> zhuanlan.zhihu.com

1. l_p - 范数的定义

在很多机器学习相关的著作和教材中，我们经常看到各式各样的距离及范数，如

$$||\boldsymbol{x}||、||\boldsymbol{X}||$$

其中，$\boldsymbol{x}, \boldsymbol{X}$ 分别表示向量和矩阵。

当然，也会看到欧式距离、均方误差等。例如，向量 $\boldsymbol{x} = [3, -2, 1]^T$ 的欧式范数 (Euclidean norm) 为

$$\|\boldsymbol{x}\|_2 = \sqrt{3^2 + (-2)^2 + 1^2} = 3.742$$

用于表示向量的大小，这个范数也被叫做 l_2 - 范数。

为方便统一，一般将任意向量 $\boldsymbol{x}$ 的 l_p - 范数定义为

$$\|\boldsymbol{x}\|_p = \sqrt[p]{\sum_i |x_i|^p}$$

1.1 l_0 - 范数的定义

根据 l_p - 范数的定义，当 $\boldsymbol{p = 0}$，我们就有了 l_0 - 范数，即

$$\|\boldsymbol{x}\|_0 = \sqrt[0]{\sum_i x_i^0}$$

表示向量 $\boldsymbol{x}$ 中非 0 元素的个数，等同于 $\|\boldsymbol{x}\|_0 = \#(i|x_i \neq 0)$。

在诸多机器学习模型中，比如压缩感知 (compressive sensing)，我们很多时候希望最小化向量的 l_0 - 范数。一个标准的 l_0 - 范数优化问题往往可以写成如下形式：

$$\begin{aligned} &\mathbf{min} ||\boldsymbol{x}||_0 \\ &\mathbf{s.t.} \boldsymbol{Ax} = \boldsymbol{b} \end{aligned}$$

然而，由于 l_0 - 范数仅仅表示向量中非 0 元素的个数，因此，这个优化模型在数学上被认为是一个 NP-hard 问题，即直接求解它很复杂、也不可能找到解。

需要注意的是，正是由于该类优化问题难以求解，因此，压缩感知模型是将 l_0 - 范数最小化问题转换成 l_1 - 范数最小化问题。

1.2 l_1 - 范数的定义

根据 l_p - 范数的定义，当 $\boldsymbol{p=1}$ 时，任意向量 $\boldsymbol{x}$ 的 l_1 - 范数为

$$||\boldsymbol{x}||_1=\sum_i|\boldsymbol{x_i}|$$

等于向量中所有元素绝对值之和。

相应地，一个 l_1 - 范数优化问题为

$$\begin{aligned}&\mathbf{min}||\boldsymbol{x}||_1\\&\mathbf{s.t.}\boldsymbol{Ax}=\boldsymbol{b}\end{aligned}$$

这个问题相比于 l_0 - 范数优化问题更容易求解，借助现有凸优化算法（线性规划或是非线性规划），就能够找到我们想要的可行解。鉴于此，依赖于 l_1 - 范数优化问题的机器学习模型如压缩感知就能够进行求解了。

1.3 l_2 - 范数的定义

l_2 - 范数表示向量（或矩阵）的元素平方和，即

$$\|\boldsymbol{x}\|_2=\sqrt{\sum_i\boldsymbol{x}_i^2}$$

l_2 - 范数的优化模型如下：

$$\begin{aligned}&\mathbf{min}||\boldsymbol{x}||_2\\&\mathbf{s.t.}\boldsymbol{Ax}=\boldsymbol{b}\end{aligned}$$

2. l_1 - **范数**：正则项与稀疏解

在机器学习的诸多方法中，假设给定了一个比较小的数据集让我们来做训练，我们常常遇到的问题可能就是**过拟合** (over-fitting) 了，即训练出来的模型可能将数据中隐含的噪声和毫无关系的特征也表征出来。

为了避免类似的过拟合问题，一种解决方法是在 (机器学习模型的) 损失函数中加入正则项，比如用 l_1 - **范数**表示的正则项，只要使得 l_1 - 范数的数值尽可能变小，就能够让我们期望的解变成一个**稀疏解** (即解的很多元素为 0)。

如果我们想解决的优化问题是损失函数 $\boldsymbol{f(x)}$ 最小化，那么，考虑由 l_1 - 范数构成的正则项后，优化目标就变成

$$\mathbf{min}\boldsymbol{f}(\boldsymbol{x})+||\boldsymbol{x}||_1$$

尽管类似的优化模型看起来很 “简练”，在很多著作和教材中也会加上这样一句说明：

> 只要优化模型的解 $\boldsymbol{x}$ 的 l_1 - 范数比较小，那么这个解就是稀疏解，并且稀疏解可以避免过拟合。其中，“稀疏” 一词可以理解为 $\boldsymbol{x}$ 中的大多数元素都是 0，只有少量的元素是非 0 的。

但对于一些机器学习的初学者来说，给出这样没有解释的东西无疑是当头一棒。

为了理解 l_1 - 范数的正则项和稀疏性之间的关系，我们可以想想下面三个问题：

- 为什么 l_1 范数就能使得我们得到一个稀疏解呢？
- 为什么稀疏解能够避免过拟合？
- 正则项在模型中扮演者何种角色？

2.1 什么是过拟合问题？

在讨论上面三个问题之前，我们先来看看什么是**过拟合问题**：假设我们现在买了一个机器人，想让它学会区分汉字，例如

认定**前 5 个字属于第一类，后 5 个字属于第二类。**在这里，10 个字是所有的训练 “数据” 。

不幸的是，机器人其实很聪明，它能够把所有的字都 “记住” ，看过这 10 个字以后，机器人学会了一种**分类**的方式：它**把前 5 个字的一笔一划都准确地记在心里。**只要我们给任何一个字，如 “**揪**”(不在 10 个字里面)，它就会很自信地告诉你，**非此即彼**，这个字属于第二类。

当然，对于这 10 个字，机器人可以区分地非常好，准确率 100%. 但是，对于 

机器人没见过这个字 (不在 10 个字里面)，它将这个字归为第二类，这可能就错了。

因为我们可以明显看到，前 5 个字都带提手旁：

所以，**“揪” 属于第一类。**机器人的失败在于它太聪明，而训练数据又太少，不允许它那么聪明，这就是过拟合问题。

2.2 正则项是什么？为什么稀疏可以避免过拟合？

我们其实可以**让机器人变笨一点，希望它不要记那么多东西。**

接下来，我们开始新一轮的测试...还是给它前面测试过的那 10 个字，但现在机器人已经没办法记住前 5 个字的一笔一划了（存储有限），它此时只能**记住一些简单的模式**，于是，**第一类字都带有提手旁**就被它成功地发现了。

实际上，这就是 l_1 - **范数正则项的作用。**

l_1 - 范数会让你的模型**变傻一点**，相比于记住事物本身，此时机器人更倾向于从数据中找到一些简单的模式。

机器人原来的解：**[把, 打, 扒, 捕, 拉]**

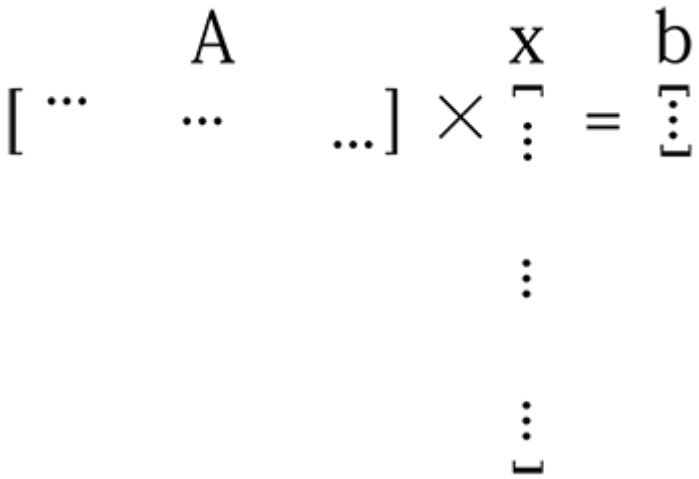
机器人变傻以后的解：**[扌，0, 0, 0, 0]**

假设我们有一个待训练的机器学习模型，如下：

$$\boldsymbol{Ax} = \boldsymbol{b}$$

其中， A 是一个训练数据构成的矩阵， $\boldsymbol{b}$ 是一个带有标签的向量，这里的 $\boldsymbol{x}$ 是我们希望求解出来的解。

当训练样本很少 (training data is not enough)、向量 $\boldsymbol{x}$ 长度很长时，这个模型的解就很多了。



如图，矩阵 A 的行数远少于向量 $\boldsymbol{x}$ 的长度。

我们希望的是找到一个比较合理的解，即向量 $\boldsymbol{x}$ 能够发现有用的特征 (useful features)。使用 l_1 - 范数作为正则项，向量 $\boldsymbol{x}$ 会变得稀疏，非零元素就是有用的特征了。

当然这里也有一个比较生动的例子：

> Suppose you are the king of a kingdom that has a large population and an OK overall GDP, but the per capita is very low. Each one of your citizens is lazy and unproductive and you are mad. Therefore you command “be productive, strong and hard working, or you die!” And you enforce the same GDP as before. As a result, many people died due to your harshness, those who survived your tyranny became really capable and productive. [example]

如果把总人口总量视作向量 $\boldsymbol{x}$ 的长度，那么 **“优胜劣汰” 其实相当于增加了一个正则项。**在稀疏的结果中，我们能够保证向量 $\boldsymbol{x}$ 的每个元素都是有用的！

到这里，我们知道了为什么稀疏可以避免过拟合。

2.3 为什么增加 l_1 - 范数能够保证稀疏?

根据 l_1 - 范数的定义，向量的l_1 - 范数是所有元素的绝对值之和，以向量 $[\boldsymbol{x}, \boldsymbol{y}]^T$ 为例，其 l_1 - 范数为 $|\boldsymbol{x}| + |\boldsymbol{y}|$ 。

选取两个向量：

$\boldsymbol{x}_1 = [0.1, 0.1]^T$ 和 $\boldsymbol{x}_2 = [1000, 0]^T$

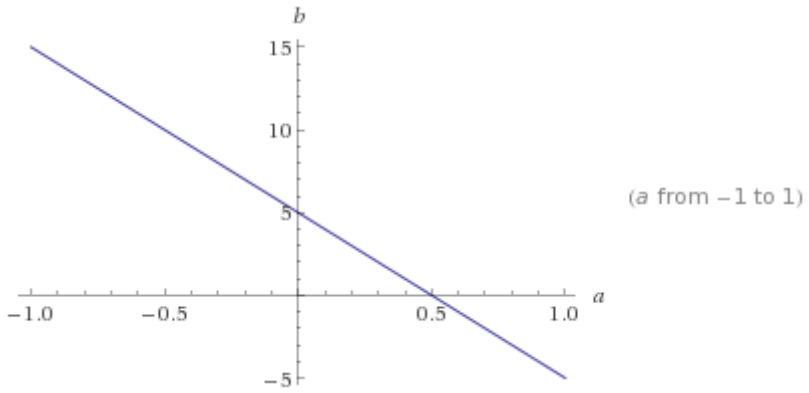
其中， $\boldsymbol{x_1}$ 很明显不是一个稀疏向量，但其 l_1 - 范数 $||\boldsymbol{x}_1||_1 = |0.1| + |0.1| = 0.2$ 却远小于稀疏向量 $\boldsymbol{x_2}$ 的 l_1 - 范数 $||\boldsymbol{x}_2||_1 = |1000| + |0| = 1000$.

仅仅是看 l_1 - 范数的数值大小，我们可能很难比较向量的稀疏程度，因此，需要结合损失函数。

再回到前面的问题： $\boldsymbol{Ax} = \boldsymbol{b}$ ，在平面直角坐标系上，假设一次函数 $y = ax + b$ 经过点 $(10, 5)$ ，则

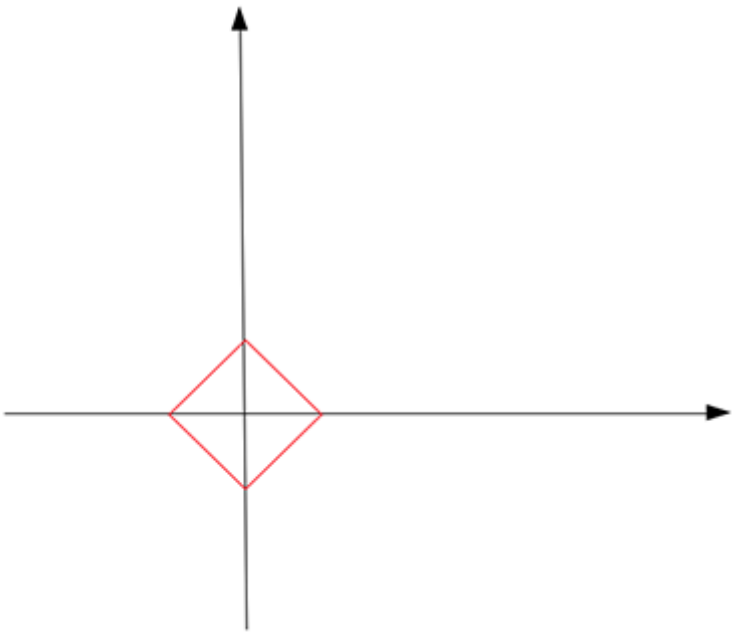
$$[10 \quad 1] \times \begin{pmatrix} a \\ b \end{pmatrix} = (5)$$

由于 $b = 5 - 10a$ ，所以，参数 a, b 的解有无数组 (在蓝线上的点都是解)。

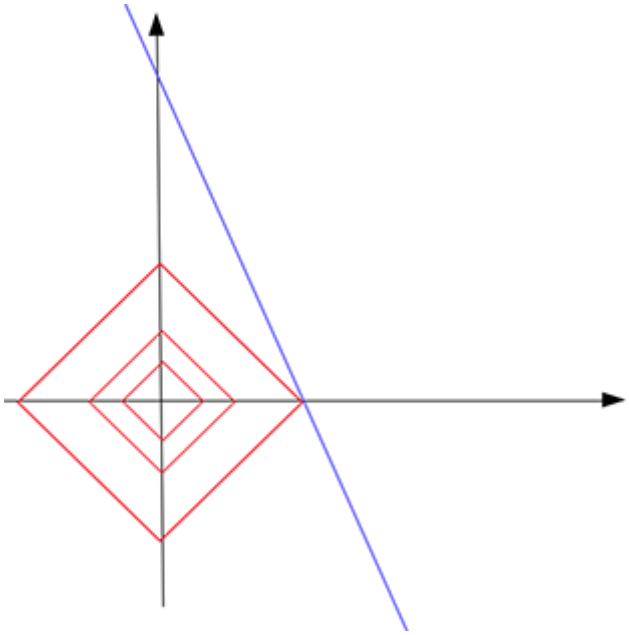


怎样通过 l_1 - 范数找到一个稀疏解呢?

我们不妨先假设向量的 l_1 - 范数是一个常数 $\boldsymbol{c}$ ，如下图:



它的形状是一个正方形 (红色线)，不过在这些边上只有**很少的点是稀疏**的，即**与坐标轴相交的 4 个顶点。**



把红色的正方形（l_1 - 范数为常数）与蓝色的线 (解) 放在同一个坐标系，于是，我们发现蓝线与横轴的交点恰好是满足稀疏性要求的解。同时，这个交点使得 l_1 - 范数取得最小值。

3. 相关参考

参考 1：L1 Norm Regularization and Sparsity Explained

参考 2：l0-Norm, l1-Norm, l2-Norm, … , l-infinity Norm