

章节 1：不容小觑的线性回归算法

从线性回归开始

线性回归是机器学习中有监督机器学习下的一种算法。

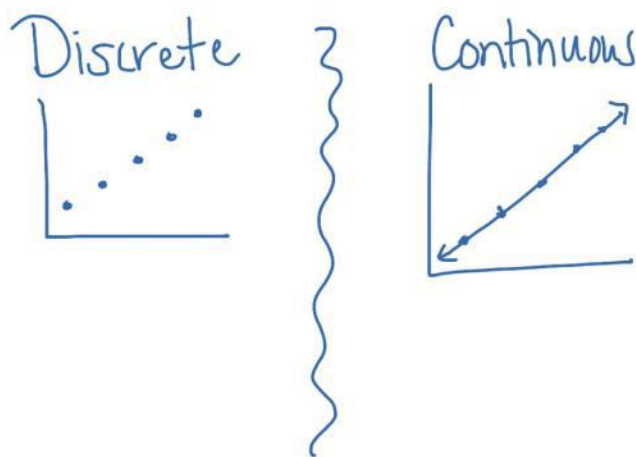
回归问题主要关注确定一个唯一的因变量(dependent variable)(需要预测的值)和一个或多个数值型的自变量(independent variables)(预测变量)之间的关系。

需要预测的值：即目标变量，target，y，连续值

预测变量：影响目标变量的因素，predictors，X1...Xn，可以是连续值也可以是离散值

之间的关系：即模型，model，是我们要求解的

连续值和离散值



简单线性回归

前面提到过，算法说白了就是公式，简单线性回归属于一个算法，它所对应的公式。

$$y=a+bx$$

这个公式中， y 是目标变量即未来要预测的值， x 是影响 y 的因素， a, b 是公式上的参数即要求的模型。其实 a 就是咱们的截距， b 就是斜率嘛！

所以很明显如果模型求出来了，未来影响 y 值的未知数就是一个 x 值，也可以说影响 y 值的因素只有一个，所以这是算法包含“简单”这个词的原因。

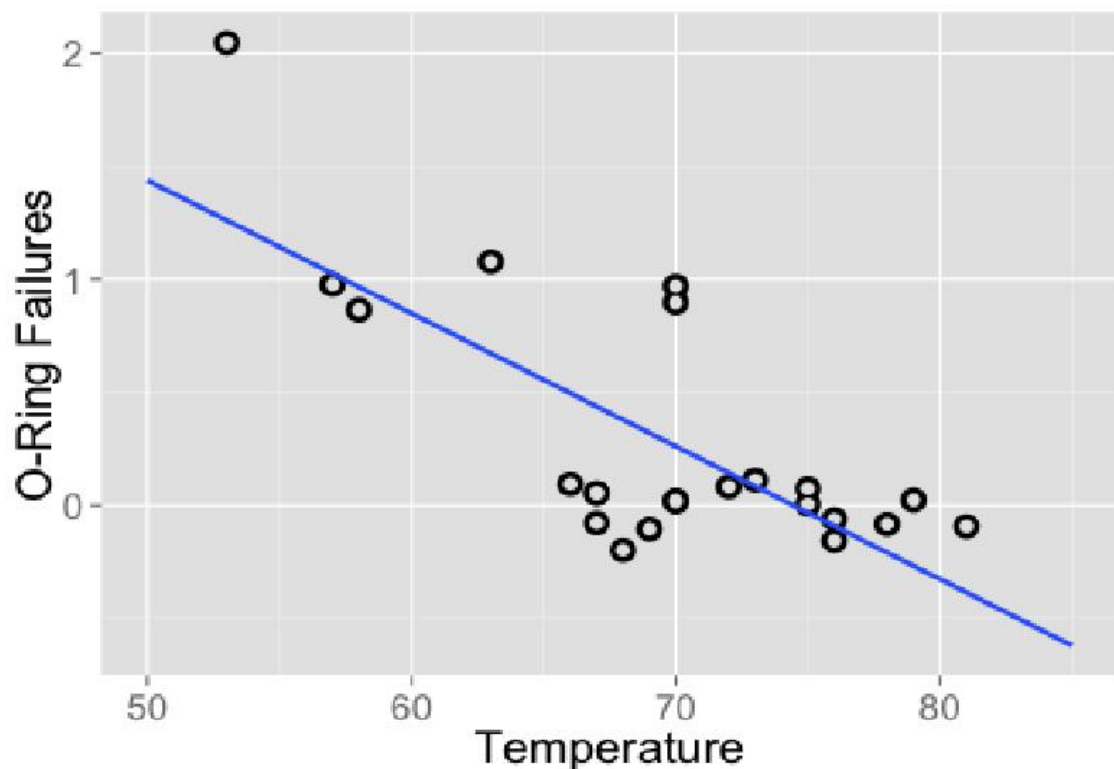
同时可以发现从 x 到 y 的计算， x 只是一次方，所以这是算法叫“线性”回归的原因。

其实，大家上小学时就已经会解这种一元一次方程了。为什么那个时候不叫人工智能算法呢？

因为人工智能算法要求的是最优解！

$y=a+bx$	y	x	a	b
已知条件 1	11	5	1	2
已知条件 2	9	4	1	2
已知条件 3	10	4	2	2

看一个美国人发火箭的例子



最优解

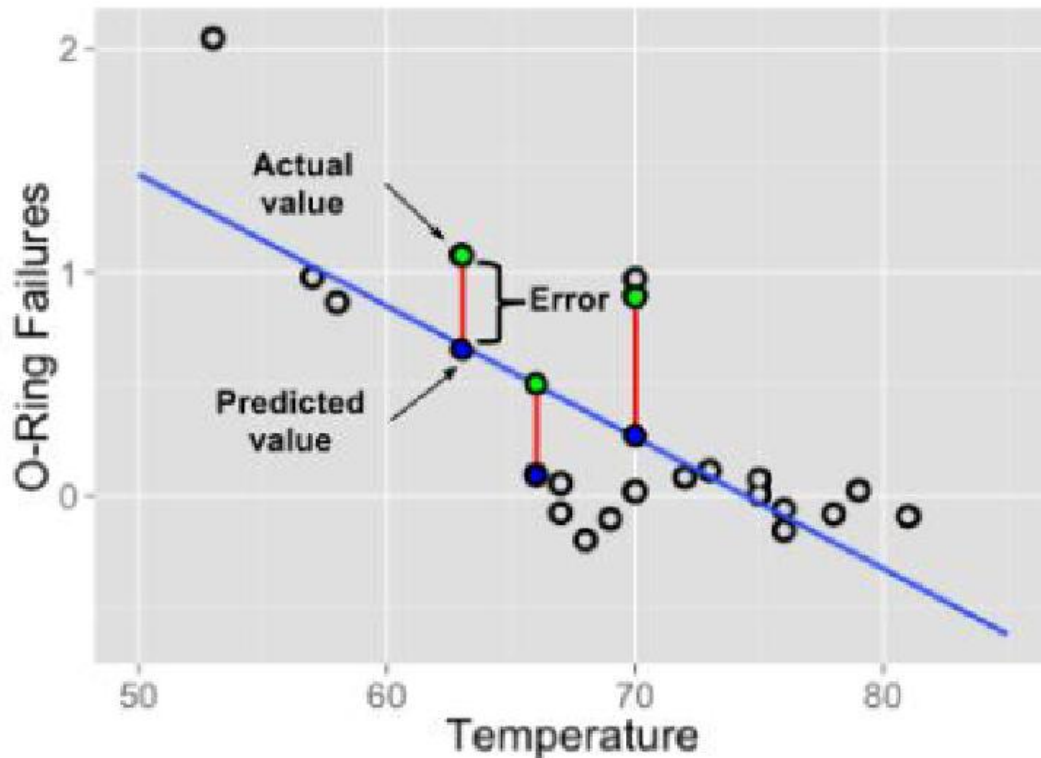
Actual value : 真实值，即已知的 y

Predicted value：预测值，是把已知的 x 带入到公式里面和猜出来的参数 a,b 计算得到的

Error：误差，预测值和真实值的差距

最优解：尽可能的找到一个模型使得整体的误差最小，整体的误差通常叫做损失 Loss

Loss：整体的误差，loss 通过损失函数 loss function 计算得到



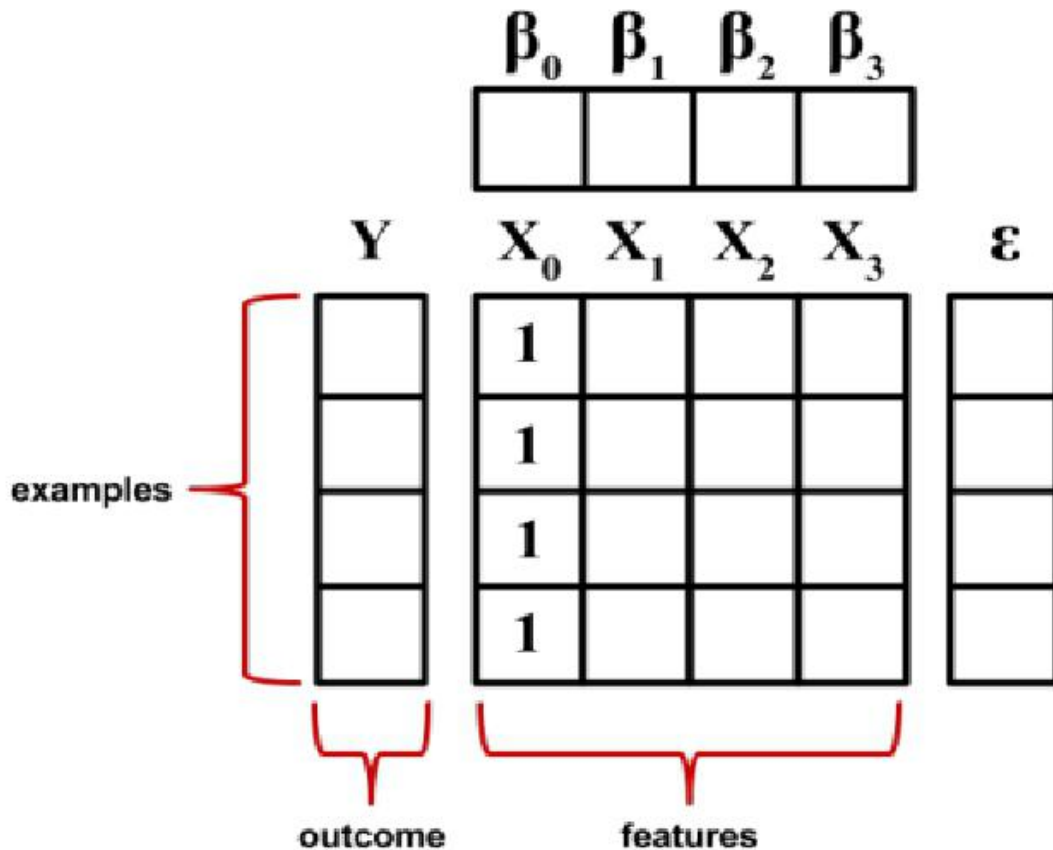
多元线性回归

现实生活中，往往影响结果 y 的因素不止一个，这时 x 就从一个变成了 n 个， $X_1 \dots X_n$ 同时简单线性回归的公式也就不适用了

$$\hat{y} = \beta_0 + \beta_1 X$$

\hat{y} ↑ Predicted value
 β_0 ↑ Intercept
 β_1 ↑ Slope
 X ↑ Predictor

多元线性回归公式



上图中，examples 就是已知的样本，examples 中包含 X 也包含 Y ， Y 就是 outcome 已知结果，如果咱们有 m 条历史记录，就是有 m 条样本，也就是有 m 个 Y 值，或者说 Y 为包含 m 个值的一维向量。

同时每条样本的 X 从 X_1 到 X_n 有 n 个影响结果的因素，图中为了简化相当于 $n=3$ 即有 3 个影响结果的因素，在机器学习中，我们也会把影响结果的因素叫特征 feature，因为有多 个所以图里就是 features，值得一提的是 X_0 一列，是为了后面可以通过公式计算出截距项而加的，同时会把 X_0 一列所有值设置恒为 1，这样 X 就是 m 行 4 列的二维数组即矩阵。

图中 ϵ 代表 error 误差，每条样本预测的值和真实值之间都会有误差，所以有 m 条样本就对应 m 个 ϵ 值， ϵ 和 Y 一样是包含 m 个值得一维向量。

最后图中还出现了 β 符号，从 0 到 n ， $n=3$ ，总共有 4 个，其实细心的你会发现这个正好和特征数量一样，我们可以理解或叫做这是特征的权值，代表对应特征的重要程度，也叫权重，英文 weights，进而后面课程中也会用符号 W 代替 β 。

举例：

（比如我们要训练一个模型未来判别一个人有多漂亮，给打个分，历史数据是人的一些指标和已知得分，那么如果有 1000 个人的数据，就是 1000 个 examples，那么 $m=1000$ 。 Y 里面就存放 1000 个已知分数。同时一个人有哪些特征啊？比如有鼻子，有眼睛，有收入，有穿衣服，那么鼻子、眼睛、收入、衣服等就是特征，每个人到底是高鼻梁还是塌鼻子，眼睛什么颜色的，收入情况多少，衣服款式什么样的等就是具体的特征值。我们要算的 W 权值就是把这些特征所对应的重要程度计算出来，未来就可以拿到一个人的特征值去计算具体

的分数了。)

$$Y = a + bx \quad (1)$$

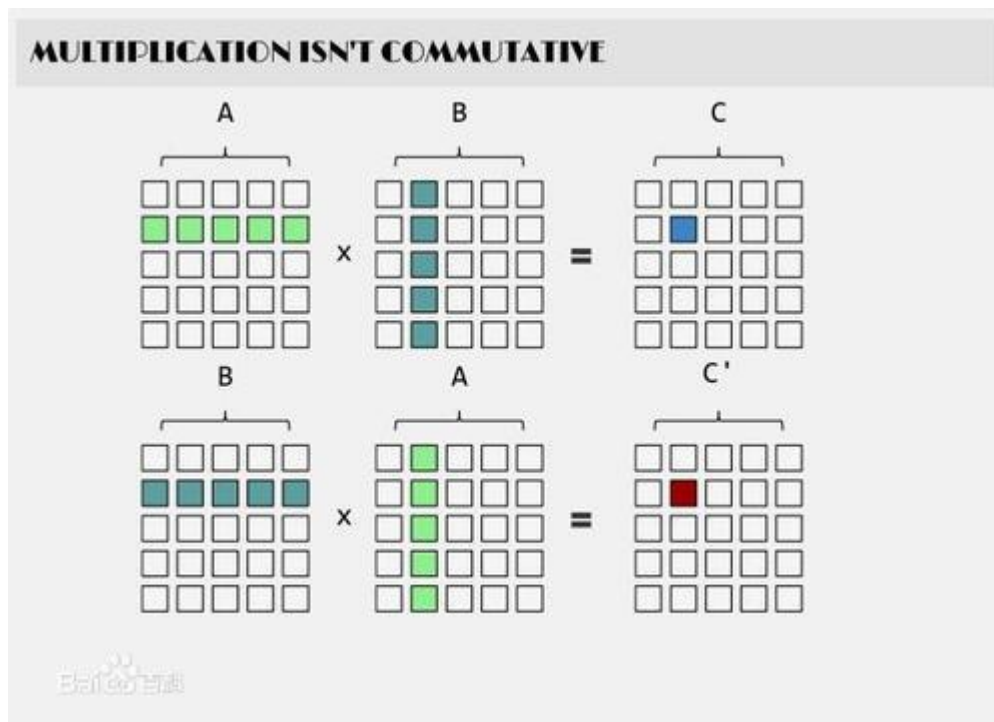
$$y = a + bx + \epsilon \quad (2)$$

$$y = \beta_0 + \beta_1 x_1 + \dots + \beta_n x_n + \epsilon \quad (3)$$

在多元线性回归中 W 是一维向量，代表的是 W_0 到 W_n ，我们也可以用线性代数的方式去表达公式，这时算法要求解的就是这个向量，如果维度很多我们当然需要计算机帮助我们求解了。

$$y = W^T X + \epsilon \quad y = XW + \epsilon$$

同学们了解矩阵相乘的计算方式就很好理解上面的式子等级与之前的相乘相加!!



章节 2：深入线性回归算法的推导

深入理解线性回归

前面我们通过讲线性回归相信大家已经理解了回归任务是做什么的,但是还不知道具体怎么做,就是说怎么求出最优解,为了透彻理解我们必须再补充一些概念,只有有了这些概念我们后面才能推导出线性回归所需要的损失函数,进而去进一步理解最优解该如何去求。

理解回归一词来源

回归简单来说就是“回归平均值”(regression to the mean)。但是这里的 mean 并不是把历史数据直接当成未来的预测值,而是会把期望值当作预测值。至于原因请允许我娓娓道来。追根溯源回归这个词是一个叫高尔顿的人发明的,他通过大量观察数据发现:

父亲是比较高的,儿子也是比较高的

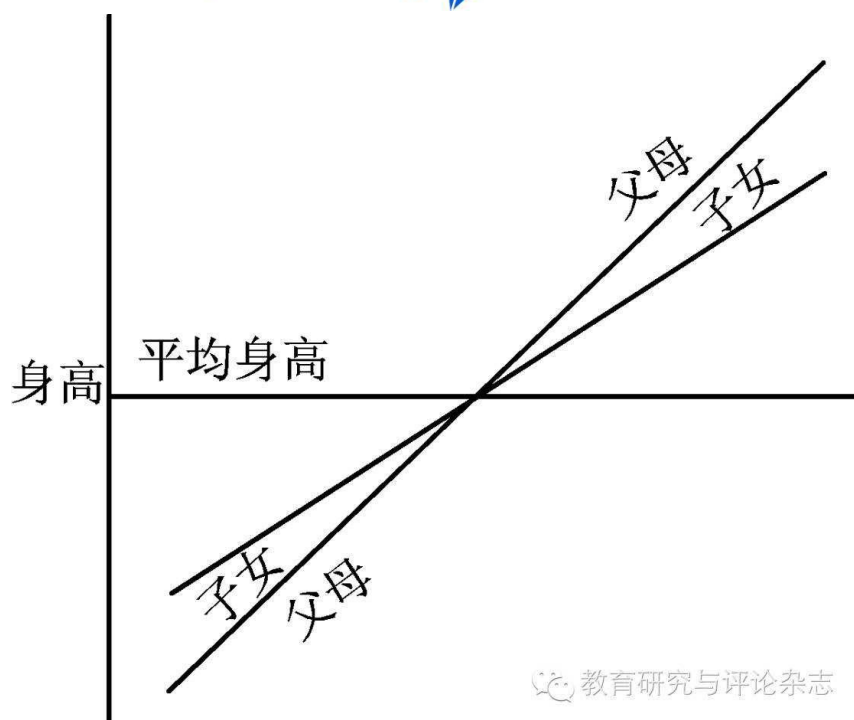
父亲是比较矮的,儿子也是比较矮的

父亲是 2.26,儿子可能很高但是不会达到 2.26

父亲是 1.65,儿子可能不高,但是比 1.65 高

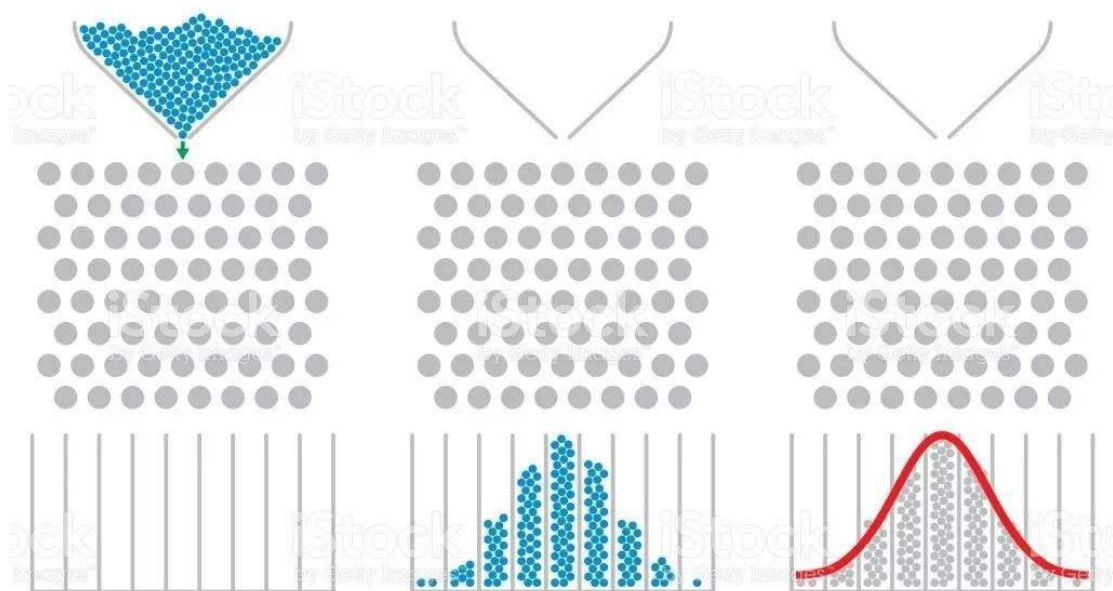
大自然让我们回归到一定的区间之内

高尔顿是谁?达尔文的表哥,这下可以相信他说的八成是对的了吧。



中心极限定理

高尔顿钉板



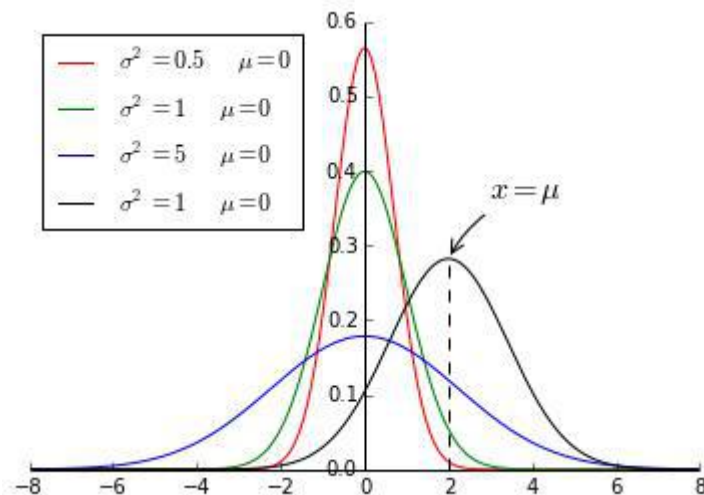
中心极限定理 (central limit theorem) 是概率论中讨论**随机变量**序列部分和分布渐近于正态分布的一类定理。这组定理是数理统计学和误差分析的理论基础，指出了大量随机变量累积分布函数逐点收敛到正态分布的累积分布函数的条件。

它是概率论中最重要的一类定理，有广泛的实际应用背景。在自然界与生产中，一些现象受

到许多相互独立的随机因素的影响,如果每个因素所产生的影响都很微小时,总的影响可以看作是服从正态分布的。中心极限定理就是从数学上证明了这一现象。

正太分布与预测的关系

也叫高斯分布



举例：足球队身高的例子，篮球队身高的例子，预测前提就是首先知道我们的数据集更服从哪种分布

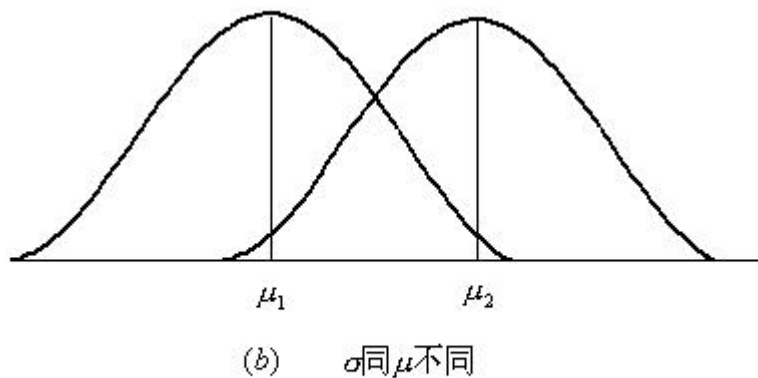


图 1 正态分布的两个参数

如果我们有一组身高的数据,从上图我们是可以直观上看起来会服从哪一个分布,但是计算机怎么知道?它必须要通过计算数值比大小才能知道,关键比较大小的这个数值该怎么算呢?

这个时候,如果我们知道一个样本目标变量即一个人的身高在篮球队出现的概率,同时如果也知道这个样本目标变量即还是那个身高在足球队出现的概率,我们通过概率值就可以知道这个人更有可能是打篮球的,还是踢足球的。

那如果我们把问题扩展到我们的所有样本呢?那问题就变成了去看这组样本是来自于篮球

队的还是来自于足球队的问题了。这里我们就需要有更科学的猜也就是估计的方法了。
还有一个问题,那就是仔细想会发现我们好像并不能一开始就很确定我们的一组数据是随机出现的,并且互相独立的,从而去假设它们呈现正太分布?

再理解一遍误差

再讨论误差的目的是为了我们先来回答后一个问题的解决方法。

第 i 个样本实际的值 y_i 等于 预测的值 \hat{y}_i 加 误差 ε_i , 或者公式可以表达为如下

$$\varepsilon_i = y_i - \hat{y}_i$$

假定所有的样本的误差都是独立的,有上下的震荡,震荡认为是随机变量,足够多的随机变量叠加之后形成的分布,根据中心极限定理,它服从的就是正态分布,因为它是正常状态下的分布,也就是高斯分布!均值是某一个值,方差是某一个值。

方差我们先不管,均值我们总有办法让它去等于零 0 的,因为我们这里是有 W_0 截距的,所有误差我们就可以认为是独立分布的, $1 \leq i \leq m$, 服从均值为 0, 方差为某定值的高斯分布。

机器学习中我们假设误差符合均值为 0, 方差为定值的正态分布!!

可以举例北京不同区县房价的误差,来理解我们假设它是 互相独立, 随机变量 的合理性!

最大似然估计

为了回答前一个问题的解决方法,我们来学习一下最大似然估计。

在统计学中,最大似然估计(英语:maximum likelihood estimation, 缩写为 MLE), 也称最大概似估计,是用来估计一个概率模型的参数的一种方法。这个方法最早是遗传学家以及统计学家罗纳德·费雪 fisher 爵士在 1912 年至 1922 年间开始使用的。“似然”是对 likelihood 的一种较为贴近文言文的翻译,“似然”用现代的中文来说即“可能性”。故而,若称之为“最大可能性估计”则更加通俗易懂。在英语语境里,likelihood 和 probability 的日常使用是可以互换的,都表示对机会(chance)的同义替代。

给定一个概率分布 D , 已知其概率密度函数(连续分布)或概率质量函数(离散分布)为 f_D , 以及一个分布参数 θ 我们可以从这个分布中抽出一个具有 n 个值的采样 X_1, X_2, \dots, X_n , 利用 f_D 计算出其似然函数:

$$L(\theta | x_1, \dots, x_n) = f_{\theta}(x_1, \dots, x_n).$$

若 D 是离散分布, f_{θ} 即是在参数为 θ 时观测到这一采样的概率。若其是连续分布, f_{θ} 则为 X_1, X_2, \dots, X_n 联合分布的概率密度函数在观测值处的取值。一旦我们获得 X_1, X_2, \dots, X_n , 我

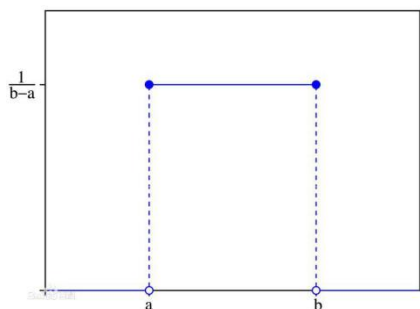
们就能求得一个关于 θ 的估计。最大似然估计会寻找关于 θ 的最可能的值（即，在所有可能的 θ 取值中，寻找一个值使这个采样的“可能性”最大化）。从数学上来说，我们可以在 θ 的所有可能取值中寻找一个值使得似然函数取到最大值。这个使可能性最大的 θ 值即称为 θ 的最大似然估计。由定义，最大似然估计是关于样本的函数。

因为我们前面说了既然世间万物很多事情都服从中心极限定理，而机器学习中就假设了数据预测的误差服从正太分布，很明显正太分布是连续的分布，所以故而需要误差对应的正太分布的概率密度函数。

概率密度函数

在数学中，连续型随机变量的概率密度函数是一个描述这个随机变量的输出值，在某个确定的取值点附近的可能性的函数。而随机变量的取值落在某个区域之内的概率则为概率密度函数在这个区域上的积分。

最简单的概率密度函数是均匀分布的密度函数。最简单的概率密度函数是均匀分布的密度函数。也就是说，当 x 不在区间 a, b 上的时候，函数值等于 0；而在区间 a, b 上的时候，函数值等于这个函数。这个函数并不是完全的连续函数，但是是可积函数。



最常见的连续概率分布是正态分布，而这正是我们所需要的，其概率密度函数如下：

$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

随着参数 μ 和 σ 变化，概率分布也产生变化。

下面重要的步骤来了，我们要把一组数据误差出现的总似然，也就是一组数据之所以对应误差出现的整体可能性表达出来了，因为数据的误差我们假设服从一个正太分布，并且通过截距项来本质上平移整体分布的位置从而使得 $\mu=0$ ，所以对于一条样本的误差我们可以表达其概率密度函数的值为如下：

$$f(\varepsilon_i | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\varepsilon_i-0)^2}{2\sigma^2}}$$

正太分布的线性回归的最大总似然

接下来我们就是要将最大似然函数通过正太分布概率密度函数表达出来

$$L_{\theta}(\varepsilon_1, \dots, \varepsilon_m) = f(\varepsilon_1, \dots, \varepsilon_m \mid \mu, \sigma^2)$$

这时，因为我们假设了误差服从正太分布，符合中心极限定理，那么也就是样本误差服从了互相独立的假设，所以我们可以把上面式子写出连乘的形式

关于独立为什么可以连乘，大家回想一下关于概率的公式

$$P(A \cap B) = P(A) \cdot P(B)$$

$$f(\varepsilon_1, \dots, \varepsilon_m \mid \mu, \sigma^2) = f(\varepsilon_1 \mid \mu, \sigma^2) * f(\varepsilon_2 \mid \mu, \sigma^2) * \dots * f(\varepsilon_m \mid \mu, \sigma^2)$$

所以

$$L_{\theta}(\varepsilon_1, \dots, \varepsilon_m) = \prod_{i=1}^m f(\varepsilon_i \mid \mu, \sigma^2) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\varepsilon_i - 0)^2}{2\sigma^2}}$$

因为我们现在讲的是线性回归，所以误差函数可以写为如下：

$$\varepsilon_i = \hat{y}_i - y_i = |y_i - W^T x_i| = |y_i - \theta^T x_i|$$

从上式中我们可以看出来，这样我们的历史数据中的 X 和 y 就都可以被用上去求解了

所以正太分布假设下的最大似然估计函数可以写成如下：

$$L_{\theta}(\varepsilon_1, \dots, \varepsilon_m) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\varepsilon_i - 0)^2}{2\sigma^2}} = \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}}$$

推导出线性回归损失函数 MSE

明确目标

下面我们要推导出线性回归损失函数，为什么要干这件事？因为第一章里面我们说过要去求解出最优解，我们往往干的事情就是最小化损失函数。所以我们必须首先知道这个算法对应的损失函数是什么？

上面我们已经有了总似然的表达公式，而我们也有最大总似然这种数学思想，所以我们可以

先往后沿着把总似然最大化这个思路继续看看会发生什么。

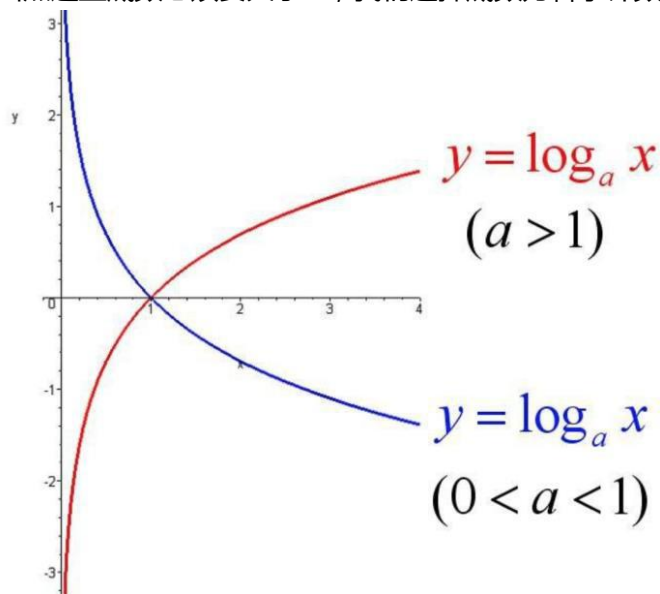
说白了，最大似然估计就是一种参数估计的方式，就是把总似然最大的那一时刻对应的参数 θ 当成是要求的最优解！

$$\arg \max_{\theta} L_{\theta}(\varepsilon_1, \dots, \varepsilon_m) = \arg \max_{\theta} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}}$$

这时我们就可以把通过最大化似然函数的形式成为我们的目标函数，因为我们的目标就是最大化这个式子从而求解 θ

对数似然函数

首先我们了解一下 \log 对数函数的特性，我们可以发现它的特点是当底数 $a > 1$ 时，它是一个单调递增的函数，单调递增怎么了？很棒！意味着如果 $x_1 < x_2$ ，那么必然 $y_1 < y_2$ ，更棒的是，我们上面的式子是要找出总似然最大时对应的 θ 是多少，所以是不是就意味着等价于找出总似然的对数形式最大时对应的 θ 是多少呢！！！必须的，求出来的 θ 一定是一样的。当然这里底数必须要大于 1，我们选择底数为科学计数 e ，至于原因后面马上就知道了。



$$\arg \max_{\theta} L_{\theta}(\varepsilon_1, \dots, \varepsilon_m) = \arg \max_{\theta} \log_e \left(\prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}} \right)$$

接下来 \log 函数继续为我们带来惊喜，数学上连乘是个大麻烦，即使交给计算机去求解它也得哭出声来。惊喜是：

积、商、幂的对数运算法则：

如果 $a > 0$ ，且 $a \neq 1$ ， $M > 0$ ， $N > 0$ 有：

$$\log_a(MN) = \log_a M + \log_a N$$

$$\log_a \frac{M}{N} = \log_a M - \log_a N$$

$$\log_a M^n = n \log_a M (n \in R)$$

$$\log_a (M_1 M_2 \cdots M_n) = \log_a M_1 + \log_a M_2 + \cdots + \log_a M_n$$

$$\log_a a^n = n (n \in R) \quad \log_a \frac{1}{M} = -\log_a M$$

$$\log_a \sqrt[n]{M^p} = \log_a M^{\frac{p}{n}} = \frac{p}{n} \log_a M$$

$$\begin{aligned} l(\theta) &= \arg \max_{\theta} \log_e \left(\prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}} \right) \\ &= \arg \max_{\theta} \sum_{i=1}^m \log_e \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}} \right) \end{aligned}$$

继续往后推导出损失函数 MSE

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma}} \exp \left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2} \right) \\ &= \sum_{i=1}^m \log \frac{1}{\sqrt{2\pi\sigma}} \exp \left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2} \right) \\ &= m \log \frac{1}{\sqrt{2\pi\sigma}} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2 \\ J(\theta) &= \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \end{aligned}$$

补充说明：

$$\hat{y}^{(i)} = h_{\theta}(x^{(i)}) = x^i \theta$$

$$\hat{y} = h_{\theta}(X) = X\theta$$

因为前面有个负号，所以最大总似然变成了最小话负号后面的部分。

到这里，我们就已经推导出来了 MSE 损失函数，从公式我们也可以看出来 MSE 名字的来历，mean squared error，上式也叫做最小二乘。

总结与扩展

这种最小二乘估计，其实我们就可以认为，假定了误差服从正太分布，认为样本误差的出现是随机的，独立的，使用最大似然估计思想，利用损失函数最小化 MSE 就能求出最优解！所以反过来说，如果我们的数据误差不是互相独立的，或者不是随机出现的，那么就不适合去假设为正太分布，就不能去用正太分布的概率密度函数带入到总似然的函数中，故而说白了就不能用 MSE 作为损失函数去求解最优解了。

还有譬如假设误差服从泊松分布，或其他分布那就得用其他分布的概率密度函数去推导出损失函数了。

所以有时我们也可以把线性回归看成是广义线性回归，General Linear Model。比如，逻辑回归，泊松回归都属于广义线性回归的一种，这里我们线性回归可以说是最小二乘线性回归。

章节 3：解析解方法求解线性回归

解析解的推导

我们现在有了损失函数形式，也明确了目标就是要最小化损失函数，那么接下来问题就是 theta 什么时候可以使得损失函数最小了。

最小二乘形式变化个写法

我们先把损失函数变化个形式，

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 = \frac{1}{2} (X\theta - y)^T (X\theta - y)$$

补充说明：

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})(h_{\theta}(x^{(i)}) - y^{(i)}) \end{aligned}$$

这里就等价于一个长度为 m 向量乘以它自己，说白了就是对应位置相乘相加

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = a_1 b_1 + a_2 b_2 + \cdots + a_{n-1} b_{n-1} + a_n b_n.$$

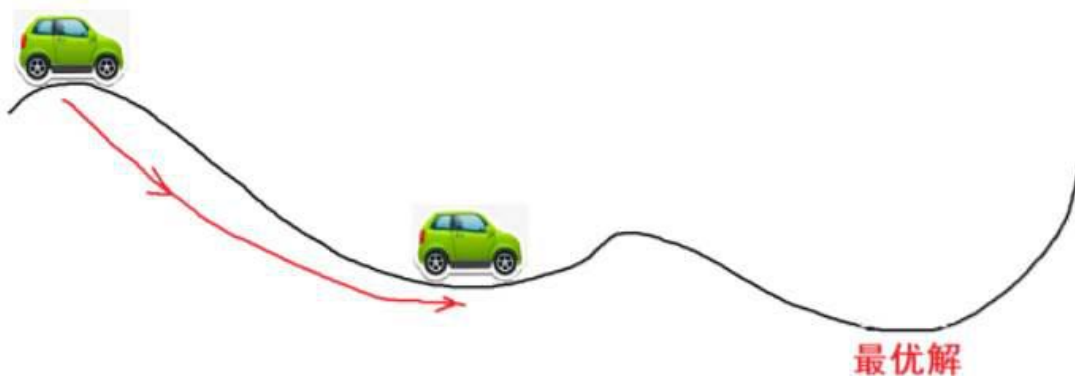
用连加号写：

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i.$$

$$\begin{aligned}
 J(\theta) &= \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \\
 &= \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})(h_{\theta}(x^{(i)}) - y^{(i)}) \\
 &= \frac{1}{2} (X\theta - y)^T (X\theta - y) \\
 &= \frac{1}{2} ((X\theta)^T - y^T)(X\theta - y) \\
 &= \frac{1}{2} (\theta^T X^T - y^T)(X\theta - y) \\
 &= \frac{1}{2} (\theta^T X^T X\theta - \theta^T X^T y - y^T X\theta + y^T y)
 \end{aligned}$$

推导出 θ 的解析解形式

为了方便理解，大家可以把下图的横轴看成是 θ 轴，纵轴看成是 loss 损失，曲线是 loss function，然后你开着小车去寻找最优解



如果我们把最小二乘看成是一个函数曲线，极小值（最优解）一定是个驻点，驻点顾名思义就是可以停驻的点，而图中你可以看出驻点的特点是统统梯度为 0

梯度：函数在某点上的切线的斜率

如何求？求函数在某个驻点上的一阶导数即为切线的斜率

更进一步，或者反过来说，就是我们是不是可以把函数的一阶导函数形式推导出来

$$\begin{aligned} J'(\theta) &= \left[\frac{1}{2} (\theta^T X^T X \theta - \theta^T X^T y - y^T X \theta + y^T y) \right]' \\ &= \frac{1}{2} [(\theta^T X^T X \theta)' - (\theta^T X^T y)' - (y^T X \theta)' + (y^T y)'] \end{aligned}$$

我们需要明确的是，我们的已知是 X 和 y ，未知是 θ ，所以和 θ 没关系的部分求导都可以忽略不记

又因为求导公式：

$$(kA)^T = kA^T$$

$$(A + B)^T = A^T + B^T$$

$$(AB)^T = B^T A^T$$

$$(A^T)^T = A$$

$$\frac{\partial \theta^T A \theta}{\partial \theta} = 2A\theta$$

$$\frac{\partial \theta^T A}{\partial \theta} = A$$

$$\frac{\partial A \theta}{\partial \theta} = A^T$$

$$\begin{aligned} J'(\theta) &= \frac{1}{2} [(\theta^T X^T X \theta)' - (\theta^T X^T y)' - (y^T X \theta)' + (y^T y)'] \\ &= \frac{1}{2} [2X^T X \theta - X^T y - (y^T X)^T] \\ &= \frac{1}{2} [2X^T X \theta - 2X^T y] \\ &= X^T X \theta - X^T y \\ &= 0 \end{aligned}$$

然后设置导函数为 0，去进一步解出来驻点对应的 θ 值为多少

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

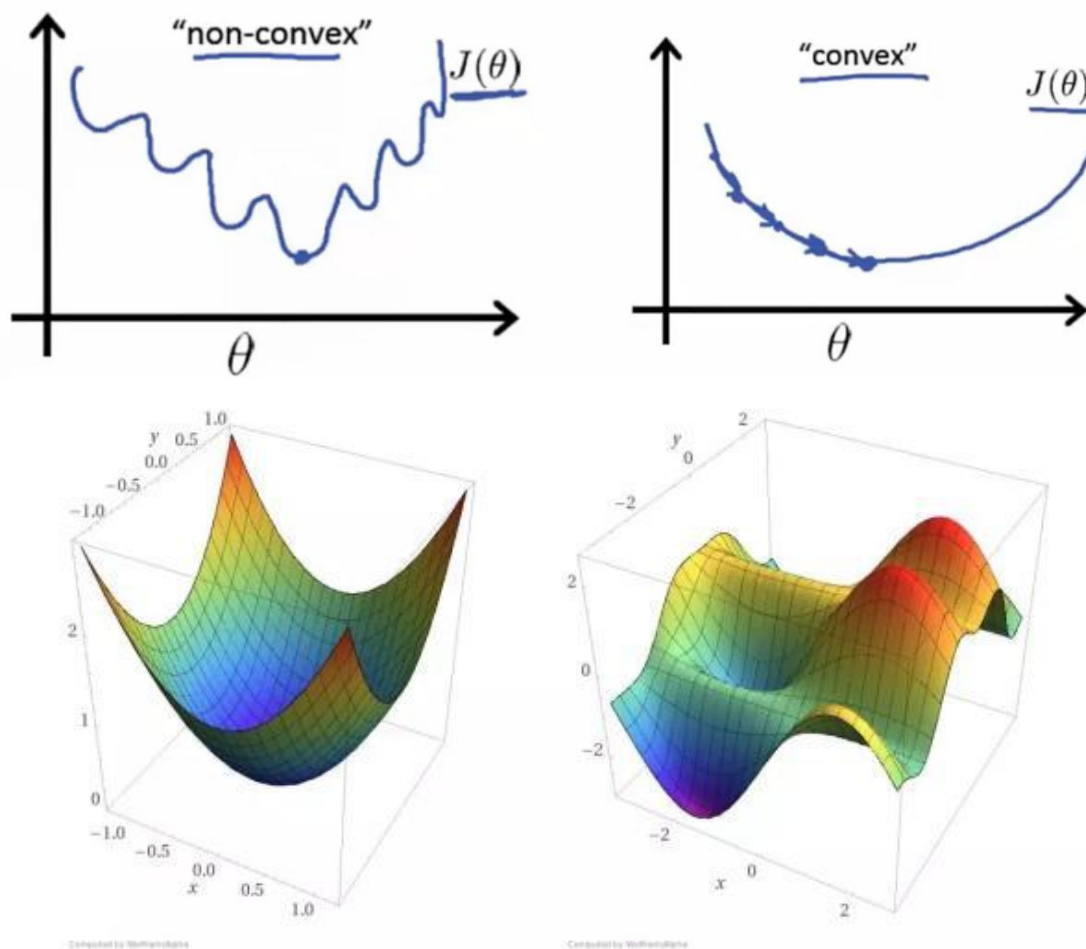
数值解是在一定条件下通过某种近似计算得出来的一个数值,能在给定的精度条件下满足方程 解析解为方程的解析式(比如求根公式之类的),是方程的精确解,能在任意精度下满足方程

这样,如果有数据集 X, y 时,我们就可以带入上面解析解公式,去直接求出对应的 θ 值了,比如我们可以设想 X 为 m 行 n 列的矩阵, y 为 m 行 1 列的列向量

X^T 就是 n 行 m 列的矩阵,然后 $X^T X$ 就是 n 行 n 列的矩阵,矩阵求逆形状不变,然后继续乘以 X^T 后形状变为 n 行 m 列的矩阵,最后乘以 y ,结果 θ 就是 n 行 1 列的列向量!

判定损失函数凸函数

判定损失函数是凸函数的好处在于我们可能很肯定的知道我们求得的极值即最优解,一定是全局最优解



判定凸函数的方式:

判定凸函数的方式非常多,其中一个方法是看黑塞矩阵是否是半正定的。

黑塞矩阵 (hessian matrix) 是由目标函数在点 X 处的二阶偏导数组成的对称矩阵

对于我们的式子来说就是在导函数的基础上再次对 θ 来求偏导，说白了不就是 $X^T X$ 所谓正定就是 A 的特征值全为正数，那么是正定的。半正定就是 A 的特征值大于等于 0，就是半正定。

这里我们对 J 损失函数求二阶导的黑塞矩阵是 $X^T X$ ，之后得到的一定是半正定的，自己和自己做点乘嘛！

此处不用深入去找数学推导证明这一点，还有就是机器学习中往往损失函数都是凸函数，到深度学习中损失函数往往是非凸函数，即找到的解未必是全局最优，只要模型堪用就好！

ML 学习特点，不强调模型 100% 正确，是有价值的，堪用的！

章节 4：算法开发环境的搭建与测试

Python 语言

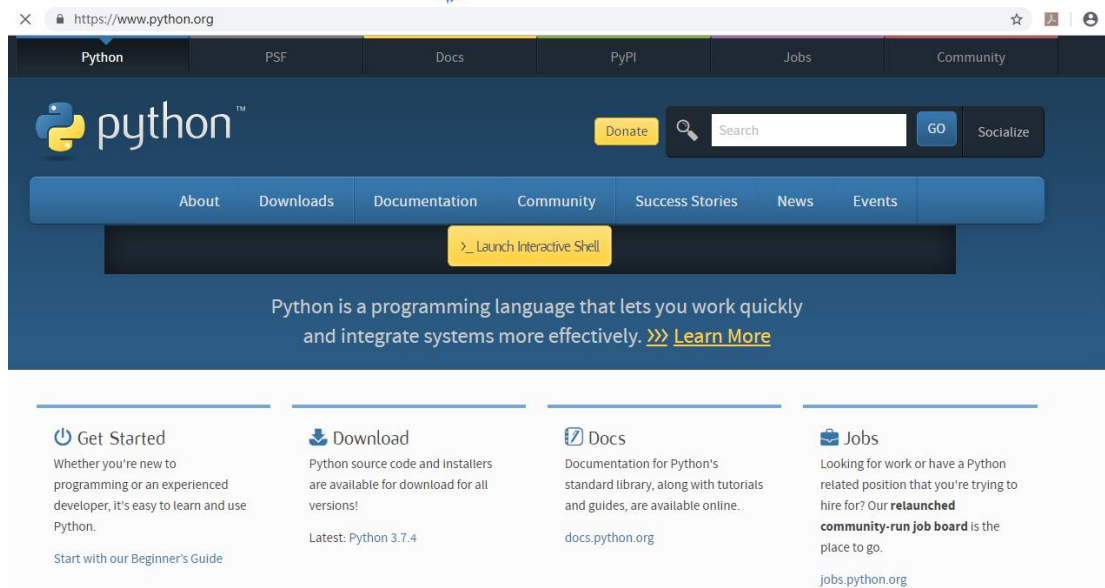
在公司里面做为 AI 算法工程师，大部分公司工作时都要求用 python 语言，而 python 语言也随着近些年 AI 的发展流行度在不断的攀升。我们从语言流行度排行榜上面就可以看出来。

<https://www.tiobe.com/tiobe-index/>

Anaconda 环境安装

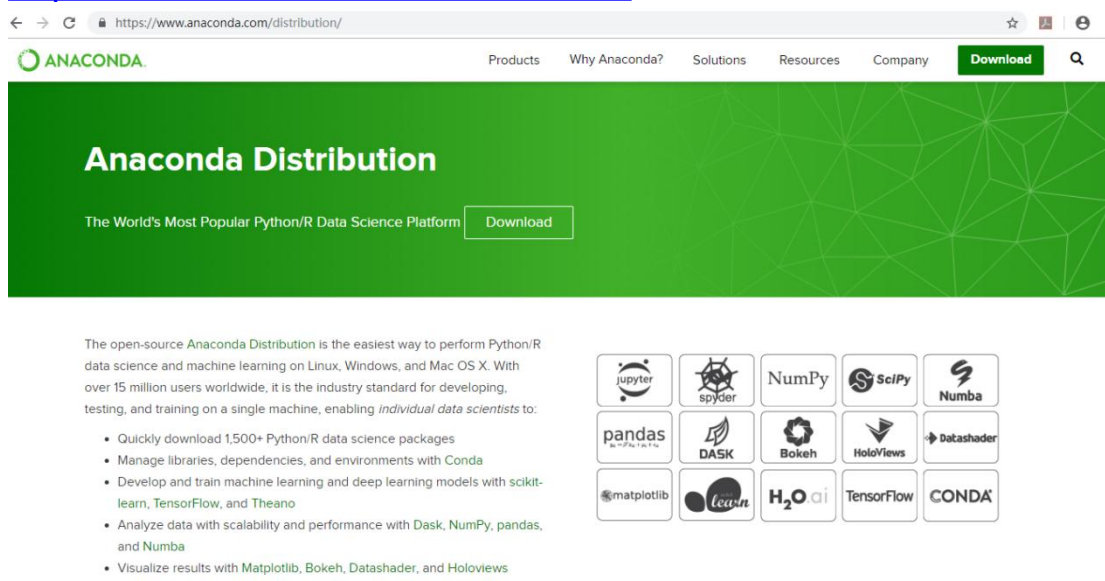
这里不推荐安装原生的 python 环境，如果电脑中已有 python 环境也没关系，只是后期需要利用 pip install 比较多的模块

<https://www.python.org/>



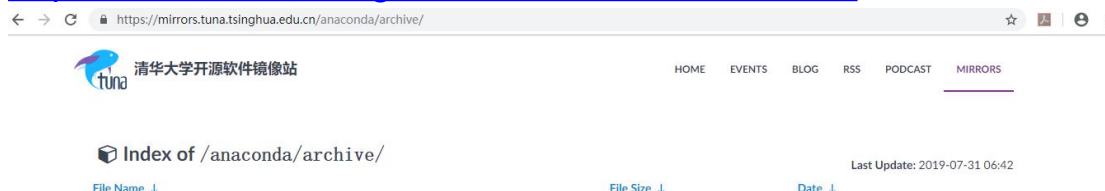
更推荐是安装 Anaconda 环境，首先 Anaconda 环境本身已经包含了原生的 python 环境所有功能，其次是 Anaconda 环境在 python 环境基础上还附带了许多 AI 所需要的模块，而且互相之间的版本兼容是比较不错的，公司里做算法一般都会安装 Anaconda

<https://www.anaconda.com/distribution/>



从官网下载或许会很慢，可以从清华镜像去下载

<https://mirrors.tuna.tsinghua.edu.cn/anaconda/archive/>



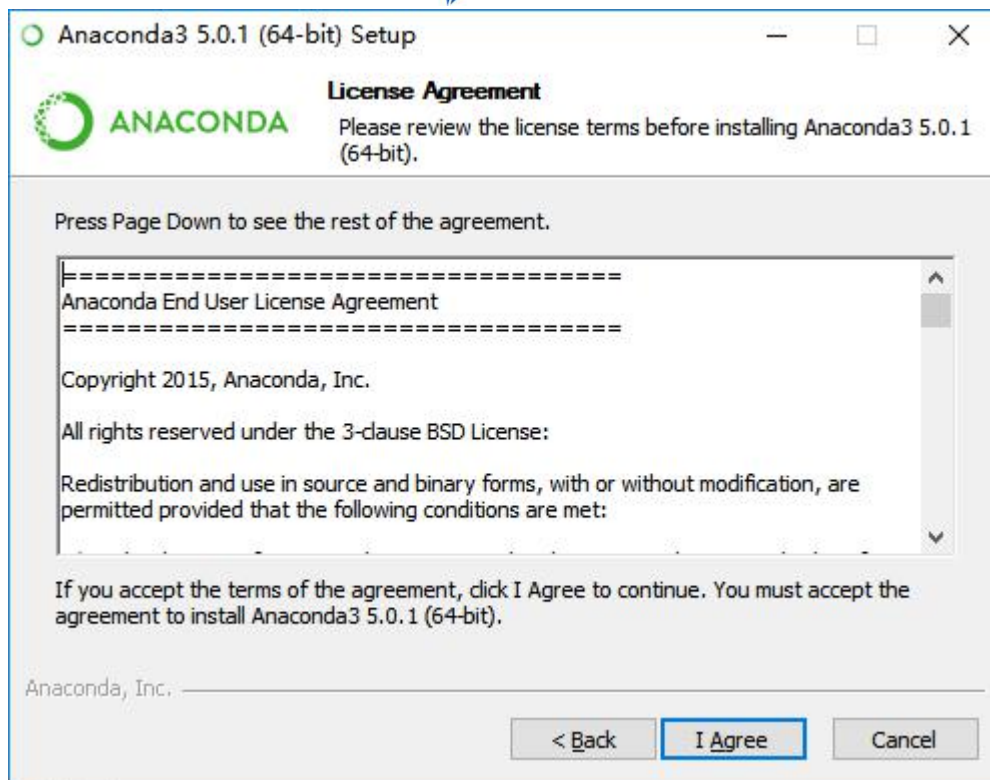
Anaconda3-5.0.0-MacOSX-x86_64.sh	489.9 MiB	2017-09-27 05:34
Anaconda3-5.0.0-Windows-x86.exe	415.8 MiB	2017-09-27 05:34
Anaconda3-5.0.0-Windows-x86_64.exe	510.0 MiB	2017-09-27 06:17
Anaconda3-5.0.0-Linux-x86.sh	429.8 MiB	2017-10-03 00:33
Anaconda3-5.0.0-Linux-x86_64.sh	524.0 MiB	2017-10-03 00:34
Anaconda3-5.0.1-Linux-x86.sh	431.0 MiB	2017-10-26 00:41
Anaconda3-5.0.1-Linux-x86_64.sh	525.3 MiB	2017-10-26 00:42
Anaconda3-5.0.1-MacOSX-x86_64.pkg	568.9 MiB	2017-10-26 00:42
Anaconda3-5.0.1-MacOSX-x86_64.sh	491.0 MiB	2017-10-26 00:42
Anaconda3-5.0.1-Windows-x86.exe	420.4 MiB	2017-10-26 00:44
Anaconda3-5.0.1-Windows-x86_64.exe	514.8 MiB	2017-10-26 00:45
Anaconda3-5.1.0-Linux-ppc64le.sh	285.7 MiB	2018-02-15 23:22
Anaconda3-5.1.0-Linux-x86.sh	449.7 MiB	2018-02-15 23:23
Anaconda3-5.1.0-Linux-x86_64.sh	551.2 MiB	2018-02-15 23:24
Anaconda3-5.1.0-MacOSX-x86_64.pkg	594.7 MiB	2018-02-15 23:24
Anaconda3-5.1.0-MacOSX-x86_64.sh	511.3 MiB	2018-02-15 23:24

🟢 Anaconda3-5.0.1-Windows-x86_64.exe

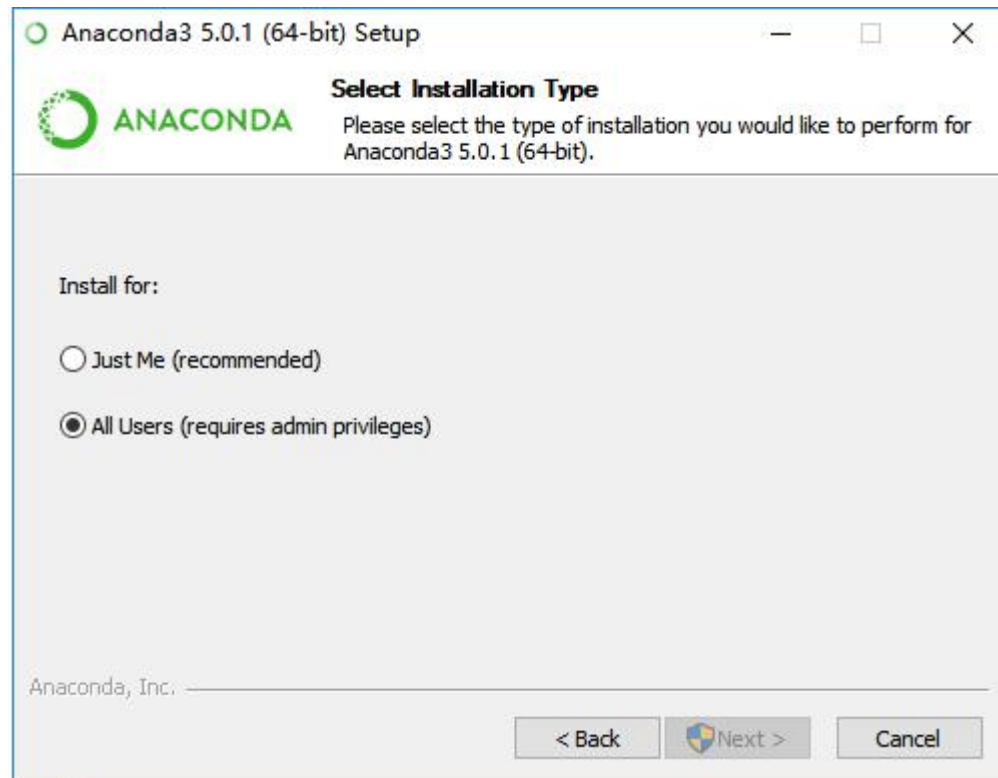
这是一个对应 Windows 版本的 Anaconda 安装包 老师这里安装教程是对应 Windows10 的环境，这个版本对应的 python3.6 环境，即使你选择用的安装包是 Anaconda3 更高的版本，内部对应 python3.7 的版本也没有关系，至于为什么没有关系后面会讲到。

First，双击打开安装文件



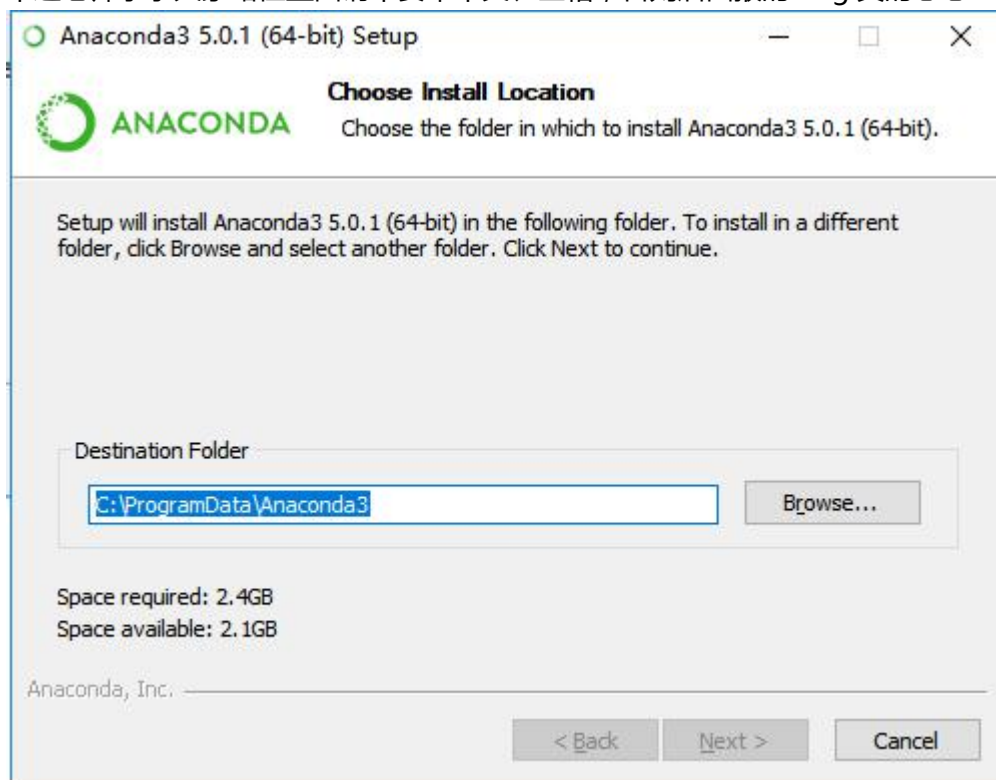


Second，这里选择 “Just Me” 或 “All Users” 其实都是可以的，只要安装后记住路径就好

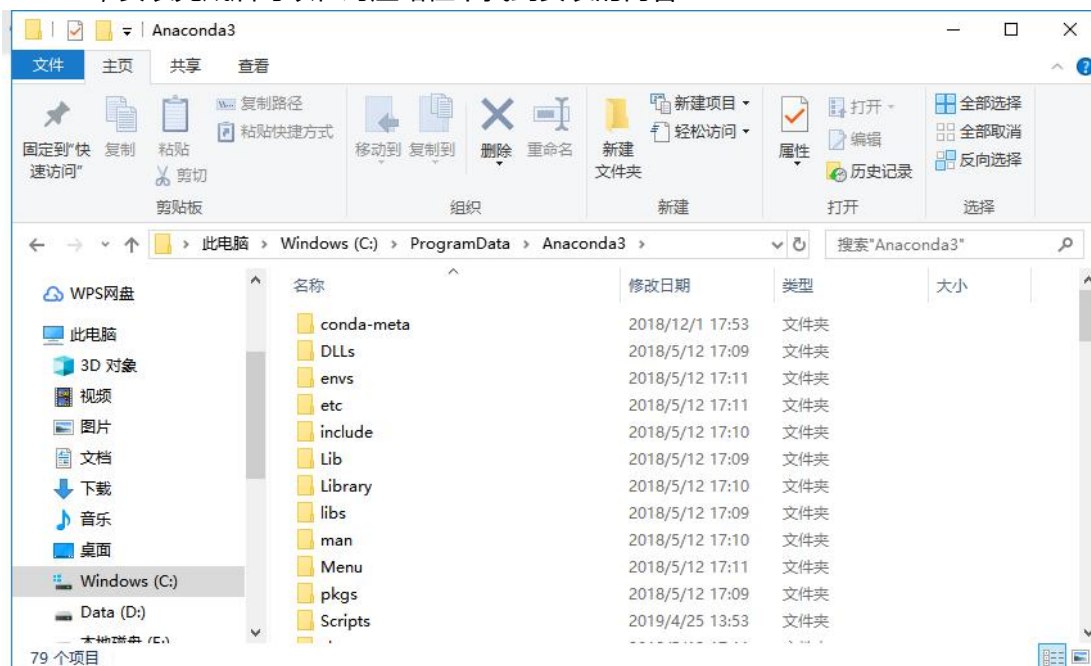


Third，老师这里已经安装过，所以 next 被置灰，如果同学们没有安装过可以一路 next，

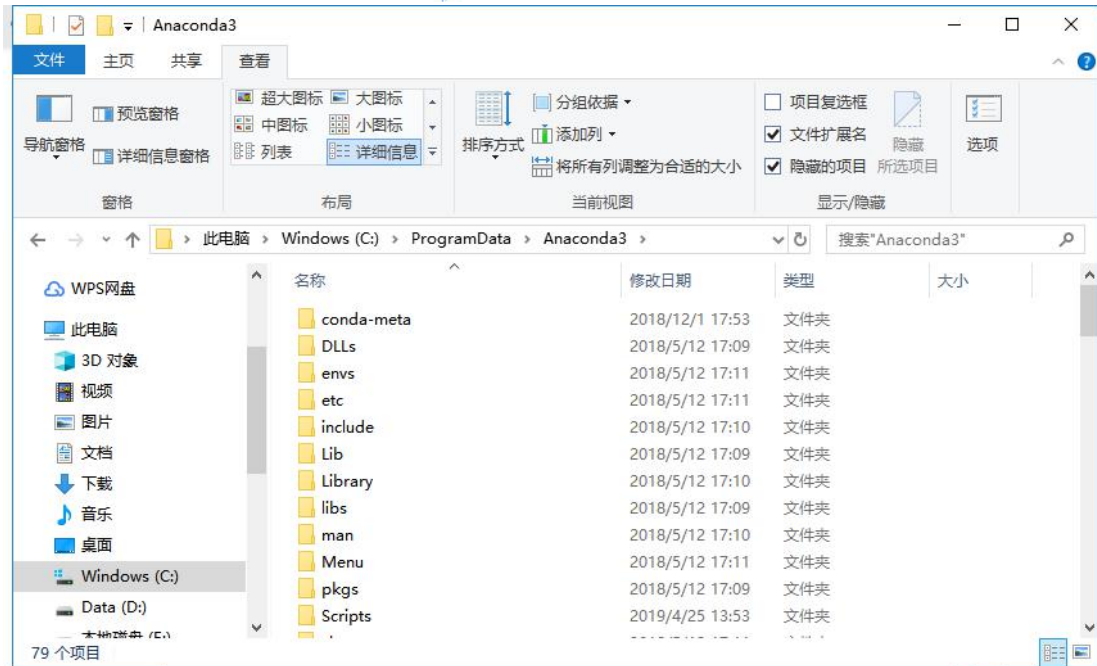
不过老师求求大家路径里面请不要带中文、空格，否则后面报的 Bug 真的恶心



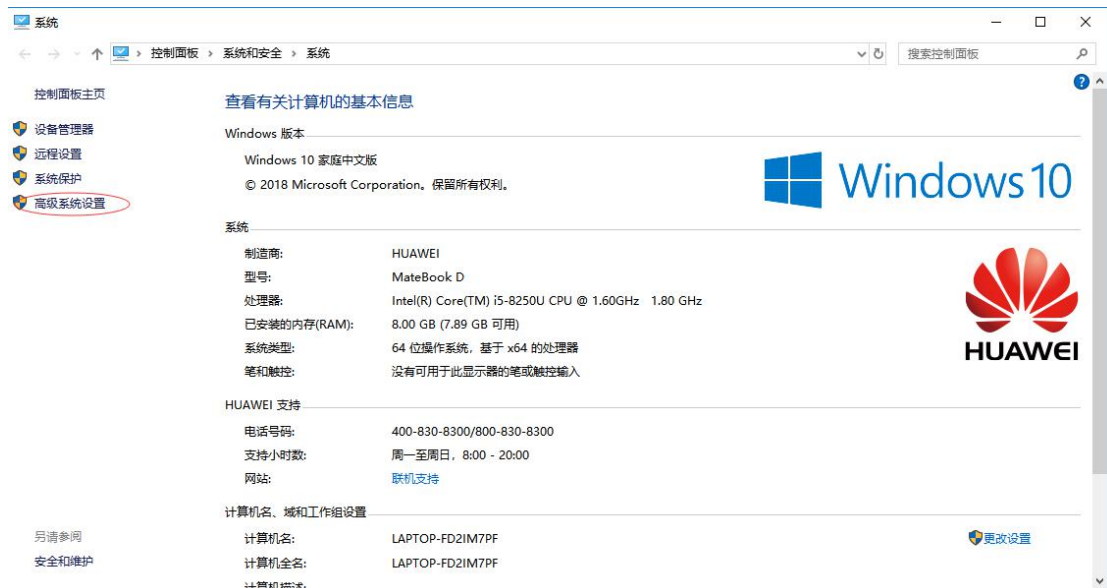
Fourth，安装完成后可以在对应路径下找到安装的内容

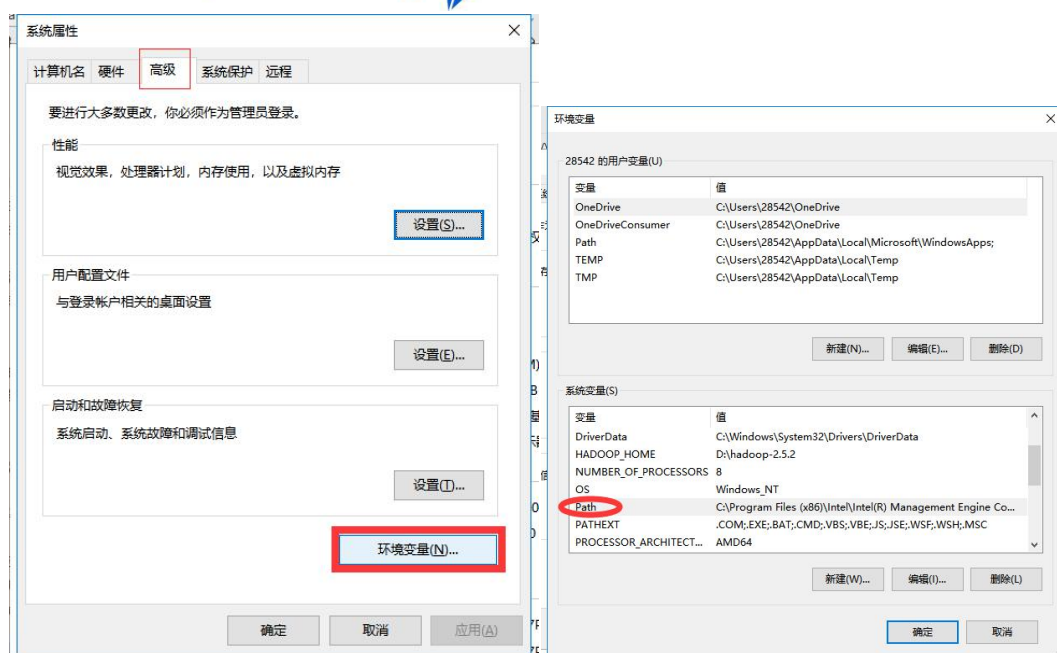


这里如果找不到说明 ProgramData 这个隐藏文件夹没有展示出来，勾选上“查看”里面的“隐藏的项目”，为了看到代码后缀请顺手勾选上“文件扩展名”

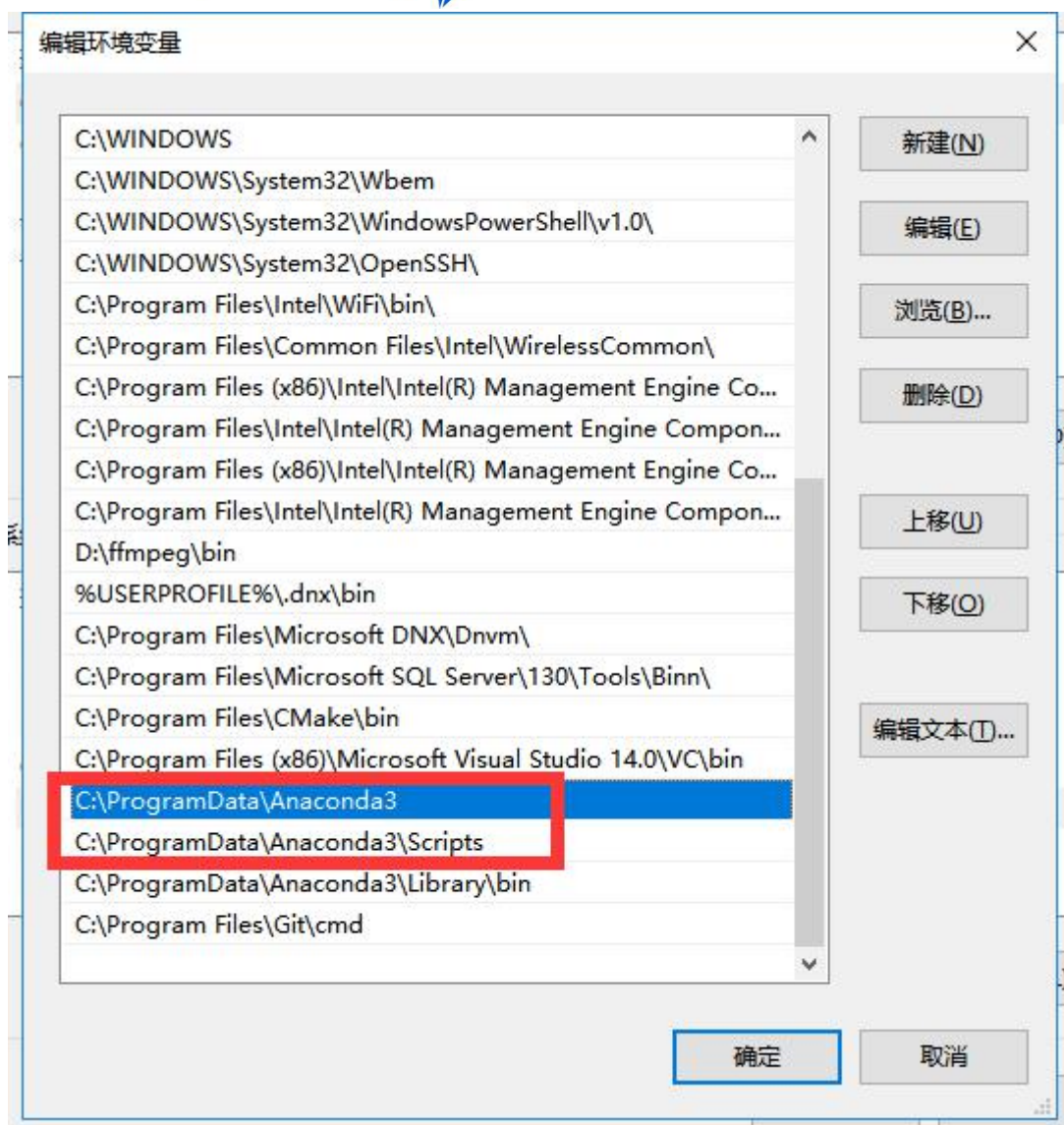


Fifth, 确认配置了环境变量, 请右键“此电脑”然后点击“属性”, 然后选择“高级系统设置”

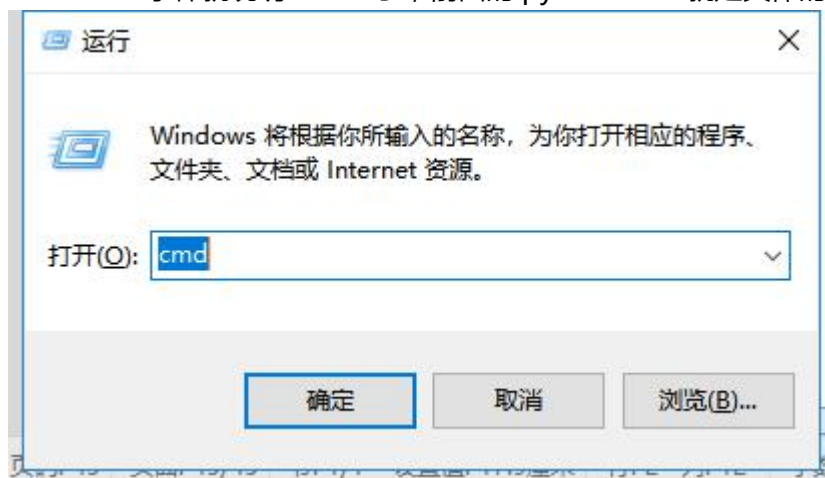




双击点开“path”，一定要确保下图红框中的两个路径（你安装 anaconda 的路径）配置好了，然后一路确定就好



Final, win+R, 输入 cmd 并确定, cmd 命令行中输入 python 回车, 看到类似下图中有 anaconda 字样就说明 OKAY 了, 前面的 python3.6.4 就是具体的 python 环境版本



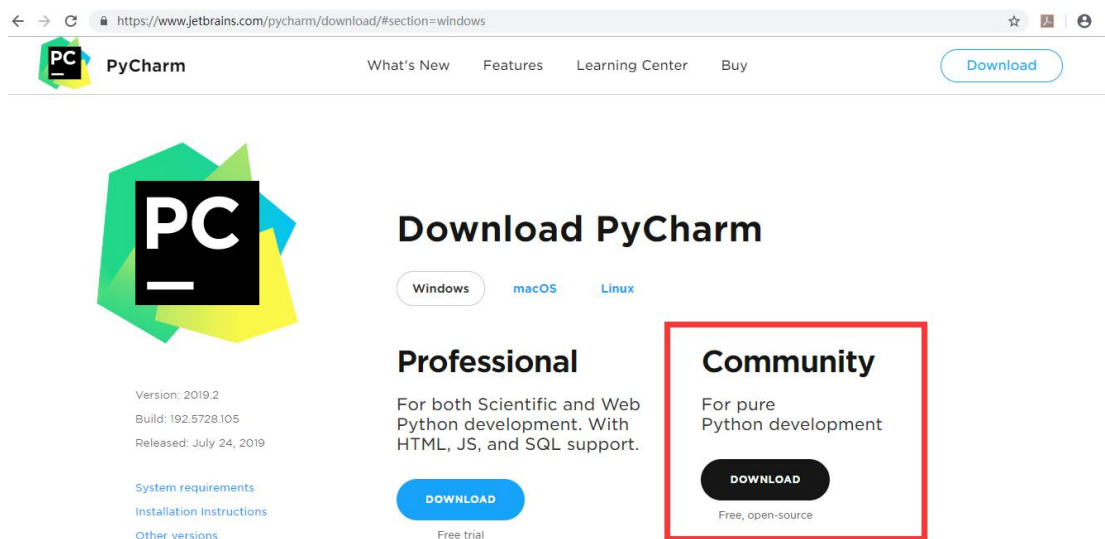

```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows [版本 10.0.17134.885]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\28542>python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) [MSC v.1900 64 b
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

PyCharm 开发工具安装和配置

<https://www.jetbrains.com/pycharm/>

熟悉 Java 或 Scala 的 IDEA 开发工具的同学，肯定知道 jetbrains 这家公司，他家针对 Python 的 PyCharm 和 IDEA 开发工具的开发风格是非常类似的，快捷键很多都是一致的，偷偷告诉大家，他家还有一个 Clion 也非常好用，如果你有开发 C 或 C++ 的需要的话



当大家登陆他家 pycharm 的页面时候，我们来做 AI 其实只需要下载 Community 社区版就已经够用了，所以就省了什么 license 等麻烦了

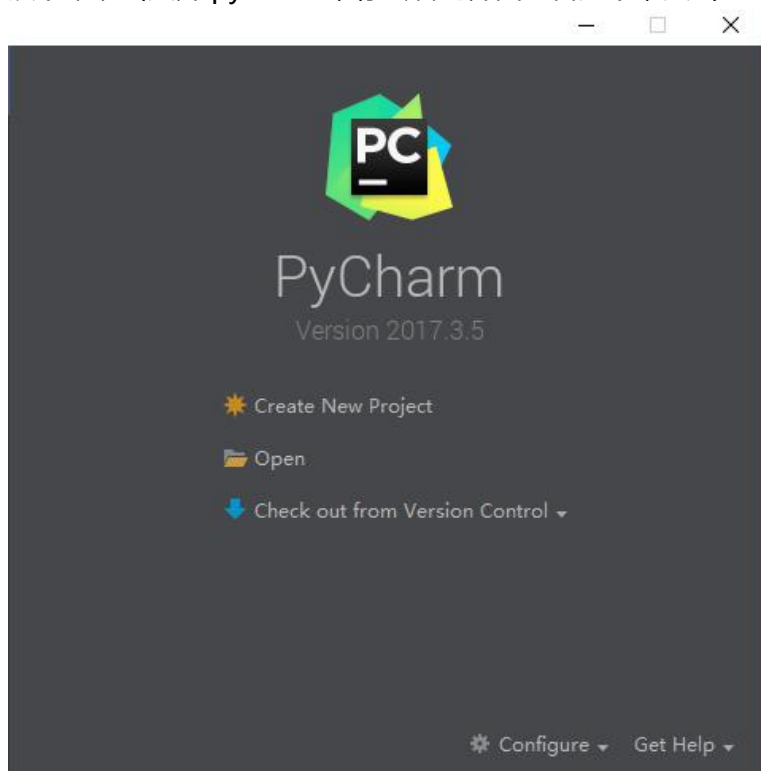
 pycharm-community-2017.3.3.exe

有了 exe 文件双击打开一路点 next 就可以了，这时路径里有没有中文或空格都不重要了，老师用的就是它提示的默认路径安装的，至于为什么前面 python 环境 anaconda 我们要避免中文和空格，是因为我们需要 pycharm 更容易的找到 python 环境，方便后面配置，got it?

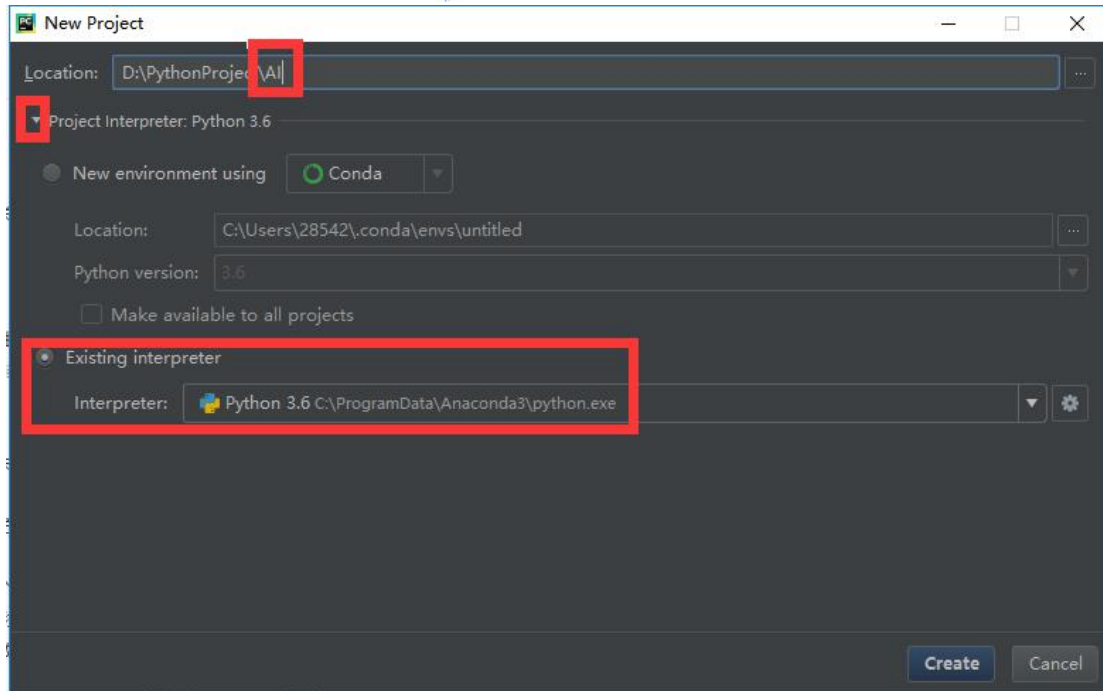


上图里面大家看到老师的安装路径就是默认路径，带着空格也是 okay 的，同时你看到上图中有一个叫 Jupyter Notebook，这个其实是安装好 anaconda 后就会自带的一个挺好用的开发工具，后面我们也会用到，一般在开发 AI 相关程序代码中，这两个就够用了。当然有同学习惯了用 sublime 等编辑器开发 python 相关程序，老师也不拦着，只不过老师就不在这里一一介绍了。

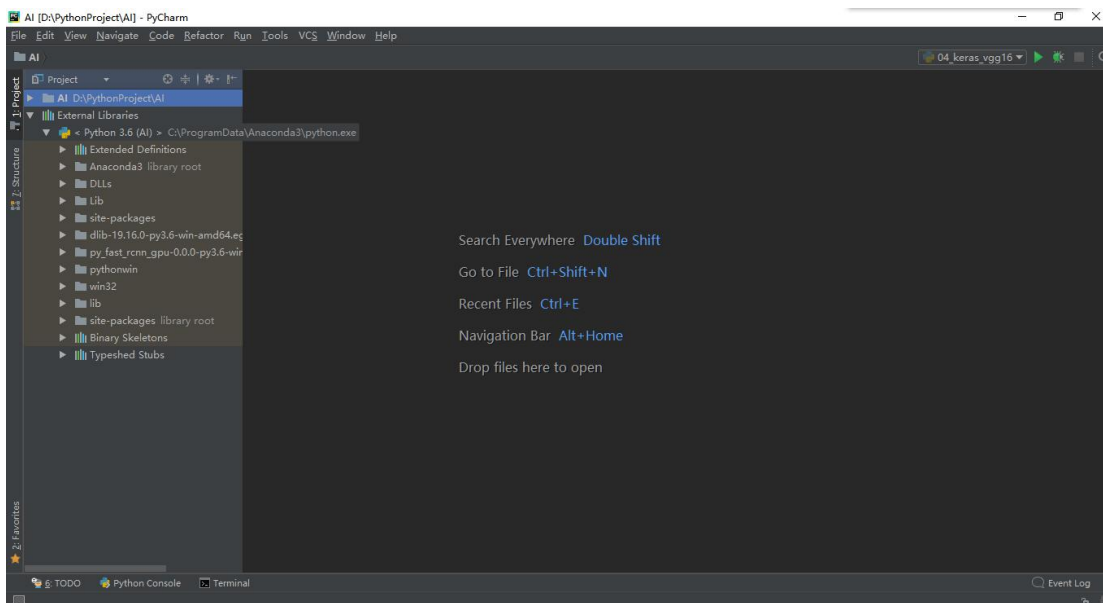
接下来双击点开 pycharm，你会看到界面包含如下，点击 “Create New Project”



然后请如下图把 location 中的 “untitled” 改为你自己命名的项目名称，比如如图 “AI”，然后点击红色标注出来三角，在点击后出现的内容中勾选 “Existing Interpreter”，然后去点击对应的齿轮图标，选择的就是刚刚咱们安装的 anaconda 环境目录中的 python.exe 执行文件，最后 “create” 就好

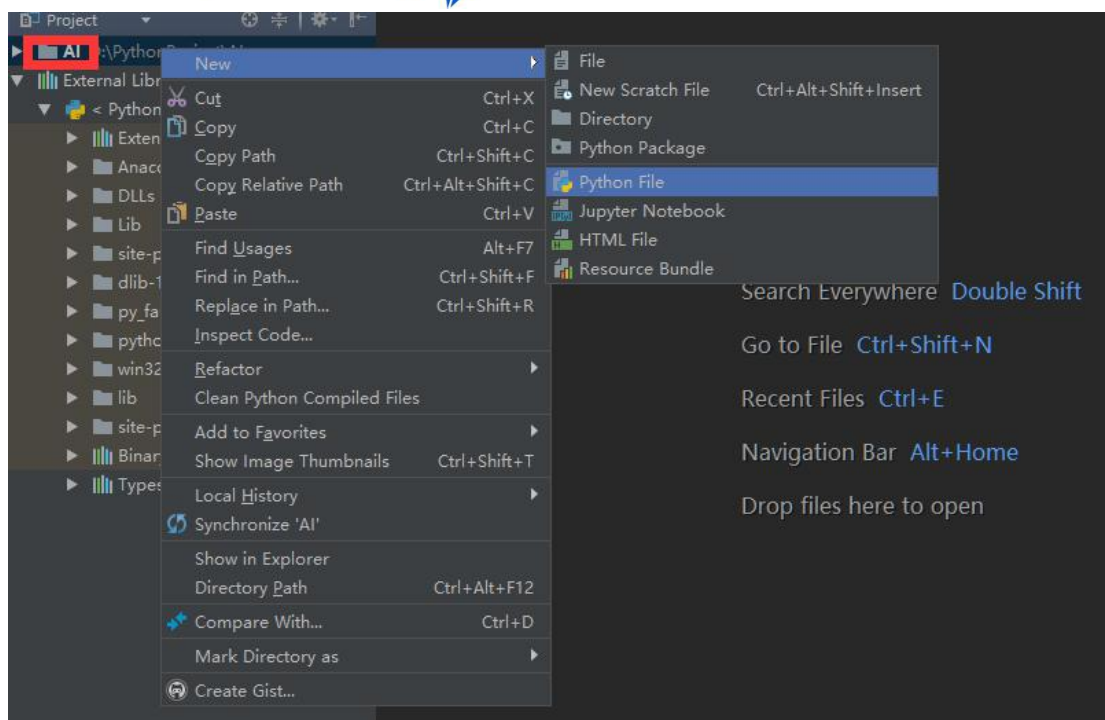


创建后进入界面如下



编写 python 脚本测试环境

之后鼠标可以在 AI 这个项目名称上右键，然后 “New” --> “Python File”



Python 是弱类型编程语言

```
int x = 10;
x = 10
x = "yasaka"
print(x)
```

Python 即可以面向对象编程，也可以面向函数编程

面向对象思想：创建一个汽车，然后调用汽车 run 方法

面向函数思想（过程）：写个函数，函数里面让 4 个轮子扛着一个沙发同样的速度跑起来

Python 里面虽然支持不同编程思想，但是 python 更是一切皆对象，甚至在内存里面存储个数值 10.01，这个都是一个对象

python 语言中的 main 函数，右键执行看看两个 print 函数的打印顺序

```
x = 10
print(x)

if __name__ == '__main__':
    print('yasaka')
```

第一点使用 main 函数的原因，大量代码写在外面太乱了，往往我们会通过 def 封装函数，同时封装在 def 中也可以增加代码的复用性

```
def my_print(x):
    print(x)
```

```
if __name__ == '__main__':  
    my_print('yasaka')
```

顺带手说一下，这里函数没有用花括号，区别段落用缩进，4 个空格，如果是 Tab 键需要额外注意，2 个空格没有空格都会报错

第二点 python 里面什么是一个 module 模块，其实在 python 语言中一个脚本就可以称为一个模块，所谓的一个模块就是 python 脚本中封装了很多函数方法可以供其他脚本去调用。

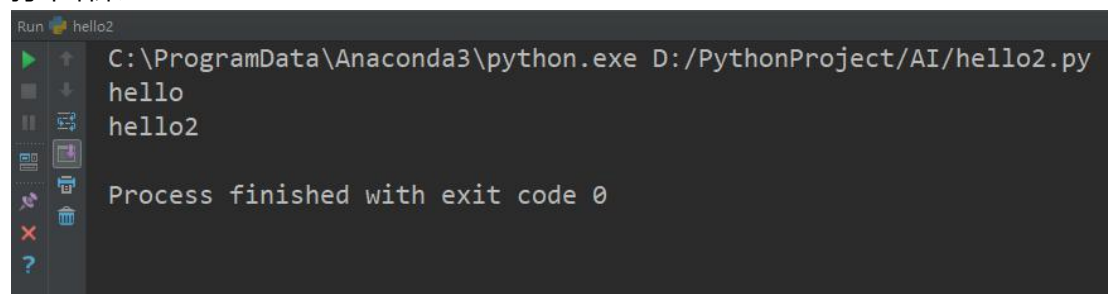
比如我们已经有一个脚本 hello.py 了

```
print('hello')  
  
def my_print(x):  
    print(x)  
  
if __name__ == '__main__':  
    my_print('yasaka')
```

我们再创建一个脚本 hello2.py

```
from hello import my_print  
  
my_print("hello2")
```

打印结果：



```
Run hello2  
C:\ProgramData\Anaconda3\python.exe D:/PythonProject/AI/hello2.py  
hello  
hello2  
Process finished with exit code 0
```

这里你会看到打印 hello2 之前有个 hello 先打印了，这时因为在导入 hello.py 这个脚本的时候它会把 main 函数之外的内容都先执行一下，所以一个模块里面封装了一些供别人调用的代码，一定不要在模块的函数体之外写一些测试的代码，要写也写在 main 函数中去

章节 5：代码实战求解线性回归算法模型

代码实战解析求解模型

规范的表头，还有作者名字

中间导入这个脚本需要的模块，numpy 是去做数值计算的，matplotlib 是关于绘图的，as 是给模块起个名字，方便后面调用模块的一些函数方法

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
# 文件名: linear_regression_0.py

import numpy as np
import matplotlib.pyplot as plt

__author__ = 'yasaka'
```

我们通过 numpy 随机创建数据用于训练，这里相当于是随机 X 维度 X1，rand 是随机均匀分布，是在创建用于训练的 X 矩阵，100 行一列的数组

```
X = 2 * np.random.rand(100, 1)
print(len(X))
print(X)
```

rand 的源码说明，这里三个双引号是 docstring，作为方法的注释说明，可以通过 help() 方法名去查看到这些注释，如果只是 # 来注释方法无法看到

这里 [] 左边闭区间，右边开区间，也就是每个数字随机的时候有可能是 0，绝不可能是 1，然后它随机的时候

```
def rand(*dn): # known case of numpy.random.mtrand.rand
    """
    rand(d0, d1, ..., dn)

    Random values in a given shape.

    Create an array of the given shape and populate it with
    random samples from a uniform distribution
    over ``[0, 1)``.
```


很多时候我们可以直接看 example 例子来学习函数方法如何去用，我们看到 array 这是 numpy 封装的数组，原生的 python 是没有 array 数组这个类型的，它只有 tuple 元组，list 列表，dict 字典等。

Examples

```
-----
>>> np.random.rand(3,2)
array([[ 0.14022471,  0.96360618], #random
       [ 0.37601032,  0.25528411], #random
       [ 0.49313049,  0.94909878]]) #random
```

接下来人为的设置真实的 Y 一列，这里的 4 其实就是把 $W_1=4$ ，然后 5 就是 $W_0=5$ ，
 $y = 5 + 4 * X + np.random.randn(100, 1)$

$np.random.randn(100, 1)$ 是设置 error，因为 randn 是标准正太分布，符合我们前面讲过的误差服从均值为 0 的正太分布，这就意味着这样构建的 X 和 Y 的关系去使用我们前面讲过的多元线性回归去拟合求解会很合适。

同时，我们有 100 个真实值 y，我们有 $5+4X$ 就是 100 个 y_{hat} ，那么我们的误差就是真实值减去预测值也会有 100 个误差值。所以这也就是我们去用 randn 去创建 100 行 1 列的数组的原因。

```
def randn(*dn): # known case of numpy.random.mtrand.randn
    """
    randn(d0, d1, ..., dn)

    Return a sample (or samples) from the "standard normal" distribution.

    If positive, int_like or int-convertible arguments are provided,
    `randn` generates an array of shape ``(d0, d1, ..., dn)`` filled
    with random floats sampled from a univariate "normal" (Gaussian)
    distribution of mean 0 and variance 1 (if any of the :math:`d_i` are
    floats, they are first converted to integers by truncation). A single
    float randomly sampled from the distribution is returned if no
    argument is provided.

    This is a convenience function. If you want an interface that takes a
    tuple as the first argument, use `numpy.random.standard_normal` instead.
```

Examples

```
-----
>>> np.random.randn()
2.1923875335537315 #random

Two-by-four array of samples from N(3, 6.25):

>>> 2.5 * np.random.randn(2, 4) + 3
array([[ -4.49401501,  4.00950034, -1.81814867,  7.29718677], #random
       [ 0.39924804,  4.68456316,  4.99394529,  4.84057254]]) #random
```

前面我们讲过在多元线性回归中，W 的个数其实是和 X 特征的数量一一对应的，比如我们有 X_1 、 X_2 两个特征，我们通过多元线性回归就可以求解得到 W_1 、 W_2 两个参数，比如我

们有 X0、X1 两个特征，我们通过多元线性回归就可以求解得到 W0、W1 两个参数。
那么此时我们只有 X1 一列向量，我们还需要一列 X0 才可以去求得 W0，因为我们上面的式子中有个 W0=5 嘛，所以接下来整合 X0 和 X1

```
X_b = np.c_[np.ones((100, 1)), X]
print(X_b)
```

这里的 c_就是拼接，可以把它理解为 concatenate 拼接，而且我们把 X0 一列都设置为 1，这样多元线性回归公式就可以把 1 去乘以 W0，这样就符合公式的定义，还有注意拼接的顺序就是求得对应 W 的顺序。

```
Examples
-----
>>> np.c_[np.array([1,2,3]), np.array([4,5,6])]
array([[1, 4],
       [2, 5],
       [3, 6]])
>>> np.c_[np.array([[1,2,3]]), 0, 0, np.array([[4,5,6]])]
array([[1, 2, 3, 0, 0, 4, 5, 6]])
```

	X0	X1
m 行	1	0.14
	1	0.96

	1	0.37

接下来数据准备好了，我们就可以应用我们之前推导出来的解析解公式

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

```
theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
print(theta_best)
```

这里的.T 就是 transpose 转置的作用，dot 就是向量点乘，np.linalg.inv 是矩阵求逆

```
# Matrix inversion
def inv(a):
    """
    Compute the (multiplicative) inverse of a matrix.

    Given a square matrix `a`, return the matrix `ainv` satisfying
    ``dot(a, ainv) = dot(ainv, a) = eye(a.shape[0])``.
    """
```

np.linalg

```
Core Linear Algebra Tools
=====

Linear algebra basics
=====

norm      Vector or matrix norm
inv        Inverse of a square matrix
solve      Solve a linear system of equations
det        Determinant of a square matrix
slogdet    Logarithm of the determinant of a square matrix
lstsq      Solve linear least-squares problem
pinv       Pseudo-inverse (Moore-Penrose) calculated using a singular
           value decomposition
matrix_power Integer power of a square matrix
matrix_rank Calculate matrix rank using an SVD-based method
=====
```

我们计算结果很有意思，就是你会发现最后并不能准确的计算出来 5 和 4，而是计算出来一个接近期望的值，为什么？如果是我们给 y 不设定误差项，那么计算出来的结果是什么？还有形状是什么形状？

```
[[ 5.14460608]
 [ 3.84420248]]
```

下面我们去把 model 模型使用起来，只有 model 模型预测的准，那么才有价值

```
X_new = np.array([[0],
                  [2]])
X_new_b = np.c_[np.ones((2, 1)), X_new]
print(X_new_b)
```

这里我们同样构建一个两行一列的新的 X，然后给它添加上一列恒为 1 的 X0

```
y_predict = X_new_b.dot(theta_best)
print(y_predict)
```

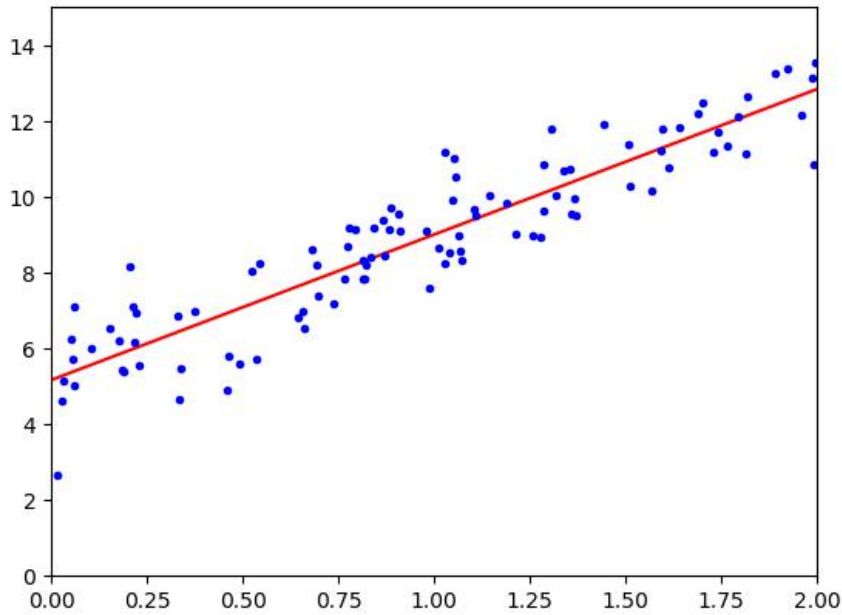
^

$$y = X\theta$$

计算的期望应该在 10、4 附近对吧，接下来我们图示出来

```
plt.plot(X_new, y_predict, 'r-')
plt.plot(X, y, 'b.')
plt.axis([0, 2, 0, 15])
plt.show()
```

红色的是代码模型那条直线，蓝色的点 100 个我们随机出来的真实数据，通过图示可以很清楚的看到我们用多元线性回归算法拟合的效果



扩展

如果我们影响 y 的因为有两个，而不是前面例子只有一个的话呢？

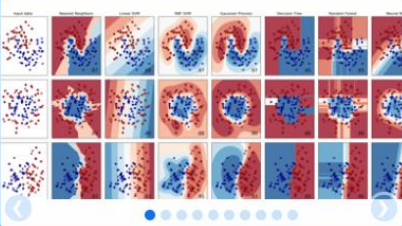
```
y = 5 + 4 * X1 + 3 * X2 + np.random.randn(100, 1)
```

那么我们是不是就还得有一个 $X2$ 一列数据

```
X1 = 2 * np.random.rand(100, 1)
X2 = 3 * np.random.rand(100, 1)
X_b = np.c_[np.ones((100, 1)), X1, X2]
print(X_b)
```

sklearn 调用 LinearRegression 类

sklearn 简介



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning

Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction. — Examples

代码训练模型和模型预测

从 sklearn 模块下的线性模块下导入多元线性回归类

```
import numpy as np
from sklearn.linear_model import LinearRegression
```

python 中驼峰原则是一个类的定义

```
class LinearRegression(LinearModel, RegressorMixin):
    """
    Ordinary Least squares Linear Regression.
```

构建真实的 X 和 y

```
X1 = 2 * np.random.rand(100, 1)
X2 = 2 * np.random.rand(100, 1)
X = np.c_[X1, X2]
y = 4 + 3 * X1 + 5 * X2 + np.random.randn(100, 1)
```

python 中驼峰原则创建一个类的对象，当然得加上小括号，不过因为是 python 所以不用 new 关键字，然后把对象赋给一个变量

```
lin_reg = LinearRegression()
```

接下来，我们去调用 fit 类的方法，这是非常 sklearn 的风格，然后去打印出结果，截距项和参数系数

```
lin_reg.fit(X, y)
```

```
print(lin_reg.intercept_, lin_reg.coef_)
```

```
[3.62101244] [[2.89958232 5.2372749 ]]
```

我们回想一下上一小节的代码，我们为了去计算出截距项，我们还人为的添加了一个 X_0 列向量，这里我们并没有这样做，所以可想而知这个 sklearn 封装的类里面肯定是有点门道的，里面是可以有些超参数传递的，python 里面的生成对象的第一个参数都是 self，我们可以不用管这个，直接可以把 self 之后的参数看成是可以传的参数

```
class LinearRegression
def __init__(self, fit_intercept=True, normalize=False, copy_X=True,
n_jobs=None)
```

```
eg = LinearRegression()
```

这里我们可以看到有一个参数是 `fit_intercept=True`，之所以是有等号是代表是有默认值，True 代表这个多元线性回归会帮助我们去计算截距项 W_0 ，本质就是会帮助我们去把 X 身上加上 X_0 一列再去求解方程。

```
def fit(self, X, y, sample_weight=None):
    """
    Fit linear model.

    Parameters
```

fit 方法里面我们可以看到有一个 `_preprocess_data()` 方法去对 X, y 进行操作并且返回 X, y

```
X, y, X_offset, y_offset, X_scale = self._preprocess_data(
    X, y, fit_intercept=self.fit_intercept, normalize=self.normalize,
    copy=self.copy_X, sample_weight=sample_weight)
```

如果我们现在把这个参数调一下 `fit_intercept=False`

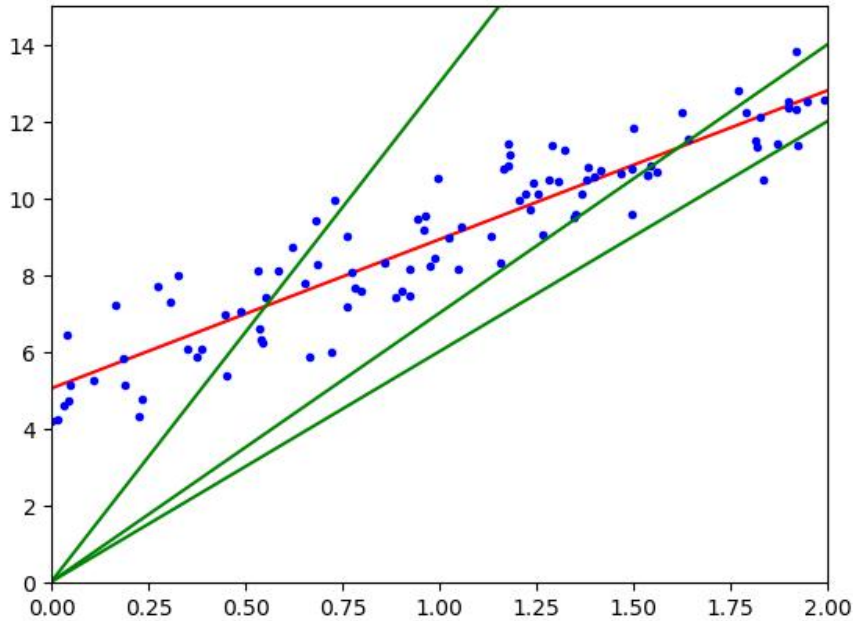
```
lin_reg = LinearRegression(fit_intercept=False)
```

```
lin_reg.fit(X, y)
```

```
print(lin_reg.intercept_, lin_reg.coef_)
```

```
0.0 [[4.66164004 6.86917197]]
```

分析一下有截距项和没有截距项的区别



好了，我们下面去使用模型去预测

```
X_new = np.array([[0],
                  [2],
                  [2]])
print(lin_reg.predict(X_new))
```

为什么会出现下面的错误呢？

```
return self._decision_function(X)
File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\base.py", line 100, in predict
    dense_output=True) + self.intercept_
File "C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\extmath.py", line 100, in dot
    return np.dot(a, b)
ValueError: shapes (3,1) and (2,1) not aligned: 1 (dim 1) != 2 (dim 0)
```

记住，你怎么训练模型的，模型就具备什么样的功能！这里我们训练模型的时候用的训练集的 X 是两个特征，那么我们这里去使用模型的时候传给模型的 X_new 也得是两个维度

```
X_new = np.array([[0, 0],
                  [2, 1],
                  [2, 4]])
print(lin_reg.predict(X_new))
```

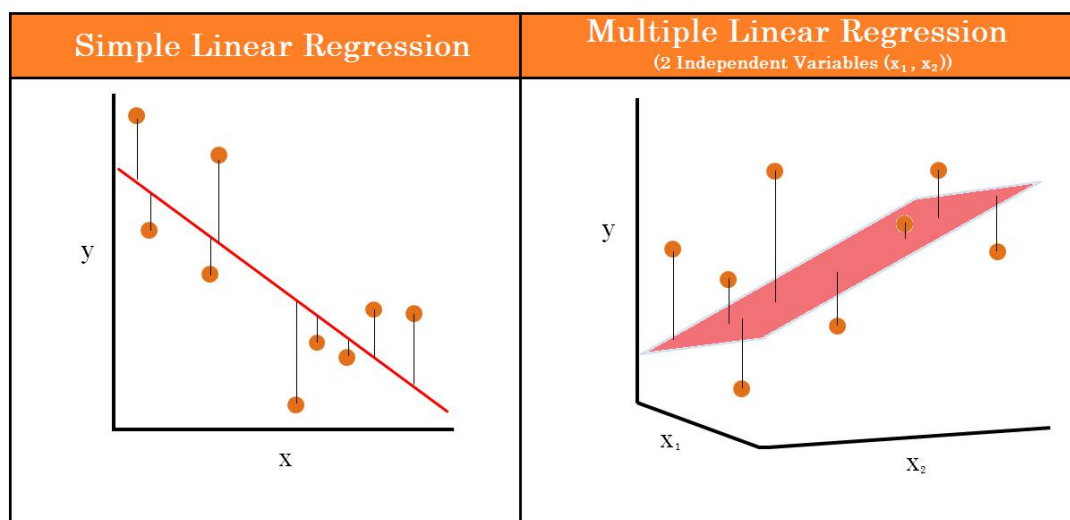
扩展概念

维度

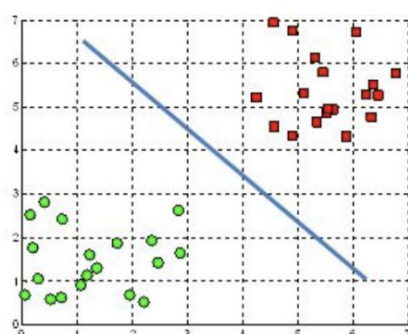
举例：

一个不断升高维度得游戏，让同学们理解维度的提高对于机器学习的重要性

通常我们会认为我们人类生活在地球上是一个 3 维空间中，如果加上时间就是 4 维，比如点是 0 维，线是一维，面是二维空间，立体空间是 3 维空间



A hyperplane in \mathbb{R}^2 is a line



A hyperplane in \mathbb{R}^3 is a plane

