

支持向量机通俗导论

理解 SVM 的三层境界

作者：July

致谢：pluskid、白石、JerryLead

出处：[结构之法算法之道](#) blog

目录

1 第一层：了解 SVM	4
1.1 分类标准的起源：Logistic 回归	4
1.2 线性分类的一个例子	5
1.3 函数间隔 Functional margin 与几何间隔 Geometrical margin	8
1.4 最大间隔分类器 Maximum Margin Classifier 的定义	9
2 第二层：深入 SVM	12
2.1 从线性可分到线性不可分	12
2.1.1 从原始问题到对偶问题的求解	12
2.1.2 K.K.T. 条件	13
2.1.3 对偶问题求解的 3 个步骤	14
2.1.4 线性不可分的情况	16
2.2 核函数：Kernel	18
2.2.1 特征空间的隐式映射：核函数	18
2.2.2 核函数：如何处理非线性数据	19
2.2.3 几个核函数	22
2.2.4 核函数的本质	23
2.3 使用松弛变量处理 outliers 方法	24
3 证明 SVM	29
3.1 线性学习器	29
3.1.1 感知器算法	29
3.2 非线性学习器	31
3.2.1 Mercer 定理	31
3.3 损失函数	32
3.4 最小二乘法	34
3.4.1 什么是最小二乘法？	34
3.4.2 最小二乘法的解法	35
3.5 SMO 算法	37
3.5.1 SMO 算法的推导	37
3.5.2 SMO 算法的步骤	42
3.5.3 SMO 算法的实现	44
3.6 SVM 的应用	45
3.6.1 文本分类	45

前言

动笔写这个支持向量机 (Support Vector Machine¹) 是费了不少劲和困难的, 原因很简单, 一者这个东西本身就并不好懂, 要深入学习和研究下去需花费不少时间和精力, 二者这个东西也不好讲清楚, 尽管网上已经有朋友写得不错了 (见文末参考链接), 但在描述数学公式的时候还是显得不够。得益于同学白石的数学证明, 我还是想尝试写一下, 希望本文在兼顾通俗易懂的基础上, 真真正正能足以成为一篇完整概括和介绍支持向量机的导论性的文章。

本文在写的过程中, 参考了不少资料, 包括《支持向量机导论》、《统计学习方法》及网友 pluskid 的支持向量机系列等等, 于此, 还是一篇学习笔记, 只是加入了自己的理解和总结, 有任何不妥之处, 还望海涵。全文宏观上整体认识支持向量机的概念和用处, 微观上深究部分定理的来龙去脉, 证明及原理细节, 力保逻辑清晰 & 通俗易懂。

同时, 阅读本文时建议大家尽量使用 chrome 等浏览器, 如此公式才能更好的显示, 再者, 阅读时可拿张纸和笔出来, 把本文所有定理、公式都亲自推导一遍或者直接打印下来 (可直接打印网页版或本文文末附的 PDF, 享受随时随地思考、演算的极致快感), 在文稿上演算。

OKL, 还是那句原话, 有任何问题, 欢迎任何人随时不吝指正 & 赐教, 感谢。

¹http://en.wikipedia.org/wiki/Support_vector_machine

1 第一层：了解 SVM

支持向量机，因其英文名为 **Support Vector Machine**，故一般简称 SVM，通俗来讲，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，其学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

1.1 分类标准的起源：Logistic 回归

理解 SVM，咱们必须先弄清楚一个概念：线性分类器。

给定一些数据点，它们分别属于两个不同的类，现在要找到一个线性分类器把这些数据分成两类。如果用 x 表示数据点，用 y 表示类别（ y 可以取 1 或者 -1，分别代表两个不同的类），一个线性分类器的学习目标便是要在 n 维的数据空间中找到一个超平面（**hyper plane**），这个超平面的方程可以表示为（ w^T 中的 T 代表转置）：

$$w^T x + b = 0 \quad (1.1)$$

可能有读者对类别取 1 或 -1 有疑问，事实上，这个 1 或 -1 的分类标准起源于 *Logistic 回归*²。

定义 1 (Logistic 回归) *Logistic 回归* 目的是从特征学习出一个 0/1 分类模型，而这个模型是将特征的线性组合作为自变量，由于自变量的取值范围是负无穷到正无穷，因此，使用 *Logistic 函数*（或称作 *sigmoid 函数*）将自变量映射到 $(0, 1)$ 上，映射后的值被认为是属于 $y = 1$ 的概率。

假设函数：

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \quad (1.2)$$

其中， x 是 n 维特征向量，函数 g 就是 Logistic 函数。而 $g(z) = \frac{1}{1+e^{-z}}$ 的图像如图1，将无穷映射到了 $(0, 1)$ 。

而假设函数就是属于 $y = 1$ 的概率：

$$P(y = 1|x; \theta) = h_{\theta}(x) \quad (1.3)$$

$$P(y = 0|x; \theta) = 1 - h_{\theta}(x) \quad (1.4)$$

从而，当我们要判别一个新来的特征属于哪个类时，只需求 $h_{\theta}(x)$ 即可，若 $h_{\theta}(x)$ 大于 0.5 就是 $y = 1$ 的类，反之属于 $y = 0$ 类。

²http://en.wikipedia.org/wiki/Logistic_regression

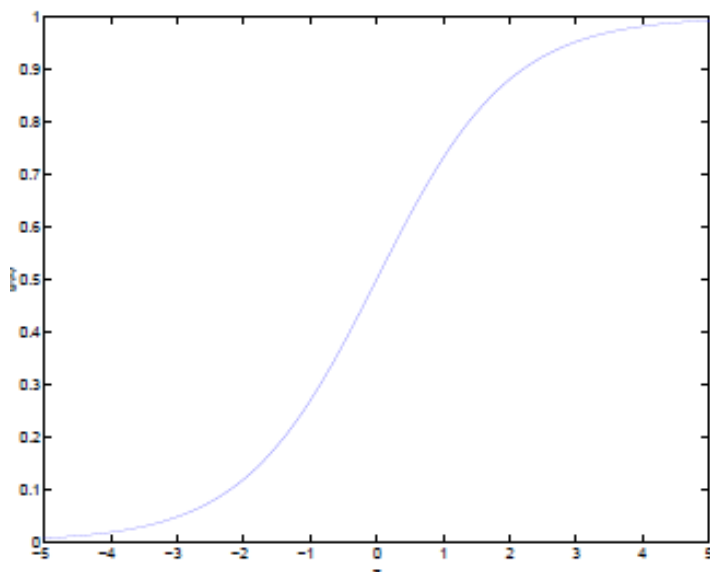


图 1: Logistic 函数图像

此外, $h_{\theta}(x)$ 只和 $\theta^T x$ 有关, $\theta^T x > 0$, 那么 $h_{\theta}(x) > 0.5$, 而 $g(z)$ 只是用来映射, 真实的类别决定权还是在于 $\theta^T x$ 。再者, 当 $\theta^T x \gg 0$ 时, $h_{\theta}(x) = 1$, 反之 $h_{\theta}(x) = 0$ 。如果我们只从 $\theta^T x$ 出发, 希望模型达到的目标就是让训练数据中 $y = 1$ 的特征 $\theta^T x \gg 0$, 而 $y = 0$ 的特征 $\theta^T x \ll 0$ 。Logistic 回归就是要学习得到 θ , 使得正例的特征远大于 0, 负例的特征远小于 0, 而且要在全部训练实例上达到这个目标。

接下来, 尝试把 Logistic 回归做个变形。首先, 将使用的结果标签 $y = 0$ 和 $y = 1$ 替换为 $y = -1$, $y = 1$, 然后将 $\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ ($x_0 = 1$) 中的 θ_0 替换为 b , 最后将后面的 $\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ 替换为 $w^T x$ 。如此, 则有了 $\theta^T x = w^T x + b$ 。也就是说除了 y 由 $y = 0$ 变为 $y = -1$ 外, 线性分类函数跟 Logistic 回归的形式化表示 $h_{\theta}(x) = g(\theta^T x) = g(w^T x + b)$ 没区别。

进一步, 可以将假设函数 $h_{w,b}(x) = g(w^T x + b)$ 中的 $g(z)$ 做一个简化, 将其简单映射到 $y = -1$ 和 $y = 1$ 上。映射关系如下:

$$g(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases} \quad (1.5)$$

1.2 线性分类的一个例子

下面举个简单的例子, 如图2所示, 现在有一个二维平面, 平面上有两种不同的数据, 分别用圈和叉表示。由于这些数据是线性可分的, 所以可以用一条直线将这两类数据分开, 这条直线就相当于一个超平面, 超平面一边的数据点所对应的 y 全是 -1 , 另一边所对应的 y 全是 1 。

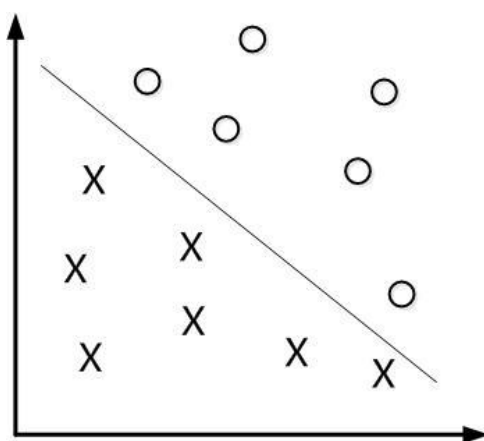


图 2: 线性分类示例

这个超平面可以用分类函数 $f(x) = w^T x + b$ 表示，当 $f(x)$ 等于 0 的时候， x 便是位于超平面上的点，而 $f(x)$ 大于 0 的点对应 $y = 1$ 的数据点， $f(x)$ 小于 0 的点对应 $y = -1$ 的点，如图3所示。

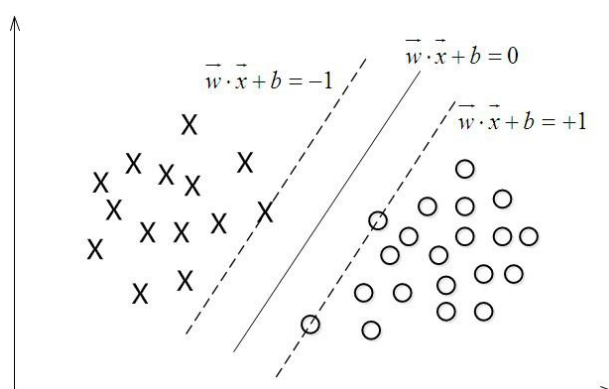


图 3: 支持向量示例

注 1 有的资料上定义特征到结果的输出函数 $u = \vec{w} \cdot \vec{x} - b$ ，与这里定义的 $f(x) = w^T x + b$ 实质是一样的。为什么？因为无论是 $u = \vec{w} \cdot \vec{x} - b$ ，还是 $f(x) = w^T x + b$ ，不影响最终优化结果。下文你将看到，当我们转化到优化 $\max_{\|w\|} \frac{1}{\|w\|}$ ，s.t. $y_i(w^T x_i + b) \geq 1, i = 1, \dots, n$ 的时候，为了求解方便，会把 $yf(x)$ 令为 1，即 $yf(x)$ 是 $y(w^T x + b)$ ，还是 $y(w^T x - b)$ ，对我们要优化的式子 $\max_{\|w\|} \frac{1}{\|w\|}$ 已无影响。

注 2 有一朋友飞狗来自 *Mare_Desiderii*，看了上面的定义之后，问道：请教一下 SVM functional margin 为 $\hat{\gamma} = y(w^T x + b) = yf(x)$ 中的 y 是只取 1 和 -1 吗？ y 的唯一作用就是确保 functional margin 的非负性？

与白石讨论后，我来具体回答下这个问题：你把问题搞混了。 y 是个分类标签，二分时 y 就取两个值，而刚好取了 -1 和 1 ，只是因为用超平面分类时，不同的类中的点函数值刚好有不同的符号，所以就用符号来进行分类。

具体阐述如下：

1. 对于二类问题，因为 y 只取两个值，这两个是可以任意取的，只要是取两个值就行；
2. 支持向量机去求解二类问题，目标是求一个特征空间的超平面，而超平面分开的两类对应于超平面的函数值的符号是刚好相反的；
3. 基于上述两种考虑，为了使问题足够简单，我们取 y 的值为 1 和 -1 ；
4. 在取定分类标签 y 为 -1 和 1 之后，那么，一个平面正确分类样本数据，就相当于用这个平面计算的那个 $y * f(x) > 0$ ；
5. 而且这样一来， $y * f(x)$ 有明确的几何含义；

总而言之：你要明白，二类问题的标签 y 是可以取任意两个值的，不管取怎样的值对于相同的样本点，只要分类相同，所有的 y 的不同取值都是等价的，之所以取某些特殊的值，只是因为这样一来计算会变得方便，理解变得容易。

正如朋友张磊所言，SVM 取 1 或 -1 的历史原因是因为感知器最初的定义，实际取值可以任意，总能明确表示输入样本是否被误分，但是用 $+1$ 、 -1 可以起码可以是问题描述简单化、式子表示简洁化、几何意义明确化。

举个例子，如你要是取 y 为 1 和 2 ，比如原来取 -1 的现在取 1 ，原来取 1 的现在取 2 ，这样一来，分类正确的判定标准变为 $(y - 1.5)f(x) > 0$ ，故取 1 和 -1 只是为了计算简单方便，没有实质变化，更非一定必须取一正一负。

当然，有些时候，或者说大部分时候数据并不是线性可分的，这个时候满足这样条件的超平面就根本不存在（不过关于如何处理这样的问题我们后面会讲），这里先从最简单的情形开始推导，就假设数据都是线性可分的，亦即这样的超平面是存在的。

换言之，在进行分类的时候，遇到一个新的数据点 x ，将 x 代入 $f(x)$ 中，如果 $f(x)$ 小于 0 则将 x 的类别赋为 -1 ，如果 $f(x)$ 大于 0 则将 x 的类别赋为 1 。

接下来的问题是，如何确定这个超平面呢？从直观上而言，这个超平面应该是最适合分开两类数据的直线。而判定“最适合”的标准就是这条直线离直线两边的数据的间隔最大。所以，得寻找有着最大间隔的超平面。

1.3 函数间隔 Functional margin 与几何间隔 Geometrical margin

在超平面 $w^T x + b = 0$ 确定的情况下， $|w^T x + b|$ 能够表示点 x 到距离超平面的远近³，而通过观察 $w^T x + b$ 的符号与类标记 y 的符号是否一致可判断分类是否正确，所以，可以用 $y(w^T x + b)$ 的正负性来判定或表示分类的正确性。于此，我们便引出了函数间隔（functional margin）的概念。

定义函数间隔（用 $\hat{\gamma}$ 表示）为：

几何距离和几何间隔

$$\hat{\gamma} = y(w^T x + b) = yf(x) \quad (1.6)$$

而超平面 (w, b) 关于 T 中所有样本点 (x_i, y_i) 的函数间隔最小值（其中， x 是特征， y 是结果标签， i 表示第 i 个样本），便为超平面 (w, b) 关于训练数据集 T 的函数间隔：

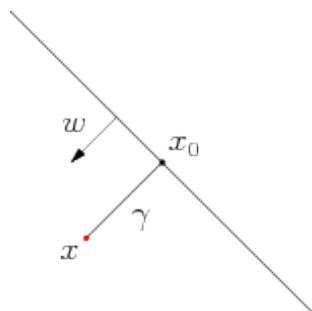
$$\hat{\gamma} = \min \hat{\gamma}_i, \quad (i = 1, \dots, n) \quad (1.7)$$

但这样定义的函数间隔有问题，即如果成比例的改变 w 和 b （如将它们改成 $2w$ 和 $2b$ ），则函数间隔的值 $f(x)$ 却变成了原来的 2 倍（虽然此时超平面没有改变），所以只有函数间隔还远远不够。

因为超平面是右侧为0的

事实上，我们可以对法向量 w 加些约束条件，从而引出真正定义点到超平面的距离——几何间隔（geometrical margin）的概念。

假定对于一个点 x ，令其垂直投影到超平面上的对应点为 x_0 ， w 是垂直于超平面的一个向量， γ 为样本 x 到分类间隔的距离，如图4所示。



函数间隔 & 几何间隔

图 4: 几何距离示例

有 $x = x_0 + \gamma \frac{w}{\|w\|}$ ，其中 $\|w\|$ 表示的是范数。

又由于 x_0 是超平面上的点，满足 $f(x_0) = 0$ ，代入超平面的方程 $w^T x + b = 0$ 即可算出⁴：

$$\text{gamma} \quad \gamma = \frac{w^T x + b}{\|w\|} = \frac{f(x)}{\|w\|} \quad (1.8)$$

³整理者注：这里的“远近”是个相对概念，如果是计算点到超平面距离为： $|w^T x + b|/\|w\|$

⁴有的书上会写成把 $\|w\|$ 分开相除的形式，如本文参考文献及推荐阅读条目《统计学习方法》，其中， $\|w\|$ 为 w 的二阶范数

为了得到 γ 的绝对值，令 γ 乘上对应的类别 y ，即可得出几何间隔（用 $\tilde{\gamma}$ 表示）的定义：

$$\tilde{\gamma} = y\gamma = \frac{\hat{\gamma}}{\|w\|} \quad (1.9)$$

从上述函数间隔和几何间隔的定义可以看出：几何间隔就是函数间隔除以 $\|w\|$ ，而且函数间隔 $y(w^T x + b) = yf(x)$ 实际上就是 $|f(x)|$ ，只是人为定义的一个间隔度量，而几何间隔 $|f(x)|/\|w\|$ 才是直观上的点到超平面的距离。

1.4 最大间隔分类器 Maximum Margin Classifier 的定义

对一个数据点进行分类，当超平面离数据点的“间隔”越大，分类的确信度（confidence）也越大。所以，为了使得分类的确信度尽量高，需要让所选择的超平面能够最大化这个“间隔”值。这个间隔如图5中的 $\text{Gap} / 2$ 所示。

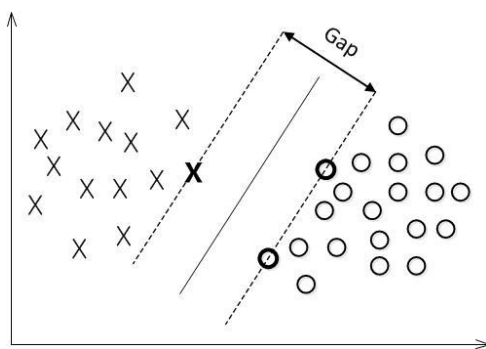


图 5: 超平面间隔示意

通过由前面的分析可知：函数间隔不适合用来最大化间隔值，因为在超平面固定以后，可以等比例地缩放 w 的长度和 b 的值，这样可以使得 $f(x) = w^T x + b$ 的值任意大，亦即函数间隔 $\hat{\gamma}$ 可以在超平面保持不变的情况下被取得任意大。但几何间隔因为除上了 $\|w\|$ ，使得在缩放 w 和 b 的时候几何间隔 $\tilde{\gamma}$ 的值是不会改变的，它只随着超平面的变动而变动，因此，这是更加合适的一个间隔。所以，这里要找的最大间隔分类超平面中的“间隔”指的是几何间隔。

于是最大间隔分类器（maximum margin classifier）的目标函数可以定义为：

$$\max \tilde{\gamma} \quad (1.10)$$

同时需满足一些条件，根据间隔的定义，有：

$$y_i(w^T x_i + b) = \hat{\gamma}_i \geq \hat{\gamma}, \quad i = 1, \dots, n \quad (1.11)$$

回顾一下几何间隔的定义 $\tilde{\gamma} = y\gamma = \frac{\hat{\gamma}}{\|w\|}$ 可知，如果令函数间隔 $\hat{\gamma}$ 等于 1，则有 $\tilde{\gamma} = 1/\|w\|$ 且 $y_i(w^T x_i + b) \geq 1, i = 1, \dots, n$ ，从而上述目标函数转化成了：

$$\max \frac{1}{\|w\|}, \quad s.t. \quad y_i(w^T x_i + b) \geq 1, i = 1, \dots, n \quad (1.12)$$

这个目标函数便是在相应的约束条件 $y_i(w^T x_i + b) \geq 1, i = 1, \dots, n$ 下，最大化这个 $1/\|w\|$ 值，而 $1/\|w\|$ 便是几何间隔 $\tilde{\gamma}$ 。

注 3 之所以令 $\hat{\gamma}$ 等于 1，是为了方便推导和优化，且这样做对于目标函数的优化：

1. 对于一个线性可分问题，任意给定一个分类正确的超平面，都有一个最小函数间隔，为了方便书写，记做 $\hat{\gamma}$ ，这个 $\hat{\gamma}$ ，是方向向量 w 和截距 b 的函数，因为分类的点是给定的；
2. 也就是说 $\tilde{\gamma} = \hat{\gamma}(w, b)$ ，而求最大几何间隔的问题就是要求出一组 w, b 使得 $\hat{\gamma}/\|w\|$ ，最大。此时约束的条件是任意给定的一个点的函数间隔大于等于 $\hat{\gamma}$ ，也就是 $y_i(w^T x_i + b) \geq \hat{\gamma}$ ；
3. 下面做一个变量替换，用 $w' = w/\hat{\gamma}$ ，和 $b' = b/\hat{\gamma}$ 代替上面的 w 和 b ，这样的新变量仍旧是 w 和 b 的函数，所以最大化仍然可以进行。于是，把这两个新的变量代入到原来的约束最大化问题中，就变成了，在 $y_i(w'^T x_i + b') \geq 1$ 的条件下，求使得 $1/\|w'\|$ 最大化的 w, b ；
4. 这样一来，通过一个变量替换，关于 w 和 b 的问题等价地换成了 w', b' 的问题，这就是支持向量机所采用的形式。

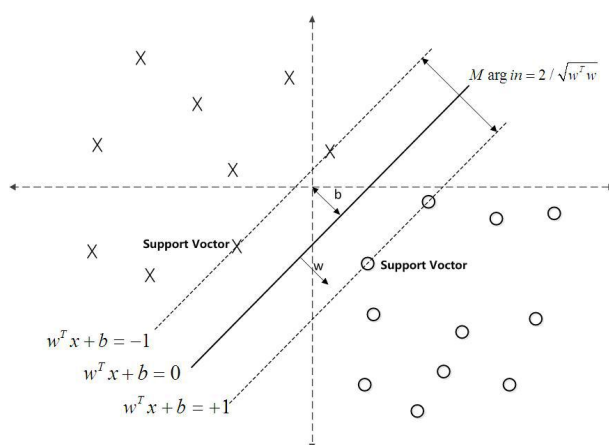


图 6: 超平面间隔示意

如图6所示，中间的实线便是寻找到的最优超平面（**Optimal Hyper Plane**），其到两条虚线的距离相等，这个距离便是几何间隔 $\tilde{\gamma}$ ，两条虚线之间的距离等于 $2\tilde{\gamma}$ ，而虚线

上的点则是支持向量。由于这些支持向量刚好在边界上，所以它们满足 $y(w^T x + b) = 1$ (还记得我们把 functional margin 定为 1 了吗？上节中：处于方便推导和优化的目的，我们可以令 $\hat{\gamma}=1$)，而对于所有不是支持向量的点，则显然有 $y(w^T x + b) > 1$ 。

OK，到此为止，算是了解到了 SVM 的第一层，对于那些只关心怎么用 SVM 的朋友便已足够，不必再更进一层深究其更深的原理。

2 第二层：深入 SVM

2.1 从线性可分到线性不可分

2.1.1 从原始问题到对偶问题的求解

接着考虑之前得到的目标函数：

$$\max \frac{1}{\|w\|}, \quad s.t. \quad y_i(w^T x_i + b) \geq 1, i = 1, \dots, n \quad (2.1)$$

由于求 $1/\|w\|$ 的最大值相当于求 $\frac{1}{2}\|w\|^2$ 的最小值，所以上述目标函数等价于（ w 由分母变成分子，从而也有原来的 \max 问题变为 \min 问题，很明显，两者问题等价）：

$$\min \frac{1}{2}\|w\|^2, \quad s.t. \quad y_i(w^T x_i + b) \geq 1, i = 1, \dots, n \quad (2.2)$$

因为现在的目标函数是二次的，约束条件是线性的，所以它是一个凸二次规划问题。这个问题可以用现成的 QP (Quadratic Programming) ⁵优化包进行求解。一言以蔽之：在一定的约束条件下，目标最优，损失最小。

此外，由于这个问题的特殊结构，还可以通过拉格朗日对偶性（**Lagrange Duality**）变换到对偶变量（**dual variable**）的优化问题，即通过求解与原问题等价的⁶对偶问题（**dual problem**）得到原始问题的最优解，这就是线性可分条件下支持向量机的对偶算法，这样做的优点在于：一者对偶问题往往更容易求解；二者可以自然的引入核函数，进而推广到非线性分类问题。

那什么是拉格朗日对偶性呢？简单来讲，通过给每一个约束条件加上一个拉格朗日乘子（Lagrange multiplier） α ，定义拉格朗日函数（通过拉格朗日函数将约束条件融合到目标函数里去，从而只用一个函数表达式便能清楚的表达出我们的问题）：

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^n \alpha_i (y_i(w^T x_i + b) - 1) \quad (2.3)$$

然后令：

$$\theta(w) = \max_{\alpha_i \geq 0} \mathcal{L}(w, b, \alpha) \quad (2.4)$$

容易验证，当某个约束条件不满足时，例如 $y_i(w^T x_i + b) < 1$ ，那么显然有 $\theta(w) = \infty$ （只要令 $\alpha_i = \infty$ 即可）。而当所有约束条件都满足时，则有 $\theta(w) = \frac{1}{2}\|w\|^2$ ，亦即最初要最小化的量。

因此，在要求约束条件得到满足的情况下最小化 $\frac{1}{2}\|w\|^2$ ，实际上等价于直接最小化 $\theta(w)$ （当然，这里也有约束条件，就是 $\alpha_i \geq 0, i = 1, \dots, n$ ），因为如果约束条件没有得到满足， $\theta(w)$ 会等于无穷大，自然不会是我们所要求的最小值。

⁵http://en.wikipedia.org/wiki/Quadratic_programming

具体写出来，目标函数变成了：

$$\min_{w,b} \theta(w) = \min_{w,b} \max_{\alpha_i \geq 0} \mathcal{L}(w, b, \alpha) = p^* \quad (2.5)$$

这里用 p^* 表示这个问题的最优值，且和最初的问题是等价的。如果直接求解，那么一上来便得面对 w 和 b 两个参数，而 α_i 又是不等式约束，这个求解过程不好做。不妨把最小和最大的位置交换一下⁶，变成：

$$\max_{\alpha_i \geq 0} \min_{w,b} \mathcal{L}(w, b, \alpha) = d^* \quad (2.6)$$

交换以后的新问题是原始问题的对偶问题，这个新问题的最优值用 d^* 来表示。而且有 $d^* \leq p^*$ ，在满足某些条件的情况下，这两者相等，这个时候就可以通过求解对偶问题来间接地求解原始问题。

换言之，之所以从 $\min \max$ 的原始问题 p^* ，转化为 $\max \min$ 的对偶问题 d^* ，一者因为 d^* 是 p^* 的近似解，二者，转化为对偶问题后，更容易求解。

下面可以先求 \mathcal{L} 对 w 、 b 的极小，再求 \mathcal{L} 对 α 的极大。

2.1.2 K.K.T. 条件

上面提到“ $d^* \leq p^*$ 在满足某些条件的情况下，两者等价”，这所谓的“满足某些条件”就是要满足 KKT 条件。

一般地，一个最优化数学模型能够表示成下列标准形式：

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & h_j(x) = 0, \quad j = 1, \dots, p \\ & g_k(x) \leq 0, \quad k = 1, \dots, q \\ & x \in \mathcal{X} \subset \mathbb{R}^n \end{aligned} \quad (2.7)$$

其中， $f(x)$ 是需要最小化的函数， $h(x)$ 是等式约束， $g(x)$ 是不等式约束， p 和 q 分别为等式约束和不等式约束的数量。

同时，得明白以下两点：

- 凸优化的概念： $\mathcal{X} \subset \mathbb{R}^n$ 为一凸集， $f: \mathcal{X} \rightarrow \mathbb{R}$ 为一凸函数。凸优化就是要找出一
点 $x^* \in \mathcal{X}$ ，使得每一 $x \in \mathcal{X}$ 满足 $f(x^*) \leq f(x)$ 。
- KKT 条件的意义：它是一个非线性规划（Nonlinear Programming）问题能有
最优化解法的必要和充分条件。

⁶整理者注：关于对偶问题的最优等价性问题，参见 Stephen Boyd 的《Convex Optimization》中第五章

而 KKT 条件就是指上面最优化数学模型的标准形式中的最小点 x^* 必须满足下面的条件：

定义 2 (KKT 条件) 如式 2.7 的优化问题，满足：

$$1. h_j(x^*) = 0, j = 1, \dots, p, \quad g_k(x^*) \leq 0, k = 1, \dots, q$$

$$2. \Delta f(x^*) + \sum_{j=1}^p \lambda_j \Delta h_j(x^*) + \sum_{k=1}^q \mu_k \Delta g_k(x^*) = 0, \lambda_j \neq 0, \mu_k \geq 0, \mu_k g_k(x^*) = 0$$

则为满足 KKT 条件。

经过论证，我们这里的问题是满足 KKT 条件的（首先已经满足 **Slater condition**，再者 f 和 g_i 也都是可微的，即 \mathcal{L} 对 w 和 b 都可导），因此现在我们便转化为求解第二个问题。

也就是说，原始问题通过满足 KKT 条件，已经转化成了对偶问题。而求解这个对偶学习问题，分为 3 个步骤：首先要让 $\mathcal{L}(w, b, \alpha)$ 关于 w 和 b 最小化，然后求对 α 的极大，最后利用 **SMO 算法** 求解对偶问题中的拉格朗日乘子。

2.1.3 对偶问题求解的 3 个步骤

(1) 首先固定 α ，要让 \mathcal{L} 关于 w 和 b 最小化，我们分别对 w ， b 求偏导数，即令 $\partial \mathcal{L} / \partial w$ 和 $\partial \mathcal{L} / \partial b$ 等于零：

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^n \alpha_i y_i x_i \quad (2.8)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^n \alpha_i y_i = 0 \quad (2.9)$$

将以上结果代入之前的 \mathcal{L} ，得到：

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w^T x_i + b) - 1) \quad (2.10)$$

从而有（具体过程见注解4）：

$$\begin{aligned} \mathcal{L}(w, b, \alpha) &= \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j - \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j - b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j \end{aligned} \quad (2.11)$$

我们可以看出，此时的拉格朗日函数只包含了一个变量，那就是 α_i （ α_i 求出了便能求出 w 和 b ），由此可见，上文第 1.2 节提出来的核心问题：分类函数也就可以轻而易举的求出来了）。

(2) 求对 α 的极大，即是关于对偶问题的最优化问题。经过上面第一个步骤的求 w 和 b ，得到的拉格朗日函数式子已经没有了变量 w ， b ，只有 α 。从上面的式子得到：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & \alpha_i \geq 0, i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (2.12)$$

这样，求出了 α_i ，从而根据：

$$w^* = \sum_{i=1}^n \alpha_i y_i x_i \quad (2.13)$$

$$b^* = -\frac{\max_{i:y_i=-1} w^{*T} x_i + \min_{i:y_i=1} w^{*T} x_i}{2} \quad (2.14)$$

即可求出 w, b ，最终得出分离超平面和分类决策函数。

(3) 在求得 $\mathcal{L}(w, b, a)$ 关于 w 和 b 最小化，以及对 α 的极大之后，最后一步便是利用 SMO 算法求解对偶问题中的拉格朗日乘子 α 。

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & \alpha_i \geq 0, i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (2.15)$$

上述式子要解决的是在参数 $[\alpha_1, \alpha_2, \dots, \alpha_n]^T$ 上求最大值 W 的问题，至于 x_i 和 y_i 都是已知数。要了解这个 SMO 算法是如何推导的，请跳到下文第3.5节：SMO 算法。

到目前为止，我们的 SVM 还比较弱，只能处理线性的情况，下面我们将引入核函数，进而推广到非线性分类问题。

注 4 有读者可能会问公式 (2.17) 推导过程如何而来？说实话，其具体推导过程是比较

复杂的，如下：

$$\begin{aligned}
\mathcal{L}(w, b, \alpha) &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w^T x_i + b) - 1) \\
&= \frac{1}{2} w^T w - \sum_{i=1}^n \alpha_i y_i w^T x_i - \sum_{i=1}^n \alpha_i y_i b + \sum_{i=1}^n \alpha_i \\
&= \frac{1}{2} w^T \sum_{i=1}^n \alpha_i y_i x_i - \sum_{i=1}^n \alpha_i y_i w^T x_i - \sum_{i=1}^n \alpha_i y_i b + \sum_{i=1}^n \alpha_i \\
&= \frac{1}{2} w^T \sum_{i=1}^n \alpha_i y_i x_i - w^T \sum_{i=1}^n \alpha_i y_i x_i - \sum_{i=1}^n \alpha_i y_i b + \sum_{i=1}^n \alpha_i \\
&= -\frac{1}{2} w^T \sum_{i=1}^n \alpha_i y_i x_i - \sum_{i=1}^n \alpha_i y_i b + \sum_{i=1}^n \alpha_i \\
&= -\frac{1}{2} w^T \sum_{i=1}^n \alpha_i y_i x_i - b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i \\
&= -\frac{1}{2} \left(\sum_{i=1}^n \alpha_i y_i x_i \right)^T \sum_{i=1}^n \alpha_i y_i x_i - b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i \\
&= -\frac{1}{2} \sum_{i=1}^n \alpha_i y_i (x_i)^T \sum_{i=1}^n \alpha_i y_i x_i - b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i \\
&= -\frac{1}{2} \sum_{i,j=1}^n \alpha_i y_i (x_i)^T \alpha_j y_j x_j - b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i
\end{aligned} \tag{2.16}$$

最后，得到：

$$\begin{aligned}
\mathcal{L}(w, b, \alpha) &= \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j^T - \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j - b \sum_{i=1}^n \alpha_i y_i + \sum_{i=1}^n \alpha_i \\
&= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j
\end{aligned} \tag{2.17}$$

如 *JerryLead* 所说：“倒数第 4 步”推导到“倒数第 3 步”使用了线性代数的转置运算，由于 α_i 和 y_i 都是实数，因此转置后与自身一样。“倒数第 3 步”推导到“倒数第 2 步”使用了 $(a + b + c + \dots)(a + b + c + \dots) = aa + ab + ac + ba + bb + bc + \dots$ 的乘法运算法则。最后一步是上一步的顺序调整。

2.1.4 线性不可分的情况

OK，为过渡到下节 2.2 节所介绍的核函数，让我们再来看看上述推导过程中得到的一些有趣的形式。首先就是关于我们的 **hyper plane**，对于一个数据点 x 进行分类，

实际上是通过把 x 带入到 $f(x) = w^T x + b$ 算出结果然后根据其正负号来进行类别划分的。而前面的推导中我们得到：

$$w^* = \sum_{i=1}^n \alpha_i y_i x_i \quad (2.18)$$

因此分类函数为：

$$\begin{aligned} f(x) &= \left(\sum_{i=1}^n \alpha_i y_i x_i \right)^T x + b \\ &= \sum_{i=1}^n \alpha_i y_i \langle x_i, x \rangle + b \end{aligned} \quad (2.19)$$

这里的形式的有趣之处在于，对于新点 x 的预测，只需要计算它与训练数据点的内积即可（ $\langle \cdot, \cdot \rangle$ 表示向量内积），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 **Supporting Vector** 也在这里显示出来——事实上，所有非 Supporting Vector 所对应的系数 α 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“支持向量”而不是所有的训练数据即可。

为什么非支持向量对应的 α 等于零呢？直观上来理解的话，就是这些“后方”的点——正如我们之前分析过的一样，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。

回忆一下我们2.1.1节中通过 **Lagrange multiplier** 得到的目标函数：

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w^T x_i + b) - 1) \quad (2.20)$$

注意到如果 x_i 是支持向量的话，上式中红颜色的部分是等于 0 的（因为支持向量的 functional margin 等于 1），而对于非支持向量来说，functional margin 会大于 1，因此红颜色部分是大于零的，而 α_i 又是非负的，为了满足最大化， α_i 必须等于 0。这也就是这些非 Supporting Vector 的点的局限性。

从1.4节到上述所有这些东西，便得到了一个 **maximum margin hyper plane classifier**，这就是所谓的支持向量机（**Support Vector Machine**）。当然，到目前为止，我们的 SVM 还比较弱，只能处理线性的情况，不过，在得到了对偶 dual 形式之后，通过 **Kernel** 推广到非线性的情况就变成了一件非常容易的事情了（相信，你还记得本节开头所说的：“通过求解对偶问题得到最优解，这就是线性可分条件下支持向量机的对偶算法，这样做的优点在于：一者对偶问题往往更容易求解；二者可以自然的引入核函数，进而推广到非线性分类问题”）。

2.2 核函数：Kernel

2.2.1 特征空间的隐式映射：核函数

咱们首先给出核函数的来头：在上文中，我们已经了解到了 SVM 处理线性可分的情况，而对于非线性的情况，SVM 的处理方法是选择一个核函数 $K(\cdot, \cdot)$ ，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。

此外，因为训练样例一般是不会独立出现的，它们总是以成对样例的内积形式出现，而用对偶形式表示学习器的优势在为在该表示中可调参数的个数不依赖输入属性的个数，通过使用恰当的核函数来替代内积，可以隐式得将非线性的训练数据映射到高维空间，而不增加可调参数的个数（当然，前提是核函数能够计算对应着两个输入特征向量的内积）。

在线性不可分的情况下，支持向量机首先在低维空间中完成计算，然后通过核函数将输入空间映射到高维特征空间，最终在高维特征空间中构造出最优分离超平面，从而把平面上本身不好分的非线性数据分开。如图7所示，一堆数据在二维空间无法划分，从而映射到三维空间里划分。

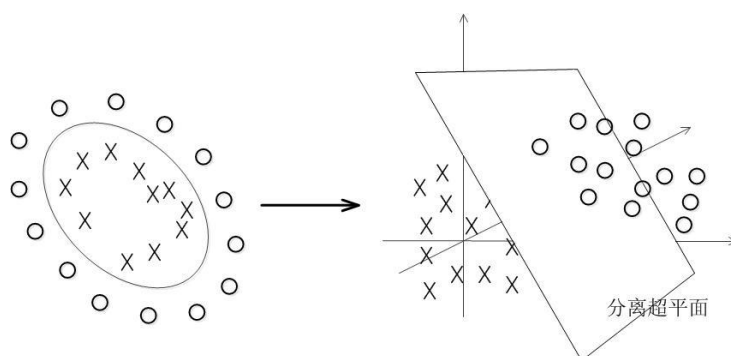


图 7: 核函数映射示意

而在我们遇到核函数之前，如果用原始的方法，那么在用线性学习器学习一个非线性关系，需要选择一个非线性特征集，并且将数据写成新的表达形式，这等价于应用一个固定的非线性映射，将数据映射到特征空间，在特征空间中使用线性学习器，因此，考虑的假设集是这种类型的函数：

$$f(x) = \sum_{i=1}^N w_i \phi_i(x) + b \quad (2.21)$$

这里 $\phi: \mathcal{X} \rightarrow \mathcal{F}$ 是从输入空间到某个特征空间的映射，这意味着建立非线性学习器分为两步：

1. 首先使用一个非线性映射将数据变换到一个特征空间 \mathcal{F} ；

2. 然后在特征空间使用线性学习器分类。

而由于对偶形式就是线性学习器的一个重要性质，这意味着假设可以表达为训练点的线性组合，因此决策规则可以用测试点和训练点的内积来表示：[这里采用的测试点和训练点放入](#)

$$f(x) = \sum_{i=1}^l \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b \quad (2.22)$$

如果有一种方式可以在特征空间中直接计算内积 $\langle \phi(x_i), \phi(x) \rangle$ ，就像在原始输入点的函数中一样，就有可能将两个步骤融合到一起建立一个非线性的学习器，这样直接计算的方法称为核函数方法：

定义 3 (核:Kernel) 核是一个函数 K ，对所有 $x, z \in \mathcal{X}$ ，满足 $K(x, z) = \langle \phi(x), \phi(z) \rangle$ ，这里 ϕ 是从 \mathcal{X} 到内积特征空间 \mathcal{F} 的映射。

2.2.2 核函数：如何处理非线性数据

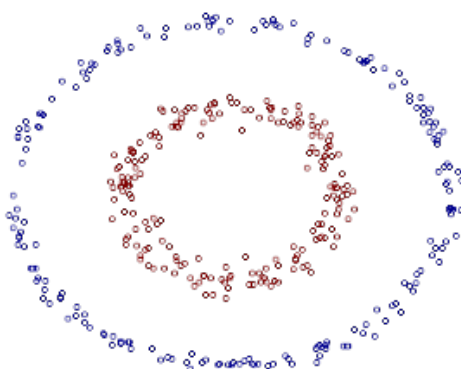


图 8: 线性不可分数据

来看个核函数的例子。如图8所示的两类数据，分别分布为两个圆圈形状，这样的数据本身就是线性不可分的，此时咱们该如何把这两类数据分开呢？

事实上，图8所述的这个数据集，是用两个半径不同的圆圈加上了少量的噪音生成得到的，所以，一个理想的分界应该是一个“圆圈”而不是一条线（超平面）。如果用 X_1 和 X_2 来表示这个二维平面的两个坐标的话，我们知道一条二次曲线（圆圈是二次曲线的一种特殊情况）的方程可以写作这样的形式：

$$a_1 X_1 + a_2 X_1^2 + a_3 X_2 + a_4 X_2^2 + a_5 X_1 X_2 + a_6 = 0 \quad (2.23)$$

注意上面的形式，如果我们构造另外一个五维的空间，其中五个坐标的值分别为 $Z_1 = X_1, Z_2 = X_1^2, Z_3 = X_2, Z_4 = X_2^2, Z_5 = X_1 X_2$ ，那么显然，上面的方程在新的坐标系

图 9: 高维空间旋转示意 (pluskid: gif 动画, 先用 Matlab 画出一张张图片, 再用 Imagemagick 拼贴成)

下可以写作:

$$\sum_{i=1}^5 a_i Z_i + a_6 = 0 \quad (2.24)$$

关于新的坐标 Z , 这正是一个 hyper plane 的方程! 也就是说, 如果我们做一个映射 $\phi: \mathbb{R}^2 \rightarrow \mathbb{R}^5$, 将 X 按照上面的规则映射为 Z , 那么在新的空间中原来的数据将变成线性可分的, 从而使用之前我们推导的线性分类算法就可以进行处理了。这正是 *Kernel* 方法处理非线性问题的基本思想。

再进一步描述 Kernel 的细节之前, 不妨再来看看这个例子映射过后的直观例子。当然, 你我可能无法把 5 维空间画出来, 不过由于我这里生成数据的时候就是用了特殊的情形, 具体来说, 我这里的超平面实际的方程是这个样子 (圆心在 X_2 轴上的一个正圆)⁷:

$$a_1 X_1^2 + a_2 (X_2 - c)^2 + a_3 = 0 \quad (2.25)$$

因此我只需要把它映射到 $Z_1 = X_1^2, Z_2 = X_2^2, Z_3 = X_2$ 这样一个三维空间中即可, 图9即是映射之后的结果, 将坐标轴经过适当的旋转, 就可以很明显地看出, 数据是可以通过一个平面来分开的。

核函数相当于把原来的分类函数:

$$f(x) = \sum_{i=1}^n \alpha_i y_i \langle x_i, x \rangle + b \quad (2.26)$$

⁷July 原文是: $\sum_{i=1}^5 a_i Z_i + a_6 = 0$, 有误, 参见 pluskid 博客: <http://blog.pluskid.org/?p=685>

映射成：

$$f(x) = \sum_{i=1}^n \alpha_i y_i \langle \phi(x_i), \phi(x) \rangle + b \quad (2.27)$$

而其中的 α 可以通过求解如下 dual 问题而得到的：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle \\ \text{s.t.} \quad & \alpha_i \geq 0, i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (2.28)$$

这样一来问题就解决了吗？似乎是的：拿到非线性数据，就找一个映射 $\phi(\cdot)$ ，然后一股脑把原来的数据映射到新空间中，再做线性 SVM 即可。不过事实上没有这么简单！其实刚才的方法稍想一下就会发现有问题：在最初的例子里，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，得到了五个维度；如果原始空间是三维，那么我们会得到 19 维的新空间，这个数目是呈爆炸性增长的，这给 $\phi(\cdot)$ 的计算带来了非常大的困难，而且如果遇到无穷维的情况，就根本无从计算了。所以就需要 Kernel 出马了。

不妨还是从最开始的简单例子出发，设两个向量 $x_1 = (\eta_1, \eta_2)^T$ 和 $x_2 = (\xi_1, \xi_2)^T$ ，而 $\phi(\cdot)$ 即是到前面说的五维空间的映射，因此映射过后的内积为：

$$\langle \phi(x_1), \phi(x_2) \rangle = \eta_1 \xi_1 + \eta_1^2 \xi_1^2 + \eta_2 \xi_2 + \eta_2^2 \xi_2^2 + \eta_1 \eta_2 \xi_1 \xi_2 \quad (2.29)$$

注 5 上面的这两个推导过程中，所说的前面的五维空间的映射，这里说的前面便是文中 2.2.1 节的所述的映射方式，回顾下之前的映射规则，再看那第一个推导，其实就是计算 x_1, x_2 各自的内积，然后相乘相加即可，第二个推导则是直接平方，去掉括号，也很容易推出来。

另外，我们又注意到：

$$(\langle x_1, x_2 \rangle + 1)^2 = 2\eta_1 \xi_1 + \eta_1^2 \xi_1^2 + 2\eta_2 \xi_2 + \eta_2^2 \xi_2^2 + 2\eta_1 \eta_2 \xi_1 \xi_2 + 1 \quad (2.30)$$

二者有很多相似的地方，实际上，我们只要把某几个维度线性缩放一下，然后再加上一个常数维度，具体来说，上面这个式子的计算结果实际上和映射

$$\phi(x_1, x_2) = (\sqrt{2}x_1, x_1^2, \sqrt{2}x_2, x_2^2, \sqrt{2}x_1x_2, 1)^T \quad (2.31)$$

之后的内积 $\langle \phi(x_1), \phi(x_2) \rangle$ 的结果是相等的，那么区别在于什么地方呢？

1. 一个是映射到高维空间中，然后再根据内积的公式进行计算；
2. 而另一个则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。

注 6 上面之中，最后的两个式子，第一个算式，是带内积的完全平方方式，可以拆开，然后，通过凑一个得到，第二个算式，也是根据第一个算式凑出来的)

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。

我们把这里的计算两个向量在隐式映射过后的空间中的内积的函数叫做核函数 (Kernel Function)，例如，在刚才的例子中，我们的核函数为：

$$K(x_1, x_2) = (\langle x_1, x_2 \rangle + 1)^2 \quad (2.32)$$

核函数能简化映射空间中的内积运算——刚好“碰巧”的是，在我们的 SVM 里需要计算的地方数据向量总是以内积的形式出现的。对比刚才我们上面写出来的式子，现在在我们的分类函数为：

$$f(x) = \sum_{i=1}^n \alpha_i y_i K(x_i, x) + b \quad (2.33)$$

其中由如下 dual 问题计算而得：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{s.t.} \quad & \alpha_i \geq 0, i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (2.34)$$

这样一来计算的问题就算解决了，避开了直接在高维空间中进行计算，而结果却是等价的！当然，因为我们这里的例子非常简单，所以我可以手工构造出对应于 $\phi(\cdot)$ 的核函数出来，如果对于任意一个映射，想要构造出对应的核函数就很困难了。

2.2.3 几个核函数

通常人们会从一些常用的核函数中选择（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），例如：

- 多项式核 $K(x_1, x_2) = (\langle x_1, x_2 \rangle + R)^d$ ，显然刚才我们举的例子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，该空间的维度是 C_{m+d}^d ，其中 m 是原始空间的维度。

- 高斯核 $K(x_1, x_2) = \exp(-\|x_1 - x_2\|^2 / 2\sigma^2)$ ，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 选得很大的话，高次特征上的权重实际上衰减得非常快，所以实际上（数值上近似一下）相当于一个低维的子空间；反过来，如果 σ 选得很小，则可以将任意的数据映射为线性可分——当然，这并不一定是好事，因为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。图10所示的例子便是把低维线性不可分的数据通过高斯核函数映射到了高维空间：

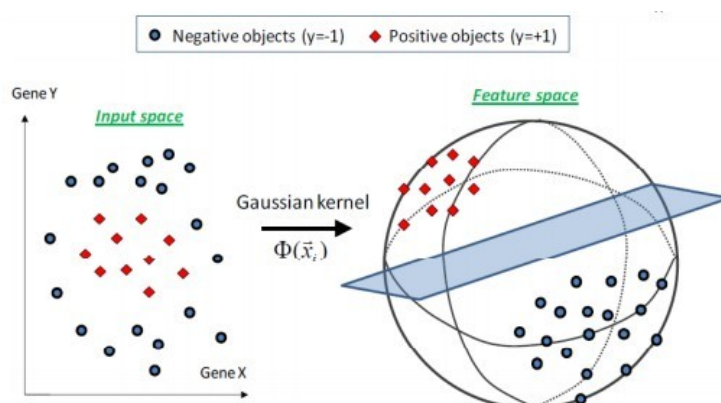


图 10: 高斯核示意

- 线性核 $K(x_1, x_2) = \langle x_1, x_2 \rangle$ ，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（意思是说，咱们有的时候，写代码，或写公式的时候，只要写个模板或通用表达式，然后再代入不同的核，便可以了，于此，便在形式上统一了起来，不用再分别写一个线性的，和一个非线性的）。

2.2.4 核函数的本质

上面说了这么一大堆，读者可能还是没明白核函数到底是个什么东西？我再简要概括下，即以下三点：

1. 实际中，我们会经常遇到线性不可分的样例，此时，我们的常用做法是把样例特征映射到高维空间中去（如上文2.2.2节最开始的图9所示，映射到高维空间后，相关特征便被分开了，也就达到了分类的目的）；
2. 但进一步，如果凡是遇到线性不可分的样例，一律映射到高维空间，那么这个维度大小是会高到可怕的（如上文中 19 维乃至无穷维的例子）。那咋办呢？

3. 此时，核函数就隆重登场了，核函数的价值在于它虽然也是讲特征进行从低维到高维的转换，但核函数绝就绝在它事先在低维上进行计算，而将实质上的分类效果表现在了高维上，也就如上文所说的避免了直接在高维空间中的复杂计算。

最后引用这里⁸的一个例子举例说明下核函数解决非线性问题的直观效果。

假设现在你是一个农场主，圈养了一批羊群，但为预防狼群袭击羊群，你需要搭建一个篱笆来把羊群围起来。但是篱笆应该建在哪里呢？你很可能需要依据牛群和狼群的位置建立一个“分类器”，比较图11这几种不同的分类器，我们可以看到 SVM 完成了一个很完美的解决方案。

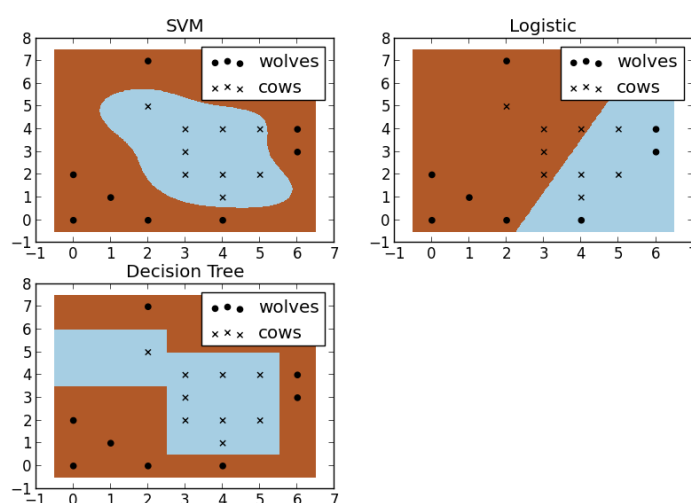


图 11: 分类器例子：圈养

这个例子从侧面简单说明了 SVM 使用非线性分类器的优势，而 Logistic 模式以及决策树模式都是使用了直线方法。

OK，不再做过多介绍了，对核函数有进一步兴趣的，[还可以看看此文⁹](http://www.yaksis.com/posts/why-use-svm.html)。

2.3 使用松弛变量处理 outliers 方法

在本文第一节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，在上文2.2.2节使用 Kernel 方法对原来的线性 SVM 进行了推广，使得非线性的情况也能处理。虽然通过映射 $\phi(\cdot)$ 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。

⁸<http://www.yaksis.com/posts/why-use-svm.html>

⁹<http://www.cnblogs.com/vivounicorn/archive/2010/12/13/1904720.html>

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声。对于这种偏离正常位置很远的数据点，我们称之为 **outlier**，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。例如图12。

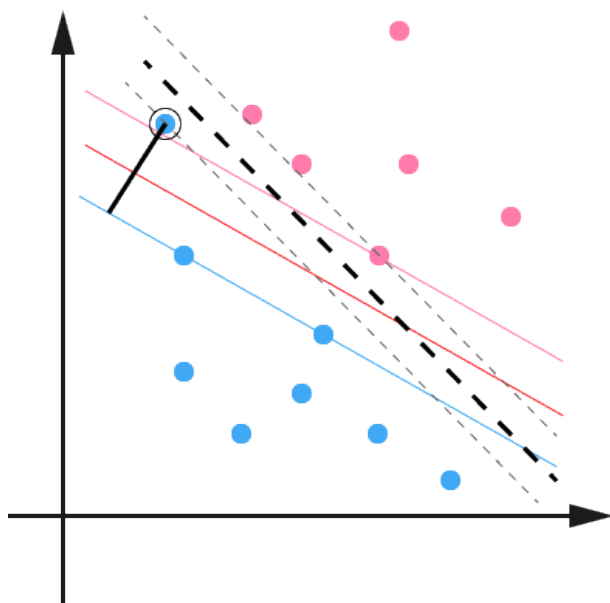


图 12: SVM 中的 outlier

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，变成途中黑色虚线所示（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。例如图中12，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。

注 7 插播下一位读者 @Copper_PKU 的理解：换言之，在有松弛的情况下 outlier 点也属于支持向量 SV，同时，对于不同的支持向量，拉格朗日参数的值也不同，如此篇论文《Large Scale Machine Learning》中，本文中的图13所示。

对于远离分类平面的点值为 0；对于边缘上的点值在 $[0, 1/L]$ 之间，其中， L 为训练数据集个数，即数据集大小；对于 outlier 数据和内部的数据值为 $1/L$ 。更多请参看本文文末参考条目第 51 条。

OK，继续回到咱们的问题。我们，原来的约束条件为：

$$y_i(w^T x_i + b) \geq 1, \quad i = 1, \dots, n \quad (2.35)$$

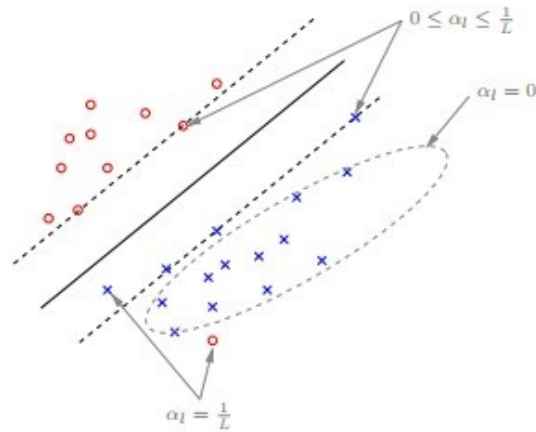


图 13: SVM 中的 outlier 所对应的 Lagrange 乘子

现在考虑到 outlier 问题，约束条件变成了：

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad i = 1, \dots, n \quad (2.36)$$

其中 ξ_i 称为松弛变量 (slack variable)，对应数据点 x_i 允许偏离 functional margin 的量。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：

$$\min \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad (2.37)$$

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 ξ 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：

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, \dots, n \\ & \xi_i \geq 0, i = 1, \dots, n \end{aligned} \quad (2.38)$$

用之前的方法将限制或约束条件加入到目标函数中，得到新的拉格朗日函数，如下所示：

$$\mathcal{L}(w, b, \xi, \alpha, r) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i (y_i(w^T x_i + b) - 1 + \xi_i) - \sum_{i=1}^n r_i \xi_i \quad (2.39)$$

分析方法和前面一样，转换为另一个问题之后，我们先让 \mathcal{L} 针对 w 、 b 和 ξ 最小化：

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^n \alpha_i y_i x_i \quad (2.40)$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{i=1}^n \alpha_i y_i = 0 \quad (2.41)$$

$$\frac{\partial L}{\partial \xi_i} = 0 \Rightarrow C - \alpha_i - r_i = 0, \quad i = 1, \dots, n \quad (2.42)$$

将 w 带回 \mathcal{L} 并化简，得到和原来一样的目标函数：

$$\max_{\alpha} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j \quad (2.43)$$

不过，由于我们得到 $C - \alpha_i - r_i = 0$ 而又有 $r_i \geq 0$ （作为 Lagrange multiplier 的条件），因此有 $\alpha_i \leq C$ ，所以整个 dual 问题现在写作：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (2.44)$$

Primal formulation:

$$\text{Minimize } \underbrace{\frac{1}{2} \sum_{i=1}^n w_i^2 + C \sum_{i=1}^N \xi_i}_{\text{Objective function}} \text{ subject to } \underbrace{y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 - \xi_i}_{\text{Constraints}} \text{ for } i = 1, \dots, N$$

Dual formulation:

$$\text{Maximize } \underbrace{\sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j}_{\text{Objective function}} \text{ subject to } \underbrace{0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^N \alpha_i y_i = 0}_{\text{Constraints}} \text{ for } i = 1, \dots, N.$$

图 14: SVM 中原问题与对偶问题对比

把前后的结果对比一下，如图14。可以看到唯一的区别就是现在 dual variable α 多了一个上限 C 。而 Kernel 化的非线性形式也是一样的，只要把 $\langle x_i, x_j \rangle$ 换成 $K(x_i, x_j)$ 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声和 outliers 的支持向量机才终于介绍完毕了。

行文至此，可以做个小结，不准确的说，SVM 它本质上即是一个分类方法，用 $w^T + b$ 定义分类函数，于是求 w 、 b ，为寻最大间隔，引出 $1/2\|w\|^2$ ，继而引入拉格朗日

因子，化为对拉格朗日乘子 α 的求解（求解过程中会涉及到一系列最优化或凸二次规划等问题），如此，求 w 、 b 与求 α 等价，而 α 的求解可以用一种快速学习算法 SMO，至于核函数，是为处理非线性情况，若直接映射到高维计算恐维度爆炸，故在低维计算，等效高维表现。

OK，理解到这第二层，已经能满足绝大部分人一窥 SVM 原理的好奇心，然对于那些想在证明层面理解 SVM 的则还很不够，但进入第三层理解境界之前，你必须要有比较好的数理基础和逻辑证明能力，不然你会跟我一样，吃不少苦头的。

3 证明 SVM

说 实话，凡是涉及到要证明的东西，理论，便一般不是怎么好惹的东西。绝大部分时候，看懂一个东西不难，但证明一个东西则需要点数学功底，进一步，证明一个东西也不是特别难，难的是从零开始发明创造这个东西的时候，则显艰难（因为任何时代，大部分人的研究所得都不过是基于前人的研究成果，前人所做的是开创性工作，而这往往是最艰难最有价值的，他们被称为真正的先驱。牛顿也曾说过，他不过是站在巨人的肩上。你，我则更是如此）。

正如陈希孺院士在他的著作《数理统计学简史》的第4章、最小二乘法中所讲：在科研上诸多观念的革新和突破是有着很多的不易的，或许某个定理在某个时期由某个人点破了，现在的我们看来一切都是理所当然，但在一切没有发现之前，可能许许多多的顶级学者毕其功于一役，耗尽一生，努力了几十年最终也是无功而返。

话休絮烦，要证明一个东西先要弄清楚它的根基在哪，即构成它的基础是哪些理论。OK，以下内容基本是上文中未讲到的一些定理的证明，包括其背后的逻辑、来源背景等东西，还是读书笔记。

本部分导述

3.1节，线性学习器，主要阐述感知机算法；

3.2节，非线性学习器，主要阐述 mercer 定理；

3.3节，损失函数；

3.4节，最小二乘法；

3.5节，SMO 算法；

3.6节，简略谈谈 SVM 的应用；

3.1 线性学习器

3.1.1 感知器算法

这个感知机算法是 1956 年提出的，年代久远，依然影响着当今，当然，可以肯定的是，此算法亦非最优，后续会有更详尽阐述。不过，有一点，你必须清楚，这个算法是为了干嘛的：不断的训练试错以期寻找一个合适的超平面（是的，就这么简单）。

下面，举个例子。如图15所示，凭我们的直觉可以看出，图中的红线是最优超平面，蓝线则是根据感知机算法在不断的训练中，最终，若蓝线能通过不断的训练移动到红线位置上，则代表训练成功。

算法 1: 感知器算法（原始形式）**Data:** 给定线性可分的数据集 $S = \{x_i\}, i = 1, \dots, l$ 和学习速率 $\eta \in \mathbb{R}^+$ **Result:** 超平面参数 w 和 b

```

1  $w_0 \leftarrow 0; b_0 \leftarrow 0; k \leftarrow 0$ 
2  $R \leftarrow \max_{1 \leq i \leq l} \|x_i\|$ 
3 while 在  $for$  循环中没有错误发生 do
4   for  $i = 1$  to  $l$  do
5     if  $y_i(\langle w_k, x_i \rangle + b_k) \leq 0$  then
6        $w_{k+1} \leftarrow w_k + \eta y_i x_i$ 
7        $b_{k+1} \leftarrow b_k + \eta y_i R^2$ 
8        $k \leftarrow k + 1$ 
9     end
10  end
11 end
12 返回  $(w_k, b_k)$ , 这里  $k$  是错误次数

```

既然需要通过不断的训练以让蓝线最终成为最优分类超平面，那么，到底需要训练多少次呢？Novikoff 定理告诉我们当间隔是正的时候感知机算法会在有限次数的迭代中收敛，也就是说 Novikoff 定理证明了感知机算法的收敛性，即能得到一个界，不至于无穷循环下去。

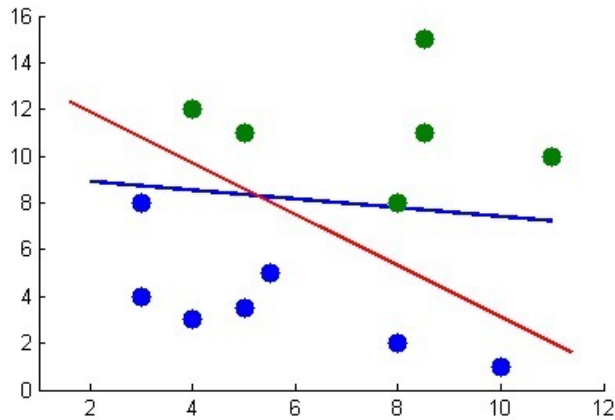


图 15: 感知器所得超平面

定理 1 (Novikoff 定理) 如果分类超平面存在，仅需在序列 S 上迭代几次，在界为 $(\frac{2R}{\gamma})^2$ 的错误次数下就可以找到分类超平面，算法停止。

这里, $R = \max_{1 \leq i \leq l} \|x_i\|$, γ 为扩充间隔。根据误分次数公式可知, 迭代次数与对应于扩充 (包括偏置) 权重的训练集的间隔有关。

顺便再解释下这个所谓的扩充间隔 γ , γ 即为样本到分类间隔的距离, 即从 γ 引出的最大分类间隔。OK, 还记得上文第1.3节开头的内容么? 如图16。

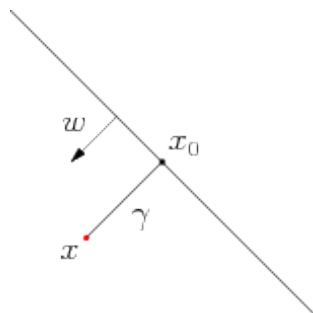


图 16: 几何距离示例

在给出几何间隔的定义之前, 咱们首先来看下, 如图16所示, 对于一个点 x , 令其垂直投影到超平面上的对应的为 x_0 , 由于 w 是垂直于超平面的一个向量, γ 为样本 x 到分类间隔的距离, 我们有:

$$x = x_0 + \gamma \frac{w}{\|w\|} \quad (3.1)$$

然后后续怎么推导出最大分类间隔请回到本文第一、二部分, 此处不重复板书。

同时有一点得注意: 感知机算法虽然可以通过简单迭代对线性可分数据生成正确分类的超平面, 但不是最优效果, 那怎样才能得到最优效果呢, 就是上文中第一部分所讲的寻找最大分类间隔超平面。此外, Novikoff 定理的证明请见这里¹⁰。

3.2 非线性学习器

3.2.1 Mercer 定理

定理 2 (Mercer 定理) 如果函数 K 是 $\mathcal{X}^n \times \mathcal{X}^n \rightarrow \mathcal{R}$ 上的映射 (也就是从两个 n 维向量映射到实数域)。那么如果 K 是一个有效核函数 (也称为 Mercer 核函数), 那么当且仅当对于训练样例 $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$, 其相应的核函数矩阵是对称半正定的。

要理解这个 Mercer 定理, 先要了解什么是半正定矩阵, 要了解什么是半正定矩阵, 先得知道什么是正定矩阵¹¹ (矩阵理论 “博大精深”, 我自己也未能彻底理清, 等我理

¹⁰<http://www.cs.columbia.edu/~mcollins/courses/6998-2012/notes/perc.converge.pdf>

¹¹http://en.wikipedia.org/wiki/Positive-definite_matrix

清了再续写此节，顺便推荐我正在看的一本《矩阵分析与应用》）。然后这里¹²有一个此定理的证明，可以看下。

正如 @Copper_PKU 所说：核函数在 SVM 的分类效果中起了重要的作用，最后这里有个 tutorial¹³可以看看。

3.3 损失函数

在本文第1节有这么一句话“支持向量机 (SVM) 是 90 年代中期发展起来的基于统计学习理论的一种机器学习方法，通过寻求结构化风险最小来提高学习机泛化能力，实现经验风险和置信范围的最小化，从而达到在统计样本量较少的情况下，亦能获得良好统计规律的目的。”但初次看到的读者可能并不了解什么是结构化风险，什么又是经验风险。要了解这两个所谓的“风险”，还得又从监督学习说起。

监督学习实际上就是一个经验风险或者结构风险函数的最优化问题。风险函数度量平均意义下模型预测的好坏，模型每一次预测的好坏用损失函数来度量。它从假设空间 \mathcal{F} 中选择模型 f 作为决策函数，对于给定的输入 X ，由 $f(X)$ 给出相应的输出 Y ，这个输出的预测值 $f(X)$ 与真实值 Y 可能一致也可能不一致，用一个损失函数来度量预测错误的程度。损失函数记为 $L(Y, f(X))$ 。

常用的损失函数有以下几种（基本引用自《统计学习方法》）¹⁴：

1. 0-1 损失函数

$$L(Y, f(X)) = \begin{cases} 1 & Y \neq f(X) \\ 0 & Y = f(X) \end{cases} \quad (3.2)$$

2. 平方损失函数

$$L(Y, f(X)) = (Y - f(X))^2 \quad (3.3)$$

3. 绝对损失函数

$$L(Y, f(X)) = |Y - f(X)| \quad (3.4)$$

4. 对数损失函数

$$L(Y, P(Y|X)) = -\log P(Y|X) \quad (3.5)$$

给定一组训练数据：

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_i, y_i), \dots, (x_N, y_N)\} \quad (3.6)$$

¹²此链接已经失效：<http://ftp136343.host106.web522.com/a/biancheng/matlab/2013/0120/648.html>

¹³<http://www.cs.berkeley.edu/~bartlett/courses/281b-sp08/7.pdf>

¹⁴整理者注：SVM 中的损失函数为 Hinge Loss。

模型 $f(x)$ 关于训练数据集的平均损失称为经验风险，如下：

$$R_{emp}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) \quad (3.7)$$

关于如何选择模型，监督学习有两种策略：经验风险最小化和结构风险最小化。

经验风险最小化的策略认为，经验风险最小的模型就是最优模型，则按照经验风险最小化求最优模型就是求解如下最优化问题：

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) \quad (3.8)$$

当样本容量很小时，经验风险最小化的策略容易产生过拟合的现象。结构风险最小化可以防止过拟合。结构风险是在经验风险的基础上加上表示模型复杂度的正则化项或者罚项，结构风险定义如下：

$$R_{srm}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f) \quad (3.9)$$

其中， $J(f)$ 为模型的复杂度，模型 f 越复杂， $J(f)$ 的值就越大，模型越简单， $J(f)$ 的值就越小，也就是说 $J(f)$ 是对复杂模型的惩罚。 $\lambda \geq 0$ 是系数，用以权衡经验风险和模型复杂度。结构风险最小化的策略认为结构风险最小的模型是最优的模型，所以求最优的模型就是求解下面的最优化问题：

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f) \quad (3.10)$$

这样，监督学习问题就变成了经验风险或结构风险函数的最优化问题，如式 (3.8) 和 (3.10)。

如此，SVM 有第二种理解，即最优化 + 损失最小，或如 @ 夏粉 _ 百度所说“可从损失函数和优化算法角度看 SVM，boosting，LR 等算法，可能会有不同收获”。

OK，关于更多统计学习方法的问题，请参看此文¹⁵。

关于损失函数，如下文读者评论中所述：可以看看张潼的这篇《Statistical behavior and consistency of classification methods based on convex risk minimization》。各种算法中常用的损失函数基本都具有 Fisher 一致性¹⁶，优化这些损失函数得到的分类器可以看作是后验概率的“代理”。此外，张潼还有另外一篇论文《Statistical analysis of some multi-category large margin classification methods》，在多分类情况下 margin loss 的分析，这两篇对 Boosting 和 SVM 使用的损失函数分析的很透彻。

¹⁵<http://blog.csdn.net/ql1125596718/article/details/8351337>

¹⁶整理者注：http://en.wikipedia.org/wiki/Fisher_consistency

3.4 最小二乘法

3.4.1 什么是最小二乘法？

既然本节开始之前提到了最小二乘法，那么下面引用《正态分布的前世今生》里的内容稍微简单阐述下。

我们口头中经常说：一般来说，平均来说。如平均来说，不吸烟的健康优于吸烟者，之所以要加“平均”二字，是因为凡事皆有例外，总存在某个特别的人他吸烟但由于经常锻炼所以他的健康状况可能会优于他身边不吸烟的朋友。而最小二乘法的一个最简单的例子便是算术平均。

定义 4 (最小二乘法) 最小二乘法（又称最小平方法）是一种数学优化技术。它通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。用函数表示为：

$$\min_{\vec{x}} \sum_{i=1} (y_m - y_i)^2 \quad (3.11)$$

使误差¹⁷平方和达到最小以寻求估计值的方法，就叫做最小二乘法，用最小二乘法得到的估计，叫做最小二乘估计。当然，取平方和作为目标函数只是众多可取的方法之一。

最小二乘法的一般形式可表示为：

$$\min_{\vec{x}} \|\vec{y}_m(\vec{x}) - \vec{y}\|_2 \quad (3.12)$$

有效的最小二乘法是勒让德在 1805 年发表的，基本思想就是认为测量中有误差，所以所有方程的累积误差为：累积误差 = $\sum(\text{观测值} - \text{理论值})^2$ 。

我们求解出导致累积误差最小的参数即可：

$$\begin{aligned} \hat{\beta} &= \arg \min_{\beta} \sum_{i=1}^n e_i^2 \\ &= \arg \min_{\beta} \sum_{i=1}^n [y_i - (\beta_0 + \beta_1 x_{1i} + \dots + \beta_p x_{pi})]^2 \end{aligned} \quad (3.13)$$

勒让德在论文中对最小二乘法的优良性做了几点说明：

1. 最小二乘使得误差平方和最小，并在各个方程的误差之间建立了一种平衡，从而防止某一个极端误差取得支配地位；
2. 计算中只要求偏导后求解线性方程组，计算过程明确便捷；

¹⁷所谓误差，当然是观察值与实际真实值的差量

3. 最小二乘可以导出算术平均值作为估计值。

对于最后一点，从统计学的角度来看是很重要的一个性质。推理如下：假设真值为 θ ， x_1, \dots, x_n 为 n 次测量值，每次测量的误差为 $e_i = x_i - \theta$ ，按最小二乘法，误差累积为：

$$L(\theta) = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (x_i - \theta)^2 \quad (3.14)$$

求解 θ 使 $L(\theta)$ 达到最小，正好是算术平均 $\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$ 。

由于算术平均是一个历经考验的方法，而以上的推理说明，算术平均是最小二乘的一个特例，所以从另一个角度说明了最小二乘方法的优良性，使我们对最小二乘法更加有信心。

最小二乘法发表之后很快得到了大家的认可接受，并迅速的在数据分析实践中被广泛使用。不过历史上又有人把最小二乘法的发明归功于高斯，这又是怎么回事呢？高斯在 1809 年也发表了最小二乘法，并且声称自己已经使用这个方法多年。高斯发明了小行星定位的数学方法，并在数据分析中使用最小二乘方法进行计算，准确的预测了谷神星的位置。

说了这么多，貌似跟本文的主题 SVM 没啥关系呀，别急，请让我继续阐述。本质上说，最小二乘法即是一种参数估计方法，说到参数估计，咱们得从一元线性模型说起。

3.4.2 最小二乘法的解法

什么是一元线性模型呢？请允许我引用这里¹⁸的内容，先来梳理下几个基本概念：

1. 监督学习中，如果预测的变量是离散的，我们称其为分类（如决策树，支持向量机等¹⁹），如果预测的变量是连续的，我们称其为回归。
2. 回归分析中，如果只包括一个自变量和一个因变量，且二者的关系可用一条直线近似表示，这种回归分析称为一元线性回归分析。
3. 如果回归分析中包括两个或两个以上的自变量，且因变量和自变量之间是线性关系，则称为多元线性回归分析。
4. 对于二维空间线性是一条直线；对于三维空间线性是一个平面，对于多维空间线性是一个超平面。

¹⁸<http://blog.csdn.net/q11125596718/article/details/8248249>

¹⁹整理者注：支持向量机也能用于回归，如 SVR。

对于一元线性回归模型, 假设从总体中获取了 n 组观察值 $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ 。对于平面中的这 n 个点, 可以使用无数条曲线来拟合。要求样本回归函数尽可能好地拟合这组值。综合起来看, 这条直线处于样本数据的中心位置最合理。

选择最佳拟合曲线的标准可以确定为: 使总的拟合误差 (即总残差) 达到最小。有以下三个标准可以选择:

1. 用“残差和最小”确定直线位置是一个途径。但很快发现计算“残差和”存在相互抵消的问题。
2. 用“残差绝对值和最小”确定直线位置也是一个途径。但绝对值的计算比较麻烦。
3. 最小二乘法的原则是以“残差平方和最小”确定直线位置。用最小二乘法除了计算比较方便外, 得到的估计量还具有优良特性。这种方法对异常值非常敏感。

最常用的是普通最小二乘法 (Ordinary Least Square, OLS): 所选择的回归模型应该使所有观察值的残差平方和达到最小, 即采用平方损失函数。

我们定义样本回归模型为:

$$Y_i = \hat{\beta}_0 + \hat{\beta}_1 X_i + e_i \quad (3.15)$$

$$\Rightarrow e_i = Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i \quad (3.16)$$

其中, e_i 为样本 (X_i, Y_i) 的误差。

接着, 定义平方损失函数 Q :

$$Q = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^n (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)^2 \quad (3.17)$$

则通过 Q 最小确定这条直线, 即确定 $\hat{\beta}_0, \hat{\beta}_1$, 以为 $\hat{\beta}_0, \hat{\beta}_1$ 变量, 把它们看作是 Q 的函数, 就变成了一个求极值的问题, 可以通过求导数得到。

求 Q 对两个待估参数的偏导数:

$$\frac{\partial Q}{\partial \hat{\beta}_0} = 2 \sum_{i=1}^n (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)(-1) = 0 \quad (3.18)$$

$$\frac{\partial Q}{\partial \hat{\beta}_1} = 2 \sum_{i=1}^n (Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_i)(-X_i) = 0 \quad (3.19)$$

根据数学知识我们知道, 函数的极值点为偏导为 0 的点, 解得:

$$\hat{\beta}_1 = \frac{n \sum X_i Y_i - \sum X_i \sum Y_i}{n \sum X_i^2 - (\sum X_i)^2} \quad (3.20)$$

$$\hat{\beta}_0 = \frac{\sum X_i^2 \sum Y_i - \sum X_i \sum X_i Y_i}{n \sum X_i^2 - (\sum X_i)^2} \quad (3.21)$$

这就是最小二乘法的解法，就是求得平方损失函数的极值点。自此，你看到求解最小二乘法与求解 SVM 问题何等相似，尤其是定义损失函数，而后通过偏导求得极值。

OK，更多请参看陈希孺院士的《数理统计学简史》的第 4 章、最小二乘法。

3.5 SMO 算法

在上文中，我们提到了求解对偶问题的序列最小最优化 SMO 算法，但并未提到其具体解法。首先看下最后悬而未决的问题：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (3.22)$$

等价于求解：

$$\begin{aligned} \min_{\alpha} \Psi(\vec{\alpha}) = \min_{\alpha} \quad & \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j K(x_i, x_j) \alpha_i \alpha_j - \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (3.23)$$

1998 年，Microsoft Research 的 John C. Platt 在论文《Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines》中提出针对上述问题的解法：SMO 算法，它很快便成为最快的二次规划优化算法，特别是在针对线性 SVM 和数据稀疏时性能更优。

接下来，咱们便参考 John C. Platt 的这篇文章来看看 SMO 的解法是怎样的。

3.5.1 SMO 算法的推导

咱们首先来定义特征到结果的输出函数：

$$u = \vec{w} \cdot \vec{x} - b \quad (3.24)$$

注 8 这个 u 与我们之前定义的 $f(x) = w^T x + b$ 的实质是一样的。

接着，重新定义下咱们的原始优化问题，权当重新回顾，如下：

$$\min_{w,b} \frac{1}{2} \|\vec{w}\|^2 \quad \text{subject to} \quad y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \forall i \quad (3.25)$$

求导得到：

$$\vec{w} = \sum_{i=1}^N y_i \alpha_i \vec{x}_i, \quad b = \vec{w} \cdot \vec{x}_k - y_k, \text{ for some } \alpha_k > 0 \quad (3.26)$$

代入 $u = \vec{w} \cdot \vec{x} - b$ 中，可得 $u = \sum_{j=1}^N y_j \alpha_j K(\vec{x}_j, \vec{x}) - b$ 。

通过引入 Lagrange 乘子转化为对偶问题后，得：

$$\begin{aligned} \min_{\alpha} \Psi(\vec{\alpha}) &= \min_{\alpha} \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j \langle x_i, x_j \rangle \alpha_i \alpha_j - \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad &\alpha_i \geq 0, i = 1, \dots, n \\ &\sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (3.27)$$

注 9 这里得到的 \min 函数与我们之前的 \max 函数实质也是一样，因为把符号变下，即由 \min 转化为 \max 的问题，且 y_i 也与之前的 $y^{(i)}$ 等价， y_j 亦如此。

经过加入松弛变量后，模型修改为：

$$\begin{aligned} \min_{w,b,\xi} \frac{1}{2} \|\vec{w}\|^2 + C \sum_{i=1}^n \xi_i \quad \text{subject to} \quad &y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 - \xi_i, \forall i \\ &0 \leq \alpha_i \leq C, \forall i \end{aligned} \quad (3.28)$$

从而，最终我们的问题变为：

$$\begin{aligned} \min_{\alpha} \Psi(\vec{\alpha}) &= \min_{\alpha} \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n y_i y_j K(x_i, x_j) \alpha_i \alpha_j - \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad &0 \leq \alpha_i \leq C, i = 1, \dots, n \\ &\sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (3.29)$$

下面要解决的问题是：在 $\alpha_i = \{\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n\}$ 上求上述目标函数的最小值。为了求解这些乘子，每次从中任意抽取两个乘子 α_1 和 α_2 ，然后固定 α_1 和 α_2 以外的其它乘子 $\{\alpha_3, \dots, \alpha_n\}$ ，使得目标函数只是关于 α_1 和 α_2 的函数。这样，不断的从一堆乘子中任意抽取两个求解，不断的迭代求解子问题，最终达到求解原问题的目的。

而原对偶问题的子问题的目标函数可以表达为：

$$\Psi = \frac{1}{2}K_{11}\alpha_1^2 + \frac{1}{2}K_{22}\alpha_2^2 + sK_{12}\alpha_1\alpha_2 + y_1\alpha_1v_1 + y_2\alpha_2v_2 - \alpha_1 - \alpha_2 + \Psi_{constant} \quad (3.30)$$

其中：

$$K_{ij} = K(\vec{x}_i, \vec{x}_j), \quad (3.31)$$

$$v_i = \sum_{j=1}^n y_j \alpha_j^* K_{ij} = u_i + b^* - y_1 \alpha_1^* K_{1i} - y_2 \alpha_2^* K_{2i} \quad (3.32)$$

为了解决这个子问题，首要问题便是每次如何选取 α_1 和 α_2 。实际上，其中一个乘子是违法 KKT 条件最严重的，另外一个乘子则由另一个约束条件选取。

根据 KKT 条件可以得出目标函数中取值 α_i 的意义：

$$\alpha_i = 0 \Leftrightarrow y_i u_i \geq 1, \quad (3.33)$$

$$0 < \alpha_i < C \Leftrightarrow y_i u_i = 1, \quad (3.34)$$

$$\alpha_i = C \Leftrightarrow y_i u_i \leq 1. \quad (3.35)$$

这里的 α_i 还是 Larange 乘子：

1. 对于第 1 种情况，表明 α_i 是正常分类，在边界内部（我们知道正确分类的点 $y_i f(x_i) \geq 0$ ）；
2. 对于第 2 种情况，表明了 α_i 是支持向量，在边界上；
3. 对于第 3 种情况，表明了 α_i 是在两条边界之间。

而最优解需要满足 KKT 条件，即上述 3 个条件都得满足，以下几种情况出现将会出现不满足：

- $y_i u_i \leq 1$ 但是 $\alpha_i < C$ 则是不满足的，而原本 $\alpha_i = C$
- $y_i u_i \geq 1$ 但是 $\alpha_i > 0$ 则是不满足的，而原本 $\alpha_i = 0$
- $y_i u_i = 1$ 但是 $\alpha_i = 0$ 或者 $\alpha_i = C$ 则表明不满足的，而原本应该是 $0 < \alpha_i < C$

也就是说，如果存在不满足 KKT 条件的 α_i ，那么需要更新这些 α_i ，这是第一个约束条件。此外，更新的同时还要受到第二个约束条件的限制，即 $\sum_{i=1}^m \alpha_i y_i = 0$ 。

因此，如果假设选择的两个乘子 α_1 和 α_2 ，它们在更新之前分别是 α_1^{old} 、 α_2^{old} ，更新之后分别是 α_1^{new} 、 α_2^{new} ，那么更新前后的值需要满足以下等式才能保证和为 0 的约束：

$$\alpha_1^{new} y_1 + \alpha_2^{new} y_2 = \alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \zeta \quad (3.36)$$

其中, ζ 是常数。两个因子不好同时求解, 所以可先求第二个乘子 α_2 的解 (α_2^{new}), 得到 α_2 的解 (α_2^{new}) 之后, 再用 α_2 的解 (α_2^{new}) 表示 α_1 的解 (α_1^{new})。为了求解 α_2^{new} , 得先确定 α_2^{new} 的取值范围。假设它的上下边界分别为 H 和 L , 那么有:

$$L \leq \alpha_2^{new} \leq H \quad (3.37)$$

接下来, 综合 $0 \leq \alpha_i \leq C, i = 1, \dots, n$ 和 $\alpha_1^{new} y_1 + \alpha_2^{new} y_2 = \alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \zeta$ 这两个约束条件, 求取 α_2^{new} 的取值范围。

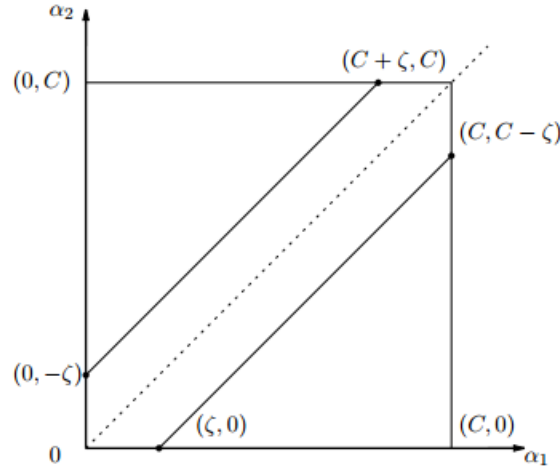


图 17: SMO 算法情况 1

当 $y_1 \neq y_2$ 时, 根据 $\alpha_1^{new} y_1 + \alpha_2^{new} y_2 = \alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \zeta$ 这可得 $\alpha_1^{old} - \alpha_2^{old} = \zeta$, 所以有 $L = \max(0, -\zeta)$, $H = \min(C, C - \zeta)$, 如图17所示。

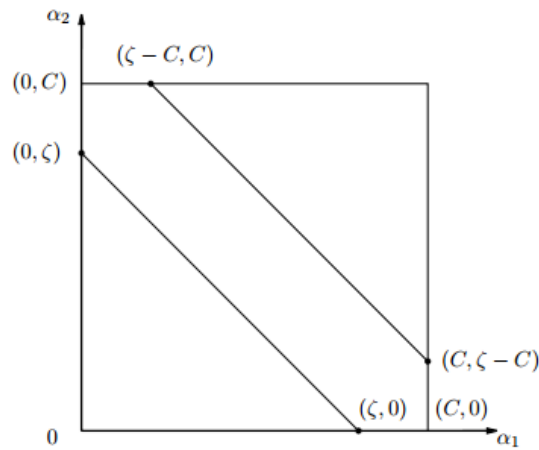


图 18: SMO 算法情况 2

当 $y_1 = y_2$ 时, 同样根据 $\alpha_1^{new} y_1 + \alpha_2^{new} y_2 = \alpha_1^{old} y_1 + \alpha_2^{old} y_2 = \zeta$ 可得: $\alpha_1^{old} + \alpha_2^{old} = \zeta$, 所以有, $L = \max(0, \zeta - C)$, $H = \min(C, \zeta)$, 如图18所示。

如此, 根据 y_1 和 y_2 异号或同号, 可得出 α_2^{new} 的上下界分别为:

$$\begin{cases} L = \max(0, \alpha_2^{old} - \alpha_1^{old}), H = \min(C, C + \alpha_2^{old} - \alpha_1^{old}) & \text{if } y_1 \neq y_2 \\ L = \max(0, \alpha_2^{old} + \alpha_1^{old} - C), H = \min(C, \alpha_2^{old} + \alpha_1^{old}) & \text{if } y_1 = y_2 \end{cases} \quad (3.38)$$

回顾下第二个约束条件 $\alpha_1^{new}y_1 + \alpha_2^{new}y_2 = \alpha_1^{old}y_1 + \alpha_2^{old}y_2 = \zeta$ 可, 令上式两边乘以 y_1 , 可得

$$\alpha_1 + s\alpha_2 = \alpha_1^* + s\alpha_2^* = w \quad (3.39)$$

其中, $w = -y_1 \sum_{i=3}^n y_i \alpha_i^*$ 。

因此 α_1 可以用 α_2 表示, $\alpha_1 = w - s\alpha_2$, 从而把子问题的目标函数转换为只含 α_2 的问题:

$$\begin{aligned} \Psi = & \frac{1}{2}K_{11}(w - s\alpha_2)^2 + \frac{1}{2}K_{22}\alpha_2^2 + sK_{12}(w - s\alpha_2)\alpha_2 \\ & + y_1(w - s\alpha_2)v_1 - w + s\alpha_2 + y_2\alpha_2v_2 - \alpha_2 + \Psi_{constant} \end{aligned} \quad (3.40)$$

对 α_2 求导, 可得:

$$\frac{\partial \Psi}{\partial \alpha_2} = -sK_{11}(w - s\alpha_2) + K_{22}\alpha_2 - K_{12}\alpha_2 + sK_{12}(w - s\alpha_2) - y_2v_1 + s + y_2v_2 - 1 = 0 \quad (3.41)$$

化简下:

$$\alpha_2(K_{11} + K_{22} - 2K_{12}) = s(K_{11} - K_{12})w + y_2(v_1 - v_2) + 1 - s \quad (3.42)$$

然后将 $s = y_1y_2$ 、 $\alpha_1 + s\alpha_2 = \alpha_1^* + s\alpha_2^* = w$ 和 $K_{ij} = K(\vec{x}_i, \vec{x}_j)$ 、 $v_i = \sum_{j=3}^n y_j \alpha_j^* K_{ij} = u_i + b^* - y_1\alpha_1^* K_{1i} - y_2\alpha_2^* K_{2i}$ 代入上式可得:

$$\alpha_2^{new,unc}(K_{11} + K_{22} - 2K_{12}) = \alpha_2^{old}(K_{11} + K_{22} - 2K_{12}) + y_2(u_1 - u_2 + y_2 - y_1) \quad (3.43)$$

令 $E_i = u_i - y_i$ (表示预测值与真实值之差), $\eta = K(\vec{x}_1, \vec{x}_1) + K(\vec{x}_2, \vec{x}_2) - 2K(\vec{x}_1, \vec{x}_2)$, 然后上式两边同时除以 η , 得到一个关于单变量 α_2 的解:

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\eta} \quad (3.44)$$

这个解没有考虑其约束条件 $0 \leq \alpha_2 \leq C$, 即是未经剪辑时的解。

然后考虑约束 $0 \leq \alpha_2 \leq C$ 可得到经过剪辑后的 α_2^{new} 的解析解为:

$$\alpha_2^{new,unc} = \begin{cases} H, & \alpha_2^{new,unc} > H \\ \alpha_2^{new,unc} & L \leq \alpha_2^{new,unc} \leq H \\ L, & \alpha_2^{new,unc} < L \end{cases} \quad (3.45)$$

求出了后 α_2^{new} , 便可以求出 α_1^{new} , 得 $\alpha_1^{new} = \alpha_1^{old} + y_1y_2(\alpha_2^{old} - \alpha_2^{new})$ 。

那么如何选择乘子 α_1 和 α_1 呢?

- 对于 α_1 ，即第一个乘子，可以通过刚刚说的那 3 种不满足 KKT 的条件来找；
- 而对于第二个乘子 α_2 可以寻找满足条件： $\max |E_i - E_j|$ 的乘子。

而 b 在满足下述条件：

$$b = \begin{cases} b_1, & 0 < \alpha_1^{new} < C \\ b_2 & 0 < \alpha_2^{new} < C \\ (b_1 + b_2)/2 & \text{otherwise} \end{cases} \quad (3.46)$$

下更新 b ：

$$b_1^{new} = b^{old} - E_1 - y_1(\alpha_1^{new} - \alpha_1^{old})K(x_1, x_1) - y_2(\alpha_2^{new} - \alpha_2^{old})K(x_1, x_2) \quad (3.47)$$

$$b_2^{new} = b^{old} - E_2 - y_1(\alpha_1^{new} - \alpha_1^{old})K(x_1, x_2) - y_2(\alpha_2^{new} - \alpha_2^{old})K(x_2, x_2) \quad (3.48)$$

且每次更新完两个乘子的优化后，都需要再重新计算 b ，及对应的 E_i 值。

最后更新所有 α_i ， y 和 b ，这样模型就出来了，从而即可求出咱们开头提出的分类函数：

$$f(x) = \sum_{i=1}^n \alpha_i y_i \langle x_i, x \rangle + b \quad (3.49)$$

此外，这里²⁰也有一篇类似的文章，大家可以参考下。

3.5.2 SMO 算法的步骤

综上，总结 SMO 的主要步骤，如下²¹：

算法 2: SMO 算法

Data: 给数据集 $\{x_i, y_i\}, i = 1, \dots, n$

Result: Lrange 乘子 α_i

```

1 while 未收敛 do
2   第一步，选择一对  $\alpha_i$  和  $\alpha_j$ ，选取方法使用启发式方法；
3   第二步，固定除  $\alpha_i$  和  $\alpha_j$  之外的其他参数，确定  $W(\alpha)$  极值条件下的  $\alpha_i, \alpha_j$ 
   由  $\alpha_i$  表示。
4 end
```

²⁰<http://www.cnblogs.com/jerrylead/archive/2011/03/18/1988419.html>

²¹整理者注：这里稍微修改了一下原文。

假定在某一次迭代中，需要更新 x_1 和 x_2 对应的拉格朗日乘子 α_1, α_2 ，那么这个小规模的二次规划问题写为：

$$\begin{aligned}
 L_s = \max_{\alpha} \{ & (\alpha_1 + \alpha_2) + \sum_{i=3}^n \alpha_i - \frac{1}{2} \|\alpha_1 y_1 \phi(x_1) + \alpha_2 y_2 \phi(x_2) + \sum_{i=3}^n \alpha_i y_i \phi(x_i)\|^2 \} \\
 \text{s.t. } & \alpha_1 y_1 + \alpha_2 y_2 = - \sum_{i=3}^n \alpha_i y_i, 0 < \alpha_i < C, \forall i
 \end{aligned} \tag{3.50}$$

那么在每次迭代中，如何更新乘子呢？引用这里²²的两张 PPT 说明下：更新 Larange 乘子 α_1, α_2 ：

- 步骤 1：计算上下界 L 和 H ：

$$* \quad L = \max(0, \alpha_2^{old} - \alpha_1^{old}), H = \min(C, C + \alpha_2^{old} - \alpha_1^{old}) \quad \text{if } y_1 \neq y_2$$

$$* \quad L = \max(0, \alpha_2^{old} + \alpha_1^{old} - C), H = \min(C, \alpha_2^{old} + \alpha_1^{old}) \quad \text{if } y_1 = y_2$$

- 步骤 2：计算 L_s 的二阶导数

$$* \quad \eta = 2\phi(x_1)^T \phi(x_2) - \phi(x_1)^T \phi(x_1) - \phi(x_2)^T \phi(x_2)$$

- 步骤 3：更新 L_s ：

$$\begin{aligned}
 \alpha_2^{new} &= \alpha_2^{old} - \frac{y_2(e_1 - e_2)}{\eta} \\
 e_i &= g^{old}(x_i) - y_i
 \end{aligned}$$

- 步骤 4：计算变量 α_2 ：

$$\alpha^{temp} = \begin{cases} H, & \text{if } \alpha_2^{new} \geq H \\ \alpha_2^{new}, & \text{if } L \leq \alpha_2^{new} \leq H \\ L, & \text{if } \alpha_2^{new} \leq L \end{cases}$$

- 步骤 5：更新 α_1

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2 (\alpha_2^{old} - \alpha^{temp})$$

知道了如何更新乘子，那么选取哪些乘子进行更新呢？具体选择方法有以下两个步骤：

1. 步骤 1：先“扫描”所有乘子，把第一个违反 KKT 条件的作为更新对象，令为 α_2 ；

²²<http://staff.ustc.edu.cn/~ketang/PPT/PRLec5.pdf>

2. 步骤 2: 在所有不违反 KKT 条件的乘子中, 选择使 $|E_1 - E_2|$ 最大的 α_1 进行更新, 使得能最大限度增大目标函数的值 (类似于梯度下降。此外 $E_i = u_i - y_i$, 而 $u = \vec{w} \cdot \vec{x} - b$, 求出来的 E 代表函数 u_i 对输入 x_i 的预测值与真实输出类标记 y_i 之差)。

最后, 每次更新完两个乘子的优化后, 都需要再重新计算 b , 及对应的 E_i 值。

综上, SMO 算法的基本思想是将 Vapnik 在 1982 年提出的 Chunking 方法推到极致, SMO 算法每次迭代只选出两个分量 α_i 和 α_j 进行调整, 其它分量则保持固定不变, 在得到解 α_i 和 α_j 之后, 再用 α_i 和 α_j 改进其它分量。与通常的分解算法比较, 尽管它可能需要更多的迭代次数, 但每次迭代的计算量比较小, 所以该算法表现出整理的快速收敛性, 且不需要存储核矩阵, 也没有矩阵运算。

3.5.3 SMO 算法的实现

函数间隔, 几何间隔

行文至此, 我相信, SVM 理解到了一定程度后, 是的确能在脑海里从头至尾推导出相关公式的, 最初分类函数, 最大化分类间隔, $\max 1/\|w\|$, $\min 1/2\|w\|^2$, 凸二次规划, 拉格朗日函数, 转化为对偶问题, SMO 算法, 都为寻找一个最优解, 一个最优分类平面。一步步梳理下来, 为什么这样那样, 太多东西可以追究, 最后实现。如图19所示:

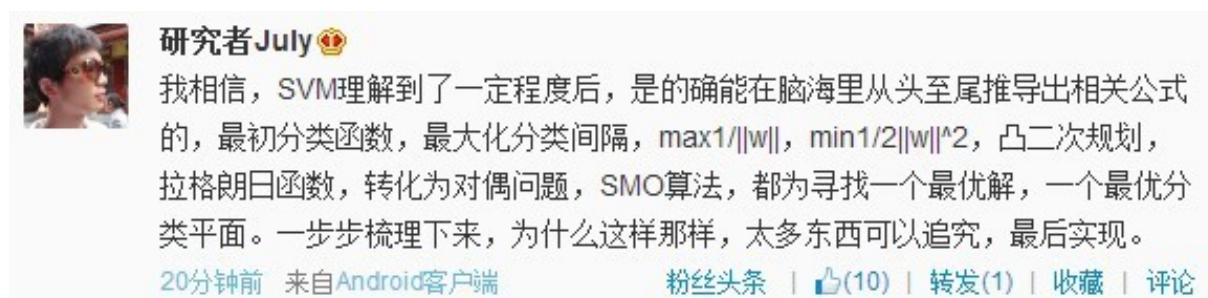


图 19: 来自 July 的微博

至于下文中将阐述的核函数则是为了更好的处理非线性可分的情况, 而松弛变量则是为了纠正或约束少量“不安分”或脱离集体不好归类的因子。

台湾的林智仁教授写了一个封装 SVM 算法的 libsvm²³库, 大家可以看看, 此外这里还有一份 libsvm 的注释文档²⁴。

除了在这篇论文《fast training of support vector machines using sequential minimal optimization》中 platt 给出了 SMO 算法的逻辑代码之外, 这里²⁵也有一份 SMO 的实

²³<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

²⁴http://www.pami.sjtu.edu.cn/people/gpliu/document/libsvm_src.pdf

²⁵<http://blog.csdn.net/techq/article/details/6171688>

现代码，大家可以看下。

3.6 SVM 的应用

或许我们已经听到过，SVM 在很多诸如文本分类，图像分类，生物序列分析和生物数据挖掘，手写字符识别等领域有很多的应用，但或许你并没强烈的意识到，SVM 可以成功应用的领域远远超出现在已经在开发应用了的领域。

3.6.1 文本分类

一个文本分类系统不仅是一个自然语言处理系统，也是一个典型的模式识别系统，系统的输入是需要进行分类处理的文本，系统的输出则是与文本关联的类别。由于篇幅所限，其它更具体内容本文将不再详述。

OK，本节虽取标题为证明 SVM，但聪明的读者们想必早已看出，其实本部分并无多少证明部分（特此致歉），怎么办呢？可以参阅《支持向量机导论》一书，此书精简而有趣。本节完。

读者评论

本文发表后，微博上的很多朋友给了不少意见，以下是节选的一些精彩评论：

1. “压力”陡增的评论 →//@ 藏了个锋：我是看着 July 大神的博文长大的啊//@zlkysl：就是看了最后那一篇才决定自己的研究方向为 SVM 的。
-<http://weibo.com/1580904460/zraWk0u6u?mod=weibotime>。
2. @ 张金辉：“SVM 的三重境界，不得不转的一篇。其实 Coursera 的课堂上 Andrew Ng 讲过支持向量机，但显然他没有把这作为重点，加上 Ng 讲支持向量机的方法我一时半会难以完全消化，所以听的也是一知半解。真正开始了解支持向量机就是看的这篇“三重境界”，之后才对这个算法有了大概的概念，以至如何去使用，再到其中的原理为何，再到支持向量机的证明等。总之，这篇文章开启了我长达数月的研究支持向量机阶段，直到今日。”-<http://zhan.renren.com/profile/249335584>。
3. @ 孤独之守望者：“最后，推出 svm 的 cost function 是 hinge loss，然后对比其他的方法的 cost function，说明其实他们的目标函数很像，那么问题是 svm 为什么这么 popular 呢？您可以再加些 VC dimension 跟一些 error bound 的数学，点一下，提供一个思路 and 方向”。-<http://weibo.com/1580904460/AiohoyDwq?mod=weibotime>。
4. @ 夏粉 _ 百度：“在面试时，考察 SVM 可考察机器学习各方面能力：目标函数，优化过程，并行方法，算法收敛性，样本复杂度，适用场景，调参经验，不过个人认为考察 boosting 和 LR 也还不错啊。此外，随着统计机器学习不断进步，SVM 只被当成使用了一个替代 01 损失 hinge 研究，更通用的方法被提出，损失函数研究替代损失与贝叶斯损失关系，算法稳定性研究替代损失与推广性能关系，凸优化研究如何求解凸目标函数，SVM,boosting 等算法只是这些通用方法的一个具体组建而已。”
5. @ 居里猴姐：关于 SVM 损失函数的问题，可以看看张潼老师的这篇《Statistical behavior and consistency of classification methods based on convex risk minimization》。各种算法中常用的损失函数基本都具有 fisher 一致性，优化这些损失函数得到的分类器可以看作是后验概率的“代理”。此外，张潼老师还有另外一篇论文《Statistical analysis of some multi-category large margin classification methods》，在多分类情况下 margin loss 的分析，这两篇对 Boosting 和 SVM 使用的损失函数分析的很透彻。
6. @ 夏粉 _ 百度：SVM 用了 hinge 损失，hinge 损失不可导，不如其它替代损失方便优化并且转换概率麻烦。核函数也不太用，现在是大数据时代，样本非常大，

无法想象一个 n^2 的核矩阵如何存储和计算。而且，现在现在非线性一般靠深度学习了。//@Copper_PKU: 请教 svm 在工业界的应用典型的有哪些？工业界如何选取核函数，经验的方法？svm 的训练过程如何优化？

7. @Copper_PKU: July 的 svm tutorial 我个人觉得还可以加入和修改如下部分：(1) 对于支持向量解释，可以结合图和拉格朗日参数来表达，松弛中 sv 没有写出来。(2) SMO 算法部分，加入 Joachims 论文中提到的算法，以及 SMO 算法选取 workset 的方法，包括 SMO 算法的收敛判断，还有之前共轭梯度求解方法，虽然是较早的算法，但是对于理解 SMO 算法有很好的效果。模型的优化和求解都是迭代的过程，加入历史算法增强立体感。- http://weibo.com/1580904460/Akw6dl3Yk#_rnd1385474436177。
8. //@ 廖临川: 之所以 sgd 对大训练集的效果更好，1. 因为 SGD 优化每次迭代使用样本子集，比使用训练全集（尤其是百万数量级）要快得多；2. 如果目标函数是凸的或者伪凸的，SGD 几乎必然可以收敛到全局最优；否则，则收敛到局部最优；3.SGD 一般不需要收敛到全局最优，只要得到足够好的解，就可以立即停止。//@Copper_PKU: sgd 的核心思想：是迭代训练，每拿到一个样本就算出基于当前 $w(t)$ 的 loss function, t 代表训练第 t 次，然后进行下一 $w(t+1)$ 的更新, $w(t+1)=w(t)-(learning\ rate) * loss\ function$ 的梯度，这个类比神经网络中 bp 中的参数训练方法。sample by sample 就是每次仅处理一个样本而不是一个 batch。
9. //@Copper_PKU: 从损失函数角度说：primal 问题可以理解为正则化项 +loss-function, 求解目标是在两个中间取平衡如果强调 loss function 最小则会 overfitting, 所以有 C 参数。//@ 研究者 July: SVM 还真就是在一定限定条件下，即约束条件下求目标函数的最优值问题，同时，为减少误判率，尽量让损失最小。

非常享受这种全民大讨论的年代，没有谁一定就对或一定就错，而是各自发表各自的理解见解，真棒！

参考文献及推荐阅读

1. 《支持向量机导论》，[美] Nello Cristianini / John Shawe-Taylor 著；
2. 支持向量机导论一书的支持网站：<http://www.support-vector.net/>；
3. 《数据挖掘导论》，[美] Pang-Ning Tan / Michael Steinbach / Vipin Kumar 著；
4. 《数据挖掘：概念与技术》，(加)Jiawei Han;Micheline Kamber 著；
5. 《数据挖掘中的新方法：支持向量机》，邓乃扬田英杰著；
6. 《支持向量机—理论、算法和扩展》，邓乃扬田英杰著；
7. 支持向量机系列，pluskid：http://blog.pluskid.org/?page_id=683；
8. http://www.360doc.com/content/07/0716/23/11966_615252.shtml；
9. 数据挖掘十大经典算法初探 http://blog.csdn.net/v_july_v/article/details/6142146；
10. 《模式识别支持向量机指南》，C.J.C Burges 著；
11. 《统计学习方法》，李航著；
12. 《统计自然语言处理》，宗成庆编著，第十二章、文本分类；
13. SVM 入门系列，Jasper：<http://www.blogjava.net/zhenandaci/category/31868.html>；
14. 最近邻决策和 SVM 数字识别的实现和比较，作者不详；
15. 斯坦福大学机器学习课程原始讲义：<http://www.cnblogs.com/jerrylead/archive/2012/05/08/2489725.html>；
16. 斯坦福机器学习课程笔记：<http://www.cnblogs.com/jerrylead/tag/Machine20Learning/>；
17. <http://www.cnblogs.com/jerrylead/archive/2011/03/13/1982639.html>；
18. SMO 算法的数学推导：<http://www.cnblogs.com/jerrylead/archive/2011/03/18/1988419.html>；
19. 数据挖掘中所需的概率论与数理统计知识、上；

20. 关于机器学习方面的文章, 可以读读: <http://www.cnblogs.com/vivounicorn/category/289453.html>;
21. 数学系教材推荐: http://blog.sina.com.cn/s/blog_5e638d950100dsw.html;
22. 《神经网络与机器学习 (原书第三版)》, [加] Simon Haykin 著;
23. 正态分布的前世今生: <http://t.cn/zlH3Ygc>;
24. 《数理统计学简史》, 陈希孺院士著;
25. 《最优化理论与算法 (第 2 版)》, 陈宝林编著;
26. A Gentle Introduction to Support Vector Machines in Biomedicine: http://www.nyuinformatics.org/downloads/supplements/SVM_Tutorial_2010/Final_WB.pdf, 此 PPT 很赞, 除了对引入拉格朗日对偶变量后的凸二次规划问题的深入度不够之外, 其它都挺好, 配图很精彩, 本文有几张图便引自此 PPT 中;
27. 来自卡内基梅隆大学 carnegie mellon university(CMU) 的讲解 SVM 的 PPT: <http://www.autonlab.org/tutorials/svm15.pdf>;
28. 发明 libsvm 的台湾林智仁教授 06 年的机器学习讲义 SVM: http://wenku.baidu.com/link?url=PWTGMYNb4HGUrUQUZwTH2B4r8pIMgLMiWiK1ymVORrds_11V0kHwp-JWab7IALDiors6mD93dtuWHwFWxsAk6p0rzchR8Qh5_4jWHC;
29. <http://staff.ustc.edu.cn/~ketang/PPT/PRlec5.pdf>;
30. Introduction to Support Vector Machines (SVM), By Debprakash Patnai M.E (SSA)<https://www.google.com.hk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CCwQFjAA&url=http%3a%2f%2fwww%2epws%2estu%2eedu%2etw%2fccfang%2findex%2efiles%2fAI%2fAI%26ML-Support%2520Vector%2520Machine-1%2eppt&ei=JRR6UqT5C-iyiQfWyIDgCg&usg=AFQjCNGw1fTbpH4ltQjjmx1d25ZqbCN9nA>;
31. 多人推荐过的 libsvm: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>;
32. 《machine learning in action》, 中文版为《机器学习实战》;
33. SMO 算法的提出: Sequential Minimal Optimization A Fast Algorithm for Training Support Vector Machines: <http://research.microsoft.com/en-us/um/people/jplatt/smoTR.pdf>;
34. 《统计学习理论的本质》, [美] Vladimir N. Vapnik 著, 非常晦涩, 不做过多推荐;

35. 张兆翔, 机器学习第五讲之支持向量机<http://irip.buaa.edu.cn/~zxzhang/courses/MachineLearning/5.pdf>;
36. VC 维的理论解释: <http://www.svms.org/vc-dimension/>, 中文 VC 维解释<http://xiaoxia001.iteye.com/blog/1163338>;
37. 来自 NEC Labs America 的 Jason Weston 关于 SVM 的讲义http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf;
38. 来自 MIT 的 SVM 讲义: <http://www.mit.edu/~9.520/spring11/slides/class06-svm.pdf>;
39. PAC 问题: <http://www.cs.huji.ac.il/~shashua/papers/class11-PAC2.pdf>;
40. 百度张潼老师的两篇论文: 《Statistical behavior and consistency of classification methods based on convex risk minimization》http://home.olemiss.edu/~xdang/676/Consistency_of_Classification_Convex_Risk_Minimization.pdf, 《Statistical analysis of some multi-category large margin classification methods》http://machinelearning.wustl.edu/mlpapers/paper_files/Zhang04a.pdf;
41. <http://jacoxu.com/?p=39>;
42. 《矩阵分析与应用》, 清华张贤达著;
43. SMO 算法的实现: <http://blog.csdn.net/techq/article/details/6171688>;
44. 常见面试之机器学习算法思想简单梳理: <http://www.cnblogs.com/tornadomeet/p/3395593.html>;
45. 矩阵的 wikipedia 页面: <http://zh.wikipedia.org/wiki/matrix>;
46. 最小二乘法及其实现: <http://blog.csdn.net/ql1125596718/article/details/8248249>;
47. 统计学习方法概论: <http://blog.csdn.net/ql1125596718/article/details/8351337>;
48. <http://www.csdn.net/article/2012-12-28/2813275-Support-Vector-Machine>;
49. A Tutorial on Support Vector Regression: <http://alex.smola.org/papers/2003/SmoSch03b.pdf>; SVR 简明版: <http://www.cmlab.csie.ntu.edu.tw/~cyy/learning/tutorials/SVR.pdf>。

50. SVM Org: <http://www.support-vector-machines.org/>;
51. R. Collobert. Large Scale Machine Learning. Université Paris VI phd thesis. 2004: http://ronan.collobert.com/pub/matos/2004_phdthesis_lip6.pdf;
52. Making Large-Scale SVM Learning Practical: http://www.cs.cornell.edu/people/tj/publications/joachims_99a.pdf;
53. 文本分类与 SVM: <http://blog.csdn.net/zhzh1202/article/details/8197109>;
54. Working Set Selection Using Second Order Information
55. for Training Support Vector Machines: <http://www.csie.ntu.edu.tw/~cjlin/papers/quadworkset.pdf>;
56. SVM Optimization: Inverse Dependence on Training Set Size: <http://icml2008.cs.helsinki.fi/papers/266.pdf>;
57. Large-Scale Support Vector Machines: Algorithms and Theory: <http://cseweb.ucsd.edu/~akmenon/ResearchExam.pdf>;
58. 凸优化的概念: <http://cs229.stanford.edu/section/cs229-cvxopt.pdf>;
59. 《凸优化》，作者: Stephen Boyd / Lieven Vandenberghe, 原名: Convex Optimization;
60. Large-scale Non-linear Classification: Algorithms and Evaluations, Zhuang Wang, 讲了很多 SVM 算法的新进展: http://ijcai13.org/files/tutorial_slides/te2.pdf;
61. 基于 SMO 算法实现 SVM: <http://www.cs.iastate.edu/~honavar/smo-svm.pdf>;
62. copper 推荐的一些 SVM 相关的论文 (当然, 其中不少论文在上面的条目中都已经提到): <http://c.blog.sina.com.cn/profile.php?blogid=68d0b92d89000h35>。

后记

OK, 此文从最初 2012 年 5 月开始动笔, 到后续不断的修改, 创造了三个之最, 即所写时间最长, 所花心血最大, 所改次数最多, 因为我的目标是让没有任何机器学习基础的都能看懂此文, 所以总是不停的改, 不停的改, 不想放过任何一个小的细节。再者, 引用侯捷的一句话是: 天下大作, 必作于细。

最后, 非常感谢 pluskid 及诸多朋友们的文章及著作, 让我有机会在其基础上总结、深入。有任何问题, 敬请广大读者随时不吝批评指正, 感谢。

Divide-and-conquer algorithm

This algorithm was first introduced by Preparata and Hong (1977[164]). The divide and conquer algorithm work for general dimensional data. Time complexity of two and three dimensional data is reported as $O(n \log n)$ (Edelsbrunner, 1987[66]). Time complexity for 4 dimensional data is known to be $O((n + N) \log^2 N)$ and for 5 dimensional data is $O((n + N) \log^3 N)$ where n is sample size and N is the number of vertices. However, it is known that merging process is very difficult when data dimension is more than two. $O((n+N)\log N \log N)$

The Divide-and-conquer algorithm for two dimension can be stated simply as follows.

1. Sort the points by x coordinate
2. Divide the points into two sets \mathcal{A} and \mathcal{B} , \mathcal{A} containing the left upper $n/2$ points and \mathcal{B} the right lower $n/2$ points
3. Compute the convex hulls $A = \text{conv}(\mathcal{A})$ and $B = \text{conv}(\mathcal{B})$ recursively.
4. Merge A and B : compute $\text{conv}(A \cup B)$

The higher dimension ($d > 2$) divide-and-conquer algorithm stated in *Algorithm 4* is adopted from O'Rourke (2001).