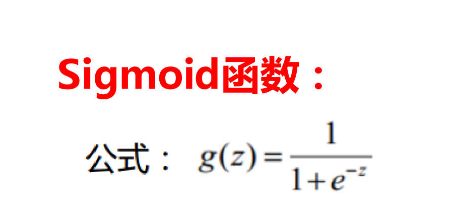
**对于对率回归的一些心得**

对率回归也称逻辑回归，是用于处理因变量为分类变量的回归问题，常见的是二分类问题，也可以处理多分类问题，它实际上是属于一种分类方法。

首先，对率函数为



然后将线性回归函数WTX + B代入上述函数，结果映射到[0,1]的区间上，于是我们得到：

Chart, box and whisker chart

Description automatically generated

于是，我们可以得到一个连续的结果，：

Chart

Description automatically generated

令p1(ˆxi; β)的值为y 为正列的概率，p0(ˆxi; β)为y 为反列的概率，并且设置β 为w 和 b 的向量， 可以得出：

Text, letter

Description automatically generated

通过最大似然估计，梯度下降等原理，最后求偏导可以得到一个较优的最小β值（不太可能得到最优值），其中过程过于复杂，因此不再赘述

打开spyder,代码如下：

import pandas as pd  
import numpy as np  
​  
#获取文件  
df = pd.read\_csv(r'C:\Users\14272\Desktop\竞赛\借贷风险预测\loan\_risk\_forecast\train.csv',encoding='utf-8',engine='python')  
print(df.columns)  
​  
​  
"""  
切分数据集  
拟定自变量与因变量  
"""  
X\_whole = df.iloc[:,1:21]  
X\_whole.info()  
y\_whole = df[['y']]  
​  
"""  
写入逻辑回归模型  
"""  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.linear\_model import LogisticRegression  
import time  
#交叉验证选择较优惩罚因子  
scores = []  
c\_param\_range = [0.01,0.1,1,10,100]  
z = 1  
for i in c\_param\_range:  
   start\_time = time.time()  
   ir = LogisticRegression(C=i,penalty='l2',solver='lbfgs',class\_weight='balanced')  
   score = cross\_val\_score(ir,X\_whole,y\_whole,cv=10,scoring='recall')  
   score\_mean = sum(score)/len(score)  
   scores.append(score\_mean)  
   end\_time = time.time()  
   print("第{}次...".format(z))  
   print("i值为{}".format(i))  
   print("time spend:{:.2f}".format(end\_time - start\_time))  
   print("recall值为:{}".format(score\_mean))  
   z +=1  
​  
best\_c = c\_param\_range[np.argmax(scores)]  
print()  
print('最优惩罚因子为:{}'.format(best\_c))  
​  
​  
ir = LogisticRegression(C=best\_c,penalty='l2',solver='lbfgs',class\_weight='balanced')  
ir.fit(X\_whole,y\_whole)  
train\_predicted\_1 = ir.predict(X\_whole)

LogisticRegression参数如下：

**【版本为\***\*0.2.1】\*\*

逻辑回归参数详解：

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)

Penalty：正则化方式，有l1和l2两种。用于指定惩罚项中使用的规范。newton-cg、sag和lbfgs求解算法只支持L2规范。L1G规范假设的是模型的参数满足拉普拉斯分布，L2假设的模型参数满足高斯分布，所谓的范式就是加上对参数的约束，使得模型更不会过拟合(overfit)，但是如果要说是不是加了约束就会好，这个没有人能回答，只能说，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：按默认即可。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：float，默认值：1e-4，容许停止标准，即我们说的要迭代停止所需达到的精度要求。

C：正则化强度。为浮点型数据。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：指定是否应该将常量(即偏差或截距)添加到决策函数中。相当于是否加入截距项b。默认加入。

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None，也就是不考虑权重，在出现样本不平衡时，可以考虑调整class\_weight系数去调整，防止算法对训练样本多的类别偏倚。

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。举个例子，比如对于0,1的二元模型，我们可以定义class\_weight={0:0.9,1:0.1}，这样类型0的权重为90%，而类型1的权重为10%。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。比如对合法用户和非法用户进行分类，将非法用户分类为合法用户的代价很高，我们宁愿将合法用户分类为非法用户，这时可以人工再甄别，但是却不愿将非法用户分类为合法用户。这时，我们可以适当提高非法用户的权重。

第二种是样本是高度失衡的，比如我们有合法用户和非法用户的二元样本数据10000条，里面合法用户有9995条，非法用户只有5条，如果我们不考虑权重，则我们可以将所有的测试集都预测为合法用户，这样预测准确率理论上有99.95%，但是却没有任何意义。这时，我们可以选择balanced，让类库自动提高非法用户样本的权重。提高了某种分类的权重，相比不考虑权重，会有更多的样本分类划分到高权重的类别，从而可以解决上面两类问题。

random\_state：伪随机数产生器在对数据进行洗牌时使用的种子。仅在正则化优化算法为sag,liblinear时有用。

**Solver**：{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}，优化拟合参数算法选择，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**坐标轴下降法**来迭代优化损失函数。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。

**newton-cg**：牛顿法，sag方法使用一阶导数，而牛顿法采用了二阶泰勒展开，这样缩减了迭代轮数，但是 需要计算Hsssian矩阵的逆，所以计算复杂度较高。【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。】

**Lbfgs**：拟牛顿法，考虑到牛顿法的Hessian矩阵求逆太过复杂，尤其在高维问题中几乎不可行，想到了用较低的代价寻找Hessian矩阵的近似逆矩阵，便有了拟牛顿法。【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。】

**Sag**：即随机平均梯度下降，类似于我们的stocGradAscent1函数，思想是常用的一阶优化方法，是求解无约束优化问题最经典，最简单的方法之一。【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。【线性收敛的随机优化算法的的变种。】

**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，而liblinear受限于一对剩余(OvR)。啥意思，就是用liblinear的时候，如果是多分类问题，得先把一种类别作为一个类别，剩余的所有类别作为另外一个类别。一次类推，遍历所有类别，进行分类。

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saga通吃L1正则化和L2正则化。

同时，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，而如果样本量非常大，比如大于10万，sag是第一选择。但是sag不能用于L1正则化，所以当你有大量的样本，又需要L1正则化的话就要自己做取舍了。要么通过对样本采样来降低样本量，要么回到L2正则化。

从上面的描述，大家可能觉得，既然newton-cg, lbfgs和sag这么多限制，如果不是大样本，我们选择liblinear不就行了嘛！错，因为liblinear也有自己的弱点！我们知道，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。郁闷的是liblinear只支持OvR，不支持MvM，这样如果我们需要相对精确的多元逻辑回归时，就不能选择liblinear了。也意味着如果我们需要相对精确的多元逻辑回归不能使用L1正则化了。

**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，算法收敛的最大迭代次数。

**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（重新使用上一次的调用作为初始化）。

**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。

**总结：**

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。

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。这个问题没有标准答案，取决于实际应用中的需求。现有一些解决方案，每种方案都各有优缺点。

我们需要根据数据的情况，这是Sklearn的参数，以期达到更好的分类效果。

**Sklearn\***\*中损失函数的优化方法，不是梯度下降法。\*\*

**Liblinear\***\*：坐标轴下降法\*\*