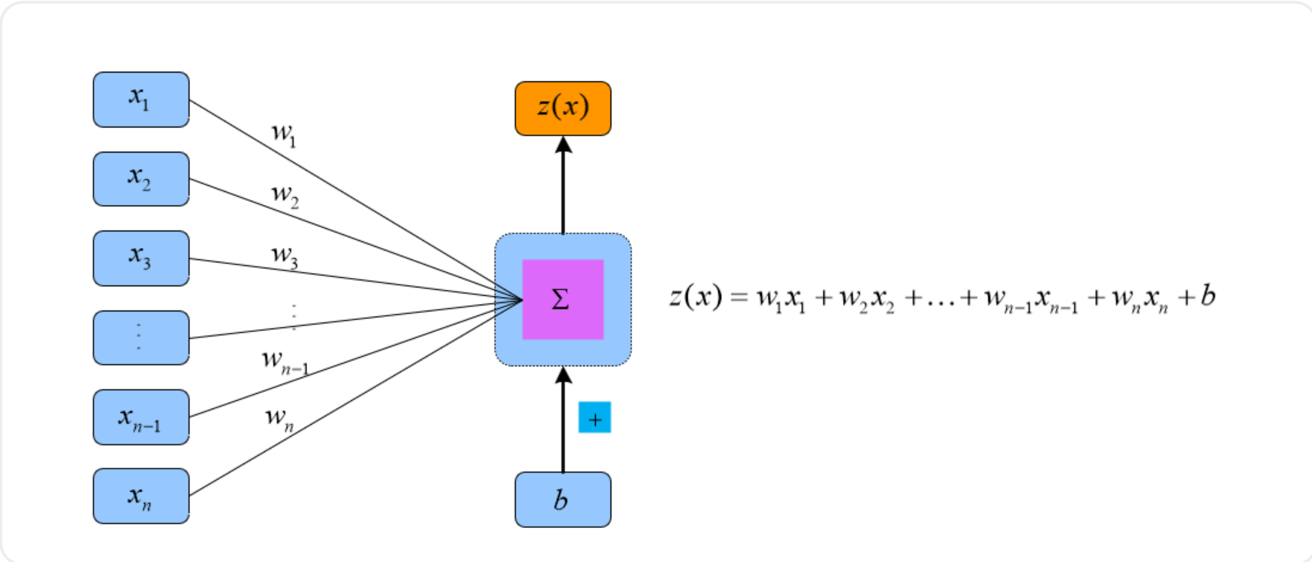
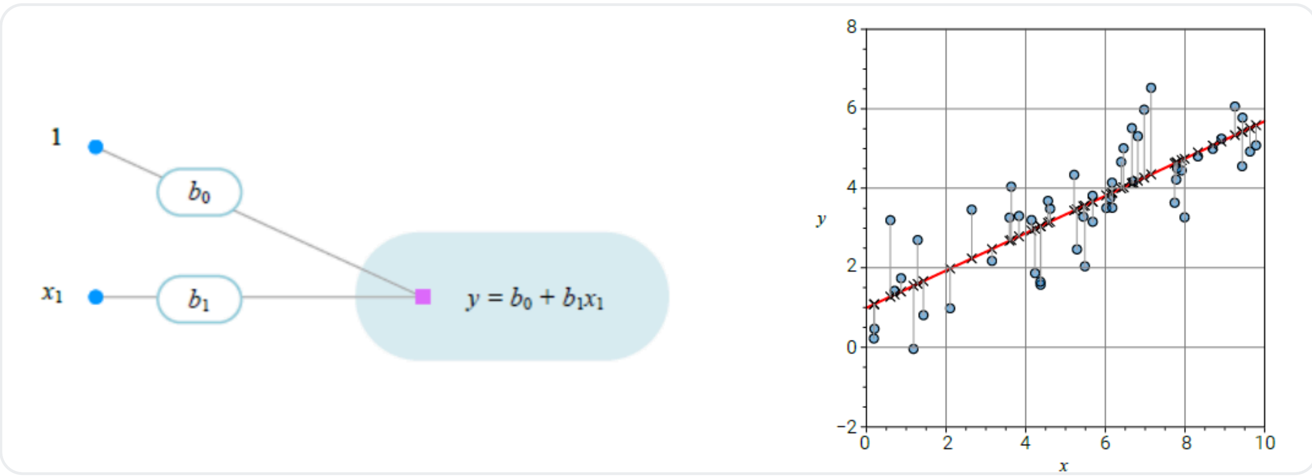
从线性回归到神经网络之逻辑回归篇

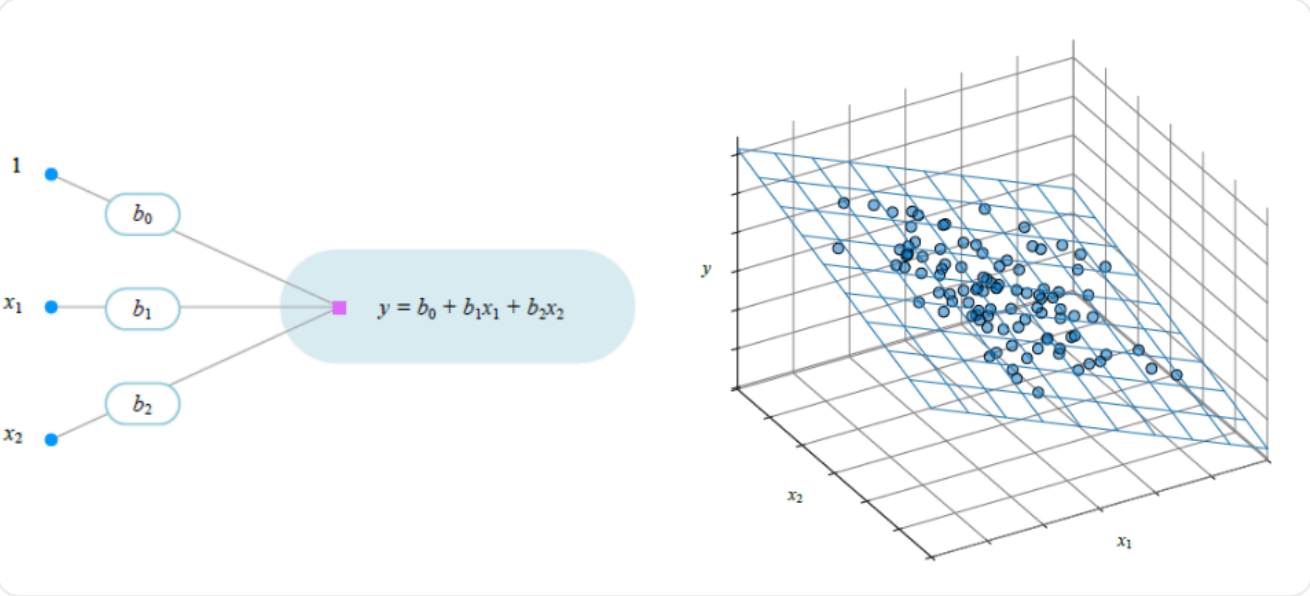
线性回归是机器学习中最基础的模型之一，它主要用于预测数值型的目标变量。



在二维情况下，我们可以通过寻找一条直线，将二维直角坐标系上的数据划分为两部分。这种方法可以帮助我们预测新的数据点在直线的哪一侧。例如，在房价预测中，我们可以将房屋的面积作为x轴，房价作为y轴，通过线性回归找到一条直线，预测不同面积房屋的价格。



在三维情况下，我们可以寻找一个二维平面，将三维空间划分为两部分。这种方法可以应用于更复杂的数据分析，如三维空间中的物体分类。例如，我们可以将物体的长度、宽度和高度作为三个坐标轴，通过线性回归找到一个平面，将不同类别的物体分开。



对于更高维度的空间，我们则试图寻找一个超平面将空间划分为两部分。这种方法可以应用于更复杂的数据分析，如高维空间中的数据分类。例如，在图像识别中，我们可以将图像的像素值作为高维空间的坐标，通过线性回归找到一个超平面，将不同类别的图像分开。

那什么是逻辑回归呢?接下来我们来看逻辑回归,它在线性回归的基础上，通过sigmoid函数，将该线性函数转化为非线性函数。这种方法使得逻辑回归可以用于分类问题，如二分类和多分类。例如，在邮件分类中，我们可以将邮件的特征作为输入，通过逻辑回归模型预测邮件是否为垃圾邮件。

所以它名为回归的分类算法

名为回归的分类算法

- 知道逻辑回归的损失函数、优化方法
- 知道逻辑回归的应用场景
- 应用LogisticRegression实现逻辑回归预测
- 知道精确率、召回率等指标的区别
- 知道如何解决样本不均衡情况下的评估
- 会绘制ROC曲线图形

我们已经通过线性回归模型成功解决了回归问题，本课就来处理分类问题。分类问题与回归问题，是机器学习两大主要应用。

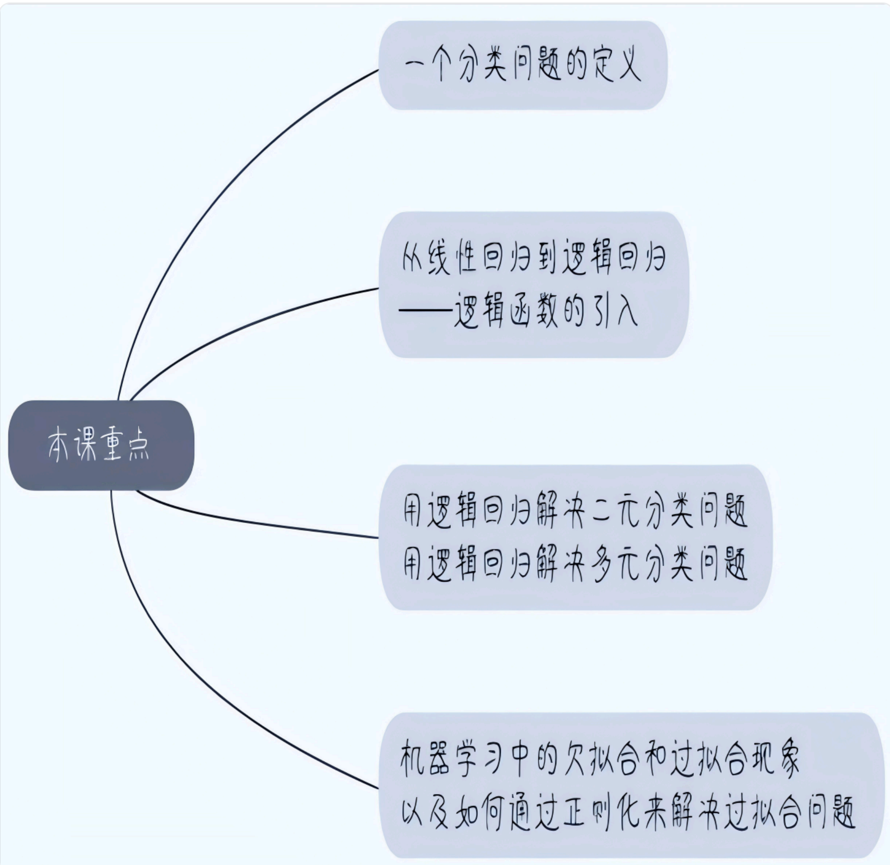
分类问题覆盖面很广泛：有二元分类，如根据考试成绩推断是否被录取、根据消费记录判断信用卡是否可以申请，以及预测某天是否将发生地震等；有多元分类，如消费群体的划分、个人信用的评级等；还有图像识别、语音识别等，在本质上也是很多个类别的分类问题。

垃圾分类器帮助市民确定垃圾的类别

本课要讲的专用于分类的机器学习算法，叫逻辑回归（logistic regression），简称Logreg，

你刚才说，机器学习两大主要应用是回归问题和分类问题，可你又说这个逻辑回归算法，专用于分类问题，这我就不明白了，专用于分类问题的算法，为什么叫逻辑回归，不叫‘逻辑分类’算法呢？”

逻辑回归算法的本质其实仍然是回归。这个算法也是通过调整权重w和偏置b来找到线性函数来计算数据样本属于某一类的概率。比如二元分类，一个样本有60%的概率属于A类，有20%的概率属于B类，算法就会判断样本属于A类。不过，在介绍这些细节之前，还是先看本课重点吧。



从一个例子开始：

问题的定义:判断客户是否患病

小冰最近在她的网店取得了不错的成绩，这让她的朋友们也开始寻求合作机会。其中一位朋友向她介绍了一款新型血压计，并希望小冰能帮忙推广。为了更精准地了解潜在客户的需求，小冰决定在她的朋友圈发起一项关于心脏健康的小调查。

她设计了一份问卷，包含了一些由她朋友提供的、涉及专业医学知识的问题。虽然其中很多术语对小冰来说有些陌生，但她相信这些内容能够帮助她更好地理解朋友们的健康状况。

小冰发出了1000份问卷，最终收到了数百份反馈。这些数据不仅帮助她了解了朋友们的心脏健康情况，还为她提供了推广新型血压计的宝贵信息。通过这次调查，小冰对如何更有效地推广产品有了更深的认识。

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
57	1	0	140	192	0	1	148	0	0.4	1	0	1	1
56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
44	1	1	120	263	0	1	173	0	0	2	0	3	1
52	1	2	172	199	1	1	162	0	0.5	2	0	3	1
57	1	2	150	168	0	1	174	0	1.6	2	0	2	1
54	1	0	140	239	0	1	160	0	1.2	2	0	2	1
48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
49	1	1	130	266	0	1	171	0	0.6	2	0	2	1
64	1	3	110	211	0	0	144	1	1.8	1	0	2	1
58	0	3	150	283	1	0	162	0	1	2	0	2	1
50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
58	0	2	120	340	0	1	172	0	0	2	0	2	1
66	0	3	150	226	0	1	114	0	2.6	0	0	2	1

以下是测评中各列数值的中文含义。

- age:年龄。
- sex:性别。
- cp:胸痛类型。
- trestbps:休息时血压。
- chol:胆固醇。
- fbs:血糖。
- restecg:心电图。
- thalach:最大心率。
- exang:运动后心绞痛。
- oldpeak:运动后ST段压低。
- slope:运动高峰期ST段的斜率。
- ca:主动脉荧光造影染色数。
- thal:缺陷种类。
- target:0代表无心脏病，1代表有心脏病。
- 在这个问卷中要注意以下两点。
- 从A栏到M栏，是调查的信息，包括年龄、性别、心脏功能的一些指标等，从机器学习的角度看就是特征字段。
- 问卷的最后一栏，第N栏的target字段，是调查的目标，也就是潜在客户患病还是未患病，这是标签字段。
- 在收回的问卷中，有以下3种情况。
    - 有一部分人已经是心脏病患者，这批人是我们的潜在客户群，则target =1。
    - 有一部分人确定自己没有心脏问题，那么目前他们可能就不大需要血压计这个产品，则target =0。
    - 还有一部分人只填好了调查表的前一部分，但是最后一个问题，是否有心脏病?他们没有填写答案，target字段是空白的。可能他们自己不知道，也可能他们不愿意提供这个答案给我们。这些数据就是无标签的数据。

现在我们已经掌握了这么多‘有标签’的数据，那么能不能用刚才你所说的逻辑回归模型，对没有提供答案的人以及未来的潜在客户进行是否有心脏病的推测。如果能够推知这些潜在客户是否患心脏病，就等于知道这些潜在客户是否需要心脏保健相关产品(血压计)。这是多么精准的营销策略啊!”

“当然可以。“只要你的数据是准确的，这个情况就很适合用逻辑回归来解决。你刚才说问卷中的很多专业性内容你不是很懂，那没有关系。机器学习的一大优势，就是可以对我们本身并不是特别理解的数据，也产生精准的洞见。”

这几百张已收回的有标签(就是已经回答了最后一个问题：是否患心脏病?)的调查问卷，正是珍贵的机器学习‘训练集’和‘验证集’。下面就让逻辑回归算法来完成一个专业医生才能够做出的判断。”

介绍算法之前，同学们先思考一下什么是事物的“类别”

机器学习中的分类

事物的类别，这个概念并不难理解。正确的分类观是建立科学体系、训练逻辑思维能力的重要一步。从小学自然科学课，老师就开始教孩子们如何给各种事物、现象分类。
机器学习中的分类问题的覆盖面要比我们所想象的还广泛得多，下面举几个例子。

- 根据客户的收入、存款、性别、年龄以及流水，为客户的信用等级分类。
- 读入图片，为图片内容分类(猫、狗、虎、兔)。
- 手写数字识别，输出类别0～9。
- 手写文字识别，也是分类问题，只是输出类别有很多，有成千上万个类。

而机器学习的分类方法，也是要找到一个合适的函数，拟合输入和输出的关系，输入一个或一系列事物的特征，输出这个事物的类别。
对于计算机来说，输入的特征必须是它所能够识别的。例如，我们无法把人(客户、患者)输入计算机，那么只能找到最具代表性的特征(年龄、血压、账户存款余额等)转换成数值后输入模型。

输入特征，通过函数输出类别

所有的特征，都要转换成数值形式，才易于被机器学习，机器不能够识别“男”“女”,只能识别“1”“2”。这种文本到数值的转换是必做的特征工程。 而输出，则是离散的数值，如0、1、2、3等分别对应不同类别。例如，二元分类中的成功/失败、健康/患病，及多元分类中的猫、狗、长颈鹿等。这些类别之间是互斥关系，如一个动物是狗，就不能同时是猫；一个患者被诊断为患心脏病，就不能同时被认为是健康的。

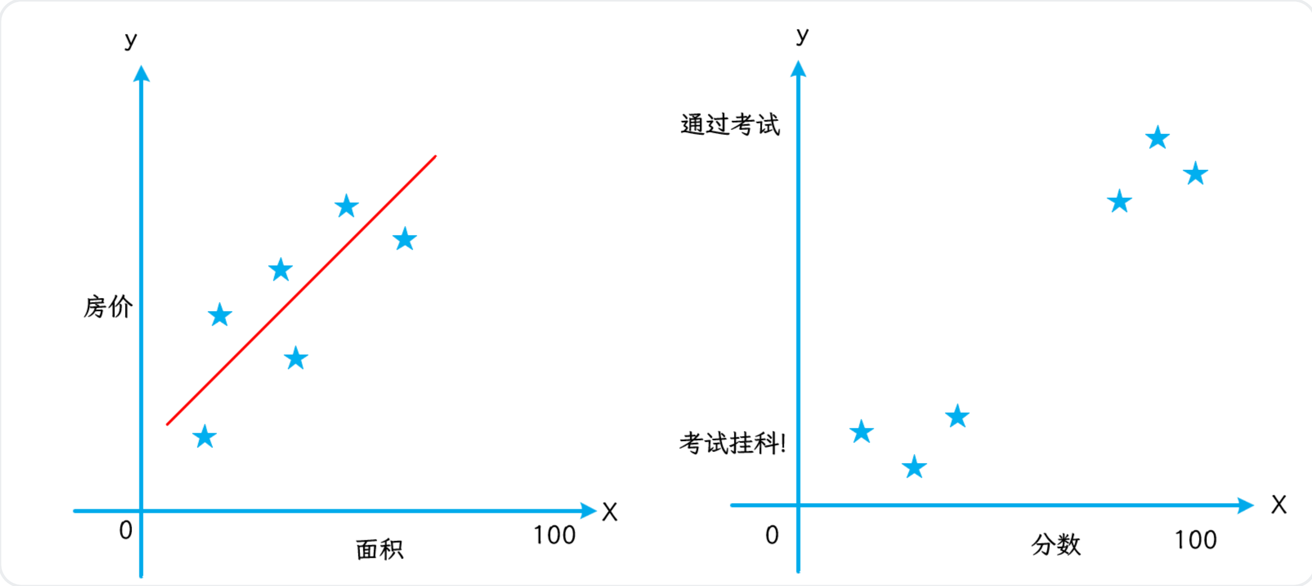
这里先给一点逻辑回归的算法细节，在输出明确的离散分类值之前，算法首先输出的其实是一个可能性，你们可以把这个可能性理解成一个概率。

- 机器学习模型根据输入数据判断一个人患心脏病的可能性为80%,那么就把这个人判定为“患病”类，输出数值1。
- 机器学习模型根据输入数据判断一个人患心脏病的可能性为30%,那么就把这个人判定为“健康”类，输出数值0。

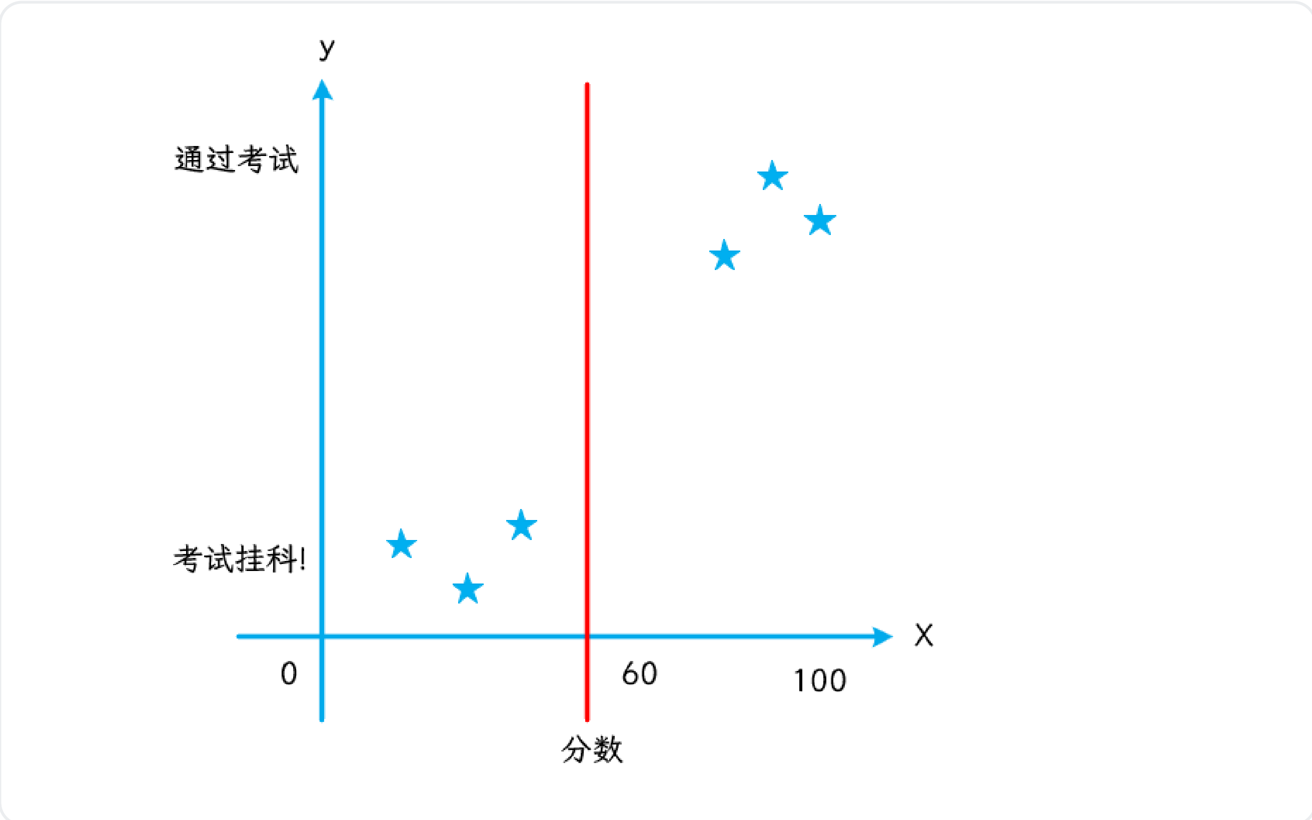
机器学习的分类过程，也就是确定某一事物隶属于某一个类别的可能性大小的过程。

用线性回归+阶跃函数完成分类

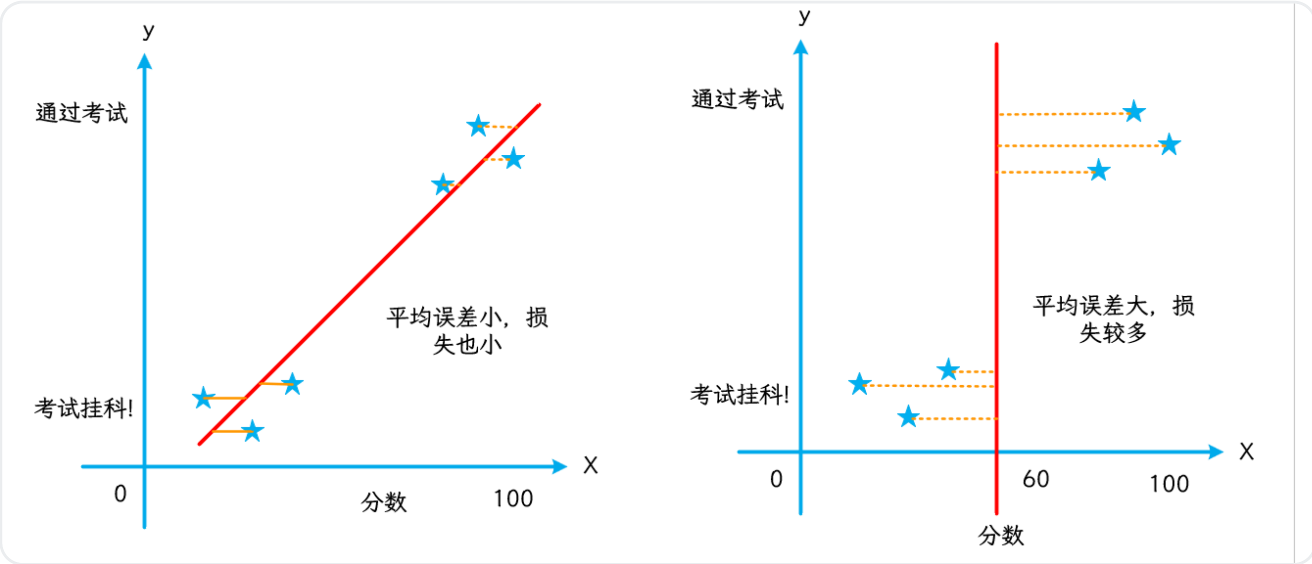
温故而知新，学习逻辑回归模型先从复习线性回归模型开始。同学们看看下页这两个图有何区别。左边的x和y之间明显呈现出连续渐变的特征，是线性关系，适合用回归模型建模；而右边的x是0～100的值，代表成绩，y则不是0～1的连续值，y只有两个结果，要么通过考试
(y=1),要么考试挂科(y=0)



现在的问题是：如何对左边这个进行划分呢？“这个分类问题建模太简单了，我一眼此时小冰举手，说：就判断出来了。这个模型就是两句话，x大于等于60,y为1;x小于60,y为0。这个模型和回归有点像，也能用一条直线表示，你看。”说着，直接在图中画上了一条竖线，如下图所示。

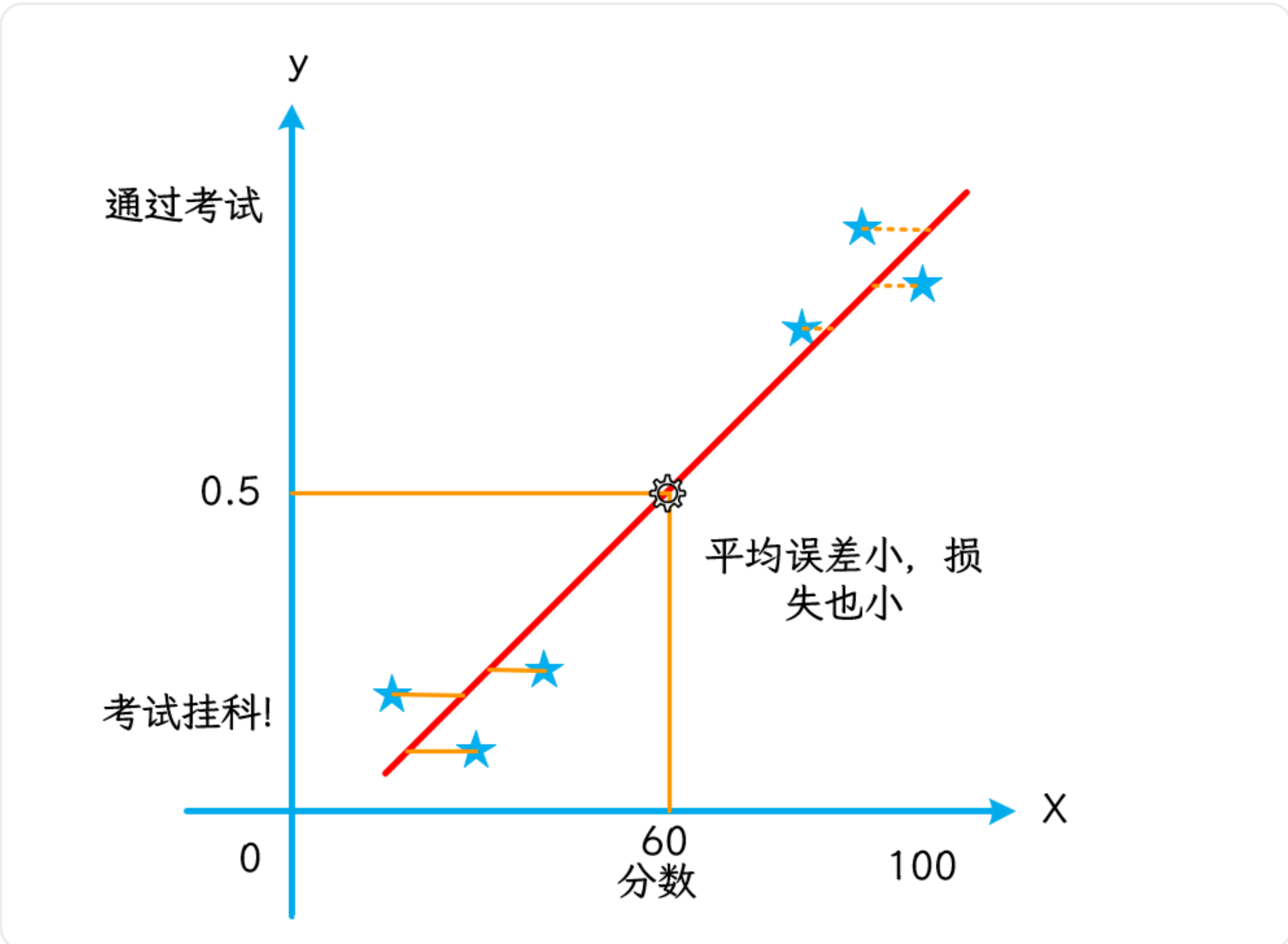


还是要虚心一些。你说这个分类问题简单，是不错，但是你画的这条回归线，可是大错特错了。如果用线性回归来拟合这个通过考试与否的问题，最佳的回归线应该这样去画。”如下面左图所示：



“那么,怎么把这条线性回归函数线转换成逻辑分类器呢?这就要涉及这一课中最重要的逻辑函数了。”
“讲逻辑函数之前，还是先仔细看看这条线性图像。这条线上，有一个神奇的点，能通过成绩来预测考试成功还是失败。你们猜猜是哪个点呢？”

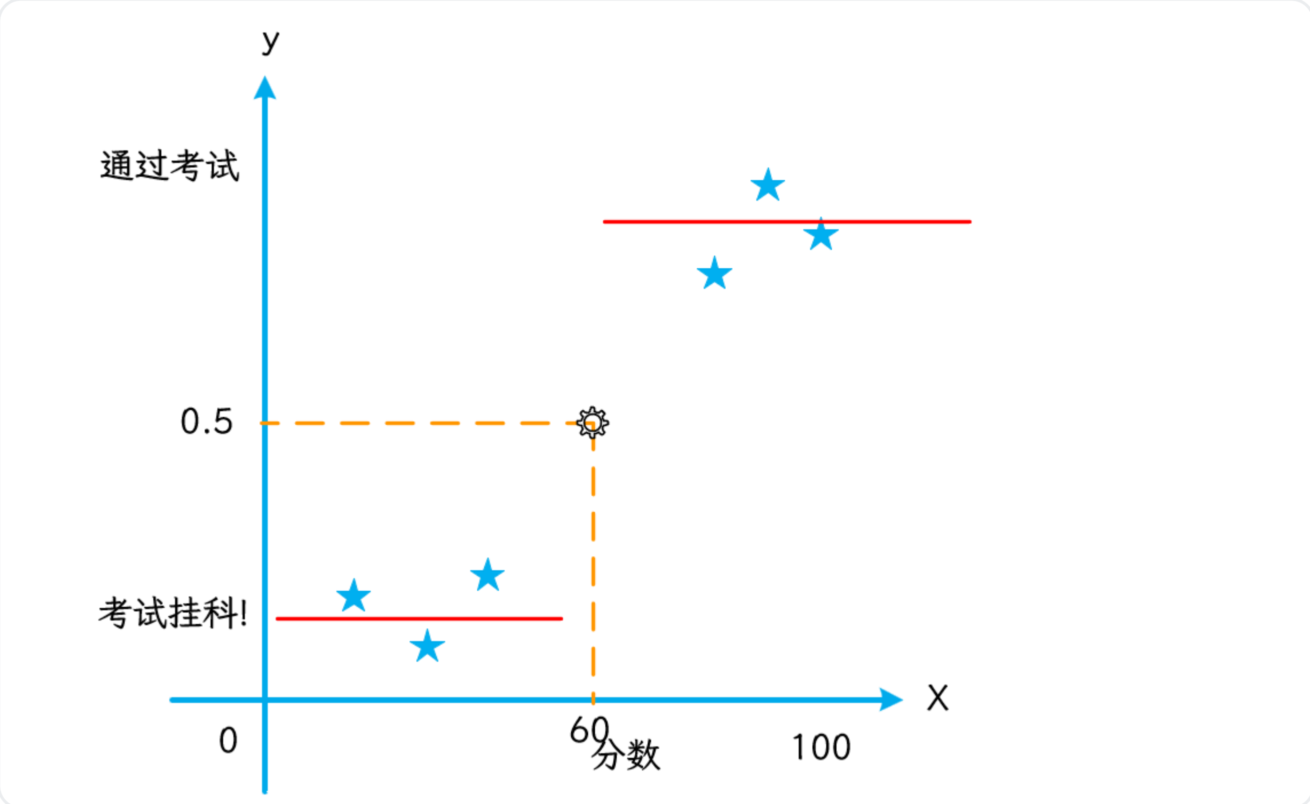
60分这个点在这个例子中，的确是相当重要的点。也就是说，当考试成绩在60分左右的时候，考试成功的概率突然增大了。我们注意到在60分左右的两个人，一个通过了考试，一个则挂科了。那么根据此批数据，在60分这个点，考试通过的概率为50%。而60分这个x点，用线性回归函数做假设函数时，所对应的y值刚好是0.5,也就是50%(如下图所示)。”



“总结一下：我们是用线性回归的方法，对这个分类问题进行拟合，得到了一个回归函数。这个函数的参数如何确定，前面已经讲过。但是，这个模型很明显还是不太理想的，对大多数具体的点来说效果不太好。这是因为y值的分布连续性很差，所以要额外处理一下。”

“如何处理呢?大家思考一下分类问题和回归问题的本质区别：对于分类问题来说，尽管分类的结果和数据的特征之间仍呈现相关关系，但是y的值不再是连续的，是0～1的跃迁。但是在这个过程中，什么仍然是连续的呢?”
答案是概率：“概率?”其实，随着成绩的上升，通过考试的概率是逐渐升高的，当达到一个关键点(阈值),如此例中的60分的时候，通过考试的概率就超过了0.5。那么从这个点开始，之后y的预测值都为1。”

因此，只要将线性回归的结果做一个简单的转换，就可以得到分类器的结果。这个转换如下图所示。

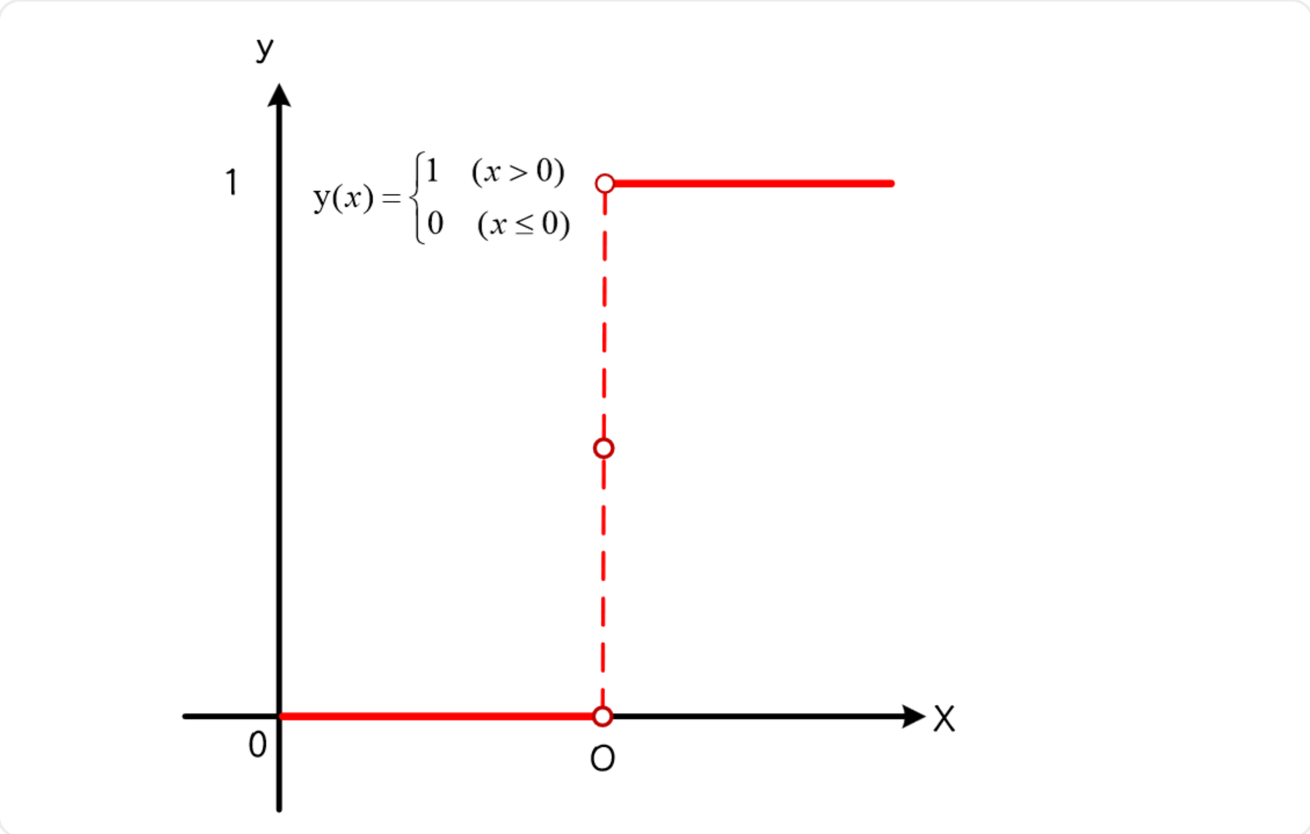


这可以分为以下两种情况。

❶ 线性回归模型输出的结果大于0.5,分类输入1。

❷ 线性回归模型输出的结果小于0.5,分类输入0。

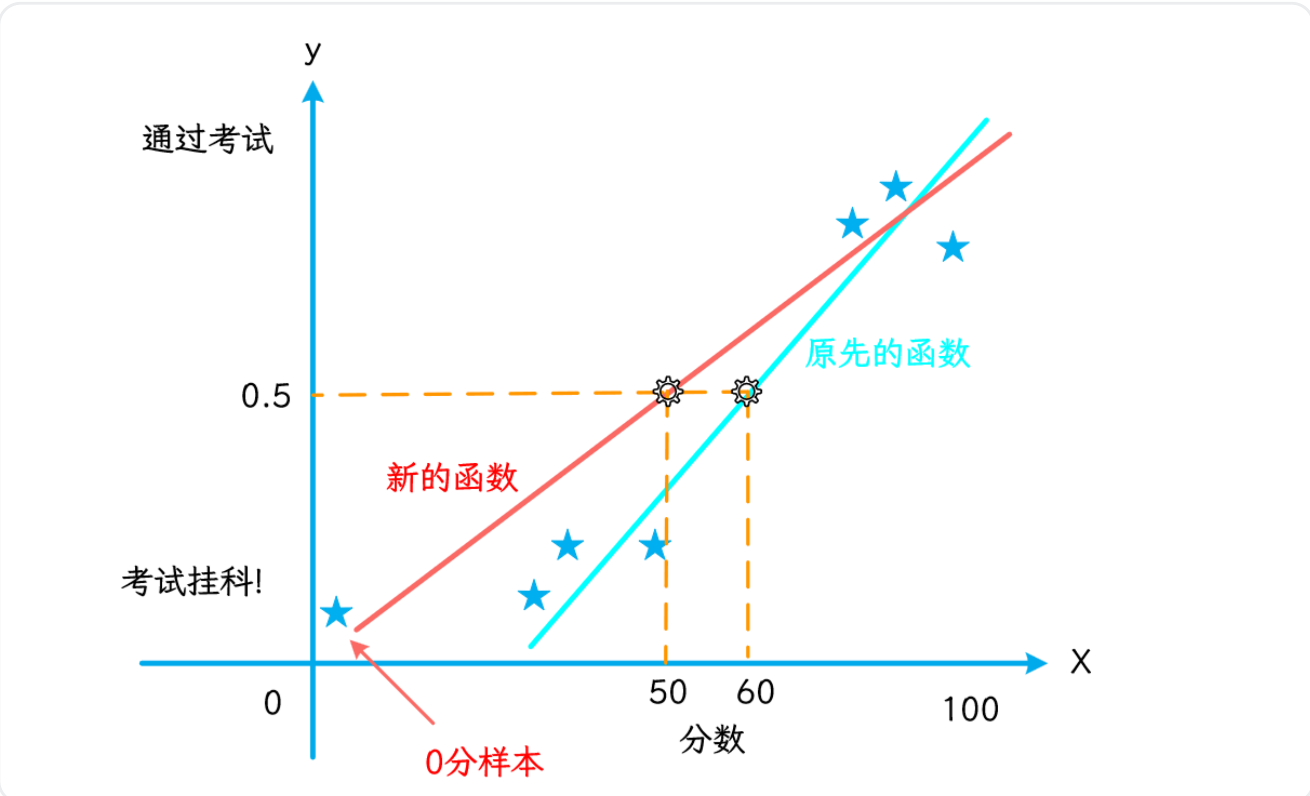
这里我们可以用阶跃函数。首先利用线性回归模型的结果，找到了概率为0.5时所对应的特征点(分数=60分),然后把线性的连续值，转换为0/1的分类值，也就是true/false逻辑值，去更好地拟合分类数据。



对于分类编码为0、1的标签，一般分类阈值取0.5;如果分类编码为-1、+1,则分类阈值取0。通过这个阈值把回归的连续性结果转换成了分类的阶跃性、离散性结果。

对目前这个根据考试成绩预测结果的问题，分类成功率为100%。至此，似乎任务完成了！

实则不然。直接应用线性回归+阶跃函数这个组合模型作为分类器还是会有局限性。你们看看下图这个情况。如果在这个数据集中，出现了一个意外：有一位同学考了0分！



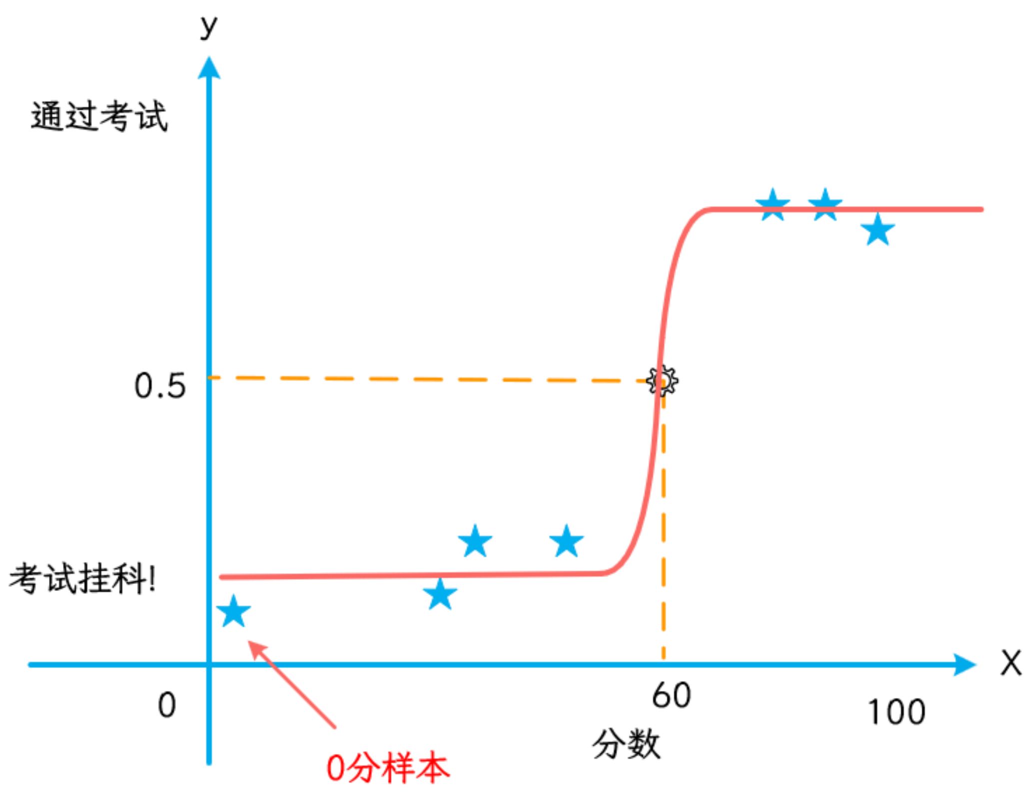
一个0分离群样本（特例）竟然让模型的通过分数产生大幅移动

这个同学考了0分不要紧，但是因为数据集的样本数量本来就不多，一个离群的样本会造成线性回归模型发生改变。为了减小平均误差，回归线现在要往0分那边稍作移动。因此，概率0.5这个阈值点所对应的x分数也发生了移动，目前变成了50分。这样，如果有一个同学考了51分，本来是没有及格，却被这个模型判断为及格(通过考试的概率高于0.5)。这个结果与我们的直觉不符。

通过Sigmiod函数进行转换

因此，我们需要想出一个办法对当前模型进行修正，使之既能够更好地拟合以概率为代表的分类结果，又能够抑制两边比较接近0和1的极端例子，使之钝化，同时还必须保持函数拟合时对中间部分数据细微变化的敏感度。

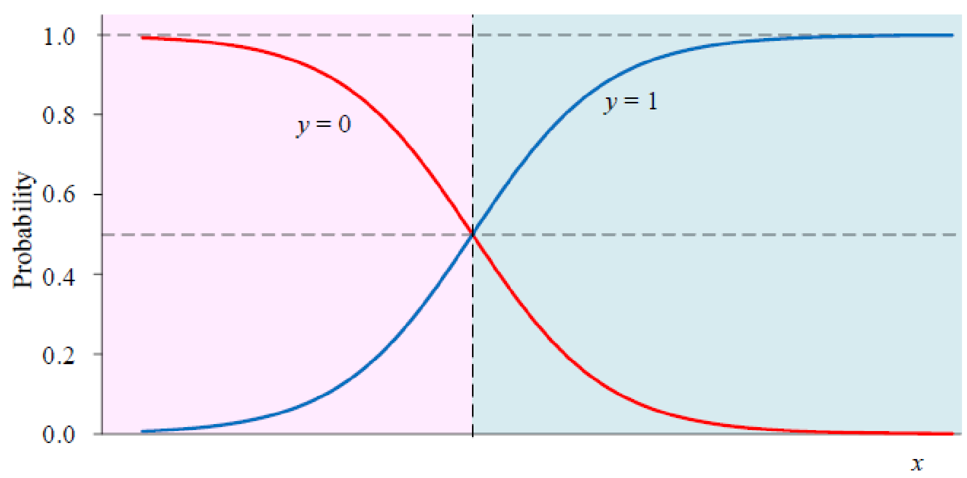
要达到这样效果的函数是什么样的呢？请看下面的图



如果有一个对分类值域两边不敏感的函数，就再也不惧离群样本了

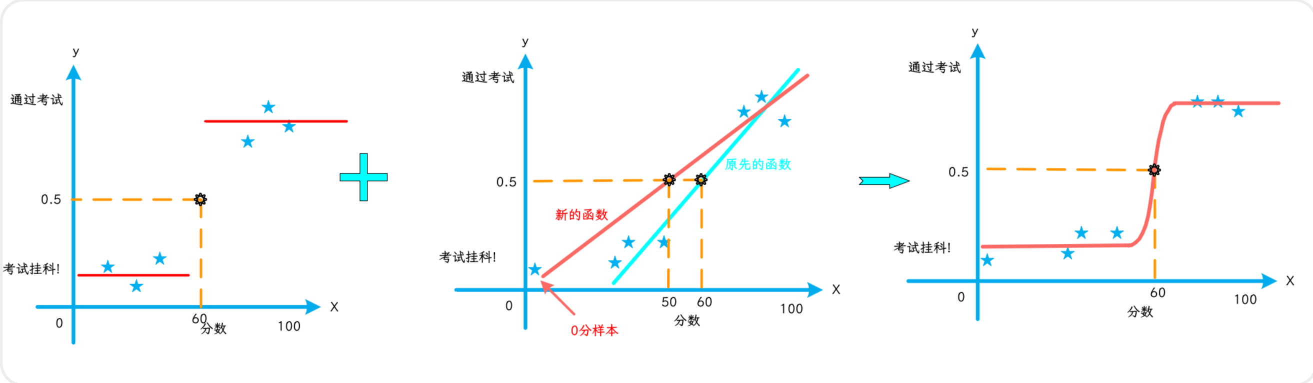
如果有这种S形的函数，不管有多少个同学考0分，都不会对这个函数的形状产生大的影响。因为这个函数对于靠近0分和100分附近的极端样本是很不敏感的，类似样本的分类概率将无限逼近0或1，样本个数再多也无所谓。但是在0.5这个分类概率临界点附近的样本将对函数的形状产生较大的影响。也就是说，样本越靠近分类阈值，函数对它们就越敏感。

这种S形的函数，被称为logistic function，翻译为逻辑函数。在机器学习中，logistic function被广泛应用于逻辑回归分类和神经网络激活过程。



标签为1 和为0 的概率关系

而且，这个函数像是线性函数和阶跃函数的结合，如下图所示。



这个逻辑函数像是线性函数和阶跃函数的结合

有这样的函数吗?

恰好，有一个符合需要的函数，这个函数叫Sigmoid函数。它是最为常见的机器学习逻辑函数。Sigmoid函数的公式为:

$$g(z)=\frac{1}{1+e^{-z}}$$

为什么这里自变量的符号用的是z而不是x?因为它是一个中间变量，代表的是线性回归的结果。而这里g(z)输出的结果是一个0~1的数字，也代表着分类概率。

Sigmoid函数的代码实现很简单:

```
# Sigmoid 函数的代码实现很简单 :
# 首先定义一个Sigmoid函数，输入Z，返回y'
def sigmoid(z):
    y_hat = 1/(1+ np.exp(-z))
    return y_hat
```

通过Sigmoid函数就能够比阶跃函数更好地把线性函数求出的数值，转换为一个0~1的分类概率值!

逻辑回归的假设函数

有了Sigmoid函数，就可以开始正式建立逻辑回归的机器学习模型。上一课说过，建立机器学习的模型，重点要确定假设函数h(x),来预测y'。

总结一下上面的内容，把线性回归和逻辑函数整合起来，形成逻辑回归的假设函数。

(1)首先通过线性回归模型求出一个中间值z, $z=w_0x_1+w_1x_1+\cdots+W_nx_n+b=W^TX$。它是一个连续值，区间并不在[0,1]之间，可能小

于0或者大于1,范围从无穷小到无穷大。

(2）然后通过逻辑函数把这个中间值 z 转化成 $0\sim1$ 的概率值，以提高拟合效果 $g(z)=\frac{1}{1+e^{-z}}$ 。

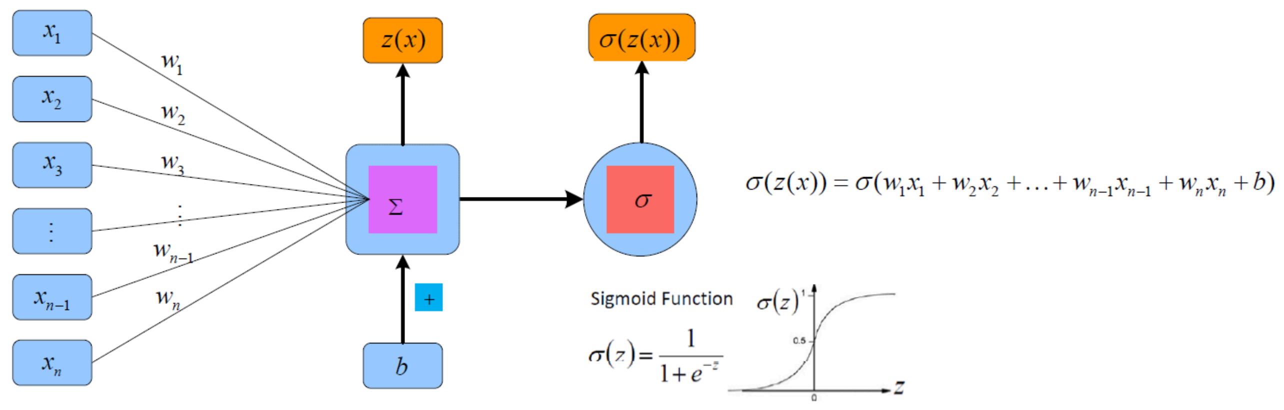
(3）结合步骤（1）和（2），把新的函数表示为假设函数的形式:

$$h(x)=\frac{1}{1+\mathrm{e}^{-(\boldsymbol{W}^{\top}X)}}$$

这个值也就是逻辑回归算法得到的 y' 。

(4）最后还要根据 y 所代表的概率，确定分类结果。

1. 如果h(x)值大于等于0.5,分类结果为1。
2. 如果h(x)值小于0.5,分类结果为0。因此，逻辑回归模型包含4个步骤，如下图所示。



综上，逻辑回归所做的事情，就是把线性回归输出的任意值，通过数学上的转换，输出为0～1的结果，以体现二元分类的概率(严格来说为后验概率)。

上述过程中的关键在于选择Sigmoid函数进行从线性回归到逻辑回归的转换。Sigmoid函数的优点如下。

1. Sigmoid函数是连续函数，具有单调递增性(类似于递增的线性函数)。
2. Sigmoid函数具有可微性，可以进行微分，也可以进行求导。
3. 输出范围为[0,1]，结果可以表示为概率的形式，为分类输出做准备。
4. 抑制分类的两边，对中间区域的细微变化敏感，这对分类结果拟合效果好。

逻辑回归的损失函数

“同学们，现在有了逻辑回归的假设函数，下一步我们将做什么?”答:“确定函数的具体参数。”

下一步是确定函数参数的过程，也同样是通过计算假设函数带来的损失，找到最优的w和b的过程，也就是把误差最小化。拿客户是否患心脏病的例子来说，对于已经被确诊的患者，假设函数h(x)预测出来的概率P，其实也就是y'，越接近1，则误差越小；对于健康的患者，假设函数h(x)预测出来的概率P，即

y'，越接近0，则误差越小。那么如何确定损失?”大家回答: “还需要一个损失函数。”

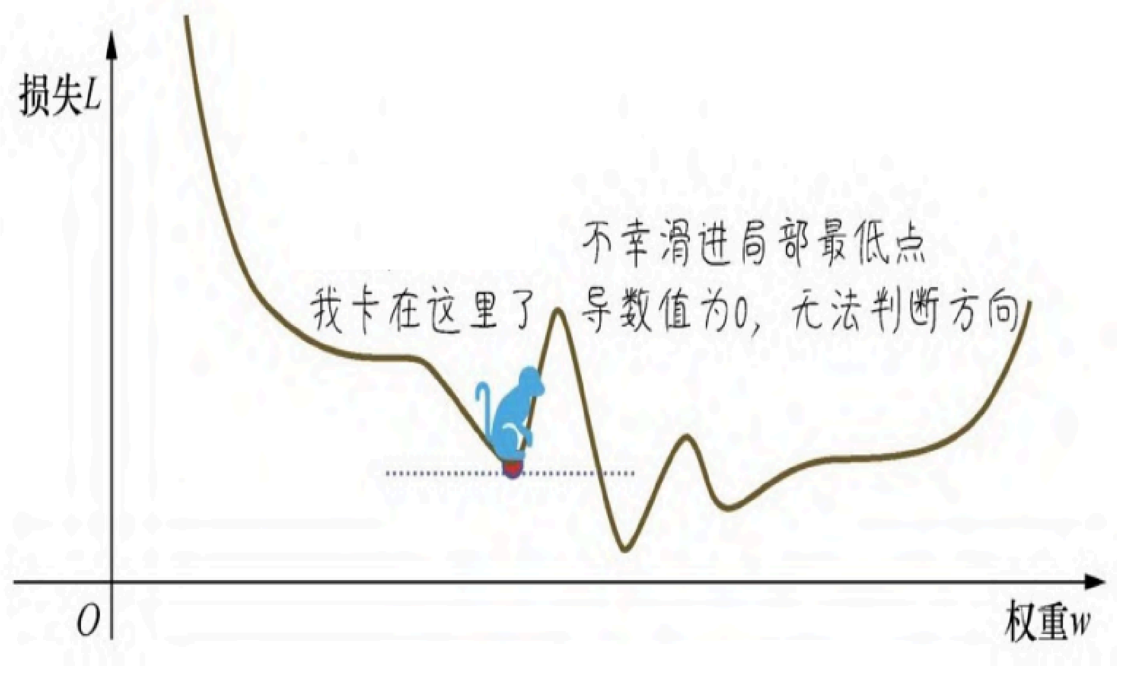
把训练集中所有的预测所得概率和实际结果的差异求和，并取平均值，就可以得到平均误差，这就是逻辑回归的损失函数:

$$L(w,b)=\frac{1}{N}\sum_{(x,y)\in D}\text{Loss}(h(x),y)=\frac{1}{N}\sum_{(x,y)\in D}\text{Loss}\left(y',y\right)$$

这个损失函数和线性回归的损失函数是完全一致的。那么同学们是否还记得线性回归的损失函数是什么?

是均方误差函数MSE。

然而，在逻辑回归中，不能使用MSE。因为经过了一个逻辑函数的转换之后，MSE对于w和b而言，不再是一个凸函数，这样的话，就无法通过梯度下降找到全局最低点，

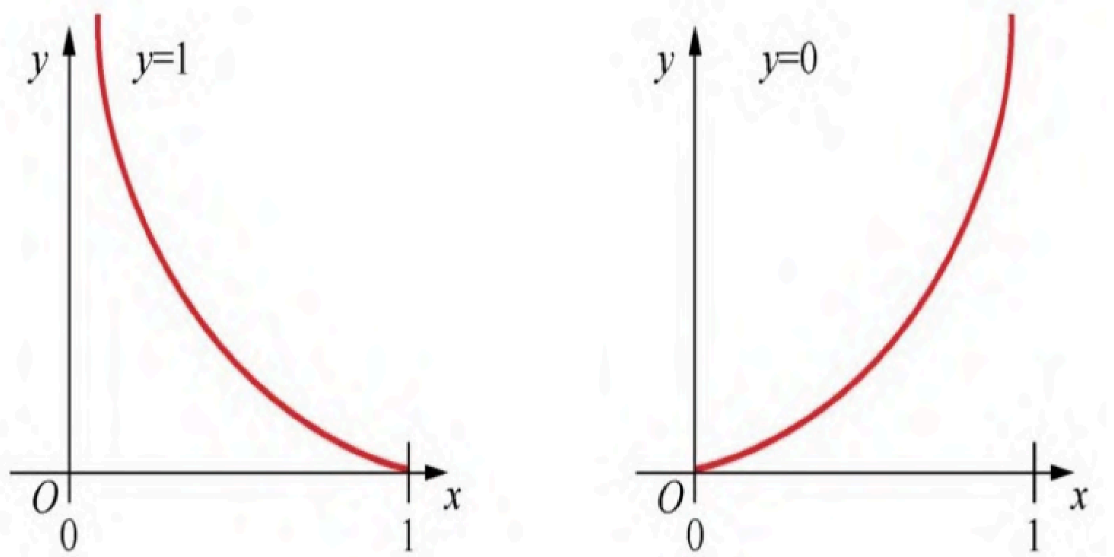


MSE对于逻辑回归不再是凸函数

为了避免陷入局部最低点，我们为逻辑回归选择了符合条件的新的损失函数，公式如下:

$$\begin{cases} y=1, \operatorname{Loss}(h(x), y)=-\log (h(x)) \\ y=0, \operatorname{Loss}(h(x), y)=-\log (1-h(x)) \end{cases}$$

有人可能想问，怎么一下子出来两个函数?这么奇怪!
应该说，这是一个函数在真值为0或者1的时候的两种情况。这不就是以自然常数为底数的对数吗?而且，从图形上看(如下图所示)，这个函数将对错误的猜测起到很好的惩罚效果。



1. 如果真值是1，但假设函数预测概率接近于0的话，得到的损失值将是巨大的。
2. 如果真值是0，但假设函数预测概率接近于1的话，同样将得到天价的损失值。
   而上面这种对损失的惩罚力度正是我们所期望的。整合起来，逻辑回归的损失函数如下:

$$L(w, b)=-\frac{1}{N} \sum_{(x, y) \in D}\left[y^{*} \log (h(x))+(1-y)^{*} \log (1-h(x))\right]$$

这个公式其实等价于上面的损失函数在0、1时的两种情况，同学们可以自己代入y=0和y=1两种取值分别推演一下。

下面是逻辑回归的损失函数的Python实现：

```python
# 然后定义损失函数
def loss_function(X,y,w,b):
    y_hat = sigmoid(np.dot(X,w) + b) # Sigmoid逻辑函数 + 线性函数(wX+b)得到y'
    loss = -(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)) # 计算损失
    cost = np.sum(loss) / X.shape[0]  # 整个数据集平均损失
    return cost # 返回整个数据集平均损失
```

逻辑回归的梯度下降

我们所选择的损失函数经过 Sigmoid 变换之后是可微的，也就是说每一个点都可以求导，而且它是凸函数，存在全局最低点。梯度下降的目的就是把 w 和 b 调整、再调整，直至最低的损失点。

逻辑回归的梯度下降过程和线性回归一样，也是先进行微分，然后把计算出来的导数乘以一个学习速率α，通过不断的迭代，更新w和b，直至收敛。

逻辑回归的梯度计算公式如下：

$$\text{梯度} = h'(x) = \frac{\partial}{\partial w}L(w,b) = \frac{\partial}{\partial w}\left\{-\frac{1}{N}\sum_{(x,y)\in D}[y*\log(h(x)) + (1-y)*\log(1-h(x))]\right\}$$

这里省略了大量计算微分的细节，直接给出推导后的结果：

$$\text{梯度} = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - h\left(x^{(i)}\right)\right)\cdot x^{(i)}$$

这个公式和线性回归的梯度公式形式非常一致。一致性强，就让人觉着舒服，

言归正传，引入学习速率之后，参数随梯度变化而更新的公式如下：

$$w = w - \alpha\cdot\frac{\partial}{\partial w}L(w)$$

即：

$$w = w - \frac{\alpha}{N}\sum_{j=1}^{N}\left(y^{(i)} - \left(w\cdot x^{(i)}\right)\right)\cdot x^{(i)}$$

下面的代码段实现了一个完整的逻辑回归的梯度下降过程：

```python
# 然后构建梯度下降的函数
def gradient_descent(X,y,w,b,lr,iter) : #定义逻辑回归梯度下降函数
    l_history = np.zeros(iter) # 初始化记录梯度下降过程中误差值(损失)的数组
    w_history = np.zeros((iter,w.shape[0],w.shape[1])) # 初始化权重记录的数组
    b_history = np.zeros(iter) # 初始化记录梯度下降过程中偏置的数组
    for i in range(iter): #进行机器训练的迭代
        y_hat = sigmoid(np.dot(X,w) + b) #Sigmoid逻辑函数+线性函数(wX+b)得到y'
        derivative_w = np.dot(X.T,((y_hat-y)))/X.shape[0]  # 给权重向量求导
        derivative_b = np.sum(y_hat-y)/X.shape[0] # 给偏置求导
        w = w - lr * derivative_w # 更新权重向量，lr即学习速率alpha
        b = b - lr * derivative_b   # 更新偏置，lr即学习速率alpha
        l_history[i] =  loss_function(X,y,w,b) # 梯度下降过程中的损失
        print ("轮次", i+1 , "当前轮训练集损失: ",l_history[i])
        w_history[i] = w # 梯度下降过程中权重的历史 请注意w_history和w的形状
        b_history[i] = b # 梯度下降过程中偏置的历史
    return l_history, w_history, b_history
```

这段代码和上一课中线性回归的梯度下降函数代码段整体结构一致，都是对 weight.T 进行点积。此处只是增加了 Sigmoid 函数的逻辑转换，然后使用了新的损失函数。在实战环节中，我还会更为稍微详细地解释里面的一些细节。

到此为止，逻辑回归的理论全部讲完了。其实，除了引入一个逻辑函数，调整了假设函数和损失函数之外，逻辑回归的思路完全遵循线性回归算法。其中的重点在于把 y' 的值压缩到了［0，1］区间，并且最终以 0 或 1 的形式输出，形成二元分类。

“考一考你们，逻辑回归中用于计算损失的 y' 的值，是［0，1］区间的概率值，还是最终的分类结果 0、1 值呢？”

最终的分类结果。”？

用于计算损失的 y' 是逻辑函数给出的概率值 P，不是最终输出的分类结果 0 或 1。如果以 0 或 1 作为 y' 计算损失，那是完全无法求导的。因此，逻辑回归的假设函数 $h(x)$，也就是 y'，给出的是［0，1］区间的概率值，不是最终的 0、1 分类结果。这一点大家仔细思索清楚，如果还不明白的话可以课后找我单独讨论。同学们先休息一下，之后开始介绍二元分类的案例，二元分类是多元分类的基础。至于多元分类如何处理，等我们讲完二元分类再说。

通过逻辑回归解决二元分类问题

之前那个心脏健康状况调查问卷的案例就是一个二元分类问题—因为标签字段只有两种可能性：患病或者健康。

这个数据集其实是一个心脏病科研数据集。在 Kaggle 的 Datasets 页面中搜索关键词“heart”就可以找到它，也可以下载源码包中的文件并新建一个 Heart Dataset 数据集。

数据的准备与分析

这个数据集的收集工作完成得不错，尤其难能可贵的是所有的数据都已经数字化，减少了很多格式转换的工作。其中包含以下特征字段

- age：年龄。
- sex：性别（1 = 男性，0 = 女性）。
- cp：胸痛类型。
- trestbps：休息时血压。
- chol：胆固醇。
- fbs：血糖（1 = 超标，0 = 未超标）。
- restecg：心电图。
- thalach：最大心率。
- exang：运动后心绞痛（1 = 是，0 = 否）。
- oldpeak：运动后 ST 段压低。
- slope：运动高峰期 ST 段的斜率。
- ca：主动脉荧光造影染色数。
- thal：缺陷种类。

标签是 target 字段：0 代表无心脏病，1 代表有心脏病。

数据读取：

```python
import numpy as np # 导入NumPy数学工具箱
import pandas as pd # 导入Pandas数据处理工具箱
df_heart = pd.read_csv("./heart.csv")  # 读取文件
df_heart.head() # 显示前5行数据
```

前 5 行数据显示如下图所示

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

用 value_counts 方法输出数据集中患心脏病和没有患心脏病的人数：

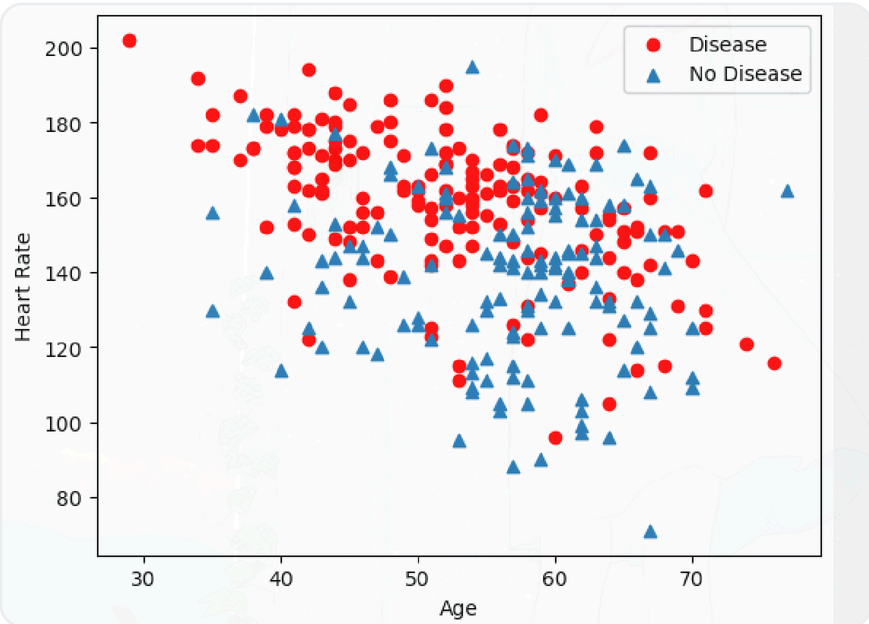
```
df_heart.target.value_counts() # 输出分类值，及各个类别数目
```

这个步骤是必要的。因为如果某一类别比例特别低（例如 300 个数据中只有 3 个人患病），那么这样的数据集直接通过逻辑回归的方法做分类可能是不适宜的。本例中患病和没有患病的人数比例接近。

还可以对某些数据进行相关性的分析，例如可以显示年龄 / 最大心率这两个特征与是否患病之间的关系(这里可以跑EDA)：

```
import matplotlib.pyplot as plt # 导入绘图工具
# 以年龄+最大心率作为输入，查看分类结果散点图
plt.scatter(x=df_heart.age[df_heart.target==1],
            y=df_heart.thalach[(df_heart.target==1)], c="red")
plt.scatter(x=df_heart.age[df_heart.target==0],
            y=df_heart.thalach[(df_heart.target==0)], marker='^')
plt.legend(["Disease", "No Disease"]) # 显示图例
plt.xlabel("Age") # X轴-Age
plt.ylabel("Heart Rate") # Y轴-Heart Rate
plt.show() # 显示散点图
```

输出结果如下图所示。



输出结果显示出心率（Heart Rate）越高，患心脏病的可能性看起来越大，因为代表患病样本的圆点，多集中在图的上方。

构建特征集和标签集

下面的代码构建特征张量和标签张量，并输出张量的形状

```
X = df_heart.drop(['target'], axis = 1) # 构建特征集
y = df_heart.target.values # 构建标签集
y = y.reshape(-1, 1) # -1 是相对索引 , 等价于 len(y)
print(" 张量 X 的形状 :", X.shape)
print(" 张量 X 的形状 :", y.shape)

 张量 X 的形状 : (303, 13)
 张量 X 的形状 : (303, 1)
```

拆分数据集

按照 80%/20% 的比例准备训练集和测试集：

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2)
```

数据准备部分的这几段代码大多数在上一课中已经出现过了

数据特征缩放

在第 3 课中，我们曾自定义了一个函数，进行数据的归一化。下面用 Sklearn 中内置的数据缩放器 MinMaxScaler，进行数据的归一化：

```
from sklearn.preprocessing import MinMaxScaler # 导入数据缩放器
scaler = MinMaxScaler() # 选择归一化数据缩放器，MinMaxScaler
X_train = scaler.fit_transform(X_train) # 特征归一化 训练集fit_transform
X_test = scaler.transform(X_test) # 特征归一化 测试集transform
```

这里有一个很值得注意的地方，就是对数据缩放器要进行两次调用。针对*X_train* 和*X_test*，要使用不同的方法，一个是 fit_transform（先拟合再应用），一个是 transform（直接应用）。这是因为，所有的最大值、最小值、均值、标准差等数据缩放的中间值，都要从训练集得来，然后同样的值应用到训练集和测试集。

本例中当然不需要对标签集进行归一化，因为标签集所有数据已经在［0，1］区间了。

NOTE

仅就这个数据集而言，MinMaxScaler 进行的数据特征缩放不仅不会提高效率，似乎还会令预测准确率下降。大家可以尝试一下使用和不使用 MinMaxScaler，观察其对机器学习模型预测结果所带来的影响。这个结果提示我们：没有绝对正确的理论，实践才是检验真理的唯一标准。

建立逻辑回归模型

数据准备工作结束后，下面构建逻辑回归模型。

1. 逻辑函数的定义

首先定义 Sigmoid 函数，一会儿会调用它：

```
# 首先定义一个Sigmoid函数，输入Z，返回y'
def sigmoid(z):
    y_hat = 1/(1+ np.exp(-z))
    return y_hat
```

这函数接收中间变量z（线性回归函数的输出结果），返回y'，即y_hat

2. 损失函数的定义

然后定义损失函数：

```
# 然后定义损失函数
def loss_function(X,y,w,b):
    y_hat = sigmoid(np.dot(X,w) + b) # Sigmoid逻辑函数 + 线性函数(wX+b)得到y'
    loss = -(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)) # 计算损失
    cost = np.sum(loss) / X.shape[0]  # 整个数据集平均损失
    return cost # 返回整个数据集平均损失
```

语句 y_hat = sigmoid（np.dot（X，w）+ b）中并没有把偏置当作*w0* 看待，因此，*X*特征集也就不需要在前面加一行 1。这里的线性回归函数是多变量的，因此（*X*，*w*）点积操作之后，用 Sigmoid 函数进行逻辑转换生成*y'*。y'* 生成过程中需要注意的仍然是点积操作中张量*X*和*W*的形状。

❶ X—(242,13),2D矩阵。

❷ W——(13,1),也是2D矩阵，因为第二阶为1,也可以看作向量，为了与X进行矩阵点积操作，把W直接构建成2D矩阵。

那么点积之后生成的y_hat,就是一个形状为(242,1)的张量，其中存储了每一个样本的预测值。之后的两个语句是损失函数的具体实现：

$$L(w,b)=-\frac{1}{N}\sum_{(x,y)\in D}\left[y^{*}\log(h(x))+(1-y)^{*}\log(1-h(x))\right]$$

语句 `loss = -((y*np.log(y_hat) + (1-y)*np.log(1-y_hat))` 计算了每一个样本的预测值*y*`到真值*y*的误差，其中用到了 Python 的广播功能，比如 1-*y*中的标量 1 就被广播为形状（242，1）的张量。

语句 cost = np.sum（loss）/ X.shape［0］是将所有样本的误差取平均值，其中 X.shape0］就是样本个数，cost，英文意思是成本，也就是数据集中各样本的平均损失。有了这个函数，无论是训练集还是测试集，输入任意一组参数 *w*、*b*，都会返回针对当前数据集的平均误差值（也叫损失或者成本）。这个值我们会一直监控它，直到它收敛到最小。

3. 梯度下降的实现

下面构建梯度下降的函数，这也是整个逻辑回归模型的核心代码。这个函数共 6 个输入参数，除了模型内部参数*w*、*b*，数据集*X*、*y*之外，还包含我们比较熟悉的两个超参数，学习速率 lr（learningrate，也就是 alpha）和迭代次数 iter：

```
# 然后构建梯度下降的函数
def gradient_descent(X,y,w,b,lr,iter) : #定义逻辑回归梯度下降函数
    l_history = np.zeros(iter) # 初始化记录梯度下降过程中误差值(损失)的数组
    w_history = np.zeros((iter,w.shape[0],w.shape[1])) # 初始化权重记录的数组
    b_history = np.zeros(iter) # 初始化记录梯度下降过程中偏置的数组
    for i in range(iter): #进行机器训练的迭代
        y_hat = sigmoid(np.dot(X,w) + b) #Sigmoid逻辑函数+线性函数(wX+b)得到y'
        derivative_w = np.dot(X.T,((y_hat-y)))/X.shape[0]  # 给权重向量求导
        derivative_b = np.sum(y_hat-y)/X.shape[0] # 给偏置求导
        w = w - lr * derivative_w # 更新权重向量，lr即学习速率alpha
        b = b - lr * derivative_b   # 更新偏置，lr即学习速率alpha
        l_history[i] =  loss_function(X,y,w,b) # 梯度下降过程中的损失
        print ("轮次", i+1 , "当前轮训练集损失：",l_history[i])
        w_history[i] = w # 梯度下降过程中权重的历史 请注意w_history和w的形状
        b_history[i] = b # 梯度下降过程中偏置的历史
    return l_history, w_history, b_history
```

这段代码在迭代过程中，求 *y_hat* 和损失的过程与损失函数中的部分代码相同。关键在于后面求权重*w*和偏置*b*的梯度（导数）部分，也就是下面公式的代码实现：

$$\text{梯度}=\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)}-h\left(x^{(i)}\right)\right)\cdot x^{(i)}$$

注意权重和偏置梯度的求法，之所以有差别，是因为偏置 *b*不需要与*x*特征项进行点积。（我们说过了，如果把偏置看作*w0*，就还需要加上一维值为 1 的*x0*，本例并没有这么做，而是分开处理。）还要注意权重的梯度是一个形状为（13，1）的张量，其维度和特征轴维度相同，而偏置的梯度则是一个值。求得导数之后，就通过学习速率对权重和偏置，分别进行更新，也就是用代码实现了下面的梯度下降公式。

$$w = w - \alpha \cdot \frac{\partial}{\partial w} L(w)$$

这样梯度下降基本上就完成了。之后返回的是梯度下降过程中每一次迭代的损失，以及权重和偏置的值，这些数据将帮助我们构建损失函数随迭代次数而变化的曲线。*w_history* 和 *b_history* 返回迭代过程中的历史记录。这里需要注意的是 *w_history* 是一个3D 张量，因为 *w* 已经是一个 2D 张量了，因此语句 w_history［i］= w，就是把权重赋值给 *w_history* 的后两个轴。而 *w_history* 的第一个轴则是迭代次数轴。张量阶数高的时候，数据操作的逻辑显得有点复杂，同学们在调试代码时可以不时地观察这些张量的 shape 属性，并输出其内容。

4． 分类预测的实现

梯度下降完成之后，就可以直接调用 gradient_descent 进行机器的训练，返回损失、最终的参数值：

```
# 用逻辑回归函数训练机器
loss_history, weight_history, bias_history =  \
            logistic_regression(X_train,y_train,weight,bias,alpha,iterations)
```

但是我们先不急着开始训练机器，先定义一个负责分类预测的函数

```
def predict(X,w,b): # 定义预测函数
    z = np.dot(X,w) + b # 线性函数
    y_hat = sigmoid(z) # 逻辑函数转换
    y_pred = np.zeros((y_hat.shape[0],1)) # 初始化预测结果变量
    for i in range(y_hat.shape[0]):
        if y_hat[i,0] < 0.5:
            y_pred[i,0] = 0 # 如果预测概率小于0.5，输出分类0
        else:
            y_pred[i,0] = 1 # 如果预测概率大于0.5，输出分类1
    return y_pred # 返回预测分类的结果
```

这个函数就通过预测概率阈值 0.5，把 *y_hat* 转换成 *y_pred*，也就是把一个概率值转换成 0 或 1 的分类值。*y_pred* 是一个和 *y* 标签集同样维度的向量，通过比较 *y_pred* 和真值，就可以看出多少个预测正确，多少个预测错误。

开始训练机器

首先把上面的所有内容封装成一个逻辑回归模型：

```
def logistic_regression(X,y,w,b,lr,iter): # 定义逻辑回归模型
    l_history,w_history,b_history = gradient_descent(X,y,w,b,lr,iter)#梯度下降
    print("训练最终损失:", l_history[-1]) # 打印最终损失
    y_pred = predict(X,w_history[-1],b_history[-1]) # 进行预测
    traning_acc = 100 - np.mean(np.abs(y_pred - y_train))*100 # 计算准确率
    print("逻辑回归训练准确率: {:.2f}%".format(traning_acc))  # 打印准确率
    return l_history, w_history, b_history # 返回训练历史记录
```

代码中的变量 traning_acc，计算出了分类的准确率。对于分类问题而言，准确率也就是正确预测数相对于全部样本数的比例，这是最基本的评估指标。等会儿咱们会调用这个函数，实现逻辑回归。训练机器之前，还要准备好参数的初始值：

```
#初始化参数
dimension = X.shape[1] # 这里的维度 len(X)是矩阵的行的数，维度是列的数目
weight = np.full((dimension,1),0.1) # 权重向量，向量一般是1D，但这里实际上创建了2D张量
bias = 0 # 偏置值
#初始化超参数
alpha = 1 # 学习速率
iterations = 500 # 迭代次数
```

下面调用逻辑回归模型，训练机器：

```
# 用逻辑回归函数训练机器
loss_history, weight_history, bias_history =  \
            logistic_regression(X_train,y_train,weight,bias,alpha,iterations)

轮次 500 当前轮训练集损失:  0.3475209120399868
训练最终损失: 0.3475209120399868
逻辑回归训练准确率: 86.36%
```

训练过程十分顺利，损失随着迭代次数的上升逐渐下降，最后呈现收敛状态。训练 500 轮之后的预测准确率为 86.36%。成绩不错！

测试分类结果

上面的 86.36% 只是在训练集上面形成的预测准确率，还并不能说明模型具有泛化能力，我们还需要在准备好的测试集中对这个模型进行真正的考验。

下面的代码用训练好的逻辑回归模型对测试集进行分类预测：

```
y_pred = predict(X_test,weight_history[-1],bias_history[-1]) # 预测测试集
testing_acc = 100 - np.mean(np.abs(y_pred - y_test))*100 # 计算准确率

print("逻辑回归测试准确率: {:.2f}%".format(testing_acc))
```

如果要亲眼看一看分类预测的具体值，可以调用刚才定义的 predict 函数把 y_pred 显示出来：

```
print ("逻辑回归预测分类值:",predict(X_test,weight_history[-1],bias_history[-1]))
```

绘制损失曲线

还可以绘制出针对训练集和测试集的损失曲线：

```
loss_history_test = np.zeros(iterations) # 初始化历史损失
for i in range(iterations): #求训练过程中不同参数带来的测试集损失
    loss_history_test[i] = loss_function(X_test,y_test,
                                         weight_history[i],bias_history[i])
index = np.arange(0,iterations,1)
plt.plot(index, loss_history,c='blue',linestyle='solid')
plt.plot(index, loss_history_test,c='red',linestyle='dashed')
plt.legend(["Training Loss", "Test Loss"])
plt.xlabel("Number of Iteration")
plt.ylabel("Cost")
plt.show() # 同时显示显示训练集和测试集损失曲线
```

可以明显地观察到，在迭代 80～100 次后，训练集的损失进一步下降，越来越小，但是测试集的损失并没有跟着下降，反而显示呈上升趋势（如下图所示）。这是明显的过拟合现象。因此迭代应该在 100 次之前结束。

因此，损失曲线告诉我们，对于这个案例，最佳迭代次数是 80～100 次，才能够让训练集和测试集都达到比较好的预测效果。这是模型在训练集上面优化，在测试集上泛化的一个折中方案。

Sklearn中的实现

这里我们可以直接使用sklearn实现

```
from sklearn.linear_model import LogisticRegression # 导入逻辑回归模型
lr = LogisticRegression() # lr, 就代表是逻辑回归模型
lr.fit(X_train, y_train) # fit, 就相当于是梯度下降
print("SK learn 逻辑回归测试准确率 {:.2f}%".format(lr.score(X_test, y_test)*100))
```

同学们看到输出结果如下。

```
SK-learn逻辑回归测试准确率81.97%
```

“这就是 Sklearn 库函数的厉害之处，里面封装了很多逻辑。这节省了很大的工作量。同学们请注意，这里的 fit 方法就等价于我们在前面花了大力气编写的梯度下降代码。”

“为什么这个 Sklearn 的测试准确率比咱们的 81.97% 高这么多，是我们的算法哪里出问题了？”

大概有两点是可以优化的，你们猜猜呢？”

“一个可能是你刚才说的过拟合的问题，把迭代次数从 500 调整到 100 以内可能会好一点。”

根据上面的损失函数图像，过拟合的确是目前的一个问题，可以考虑减少迭代次数。而另外一个影响效率的原因是这个数据集里面的某些数据格

式其实不对，需要做一点小小的特征工程。

哑变量的使用:

你们可能注意到数据集中的性别数据是 0、1 两种格式。我们提到过，如果原始数据是男、女这种字符，首先要转换成 0、1 数据格式。那么你们再观察像 'cp'、'thal' 和 'slope' 这样的数据，它们也都代表类别。比如，cp 这个字段，它的意义是“胸痛类型”，取值为 0、1、2、3。这些分类值，是大小无关的。

但是问题在于，计算机会把它们理解为数值，认为 3 比 2 大，2 比 1 大。这种把“胸痛类型”的类别像“胸部大小”的尺码一样去解读是不科学的，会导致误判。因为这种类别值只是一个代号，它的意义和年龄、身高这种连续数值的意义不同。解决的方法，是把这种类别特征拆分成多个哑特征，比如 cp 有 0、1、2、3 这 4 类，就拆分成个 4 特征，cp_0 为一个特征、cp_1 为一个特征、cp_2 为一个特征、cp_3 为一个特征。每一个特征都还原成二元分类，答案是 Yes 或者 No，也就是数值 1 或 0。

这个过程是把一个变量转换成多个哑变量（dummy variable），也叫虚拟变量、名义变量的过程。哑变量用以反映质的属性的一个人工变量，是量化了的质变量，通常取值为 0 或 1。

```python
# 把 3 个文本型变量转换为哑变量
a = pd.get_dummies(df_heart['cp'], prefix = "cp")
b = pd.get_dummies(df_heart['thal'], prefix = "thal")
c = pd.get_dummies(df_heart['slope'], prefix = "slope")
# 把哑变量添加进 dataframe
frames = [df_heart, a, b, c]
df_heart = pd.concat(frames, axis = 1)
df_heart = df_heart.drop(columns = ['cp', 'thal', 'slope'])
df_heart.head() # 显示新的 dataframe
```

	age	sex	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	ca	...	cp_1	cp_2	cp_3	thal_0	thal_1
0	63	1	145	233	1	0	150	0	2.3	0	...	False	False	True	False	True
1	37	1	130	250	0	1	187	0	3.5	0	...	False	True	False	False	False
2	41	0	130	204	0	0	172	0	1.4	0	...	True	False	False	False	False
3	56	1	120	236	0	1	178	0	0.8	0	...	True	False	False	False	False
4	57	0	120	354	0	1	163	1	0.6	0	...	False	False	False	False	False

```
SK-learn逻辑回归测试准确率85.25%
```

原本的 'cp'、'thal' 和 'slope' 变成了哑变量 'cp_0'、'cp_1'、'cp_2'，等等，而且取值全部是0 或者 1。这样计算机就不会把类别误当大小相关的值处理。把这个小小的特征工程做好之后，特征的数目虽然增多了，不过重新运行咱们自己做的逻辑回归模型，会发现模型的效率将有显著的提升。