逻辑斯特回归

分类模型，常用作二分类

结果常通过概率来表示，哪一个类别概率大就判断为哪一个分类。我们首先熟悉概率的基本知识以及对极大似然估计得到对参数的最佳估计值。

一、概率知识

（1）定义：概率：对一个事情可能性的衡量

（2）范围：0<=P<=1

（3）计算方法：

公式法（最常用的）：

$P(\text{随机事件})=\frac{\text{随机事件可能出现的结果数}}{\text{随机事件所有可能出现的结果数}}$.其中 P(必然事件)=1, P（不可能事件）=0；0<P(随机事件)<1.

例：图 1 中每一个标有数字的方块均是可以翻动的木牌，其中只有两块木牌的背面贴有中奖标志，则随机翻动一块木牌中奖的概率为_____.

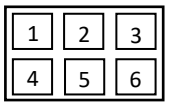
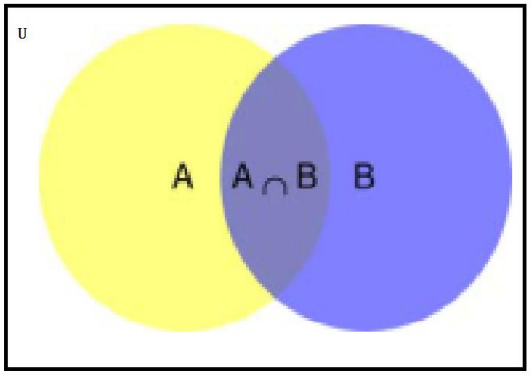


图 1

解析: 本题考查用公式法求概率,在随机翻动木牌过程中，一共有 6 种可能的翻牌结果,其中有 2 种为中奖,所以 $P(\text{中奖})=\frac{2}{6}=\frac{1}{3}$.

（4）条件概率



问题：在B事件发生的情况下A事件发生的概率是多少？（条件概率）

*** P（A|B）=P(A n B)/P（B）

$$P(A|B)=\frac{P(A\cap B)}{P(B)}$$

二、极大似然估计

极大似然估计是概率的 log 概率的最大化问题，即 max(log（P）)，log 函数是单调函数，通常取底数大于 1 的情况，所以 log 函数不影响原来概率大小的判断。如果取极大似然函数的相反数，得到的就是我们熟悉的损失函数 cost，同时极大似然函数也对应的极小化我们的损失函数。

1.为什么要有极大似然估计？

例子：我与一位猎人一起外出打猎，一只野兔从前方穿过，只听到一声枪响，野兔应声倒下。问是谁打中的呢？

答：极有可能是猎人。

显然候选人就两个，我和猎人。若选择我，则事件发生的发生概率为 0.01%，因为我不会打猎；若选择猎人，则事件发生的概率为 99%，而事件已经发生，因此选择猎人更为合适。

极大似然法的基本思想在社会思维意识中常有所体现。例如某地发生了一个疑难案件，警察欲破案或民众推测嫌疑人，一般是将重点集中在作案可能性较大的可疑人身上。

极大似然估计的思想

设总体中含有待估参数 w，可以取很多值。已经知道了样本观测值（例子中的兔子被猎人打死了），从 w 的一切可能值中（引例中是我和猎人）选出一个使该观察值出现的概率为最大的值，作为 w 参数的估计值，这就是极大似然估计。（顾名思义：就是看上去那个是最大可能的意思）

2.极大似然估计步骤：

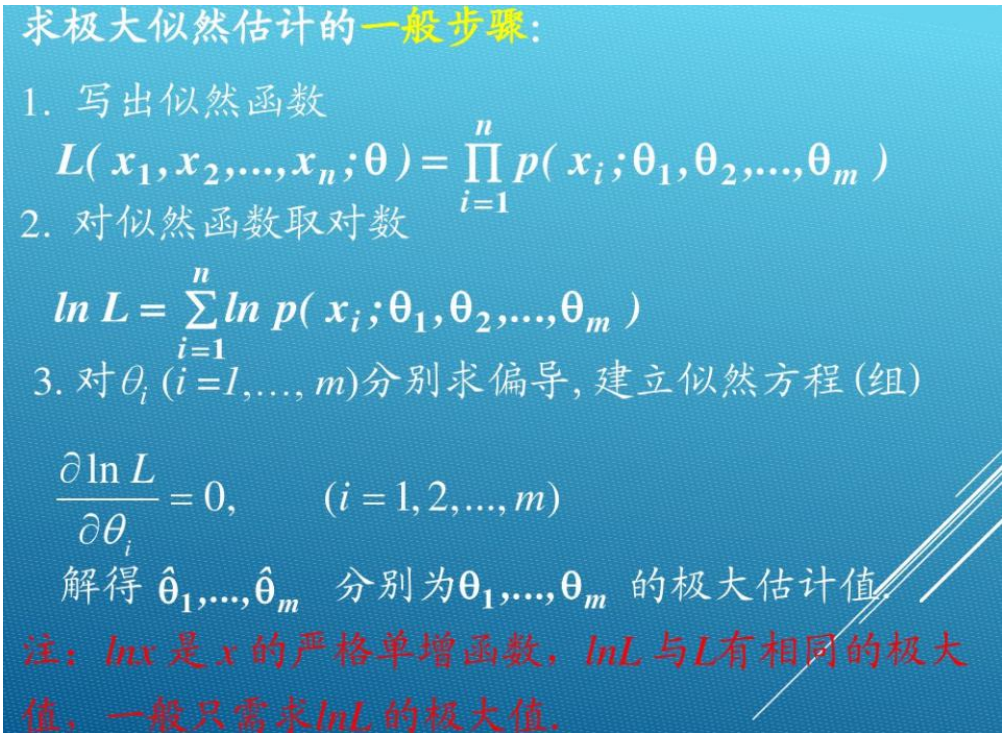
求极大似然函数估计值的一般步骤：

（1） 写出似然函数；

（2） 对似然函数取对数，并整理；

（3） 求导数 ；

（4） 解似然方程



极大似然估计，只是一种概率论在统计学的应用，它是参数估计的方法之一。说的是已知某个随机样本满足某种概率分布，但是其中具体的参数不清楚，参数估计就是通过若干次试验，观察其结果，利用结果推出参数的大概值。极大似然估计是建立在这样的思想上：已知某个参数能使这个样本出现的概率最大，我们当然不会再去选择其他小概率的样本，所以干脆就把这个参数作为估计的真实值。

当然极大似然估计只是一种粗略的数学期望，要知道它的误差大小还要做区间估计。

3.极大似然估计的例子

设某工序生产的产品的不合格率为 p，抽 n 个产品作检验，发现有 T 个不合格，试求 p 的极大似然估计.

分析：设 X 是抽查一个产品时的不合格品个数，则 X 服从参数为 p 的二点分布 $b(1,p)$. 抽查 n 个产品，则得样本 $X_1,X_2,\cdots,X_n$，其观察值为 $x_1,x_2,\cdots,x_n$，假如样本有 T 个不合格，即表示 $x_1,x_2,\cdots,x_n$ 中有 T 个取值为 1， $n-T$ 个取值为

0．按离散分布场合方法，求 p 的极大似然估计.

解：（1）写出似然函数：$L(p)=\prod_{i=1}^{n}p^{x_i}(1-P)^{1-x_i}$

（2）对 $L(p)$ 取对数，得对数似然函数 $l(p)$：

$$l(p)=\sum_{i=1}^{n}[x_i\ln p+(1-x_i)\ln(1-p)]=n\ln(1-p)+\sum_{i=1}^{n}x_i[\ln p-\ln(1-p)]$$

（3）由于 $l(p)$ 对 p 的导数存在，故将 $l(p)$ 对 p 求导，令其为 0，得似然方程：$\frac{dl(p)}{dp}=-\frac{n}{1-p}+\sum_{i=1}^{n}x_i(\frac{1}{p}+\frac{1}{1-p})=-\frac{n}{1-p}+\frac{1}{p(1-p)}\sum_{i=1}^{n}x_i=0$

（4）解似然方程得：$\hat{p}=\frac{1}{n}\sum_{i=1}^{n}x_i=\bar{x}$

（5）经验证，在 $\hat{p}=\bar{x}$ 时，$\frac{d^2l(p)}{dp^2}<0$，这表明 $\hat{p}=\bar{x}$ 可使似然函数达到最大

（6）上述过程对任一样本观测值都成立，故用样本代替观察值便得 p 的极大似然估计为：$\hat{p}=\overline{X}$

将观察值代入，可得 p 的极大似然估计值为：$\hat{p}=\bar{x}=\frac{T}{n}$，其中 $T=\sum_{i=1}^{n}x_i$.

若总体 X 的分布中含有多个未知参数 $\theta_1,\theta_2,\cdots,\theta_k$ 时，似然函数 L 是这些参数的多元函数 $L(\theta_1,\cdots,\theta_k)$. 代替方程（3），我们有方程组 $\frac{\partial(\ln L)}{\partial\theta_i}=0(i=1,2,\cdots,k)$，由这个方程组解得 $\hat{\theta}_1,\hat{\theta}_2,\cdots,\hat{\theta}_k$ 分别是参数 $\theta_1,\theta_2,\cdots,\theta_k$ 的极大似然估计值.

4.无约束优化方法简介（补充）

无约束优化方法是优化技术中极为重要和基本内容之一。它不仅可以直接用来求解无约束优化问题，而且很多约束优化问题也常将其转化为无约束优化问题（比如在 SVM 中我们将有约束条件的最优化问题利用拉格朗日函数转化为无约

束条件的问题，从而利用梯度下降方法等最优化理论进行求解最优解），然后用无约束优化方法来求解。

最速下降法和牛顿法是比较常见的求解无约束问题的最优化方法，这两种算法作为基本算法，在最优化方法中占有重要的地位。其中最速下降法又称梯度法，其**优点是**工作量少，存储变量较少，初始点要求不高；**缺点**是收敛慢，效率低。牛顿法的**优点是**收敛速度快；**缺点是**对初始点要求严格，方向构造困难，计算复杂且占用内存较大。

三、梯度下降法

为什么使用梯度下降法？

（1）**在机器学习的优化问题中，梯度下降法和牛顿法是常用的两种凸函数求极值的方法，他们都是为了求得目标函数的近似解。**在逻辑斯蒂回归模型的参数求解中，一般用改良的梯度下降法，也可以用牛顿法。两种方法我们在课程最后会进行比较。

（2）**梯度实际上是函数值变化最快的方向。**

比如说，你站在一个山上，梯度所指示的方向是高度变化最快的方向。你沿着这个方向走，能最快的改变（增加或是减小）你所在位置的高度，但是如果你乱走，可能走半天所在位置高度也没有变化多少。也就是说，如果你一直沿着梯度走，你就能最快的到达山的某个顶峰或低谷（偶尔会到鞍点，不过这几乎不可能）。

所以实际上，梯度下降法是用来数值搜索局部极小值或极大值的，它是实际应用中一种非常高效，高速且可靠的方法。

接下来首先了解几个数学上的概念：

方向导数：是一个数；反映的是 f(x,y)在 P0 点沿方向 v 的变化率。

偏导数：是多个数（每元有一个）；是指多元函数沿坐标轴方向的方向导数，因此二元函数就有两个偏导数。

偏导函数：是一个函数；是一个关于点的偏导数的函数。

梯度：是一个向量；每个元素为函数对一元变量的偏导数；它既有大小（其大小为最大方向导数），也有方向。

3.1 偏导数

复习偏导数定义：　$z=f(P)=f(x,y)$

$$\frac{\partial f(x,y)}{\partial x}=\lim_{\Delta x\to 0}\frac{f(x+\Delta x,y)-f(x,y)}{\Delta x}$$

$$\frac{\partial f(x,y)}{\partial y}=\lim_{\Delta y\to 0}\frac{f(x,y+\Delta y)-f(x,y)}{\Delta y}$$

偏导数反映了函数$f(P)$沿坐标轴方向上的变化率

例1　求$z=x^2\sin 2y$的偏导数。

解　为求$\frac{\partial z}{\partial x}$,视$y$看作常数，对$x$求导，得

$$\frac{\partial z}{\partial x}=2x\sin 2y$$

为求$\frac{\partial z}{\partial y}$,视$x$看作常数，对$y$求导，得　$\frac{\partial z}{\partial y}=2x^2\cos 2y$

例2　设$f(x,y)=x+y-\sqrt{x^2+y^2}$,求$f_x'(3,4),\ f_y'(0,5)$

解　因为$f_x'(x,y)=1-\frac{2x}{2\sqrt{x^2+y^2}}=1-\frac{x}{\sqrt{x^2+y^2}}$

$$f_y'(x,y)=1-\frac{2y}{2\sqrt{x^2+y^2}}=1-\frac{y}{\sqrt{x^2+y^2}}$$

所以

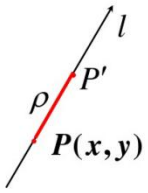
$$f'(3,4)=1-\frac{3}{5}=\frac{2}{5}\qquad f_y'(0,5)=1-1=0$$

3.1 方向导数

定理： 若函数 $f(x,y)$ 在点 $P(x,y)$ 处 可微，
则函数在该点沿任意方向 l 的方向导数存在，且有

$$\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}\cos\alpha+\frac{\partial f}{\partial y}\cos\beta$$

其中 α,β 为 l 的方向角.



证明： 由函数 $f(x,y)$ 在点 P 可微，得

$$\Delta f=\frac{\partial f}{\partial x}\Delta x+\frac{\partial f}{\partial y}\Delta y+o(\rho)$$

$$=\rho\left(\frac{\partial f}{\partial x}\cos\alpha+\frac{\partial f}{\partial y}\cos\beta\right)+o(\rho)$$

故 $$\frac{\partial f}{\partial l}=\lim_{\rho\to 0}\frac{\Delta f}{\rho}=\frac{\partial f}{\partial x}\cos\alpha+\frac{\partial f}{\partial y}\cos\beta$$

二元函数 $f(x,y)$

$$\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}\cos\alpha+\frac{\partial f}{\partial y}\cos\beta$$

其中 α,β 为方向 l 的方向角

特别：

- 当 l 与 x 轴同向 $\left(\alpha=0,\beta=\frac{\pi}{2}\right)$ 时，有 $\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}$
- 当 l 与 x 轴反向 $\left(\alpha=\pi,\beta=\frac{\pi}{2}\right)$ 时，有 $\frac{\partial f}{\partial l}=-\frac{\partial f}{\partial x}$

例 1 求 $z=xe^{2y}$ 在点 $P(1,0)$ 处沿从点 $P(1,0)$ 到点 $Q(2,-1)$ 的方向的方向导数

解： 方向 l 即为 $\overrightarrow{PQ}=\{1,-1\}$

$$\therefore\cos\alpha=\frac{1}{\sqrt{2}},\qquad\cos\beta=\frac{-1}{\sqrt{2}},$$

$$\because\left.\frac{\partial z}{\partial x}\right|_{(1,0)}=\left.e^{2y}\right|_{(1,0)}=1\qquad\left.\frac{\partial z}{\partial y}\right|_{(1,0)}=\left.2xe^{2y}\right|_{(1,0)}=2$$

$$\therefore\frac{\partial z}{\partial l}=1\cdot\cos\alpha+2\cdot\cos\beta=-\frac{\sqrt{2}}{2}$$

3.2 梯度

梯度是一个向量；既有大小，也有方向。
梯度的方向是方向导数中取到最大值的方向，梯度的值是方向导数的最大值。

方向导数公式 $\frac{\partial f}{\partial l}=\frac{\partial f}{\partial x}\cos\alpha+\frac{\partial f}{\partial y}\cos\beta+\frac{\partial f}{\partial z}\cos\gamma$

令向量 $\vec{G}=\left(\frac{\partial f}{\partial x},\frac{\partial f}{\partial y},\frac{\partial f}{\partial z}\right)$

$\vec{l}^{0}=(\cos\alpha,\cos\beta,\cos\gamma)$

$$\frac{\partial f}{\partial l}=\vec{G}\cdot\vec{l}^{0}=|\vec{G}||\vec{l}^{0}|\cos(\vec{G},\vec{l}^{0})\quad(|\vec{l}^{0}|=1)$$

当$\vec{l}^{0}$与$\vec{G}$方向一致时，方向导数取最大值：

$$\max\left(\frac{\partial f}{\partial l}\right)=|\vec{G}|$$

这说明 $\vec{G}$：方向：f 变化率最大的方向；模：f 的最大变化率之值

例 求函数 $u=x^2+2y^2+3z^2+3x-2y$ 在点(1,1,2)处的梯度，并问在哪些点处梯度为零？

解 由梯度计算公式得

$$gradu(x,y,z)=\left(\frac{\partial u}{\partial x},\frac{\partial u}{\partial y},\frac{\partial u}{\partial z}\right)$$

$$=(2x+3,4y-2,6z)$$

故 $gradu(1,1,2)=(5,2,1)=5\vec{i}+2\vec{j}+12\vec{k}$

在$P_0\left(-\frac{3}{2},\frac{1}{2},0\right)$处梯度为 0.

函数 z=f(x,y)在点 P0 处的梯度方向是函数变化率(即方向导数)最大的方向。

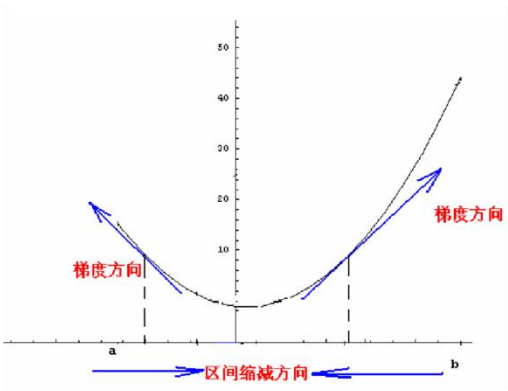
梯度的方向就是函数 f(x,y)在这点增长最快的方向，梯度的模为方向导数的最大值。

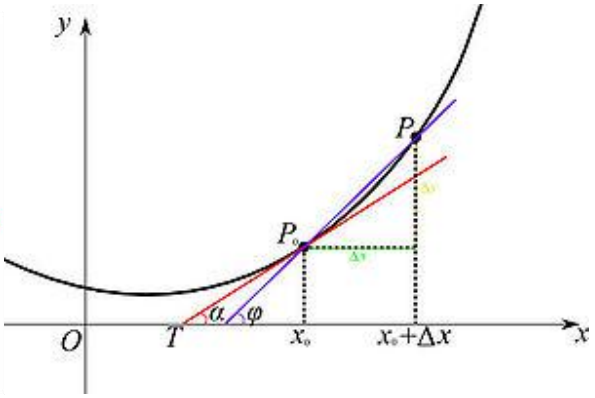
3.4 梯度下降

我们先来玩一个游戏，假如你在一座山上，蒙着眼睛，但是你必须到达山谷中最低点的湖泊，你该怎么办？

对，梯度可以帮助你完成这个游戏。

通过上述数学上面的学习和推导，我们已经了解梯度值是方向导数的最大值，梯度的方向是方向导数取最大值的时候所对应的方向。如下图所示。我们在机器学习中频繁使用梯度的知识求解参数的最优解，这里需要目标凸函数是凸函数的限制，因此在目标函数中经常会取平方项而构成凸函数。





目标函数：

$$J(w)=\frac{1}{2}\sum_{i=1}^{m}(h_w(x^i)-y^{(i)})^2$$

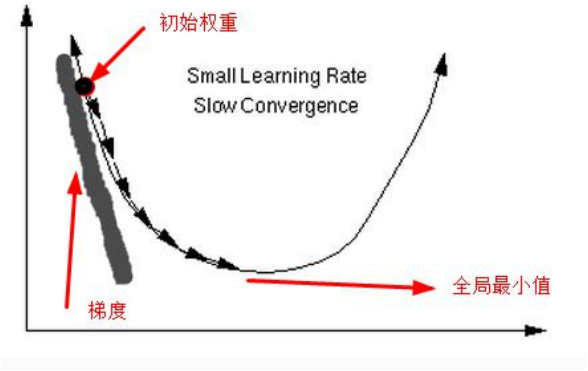
注意：

1. 系数 1/2 只是出于方便考虑，他是我们更加容易求出梯度。

2. 我们使用平方的另一个方面是相对与其他的一些函数（如阶跃函数，在 x=0 处不可导），上述函数在定义域内函数是处处可导的。

3. 加了平方项构成凸函数，我们可以通过梯度下降优化算法来得到权重，并且最优解能使得误差函数最小。

梯度下降算法可以描述为“下山”，直到获得一个局部或者全局最小值。在每次的迭代中，根据学习速率和梯度的斜率，能够确定每次移动的步数。



由此，我们基于上述的损失函数沿着梯度方向权重更新如下：

$$w := w - \eta \Delta J(w), \quad \Delta J(w) = \frac{\partial J}{\partial w_j}$$

我们需要对损失函数 J 中的每个权重 w_j 进行计算，并且把每次做出的更新累加到最后的权值 w 上。其中 η 为学习率，控制每次跨越的步数。$\Delta J(w)$（代价函数的导数）决定朝那个方向去更新权值。

假设这里使用的是简单的线性函数：$h(z^{(i)}) = \sum_i w_j^{(i)} x_j^{(i)}$

$$\Delta J(w) = \frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (y^{(i)} - h(z^{(i)}))^2$$

$$= \frac{1}{2} \frac{\partial}{\partial w_j} \sum_i (y^{(i)} - h(z^{(i)}))^2$$

$$= \frac{1}{2} \bullet 2 \bullet \sum_i (y^{(i)} - h(z^{(i)})) \frac{\partial}{\partial w_j} \sum_i (y^{(i)} - h(z^{(i)}))$$

$$= \sum_i (y^{(i)} - h(z^{(i)})) \frac{\partial}{\partial w_j} \sum_i (y^{(i)} - h(z^{(i)}))$$

$$= \sum_i (y^{(i)} - h(z^{(i)})) \frac{\partial}{\partial w_j} \sum_i (y^{(i)} - \sum_i w_j^{(i)} x_j^{(i)})$$

$$= \sum_i (y^{(i)} - h(z^{(i)}))(-x_j^{(i)})$$

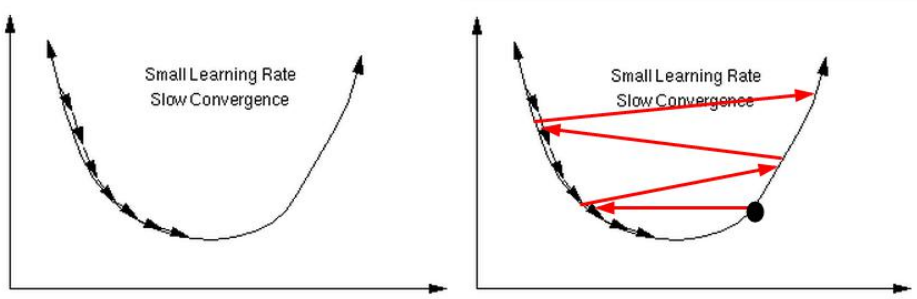
$$= -\sum_i (y^{(i)} - h(z^{(i)}))(x_j^{(i)})$$

上面的梯度下降需要遍历所有的样本点，也就是说权重是基于训练集中所有的样本而完成的（而不是每一次样本的逐渐更新），这就是我们常称作的批量梯度下降法（BGD）。

学习速率的分析：

1.学习速率η设置的过小，使得收敛速度慢，找到全局最优解需要的时间长，计算量大。

2.学习速率η设置的过大，梯度下降算法可能会跳过全局最优解。如下图所示。



优缺点分析：

基于批量梯度下降算法每次每一次遍历的是全部样本，收敛速度慢。一般工程中用到它的改进版本：随机梯度下降。与批量不同的是，随机梯度下降每次只使用一个训练样本渐进的更新权重，一般该方法因为权重更新频繁，通常可以更快收敛，找到最低点或最优值。

随机梯度下降特点：

1.不会精确收敛到最低点，会在最低点附近徘徊

2.速度快

3.无需遍历所有样本点。

3.5 梯度下降求解函数极值：

求函数 $f(x)=x^4-3x^3+2$ 的最小值点

$$f'(x)=4x^3-9x^2$$

$$x_{k+1}=x_k-\alpha f'(x_k)$$

$$f'(x)=4x^3-9x^2=x^2(4x-9)=0$$

$$x=0或x=\frac{9}{4}=2.25$$

α 为步长

满足精度 $|f(x_{k+1})-f(x_k)|<0.0001$

```
#给定初值，xOld记录上一步的x值，xNew下一步迭代的x值
xOld = 0
xNew = 6

#步长
eps = 0.01

#可接受误差
precision = 0.00001

#定义原函数
def f(x):
    return x**4 - 3 * x**3 + 2

#定义导函数
def f_prime(x):
    return 4 * x**3 - 9 * x**2

#主函数
if __name__ == '__main__':
    #循环直到函数值之差满足最小误差
    while abs(f(xNew) -f(xOld)) > precision:
        xOld = xNew
        xNew = xOld - eps * f_prime(xOld)
    #输出极小值点的横纵坐标
    print(xNew,f(xNew))
```

结果输出：

```
(2.2489469258218673, -6.542957528732806)
```

3.6 梯度下降法总结

（1）方向导数是各个方向上的导数
（2）偏导数连续才有梯度存在

（3）梯度的方向是方向导数中取到最大值的方向，梯度的值是方向导数的最大值。

3.7 梯度下降法求解线性回归问题参数的最优解

线性回归中利用极大似然估计和高斯分布推导最小二乘法得到了参数 theta 的解析解（解析解带有矩阵的逆，有时候计算复杂度很高），但是很多时候更希望有数值解，我们就可以使用梯度下降法求解。

参数 theta 解析解：

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

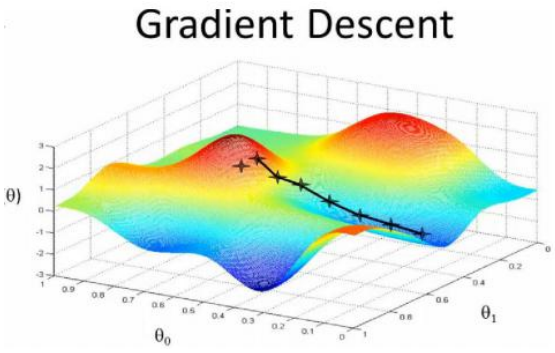
接下来我们沿着梯度的方向下降，注意：给定一个学习率 a，参数要设置适当，设置太小速度太慢，设置太大可能错过最低点。

梯度下降法步骤：

- 随机初始化 theta
- 沿着梯度方向迭代，更新后的 theta 使得 J 最小，a 是学习率

$$\theta = \theta - \alpha \cdot \frac{\partial J(\theta)}{\partial \theta}$$

参考下面图示：



求解目标函数的参数最优解

从新考虑 J 目标函数：

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

利用梯度下降法求导：

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}\left(h_\theta(x)-y\right)^2 \\ &= 2\cdot\frac{1}{2}\left(h_\theta(x)-y\right)\cdot\frac{\partial}{\partial\theta_j}\left(h_\theta(x)-y\right) \\ &= \left(h_\theta(x)-y\right)\cdot\frac{\partial}{\partial\theta_j}\left(\sum_{i=0}^{n}\theta_i x_i - y\right) \\ &= \left(h_\theta(x)-y\right)x_j\end{aligned}$$

批量梯度下降

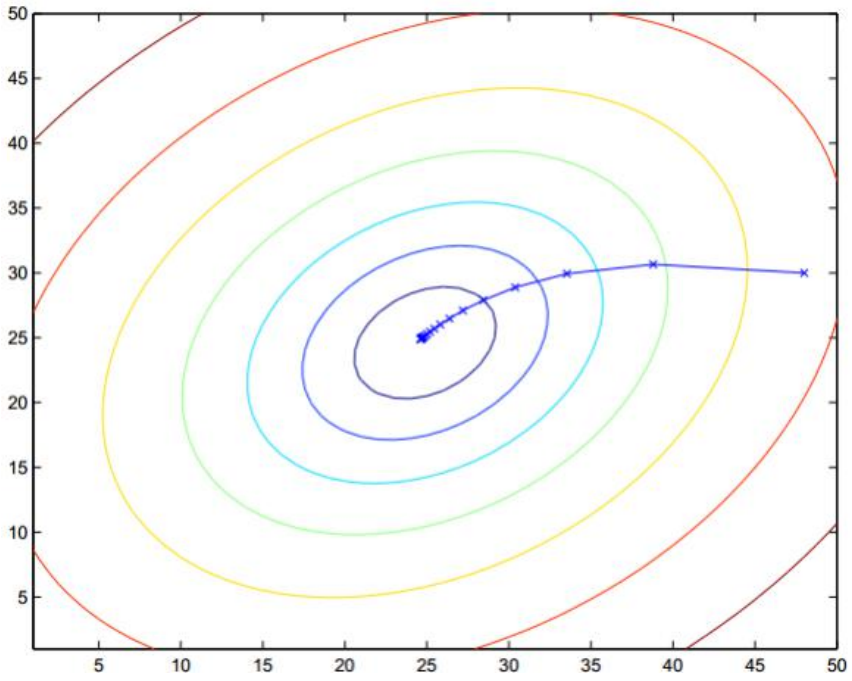
更新规则--所有样本都参与了 theta 的更新和求解，这称之为批量梯度方法，批量梯度下降法可以找到**线性回归**的全局最小值（为什么？因为目标函数是凸函数，凸函数有且只有一个全局最小值），但算法本身局限在于可能存在局部最优解，但不是全局最优解。

Repeat until convergence {

$$\theta_j := \theta_j + \alpha\sum_{i=1}^{m}\left(y^{(i)}-h_\theta(x^{(i)})\right)x_j^{(i)}$$

}

下图为批量梯度下降图示：



随机梯度下降法 SGD

（特点）更快，在线，可以跳过局部最小值，有可能找不到全局最优值，有时候会在局部最优值点发生震荡，但是一般情况下在一定位置发生震荡，认为模型收敛了。SGD 比 BGD 更能收敛到全局最优值。

Loop {
    for i=1 to m, {
        $\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$
    }
}

Mini-batch

上述两种梯度下降的折中

通常将 Mini-batch SGD 也称之为 SGD

如果不是每拿到一个样本更改梯度，而是若干样本的平均梯度作为更新方向，则是 mini-batch 梯度下降算法。

Repeat until convergence {

$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$

}

Loop {
    for i=1 to m, {
        $\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$
    }
}

四、牛顿法

牛顿法和我们之前听过的梯度下降法都是属于最优化理论部分，最常见的情形就是利用目标函数的导数通过多次迭代来求解无条件约束最优化问题。实现简单，coding 方便，是训练模型的必备利器之一。

4.0 泰勒公式（讲解）

$$f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f^{(2)}(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n + R_n(x)$$

$$f(x) = f(0) + \frac{f'(0)}{1!}x + \frac{f^{(2)}(0)}{2!}x^2 + \frac{f^{(3)}(0)}{3!}x^3 + \cdots + \frac{f^{(n)}(0)}{(n)!}x^n + R_n(x)$$

下面是两个例子：

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \qquad -\infty < x < \infty$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \cdots + (-1)^{m-1} \frac{x^{2m-1}}{(2m-1)!} + R_{2m}$$

4.1.求解方程根

不是所有的方程 $f(x)=0$ 都有求根公式，或者求根公式很复杂，导致求解困难，可以利用牛顿法，可以迭代求解。

利用泰勒公式，在 x_0 处展开到一阶

$$f(x)=f(x_0)+(x-x_0)f'(x_0)$$

求解方程 $f(x)=0$，即 $f(x_0)+(x-x_0)f'(x_0)=0$

解得：

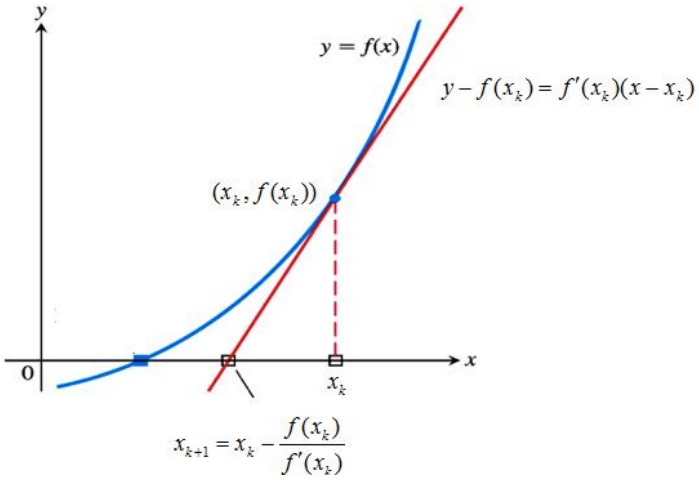
$$x=x_1=x_0-\frac{f(x_0)}{f'(x_0)}$$

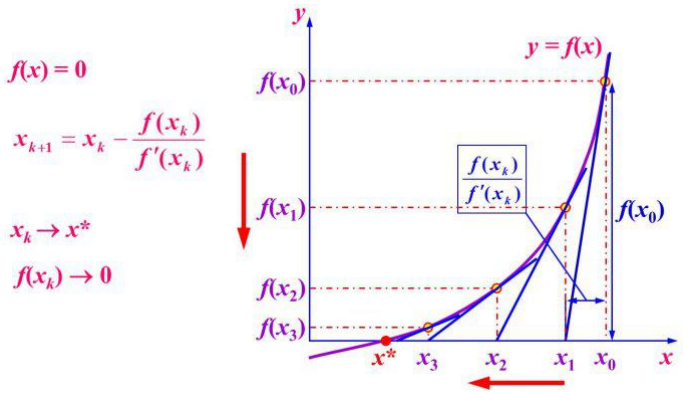
利用泰勒公式的一阶展开，$f(x)=f(x_0)+(x-x_0)f'(x_0)$ 处并不是完全相等，而是近似相等，这里求得的 x_1 并不能让 $f(x)=0$，只能说 $f(x_1)$ 比 $f(x_0)$ 更接近 $f(x)=0$，于是继续迭代求解。可以推出：

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$

通过迭代，上面式子一定能在 $f(x^*)=0$ 的时候收敛。也成为牛顿法的一次迭代（收敛）。

通过下图理解迭代求解的过程：





4.2.实战牛顿法求解方程

例：利用牛顿法求方程 $f(x)=x^3-2=0$，解为 $x=\sqrt[3]{2}=1.259921$

利用以下公式求解：

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$

```
#定义原函数
def f(x):
    return x**3.0-2.0

#定义导函数
def df(x):
    return 3.0* x**2.0

#定义迭代值
def g(x):
    return x- f(x)/df(x)
#定义初始值
x=1.0
#定义误差
r=1.0

#循环100次
for i in range (100):
    #迭代值赋值
    x1=g(x);
    #误差赋值
    r=abs(x1-x)
    #可接受误差
    if r<1e-10:
        print 'step:%d' %i
        break
    #更新下一步起始位置
    x=x1
    #显示迭代步骤
    print 'step:%d, x=%f' %(i,x)

print 'remaind error= %f' %r
print 'x=%f' %x
print 'check f(x)=%f , the result is %r' %(f(x), f(x)==0)
```

结果：

```
step:0, x=1.333333
step:1, x=1.263889
step:2, x=1.259933
step:3, x=1.259921
step:4, x=1.259921
step:5
remaind error= 0.000000
x=1.259921
use time: 0.000061 seconds
check f(x)=0.000000, the result is True
```

4.3 牛顿迭代法

（1）一次迭代

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

（2）二次迭代（收敛）

(1) 找到一个θ，s.t.$f(\theta) = 0$

根据导数定义$\Delta = \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$

$$f'(\theta^{(0)}) = \frac{f(\theta^{(0)})}{\Delta} = \frac{f(\theta^{(0)})}{\theta^{(0)} - \theta^{(1)}}$$

$$\theta^{(1)} = \theta^{(0)} - \frac{f(\theta^{(0)})}{f'(\theta^{(0)})}$$

同理，当第$k+1$次迭代后的结果为：

$$\theta^{(k+1)} = \theta^{(k)} - \frac{f(\theta^{(k)})}{f'(\theta^{(k)})}$$

（2）假设目标函数为$l(\theta)$,找一个θ，$s.t. l'(\theta) = 0$

$$\theta^{(k+1)} = \theta^{(k)} - \frac{l'(\theta^{(k)})}{l''(\theta^{(k)})}$$

（3）具体推导如下过程：

考虑如下无约束极小化问题：

$$\min_x f(x)$$

其中 x 为实数，假设 $f(x)$ 为凸函数，二阶可导。

在当前点进行泰勒二阶展开：

$$\varphi(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2$$

求一阶导数

$$\varphi'(x) = f'(x_k) + f''(x_k)(x - x_k)$$

在极小值点满足 $\varphi'(x) = 0$，求得

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

下一个搜索点为展开后的极小值点。

4.4.多维特征的牛顿迭代法

若扩展到多维，上式变为

$$x_{k+1} = x_k - H_k^{-1} \cdot g_k$$

其中 g_k 为多元函数 $f(x_1, x_2, ..., x_n)$ 的梯度(一阶导数)， H_k 为多元函数 $f(x_1, x_2, ..., x_n)$ 的二阶导数，即 Hessian 矩阵

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

牛顿法是具有二次收敛性的算法，收敛速度比较快。但是其步长固定，因此不能保证稳定的下降。

$$x_{k+1} = x_k - H_k^{-1} \cdot g_k = x_k + d_k$$

式子表达下一点的计算方法：x_k在方向d_k上按步长 1 移动到下一点x_{k+1}，d_k称为牛顿方向。

$$d_k = -H_k^{-1} \cdot g_k$$
$$H_k \cdot d_k = -g_k$$

注意实际中d_k是通过求解线性方程组$H_k \cdot d_k = -g_k$获得的。

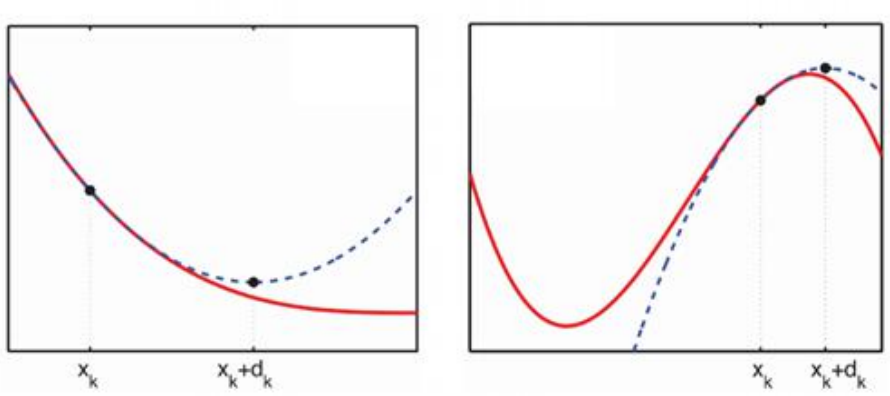
如果函数f在定义域内二阶连续可导，那么f的 Hessian 矩阵H在定义域内为对称矩阵，如果函数f连续，则二阶偏导数的求导顺序没有区别，即

$$\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial y}\right) = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial x}\right)$$

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

H 阵为对称阵。

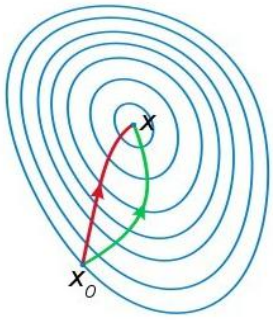
$$\varphi(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(x_k)(x - x_k)^2$$



当目标函数是二次函数时，二阶泰勒展开函数与原目标函数完全相同的二次式，Hessian 矩阵为常数矩阵。只需要一步迭代即可达到 $f(x)$ 的极小点，因此牛顿法是一种具有二阶收敛性的算法。

对于非二次函数，若函数的二次性态较强，或迭代点已进入极小点的邻域，则收敛速度也是很快的，这是牛顿法的主要优点。

牛顿法可以利用到曲线本身的信息，比梯度下降法更容易收敛（迭代更少次数），如下图是一个最小化一个目标方程的例子，红色曲线是利用牛顿法迭代求解，绿色曲线是利用梯度下降法求解。



$$x_{k+1} = x_k - H_k^{-1} \cdot g_k = x_k + d_k$$

4.5 两个改进方法

阻尼牛顿法：在牛顿方向上附加了步长因子，每次调整时会在搜索空间，在该方向找到最优步长，然后调整。

拟牛顿法：对 H_K 或 H_K^{-1} 取近似值，可减少计算量。

4.6.牛顿法求解实例

例 用牛顿法求解：$\min f(x) = \frac{1}{2}x_1^2 + \frac{9}{2}x_2^2 \quad x^0 = (9,1)^T$

牛顿法公式 $x_{k+1} = x_k - H_k^{-1} \cdot g_k = x_k + d_k$

解 $\nabla f(x)=\begin{pmatrix} x_1 \\ 9x_2 \end{pmatrix}$，$\nabla^2 f(x)=\begin{pmatrix} 1 & 0 \\ 0 & 9 \end{pmatrix}$

$$d^0=-[\nabla^2 f(x)]^{-1}\nabla f(x)=-\begin{pmatrix} 1 & 0 \\ 0 & 9 \end{pmatrix}^{-1}\begin{pmatrix} 9 \\ 9 \end{pmatrix}=-\begin{pmatrix} 1 & 0 \\ 0 & \frac{1}{9} \end{pmatrix}\begin{pmatrix} 9 \\ 9 \end{pmatrix}=-\begin{pmatrix} 9 \\ 1 \end{pmatrix}$$

$$x^1=x^0+d^0=\begin{pmatrix} 9 \\ 1 \end{pmatrix}-\begin{pmatrix} 9 \\ 1 \end{pmatrix}=-\begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

因$\left\|\nabla f(x^1)\right\|=0$，所以迭代终止，最优点为：$x^*=x^1=(0,0)^T$

4.7.梯度下降法与牛顿法的比较

	梯度下降法	牛顿法
迭代公式	$x_{k+1}=x_k-\alpha f(x_k)$	$x_{k+1}=x_k-H_k^{-1}\cdot g_k$
物理意义	1、搜寻函数的最低点 2、搜索方向是目标函数一阶导数的方向 3、每次迭代步长由参数 α 决定	1、搜索目标函数一阶导数等于 0 的点 2、搜索方向是目标函数二阶导数 3、无步长参数
复杂度	只需计算一阶导数，时间复杂度低	需要计算 Hessian 矩阵及其逆矩阵，时间复杂度高
收敛速度	收敛慢，迭代次数多	收敛快，迭代次数少
参数选择	需要选择步长	无需选择参数
适用范围	适用特征数较多的场景	适用特征数较少的场景

牛顿法是二阶收敛，梯度下降是一阶收敛，所以牛顿法就更快。如果更通俗地说的话，比如你想找一条最短的路径走到一个盆地的最底部，梯度下降法每次只从你当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，可以说牛顿法比梯度下降法看得更远一点，能更快地走到最底部。

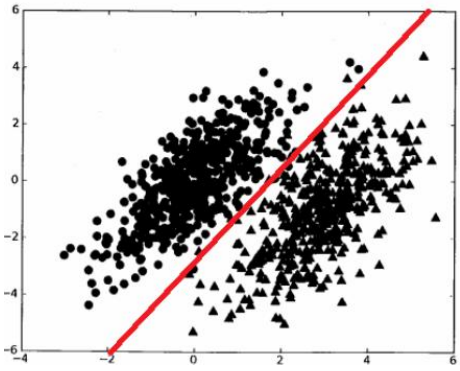
五、Logistic Regression (逻辑回归)

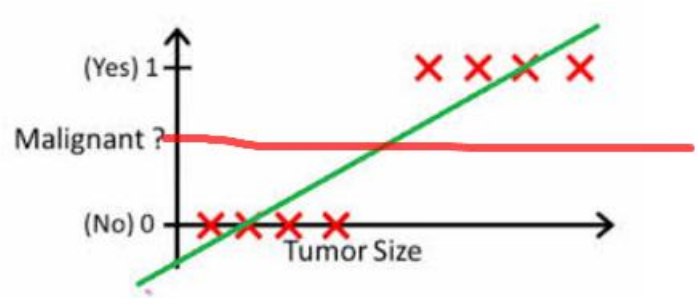
分类问题的首选算法。（类别+概率）

应用场景：对于许多应用实践来说，我们不但对类标预测感兴趣，而且对事件属于某一类别的概率进行预测也非常有用。例如，将逻辑斯特回归应用于天气预报，不仅要预测某天是否会下雨，还要推测下雨有多大的可能性。同样，逻辑斯特回归还可用于预测出现某些症状的情况下，患者患有某种疾病的可能性，这也是逻辑斯特回归在医疗领域得到了很广泛的研究。

5.1.例子

如果 y 是连续值，我们可以采用回归分析，但如果 y 是离散值，使用 logistic 回归解决回归问题。

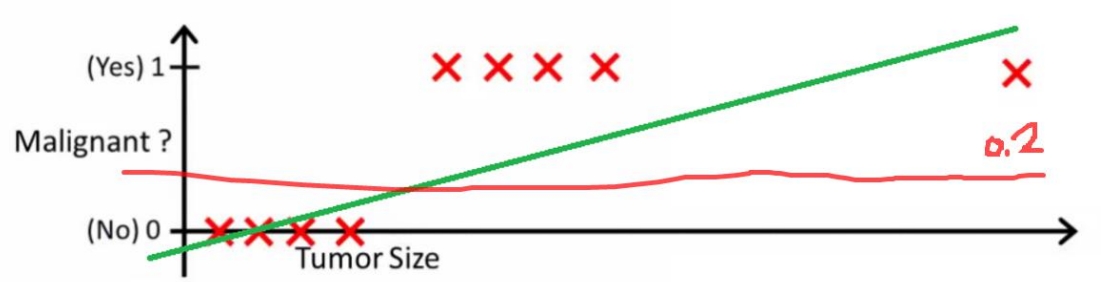




设置 h(x) > 0.5

通过上图中的分割线，如果新样本是一个比较特别的点，如下图最右侧的 X，新样本应该预测为 1，但根据上面的分割线却被分割到了第 0 类，因此，对于分类的问题用先行回顾较差。

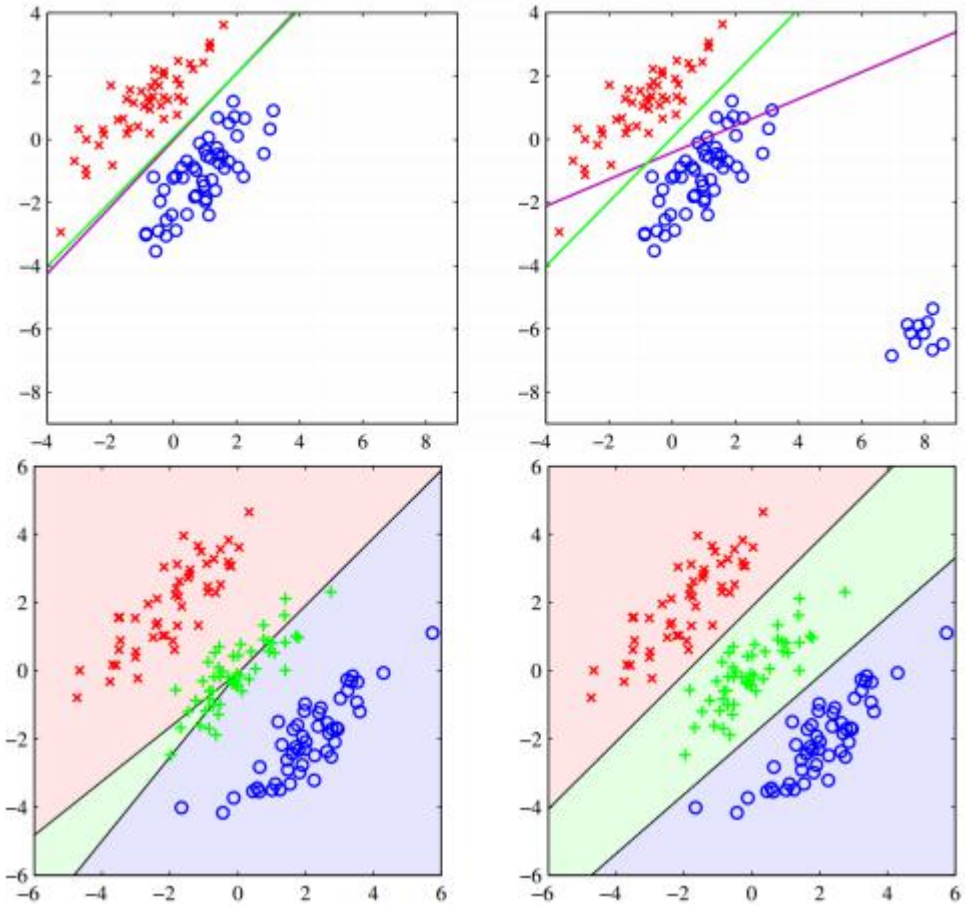
这种分类实际上可以可以设定一个 y 值，如果 h(x) > 0.5 是第一类，h(x) <0.5 是第 0 类，下图可能会设置 y=0.2 可以更好的分类。



设置 h(x) > 0.2

看图观察：

图 1 是线性回归，图 2 是 logistic 回归（效果好）
图 3 是线性回归处理多类别问题，图 4 是 softmax 处理多类别问题（效果好）



5.2 基本模型

5.2.1 函数模型

（1）逻辑斯特函数的由来

假设一事件发生的概率为 P，则不发生的概率为 1-P，我们把发生概率/不发生概率称之为发生的概率比，数学公式表示为：

$$\frac{p}{1-p}$$

更进一步我们定义 logit 函数，它是概率比的对数函数（log-odds）

$$\text{logit}(p)=\log(\frac{p}{1-p})$$

Logit 函数输入值范围介于[0,1]之间，它能将输入转换到整个实数范围内。

对 logit 函数求反函数，我们将 logit 的反函数叫做 logistic 函数**（课堂推到如何求解反函数，属于高中阶段的知识）**

$$\text{logistic}(z)=sig\,\text{mod}(z)=\frac{1}{1+e^{-z}}$$

（2）进一步探究

测试数据为 $X(x_0,x_1,x_2.....x_n)$

要学习的参数为：$\theta(\theta_0,\theta_1,\theta_2.....\theta_n)$

模型的线性表示：（样本特征与权重的线性组合）

$$Z=\theta_0x_0+\theta_1x_1+\theta_2x_2+...\theta_nx_n$$

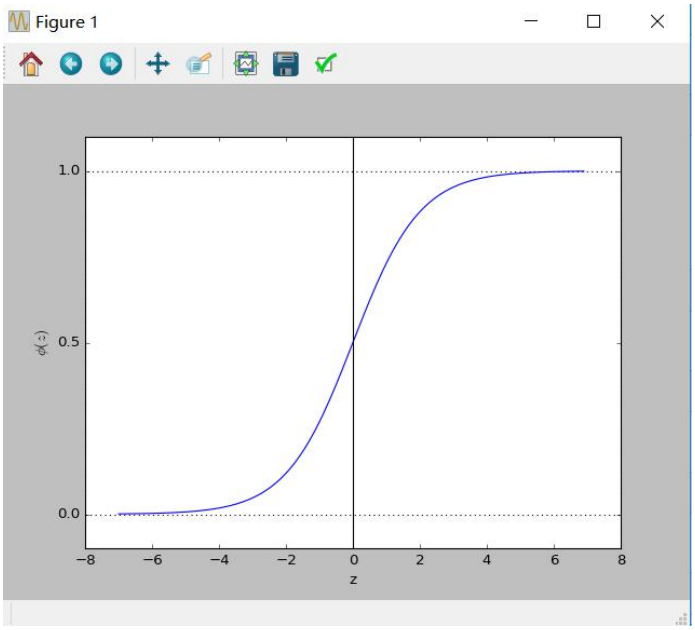
向量表示：

$$Z=\theta_0x_0+\theta_1x_1+\theta_2x_2+...\theta_nx_n=\Theta^Tx_n$$

处理二值数据，引入 Sigmoid 函数时曲线平滑化

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

该函数的图像如下图：



对图像的理解：sidmod 函数以实数值作为输入并将其反射到[0，1]区间，拐点在 y=0.5 地方。

5.2.2 Sigmod 函数绘图实战

需求：绘制[-7，7]的 sigmod 函数图像

```
import matplotlib.pyplot as plt
import numpy as np
def sigmod(z):
    return 1.0/(1.0+np.exp(-z))
z=np.arange(-7,7,0.1)
phi_z=sigmod(z)
plt.plot(z,phi_z)
plt.axvline(0.0,color='k')
plt.axhspan(0.0,1.0,facecolor='1.0',alpha=1.0,ls='dotted')
plt.yticks([0.0,0.5,1.0])
plt.ylim(-0.1,1.1)
plt.xlabel('z')
plt.ylabel('$\phi (z)$')
plt.show()
```

2.3 函数表示

预测函数：

$$h_\theta(X) = g(\Theta^T X) = \frac{1}{1+e^{-\Theta^T X}}$$

用概率表示：

正例(y=1)：

$$P(y=1 \mid X;\Theta) = h_\theta(X)$$

反例(y=0):

$$P(y=0 \mid X;\Theta) = 1 - h_\theta(X)$$

用一个函数综合表述上面两个函数：

$P(y \mid X;\Theta) = h_\theta(X)^y[1-h_\theta(X)]^{1-y}$

当y = 0时只有第二项有效，与$P(y=0 \mid X;\Theta)$一致

当y = 1时只有第一项有效，与$P(y=1 \mid X;\Theta)$一致

5.3 似然函数

逻辑斯特回归似然函数一般定义为：

注意：这里数据集中的每个样本都是相互独立的

$$L(\theta) = P(\hat{y} \mid X;\Theta) = \prod_i P(y^{(i)} \mid x^{(i)};\theta) = \prod_i h_\theta(x^{(i)})^{y^{(i)}}[1-h_\theta(x^{(i)})]^{1-y^{(i)}}$$

注意：$x^{(i)}$表示输入向量x的第i个分量（i维或列）

$x_j^{(i)}$表示输入向量第x_j个样本的第i个分量

接下来的目标变为了：找参数θ的极大似然估计

（1）目标函数加 log 处理

$$L(\theta) = \log(L(\theta)) = \log(\prod_i P(y^{(i)} \mid x^{(i)};\theta))$$

$$= \log(\prod_i h_\theta(x^{(i)})^{y^{(i)}}[1-h_\theta(x^{(i)})]^{1-y^{(i)}})$$

$$= \sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log[1-h_\theta(x^{(i)})]$$

使用到的公式：

$\log(AB) = \log A + \log B$,一般底数为2

$\log_{a^b}^{c^d} = \frac{d}{b}\log_a^c, a$为底数c为真数

（2）目标函数求导，并利用梯度上升算法求解参数

第一种求导思路：

$$L(\theta)=\log(L(\theta))=\log(\prod_i P(y^{(i)}\mid x^{(i)};\theta))$$

$$=\log(\prod_i h_\theta(x^{(i)})^{y^{(i)}}[1-h_\theta(x^{(i)})]^{1-y^{(i)}})$$

$$=\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)})+(1-y^{(i)})\log[1-h_\theta(x^{(i)})]$$

$$=\sum_{i=1}^{m}[y^{(i)}\log(\frac{h_\theta(x^{(i)})}{1-h_\theta(x^{(i)})})+\log(1-y^{(i)})$$

而$h_\theta(x^{(i)})=\frac{1}{1+e^{-\theta x^{(i)}}},\log(\frac{h_\theta(x^{(i)})}{1-h_\theta(x^{(i)})})=\theta x^{(i)},$

因为$1-y^{(i)}=1-\frac{1}{1+e^{-\theta x^{(i)}}}=\frac{1+e^{-\theta x^{(i)}}-1}{1+e^{-\theta x^{(i)}}}=\frac{e^{-\theta x^{(i)}}}{1+e^{-\theta x^{(i)}}}$

则$\log(1-y^{(i)})=\log(\frac{e^{-\theta x^{(i)}}}{1+e^{-\theta x^{(i)}}})=-\log(\frac{e^{-\theta x^{(i)}}+1}{e^{-\theta x^{(i)}}})=-\log(1+e^{\theta x^{(i)}})$

$$=\sum_{i=1}^{m} y^{(i)}(\theta x^{(i)})-\log[1+\exp(\theta x^{(i)})]$$

求导：

$$\frac{\partial}{\partial\theta_j}L(\theta)=\sum_{i=1}^{m}[y^{(i)}x^{(i)}-h_\theta(x^{(i)})x^{(i)}]$$

$$=\sum_{i=1}^{m}[y^{(i)}-h_\theta(x^{(i)})]x_j{}^{(i)}$$

注：$-(\log\frac{1}{1+e^{\theta x}})'=-[\frac{1}{\frac{1}{1+e^{\theta x}}\bullet 1}\bullet\frac{-xe^{\theta x}}{(1+e^{\theta x})^2}]=\frac{xe^{\theta x}}{(1+e^{\theta x})}=\frac{x}{(1+e^{-\theta x})}=h_\theta(x)\bullet x$

求导的第二种思路：（建议这种方式）

条件：$h_\theta(z)=\frac{1}{1+e^{-z}}$,其中$z=\theta^T x=\theta_0 x_0+\theta_1 x_1+...+\theta_m x_m$

$$L(\theta)=\log(L(\theta))=\log(\prod_i P(y^{(i)}\mid x^{(i)};\theta))$$

$$=\log(\prod_i h_\theta(z^{(i)})^{y^{(i)}}[1-h_\theta(z^{(i)})]^{1-y^{(i)}})$$

$$=\sum_{i=1}^{m} y^{(i)}\log h_\theta(z^{(i)})+(1-y^{(i)})\log[1-h_\theta(z^{(i)})]$$

$$\frac{\partial L(\theta)}{\partial\theta}=[y^{(i)}\frac{1}{h_\theta(z^{(i)})}-(1-y^{(i)})\frac{1}{1-h_\theta(z^{(i)})}]\frac{\partial h_\theta(z^{(i)})}{\partial\theta}$$

$$\frac{\partial h_\theta(z)}{\partial\theta}=\frac{\partial}{\partial\theta}\frac{1}{1+e^{-z}}=\frac{1}{(1+e^{-z})^2}e^{-z}=\frac{1}{(1+e^{-z})}(1-\frac{1}{1+e^{-z}})=h_\theta(z)(1-h_\theta(z))$$

代入上述

$$\frac{\partial L(\theta)}{\partial\theta}=[y^{(i)}\frac{1}{h_\theta(z^{(i)})}-(1-y^{(i)})\frac{1}{1-h_\theta(z^{(i)})}]h_\theta(z^{(i)})(1-h_\theta(z^{(i)}))\frac{\partial z}{\partial\theta_j}$$

$$=[y^{(i)}(1-h_\theta(z^{(i)}))-(1-y^{(i)})h_\theta(z^{(i)})]x_j$$

$$=[y^{(i)}-h_\theta(z^{(i)})]x_j$$

（3）梯度上升算法求解参数

梯度下降法更新θ_j参数的值：

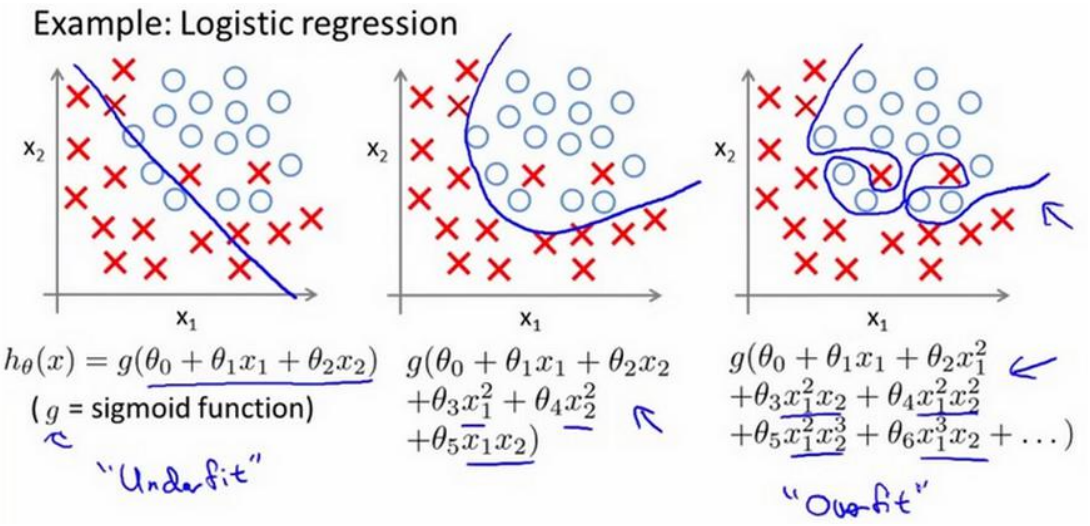
$$\theta_j:=\theta_j+\eta\sum_{i=1}^{m}[y^{(i)}-h_\theta(x^{(i)})]x_j{}^{(i)}$$

请一定要注意，求解似然函数一般是求解最大值，使用梯度上升算法，使得最优解不断增大。相对应的是损失函数，一般是似然函数的相反数，所以使用的是梯度下降法求解损失函数的最小值参数。

通过推导我们发现逻辑斯特回归的代价函数与线性回归形式上很像，不同之处在于模型假设不一样，线性回归是$\theta^T x$， 而逻辑回归在此基础上多了一层映射$h_\theta(X)=g(\Theta^T X)=\frac{1}{1+e^{-\Theta^T X}}$。

上述的似然函数，我们可以在目标函数中加一个负号变为损失函数或代价函数，从而原来的梯度上升的算法就变为了梯度下降算法求解参数的值。

5.4.通过正则化解决拟合问题



过拟合的问题是机器学习中常见的问题，它是指模型在训练数据上表现良好，但是用于未知数据（测试数据）时性能不佳。如果一个模型出现了过拟合的问题，我们也说这个模型有高方差，这就有可能是因为使用了相关数据中过多的参数，从而使得模型变得复杂。同样，模型也可能出现欠拟合，这意味着模型过于简单，无法发现训练数据集中的隐含的模式，这也会使得模型应用于未知数据的时候表现不佳。

偏差-方差权衡 bias 和 variance 就是通过正则化调整模型的复杂度，我们通过正则化解决共线性（特征高度相似的）一个很用的办法。它可以过滤掉数据中的噪声。具体的做法就是引入额外的信息或偏差对极端的参数权重做出惩罚。

最常用的是 L2 正则化，有时也称为 L2 收缩或衰减，写作如下公式：

$$\frac{\lambda}{2}\|\mathrm{w}\|=\frac{\lambda}{2}\sum_{j=1}^{m}w_j^2\text{，}\ \lambda\text{为正则化系数}$$

在逻辑斯特函数中，只需要在最大似然函数中加入正则化项，以降低系数带来的副作用：

$$L(\theta)=\sum_{i=1}^{m}y^{(i)}\log h_\theta(x^{(i)})+(1-y^{(i)})\log[1-h_\theta(x^{(i)})]+\frac{\lambda}{2}\|\mathrm{w}\|^2$$

通过正则化系数 λ，保持权值较小的时候，我们就可以控制模型与训练数据的拟合程度，加入 λ 值，可以增强正则化的强度。

Softmax 回归

多类别处理问题方法

1.类别概率

$$p(c=k\mid x;\theta)=\frac{\exp\left(\theta_k^T x\right)}{\sum_{l=1}^{K}\exp\left(\theta_l^T x\right)},\quad k=1,2,\cdots,K$$

2.似然函数

$$L(\theta)=\prod_{i=1}^{m}\prod_{k=1}^{K}p\left(c=k\mid x^{(i)};\theta\right)^{y_k^{(i)}}=\prod_{i=1}^{m}\prod_{k=1}^{K}\left(\frac{\exp\left(\theta_k^T x^{(i)}\right)}{\sum_{l=1}^{K}\exp\left(\theta_l^T x^{(i)}\right)}\right)^{y_k^{(i)}}$$

3.对数似然

$$J_m(\theta)=\ln L(\theta)=\sum_{i=1}^{m}\sum_{k=1}^{K}y_k^{(i)}\cdot\left(\theta_k^T x^{(i)}-\ln\sum_{l=1}^{K}\exp\left(\theta_l^T x^{(i)}\right)\right)$$

$$J(\theta)=\sum_{k=1}^{K}y_k\cdot\left(\theta_k^T x-\ln\sum_{l=1}^{K}\exp\left(\theta_l^T x\right)\right)$$

4.随机梯度

$$\frac{\partial J(\theta)}{\partial\theta_k}=\left(y_k-p(y_k\mid x;\theta)\right)\cdot x$$

六、Scikit-learn 实现 LR

6.1.代码实战

```
from sklearn import datasets
```

```
import numpy as np
```

```
iris=datasets.load_iris()
```

```
X=iris.data[:,[2,3]]#花瓣长度和花瓣宽度
```

```
y=iris.target
```

```
from sklearn.cross_validation  import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
sc.fit(X_train)
X_train_std=sc.transform(X_train)
X_test_std=sc.transform(X_test)
```

```
X_combined_std=np.vstack((X_train_std,X_test_std))
y_combined=np.hstack((y_train,y_test))
```

```
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
```

```
lr=LogisticRegression(C=1000.0,random_state=0)
```

```
lr.fit(X_train_std,y_train)
```

```
LogisticRegression(C=1000.0, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=0,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

```
#自定义绘图函数
```

```
from matplotlib.colors import ListedColormap
```

```
def plot_decision_regions(X,y,classifier,test_idx=None,resolution=0.02):
    makers=('s','x','o','^','v')
    colors=('red','blue','lightgreen','gray','cyan')
    cmap=ListedColormap(colors[:len(np.unique(y))])
    #plot decision surface
    x1_min,x1_max=X[:,0].min()-1,X[:,0].max()+1
    x2_min,x2_max=X[:,1].min()-1,X[:,1].max()+1
    xx1,xx2=np.meshgrid(np.arange(x1_min,x1_max,resolution),
                        np.arange(x2_min,x2_max,resolution))
    Z=classifier.predict(np.array([xx1.ravel(),xx2.ravel()]).T)
    Z=Z.reshape(xx1.shape)
    plt.contourf(xx1,xx2,Z,alpha=0.4,cmap=cmap)
    plt.xlim(xx1.min(),xx1.max())
    plt.ylim(xx2.min(),xx2.max())
    #plot class sample
    X_test,y_test=X[test_idx,:],y[test_idx]
    for idx,cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y==cl,0],y=X[y==cl,1],\
                    alpha=0.8,c=cmap(idx),\
                    marker=makers[idx],label=cl)
    #highlight test sample
    if test_idx:
        X_test,y_test=X[test_idx,:],y[test_idx]
        plt.scatter(X_test[:,0],X_test[:,1],c='',alpha=1.0,linewidth=1,marker='o',s=55,label='test set')
```

```
plot_decision_regions(X_combined_std,y_combined,classifier=lr,test_idx=range(105,150))
```

```
plt.xlabel('x')#标准化处理之后的花瓣长度
```

```
<matplotlib.text.Text at 0x8a077b8>
```

```
plt.ylabel('y')#经标准化后的花瓣宽度
```

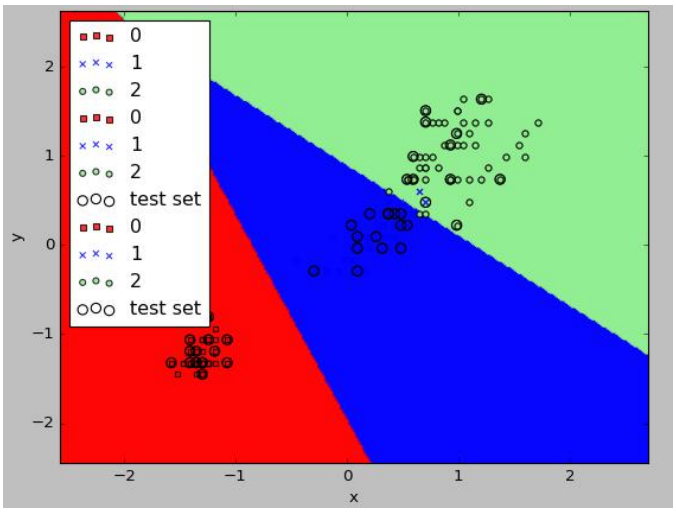
```
<matplotlib.text.Text at 0x89cacc0>
```

```
plt.legend(loc='upper left')
```

```
<matplotlib.legend.Legend at 0x10c21eb8>
```

```
plt.show()
```

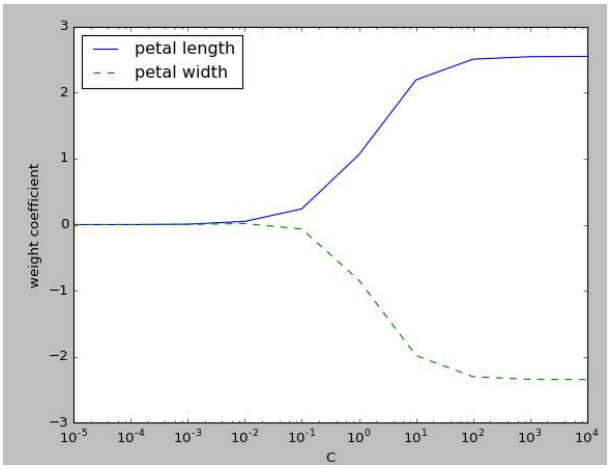
结果：



6.2.LR 正则化之后的 LR

```
weights, params = [], []
for c in np.arange(-5., 5.):
    lr = LogisticRegression(C=10.**c, random_state=0)
    lr.fit(X_train_std, y_train)
    weights.append(lr.coef_[1])
    params.append(10**c)
weights = np.array(weights)
plt.plot(params, weights[:, 0],label='petal length')
plt.plot(params, weights[:, 1], linestyle='--', label='petal width')
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.legend(loc='upper left')
plt.xscale('log')
plt.show()
```

运行结果显示：



执行上述代码，使用不同的逆正则化参数 C 拟合了 10 个逻辑斯特回归模型，上图只显示了类别 2 区别于其他类别的权重参数。

七、良恶性肿瘤判别

7.1 获取数据

In [1]:
```
# 导入pandas与numpy工具包。
import pandas as pd
import numpy as np

# 创建特征列表。
column_names = ['Sample code number', 'Clump Thickness', 'Uniformity of Cell Size',
                'Uniformity of Cell Shape',
                'Marginal Adhesion', 'Single Epithelial Cell Size', 'Bare Nuclei',
                'Bland Chromatin', 'Normal Nucleoli', 'Mitoses', 'Class']

# 使用pandas.read_csv函数从互联网读取指定数据。
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-databases
                    /breast-cancer-wisconsin/breast-cancer-wisconsin.data',
                    names = column_names )

# 将?替换为标准缺失值表示。
data = data.replace(to_replace='?', value=np.nan)
# 丢弃带有缺失值的数据（只要有一个维度有缺失）。
data = data.dropna(how='any')

# 输出data的数据量和维度。
data.shape
```

Out[1]: (683, 11)

7.2 数据切分

In [2]:
```
# 使用sklearn.cross_valiation里的train_test_split模块用于分割数据。
from sklearn.cross_validation import train_test_split

# 随机采样25%的数据用于测试，剩下的75%用于构建训练集合。
X_train, X_test, y_train, y_test = train_test_split
(data[column_names[1:10]], data[column_names[10]], test_size=0.25, random_state=33)
```

In [3]:
```
# 查验训练样本的数量和类别分布。
y_train.value_counts()
```

Out[3]:
```
2    344
4    168
Name: Class, dtype: int64
```

In [4]:
```
# 查验测试样本的数量和类别分布。
y_test.value_counts()
```

Out[4]:
```
2    100
4     71
Name: Class, dtype: int64
```

7.3 特征工程

In [5]:
```
# 从sklearn.preprocessing里导入StandardScaler。
from sklearn.preprocessing import StandardScaler
# 从sklearn.linear_model里导入LogisticRegression与SGDClassifier。
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier

# 标准化数据，保证每个维度的特征数据方差为1，均值为0。
#使得预测结果不会被某些维度过大的特征值而主导。
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

7.4LR 和 SGDC　预测

In [6]:
```
# 初始化LogisticRegression与SGDClassifier。
lr = LogisticRegression()
sgdc = SGDClassifier()

# 调用LogisticRegression中的fit函数/模块用来训练模型参数。
lr.fit(X_train, y_train)
# 使用训练好的模型lr对X_test进行预测，结果储存在变量lr_y_predict中。
lr_y_predict = lr.predict(X_test)

# 调用SGDClassifier中的fit函数/模块用来训练模型参数。
sgdc.fit(X_train, y_train)
# 使用训练好的模型sgdc对X_test进行预测，结果储存在变量sgdc_y_predict中。
sgdc_y_predict = sgdc.predict(X_test)
```

7.5LR 评分

In [7]:
```
# 从sklearn.metrics里导入classification_report模块。
from sklearn.metrics import classification_report

# 使用逻辑斯蒂回归模型自带的评分函数score获得模型在测试集上的准确性结果。
print 'Accuracy of LR Classifier:', lr.score(X_test, y_test)
# 利用classification_report模块获得LogisticRegression其他三个指标的结果。
print classification_report(y_test, lr_y_predict, target_names=['Benign', 'Malignant'])
```

```
Accuracy of LR Classifier: 0.988304093567
             precision    recall  f1-score   support

     Benign       0.99      0.99      0.99       100
  Malignant       0.99      0.99      0.99        71

avg / total       0.99      0.99      0.99       171
```

7.6SGD 评分

In [8]:
```
# 使用随机梯度下降模型自带的评分函数score获得模型在测试集上的准确性结果。
print 'Accuarcy of SGD Classifier:', sgdc.score(X_test, y_test)
# 利用classification_report模块获得SGDClassifier其他三个指标的结果。
print classification_report(y_test, sgdc_y_predict, target_names=['Benign', 'Malignant'])
```

```
Accuarcy of SGD Classifier: 0.953216374269
             precision    recall  f1-score   support

     Benign       0.93      0.99      0.96       100
  Malignant       0.98      0.90      0.94        71

avg / total       0.95      0.95      0.95       171
```

八、Python 实现 LR

Python 代码实现 LR 分类算法：

Python 实现：

```
import numpy as np
import random

# m denotes the number of examples here, not the number of features
def gradientDescent(x, y, theta, alpha, m, numIterations):
    xTrans = x.transpose()
    for i in range(0, numIterations):
        hypothesis = np.dot(x, theta)
        loss = hypothesis - y
        # avg cost per example (the 2 in 2*m doesn't really matter here.
        # But to be consistent with the gradient, I include it)
        cost = np.sum(loss ** 2) / (2 * m)
        print("Iteration %d | Cost: %f" % (i, cost))
        # avg gradient per example
        gradient = np.dot(xTrans, loss) / m
        # update
        theta = theta - alpha * gradient
    return theta



def genData(numPoints, bias, variance):
    x = np.zeros(shape=(numPoints, 2))
    y = np.zeros(shape=numPoints)
    # basically a straight line
```

```
    for i in range(0, numPoints):
        # bias feature
        x[i][0] = 1
        x[i][1] = i
        # our target variable
        y[i] = (i + bias) + random.uniform(0, 1) * variance
    return x, y

# gen 100 points with a bias of 25 and 10 variance as a bit of noise
x, y = genData(100, 25, 10)
m, n = np.shape(x)
numIterations= 100000
alpha = 0.0005
theta = np.ones(n)
theta = gradientDescent(x, y, theta, alpha, m, numIterations)
print(theta)
```

九、逻辑斯特回归 API

sklearn.linear_model.LogisticRegression 逻辑斯特回归

class sklearn.linear_model.**LogisticRegression**(*penalty='l2'*, *dual=False*, *tol=0.0001*, *C=1.0*, *fit_intercept=True*, *intercept_scaling=1*, *class_weight=None*, *random_state=None*, *solver='liblinear'*, *max_iter=100*, *multi_class='ovr'*, *verbose=0*, *warm_start=False*, *n_jobs=1*)[source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross- entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag' and 'lbfgs' solvers. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

The 'newton-cg', 'sag', and 'lbfgs' solvers support only L2 regularization with primal formulation. The 'liblinear' solver supports both L1 and L2 regularization, with a dual formulation only for the L2 penalty.

Read more in the User Guide.

Parameters:

penalty : str, ‘l1’ or ‘l2’, default: ‘l2’字符出指定了正则化策略

Used to specify the norm used in the penalization. The ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers support only l2 penalties.

New in version 0.19: l1 penalty with SAGA solver (allowing ‘multinomial’ + L1)

dual : bool, default: False 一个布尔值，如果为 true 则求解对偶形式（在 penalty=l2 且 solver=linlinear 才有对偶形式），如果为 false 求解原始形式

Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samples > n_features.

tol : float, default: 1e-4 迭代收敛与否的阈值，默认 1e-4

Tolerance for stopping criteria.

C : float, default: 1.0 一个浮点数，指定了罚项系数的倒数。值越小正则化项越大

Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

fit_intercept : bool, default: True 指定是否需要 b 的值，为 false 不会计算 b 的值

Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function.

intercept_scaling : float, default 1.只有当 solver=liblinear 才有意义。当采用 fit_intercept 时，相当于人造一个特征出来，该特征恒为 1，其权重为 b。在计算正则化项的时候，改人造特征也被考虑了。因此为了降低人造特征的影响，需要提供 **intercept_scaling**

Useful only when the solver ‘liblinear’ is used and self.fit_intercept is set to True. In this case, x becomes [x, self.intercept_scaling], i.e. a “synthetic” feature with constant value equal to intercept_scaling is appended to the instance vector. The intercept becomes intercept_scaling * synthetic_feature_weight.

Note! the synthetic feature weight is subject to l1/l2 regularization as all other features. To lessen the effect of regularization on synthetic feature weight (and therefore on the intercept) intercept_scaling has to be increased.

class_weight : dict or 'balanced', default: None 权重

Balance 每个分类的权重与该分类在样本集中出现频率成反比。为 None 则每个权重都为 1，为字典，分别给出每个分类的权重

Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)).

Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified.

New in version 0.17: class_weight='balanced'

random_state : int, RandomState instance or None, optional, default: None

The seed of the pseudo random number generator to use when shuffling the data. If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random. Used when solver == 'sag' or 'liblinear'.

solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}适用的最优化问题的算法

newton-cg 牛顿法

Lbfgs 使用 L-BFG 拟牛顿法

拟牛顿法和牛顿法的区别就在于拟牛顿法求解 Hession 矩阵的时候采用了近似求解 Hession 矩阵的方式，而不是像牛顿法的精确求解方式。

Linlinear：使用 liblinear

Sag：使用随机平均梯度下降法

对于规模较小的数据集，liblinear 比较适用，大规模数据集 sag 适用

Newton-cg、lbfgs、sag 只处理 penalty=l2 的情况。

default: ‘liblinear’ Algorithm to use in the optimization problem.

-

For small datasets, ‘liblinear’ is a good choice, whereas ‘sag’ and

-

‘saga’ are faster for large ones.

-
-

For multiclass problems, only ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’

-

handle multinomial loss; ‘liblinear’ is limited to one-versus-rest schemes.

-
-

‘newton-cg’, ‘lbfgs’ and ‘sag’ only handle L2 penalty, whereas

-

‘liblinear’ and ‘saga’ handle L1 penalty.

-

Note that ‘sag’ and ‘saga’ fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.

New in version 0.17: Stochastic Average Gradient descent solver.

New in version 0.19: SAGA solver.

max_iter : int, default: 100　最大迭代次数

Useful only for the newton-cg, sag and lbfgs solvers. Maximum number of iterations taken for the solvers to converge.

multi_class : str, {'ovr', 'multinomial'}, default: 'ovr'

指定多分类问题策略

Ovr-one-vs-rest 策略

Multinomial 是直接采用多分类逻辑回归策略

Multiclass option can be either 'ovr' or 'multinomial'. If the option chosen is 'ovr', then a binary problem is fit for each label. Else the loss minimised is the multinomial loss fit across the entire probability distribution. Does not work for liblinear solver.

New in version 0.18: Stochastic Average Gradient descent solver for 'multinomial' case.

verbose : int, default: 0

For the liblinear and lbfgs solvers set verbose to any positive number for verbosity.

warm_start : bool, default: False 为 true 那么使用前一次的结果继续训练，否则从头开始。

When set to True, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution. Useless for liblinear solver.

New in version 0.17: warm_start to support *lbfgs*, *newton-cg*, *sag*, *saga* solvers.

n_jobs : int, default: 1

Number of CPU cores used when parallelizing over classes if multi_class='ovr'". This parameter is ignored when the ``solver``is set to 'liblinear' regardless of whether 'multi_class' is specified or not. If given a value of -1, all cores are used.

Attributes:

coef_ : 权重向量 array, shape (1, n_features) or (n_classes, n_features)

Coefficient of the features in the decision function.

coef_ is of shape (1, n_features) when the given problem is binary.

intercept_ :b 的值 array, shape (1,) or (n_classes,)

Intercept (a.k.a. bias) added to the decision function.

If fit_intercept is set to False, the intercept is set to zero. intercept_ is of shape(1,) when the problem is binary.

n_iter_ : 实际迭代次数 array, shape (n_classes,) or (1,)

Actual number of iterations for all classes. If binary or multinomial, it returns only 1 element. For liblinear solver, only the maximum number of iteration across all classes is given.

Methods

decision_function(X)	Predict confidence scores for samples.
densify()	Convert coefficient matrix to dense array format.
fit(X, y[, sample_weight])	Fit the model according to the given training data.训练模型
get_params([deep])	Get parameters for this estimator.
predict(X)	Predict class labels for samples in X.
predict_log_proba(X)	Log of probability estimates.返回一个数组，数组的元素依次是 x 预测为各类概率的对数值
predict_proba(X)	Probability estimates.返回一个数组，数组元素依次是 X 预测为哥哥类比饿的概率值
score(X, y[, sample_weight])	Returns the mean accuracy on the given test data and labels.返回在 x 和 y 上的准确率
set_params(**params)	Set the parameters of this estimator.
sparsify()	Convert coefficient matrix to sparse format.