0$, 我们知道 $\boldsymbol{w}$ 是超平面的一个法向量, 设点 $\boldsymbol{x}$ 到超平面的距离为 γ, 则有

$$\boldsymbol{x} = \boldsymbol{x}_0 + \gamma\frac{\boldsymbol{w}}{||\boldsymbol{w}||} \quad (\boldsymbol{x} - \boldsymbol{x}_0 = \gamma\frac{\boldsymbol{w}}{||\boldsymbol{w}||}) \tag{1}$$

注意这里的 γ 是带符号的 (可正可负), 在这里, 把它乘上对应的类别 y 刚好就是真正的距离 $\tilde{\gamma}$.

由于 $\boldsymbol{x}_0$ 在超平面上, 满足 $f(\boldsymbol{x}_0) = \boldsymbol{w}^T\boldsymbol{x}_0 + b = 0$, 即有 $\boldsymbol{w}^T\boldsymbol{x}_0 = -b$, 为了利用此条件, 我们将式 (1) 的两边同时乘以 $\boldsymbol{w}^T$, 可得

$$\boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{w}^T\boldsymbol{x}_0 + \gamma \cdot \frac{\boldsymbol{w}^T\boldsymbol{w}}{||\boldsymbol{w}||} = -b + \gamma||\boldsymbol{w}||$$

于是有

$$\gamma = \frac{\boldsymbol{w}^T\boldsymbol{x} + b}{||\boldsymbol{w}||} = \frac{f(\boldsymbol{x})}{||\boldsymbol{w}||}$$

因此可得 Geometrical Margin 为

$$\tilde{\gamma} = y\gamma = \frac{yf(\boldsymbol{x})}{||\boldsymbol{w}||} = \frac{\hat{\gamma}}{||\boldsymbol{w}||}$$

也就是说 Geometrical Margin 和 Functional Margin 相差一个 $||\boldsymbol{w}||$ 的缩放因子.

2.2 SVM 的优化目标

上面已经提到, 对一个样本点进行分类, 它的 Margin 值越大, 分类的 Confidence 越高. 假设我们的样本容量为 N, 我们定义该样本集合的 Margin 为所有这 N 个点的 Margin 值中最小的那个.

于是, 为了使得分类的 Confidence 高, 我们希望所选择的超平面能够最大化这个 Margin 值. 这里我们有两个 Margin 可以选, 不过 Functional

Margin 明显是不太适合用来最大化的一个量, 因为在超平面固定以后, 我们可以等比例地缩放 $\boldsymbol{w}$ 的长度和 b 的值, 这样可以使得 $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$ 的值任意大, 亦即 Functional Margin $\hat{\gamma}$ 可以在超平面固定的情况下取得任意大, 而 Geometrical Margin $\tilde{\gamma}$ 则没有这个问题, 因为它还除以了 $||\boldsymbol{w}||$, 所以缩放 $\boldsymbol{w}$ 和 b 的时候 $\tilde{\gamma}$ 的值是不会改变的, 它只随着超平面的变动而变化, 因此这是一个更适合的 Margin. 因此, 我们的优化目标是 Geometrical Margin $\tilde{\gamma}$, 也即为

$$\max \tilde{\gamma} = \max \frac{\hat{\gamma}}{||\boldsymbol{w}||}$$

当然, 还要满足限制条件, 即样本集合的这个 Margin 值是所有 N 个样本点的 Margin 值中最小的那个, 也就等价于

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) = \hat{\gamma}_i \geqslant \hat{\gamma}, i = 1, 2, \cdots, N$$

观察这个优化模型, 我们发现, 正如前面指出的, 即使在超平面固定的情况下, $\hat{\gamma}$ 的值还是可以随着 $||\boldsymbol{w}||$ 的变化而无限变大. 由于我们的目标是确定超平面, 而对这个最大值具体是多少并不关心 (如同最大似然估计一样), 因此我们必须把这个无关的变量固定下来. 固定的方式有两种: 一是固定 $||\boldsymbol{w}||$; 二是固定 $\hat{\gamma}$. 基于方便推导和优化的目的, 我们选择第二种, 不妨令 $\hat{\gamma} = 1$, 此时 $\tilde{\gamma} = 1/||\boldsymbol{w}||$, 我们的优化模型变为

$$\max \frac{1}{||\boldsymbol{w}||}, \quad s.t. \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geqslant 1, i = 1, 2, \cdots, N$$

其等价形式为

$$\max \frac{1}{2}||\boldsymbol{w}||^2, \quad s.t. \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geqslant 1, i = 1, 2, \cdots, N$$

这是一个凸优化问题, 而且是一个二次规划 (Quadratic Programming) 问题, 通过求解这个优化问题, 我们就可以得到最优参数 $\boldsymbol{w}$ 和 b, 从而也就得到了超平面的方程, 一个分类器也就出现了. 现在我们先暂时不管这个优化问题的求解, 而先来看看支撑向量.

2.3 支撑向量

直观上来看, 必然存在两个与决策超平面平行的超平面, 而且它们到决策超平面的距离必然相等 (均为 $\tilde{\gamma}$), 否则可以通过调整来增大 $\tilde{\gamma}$ 的值, 也即它们的间隔 (gap) 为 $2\tilde{\gamma}$, 这两个超平面上肯定都会有样本点存在, 否则我们就可以扩大 gap, 也就是增大 $\tilde{\gamma}$ 的值了, 这些样本点, 我们就称为支撑向量 (我们知道, 在 n 维空间中, 一个点和以原点为起点, 该点为终点的向量是等价的).

显然, 由于这些支撑向量刚好在边界上, 因此它们就是取得最优值的地方, 所以它们满足 $y(\boldsymbol{w}^T\boldsymbol{x}+b)=1$(之前令 Functional Margin 为 1), 而对于所有不是支撑向量的点, 也就是在"阵地后方"的点, 都有 $y(\boldsymbol{w}^T\boldsymbol{x}+b)>1$, 事实上, 当最优的决策超平面确定下来之后, 这些后方的点就完全成了"路人"了, 飘来飘去都不会再有任何影响.

当然, 除了从几何直观上, 支撑向量的概念也可以从其优化过程的推导中得到.

回顾我们的优化模型, 如下

$$\max \frac{1}{2}||\boldsymbol{w}||^2, \quad s.t. \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)\geqslant 1, i=1,2,\cdots,N$$

虽然已经有很多优化包可以高效的求解二次规划, 比如我们在数值优化方法中学过用有效集方法求解二次规划, 但是 SVM 这个优化问题结构很特殊, 通过 Lagrange 对偶形变换到对偶变量 (Dual Variable) 的优化问题之后, 可以找到一种更加有效的方法来进行求解. 这也是 SVM 盛行的一大原因, 通常情况下这种方法比直接使用通用的 QP 优化包进行优化要高效得多. 此外, 在推导过程中, 许多有趣的特征也会被揭露出来, 包括刚才提到的支撑向量的问题.

现在我们就来进行推导, 构造 Lagrange 函数

$$\mathcal{L}(w,b,\boldsymbol{\alpha})=\frac{1}{2}||\boldsymbol{w}||^2-\sum_{i=1}^{N}\alpha_i\left(y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)-1\right)$$

其中 $\alpha_i\geqslant 0, i=1,2,\cdots,N$ 为 Lagrange 乘子 (Lagrange multiplier).

关于 Lagrange 对偶形及处理方法我们最优化课上也讲的不多, 其一般原理和结论这里也不进行讨论, 只结合我们当前的优化问题进行论述.

令 $\partial\mathcal{L}/\partial\boldsymbol{w}$ 和 $\partial\mathcal{L}/\partial b$ 等于零, 可得

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}}=0\Rightarrow\boldsymbol{w}=\sum_{i=1}^{N}\alpha_i y_i\boldsymbol{x}_i$$

$$\frac{\partial\mathcal{L}}{\partial b}=0\Rightarrow\sum_{i=1}^{N}\alpha_i y_i=0$$

代回到 $\mathcal{L}$ 中可得

$$\begin{aligned}\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\alpha})&=\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\boldsymbol{x}_i^T\boldsymbol{x}_j-\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\boldsymbol{x}_i^T\boldsymbol{x}_j-b\sum_{i=1}^{N}\alpha_i y_i+\sum_{i=1}^{N}\alpha_i\\&=\sum_{i=1}^{N}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j y_i y_j\boldsymbol{x}_i^T\boldsymbol{x}_j\end{aligned}$$

由此我们得到关于对偶变量 $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_N)^T$ 的对偶优化问题

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} \sum\limits_{i=1}^{N} \alpha_i - \dfrac{1}{2} \sum\limits_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \\ s.t. \quad \alpha_i \geqslant 0, i = 1, 2, \cdots, N \\ \qquad \sum\limits_{i=1}^{N} \alpha_i y_i = 0 \end{cases}$$

如前所述, 这个问题有更加高效的优化算法, 不过具体方法在这里先不介绍, 求得 $\boldsymbol{\alpha}$ 之后如何得到 $\boldsymbol{w}$ 和 b 也暂不介绍 (当然, 求得 $\boldsymbol{\alpha}$ 后 $\boldsymbol{w}$ 自然就算出来了), 先来看看由推导过程所得到的一些结论.

首先, 是关于超平面, 对于一个样本点 $\boldsymbol{x}$ 进行分类, 实际上是通过把 $\boldsymbol{x}$ 代入到 $f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b$ 算出结果然后根据其正符号来进行类别划分的. 在推导中我们得到

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i$$

因此可得

$$\begin{aligned} f(\boldsymbol{x}) &= \left(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \right)^T \boldsymbol{x} + b \\ &= \sum_{i=1}^{N} \alpha_i y_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + b \end{aligned}$$

也就是说, 对于新点 $\boldsymbol{x}$ 的预测, 只需要计算它与训练数据点的内积即可 (本文中的范数均为普通的 2-范数, 内积为标准欧氏内积). 这一点至关重要, 是之后使用 Kernel 进行非线性推广的基本前提.

此外, 所谓支撑向量的概念也在这里显示了出来. 事实上, 所有不是支撑向量的那些样本点所对应的系数 α_i 都是等于零的, 因此对于新点的内积计算实际上只要针对少量的支撑向量而不是所有的训练样本即可.

为什么非支撑向量对应的 α_i 等于零呢? 直观上来理解的话, 就是这些“后方”的点——我们前面提到过, 对超平面是没有影响的, 由于分类完全由超平面决定, 所以这些无关的点并不会参与分类问题的计算, 因而也就不会产生任何影响了. 这个结论也可由刚才的推导中得出, 回顾刚才的 Lagrange 函数

$$\max_{\alpha_i \geqslant 0} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \max_{\alpha_i \geqslant 0} \frac{1}{2} ||\boldsymbol{w}||^2 - \sum_{i=1}^{N} \alpha_i \big(y_i (\boldsymbol{w}^T \boldsymbol{x}_i + b) - 1 \big)$$

注意到如果样本点 $\boldsymbol{x}_i$ 是支撑向量的话, 上式中红颜色的部分是等于 0 的 (因为支撑向量的 Functional Margin 等于 1), 而对非支撑向量来说, 其

Functional Margin 是大于 1 的, 因此红颜色部分是大于 0 的, 而 α_i 又是非负的, 为了满足最大化, 其 α_i 必须等于 0 (也就是说求最大值的过程中, α_i 肯定取 0), 这也就是那些非支撑向量的点的悲惨命运了 (当然, 从 KKT 条件也可以说明, 可参考周志华的《机器学习》).

至此, 基本的 SVM 模型就结束了, 但它不能处理非线性的情况, 下面我们就利用核方法把 SVM 推广到非线性的情况.

3 核方法 (Kernel)

如果数据集不是线性可分的, 或者说决策边界不是一个线性超平面, 那么我们该怎么办呢?

前面已经介绍过线性回归和 Logistic 回归, 它们看上去是线性模型, 但我们知道, 通过映射到高维空间的办法, 其实它们本质上都可以处理非线性的复杂模型. SVM 当然也不例外, 比如对于 $\boldsymbol{x} = (x_1, x_2)^T$, 我们可以将其映射到 2 次完全多项式, 即

$$\phi(\boldsymbol{x}) = (1, x_1, x_2, x_1^2, x_2^2, x_1x_2)^T$$

这样, 可以假设超平面的方程为

$$w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2 + w_5x_1x_2 + b = 0$$

我们知道这是一个圆锥曲线, 但是, 如果我们令 $z_1 = x_1, z_2 = x_2, z_3 = x_1^2, z_4 = x_2^2, z_5 = x_1x_2$, 则上面的方程在新的坐标系下可以写作

$$\sum_{i=1}^{5} w_iz_i + b = 0$$

这实际上就是 5 维空间中的一个超平面, 也就是说, 如果我们做一个映射 $\phi : \mathbb{R}^2 \mapsto \mathbb{R}^5$, 使 $\boldsymbol{x}$ 映射为 $\boldsymbol{z}$, 那么在新的空间中原来的数据就可能变成线性可分的, 从而就又化为了普通的线性 SVM, 可以利用之前推导的方法处理. 这就是 Kernel 方法处理非线性问题的基本思想, 可见下图

这样一来, 如同线性回归和 Logistic 回归一样, 好像我们的问题就解决了, 但是 SVM 还稍微不同 (毕竟这一节的主题 Kernel 还没出来呢). 回忆一下, 我们最终得到的分类函数是这样的:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_iy_i\langle \boldsymbol{x}_i, \boldsymbol{x}\rangle + b$$

现在则是映射后的空间, 即

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_iy_i\langle \phi(\boldsymbol{x}_i, \phi(\boldsymbol{x}))\rangle + b$$

而其中的 $\boldsymbol{\alpha}$ 也是通过求解如下对偶问题而得到的

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} \sum\limits_{i=1}^{N} \alpha_i - \dfrac{1}{2} \sum\limits_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \phi(\boldsymbol{x}_i, \phi(\boldsymbol{x})) \rangle \\ s.t. \quad \alpha_i \geqslant 0, i = 1, 2, \cdots, N \\ \qquad \sum\limits_{i=1}^{N} \alpha_i y_i = 0 \end{cases}$$

看似计算并没有什么问题, 但这样实际计算量是非常大的, 刚才我们对一个二维空间做映射, 选择的新空间是原始空间的所有一阶和二阶的组合, 得到了 5 个维度; 如果原始空间是三维, 那么我们会得到 19 维的新空间, 这个数目是爆炸性增长的, 也就给 $\phi(\cdot)$ 的计算带来了非常的困难, 而且如果遇到无穷维的情况 (确实会遇到的), 就根本无从计算了. 所以就要轮到 Kernel 出场了.

还是以开始的二维例子进行说明, 对于两个样本 $\boldsymbol{x}_1 = (\eta_1, \eta_2)^T$ 和 $\boldsymbol{x}_2 = (\xi_1, \xi_2)^T$, 设 $\phi(\cdot)$ 是上面的 5 维空间的映射, 即有

$$\begin{aligned} \phi(\boldsymbol{x}_1) &= (\eta_1, \eta_2, \eta_1^2, \eta_2^2, \eta_1\eta_2)^T \\ \phi(\boldsymbol{x}_2) &= (\xi_1, \xi_2, \xi_1^2, \xi_2^2, \xi_1\xi_2)^T \end{aligned}$$

因此映射后要计算的内积为

$$\langle \phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2) \rangle = \eta_1\xi_1 + \eta_1^2\xi_1^2 + \eta_2\xi_2 + \eta_2^2\xi_2^2 + \eta_1\eta_2\xi_1\xi_2 \tag{2}$$

此外, 我们注意到 (先别不要管这个是怎么猜到的)

$$\begin{aligned} (\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle + 1)^2 &= (\eta_1\xi_1 + \eta_2\xi_2 + 1)^2 \\ &= 2\eta_1\xi_1 + \eta_1^2\xi_1^2 + 2\eta_2\xi_2 + \eta_2^2\xi_2^2 + 2\eta_1\eta_2\xi_1\xi_2 + 1 \end{aligned} \tag{3}$$

二者有很多相似的地方, 实际上, 我们只要把式 (2) 某几个维度线性缩放一下, 然后再加上一个常数维度就可以得到式 (3), 更具体的, 对于一个样本 $\boldsymbol{x} = (x_1, x_2)^T$, 如果我们构造一个这样的映射 (与 $\phi(\cdot)$ 类似)

$$\varphi(\boldsymbol{x}) = (\sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2, 1)^T$$

那么在此映射下, 则有

$$\langle \varphi(\boldsymbol{x}_1), \varphi(\boldsymbol{x}_2) \rangle = 2\eta_1\xi_1 + \eta_1^2\xi_1^2 + 2\eta_2\xi_2 + \eta_2^2\xi_2^2 + 2\eta_1\eta_2\xi_1\xi_2 + 1 \tag{4}$$

这也就和式 (3) 的结果一模一样了.

区别在什么地方呢? 实际上, 式 (4) 是先映射到高维空间中, 然后再根据内积的公式进行计算; 而式 (3) 则是直接在原来的低维空间中进行计算, 而

不需要显式地写出映射后的结果. 回忆刚才提到的映射的维度爆炸, 在前一种方法已经无法计算的情况下, 后一种方法却依旧能从容处理, 甚至是无穷维度的情况也没有问题.

因此, 式 (3) 是非常巧妙的, 本来我们要把原数据通过映射 $\varphi(\cdot)$ 映射到高维空间中进行计算, 但是借助式 (3) 计算可以得到相同的结果, 我们把式 (3) 等式左边的函数就叫做核函数 (Kernel Function), 也即

$$\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle + 1)^2$$

核函数能简化映射空间中的内积运算, 而"碰巧"的是, 在 SVM 模型中, 需要计算的地方数据向量总是以内积的形式出现的. 对比刚才我们写出来的式子, 现在我们的分类函数为

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i y_i \kappa(\boldsymbol{x}_i, \boldsymbol{x}) + b$$

其中 $\boldsymbol{\alpha}$ 由如下对偶问题计算而得

$$\begin{cases} \max\limits_{\boldsymbol{\alpha}} \sum\limits_{i=1}^{N} \alpha_i - \dfrac{1}{2} \sum\limits_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}) \\ s.t. \quad \alpha_i \geqslant 0, i = 1, 2, \cdots, N \\ \qquad \sum\limits_{i=1}^{N} \alpha_i y_i = 0 \end{cases}$$

这样一来计算问题就解决了, 避开了直接在高维空间中进行计算, 而结果却是等价的. 但是, 因为我们刚才的例子比较简单, 所以可能容易构造出对应于映射 $\varphi(\cdot)$ 的核函数出来, 但是对于任意一个映射, 应该如何构造出对应的核函数呢?

最理想的情况下, 我们希望知道数据的具体形状和分布, 从而可以猜出一个刚好将数据映射成线性可分的 $\varphi(\cdot)$, 然后通过这个 $\varphi(\cdot)$ 得出对应的核函数 $\kappa(\cdot, \cdot)$, 然后再进行计算即可. 第一步可能还比较容易, 画出数据的分布图, 我们有可能凭肉眼猜到大致的映射. 可是第二步通常是非常困难甚至是完全无法做到的. 不过, 其实第一步也很难做到, 因为二维 (也就是有两个特征) 还容易画出来, 可是到高维情形呢? 所以, 我们通常并不是精确的寻找这个映射的, 而是先"随便"猜的, 当然可以多猜几次. 既然映射是猜的, 那我们也根本没有必要精确的找出对应于映射的那个核函数, 而且由于它们之间是对应关系, 因此我们干脆直接猜核函数即可. 因为猜一个核函数, 就相当于猜一个映射 (虽然不太容易求出来, 也就是说我们并不知道这个映射具体是什么). 由于我们的计算只要利用核函数即可, 所以我们并不关心也没有必要求出所对应的映射的具体形式. 这是利用核函数的关键思想.

当然, 虽说是猜, 但也是有一定讲究的, 因为并不是任意的函数都可以作为核函数. 通常人们会从一些常用的核函数族中选择 (根据问题和数据的不同, 选择不同的参数, 实际上就是得到了不同的核函数). 常用的核函数族有

1. 多项式核

$$\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle + R)^d$$

显然我们刚才举的例子就是这里多项式核的一个特例 ($R = 1, d = 2$). 虽然比较麻烦, 而且没有必要, 不过这个核所对应的映射实际上是可以写出来的, 新空间的维数是 C_{n+d}^d, 其中 n 是原始空间的维数.

2. 高斯核

$$\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp\left(-\frac{||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2}{2\sigma^2}\right)$$

这个核就是最开始提到过的会将原始空间映射为无穷维空间的那个家伙. 不过, 如果 σ 选的很大的话, 高次特征上的权重实际上衰减的非常快, 所以实际上 (数值上近似一下) 相当于一个低维的空间; 反过来, 如果 σ 选的很小, 则可以将任意的数据映射为线性可分的 (这里不给出证明). 当然, 这并不一定是件好事, 也许你已经猜到了, 对, 其随之而来的可能是非常严重的过拟合问题. 不过, 总的来说, 通过调控参数 σ, 高斯核实际上具有相当高的灵活性, 也是使用最广泛的核函数之一.

在 R 语言和 Python 的实现中, 一般采用如下形式, 并称之为 RBF(Gaussian Radial Basis Function) 核

$$\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\gamma||\boldsymbol{x}_1 - \boldsymbol{x}_2||^2)$$

相应的, 也有 Laplace Radial Basis Function (RBF) 核, 如下

$$\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = \exp(-\gamma||\boldsymbol{x}_1 - \boldsymbol{x}_2||)$$

3. 线性核

$$\kappa(\boldsymbol{x}_1, \boldsymbol{x}_2) = \langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle$$

这实际上就是原始空间中的内积. 这个核存在的主要目的是使得“映射后空间中的问题”和“映射前空间中的问题”两者在形式上统一起来了.

最后, 总结一下: 对于非线性的情况, SVM 的处理方法是选择一个核函数 $\kappa(\cdot, \cdot)$, 通过将数据映射到高维空间来解决在原始空间中线性不可分的问题. 由于核函数的优良品质, 这样的非线性扩展在计算量上并没有比原来复杂多少, 这一点是非常难得的. 当然, 这要归功于核方法, 事实上, 除了 SVM 之外, 任何将计算表示为数据点的内积的方法, 都可以使用核方法进行非线性扩展.

4 软间隔与正则化

在最开始讨论支持向量机的时候, 我们就假定数据是线性可分的, 亦即我们可以找到一个可行的超平面将数据完全分开. 后来为了处理非线性数据, 使用 Kernel 方法对原来的线性 SVM 进行了推广, 使得非线性的的情况也能处理. 虽然通过映射 $\varphi(\cdot)$ 将原始数据映射到高维空间之后, 能够线性分隔的概率大大增加, 但是对于某些情况还是很难处理. 例如可能并不是因为数据本身是非线性结构的, 而只是因为数据有噪音. 对于这种偏离正常位置很远的数据点, 我们称之为异常值 (Outliers), 在我们原来的 SVM 模型里, Outlier 的存在有可能造成很大的影响, 因为超平面本身就是只有少数几个 Support Vector 组成的, 如果这些 Support Vector 里又存在 Outlier 的话, 其影响就很大了. 例如下图

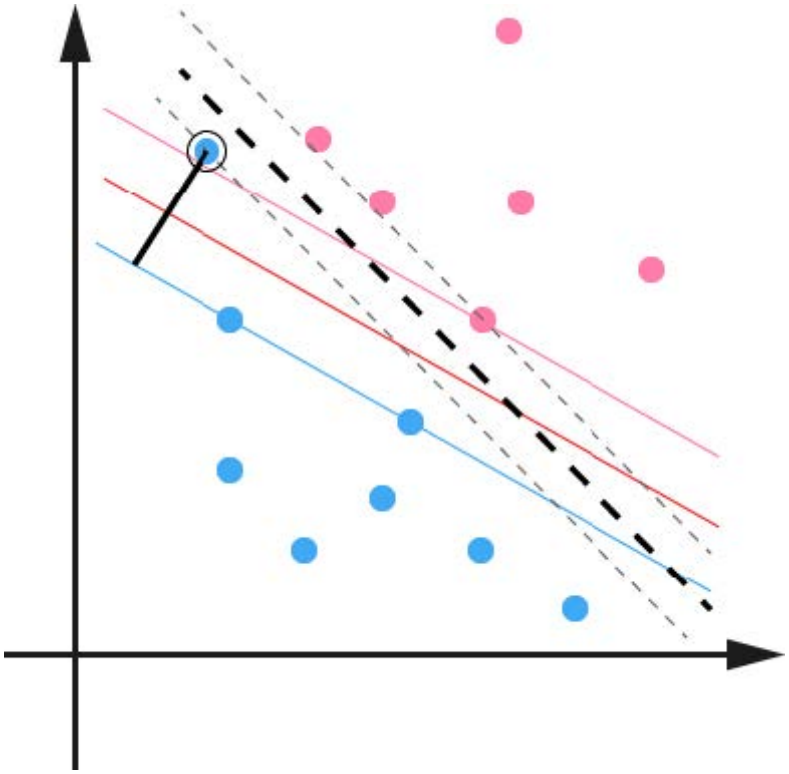

图 1: 异常值

用黑圈圈起来的那个蓝点是一个 Outlier, 它偏离了自己原本所应该在的那个半空间, 如果直接忽略掉它的话, 原来的分隔超平面还是挺好的, 但是由于这个 Outlier 的出现, 导致分隔超平面不得不被挤歪了, 变成图1中黑色虚线所示 (这只是一个示意图, 并没有严格计算精确坐标), 同时 Margin 也

相应变小了. 当然, 更严重的情况是, 如果这个 Outlier 再往右上移动一些距离的话, 我们将无法构造出能将数据分开的超平面来.

为了处理这种情况, SVM 允许数据点在一定程度上偏离一下超平面. 例如上图1中, 黑色实线所对应的距离就是该 Outlier 偏离的距离, 如果把它移动回来, 就刚好落在原来的超平面上, 而不会使得超平面发生变形了. 具体来说, 原来的约束条件

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)\geqslant 1, i=1,2,\cdots,N$$

现在变成

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)\geqslant 1-\xi_i, i=1,2,\cdots,N$$

其中 $\xi_i\geqslant 0$ 称为松弛变量 (Slack Variable), 对应数据点 $\boldsymbol{x}_i$ 允许偏离的 Functional Margin 的量. 原来的间隔称为“硬间隔”, 现在的间隔称为“软间隔”. 当然, 如果我们允许 ξ_i 任意大的话, 那任意的超平面都是符合条件的了. 所以, 我们要对 ξ_i 施加惩罚, 即在原来的目标函数后面加上一项, 使得这些 ξ_i 的总和也要最小

$$\min \frac{1}{2}||\boldsymbol{w}||^2+C\sum_{i=1}^{N}\xi_i$$

其中 C 是一个参数, 用于控制目标函数中两项 (“寻找 Margin 最大的超平面”和“保证数据点偏差量最小”) 之间的权重. 注意, 其中 $\boldsymbol{\xi}$ 是需要优化的变量 (之一), 而 C 是一个事先确定好的常量. 完整地写出来是这个样子

$$\begin{cases}\min & \frac{1}{2}||\boldsymbol{w}||^2+C\sum_{i=1}^{N}\xi_i\\ s.t. & y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)\geqslant 1-\xi_i\\ & \xi_i\geqslant 0, i=1,2,\cdots,N\end{cases}$$

采用 Lagrange 方法, 即构造

$$\mathcal{L}(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{r})=\frac{1}{2}||\boldsymbol{w}||^2+C\sum_{i=1}^{N}\xi_i-\sum_{i=1}^{N}\alpha_i(y_i(\boldsymbol{w}^T\boldsymbol{x}_i+b)-1+\xi_i)-\sum_{i=1}^{N}r_i\xi_i$$

然后求偏导

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\boldsymbol{w}}&=0\Rightarrow\boldsymbol{w}=\sum_{i=1}^{N}\alpha_i y_i\boldsymbol{x}_i\\ \frac{\partial\mathcal{L}}{\partial b}&=0\Rightarrow\sum_{i=1}^{N}\alpha_i y_i=0\\ \frac{\partial\mathcal{L}}{\partial\xi_i}&=0\Rightarrow C-\alpha_i-r_i=0, i=1,2,\cdots,N\end{aligned}$$

将 $\boldsymbol{w}$ 代回 $\mathcal{L}$ 并化简, 得到和原来一样的目标函数

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$

不过, 由于我们得到 $C - \alpha_i - r_i = 0$, 而又有 $r_i \geqslant 0$(作为 Lagrange Multiplier 的条件), 因此有 $\alpha_i \leqslant C$, 所以整个对偶问题现在写作

$$\begin{cases} \max & \sum\limits_{i=1}^{N} \alpha_i - \frac{1}{2} \sum\limits_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\ s.t. & 0 \leqslant \alpha_i \leqslant C, i = 1, 2, \cdots, N \\ & \sum\limits_{i=1}^{N} \alpha_i y_i = 0 \end{cases}$$

和之前的结果对比一下, 可以看到唯一的区别就是现在 Dual Variable $\boldsymbol{\alpha}$ 多了一个上限 C. 而 Kernel 化的非线性形式也是一样的, 只要把 $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ 换成 $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ 即可. 这样一来, 一个完整的, 可以处理线性和非线性并能容忍噪音和 Outliers 的支持向量机才终于介绍完毕了. Python 的机器学习包 sklearn 中关于 SVM 的模型就是这个, 具体可见http://scikit-learn.org/stable/modules/svm.html.

5 SMO 方法

这里介绍一下对偶问题的数值解法. 前面提到过, 对偶问题有着高效的解法, 其实就是所谓的 Sequential Minimal Optimization (SMO) 方法.

先回忆一下我们之前得出的要求解的对偶问题:

$$\begin{cases} \max & \sum\limits_{i=1}^{N} \alpha_i - \frac{1}{2} \sum\limits_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ s.t. & 0 \leqslant \alpha_i \leqslant C, i = 1, 2, \cdots, N \\ & \sum\limits_{i=1}^{N} \alpha_i y_i = 0 \end{cases}$$

对于变量 $\boldsymbol{\alpha}$ 来说, 这是一个二次函数. 对于优化问题, 我们最常用的可能就是梯度下降法 (这里是求最大值, 不过加个负号就化为最小了), 也就是 Gradient Descent, 其实还有一种叫做 Coordinate Descent 的变种, 它每次只选择一个维度, 例如本模型的 $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_N)^T$, 它每次选取一个 α_i 为变量, 而将 $\alpha_1, \cdots, \alpha_{i-1}, \alpha_{i+1}, \cdots, \alpha_N$ 都看成是常量, 从而原始的问题在这一步变成一个一元函数, 然后针对这个一元函数求最小值, 如此反复轮换不同的维度进行迭代. 这样也确实能求得问题的最小点, 至于原因这里不做介绍. Coordinate Descend 的主要用处在于那些原本很复杂, 但是如果只限

制在一维的情况下则变得很简单甚至可以直接求极值的情况, 例如我们这里的问题, 暂且不管约束条件, 如果只看目标函数的话, 当 α 只有一个分量是变量的时候, 这就是一个普通的一元二次函数的极值问题, 是比较容易的.

然而这里还有一个问题就是约束条件的存在, 其实如果没有约束条件的话, 本身就是一个一元二次函数的问题, 极易求解. 但是有了约束条件, 结果就让 Coordinate Descent 失效了, 为什么呢? 比如我们假设 α_1 是变量, 而 $\alpha_2, \cdots, \alpha_N$ 是固定的值的话, 那么其实没有什么好优化的, 直接根据第二个约束条件 $\sum_{i=1}^{N} \alpha_i y_i = 0$, α_1 的值立即就定下来了, 所以优化无从谈起. 如果还是迭代每个坐标维度, 会发现优化根本无法进行, 因为迭代了一轮之后会发现没有任何进展, 一切都停留在初始值.

所以 Sequential Minimal Optimization (SMO) 就出场了. 它一次选取两个坐标维度来进行优化, 不失一般性, 比如选取 α_1 和 α_2 为变量, 其余为常量, 则根据约束条件我们有

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \Rightarrow \alpha_2 = \frac{1}{y_2} \left(-\sum_{i=3}^{N} \alpha_i - \alpha_1 y_1 \right) \triangleq y_2(K - \alpha_1 y_1)$$

其中那个从 3 到 N 的求和加负号视为常量, 所以记为了 K, 而且由于 $y \in \{-1, +1\}$, 所以 y_2 和 $1/y_2$ 相等, 所以可以变到分子上来.

现在把这个式子代入到原来的目标函数中, 就可以消去 α_2, 从而可以变成一个一元二次函数, 具体展开的形式这里就不写了, 总之现在变成了一个非常简单的问题——带区间约束的一元二次函数极值问题. 唯一需要注意的就是这里的约束条件, 一个是 α_1 本身要满足 $0 \leqslant \alpha_1 \leqslant C$, 另一个就是 α_2 也要满足相同的约束条件, 即

$$0 \leqslant y_2(K - \alpha_1 y_1) \leqslant C$$

这也是 α_1 的约束条件, 同 $[0, C]$ 取交集才是最终的可行区间. 几何直观上看, 原本关于 α_1 和 α_2 的区间约束构成下图2中的绿色的正方形, 而另一个约束条件 $\alpha_1 y_1 + \alpha_2 y_2 = K$ 实际上是一条直线, 两个集合的交集即是图中红颜色的线段, 投影到 α_1 轴上所对应的区间即为 α_1 的取值范围, 在这个区间内求二次函数的最大值即可完成 SMO 的一步迭代.

SMO 每次选取不同的两个 Coordinate 维度进行优化, 而且从上面可以看出每一个迭代步骤实际上是一个可以直接求解的一元二次函数极值问题, 所以求解非常高效. 此外, SMO 也并不是依次或者随机地选取两个坐标维度, 而是有一些启发式的策略来选取最优的两个坐标维度, 具体的选取方法 (和其他的一些细节) 这里就不多说了.

最后, 有一点需要指出的是, 虽然我们从对偶问题的推导中得出了许多 SVM 的优良性质, 但是 SVM 的数值优化 (即使是非线性的版本) 其实并不

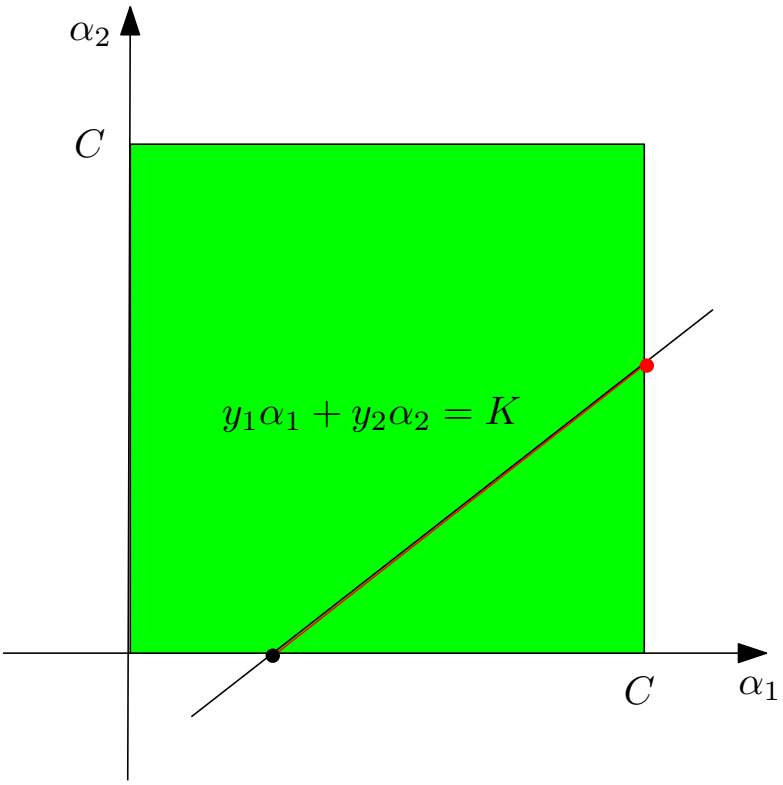

图 2: 可行域示意图

一定需要转化为对偶问题来完成, 也有人研究了直接求解原始优化问题的方法, 这里不再多述.

6 总结

6.1 关于 SVM 的补充

本篇笔记主要参考的是 pluskid 的博客, 难免有些疏漏, 这里提一些补充.

第一点, 是关于 SVM 的损失函数.

我们从软间隔说起. 软间隔允许某些样本不满足约束

$$y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geqslant 1$$

当然, 在最大化软间隔的同时, 不满足约束的样本应尽可能少. 于是, 优

化目标可写为

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{N} \ell_{0/1}(y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) - 1)$$

其中 $C > 0$ 是一个常数, 也就是正则化参数, $\ell_{0/1}$ 是"0/1 损失函数"

$$\ell_{0/1}(z) = \begin{cases} 1, & \text{if } z < 0 \\ 0, & \text{otherwise} \end{cases}$$

显然, 当 C 无穷大时, 优化目标会迫使所有样本均满足硬间隔条件, 与硬间隔的 SVM 等价; 当 C 取有限值时, 优化目标允许一些样本不满足硬间隔约束.

然而, $\ell_{0/1}$ 非凸、非连续, 数学上不好求解, 于是人们通常用其他一些函数来代替 $\ell_{0/1}$, 称为"替代损失"(surrogate loss). 替代损失函数一般数学性质较好, 如它们通常是凸的连续函数且是 $\ell_{0/1}$ 的上界, 比如

(1) hinge 损失: $\ell_{\text{hinge}}(z) = \max(0, 1 - z)$

(2) 指数损失 (exponential loss): $\ell_{\exp}(z) = \exp(-z)$

(3) 对数损失 (logistic loss): $\ell_{\log} = \log(1 + \exp(-z))$

若采用 hinge 损失, 则优化目标变为

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{N} \max(0, 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b))$$

引入松弛变量 $\xi_i \geqslant 0$, 则可将上式重写为

$$\begin{aligned} & \min_{\boldsymbol{w},b,\xi_i} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{N} \xi_i \\ s.t. \quad & y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geqslant 1 - \xi_i \\ & \xi_i \geqslant 0, i = 1, 2, \cdots, N \end{aligned}$$

这也就是上面的软间隔的 SVM, 不过这次是直接从 hinge 损失函数的角度推出来的.

那么, 能否用其他损失函数替代呢?

当然是可以的. 特别的, 如果我们用对数损失函数代替, 则优化目标变为了

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{N} \log\left(1 + \exp(-y_i f(\boldsymbol{x}_i))\right)$$

其中 $f(\boldsymbol{x}_i) = \boldsymbol{w}^T\boldsymbol{x}_i + b$.

回忆一下, 这其实不正是正则化的 Logistic 回归的损失函数吗?

在 Logistic 回归中, 如果用 $y = \pm 1$ 区分正负类, 正则化的损失函数为

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\log(1+\exp(-y_i f(\boldsymbol{x}_i))) + \lambda||\boldsymbol{\theta}||^2$$

因此, 如果用对数损失函数代替, 就变成了 Logistic 回归. 进一步, 如果也引入核函数, 那么可以得到 Kernel Logistic Regression(KLR), 相关讨论这里不再多述. 总之, 替换成一般的损失函数, 我们可以得到一般的模型形式

$$\min_f \Omega(f) + C\sum_{i=1}^{N}\ell(f(\boldsymbol{x}_i), y_i)$$

其中 f 表示我们的模型.

第二点, 是关于 SVM 模型的表达方式.

追本溯源, SVM 的基础模型是

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$$

之后, 引入映射 ϕ 后的完整模型为

$$f(\boldsymbol{x}) = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b$$

而且, 最后我们发现 $f(\boldsymbol{x})$ 可以求解表示为

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{w}^T\phi(\boldsymbol{x}) + b \\ &= \sum_{i=1}^{N}\alpha_i y_i \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}) + b \\ &= \sum_{i=1}^{N}\alpha_i y_i \kappa(\boldsymbol{x}, \boldsymbol{x}_i) + b \end{aligned}$$

也就是模型可以通过训练样本的核函数展开, 或者说模型可表示为核函数 $\kappa(\boldsymbol{x}, \boldsymbol{x}_i)$ 的线性组合. 因此, 有的资料上直接把 SVM 模型表示为

$$f(\boldsymbol{x}; \boldsymbol{w}) = \sum_{i=1}^{N} w_i \kappa(\boldsymbol{x}, \boldsymbol{x}_i) + w_0$$

其中 $w_0, w_1, \cdots, w_N$ 为模型参数.

6.2 参考资料

(1) 周志华的《机器学习》

(2) pluskid 的系列博客: http://blog.pluskid.org/?page_id=683, 这是 pluskid 写的支持向量机系列, 写的真是很好, 不多说了, 我的这篇笔记可以当做他博客的一个重新排版吧. 不过还是有一些 SVM 的算法细节 (比如 SMO 算法) 没有涉及, 当然, 我也没有涉及.

(3) zouxy09 的系列博客: http://blog.csdn.net/zouxy09/article/details/17292011, 写的也不错, 还有自己的 Python 实现. 他博客最后给出的参考资料中也有很多不错.

(4) JerryLead 的系列博客: http://www.cnblogs.com/jerrylead/archive/2011/03/13/1982639.html, 推导比较详细.

(5) 博客: http://tullo.ch/articles/svm-py/, 主要是 Python 代码的实现, 里面二次优化用到了 Python 的 cvxopt 包, 并没有使用 SMO 算法, 与此相似的还有这个: https://gist.github.com/mblondel/586753, 其对应的博客在 http://www.mblondel.org/journal/2010/09/19/support-vector-machines-in-python/

(6) Andrew Ng 的 Notes 还有他公开课的材料, 其实前面的线性回归和 Logistic 回归他的公开课里都有, 也都参考过, 习题代码均是用 Matlab 所写, 参考代码可见 https://github.com/emilmont/Artificial-Intelligence-and-Machine-Learning/tree/master/ML

(7) 笔记: https://rstudio-pubs-static.s3.amazonaws.com/65566_a44a67a726284943b8f1ec986bf9642d.html, 用 R 的 e1071 包做 SVM , 感觉是用 RMarkdown 写的, 还有另一篇也不错: https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/ch9.html

(8) stackoverflow: http://stackoverflow.com/questions/22294241/plotting-a-decision-boundary-separating-2-classes-using-matplotlibs-pyplot, 用 python 的 sklearn 包做了 SVM, 里面有数据的生成和决策边界的画图.

6.3 SVM 的应用场景

6.4 SVM 的优缺点

附录