*class* sklearn.linear_model.**LogisticRegressionCV**(*Cs=10*, *fit_intercept=True*, *cv=None*, *dual=False*, *penalty='l2'*, *scoring=None*, *solver='lbfgs'*, *tol=0.0001*, *max_iter=100*, *class_weight=None*, *n_jobs=1*, *verbose=0*, *refit=True*, *intercept_scaling=1.0*, *multi_class='ovr'*, *random_state=None*) [source]

# 1. 概述

在scikit-learn中，与逻辑回归有关的主要是这3个类。LogisticRegression，LogisticRegressionCV和logistic_regression_path。其中LogisticRegression和LogisticRegressionCV的主要区别是LogisticRegressionCV使用了交叉验证来选择正则化系数C。而LogisticRegression需要自己每次指定一个正则化系数。除了交叉验证，以及选择正则化系数C以外， LogisticRegression和LogisticRegressionCV的使用方法基本相同。

logistic_regression_path类则比较特殊，它拟合数据后，不能直接来做预测，只能为拟合数据选择合适逻辑回归的系数和正则化系数。主要是用在模型选择的时候。一般情况用不到这个类，所以后面不再讲述logistic_regression_path类。

此外，scikit-learn里面有个容易让人误解的类RandomizedLogisticRegression，虽然名字里有逻辑回归的词，但是主要是用L1正则化的逻辑回归来做特征选择的，属于维度规约的算法类，不属于我们常说的分类算法的范畴。

后面的讲解主要围绕LogisticRegression和LogisticRegressionCV中的重要参数的选择来来展开，这些参数的意义在这两个类中都是一样的。

# 2. 正则化选择参数：penalty

LogisticRegression和LogisticRegressionCV默认就带了正则化项。penalty参数可选择的值为"l1"和"l2"，分别对应L1的正则化和L2的正则化，默认是L2的正则化。

在调参时如果我们主要的目的只是为了解决过拟合，一般penalty选择L2正则化就够了。但是如果选择L2正则化发现还是过拟合，即预测效果差的时候，就可以考虑L1正则化。另外，如果模型的特征非常多，我们希望一些不重要的特征系数归零，从而让模型系数稀疏化的话，也可以使用L1正则化。

penalty参数的选择会影响我们损失函数优化算法的选择。即参数solver的选择，如果是L2正则化，那么4种可选的算法{'newton-cg', 'lbfgs', 'liblinear', 'sag'}都可以选择。但是如果penalty是L1正则化的话，就只能选择'liblinear'了。这是因为L1正则化的损失函数不是连续可导的，而{'newton-cg', 'lbfgs','sag'}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。而'liblinear'并没有这个依赖。

# 3. 优化算法选择参数：solver

solver参数决定了我们对逻辑回归损失函数的优化方法，有4种算法可以选择，分别是：

a) 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。

b) 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。

c) 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。

d) 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。

从上面的描述可以看出，newton-cg, lbfgs和sag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。

# 4. 分类方式选择参数：multi_class

multi_class参数决定了我们分类方式的选择，有 ovr和multinomial两个值可以选择，默认是 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。

从上面的描述可以看出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。

# 5. 类型权重参数： class_weight

class_weight参数用于标示分类模型中各种类型的权重，可以不输入，即不考虑权重，或者说所有类型的权重一样。如果选择输入的话，可以选择balanced让类库自己计算类型权重，或者我们自己输入各个类型的权重，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。

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。

当然，对于第二种样本失衡的情况，我们还可以考虑用下一节讲到的样本权重参数： sample_weight，而不使用class_weight。sample_weight在下一节讲。

# 6. 样本权重参数： sample_weight

上一节我们提到了样本不失衡的问题，由于样本不平衡，导致样本不是总体样本的无偏估计，从而可能导致我们的模型预测能力下降。遇到这种情况，我们可以通过调节样本权重来尝试解决这个问题。调节样本权重的方法有两种，第一种是在class_weight使用balanced。第二种是在调用fit函数时，通过sample_weight来自己调节每个样本权重。

在scikit-learn做逻辑回归时，如果上面两种方法都用到了，那么样本的真正权重是class_weight*sample_weight.

以上就是scikit-learn中逻辑回归类库调参的一个小结，还有些参数比如正则化参数C（交叉验证就是 Cs），迭代次数max_iter等，由于和其它的算法类库并没有特别不同，这里不多累述了。

**sklearn.model_selection.train_test_split随机划分训练集和测试集**

官网文档：http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split

- **一般形式：**

train_test_split是交叉验证中常用的函数，功能是从样本中随机的按比例选取train data和testdata，形式为：

```
X_train,X_test, y_train, y_test =
cross_validation.train_test_split(train_data,train_target,test_size=0.4, random_state=0)
```

- **参数解释：**



train_data：所要划分的样本特征集

train_target：所要划分的样本结果

test_size：样本占比，如果是整数的话就是样本的数量

random_state：是随机数的种子。

随机数种子：其实就是该组随机数的编号，在需要重复试验的时候，保证得到一组一样的随机数。比如你每次都填1，其他参数一样的情况下你得到的随机数组是一样的。但填0或不填，每次都会不一样。

随机数的产生取决于种子，随机数和种子之间的关系遵从以下两个规则：

种子不同，产生不同的随机数；种子相同，即使实例不同也产生相同的随机数。

ROC曲线指受试者工作特征曲线 / 接收器操作特性曲线(receiver operating characteristic curve), 是反映敏感性和特异性连续变量的综合指标,是用构图法揭示敏感性和特异性的相互关系，它通过将连续变量设定出多个不同的临界值，从而计算出一系列敏感性和特异性，再以敏感性为纵坐标、（1-特异性）为横坐标绘制成曲线，曲线下面积越大，诊断准确性越高。在ROC曲线上，最靠近坐标图左上方的点为敏感性和特异性均较高的临界值。

考虑一个二分问题，即将实例分成正类（positive）或负类（negative）。对一个二分问题来说，会出现四种情况。如果一个实例是正类并且也被 预测成正类，即为真正类（True positive）,如果实例是负类被预测成正类，称之为假正类（False positive）。相应地，如果实例是负类被预测成负类，称之为真负类（True negative）,正类被预测成负类则为假负类（false negative）。

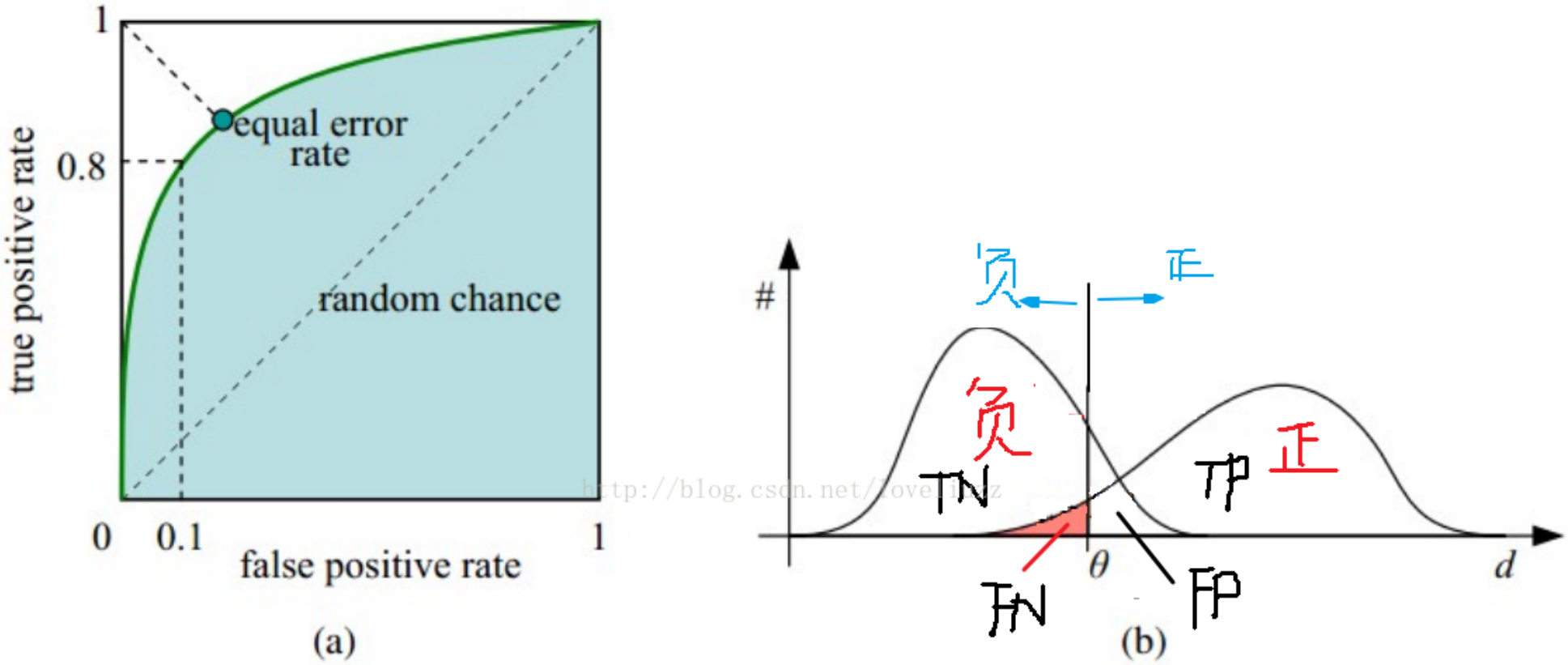
列联表如下表所示，1代表正类，0代表负类。

| | | 预测 | | |
|---|---|---|---|---|
| | | 1 | 0 | 合计 |
| 实际 | 1 | True Positive ( TP ) | False Negative ( FN ) | Actual Positive(TP+FN) |
| | 0 | False Positive ( FP) | True Negative(TN) | Actual Negative(FP+TN) |
| 合计 | | Predicted Positive(TP+FP) | Predicted Negative(FN+TN) | TP+FP+FN+TN |



从列联表引入两个新名词。其一是真正类率(true positive rate ,TPR), 计算公式为TPR=TP/ (TP+ FN)，刻画的是分类器所识别出的 正实例占所有正实例的比例。另外一个是假正类率(false positive rate, FPR),计算公式为FPR= FP / (FP + TN)，计算的是分类器错认为正类的负实例占所有负实例的比例。还有一个真负类率（True Negative Rate，TNR），也称为specificity,计算公式为TNR=TN/ (FP+ TN) = 1-FPR。

线可以用于评价一个分类器。




ROC曲线和它相关的比率

(a)理想情况下，TPR应该接近1，FPR应该接近0。

ROC曲线上的每一个点对应于一个threshold，对于一个分类器，每个threshold下会有一个TPR和FPR。

比如Threshold最大时，TP=FP=0，对应于原点；Threshold最小时，TN=FN=0，对应于右上角的点(1,1)

(b)随着阈值theta增加，TP和FP都减小，TPR和FPR也减小，ROC点向左下移动。