

Regression: Case Study

回归-案例研究

问题的导入：预测宝可梦的CP值

Estimating the Combat Power(CP) of a pokemon after evolution

我们期望根据已有的宝可梦进化前后的信息，来预测某只宝可梦进化后的cp值的大小

确定Scenario、Task和Model

Scenario

首先根据已有的data来确定Scenario，我们拥有宝可梦进化前后cp值的这样一笔数据，input是进化前的宝可梦(包括它的各种属性)，output是进化后的宝可梦的cp值；因此我们的data是labeled，使用的Scenario是**Supervised Learning**

Task

然后根据我们想要function的输出类型来确定Task，我们预期得到的是宝可梦进化后的cp值，是一个scalar，因此使用的Task是**Regression**

Model

关于Model，选择很多，这里采用的是**Non-linear Model**

设定具体参数

X : 表示一只宝可梦，用下标表示该宝可梦的某种属性

X_{cp} : 表示该宝可梦进化前的cp值

X_s : 表示该宝可梦是属于哪一种物种，比如妙瓜种子、皮卡丘...

X_{hp} : 表示该宝可梦的hp值即生命值是多少

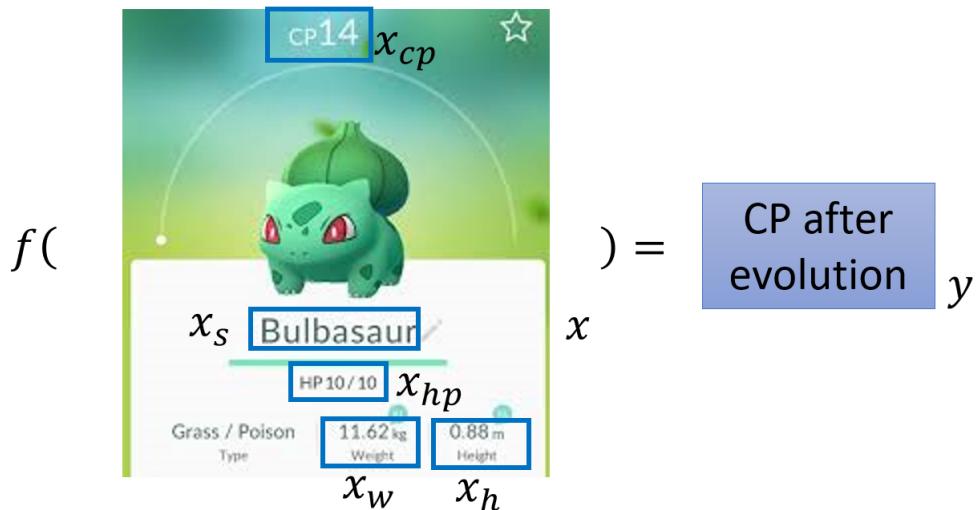
X_w : 代表该宝可梦的重量量

X_h : 代表该宝可梦的高度

$f()$: 表示我们要找的function

y : 表示function的output，即宝可梦进化后的cp值，是一个scalar

- Estimating the Combat Power (CP) of a pokemon after evolution



Regression的具体过程

回顾一下machine Learning的三个步骤:

- 定义一个model即function set
- 定义一个goodness of function损失函数去评估该function的好坏
- 找一个最好的function

Step1: Model (function set)

如何选择一个function的模型呢? 毕竟只有确定了模型才能调参。这里没有明确的思路, 只能凭经验去一种种地试

Linear Model 线性模型

$$y = b + w \cdot X_{cp}$$

y 代表进化后的cp值, X_{cp} 代表进化前的cp值, w 和**b**代表未知参数, 可以是任何数值

根据不同的w和b, 可以确定不同的无穷无尽的function, 而 $y = b + w \cdot X_{cp}$ 这个抽象出来的式子就叫做model, 是以上这些具体化的function的集合, 即function set

实际上这是一种**Linear Model**, 但只考虑了宝可梦进化前的cp值, 因而我们可以将其扩展为:

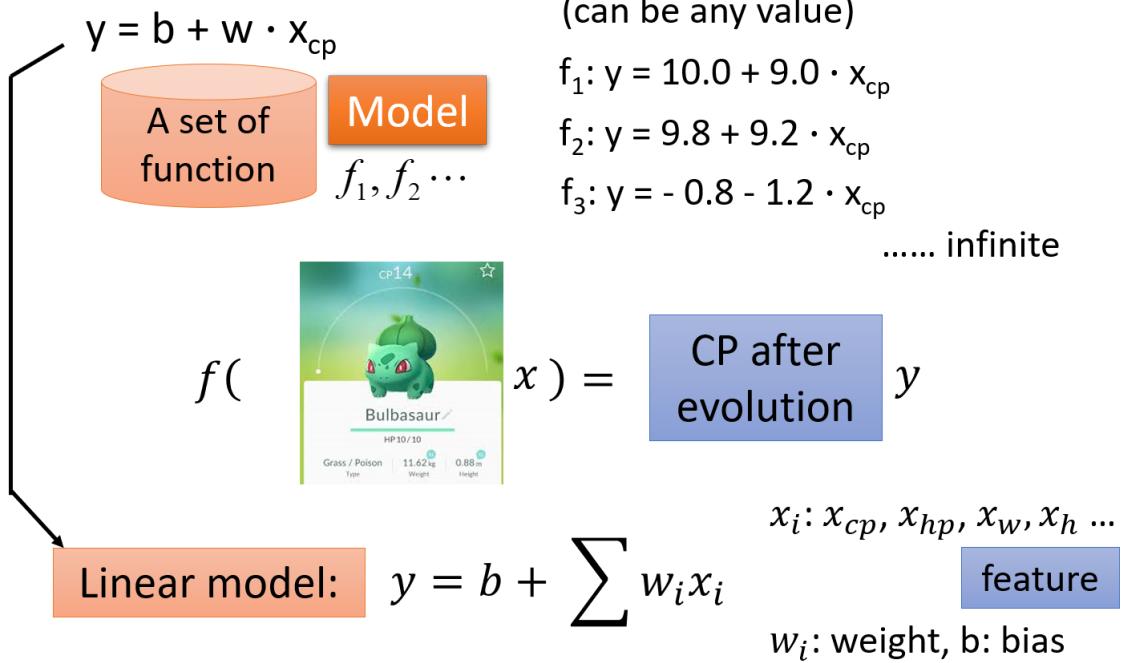
$$y = b + \sum w_i x_i$$

x_i : an attribute of input X (x_i is also called **feature**, 即特征值)

w_i : weight of x_i

b: bias

Step 1: Model



Step2: Goodness of Function

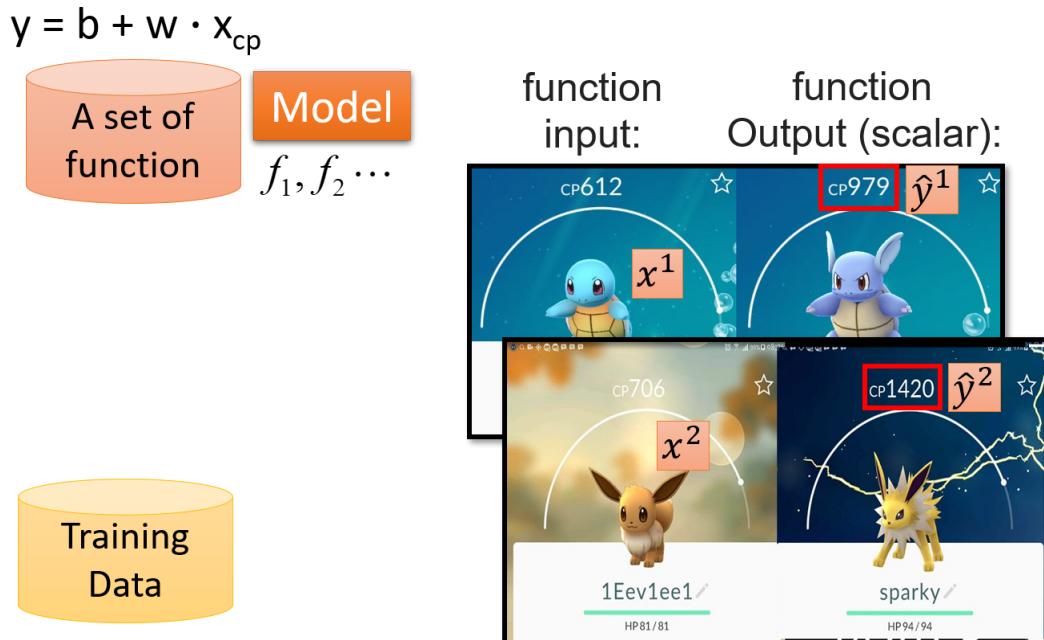
参数说明

x^i : 用上标来表示一个完整的object的编号, x^i 表示第*i*只宝可梦(下标表示该object中的component)

\hat{y}^i : 用 \hat{y} 表示一个实际观察到的object输出, 上标为*i*表示是第*i*个object

注: 由于regression的输出值是scalar, 因此 \hat{y} 里面并没有component, 只是一个简单的数值; 但是未来如果考虑structured Learning的时候, 我们output的object可能是有structured的, 所以我们还是会需要用上标下标来表示一个完整的output的object和它包含的component

Step 2: Goodness of Function



Loss function 损失函数

为了衡量function set中的某个function的好坏，我们需要一个评估函数，即Loss function，损失函数，简称L；Loss function是一个function的function

$$L(f) = L(w, b)$$

input: a function;

output: how bad/good it is

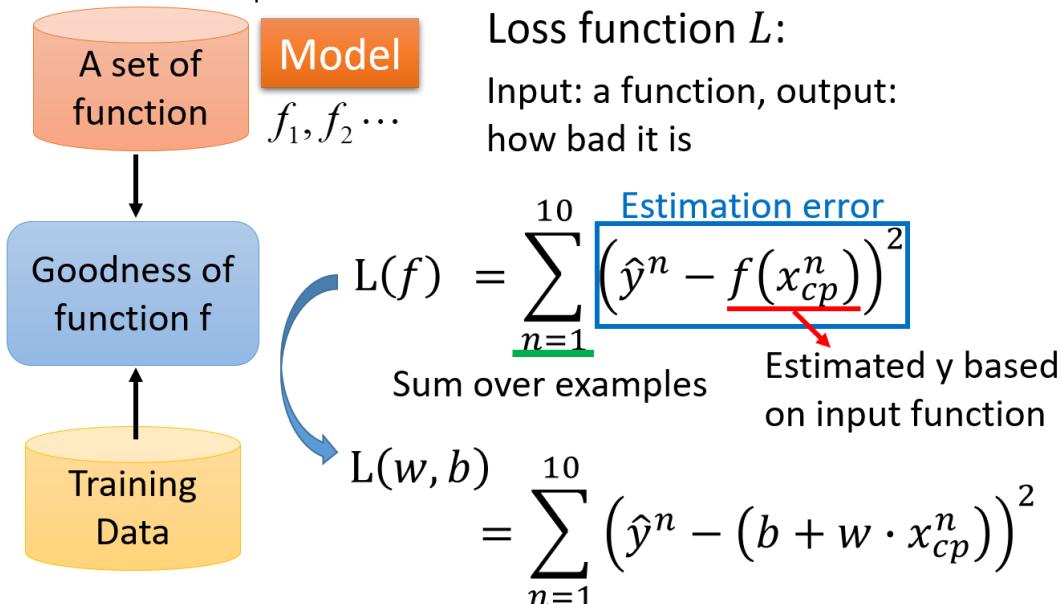
由于 $f: y = b + w \cdot x_{cp}$, 即f是由b和w决定的, 因此input f就等价于input这个f里的b和w, 因此Loss function实际上是在衡量一组参数的好坏

之前提到的model是我们自主选择的, 这里的loss function也是, 最常用的方法就是采用类似于方差和的形式来衡量参数的好坏, 即预测值与真值差的平方和; 这里真正的数值减估测数值的平方, 叫做估测误差, Estimation error, 将10个估测误差合起来就是loss function

$$L(f) = L(w, b) = \sum_{n=1}^{10} (\hat{y}^n - (b + w \cdot x_{cp}^n))^2$$

如果 $L(f)$ 越大, 说明该function表现得越不好; $L(f)$ 越小, 说明该function表现得越好

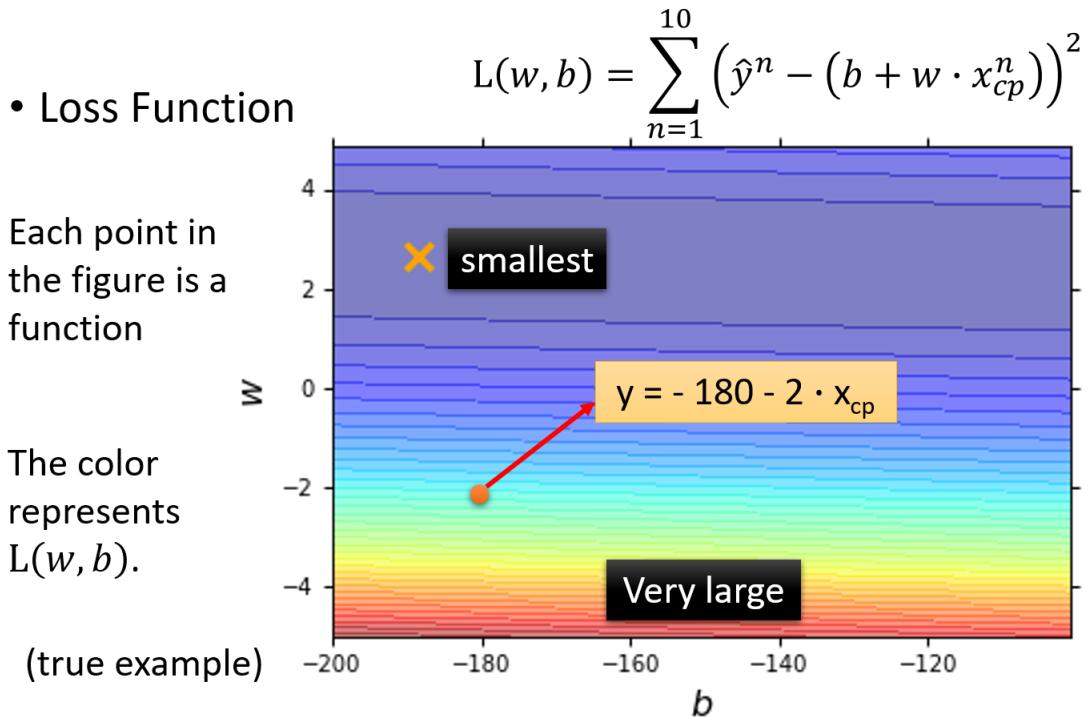
$$y = b + w \cdot x_{cp}$$



Loss function可视化

下图中是loss function的可视化, 该图中的每一个点都代表一组 (w, b) , 也就是对应着一个function; 而该点的颜色对应着loss function的结果 $L(w, b)$, 它表示该点对应function的表现有多糟糕, 颜色越偏红色代表Loss的数值越大, 这个function的表现越不好, 越偏蓝色代表Loss的数值越小, 这个function的表现越好

比如图中用红色箭头标注的点就代表了 $b=-180, w=-2$ 对应的function, 即 $y = -180 - 2 \cdot x_{cp}$, 该点所处的颜色偏向于红色区域, 因此这个function的loss比较大, 表现并不好



Step3: Pick the Best Function

我们已经确定了loss function，他可以衡量我们的model里面每一个function的好坏，接下来我们要做的事情就是，从这个function set里面，挑选一个最好的function

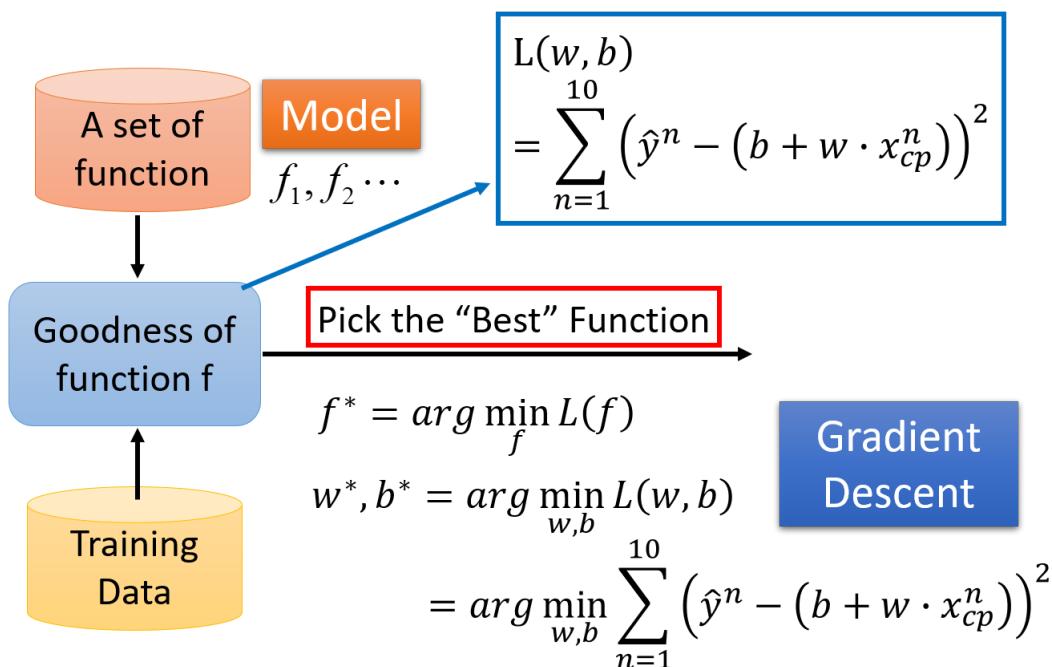
挑选最好的function这件事情，写成formulation/equation的样子如下：

$$f^* = \arg \min_f L(f), \text{ 或者是}$$

$$w^*, b^* = \arg \min_{w,b} L(w, b) = \arg \min_{w,b} \sum_{n=1}^{10} (\hat{y}^n - (b + w \cdot x_{cp}^n))^2$$

也就是那个使 $L(f) = L(w, b) = Loss$ 最小的 f 或 (w, b) ，就是我们要找的 f^* 或 (w^*, b^*) (有点像极大似然估计的思想)

Step 3: Best Function



利用线性代数的知识，可以解得这个closed-form solution，但这里采用的是一种更为普遍的方法——gradient descent(梯度下降法)

Gradient Descent 梯度下降

上面的例子比较简单，用线性代数的知识就可以解；但是对于更普遍的问题来说，gradient descent的厉害之处在于，只要 $L(f)$ 是可微分的，gradient descent都可以拿来处理这个 f ，找到表现比较好的parameters

单个参数的问题

以只带单个参数 w 的Loss Function $L(w)$ 为例，首先保证 $L(w)$ 是可微的

$w^* = \arg \min_w L(w)$ 我们的目标就是找到这个使Loss最小的 w^* ，实际上就是寻找切线 L 斜率为0的global minima最小值点(注意，存在一些local minima极小值点，其斜率也是0)

有一个暴力的方法是，穷举所有的 w 值，去找到使loss最小的 w^* ，但是这样做是没有效率的；而gradient descent就是用来解决这个效率问题的

- 首先随机选取一个初始的点 w^0 (当然也不一定要随机选取，如果有办法可以得到比较接近 w^* 的表现得比较好的 w^0 当初始点，可以有效地提高查找 w^* 的效率)
- 计算 L 在 $w = w^0$ 的位置的微分，即 $\frac{dL}{dw}|_{w=w^0}$ ，几何意义就是切线的斜率
- 如果切线斜率是negative负的，那么就应该使 w 变大，即往右踏一步；如果切线斜率是positive正的，那么就应该使 w 变小，即往左踏一步，每一步的步长step size就是 w 的改变量

w 的改变量step size的大小取决于两件事

- 一是现在的微分值 $\frac{dL}{dw}$ 有多大，微分值越大代表现在在一个越陡峭的地方，那它要移动的距离就越大，反之就越小；
- 二是一个常数项 η ，被称为learning rate，即学习率，它决定了每次踏出的step size不只取决于现在的斜率，还取决于一个事先就定好的数值，如果learning rate比较大，那每踏出一步的时候，参数 w 更新的幅度就比较大，反之参数更新的幅度就比较小

如果learning rate设置的大一些，那机器学习的速度就会比较快；但是learning rate如果太大，可能就会跳过最合适global minima的点

- 因此每次参数更新的大小是 $\eta \frac{dL}{dw}$ ，为了满足斜率为负时 w 变大，斜率为正时 w 变小，应当使原来的 w 减去更新的数值，即

$$\begin{aligned} w^1 &= w^0 - \eta \frac{dL}{dw}|_{w=w^0} \\ w^2 &= w^1 - \eta \frac{dL}{dw}|_{w=w^1} \\ w^3 &= w^2 - \eta \frac{dL}{dw}|_{w=w^2} \\ &\dots \\ w^{i+1} &= w^i - \eta \frac{dL}{dw}|_{w=w^i} \\ \text{if } (\frac{dL}{dw}|_{w=w^i} &= 0) \text{ then stop;} \end{aligned} \tag{1}$$

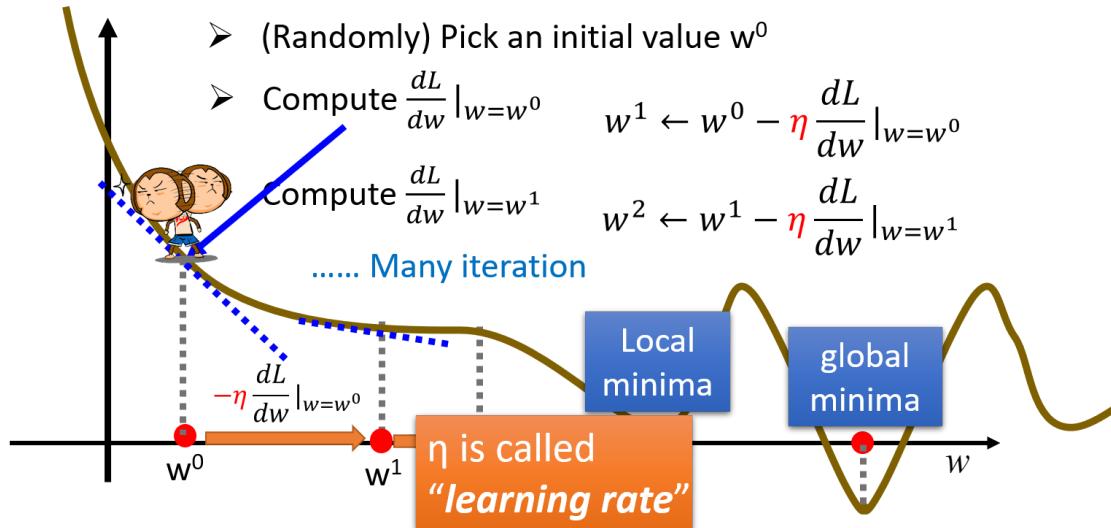
此时 w^i 对应的斜率为0，我们找到了一个极小值local minima，这就出现了一个问题，当微分为0的时候，参数就会一直卡在这个点上没有办法再更新了，因此通过gradient descent找出来的solution其实并不是最佳解global minima

但幸运的是，在linear regression上，是没有local minima的，因此可以使用这个方法

Step 3: Gradient Descent

$$w^* = \arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w :



两个参数的问题

今天要解决的关于宝可梦的问题，是含有two parameters的问题，即 $(w^*, b^*) = \arg \min_{w,b} L(w, b)$

当然，它本质上处理单个参数的问题是一样的

- 首先，也是随机选取两个初始值， w^0 和 b^0
- 然后分别计算 (w^0, b^0) 这个点上， L 对 w 和 b 的偏微分，即 $\frac{\partial L}{\partial w}|_{w=w^0, b=b^0}$ 和 $\frac{\partial L}{\partial b}|_{w=w^0, b=b^0}$
- 更新参数，当迭代跳出时， (w^i, b^i) 对应着极小值点

$$\begin{aligned} w^1 &= w^0 - \eta \frac{\partial L}{\partial w}|_{w=w^0, b=b^0} & b^1 &= b^0 - \eta \frac{\partial L}{\partial b}|_{w=w^0, b=b^0} \\ w^2 &= w^1 - \eta \frac{\partial L}{\partial w}|_{w=w^1, b=b^1} & b^2 &= b^1 - \eta \frac{\partial L}{\partial b}|_{w=w^1, b=b^1} \\ &\dots \\ w^{i+1} &= w^i - \eta \frac{\partial L}{\partial w}|_{w=w^i, b=b^i} & b^{i+1} &= b^i - \eta \frac{\partial L}{\partial b}|_{w=w^i, b=b^i} \\ \text{if } (\frac{\partial L}{\partial w} == 0 \& \& \frac{\partial L}{\partial b} == 0) \text{ then stop} \end{aligned} \tag{2}$$

实际上， L 的gradient就是微积分中的那个梯度的概念，即

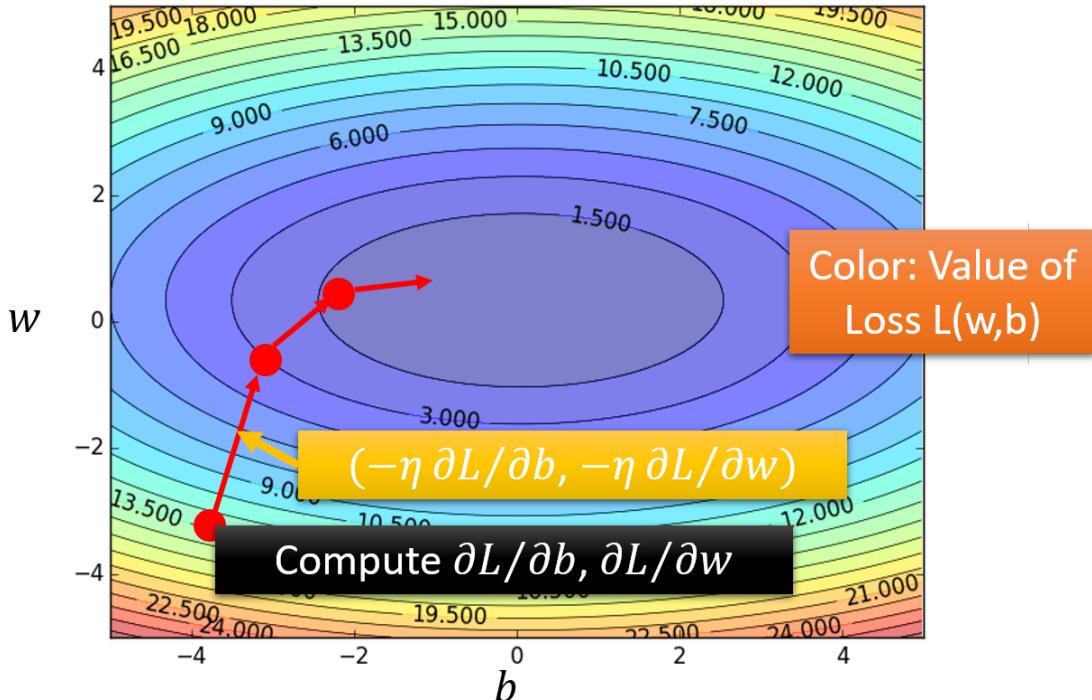
$$\nabla L = \left[\begin{array}{c} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{array} \right]_{gradient} \tag{3}$$

可视化效果如下：(三维坐标显示在二维图像中，loss的值用颜色来表示)

横坐标是 b ，纵坐标是 w ，颜色代表loss的值，越偏蓝色表示loss越小，越偏红色表示loss越大

每次计算得到的梯度gradient，即由 $\frac{\partial L}{\partial b}$ 和 $\frac{\partial L}{\partial w}$ 组成的vector向量，就是该等高线的法线方向(对应图中红色箭头的方向)；而 $(-\eta \frac{\partial L}{\partial b}, -\eta \frac{\partial L}{\partial w})$ 的作用就是让原先的 (w^i, b^i) 朝着gradient的方向即等高线法线方向前进，其中 η (learning rate)的作用是每次更新的跨度(对应图中红色箭头的长度)；经过多次迭代，最终gradient达到极小值点

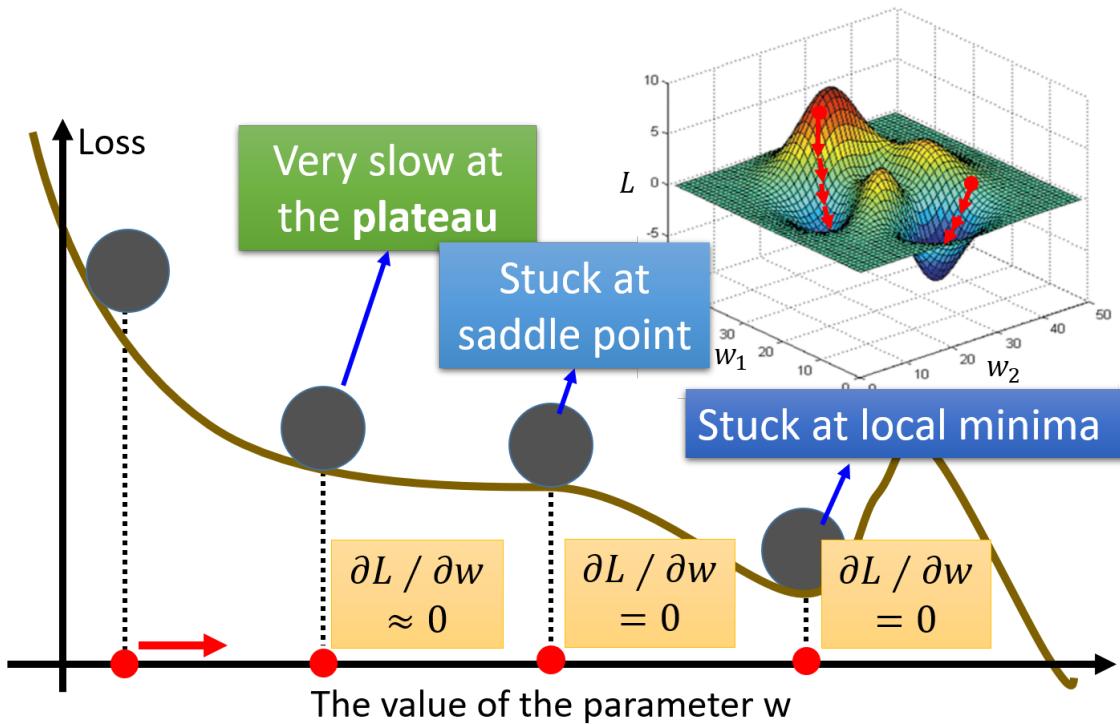
注：这里两个方向的 η (learning rate)必须保持一致，这样每次更新坐标的step size是等比例缩放的，保证坐标前进的方向始终和梯度下降的方向一致；否则坐标前进的方向将会发生偏移



Gradient Descent的缺点

gradient descent有一个令人担心的地方，也就是我之前一直提到的，它每次迭代完毕，寻找到的梯度为0的点必然是极小值点，local minima；却不一定是最小值点，global minima

这会造成一个问题是说，如果loss function长得比较坑坑洼洼(极小值点比较多)，而每次初始化 w^0 的取值又是随机的，这会造成每次gradient descent停下来的位置都可能是不同的极小值点；而且当遇到梯度比较平缓(gradient≈0)的时候，gradient descent也可能会效率低下甚至可能会stuck卡住；也就是说通过这个方法得到的结果，是看人品的(滑稽)



但是！在linear regression里，loss function实际上是convex的，是一个凸函数，是没有local optimal局部最优解的，他只有一个global minima，visualize出来的图像就是从里到外一圈一圈包围起来的椭圆形的等高线(就像前面的等高线图)，因此随便选一个起始点，根据gradient descent最终找出来的，都会是同一组参数

回到pokemon的问题上来

偏微分的计算

现在我们来求具体的L对w和b的偏微分

$$L(w, b) = \sum_{n=1}^{10} (\hat{y}^n - (b + w \cdot x_{cp}^n))^2 \quad (4)$$

$$\frac{\partial L}{\partial w} = \sum_{n=1}^{10} 2(\hat{y}^n - (b + w \cdot x_{cp}^n))(-x_{cp}^n)$$

$$\frac{\partial L}{\partial b} = \sum_{n=1}^{10} 2(\hat{y}^n - (b + w \cdot x_{cp}^n))(-1)$$

How's the results?

根据gradient descent，我们得到的 $y = b + w \cdot x_{cp}$ 中最好的参数是 $b=-188.4, w=2.7$

我们需要有一套评估系统来评价我们得到的最后这个function和实际值的误差error的大小；这里我们将training data里每一只宝可梦 i 进化后的实际cp值与预测值之差的绝对值叫做 e^i ，而这些误差之和

Average Error on Training Data为 $\sum_{i=1}^{10} e^i = 31.9$

What we really care about is the error on new data (testing data)

当然我们真正关心的是generalization的case，也就是用这个model去估测新抓到的pokemon，误差会有多少，这也就是所谓的testing data的误差；于是又抓了10只新的pokemon，算出来的Average

Error on Testing Data为 $\sum_{i=1}^{10} e^i = 35.0$ ；可见training data里得到的误差一般是要比testing data要小，

这也符合常识

How's the results? - Generalization

What we really care about is the error on new data (testing data)

$$y = b + w \cdot x_{cp}$$

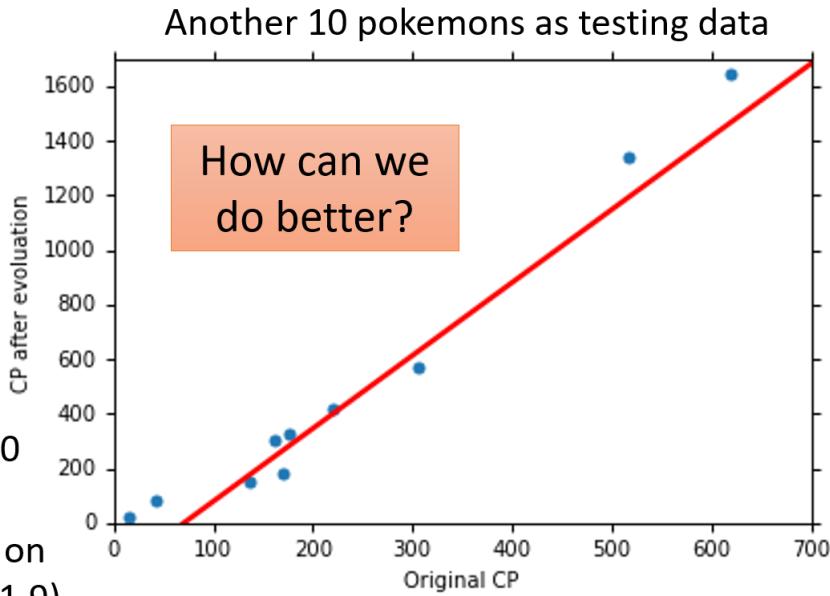
$$b = -188.4$$

$$w = 2.7$$

Average Error on Testing Data

$$= \frac{1}{10} \sum_{n=1}^{10} e^n = 35.0$$

> Average Error on Training Data (31.9)



How can we do better?

我们有没有办法做得更好呢？这时就需要我们重新去设计model；如果仔细观察一下上图的数据，就会发现在原先的cp值比较大和比较小的地方，预测值是相当不准的

实际上，从结果来看，最终的function可能不是一条直线，可能是稍微更复杂一点的曲线

考虑 $(x_{cp})^2$ 的model

Selecting another Model

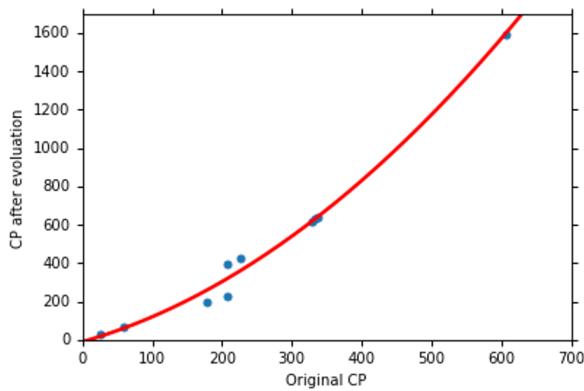
$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2$$

Best Function

$$b = -10.3$$

$$w_1 = 1.0, w_2 = 2.7 \times 10^{-3}$$

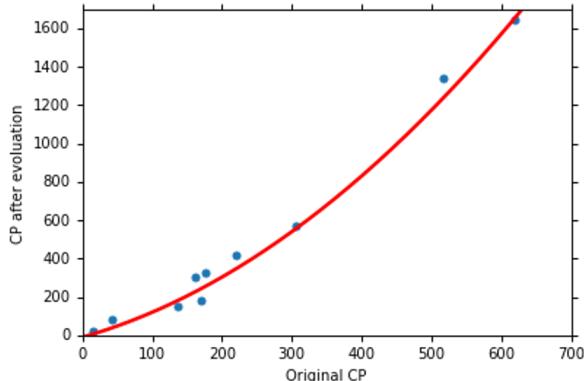
$$\text{Average Error} = 15.4$$



Testing:

$$\text{Average Error} = 18.4$$

Better! Could it be even better?



考虑 $(x_{cp})^3$ 的model

Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3$$

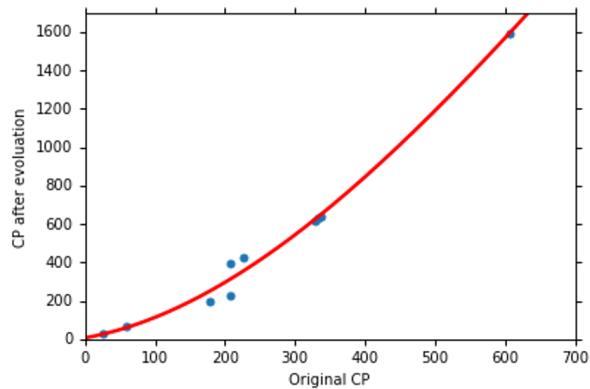
Best Function

$$b = 6.4, w_1 = 0.66$$

$$w_2 = 4.3 \times 10^{-3}$$

$$w_3 = -1.8 \times 10^{-6}$$

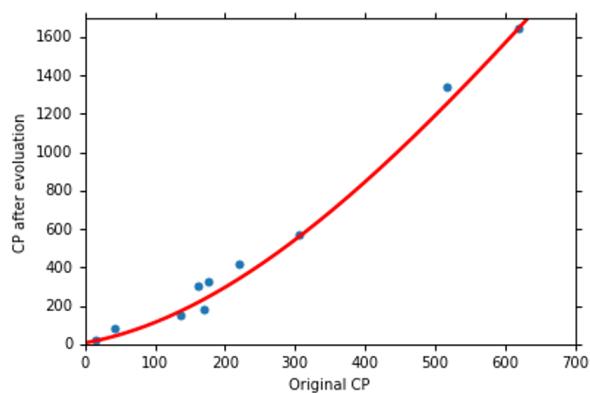
$$\text{Average Error} = 15.3$$



Testing:

$$\text{Average Error} = 18.1$$

Slightly better.
How about more
complex model?



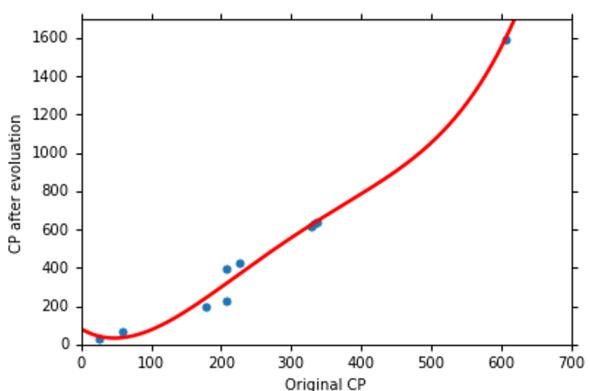
考慮 $(x_{cp})^4$ 的model

Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4$$

Best Function

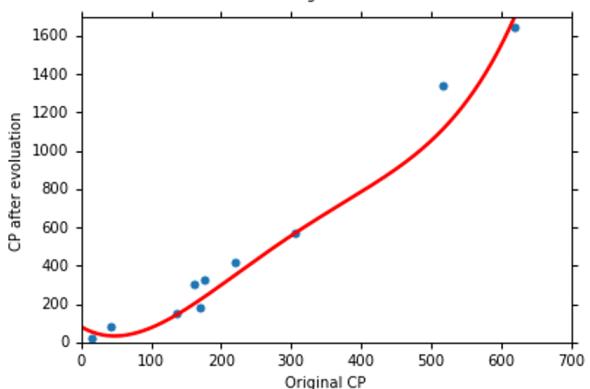
$$\text{Average Error} = 14.9$$



Testing:

$$\text{Average Error} = 28.8$$

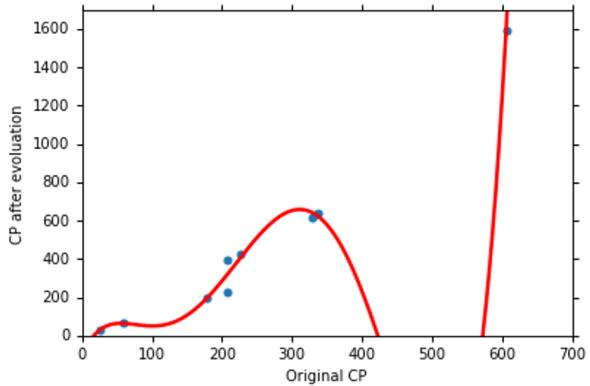
The results become
worse ...



考慮 $(x_{cp})^5$ 的model

Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4 + w_5 \cdot (x_{cp})^5$$



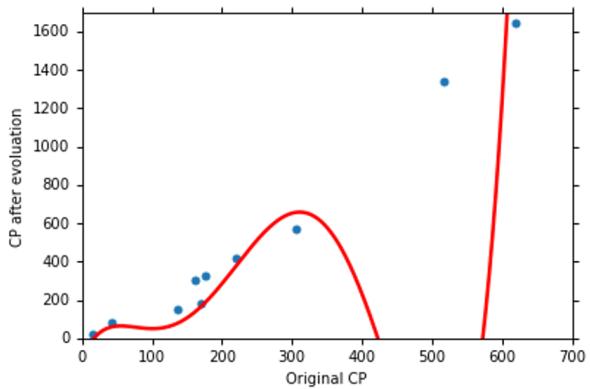
Best Function

Average Error = 12.8

Testing:

Average Error = 232.1

The results are so bad.



5个model的对比

这5个model的training data的表现：随着 $(x_{cp})^i$ 的高次项的增加，对应的average error会不断地减小；实际上这件事情非常容易解释，实际上低次的式子是高次的式子的特殊情况(令高次项 $(X_{cp})^i$ 对应的 w_i 为0，高次式就转化成低次式)

也就是说，在gradient descent可以找到best function的前提下(多次式为Non-linear model，存在local optimal局部最优解，gradient descent不一定能找到global minima)，function所包含的项的次数越高，越复杂，error在training data上的表现就会越来越小；但是，我们关心的不是model在training data上的error表现，而是model在testing data上的error表现

Model Selection

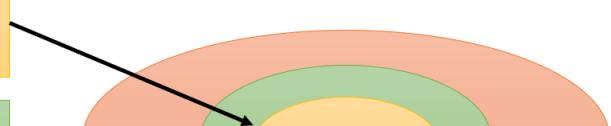
1. $y = b + w \cdot x_{cp}$

2. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2$

3. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3$

4. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4$

5. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4 + w_5 \cdot (x_{cp})^5$

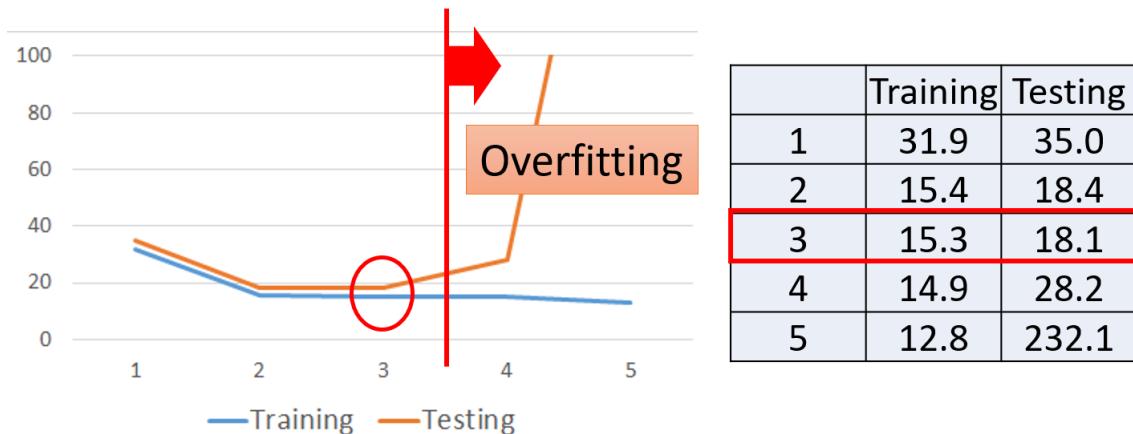


A more complex model yields lower error on training data.

If we can truly find the best function

在training data上, model越复杂, error就会越低;但是在testing data上, model复杂到一定程度之后, error非但不会减小,反而会暴增,在该例中,从含有 $(X_{cp})^4$ 项的model开始往后的model, testing data上的error出现了大幅增长的现象,通常被称为**overfitting过拟合**

Model Selection



A more complex model does not always lead to better performance on testing data.

This is **Overfitting**. → Select suitable model

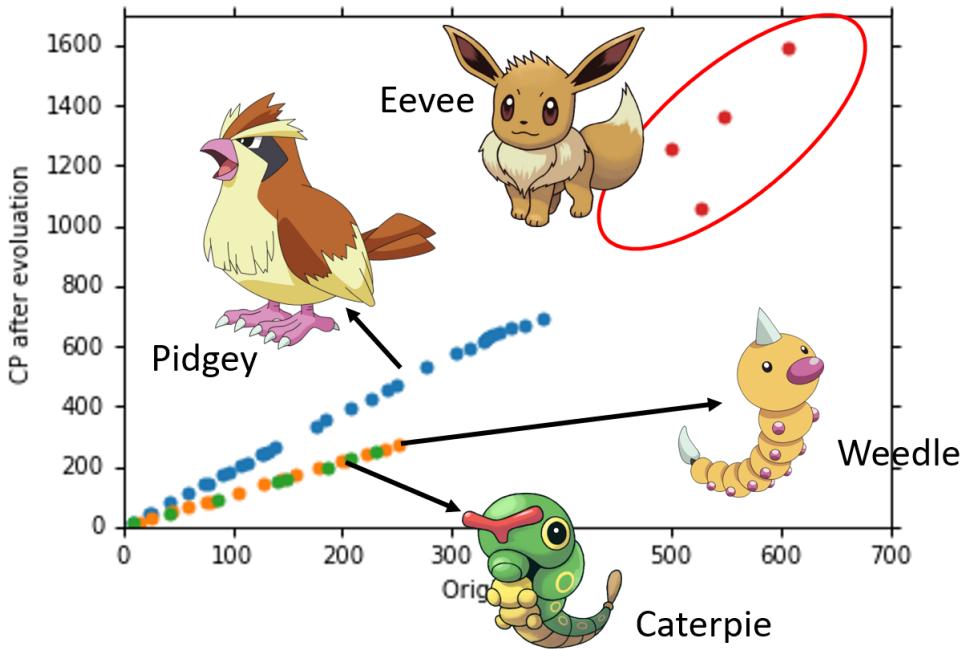
因此model不是越复杂越好,而是选择一个最适合的model,在本例中,包含 $(X_{cp})^3$ 的式子是最适合的model

进一步讨论其他参数

物种 x_s 的影响

之前我们的model只考虑了宝可梦进化前的cp值,这显然是不对的,除了cp值外,还受到物种 x_s 的影响

What are the hidden factors?



因此我们重新设计model:

$$\begin{aligned}
 & \text{if } x_s = \text{Pidgey} : \quad y = b_1 + w_1 \cdot x_{cp} \\
 & \text{if } x_s = \text{Weedle} : \quad y = b_2 + w_2 \cdot x_{cp} \\
 & \text{if } x_s = \text{Caterpie} : \quad y = b_3 + w_3 \cdot x_{cp} \\
 & \text{if } x_s = \text{Eevee} : \quad y = b_4 + w_4 \cdot x_{cp}
 \end{aligned} \tag{5}$$

也就是根据不同的物种，设计不同的linear model(这里 $x_s = \text{species of } x$)，那如何将上面的四个if语句合并成一个linear model呢？

这里引入 $\delta(\text{条件表达式})$ 的概念，当条件表达式为true，则 δ 为1；当条件表达式为false，则 δ 为0，因此可以通过下图的方式，将4个if语句转化成同一个linear model

有了上面这个model以后，我们分别得到了在training data和testing data上测试的结果：

Hp值 x_{hp} 、height值 x_h 、weight值 x_w 的影响

考虑所有可能有影响的参数，设计出这个最复杂的model：

$$\begin{aligned}
 & \text{if } x_s = \text{Pidgey} : \quad y' = b_1 + w_1 \cdot x_{cp} + w_5 \cdot (x_{cp})^2 \\
 & \text{if } x_s = \text{Weedle} : \quad y' = b_2 + w_2 \cdot x_{cp} + w_6 \cdot (x_{cp})^2 \\
 & \text{if } x_s = \text{Pidgey} : \quad y' = b_3 + w_3 \cdot x_{cp} + w_7 \cdot (x_{cp})^2 \\
 & \text{if } x_s = \text{Eevee} : \quad y' = b_4 + w_4 \cdot x_{cp} + w_8 \cdot (x_{cp})^2 \\
 & y = y' + w_9 \cdot x_{hp} + w_{10} \cdot (x_{hp})^2 + w_{11} \cdot x_h + w_{12} \cdot (x_h)^2 + w_{13} \cdot x_w + w_{14} \cdot (x_w)^2
 \end{aligned} \tag{6}$$

算出的training error=1.9，但是，testing error=102.3！**这么复杂的model很大概率会发生overfitting**(按照我的理解，overfitting实际上是我们多使用了一些input的变量或是变量的高次项使曲线跟training data拟合的更好，但不幸的是这些项并不是实际情况下被使用的，于是这个model在testing data上会表现得很糟糕)，overfitting就相当于是那个范围更大的韦恩图，它包含了更多的函数更大的范围，代价就是在准确度上表现得更糟糕

regularization解决overfitting(L2正则化解决过拟合问题)

regularization可以使曲线变得更加smooth, training data上的error变大,但是testing data上的error变小。有关regularization的具体原理说明详见下一部分

原来的loss function只考虑了prediction的error, 即 $\sum_i^n (\hat{y}^i - (b + \sum_j w_j x_j))^2$; 而regularization则是在原来的loss function的基础上加上了一项 $\lambda \sum (w_i)^2$, 就是把这个model里面所有的 w_i 的平方和用入加权(其中i代表遍历n个training data, j代表遍历model的每一项)

也就是说, 我们期待参数 w_i 越小甚至接近于0的function, 为什么呢?

因为参数值接近0的function, 是比较平滑的; 所谓的平滑的意思是, 当今天的输入有变化的时候, output对输入的变化是比较不敏感的

举例来说, 对 $y = b + \sum w_i x_i$ 这个model, 当input变化 Δx_i , output的变化就是 $w_i \Delta x_i$, 也就是说, 如果 w_i 越小越接近0的话, 输出对输入就越不sensitive敏感, 我们的function就是一个越平滑的function; 说到这里你会发现, 我们之前没有把bias——b这个参数考虑进去的原因是**bias的大小跟function的平滑程度是没有关系的**, bias值的大小只是把function上下移动而已

那为什么我们喜欢比较平滑的function呢?

如果我们有一个比较平滑的function, 由于输出对输入是不敏感的, 测试的时候, 一些noises噪声对这个平滑的function的影响就会比较小, 而给我们一个比较好的结果

$$y = b + \sum w_i x_i$$

The functions with
smaller w_i are better

$$L = \sum_n \left(\hat{y}^n - \left(b + \sum w_i x_i \right) \right)^2$$

$$+ \lambda \sum (w_i)^2$$

➤ Smaller w_i means ... smoother

$$y = b + \sum w_i x_i$$

$$y + \sum w_i \Delta x_i = b + \sum w_i (x_i + \Delta x_i)$$

➤ We believe smoother function is more likely to be correct

Do you have to apply regularization on bias?

注: 这里的 λ 需要我们手动去调整以取得最好的值

λ 值越大代表考虑smooth的那个regularization那一项的影响力越大, 我们找到的function就越平滑

观察下图可知, 当我们的 λ 越大的时候, 在training data上得到的error其实是越大的, 但是这件事情是非常合理的, 因为当 λ 越大的时候, 我们就越倾向于考虑 w 的值而越少考虑error的大小; 但是有趣的是, 虽然在training data上得到的error越大, 但是在testing data上得到的error可能会是比较小的

下图中, 当 λ 从0到100变大的时候, training error不断变大, testing error反而不断变小; 但是当 λ 太大的时候(> 100), 在testing data上的error就会越来越大

我们喜欢比较平滑的function, 因为它对noise不那么sensitive; 但是我们又不喜欢太平滑的function, 因为它就失去了对data拟合的能力; 而function的平滑程度, 就需要通过调整 λ 来决定, 就像下图中, 当 $\lambda=100$ 时, 在testing data上的error最小, 因此我们选择 $\lambda=100$

注: 这里的error指的是 $\frac{1}{n} \sum_{i=1}^n |\hat{y}^i - y^i|$

conclusion总结

关于pokemon的cp值预测的流程总结：

- 根据已有的data特点(labeled data, 包含宝可梦及进化后的cp值), 确定使用supervised learning 监督学习
- 根据output的特点(输出的是scalar数值), 确定使用regression回归(linear or non-linear)
- 考虑包括进化前cp值、species、hp等各方面变量属性以及高次项的影响, 我们的model可以采用这些input的一次项和二次型之和的形式, 如:

$$\begin{aligned}
 & \text{if } x_s = \text{Pidgey} : \quad y' = b_1 + w_1 \cdot x_{cp} + w_5 \cdot (x_{cp})^2 \\
 & \text{if } x_s = \text{Weedle} : \quad y' = b_2 + w_2 \cdot x_{cp} + w_6 \cdot (x_{cp})^2 \\
 & \text{if } x_s = \text{Pidgey} : \quad y' = b_3 + w_3 \cdot x_{cp} + w_7 \cdot (x_{cp})^2 \\
 & \text{if } x_s = \text{Eevee} : \quad y' = b_4 + w_4 \cdot x_{cp} + w_8 \cdot (x_{cp})^2 \\
 & y = y' + w_9 \cdot x_{hp} + w_{10} \cdot (x_{hp})^2 + w_{11} \cdot x_h + w_{12} \cdot (x_h)^2 + w_{13} \cdot x_w + w_{14} \cdot (x_w)^2
 \end{aligned} \tag{7}$$

而为了保证function的平滑性, loss function应使用regularization, 即

$L = \sum_{i=1}^n (\hat{y}^i - y^i)^2 + \lambda \sum_j (w_j)^2$, 注意bias——参数b对function平滑性无影响, 因此不额外再次计入loss function(y的表达式里已包含w、b)

- 利用gradient descent对regularization版本的loss function进行梯度下降迭代处理, 每次迭代都减去L对该参数的微分与learning rate之积, 假设所有参数合成一个vector:
 $[w_0, w_1, w_2, \dots, w_j, \dots, b]^T$, 那么每次梯度下降的表达式如下:

$$\text{梯度 : } \nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_0} \\ \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_j} \\ \vdots \\ \frac{\partial L}{\partial b} \end{bmatrix}_{\text{gradient}} \quad \text{gradient descent : } \begin{bmatrix} w'_0 \\ w'_1 \\ w'_2 \\ \vdots \\ w'_j \\ \vdots \\ b' \end{bmatrix}_{L=L'} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_j \\ \vdots \\ b \end{bmatrix}_{L=L_0} - \eta \begin{bmatrix} \frac{\partial L}{\partial w_0} \\ \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial w_j} \\ \vdots \\ \frac{\partial L}{\partial b} \end{bmatrix}_{L=L_0} \tag{8}$$

当梯度稳定不变时, 即 ∇L 为0时, gradient descent便停止, 此时如果采用的model是linear的, 那么vector必然落于global minima处(凸函数); 如果采用的model是Non-linear的, vector可能会落于local minima处(此时需要采取其他办法获取最佳的function)

假定我们已经通过各种方法到达了global minima的地方, 此时的vector:

$[w_0, w_1, w_2, \dots, w_j, \dots, b]^T$ 所确定的那个唯一的function就是在该 λ 下的最佳 f^* , 即loss最小

- 这里 λ 的最佳数值是需要通过我们不断调整来获取的, 因此令 λ 等于0, 10, 100, 1000, ...不断使用gradient descent或其他算法得到最佳的parameters: $[w_0, w_1, w_2, \dots, w_j, \dots, b]^T$, 并计算出这组参数确定的function—— f^* 对training data和testing data上的error值, 直到找到那个使testing data的error最小的 λ , (这里一开始 $\lambda=0$, 就是没有使用regularization时的loss function)

注: 引入评价 f^* 的error机制, 令 $\text{error} = \frac{1}{n} \sum_{i=1}^n |\hat{y}^i - y^i|$, 分别计算该 f^* 对training data和testing data(more important)的error(f^*)大小

先设定 λ ->确定loss function->找到使loss最小的 $[w_0, w_1, w_2, \dots, w_j, \dots, b]^T$ ->确定function->计算error->重新设定新的 λ 重复上述步骤->使testing data上的error最小的 λ 所对应的 $[w_0, w_1, w_2, \dots, w_j, \dots, b]^T$ 所对应的function就是我们能够找到的最佳的function

本章节总结:

- Pokémon: Original CP and species almost decide the CP after evolution
 - There are probably other hidden factors
- Gradient descent
 - More theory and tips in the following lectures
- Overfitting and Regularization
- We finally get average error = 11.1 on the testing data
 - How about new data? Larger error? Lower error?(larger->need validation)
- Next lecture: Where does the error come from?
 - More theory about overfitting and regularization
 - The concept of validation(用来解决new data的error高于11.1的问题)

附: Regularization(L1 L2 正则化解决overfitting)

Regularization -> redefine the loss function

关于overfitting的问题，很大程度上是由于曲线为了更好地拟合training data的数据，而引入了更多的高次项，使得曲线更加“蜿蜒曲折”，反而导致了对testing data的误差更大

回过头来思考，我们之前衡量model中某个function的好坏所使用的loss function，仅引入了真实值和预测值差值的平方和这一个衡量标准；我们想要避免overfitting过拟合的问题，就要使得高次项对曲线形状的影响尽可能小，因此我们要在loss function里引入高次项(非线性部分)的衡量标准，也就是将高次项的系数也加权放进loss function中，这样可以使得训练出来的model既满足预测值和真实值的误差小，又满足高次项的系数尽可能小而使曲线的形状比较稳定集中

以下图为例，如果loss function仅考虑了 $(\hat{y} - y)^2$ 这一误差衡量标准，那么拟合出来的曲线就是红色虚线部分(过拟合)，而过拟合就是所谓的model对training data过度自信，非常完美的拟合上了这些数据，如果具备过拟合的能力，那么这个方程就可能是一个比较复杂的非线性方程，正是因为这里的 x^3 和 x^2 使得这条虚线能够被弯来弯去，所以整个模型就会特别努力地去学习作用在 x^3 和 x^2 上的c、d参数。**但是在这个例子里，我们期望模型要学到的却是这条蓝色的曲线。因为它能更有效地概括数据。而且只需要一个 $y = a + bx$ 就能表达出数据的规律。**

或者是说，蓝色的线最开始时，和红色线同样也有c、d两个参数，可是最终学出来时，c 和 d 都学成了0，虽然蓝色方程的误差要比红色大，但是概括起数据来还是蓝色好

这也是我们通常采用的方法，我们不可能一开始就否定高次项而直接只采用低次线性表达式的model，因为有时候真实数据的确是符合高次项非线性曲线的分布的；而如果一开始直接采用高次非线性表达式的model，就很有可能造成overfitting，在曲线偏折的地方与真实数据的误差非常大。我们的目标应该是这样的：

在无法确定真实数据分布的情况下，我们尽可能去改变loss function的评价标准

- 我们的model的表达式要尽可能的复杂，包含尽可能多的参数和尽可能多的高次非线性项；
- 但是我们的loss function又有能力去控制这条曲线的参数和形状，使之不会出现overfitting过拟合的现象；
- 在真实数据满足高次非线性曲线分布的时候，loss function控制训练出来的高次项的系数比较大，使得到的曲线比较弯折起伏；

- 在真实数据满足低次线性分布的时候，loss function控制训练出来的高次项的系数比较小甚至等于0，使得到的曲线接近linear分布

那我们如何保证能学出来这样的参数呢？这就是 L1 L2 正规化出现的原因。

之前的loss function仅考虑了 $(\hat{y} - y)^2$ 这一误差衡量标准，而**L1 L2正规化**就是在这个loss function的后面多加了一个东西，即model中跟高次项系数有关的表达式；

- L1正规化即加上 $\lambda \sum |w_j|$ 这一项，loss function变成 $L = \sum_{i=1}^n (\hat{y}^i - y^i)^2 + \lambda \sum_j |w_j|$ ，即n个training data里的数据的真实值与预测值差值的平方和加上权重下的model表达式中所有项系数的绝对值之和
- L2正规化即加上 $\lambda \sum (w_j)^2$ 这一项，loss function变成 $L = \sum_{i=1}^n (\hat{y}^i - y^i)^2 + \lambda \sum_j (w_j)^2$ ，即n个training data里的数据的真实值与预测值差值的平方和加上权重下的model表达式中所有项系数的平方和

相对来说，L2要更稳定一些，L1的结果则不那么稳定，如果用p表示正规化程度，上面两式可总结如下： $L = \sum_{i=1}^n (\hat{y}^i - y^i)^2 + \lambda \sum_j (w_j)^p$