**logistic回归**

**简介：**

假设现在有一些数据点，我们利用一条直线对这些点进行拟合(该线称为最佳拟合直线)，这个拟合过程就称作为回归。

logistic回归是一种广义线性回归，与多重线性回归分析有很多相同之处。

相同：模型形式基本上相同，都具有 w‘x+b，其中w和b是待求参数，

不同：多重线性回归直接将w‘x+b作为因变量，即y =w‘x+b；

logistic回归则通过函数L将w‘x+b对应一个p，p =L(w‘x+b),然后根据p 与1-p的大小决定因变量的值。 

logistic回归的因变量可以是二分类的，也可以是多分类的，但是二分类的更为常用。

Logistic回归模型的适用条件：

1 ）因变量为二分类的分类变量或某事件的发生率，并且是数值型变量。

2 ）因变量要服从二项分布。

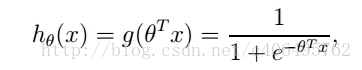
3 ）自变量和Logistic概率是线性关系

4 ）各观测对象间相互独立。 

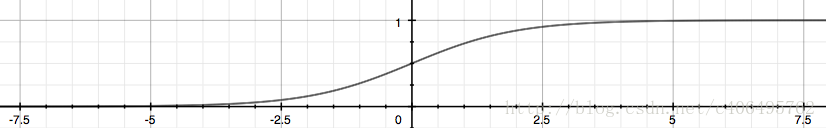
Logistic回归进行分类的主要思想是：根据现有数据对分类边界线建立回归公式，以此进行分类。其实，Logistic本质上是一个基于条件概率的判别模型。

我们首先了解一下函数：

Sigmoid函数 ：



Sigmoid函数的图形：



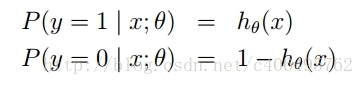
θ是参数列向量(要求解的)，x是样本列向量(给定的数据集)。

g(z)函数实现了任意实数到[0,1]的映射，这样我们的数据集，不管是大于1或者小于0，都可以映射到[0,1]区间进行分类。

hθ(x)给出了输出为1的概率。比如当hθ(x)=0.7，那么说明有70%的概率输出为1。输出为0的概率是输出为1的补集，也就是30%。

**如何得到合适的参数向量θ?**

根据函数，做出如下的假设：



在已知样本x和参数θ的情况下，样本x属性正样本(y=1)和负样本(y=0)的条件概率。

把上述两个概率公式合二为一：

IMG_256

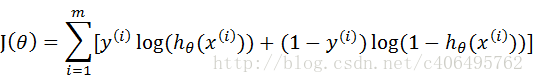
合并出来的Cost，称为代价函数。当y等于1时，(1-y)项(第二项)为0；当y等于0时，y项(第一项)为0。

对整个表达式求对数：

IMG_257

最大似然：

假定样本与样本之间相互独立，那么整个样本集生成的概率即为所有样本生成概率的乘积，再将公式对数化，得到如下公式：



m为样本的总数，y(i)表示第i个样本的类别，x(i)表示第i个样本，θ是多维向量，x(i)也是多维向量。

需要求解：满足J(θ)的最大的θ值

## **梯度上升算法**

求解的最大值，所以用梯度上升算法

那么如何求最大值，用一个例子来演示

EG：

IMG_256

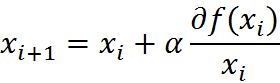
求极值，先求函数的导数：

IMG_258

令导数为0，可求出x=2即取得函数f(x)的极大值。极大值等于f(2)=4

在平时中，用迭代的方法来做。这种寻找最佳拟合参数的方法，就是最优化算法。

爬坡的数学公式表达：

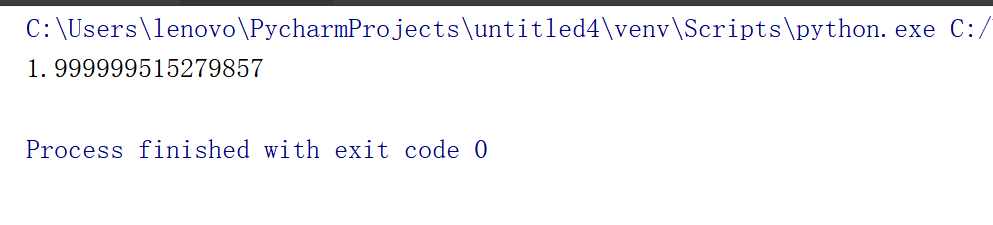


α为步长，也就是学习速率，控制更新的幅度。直到求出的x为函数极大值的近似值，停止迭代。

**Logistic.py**

**梯度上升算法的测试代码  
用上面的例子测试---f(x) = -x^2 + 4x**

结果输出如下：

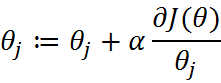
****

自己计算，极值为2

结果已经非常接近极值了。这一过程，就是梯度上升算法。

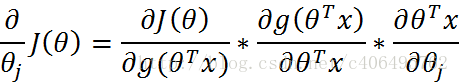
那么同理，J(θ)这个函数的极值，也可以这么求解。

公式如下：

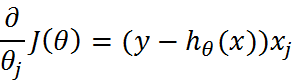


那只要求出J(θ)的偏导，然后利用梯度上升算法，求解J(θ)的极大值了。

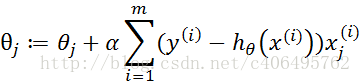
那么现在开始求解J(θ)对θ的偏导，求解如下(数学推导)：



结果为：



因此，梯度上升迭代公式为：



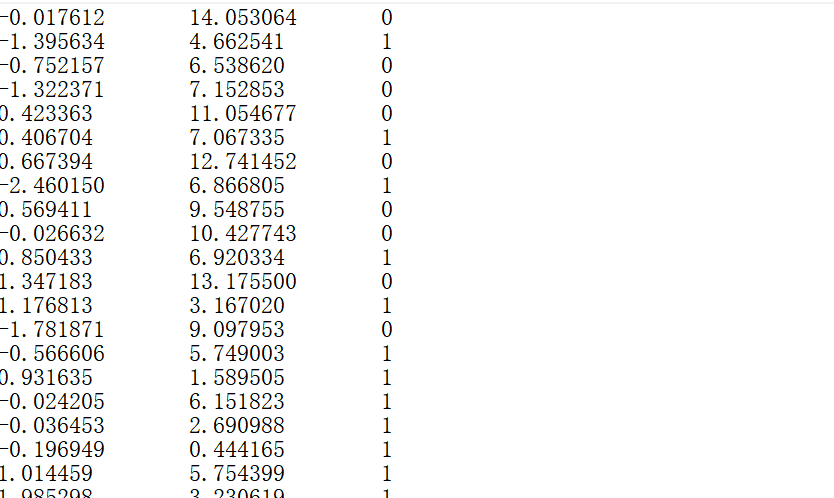
下面就可以编写代码，计算最佳拟合参数了。

先看书籍官网给出的数据集：

这个数据有两个特征，因此可以将数据在一个二维平面上展示出来。

将第一列数据看作x轴上的值，第二列数据看作y轴上的值。而最后一列数据即为分类标签。根据标签的不同，对这些点进行分类。

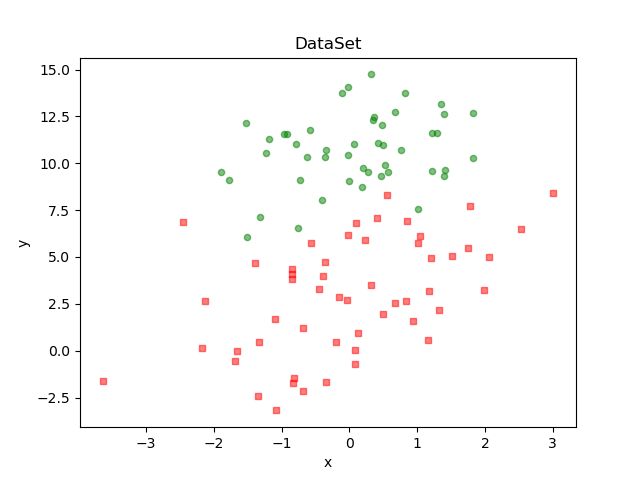
数据集如下所示：



**Logistic-1.py**

数据集这种方式，无法直观的看清楚，下面用图片的形式展示出来。

代码输出如下：



上图就可以清晰的看出数据的分布情况。

假设Sigmoid函数的输入记为z，那么z=w0x0 + w1x1 + w2x2，即可将数据分割开。

其中，x0为全是1的向量，x1为数据集的第一列数据，x2为数据集的第二列数据。这个方程未知的参数为w0，w1，w2，也就是需要求的回归系数(最优参数)。

**Logistic-2.py**

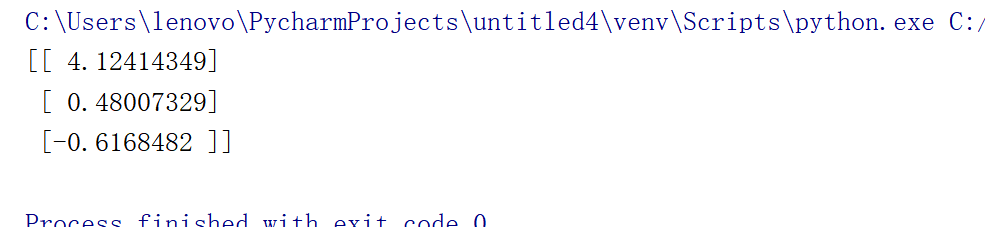
## **训练算法**

将梯度上升迭代公式矢量化：

IMG_257

根据矢量化的公式，编写代码。

代码输出如下：



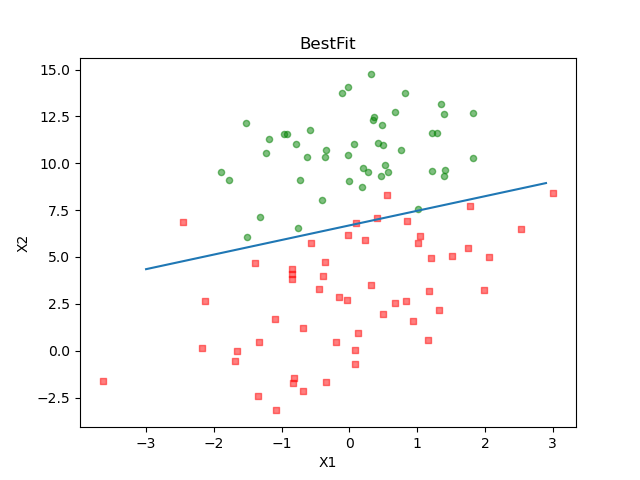
上面就是我们求解出e 回归系数[w0,w1,w2]

通过求解出的参数，可以确定不同类别数据之间的分隔线，画出决策边界。

**Logistic-3.py**

**绘制决策边界**

现在开始绘制这个分隔线：



从上图可以看出，只分错了几个点而已,正确率还可以。

# **总结**

Logistic回归的一般过程：

收集数据--准备数据（数值型）--分析数据--训练算法--测试算法--使用算法

其他：

Logistic回归的目的是寻找一个非线性函数Sigmoid的最佳拟合参数，求解过程可以由最优化算法完成。

# **改进的随机梯度上升算法**

梯度上升算法在每次更新回归系数(最优参数)时，都需要遍历整个数据集。

举个例子，假设，使用的数据集一共有100个样本。那么，dataMatrix就是一个100\*3的矩阵。每次计算h的时候，都要计算dataMatrix\*weights这个矩阵乘法运算，要进行100\*3次乘法运算和100\*2次加法运算。

因此，需要对算法进行改进，减少计算量，有一种方法叫做随机梯度上升算法。

**Logistic-4.py**

## **随机梯度上升算法**

第一个改动：alpha在每次迭代的时候都会调整，会随着迭代次数不断减小，但永远不会减小到0。

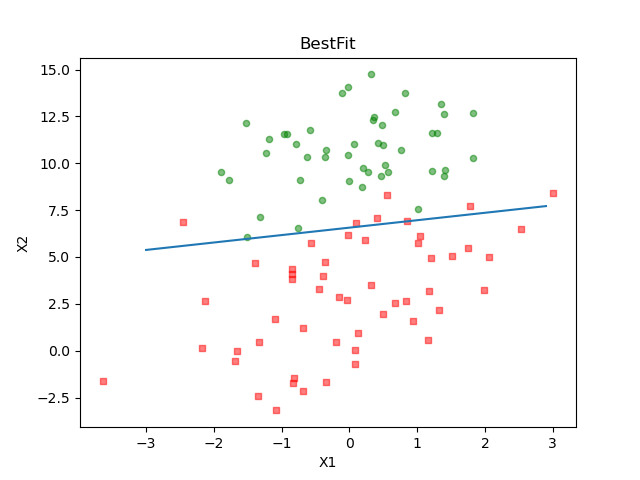
原因是为了保证在多次迭代之后新数据仍然具有一定的影响。

在降低alpha的函数中，alpha每次减少1/(j+i)，其中j是迭代次数，i是样本点的下标。

第二个改动：更新回归系数(最优参数)时，只使用一个样本点，并且选择的样本点是随机的，每次迭代不使用已经用过的样本点。

编写代码，使用改进的随机梯度上升算法分类，查看效果如何：

代码输出如下：

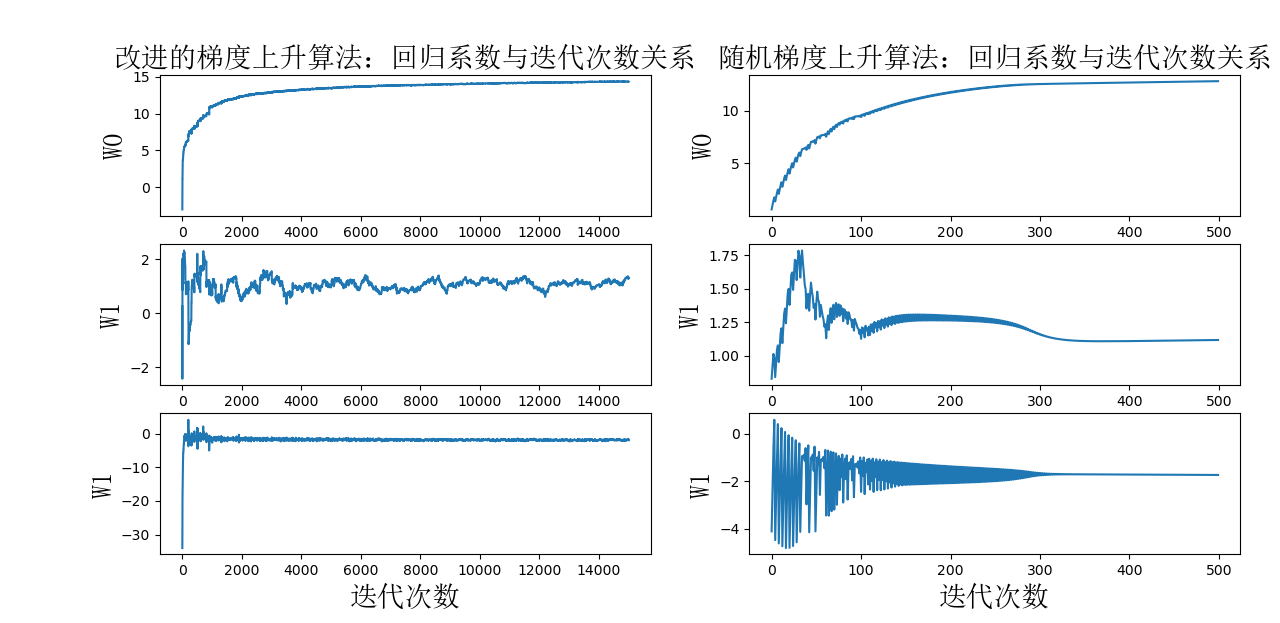


**Logistic-5.py**

## **回归系数与迭代次数的关系**

分类效果也是不错的。

不过，从这个分类结果中，看不出迭代次数和回归系数的关系，也不能直观的看到每个回归方法的收敛情况。因此，编写程序，绘制出回归系数和迭代次数的关系曲线：



根据图像所示，改进的随机梯度上升算法收敛效果更好。

让我们分析一下：

总有100个样本，改进的随机梯度上升算法迭代次数为150。

左边的图上显示15000次迭代次数的原因：使用一次样本就更新一下回归系数。因此，迭代150次，相当于更新回归系数150\*100=15000次。相当于，迭代150次，更新1.5万次回归参数。

改进的随机梯度上升算法：在遍历整个数据集20次的时候，回归系数已经收敛，训练已完成。

梯度上升算法：每次更新回归系数都要遍历整个数据集。当迭代次数为300多次的时候，回归系数才收敛。

**具体例子**

**从疝气病症状预测病马的死亡率**

根据书本官网给出的数据，数据包含了368个样本和28个特征。

该数据集中包含了医院检测马疝病的一些指标，该数据还存在一个问题，数据集中有30%的值是缺失的。

**准备数据**

一些可选的做法：

1）使用可用特征的均值来填补缺失值

2）使用特殊值来填补缺失值，如-1

3）忽略有缺失值的样本

4）使用相似样本的均值添补缺失值

5）使用另外的机器学习算法预测缺失值

预处理数据：

如果测试集中一条数据的特征值已经确实，那么我们选择实数0来替换所有缺失值，因为本文使用Logistic回归。因此这样做不会影响回归系数的值。sigmoid(0)=0.5，即它对结果的预测不具有任何倾向性。

如果测试集中一条数据的类别标签已经缺失，那么我们将该类别数据丢弃，因为类别标签与特征不同，很难确定采用某个合适的值来替换。

原始的数据集经过处理，保存为两个文件：horseColicTest.txt和horseColicTraining.txt。

有了上面的数据集，我们就可以利用该分类器来预测病马的生死问题了。

**Logistic-6.py**

## **使用Python构建Logistic回归分类器**

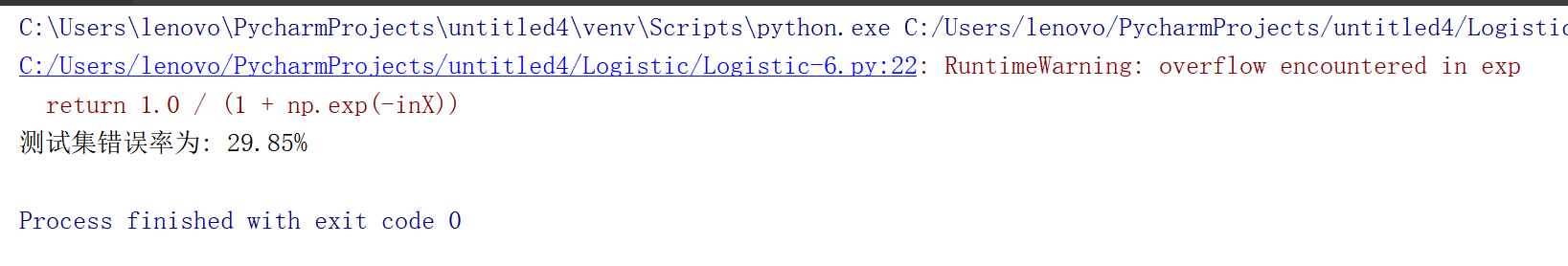
**随机梯度上升算法**

我们先用自己写的改进的随机梯度上升算法进行预测。

使用Logistic回归方法进行分类，把测试集上每个特征向量乘以最优化方法得来的回归系数，再将乘积结果求和，最后输入到Sigmoid函数中即可。

如果对应的Sigmoid值大于0.5就预测类别标签为1，否则为0。

代码输出如下：



错误率还是蛮高的，而且每次运行的错误率也是不同的。

首先，因为数据集本身缺失。

其次，使用的是改进的随机梯度上升算法，因为数据集本身就很小，就几百的数据量。用改进的随机梯度上升算法显然不合适。

Error:运行时出现了警告，所以百度了一下。

找到了一个优化的方法

因为参数值inx很大时，exp(inx)可能会发生溢出。

def sigmoid(in\_x): # RuntimeWarning: overflow encountered in exp

return 1.0/(1 + np.exp(-in\_x))

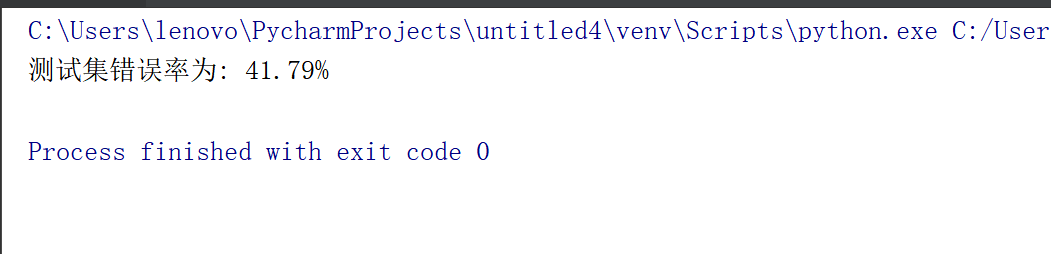
优化方法 ：

改为

if in\_x >= 0: return 1.0/(1+np.exp(-in\_x))

else: return np.exp(in\_x)/(1+np.exp(in\_x))

改完以后，代码输出如下：



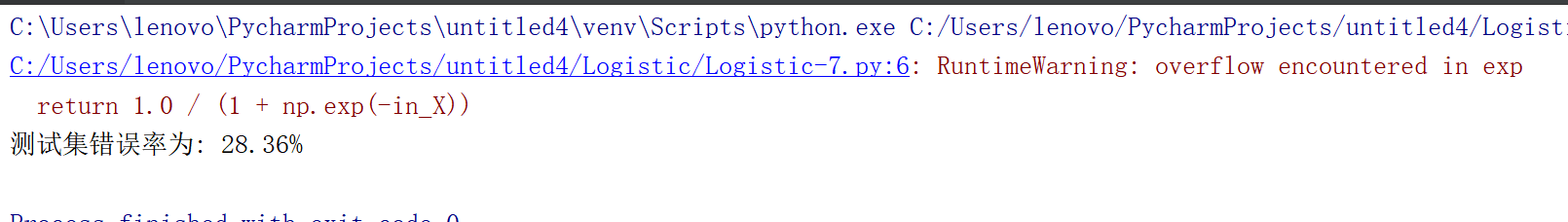
果然没有了警告，而且错误率每次都不一样。

用改进的随机梯度上升算法显然不合适，我们换一下其他的梯度上升算法。

**Logistic-7.py**

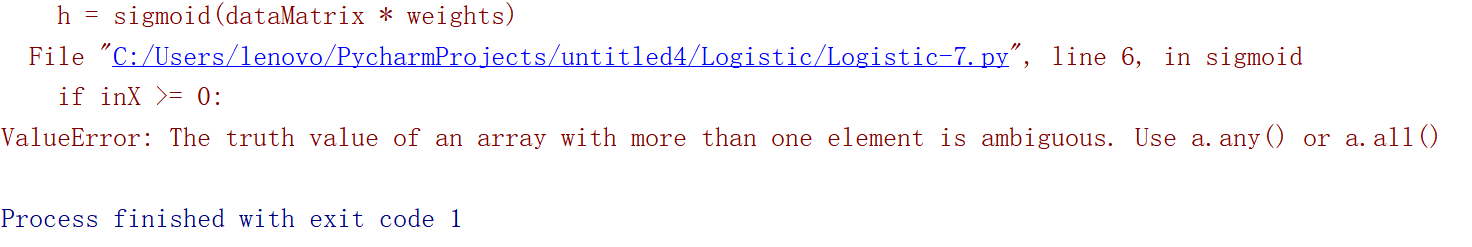
**梯度上升算法**

代码输出如下：

****

发现错误率确实下降了，但是代码警告了。

按照了上面的方式改动，发现报错了！！

****

百度了一下原因：

举个例子假设存在两个nparray:

a = np.array([1,2,3]) b = np.array([1,3,2])

print a==b

结果为:[True False False]

是对每个应用位置元素进行比较,源代码相当于:if [True False False]:

逻辑上不正确。

按照报错的提示,要使用any(), 或者 all(),

输入:

print any(a==b) print all(a==b)

对应结果为:

True False

但是改动以后，还是报错，所以继续百度，又发现了一个解决办法，

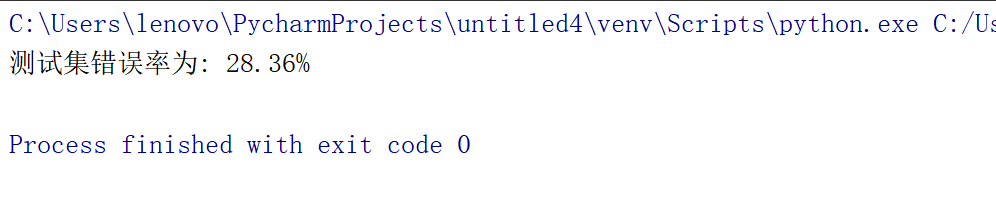
把代码改成了

def logistic\_function(x):

return .5 \* (1 + np.tanh(.5 \* x))

此版本的功能稳定,快速且相当准确

最后结果输出如下：

****

正常输出，没有报错了，而且错误率比较低。

结论：

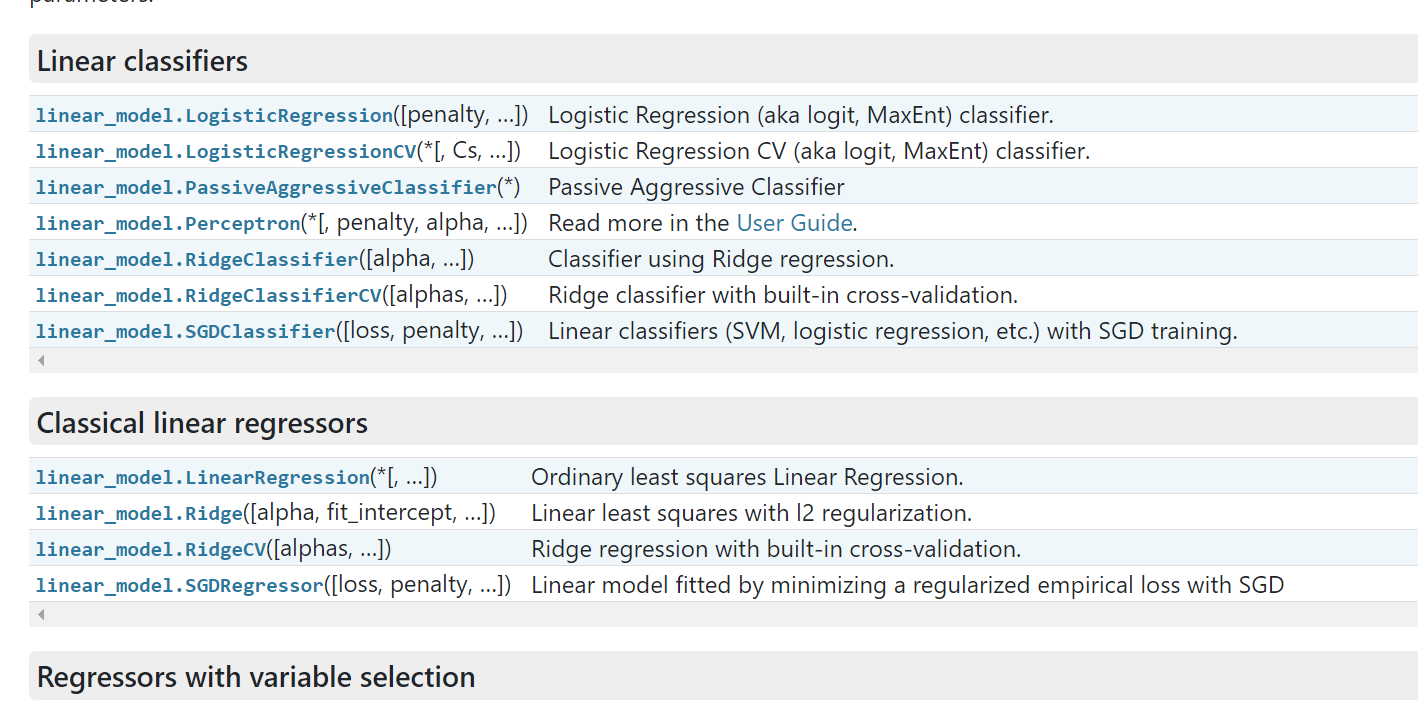
当数据集较小时，我们使用梯度上升算法

当数据集较大时，我们使用改进的随机梯度上升算法

对应的，在Sklearn中，我们就可以根据数据情况选择优化算法，比如数据较小的时候，我们使用liblinear，数据较大时，我们使用sag和saga。

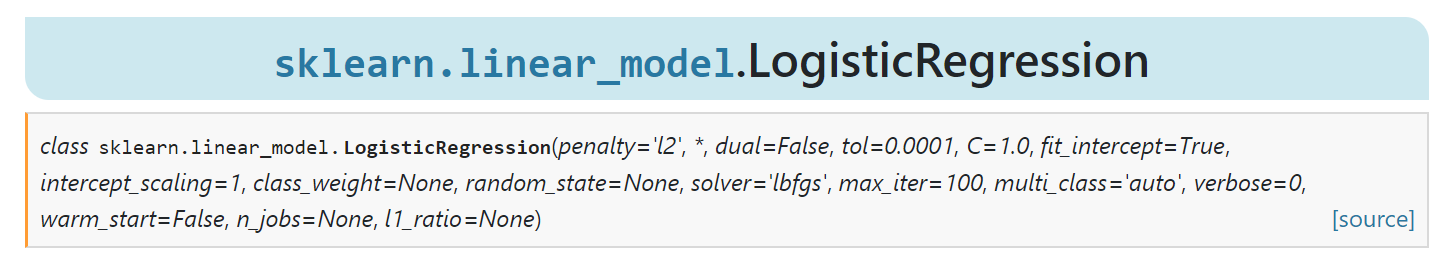
**使用Sklearn构建Logistic回归分类器**

sklearn.linear\_model模块提供了很多模型，比如Logistic回归、Lasso回归、贝叶斯脊回归等，这次使用LogisticRegressioin。



## **LogisticRegression**

LogisticRegression函数，一共有14个参数：



参数说明如下：

**penalty**：惩罚项，str类型，可选参数为l1和l2，默认为l2。

加约束的情况下，理论上应该可以获得泛化能力更强的结果。

**dual**：对偶或原始方法，bool类型，默认为False。

对偶方法只用在求解线性多核(liblinear)的L2惩罚项上。当样本数量>样本特征的时候，dual通常设置为False。

**tol**：停止求解的标准，float类型，默认为1e-4。

就是求解到多少的时候，停止，认为已经求出最优解。

**c**：正则化系数λ的倒数，float类型，默认为1.0。

必须是正浮点型数。像SVM一样，越小的数值表示越强的正则化。

**fit\_intercept**：是否存在截距或偏差，bool类型，默认为True。

**intercept\_scaling：**仅在正则化项为”liblinear”，且fit\_intercept设置为True时有用。float类型，默认为1。

**class\_weight**：用于标示分类模型中各种类型的权重，可以是一个字典或者balanced字符串，默认为不输入，也就是不考虑权重，即为None。

如果选择输入的话，可以选择balanced，让类库自己计算类型权重，或者自己输入各个类型的权重。

如果class\_weight选择balanced，那么类库会根据训练样本量来计算权重。某种类型样本量越多，则权重越低，样本量越少，则权重越高。

当class\_weight为balanced时，类权重计算方法如下：n\_samples / (n\_classes \* np.bincount(y))。n\_samples为样本数，n\_classes为类别数量，np.bincount(y)会输出每个类的样本数，例如y=[1,0,0,1,1],则np.bincount(y)=[2,3]。

那么class\_weight有什么作用呢？

在分类模型中，我们经常会遇到两类问题：

第一种是误分类的代价很高。

第二种是样本是高度失衡的。

**random\_state**：随机数种子，int类型，可选参数，默认为无，仅在正则化优化算法为sag,liblinear时有用。

**solver**：优化算法选择参数，只有五个可选参数，即newton-cg,lbfgs,liblinear,sag,saga。默认为liblinear。

solver参数决定了我们对逻辑回归损失函数的优化方法，有四种算法可以选择，分别是：

liblinear：使用了开源的liblinear库实现，内部使用了坐标轴下降法来迭代优化损失函数。

lbfgs：拟牛顿法的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。

newton-cg：也是牛顿法家族的一种，利用损失函数二阶导数矩阵即海森矩阵来迭代优化损失函数。

sag：即随机平均梯度下降，是梯度下降法的变种，和普通梯度下降法的区别是每次迭代仅仅用一部分的样本来计算梯度，适合于样本数据多的时候。

saga：线性收敛的随机优化算法的的变重。

总结：

liblinear适用于小数据集，而sag和saga适用于大数据集因为速度更快。

对于多分类问题，只有newton-cg,sag,saga和lbfgs能够处理多项损失。

newton-cg,sag和lbfgs这三种优化算法时都需要损失函数的一阶或者二阶连续导数，因此不能用于没有连续导数的L1正则化，只能用于L2正则化。而liblinear和sagaL1正则化和L2正则化都可以。

sag每次仅仅使用了部分样本进行梯度迭代，所以当样本量少的时候不要选择它。但是sag不能用于L1正则化。

逻辑回归有二元逻辑回归和多元逻辑回归。对于多元逻辑回归常见的有one-vs-rest(OvR)和many-vs-many(MvM)两种。而MvM一般比OvR分类相对准确一些。liblinear只支持OvR，不支持MvM，这样如果需要相对精确的多元逻辑回归时，就不能选择liblinear了。

**max\_iter：**算法收敛最大迭代次数，int类型，默认为10。仅在正则化优化算法为newton-cg, sag和lbfgs才有用。

**multi\_class：**分类方式选择参数，str类型，可选参数为ovr和multinomial，默认为ovr。ovr即前面提到的one-vs-rest(OvR)，而multinomial即前面提到的many-vs-many(MvM)。如果是二元逻辑回归，ovr和multinomial并没有任何区别，区别主要在多元逻辑回归上。

OvR和MvM有什么不同？

OvR的思想很简单，无论你是多少元逻辑回归，我们都可以看做二元逻辑回归。具体做法是，对于第K类的分类决策，我们把所有第K类的样本作为正例，除了第K类样本以外的所有样本都作为负例，然后在上面做二元逻辑回归，得到第K类的分类模型。其他类的分类模型获得以此类推。

而MvM则相对复杂，这里举MvM的特例one-vs-one(OvO)作讲解。如果模型有T类，我们每次在所有的T类样本里面选择两类样本出来，不妨记为T1类和T2类，把所有的输出为T1和T2的样本放在一起，把T1作为正例，T2作为负例，进行二元逻辑回归，得到模型参数。我们一共需要T(T-1)/2次分类。

可以看出OvR相对简单，但分类效果相对略差（这里指大多数样本分布情况，某些样本分布下OvR可能更好）。而MvM分类相对精确，但是分类速度没有OvR快。如果选择了ovr，则4种损失函数的优化方法liblinear，newton-cg,lbfgs和sag都可以选择。但是如果选择了multinomial,则只能选择newton-cg, lbfgs和sag了。

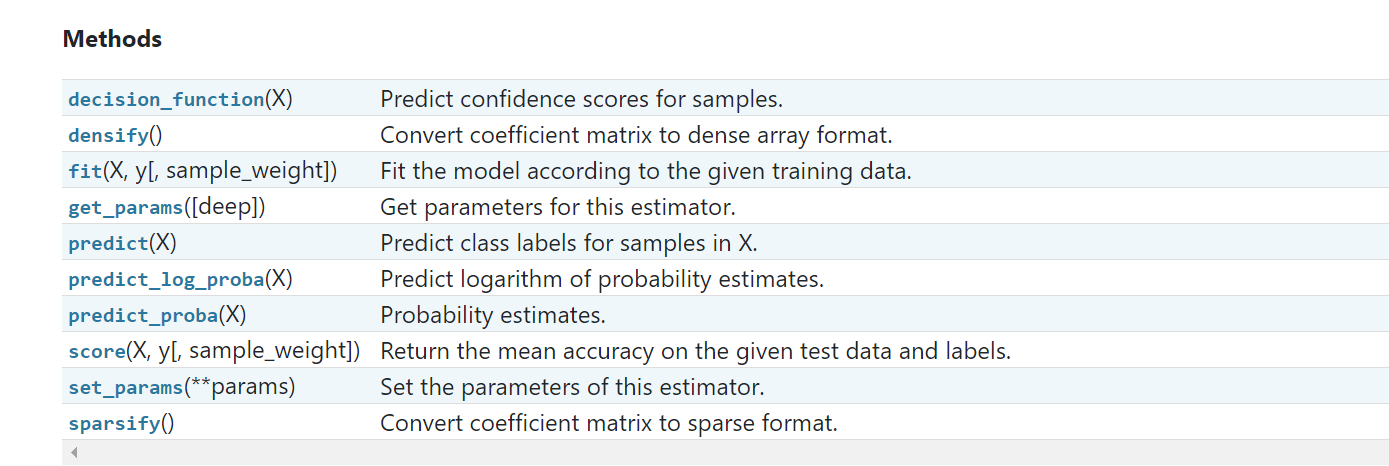
**verbose**：日志冗长度，int类型。默认为0。就是不输出训练过程，1的时候偶尔输出结果，大于1，对于每个子模型都输出。

**warm\_start：**热启动参数，bool类型。默认为False。如果为True，则下一次训练是以追加树的形式进行。

**n\_jobs：**并行数。int类型，默认为1。

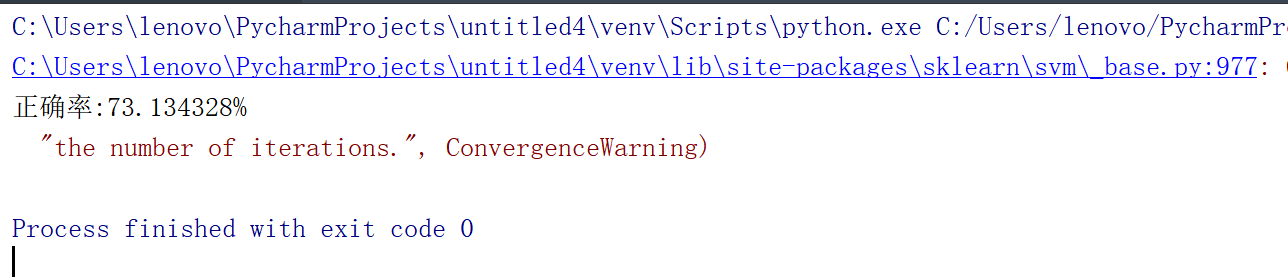
1的时候，用CPU的一个内核运行程序，2的时候，用CPU的2个内核运行程序。为-1的时候，用所有CPU的内核运行程序。

除此之外，LogisticRegression也有一些方法供可以使用



## **Logistic-8.py**

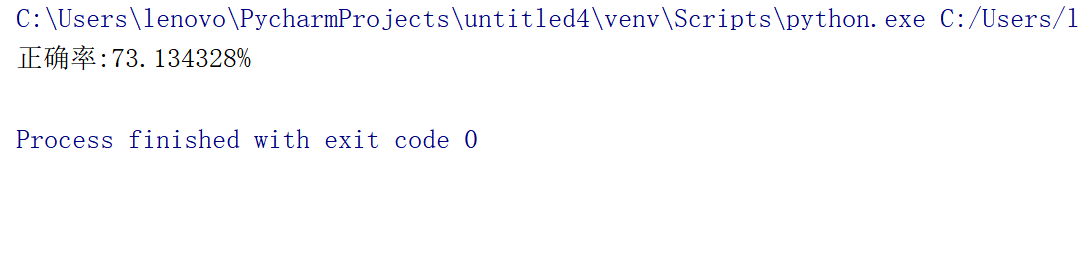
代码输出如下：



发现报错了：

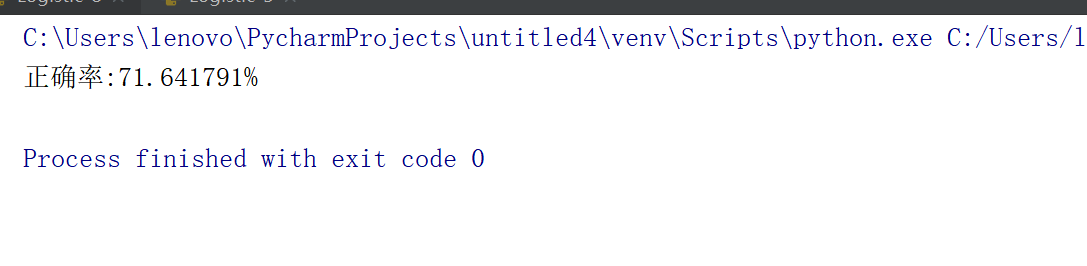
ConvergenceWarning: Liblinear failed to converge, increase the number of iterations. "the number of iterations.", ConvergenceWarning)

警告是因为算法还没有收敛。更改max\_iter一下，再运行代码：



还可以更改solver参数，比如设置为lbfgs，看一看效果

代码输出如下：



**总结**

**Logistic回归的优缺点**

优点：

实现简单，易于理解和实现；计算代价不高，速度很快，存储资源低。

缺点：

容易欠拟合，分类精度可能不高。

Logistic回归的目的是寻找一个非线性函数Sigmoid的最佳拟合参数，求解过程可以由最优化算法完成。

改进的一些最优化算法，比如sag。它可以在新数据到来时就完成参数更新，而不需要重新读取整个数据集来进行批量处理。

机器学习的一个重要问题就是如何处理缺失数据。