过拟合与正则化

过拟合的原因

模形过拟合的原因主要有以下几种：

- 训练数据集样本单一，样本不足。如果训练样本只有负样本，然后那生成的模型去预测正样本，这肯定预测不准。所以训练样本要尽可能的全面，覆盖所有的数据类型。
- 训练数据中噪声干扰过大。噪声指训练数据中的干扰数据。过多的干扰会导致记录了很多噪声特征，忽略了真实输入和输出之间的关系。
- 模型过于复杂。模型太复杂，已经能够死记硬背记录下了训练数据的信息，但是遇到没有见过的数据的时候不能够变通，泛化能力太差。我们希望模型对不同的模型都有稳定的输出。

模型太复杂是过拟合的重要因素。在有限的训练数据下，当模型太过复杂，得到了一个对于训练数据的完美拟合，而在测试集上表现性能很差。在PRML中对于过拟合、模型的复杂度(在这里为多项式复杂度)、模型的权重有如下描述：

> 通过考察不同阶数多项式的系数w*的值，如表1.1所示。我们看到随着M的增大，系数的大小通常会变大。对于M = 9的多项式，通过调节系数，让系数取相当大的正数或者负数，多项式函数可以精确地与数据匹配，但是对于数据之间的点（尤其是临近区间端点处的点），从图1.4可以看到函数表现出剧烈的震荡。 形成原因是因为有着更大的M值的更灵活的多项式被过分地调参，使得多项式被调节成了与目标值的随机噪声相符。

	$M=0$	$M=1$	$M=3$	$M=9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

表 1.1: 不同阶数的多项式的系数$\boldsymbol{w}^*$的值。观察随着多项式阶数的增加，系数的大小是如何剧烈增大的。

假设我们有线性模型如下：

$$y(x,w) = w_0 + w_1x_1 + \ldots\ldots + W_Mx^M$$

当M=9时，模型经过不同的数据集训练过后，拟合的情况如下：

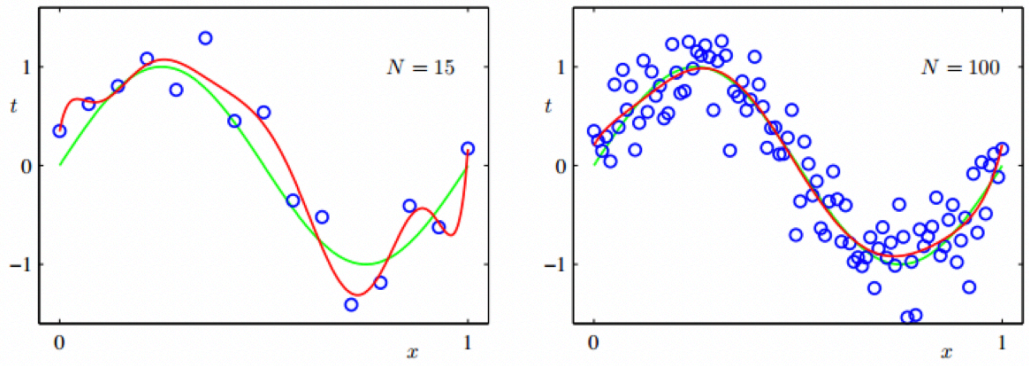


图 1.6: 使用$M = 9$的多项式对$M = 15$个数据点（左图）和$N = 100$个数据点（右图）通过最小化平方和误差函数的方法得到的解。我们看到增大数据集的规模会减小过拟合问题。

那么解决过拟合的方法有哪些呢?

- 增加训练数据。数据量越大，对于一个给定的模型复杂度，当数据集的规模增加时，过拟合问题变得不那么严重，也就是说数据越多我们能够得到更加复杂的模型，而模型也越灵活，不会出现严重的过拟合；一个粗略的启发是，数据点的数量不应该小于模型的可调节参数的数量的若干倍（比如5或10）。从模型的复杂度度量来看，参数个数并不是必须作为最合适的度量方法 。
- 加入正则化项
- Dropout
- Batch Normalization
- Bagging和其他集成方法
- 提前终止
- 参数绑定与参数终止
- 辅助分类节点

本文主要讲解正则化缓解过拟合的问题。

正则化

以L2正则化为例子:

$$L = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, w) - t_n)^2 + \frac{\lambda}{2}||w||^2$$

这样的技术在统计学的文献中被叫做收缩（shrinkage）方法，因为这种方法减小了系数的值。二次正则项的一个特殊情况被称为山脊回归（ridge regression）（Hoerl and Kennard, 1970）。在神经网络的情形中，这种方法被叫做权值衰减（weight decay，λ对应权值衰减系数）。 对于有10个点的sin(x)+noise生成的数据，加入正则化后，通过调节权重系数，得到了不同的拟合结果:

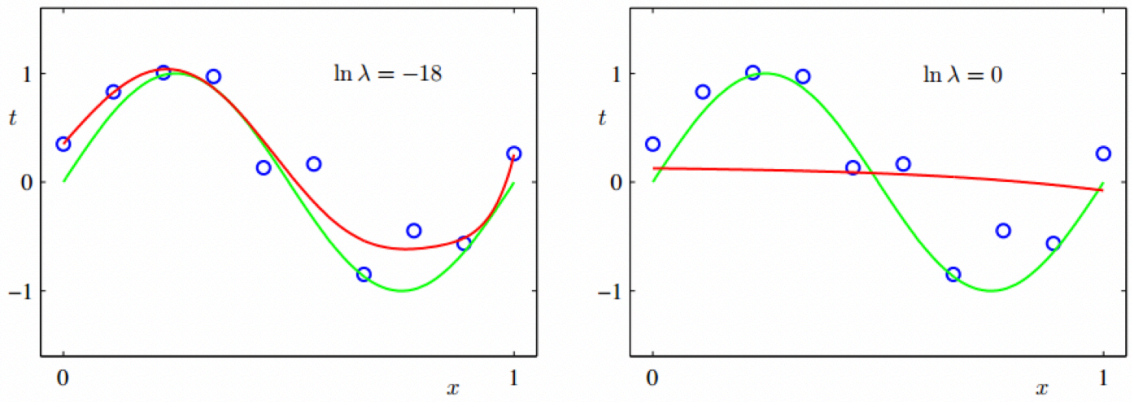


图 1.7: 使用正则化的误差函数（1.4），用$M = 9$的多项式拟合图1.2中的数据集。其中正则化参数λ选择了两个值，分别对应于$\ln\lambda = -18$和$\ln\lambda = 0$。没有正则化项的情形，即$\lambda = 0$，对应于$\ln\lambda = -\infty$，在图1.4的右下角给出。

在效果上， λ控制了模型的复杂性，因此决定了过拟合的程度。 下图为不同 使用后模型参数的情况，可见正则化能够限制参数不会被拟合得特别大，从而降低过拟合(PRML中对正则化能防止过拟合的解释，但是感觉说的还是不够清楚，不够有理有据，为什么权值小了，就能减小过拟合呢？)。下图为正则化系数大小和模型参数大小的一个关系：

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01

表 1.2: 不同的正则化参数λ下，$M = 9$的多项式的系数$\boldsymbol{w}^*$的值。注意，$\ln\lambda = -\infty$对应于没有正则化的模型，即图1.4右下角的模型。我们看到，随着λ的增大，系数的大小逐渐变小。

那机器学习中有哪些正则化方法呢?

$$L = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, w) - t_n)^2 + \frac{\lambda}{2}||w||^q$$

其中q的不同，有不同的正则化效果：

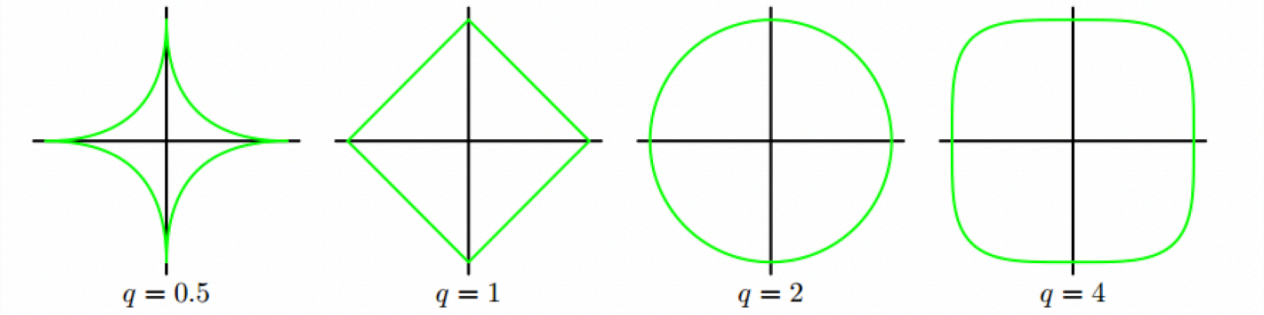


图 3.3: 对于不同的参数q，公式（3.29）中的正则化项的轮廓线。

在统计学的文献中， $q = 1$的情形被称为套索（lasso）（Tibshirani, 1996）。它的性质为：如果λ充分大，那么某些系数会变为零，从而产生了一个**稀疏（sparse）模型**，这个模型中对应的基函数不起作用。 如图3.4，对于M=2的情况，最小化误差,那么就 产生了稀疏项，对于更多维的比如M=9，就可能产生更多的0,

> 既然L0可以实现稀疏，为什么不用L0，而要用L1呢？个人理解一是因为L0范数很难优化求解（NP难问题），二是L1范数是L0范数的最优凸近似，而且它比L0范数要容易优化求解。所以大家才把目光和万千宠爱转于L1范数。OK，来个一句话总结：L1范数和L0范数可以实现稀疏，L1因具有比L0更好的优化求解特性而被广泛应用。

为什么选择稀疏特征?

> 1）家对稀疏规则化趋之若鹜的一个关键原因在于它能实现特征的自动选择。一般来说，xi的大部分元素（也就是特征）都是和最终的输出yi没有关系或者不提供任何信息的，在最小化目标函数的时候考虑xi这些额外的特征，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的信息反而会被考虑，从而干扰了对正确yi的预测。稀疏规则化算子的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些没有信息的特征，也就是把这些特征对应的权重置为0。
>
> 2）另一个青睐于稀疏的理由是，模型更容易解释。例如患某种病的概率是y，然后我们收集到的数据x是1000维的，也就是我们需要寻找这1000种因素到底是怎么影响患上这种病的概率的。假设我们这个是个回归模型：y=w1*x1*+*w2*x2+…+w1000*x1000*+*b*（*当然了，为了让y限定在(0,1)的范围，一般还得加个Logistic函数*）。*通过学习，如果最后学习到的w*就只有很少的非零元素，例如只有5个非零的wi，那么我们就有理由相信，这些对应的特征在患病分析上面提供的信息是巨大的，决策性的。

相比于q=2时，并不会让摸个参数为0，而是会让各个参数都小，针对不同的数据，有的被训练的更小。

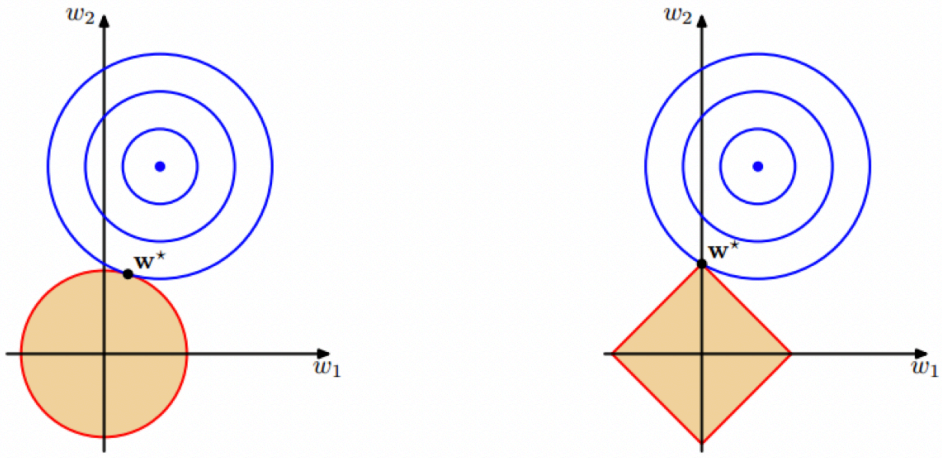


图 3.4: 未正则化的误差函数的轮廓线（蓝色）以及公式（3.30）给出的限制区域。左图是$q = 2$的二次正则化项的限制区域，右图是$q = 1$的套索正则化项的限制区域，其中参数向量$\boldsymbol{w}$的值被记作$\boldsymbol{w}^*$。套索正则化项给出了一个稀疏的解，其中$w_1^* = 0$。

为什么加入正则化就能防止过拟合呢?

PRML中说越小的参数，模型越简单，为什么越小的参数说明模型越简单?

> 楼主讲得太好，先点赞！ 楼主认为"L2 正则项的作用是让所有的 w 都变小，而 w 越小模型越简单"，大家普遍感到难以理解，我在这里讲讲我自己的不同看法，希望能够帮到大家。 我认为 L2 正则项的作用并不是让所有的 w 都变小，而是**【有选择地让某些 w 变小】**。正如题主举得医生预测疾病的例子，样本中的特征有很多，但大部分特征都是无关紧要的，只有一小部分关键的特征支撑起了整个预测模型。表现在系数 w 上就是，大部分的 w_i 都是不幸的，因为它们刚好与那些无关紧要的特征结对，它们的大小对整个模型的效果影响不大，于是在正则项的约束下它们都变小了，甚至趋近于0；而只有小部分的 w_i 比较幸运，它们刚好对应到了好的特征，于是它们肩负起了非常重大的责任，它们的微小变化会引起模型曲线在走势上的根本性变化，损失函数会急剧增大。如果正则项妄图约束这些关键的 w_i，使它们变小，那么由此造成的损失函数的扩大将远大于从正则项上获得的微小收益，所以这些关键的 w_i 可以几乎不受正则项的干涉。 但也不尽然，如果你把正则项之前的系数 λ 调到非常大，那么它就会敢于压迫那些关键的 w_i，最终造成的结果是，模型确实变简单了，但也严重偏离了预期方向，没什么卵用了。相反，如果你把 λ 调得非常小，那么正则项对每个 w_i 都惹不起，即使是那些无关紧要的 w_i 它也无力约束，最终就会导致模型过拟合（试想 λ 等于0的情况）。所以，损失函数与正则项就像是博弈的双方，它们之间的力量对比通过参数 λ 进行调和。只有把 λ 调合适了，才能得到既不过拟合，又相对简单的好模型。从这种意义上来说，L2正则项与L1正则项类似，也有"特征选择"的效果。 上面的描述比较感性，是我为了方便直观理解做的一些比喻，如果把模型的预测曲线做出来会更加严谨一些。即每个 w_i 都影响着曲线的形态，但是有主次之分。那些低阶的、关键的 w_i 控制着曲线的整体走势；而那些高阶的、次要的 w_i 则是在曲线整体走势的基础上稍微扭曲曲线的形态；当然，还会有更高阶的 w_i，它们负责在大的扭曲之上制造更小的扭曲，以此类推。 这样看来L2正则项的作用就很明显了，要改变预测曲线的整体走势肯地会造成损失函数的不满，但是把曲线的形态熨平似乎并没有什么不妥。而 λ 的大小则决定了正则项的视野，即多大的弯曲算作走势？多小的弯曲算作扭曲？

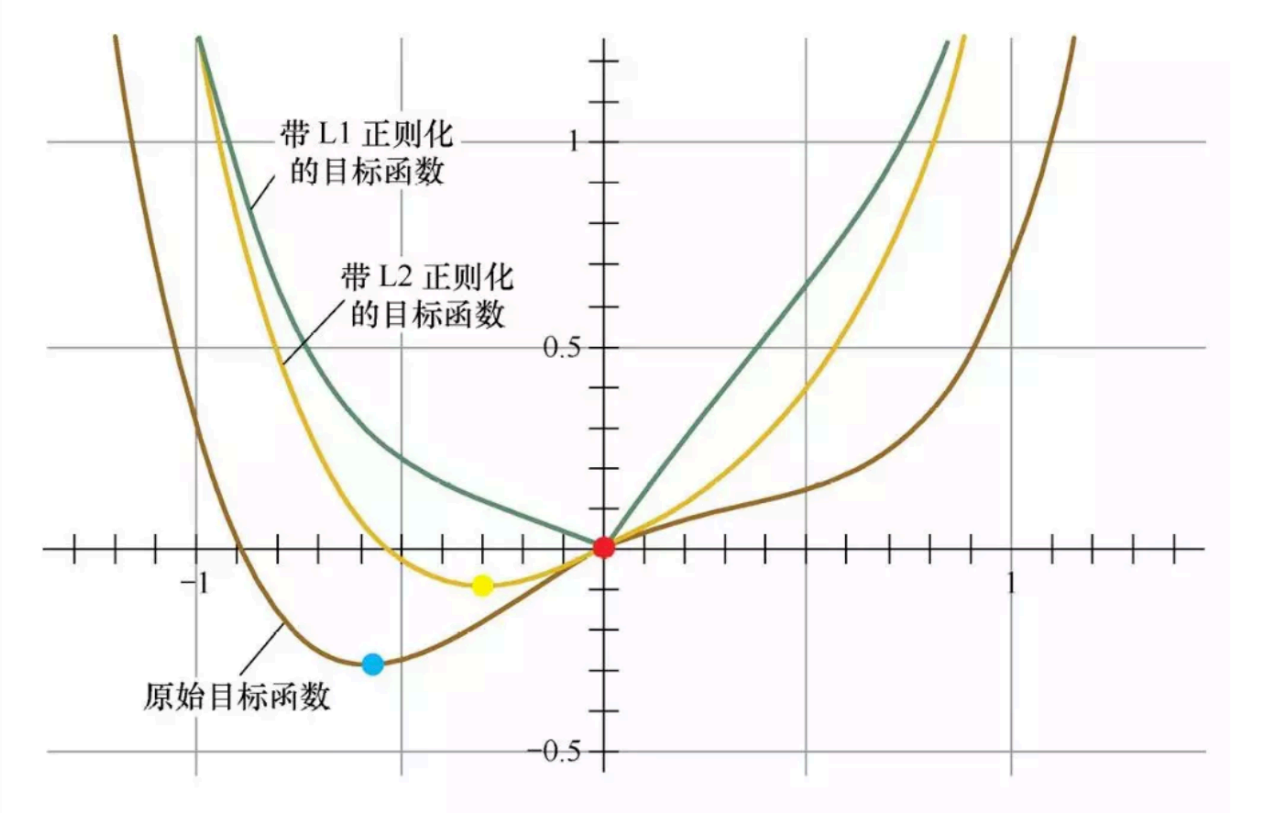
L1与L2正则效果的解释

从解空间形状来解释

L2正则化相当于为参数定义了一个圆形的解空间，而L1正则化相当于为参数定义了一个菱形的解空间。L1"棱角分明"的解空间显然更容易与目标函数等高线在脚点碰撞，因此加上L1范数容易得到稀疏解（0比较多）。而加上L2正则相比于L1正则来说，得到的解比较平滑（不是稀疏），但是同样能够保证解中接近于0（但不是等于0，所以相对平滑）的维度比较多，降低模型的复杂度。

从函数叠加来解释

我们考虑一维的情况，横轴是参数的值，纵轴是损失函数，加入正则项之后，损失函数曲线图变化如下：



可以看到，在加入L1正则项后，最小值在红点处，对应的w是0。而加入L2正则项后，最小值在黄点处，对应的w并不为0。加入L1正则项后，目标函数变为L(w)+C|w|，单就正则项部分求导，原点左边的值为-C，原点右边的值为C，因此，只要原目标函数的导数绝对值|L'(w)|<C,那么带L1正则项的目标函数在原点左边部分始终递减，在原点右边部分始终递增，最小值点自然会出现在原点处。加入L2正则项后，目标函数变为L(w)+Cw2，只要原目标函数在原点处的导数不为0，那么带L2正则项的目标函数在原点处的导数就不为0，那么最小值就不会在原点。因此L2正则只有减小w绝对值的作用，但并不能产生稀疏解。

从贝叶斯先验来解释

从贝叶斯角度来看，L1正则化相当于对模型参数w引入了拉普拉斯先验，L2正则化相当于引入了高斯先验，我们来看一下高斯分布和拉普拉斯分布的形状：

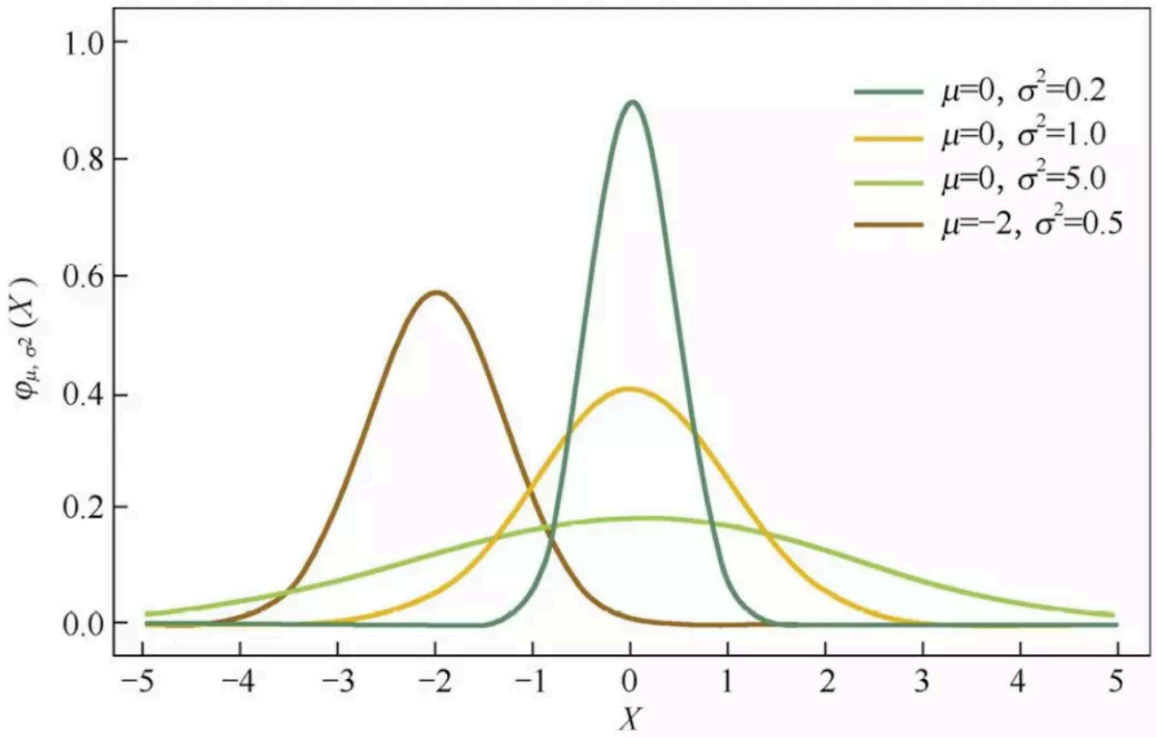


图7.8　高斯分布曲线图

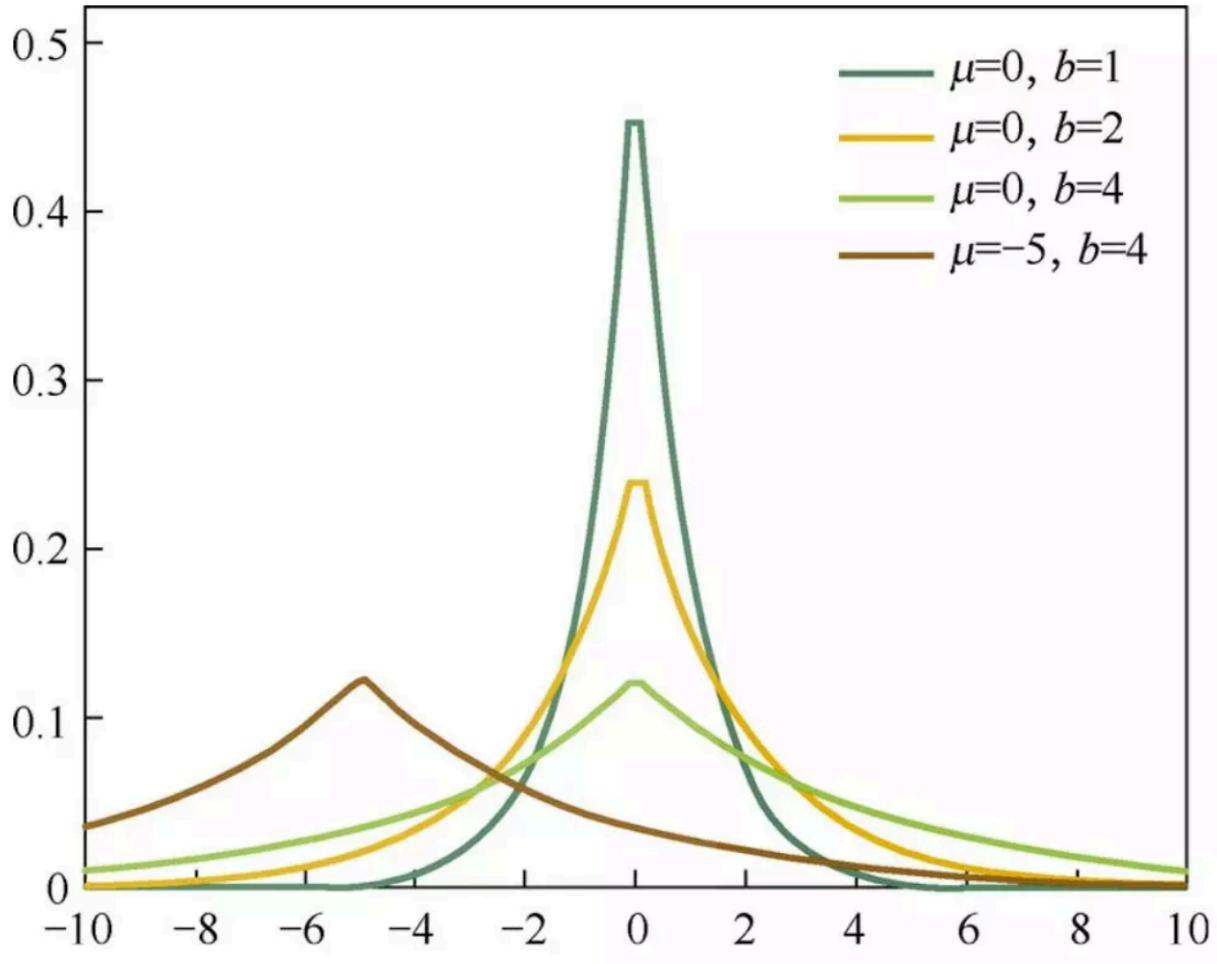


图7.9　拉普拉斯分布曲线图

均值为0时，高斯分布在极值点处是平滑的，也就是高斯先验分布认为w在极值点附近取不同值的可能性是接近的。但对拉普拉斯分布来说，其极值点处是一个尖峰，所以拉普拉斯先验分布中参数w取值为0的可能性要更高。

从数学角度解释

从优化或者数值计算的角度来说，L2范数有助于处理 condition number不好的情况下矩阵求逆很困难的问题。优化有两大难题，一是：局部最小值，二是：ill-condition病态问题。前者俺就不说了，大家都懂吧，我们要找的是全局最小值，如果局部最小值太多，那我们的优化算法就很容易陷入局部最小而不能自拔，这很明显不是观众愿意看到的剧情。那下面我们来聊聊ill-condition。ill-condition对应的是well-condition。那他们分别代表什么？假设我们有个方程组AX=b，我们需要求解X。如果A或者b稍微的改变，会使得X的解发生很大的改变，那么这个方程组系统就是ill-condition的，反之就是well-condition的。

我们的目标函数如果是二次的，对于线性回归来说，那实际上是有解析解的，求导并令导数等于零即可得到最优解为：

$$\hat{W} = (X^T X)^{-1} X^T y$$

如果当我们的样本X的数目比每个样本的维度还要小的时候，矩阵XTX将会不是满秩的，也就是XTX会变得不可逆，所以$w*$就没办法直接计算出来了。或者更确切地说，将会有无穷多个解（因为我们方程组的个数小于未知数的个数）。也就是说，我们的数据不足以确定一个解，如果我们从所有可行解里随机选一个的话，很可能并不是真正好的解，总而言之，我们过拟合了。但如果加上L2规则项，就变成了下面这种情况，就可以直接求逆了

$$\hat{W} = (X^T X + \lambda I)^{-1} X^T y$$

要得到这个解，我们通常并不直接求矩阵的逆，而是通过解线性方程组的方式（例如高斯消元法）来计算。

考虑没有规则项的时候，也就是λ=0的情况，如果矩阵$X^T X$的 condition number 很大的话，解线性方程组就会在数值上相当不稳定，而这个规则项的引入则可以改善condition number。condition number 太大会导致问题：它会拖慢迭代的收敛速度，而规则项从优化的角度来看，实际上是将目标函数变成λ-strongly convex（λ强凸）的了。如果我们有"强凸"的话，我们就可以得到一个更好的近似解。这里面有一个bound，这个 bound 的好坏也要取决于strongly convex性质中的常数α的大小。condition number 越小，上界就越小，也就是收敛速度会越快。

看到这里，不知道大家学聪明了没有。如果要获得strongly convex怎么做？最简单的就是往里面加入一项$\frac{\alpha}{2} * ||w||2$。

参考资料

1.带答案面经分享-L1正则&L2正则

2.机器学习中的范数规则化之（一）L0、L1与L2范数

3.PRML

4.模型欠拟合和过拟合解决办法

5.深度学习防止过拟合的几种方法