1 一、逻辑回归的基础介绍

逻辑回归是一个**分类模型**

它可以用来预测某件事发生是否能够发生。分类问题是生活中最常见的问题：

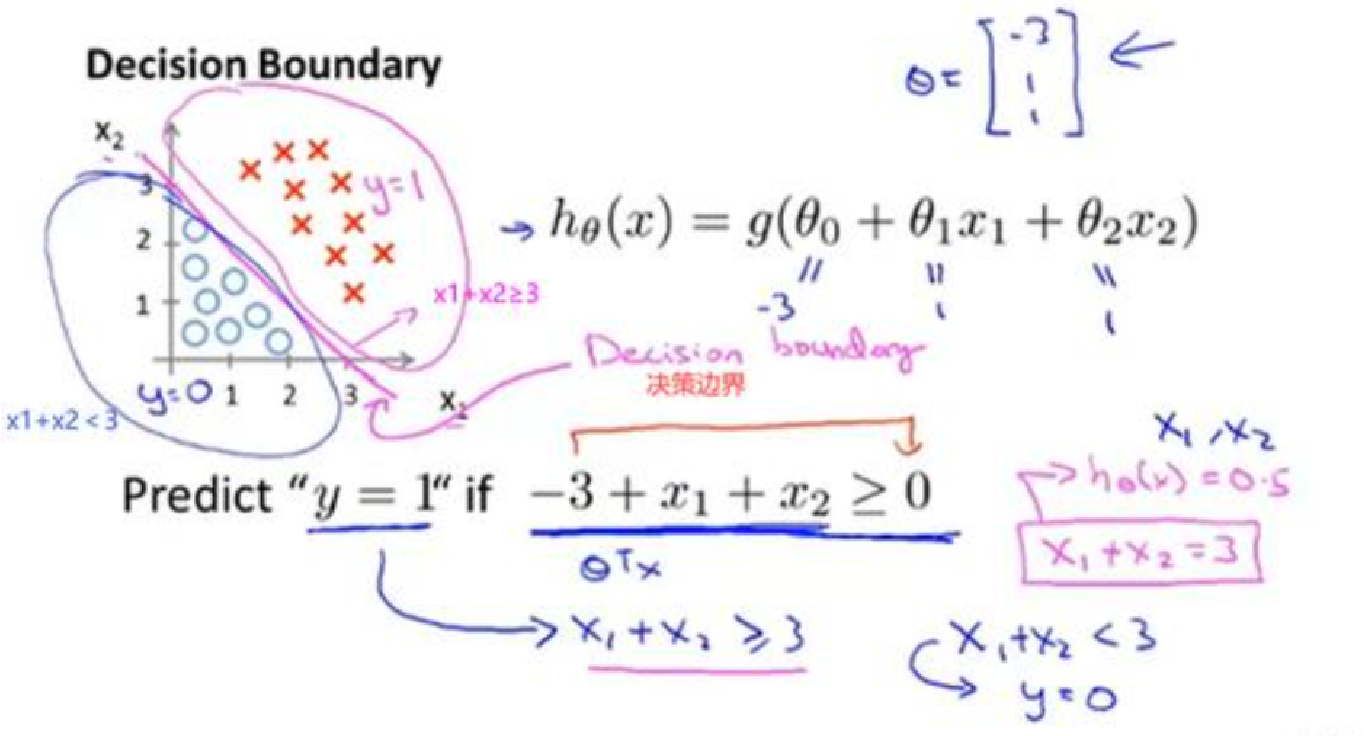
生活中：比如预测上证指数明天是否会上涨，明天某个地区是否会下雨，西瓜是否熟了

金融领域：某个交易是否涉嫌违规，某个企业是否目前是否违规，在未来一段时间内是否会有违规

互联网：用户是否会购买某件商品，是否会点击某个内容

对于已知的结果，上面问题的回答只有：$0, 1$ 。

我们以以下的一个二分类为例，对于一个给定的数据集，存在一条直线可以将整个数据集分为两个部分：



此时，决策边界为$w_1x_1 + w_2x_2 + b = 0$，此时我们很容易将

$$h(x) = w_1x_1 + w_2x_2 + b > 0$$

的样本设置为1，反之设置为0。但是这其实是一个感知机的决策过程。逻辑回归在此基础上还需要在加上一层，**找到分类概率与输入变量之间的关系，通过概率来判断类别。**

1.1 线性回归模型：

$$h(x) = w^T x + b$$

In [4]:

```
w=[0.1,0.2,0.4,0.2]
b=0.5
def linearRegression(x):
    return sum([x[i]*w[i] for i in range(len(x))])+b
linearRegression([2,1,3,1])
```

Out[4]:

```
2.3
```

1.2 logistic函数

在线性模型的基础上加上一个函数g，即$h(x) = g(w^T x + b)$。$g(z) = 1/(1 + e^{-z})$。这个函数就是sigmoid函数，也叫做logistic函数。 它可以将一个线性回归中的结果转化为一个概率值。此时$h(x)$表示的就是某件事发生的概率，我们也可以记为$p(Y = 1|x)$

In [5]:

```
import numpy as np
def sigmoid(x):
    return 1/(1+np.exp(-x))
sigmoid(linearRegression([2,1,3,1]))
```

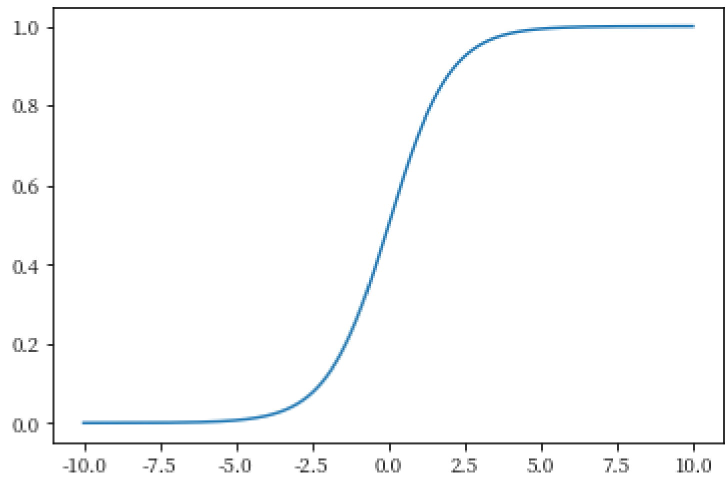
Out[5]:

```
0.9088770389851438
```

可以看一下sigmoid函数的图：

In [81]:

```
import matplotlib.pyplot as plt
x = np.arange(-10, 10, 0.01)
y = sigmoid(x)
plt.plot(x, y)
plt.show()
```



通过以上内容我们知道逻辑回归的表达式，那么我们怎么进行优化呢？x是我们输入的参数，对于我们是已知的，预测西瓜是否熟了的话，我们需要知道它的大小，颜色等信息。将其输入到预测模型中，返回一个西瓜是否熟了的概率。**那么对于怎么得到模型中的参数w和b呢？**

2 二、逻辑回归的优化方法

2.1 1：逻辑回归的损失函数

逻辑回归采用的是交叉熵的损失函数，对于一般的二分类的逻辑回归来说交叉熵函数为：
$J(\theta) = -[yln(y') + (1-y)ln(1-y')]$,其中$y'$是预测值。

注：有些地方交叉熵的log的底数是2，有些地方是e。由于$\frac{log_2(x)}{log_e(x)} = log_2(e)$是一个常数，因此无论是啥对于最后的结果都是不影响的，不过由于计算的简便性用e的会比较多一些。

实际上我们求的是训练中所有样本的损失，因此：

$$J(\theta) = -\frac{1}{m}\sum[y_i ln(y_i') + (1-y_i)ln(1-y_i')]$$

注：θ代表的是所有的参数集合

Q:为什么不采用最小二乘法进行优化？(平方差)

A:因为采用最小二乘法的话损失函数就是非凸了(凸函数的定义是在整个定义域内只有一个极值，极大或者极小，该极值就是全部的最大或者最小) 更多详细地解释可以看这里：[https://www.zhihu.com/question/65350200 (https://www.zhihu.com/question/65350200)](https://www.zhihu.com/question/65350200)

后面可能会有不少难以解释或者需要花费很大篇幅去解释的地方，大多数比较延展性的知识，我可能还是会放个链接，有需要的朋友可以当作课外拓展去了解下。

2.1.1 注：损失函数的由来

在统计学中，假设我们已经有了一组样本(X,Y)，为了计算出能够产生这组样本的参数。通常我们会采用最大似然估计的方法(一种常用的参数估计的方法)。使用到最大似然估计的话，我们还要一个假设估计，这里我们就是假设Y**是服从于伯努利分布的。**

$$P(Y=1|x) = p(x)$$

$$P(Y=0|x) = 1-p(x)$$

由于Y服从于伯努利分布，我们很容易就有似然函数：

$$L = \prod[p(x_i)^{y_i}][1-p(x_i)]^{(1-y_i)}$$

为了求解的方便我们可以两边取对数：

$$ln(L) = \sum[y_i ln(p(x_i)) + (1 - y_i) ln(1 - p(x_i))]$$

大家其实也发现了L和J两个式子的关系，其实对L求最大就相当于对J求最小。这个也是大家以后会听到的**最大似然函数和最小损失函数之间**的关系。如果被问道逻辑回归为什么要采用交叉熵作为损失函数也可以回答：**这是假设逻辑回归中Y服从于伯努利分布，然后从最大似然估计中推导来的。**

2.2 2：梯度下降法

逻辑回归的优化方法是梯度下降法。 从数学的角度来说，函数梯度的方向就是函数增长最快的方向，反之梯度的反方向就是函数减少最快的方向。因此我们想要计算一个函数的最小值，就朝着该函数梯度相反的方向前进。 我们先简单地介绍一下该算法。 假设我们需要优化的函数：$f(X) = f(x_1, \ldots, x_n)$

首先我们初始化自变量，从$X^(0) = (x_1^{(0)}, \ldots x_n^{(0)})$开始。设置一个学习率$\eta$。 对于任何$i >= 0$:

如果是最小化f

$x_1^{i+1} = x_1^i - \eta \frac{\partial f}{\partial x 1}(x^{(i)})$

$x_n^{i+1} = x_n^i - \eta \frac{\partial f}{\partial x_n}(x^{(i)}),$

反之如果求f的最大值，则

$x_1^{i+1} = x_1^i + \eta \frac{\partial f}{\partial x 1}(x^{(i)}),$

$x_n^{i+1} = x_n^i + \eta \frac{\partial f}{\partial x_n}(x^{(i)}),$

注： 很多人可能是听说过**随机梯度下降，批次梯度下降，梯度下降法**。这三者的区别非常简单，就是看样本数据，随机梯度下降每次计算一个样本的损失， 然后更新参数θ，批次梯度下降是每次根据一批次的样本来更新参数，梯度下降是全部样本。一般来说随机梯度下降速度快(每计算一个样本就可以更新一次参数，参数更新的速度极快)。同样的，随机梯度下降的收敛性会比较差，容易陷入局部最优。反过来每次用来参数更新的样本越多，速度会越慢，但是更能够达到全局最优。

2.3 3：逻辑回归的优化

以上是逻辑回归优化的目标函数$J(w, b) = -\frac{1}{m}\sum[y_i ln(\sigma(w^T x + b)) + (1 - y_i) ln(1 - \sigma(w^T x + b))]$

我们需要优化参数w, b，从而使其在我们已知的样本X, y上值最小。也就是我们常说的经验风险最小。

既然要优化目标函数，那么首先我们需要对$J(w, b)$求导。

我们先令 $g = \sigma(w^T x + b)$

$$\frac{\partial J(g)}{\partial g} = -\frac{\partial}{\partial g}[y ln(g) + (1 - y) ln(1 - g)] = -\frac{y}{g} + \frac{1 - y}{1 - g}$$

再令：$a = w^T x + b$

$$\frac{\partial g}{\partial a} = \frac{\partial(\frac{1}{1+e^{-a}})}{\partial a} = -(1 + e^{-a})^{-2} - e^{-a} = \frac{1}{1 + e^{-a}} \frac{1 + e^{-a} - 1}{1 + e^{-a}} = \sigma(a)(1 - \sigma(a)) = g(1 - g)$$

可以发现$g = \sigma(a)$，但是g对a求导之后居然是 $g(1-g)$，这也是Sigmoid函数特别有意思的一点，在后续的梯度下降优化中，这个性质可以帮助我们减少很多不必要的计算。

有了上面的基础，我们就可以求我们需要优化的参数w, b的梯度了。根据链式求导：

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial g}\frac{\partial g}{\partial a}\frac{\partial a}{\partial w} = (-\frac{y}{g} + \frac{1-y}{1-g})g(1-g)x = (g-y)x$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial g}\frac{\partial g}{\partial a}\frac{\partial a}{\partial b} = (-\frac{y}{g} + \frac{1-y}{1-g})g(1-g) = (g-y)$$

以上就是关于逻辑回归优化的介绍。根据上述的公式，我们简单地写一个根据随机梯度下降法进行优化的函数。

In [72]:

```python
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
X=datasets.load_iris()['data']
Y=datasets.load_iris()['target']
Y[Y>1]=1
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.4,stratify=Y)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def cal_grad(y, t):
    grad = np.sum(t - y) / t.shape[0]
    return grad

def cal_cross_loss(y, t):
    loss=np.sum(-y * np.log(t)- (1 - y) * np.log(1 - t))/t.shape[0]
    return loss

class LR:
    def __init__(self, in_num, lr, iters, train_x, train_y, test_x, test_y):
        self.w = np.random.rand(in_num)
        self.b = np.random.rand(1)
        self.lr = lr
        self.iters = iters
        self.x = train_x
        self.y = train_y
        self.test_x=test_x
        self.test_y=test_y


    def forward(self, x):
        self.a = np.dot(x, self.w) + self.b
        self.g = sigmoid(self.a)
        return self.g

    def backward(self, x, grad):
        w = grad * x
        b = grad
        self.w = self.w - self.lr * w
        self.b = self.b - self.lr * b

    def valid_loss(self):
        pred = sigmoid(np.dot(self.test_x, self.w) + self.b)
        return cal_cross_loss(self.test_y, pred)

    def train_loss(self):
        pred = sigmoid(np.dot(self.x, self.w) + self.b)
        return cal_cross_loss(self.y, pred)

    def train(self):
        for iter in range(self.iters):
            ##这里我采用随机梯度下降的方法

            for i in range(self.x.shape[0]):
                t = self.forward(self.x[i])
                grad = cal_grad(self.y[i], t)
                self.backward(self.x[i], grad)

```

```
                train_loss = self.train_loss()
                valid_loss = self.valid_loss()
                if iter%5==0:
                    print("当前迭代次数为: ", iter, "训练loss:", train_loss, "验证loss:'
model=LR(4,0.01,100,X_train,y_train,X_test,y_test)
model.train()
```

```
当前迭代次数为:  0 训练loss: 0.4291929345464963 验证loss: 0.42693038550585
577
当前迭代次数为:  5 训练loss: 0.11277882862499525 验证loss: 0.1055056827486
348
当前迭代次数为:  10 训练loss: 0.06494436815119954 验证loss: 0.058823992714
00256
当前迭代次数为:  15 训练loss: 0.04581966372516797 验证loss: 0.040633498852
263875
当前迭代次数为:  20 训练loss: 0.03553257648646332 4 验证loss: 0.03106087519
1807488
当前迭代次数为:  25 训练loss: 0.029098402617809858 验证loss: 0.02518038632
6062225
当前迭代次数为:  30 训练loss: 0.024686442218866525 验证loss: 0.02120735616
9834532
当前迭代次数为:  35 训练loss: 0.021467975425719914 验证loss: 0.01834458891
9580344
当前迭代次数为:  40 训练loss: 0.019013391664316245 验证loss: 0.01618386873
5692258
当前迭代次数为:  45 训练loss: 0.017077651833693537 验证loss: 0.01449494166
4880414
当前迭代次数为:  50 训练loss: 0.015510694661811146 验证loss: 0.01313821359
1260125
当前迭代次数为:  55 训练loss: 0.014215426919527437 验证loss: 0.01202418238
3760034
当前迭代次数为:  60 训练loss: 0.01312621142455474 验证loss: 0.011092840272
228175
当前迭代次数为:  65 训练loss: 0.012197059676620554 验证loss: 0.01030245804
8003647
当前迭代次数为:  70 训练loss: 0.011394776690578588 验证loss: 0.00962312158
7215629
当前迭代次数为:  75 训练loss: 0.010694791869480873 验证loss: 0.00903282811
8843856
当前迭代次数为:  80 训练loss: 0.010078522514523285 验证loss: 0.00851503351
246566
当前迭代次数为:  85 训练loss: 0.009531650098820144 验证loss: 0.00805705839
6406335
当前迭代次数为:  90 训练loss: 0.009042960436174407 验证loss: 0.00764902176
8855624
当前迭代次数为:  95 训练loss: 0.008603543458349608 验证loss: 0.00728310915
907939
```

以上是手写的逻辑回归，用于自己练习就可以啦，通常我们还是会调sklearn包的。

In [11]:

```python
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
X=datasets.load_iris()['data']
Y=datasets.load_iris()['target']
from sklearn.linear_model import LogisticRegression
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.1,stratify=Y)


model=LogisticRegression(penalty='l2',
                         class_weight=None,
                        random_state=None,  max_iter=100)
model.fit(X_train,y_train)
model.predict_proba(X_test)
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/logistic.p
y:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.2
2. Specify a solver to silence this warning.
  FutureWarning)
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/logistic.p
y:469: FutureWarning: Default multi_class will be changed to 'auto' in
0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

Out[11]:

```
array([[1.21688590e-02, 6.69378477e-01, 3.18452664e-01],
       [3.18783065e-02, 7.46131895e-01, 2.21989799e-01],
       [3.43828202e-04, 2.37078790e-01, 7.62577382e-01],
       [5.29090689e-04, 4.13567497e-01, 5.85903412e-01],
       [3.29337290e-04, 2.81221686e-01, 7.18448977e-01],
       [9.00741098e-01, 9.92046734e-02, 5.42283370e-05],
       [8.05343921e-01, 1.94609817e-01, 4.62621928e-05],
       [9.67282864e-02, 7.74831535e-01, 1.28440178e-01],
       [7.93043185e-01, 2.06919042e-01, 3.77732243e-05],
       [4.12743806e-04, 4.59051241e-01, 5.40536016e-01],
       [2.24448595e-03, 5.20988907e-01, 4.76766607e-01],
       [5.15225784e-02, 8.20568416e-01, 1.27909005e-01],
       [1.89302090e-03, 3.07587504e-01, 6.90519475e-01],
       [8.88574049e-01, 1.11415095e-01, 1.08564499e-05],
       [9.19195938e-01, 8.07856941e-02, 1.83682014e-05]])
```

penalty:惩罚系数，也就是我们常说的正则化，默认为"l2",也可以使用"l1"，之后我们会介绍逻辑回归的l1,l2正则化。

class_weight:类别权重，一般我们在分类不均衡的时候使用，比如{0:0.1,1:1}代表在计算loss的时候，0类别的loss乘以0.1。这样在0类别的数据过多时候就相当于给1类别提权了。就我个人而言，比起在类别不均衡时采用采用，我更倾向于这个。

max_iter：最大迭代次数。

以上就是逻辑回归的整体介绍，相信大家看完之后对逻辑回归也有一个比较全面的了解。但是逻辑回归是一个很奇怪的算法，表面上看起来比较简单，但是深挖起来知识特别多，千万不要说自己精通逻辑回归。

3 三、逻辑回归的拓展

3.1 1: 逻辑回归的正则化

首先我们介绍一下正则化，对这个方面比较的可以跳过。一般我们模型就是训练就是为了最小化经验风险，**正则化就是在这个基础上加上约束(也可以说是引入先验知识)**，这种约束可以引导优化误差函数的时候倾向于选择向满足约束的梯度下降的方向。

注：这里我们可以补充一下经验风险，期望风险和结构化风险。 经验风险就是训练集中的平均损失，期望风险就是(X，y)联合分布的期望损失，当样本数量N趋向于无穷大时，经验风险也趋向于期望风向。机器学习做的就是通过经验风险取估计期望风险。 结构化风险是防止防止过拟合在经验风险的基础上加上正则项。

在逻辑回归中常见的有L1正则化和L2正则化。

加上参数w绝对值的和

$$L1: J(\theta) = -\frac{1}{m}\sum[y_i ln(y_i`) + (1-y_i)ln(1-y_i`)] + |w|$$

加上参数w平方和

$$L2: J(\theta) = -\frac{1}{m}\sum[y_i ln(y_i`) + (1-y_i)ln(1-y_i`)] + ||w^2||$$

以$L2$为例，我们的优化的目标函数不再是仅仅经验风险了，我们还要在$||w^2||$最小的基础上达到最小的经验风险。用我们生活的例子就是在最小代价的基础上达成一件事，这样肯定最合理的。

如果只是到这里感觉就很容易了，但是一般别人都会问你：

1： $L1, L2$正则化有什么理论基础?

2：为什么$L1$正则化容易产生离散值?

吐槽：就是加上一个$|w|$和$||w^2||$啊，让w的取值小一点，有啥可说的。

吐槽归吐槽，问题还是要回答的。

当我们假设参数w服从于正态分布的时候，根据贝叶斯模型可以推导出$L2$正则化，当我们假设参数w服从于拉普拉斯分布的时候，根据贝叶斯模型可以推导出$L1$正则化

具体的推导如下：

逻辑回归其实是假设参数是确定，我们需要来求解参数。贝叶斯假设逻辑回归的参数是服从某种分布的。假设参数w的概率模型为$p(w)$。使用贝叶斯推理的话：

$$p(w|D) \prec p(w)p(D|w) = \Pi_{j=1}^{M} p(w_j)\Pi_{i=1}^{N} p(D_i|w_j)$$

对上式取log

$$=> argmax_w[\Sigma_{j=1}^{M} logp(w_j) + \Sigma_{i=1}^{N} logp(D_i|w)]$$
$$=> argmax_w[\Sigma_{i=1}^{N} logp(D_i|w) + \Sigma_{j=1}^{M} logp(w_j)]$$
$$=> argmin_w(-\frac{1}{N}\Sigma_{i=1}^{N} logp(D_i|w) - \frac{1}{N}\Sigma_{j=1}^{M} logp(w_j))$$

前面的式子大家很熟悉，就是逻辑回归的损失函数。现在假设$p(w)$服从某个分布，相当于引入一个先验的知识。如果$p(w)$服从均值为0拉普拉斯分布：

$$p(w) = \frac{1}{2\lambda}e^{\frac{-|w|}{\lambda}}$$

将$p(w)$代入上面的式子可以得到：

$$\Sigma_{j=1}^{M} logp(w_j) => \Sigma_{j=1}^{M}(-\frac{|w_i|}{\lambda}log(\frac{1}{2\lambda}))$$

将参数$\frac{1}{\lambda}log(\frac{1}{2\lambda})$用一个新的参数$\lambda$代替，然后就可以得到我们正则化L1的正则化项：

$$\lambda\Sigma_{j=1}^{M}|w_i|$$

两部分加起来就是逻辑回归L1正则化下的损失函数了。

L2正则化也是同样的道理，只不过是假设$p(w)$服从均值为0的正态分布。

关于$L1,L2$等高线图大家肯定也都接触过了。 从模型优化的角度上来说：

当w大于0时$|w|$的导数为1，根据梯度下降，w在更新完逻辑回归的那个参数之后，需要减去$lr * 1$。这样会将w往0出趋近， 反之当w小于0时，根据梯度下降|w|的梯度也会让w往0的方向趋近。直到w为$|w|$的梯度也为0了。

$L2$并不会出现以上情况，假设我们把一个特征复制1次。复制之前该特征的权重为w1,复制之后使用L1正则化，倾向于将另外一个特征的权重优化为0，而L2正则化倾向于将两个特征的权重都优化为w1/2,因为我们很明显的知道，当两个权重都为w1/2时。$||w^2||$才会最小。

以上只是我个人的见解，关于这个问题，我在网上也看到了大神们的各种解释，可以参照着看看:[https://zhuanlan.zhihu.com/p/50142573 (https://zhuanlan.zhihu.com/p/50142573)](https://zhuanlan.zhihu.com/p/50142573)

3.2 2: 为什么逻辑回归中经常会将特征离散化。

这个是工业界中常见的操作，一般我们不会将连续的值作为特征输入到逻辑回归的模型之中，而是将其离散成0，1变量。这样的好处有：

1：稀疏变量的内积乘法速度快，计算结果方便存储，并且容易扩展；

2：离散化后的特征对异常数据有很强的鲁棒性：比如一个特征是年龄>30是1，否则0。如果特征没有离散化，一个异常数据“年龄300岁”会给模型造成很大的干扰。

3：逻辑回归属于广义线性模型，表达能力受限；单变量离散化为N个后，每个变量有单独的权重，相当于为模型引入了非线性，能够提升模型表达能力，加大拟合；

4：离散化后可以进行特征交叉，由M+N个变量变为M*N个变量，进一步引入非线性，提升表达能力；

5：特征离散化后，模型会更稳定，比如如果对用户年龄离散化，20-30作为一个区间，不会因为一个用户年龄长了一岁就变成一个完全不同的人。当然处于区间相邻处的样本会刚好相反，所以怎么划分区间是门学问。

4 四、作业

4.1 STEP1: 按照要求计算下方题目结果

作业1：逻辑回归的表达式：

A: h(x)=wx+b

B：h(x)=wx

C: h(x)=sigmoid(wx+b)

D: h(x)=sigmoid(wx)

In [57]:

```
a1="C"
```

作业2：下面关于逻辑回归的表述是正确的(多选)：

A:逻辑回归的输出结果是概率值，在0-1之间

B:使用正则化可以提高模型的泛化性

C:逻辑回归可以直接用于多分类

D:逻辑回归是无参模型

E:逻辑回归的损失函数是交叉熵

In [58]:

```
a2="ABE"
```

作业3：计算$y = sigmoid(w1 * x1 + w2 * x2 + 1)$当 w=(0.2, 0.3)时，样本X=(1,1),y=1的时w1,w2的梯度和loss：(保存3位小数，四舍五入)

In [74]:

```
w = np.array([0.2,0.3])
x_arr = np.array([1,1])
b = 1
y = np.array([1])
linear_result = np.dot(w,x) +b
predict = np.array([sigmoid(linear_result)])
grad = cal_grad(y_arr,predict)
grad = grad * x_arr
a5 = round(cal_cross_loss(y,predict),3)
a3=round(grad[0],3)#(w1)
a4=round(grad[1],3)
```

In [67]:

```
np.dot(w,x_arr)
```

Out[67]:

```
0.5
```

In [68]:

```
print(a5)
```

```
0.201
```

作业4：在cal_grad梯度函数的基础上加上L2正则化，下面的函数是否正确?(Y/N)

In [62]:

```
def cal_grad(y, t,x,w):
    """
    x:输入x
    y:样本y
    t:预测t
    w:参数w
    """
    grad = np.sum(t - y) / t.shape[0]
    return grad*x+2*w
```

In [63]:

```
a6="Y"
```

4.2 STEP2: 将结果保存为 csv 文件

csv 需要有两列，列名：id、answer。其中，id列为题号，从a1开始到a6来表示。answer 列为各题你得出的答案选项。

In [75]:

```
import pandas as pd # 这里使用下pandas, 来创建数据框
answer=[a1,a2,a3,a4,a5,a6]

# answer=[x.upper() for x in answer]
dic={"id":['a'+str(i+1) for i in range(6)],"answer":answer}
df=pd.DataFrame(dic)
df.to_csv('answer1.csv',index=False, encoding='utf-8-sig')
df
```

Out[75]:

	id	answer
0	a1	C
1	a2	ABE
2	a3	-0.182
3	a4	-0.182
4	a5	0.201
5	a6	Y

4.3 STEP3: 提交 csv 文件，获取分数结果

现在你的答案文件已经准备完毕了，怎么提交得到评分呢?

1、拷贝提交 token

去对应关卡的[提交页面 (https://www.heywhale.com/home/activity/detail/62a07f19ac8fed662502782f/submit)](https://www.heywhale.com/home/activity/detail/62a07f19ac8fed662502782f/submit)，（每个关卡的 token 不一样！）找到对应提交窗口，看到了你的 token 嘛?

拷贝到下方 cell 里（替换掉 XXXXXXX）。

2、找到你的答案文件路径

左侧文件树，在 project 下找到 csv 答案文件，右键点击可复制路径。 本关的答案路径👇

/home/mw/project/answer1.csv

In [76]:

```
# 运行这个 cell 前记得一定要保证右上角 kernel为 Python 3 的噢
# 下载提交工具
!wget -nv -O heywhale_submit https://cdn.kesci.com/submit_tool/v4/heywhale_submi

# 运行提交工具
# 把下方 XXXXXXX 替换为你的 Token, submit_file 为要提交的文件名路径
# 文件名路径去左侧文件树下, 刷新, 找到对应的 csv 文件, 右键复制路径
!./heywhale_submit -token ab7e7554b08e5b1f -file /home/mw/project/answer1.csv
```

```
wget: /opt/conda/lib/libcrypto.so.1.0.0: no version information availa
ble (required by wget)
wget: /opt/conda/lib/libssl.so.1.0.0: no version information available
(required by wget)
wget: /opt/conda/lib/libssl.so.1.0.0: no version information available
(required by wget)
2022-10-15 09:02:51 URL:https://cdn.kesci.com/submit_tool/v4/heywhale_
submit [10675472/10675472] -> "heywhale_submit" [1]
Heywhale Submit Tool 4.0.1

> 已验证Token
> 提交文件 /home/mw/project/answer1.csv (0.06 KiB), Target Qiniu
> 已上传 100 %
> 文件已上传
> 服务器响应: 200 提交成功，请等待评审完成
> 提交完成
```

😃 运行成功、显示提交完成后，即可去[提交页面 (https://www.heywhale.com/home/activity/detail/62a07f19ac8fed662502782f/submit)](https://www.heywhale.com/home/activity/detail/62a07f19ac8fed662502782f/submit)看成绩。满分即可进入下一关。