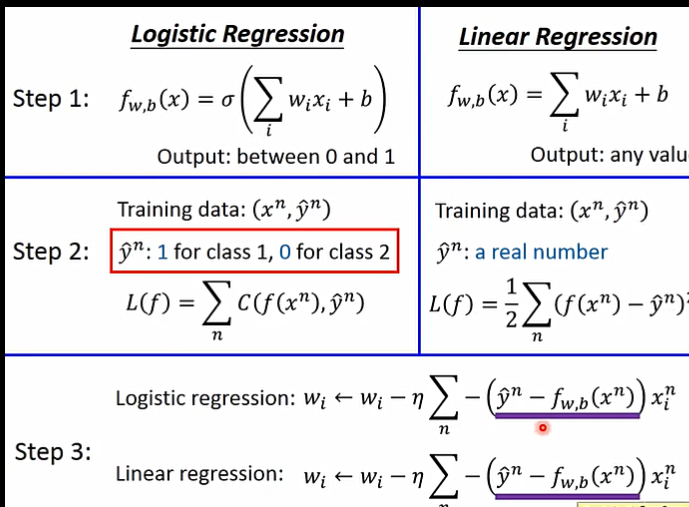
1、逻辑回归与线性回归的联系与区别



**一、线性回归与逻辑回归的区别**

　1）线性回归要求因变量服从正态分布，logistic回归对变量分布没有要求。

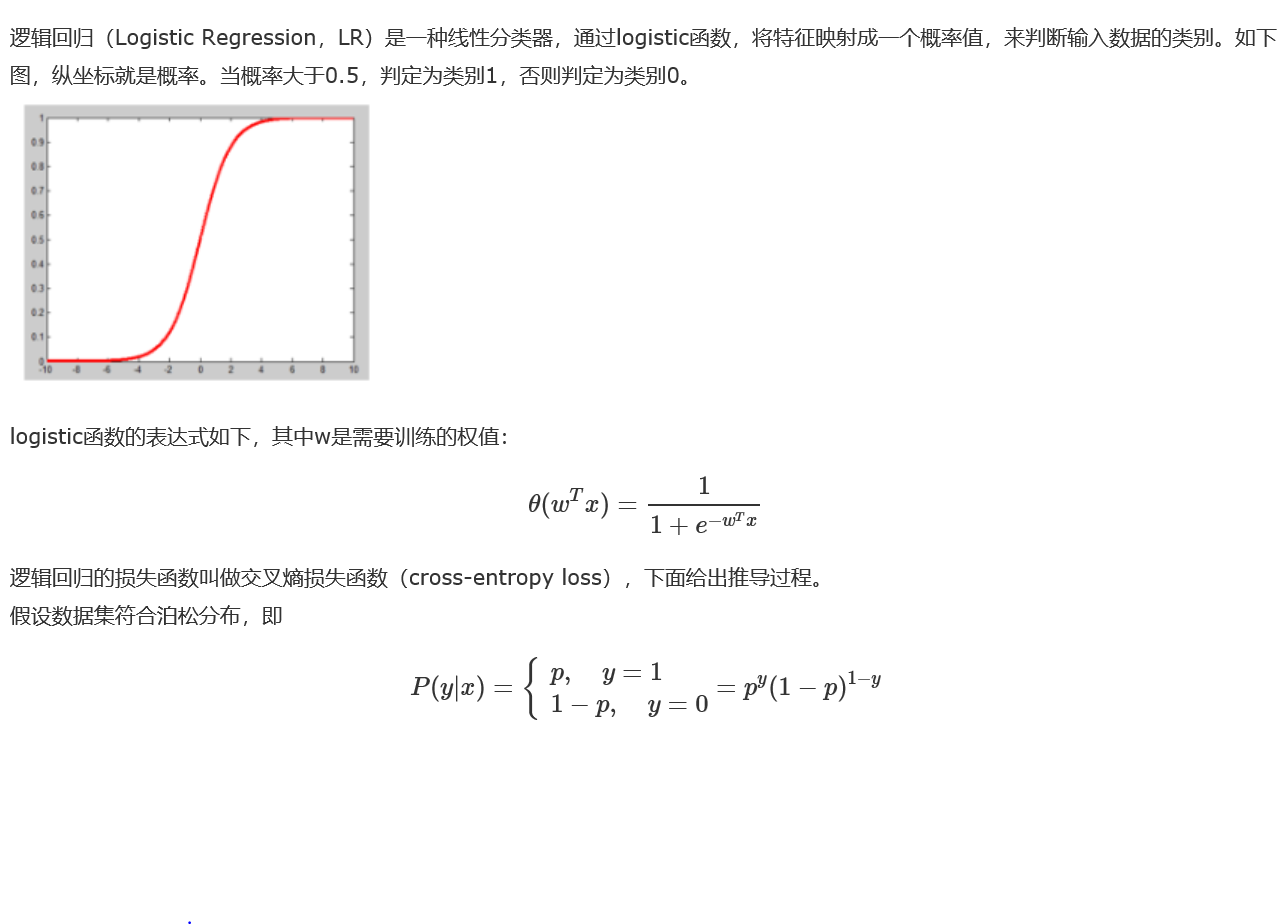
　2）线性回归要求因变量（Y）是连续性数值变量，而logistic回归要求因变量是分类型变量。

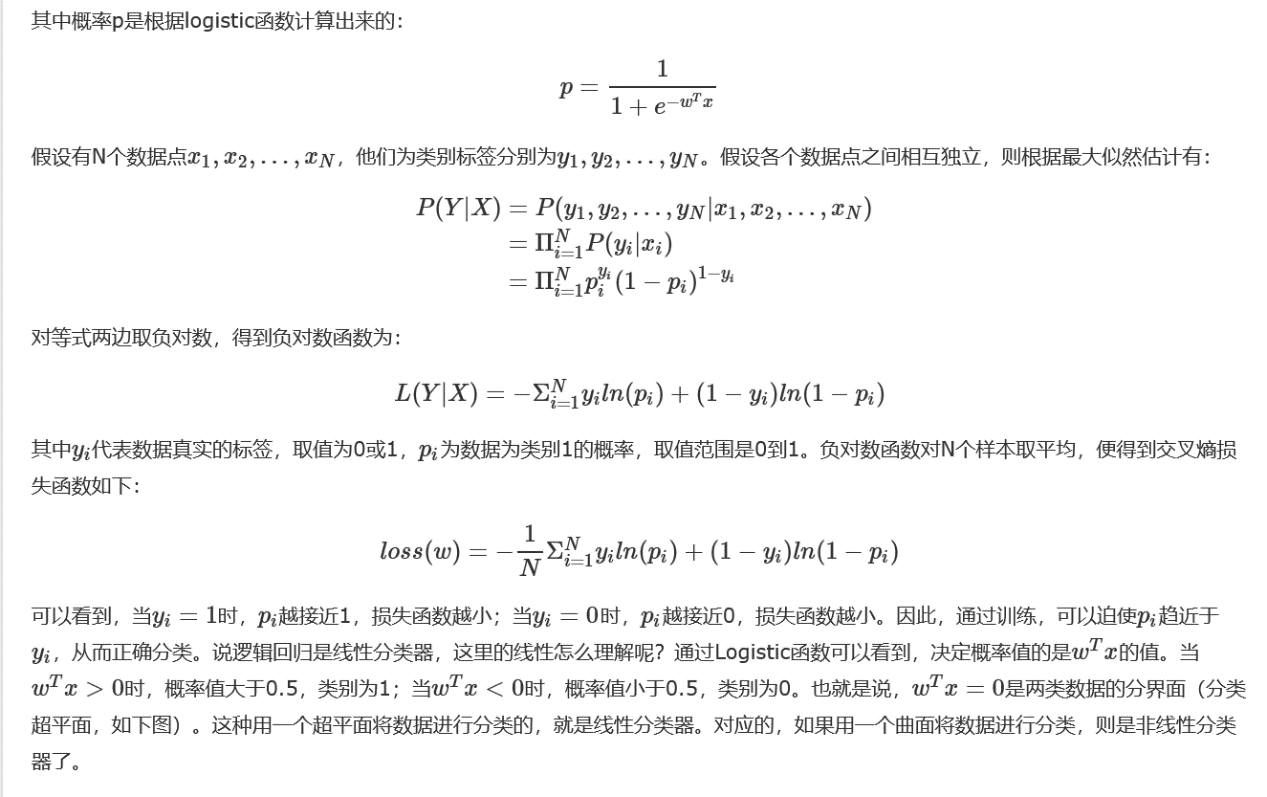
　3）线性回归要求自变量和因变量呈线性关系，而logistic回归不要求自变量和因变量呈线性关系

　4）线性回归是直接分析因变量与自变量的关系，而logistic回归是分析因变量取某个值的概率与自变量的关系

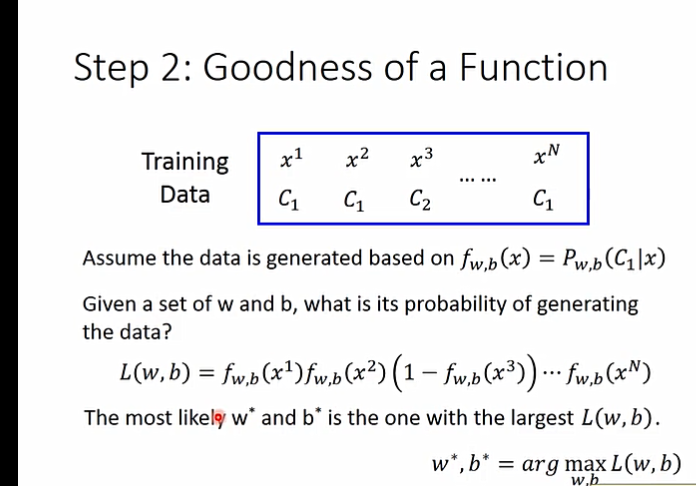
https://www.cnblogs.com/always-fight/p/10729982.html

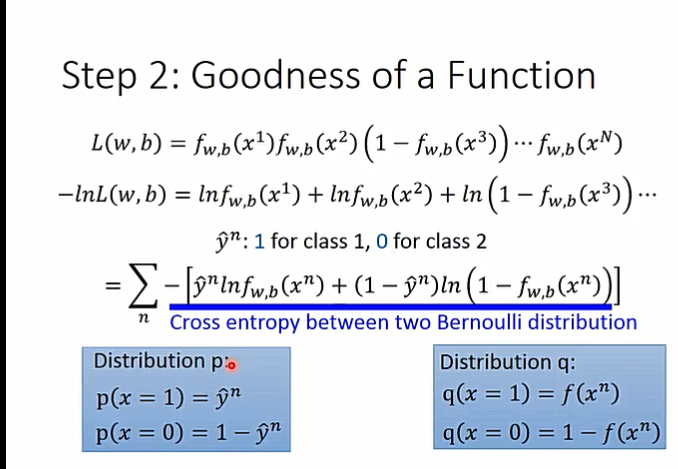
2、 逻辑回归的原理





3、逻辑回归损失函数推导及优化





**优化：随机梯度下降法**和**拟牛顿迭代法**

一中step3 中显示cross entropy和线性回归求导得到的结果相同

优化过程相似

4、 正则化与模型评估指标

过滤式与包裹式的结合，就是在学习器训练过程中自动进行特征选择。

L1 正则可以产生稀疏解。

**三、L1、L2正则化：**

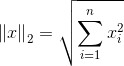
给定向量 x=(x_{1},x_{2},...,x_{n})，

**L0范数：**并不是一个真正的范数，它主要被用来度量向量中非零元素的个数；

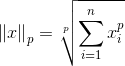
**L1范数：**向量各个元素绝对值之和；

\left \| x \right \|_{1}=\sum_{i=1}^{n}|x_{i}|

**L2范数：**向量各个元素的平方求和然后求平方根；



**Lp范数：**向量各个元素绝对值的p次方求和然后求 1/p 次方；



**L\infty范数：**响亮的各个元素求绝对值，取最大那个元素的绝对值

\left \| x \right \|_{\infty }=max\left ( |x_{i}| \right )

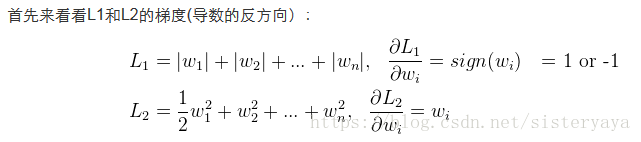
**L1范数正则化：**就是通过向目标函数中添加L1范数，使的学习到的结果满足稀疏化，用于特征选择。

**1、添加L1和L2正则化有什么用？**

* L1正则化可以产生稀疏权值矩阵，即产生一个稀疏模型（很多0），可以用于特征选择；
* L2正则化可以防止模型过拟合（overfitting）；一定程度上，L1也可以防止过拟合；

**2、为什么L1可以实现稀疏化，L2不可以？**

**1）数学公式角度**

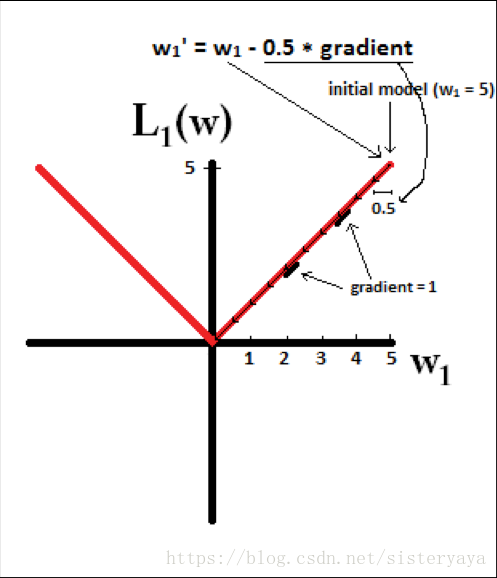


所以(不失一般性，我们假定：wi等于不为0的某个正的浮点数，学习速率η 为0.5)

**L1的权值更新公式为：**



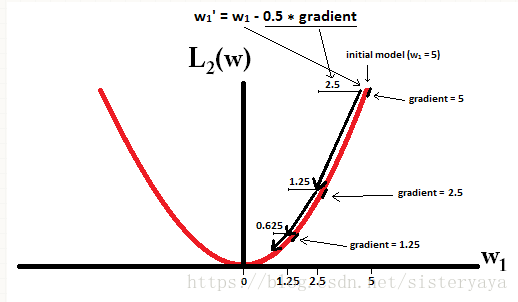
 也就是说权值每次更新都固定减少一个特定的值(比如0.5)，那么经过若干次迭代之后，权值就有可能减少到0。



**L2的权值更新公式为：**



       也就是说权值每次都等于上一次的1/2，那么，虽然权值不断变小，但是因为每次都等于上一次的一半，所以很快会收敛到较小的值但不为0。



**总结：**

L1能产生等于0的权值，即能够剔除某些特征在模型中的作用（特征选择），即产生稀疏的效果。

L2可以得迅速得到比较小的权值，但是难以收敛到0，所以产生的不是稀疏而是平滑的效果。

**2）几何图像角度**

1、L1正则



\alpha越大，方形越小，也可以取到很小的值；

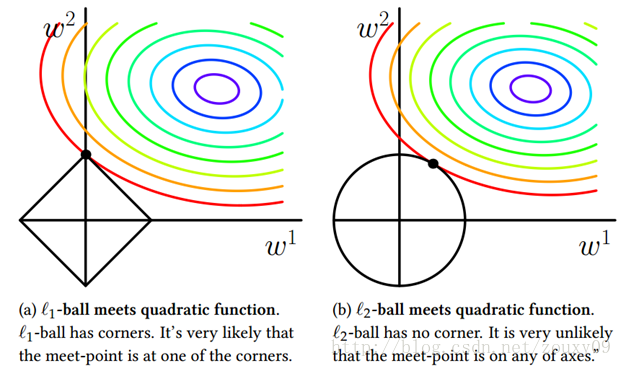
2、L2正则



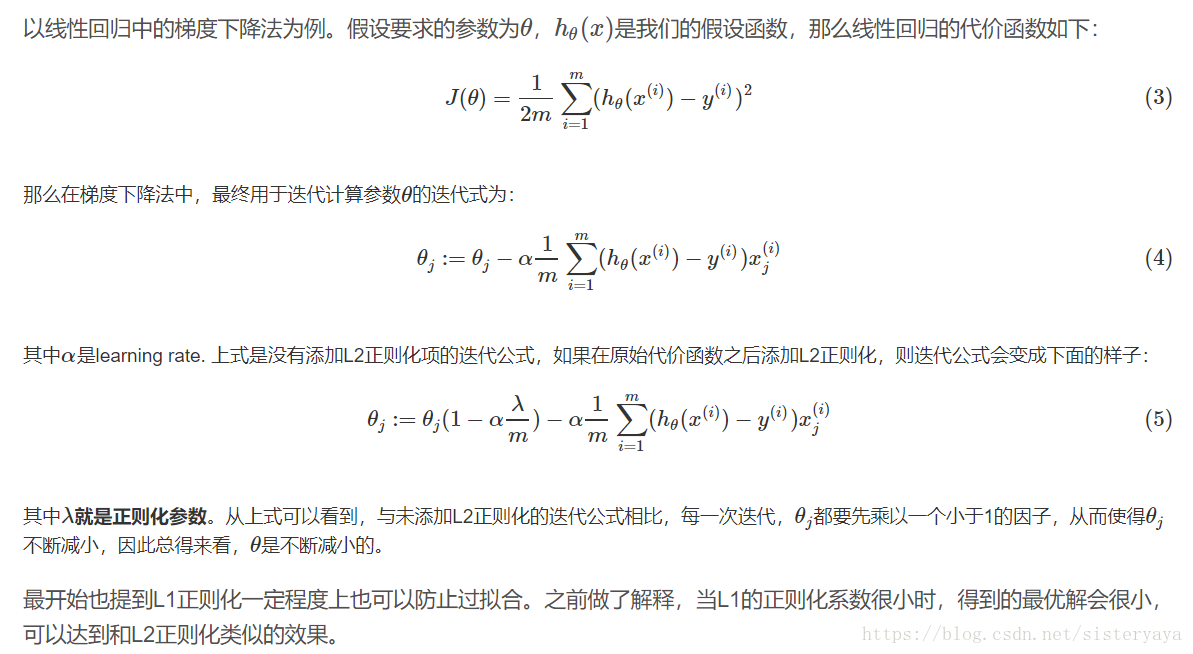
        在二维空间表示，左边为L1函数，图示最优解落在坐标轴上，意味着某些参数为0，从而实现稀疏化；右边为L2，函数图像为圆形，与方形相比，没有棱角，在坐标轴相交的可能性大大减小，因此没有稀疏性。

**西瓜书上解释的很清楚：假设 x 仅有两个属性，所以上面两个优化目标解出的 w 都只有两个分量，即w_{1},w_{2}，我们将其作为两个坐标轴，先绘制出不带正则的损失函数的“等值线”，即在（w_{1},w_{2}）空间中 J_{0} 取值相同的点的直线，再分别绘制出 L_{1} 范数和 L_{2} 范数的等值线，即在（w_{1},w_{2}）空间中 L_{1} 范数取值相同的点的连线，以及 L_{2} 范数取值相同的点的连线，如下图所示，我们优化目标的解要在目标函数和正则化项之间折中，即它们的等值线的相交处；可以看出，采用L1范数时交点常出现在坐标轴上，即 w_{1} 或 w_{2} 为0，而采用L2范数时交点更易出现在某个象限内，即 w_{1} 或 w_{2} 非0；换句话说，采用L1范数更容易得到稀疏解。**

* L1正则化可以产生稀疏权值矩阵，即产生一个稀疏模型（很多0），可以用于特征选择；
* L2正则化可以防止模型过拟合（overfitting）；一定程度上，L1也可以防止过拟合；



**3、那为什么L2正则化可以获得值很小的参数？（L2正则为什么可以防止过拟合）**



因为 \theta 是不断减小的，达到了**权重衰减**的效果。

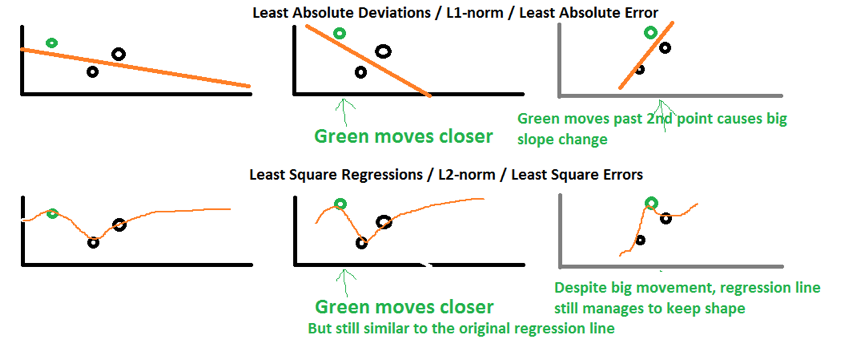
**过拟合：可以描述为特征维数很大，但样本很少；也可以说模型太复杂，将噪音数据也一并学习了。**

**L2正则为什么可以解决过拟合？过拟合**就是拟合函数需要顾忌到每一个点，使函数波动很大，函数波动大的地方函数的导数值是非常大的，导致导数值大的原因是系数太大，因为自变量是可大可小的，所以只有系数足够大时，导数才会很大。那么正则化就是通过约束参数的范围使其不要太大，所以它能在一定程度上减小过拟合。

**所以解决过拟合的方法：**增加训练样本，减少特征（降低模型复杂度）等

**4、L2比L1稳定？**

绿色的样本点稍作变动，就会影响回归线的斜率，L1变化很大，会影响到其他点的预测值，相比之下，L2比较稳定；



https://blog.csdn.net/sisteryaya/article/details/81490014#%E4%BA%8C%E3%80%81L1%E3%80%81L2%E6%AD%A3%E5%88%99%E5%8C%96%EF%BC%9A

5、逻辑回归的优缺点

LR是解决工业规模问题最流行的算法。在工业应用上，如果需要分类的数据拥有很多有意义的特征，每个特征都对最后的分类结果有或多或少的影响，那么最简单最有效的办法就是将这些特征线性加权，一起参与到决策过程中。比如预测广告的点击率，从原始数据集中筛选出符合某种要求的有用的子数据集等等。

   优点：1）适合需要得到一个分类概率的场景。2）计算代价不高，容易理解实现。LR在时间和内存需求上相当高效。它可以应用于分布式数据，并且还有在线算法实现，用较少的资源处理大型数据。3）LR对于数据中小噪声的鲁棒性很好，并且不会受到轻微的多重共线性的特别影响。（严重的多重共线性则可以使用逻辑回归结合L2正则化来解决，但是若要得到一个简约模型，L2正则化并不是最好的选择，因为它建立的模型涵盖了全部的特征。）

   缺点：1）容易欠拟合，分类精度不高。2）数据特征有缺失或者特征空间很大时表现效果并不好。

<https://blog.csdn.net/touch_dream/article/details/79371462>

6、样本不均衡问题解决办法

**解决样本不均衡的问题很多，主流的几个如下：**

1.样本的过采样和欠采样。

2..使用多个分类器进行分类。

3.将二分类问题转换成其他问题。

4.改变正负类别样本在模型中的权重。

**一、样本的过采样和欠采样。**

1.过采样：将稀有类别的样本进行复制，通过增加此稀有类样本的数量来平衡数据集。该方法适用于数据量较小的情况。

2.欠抽样：从丰富类别的样本中随机选取和稀有类别相同数目的样本，通过减少丰富类的样本量啦平衡数据集。该方法适用于数据量较大的情况。

3.也可以将过采样和欠采样结合在一起使用。

4.使用SMOTE方法来构造样本。

　　SMOTE算法是一种过采样的算法。这个算法不是简单的复制已有的数据，而是在原有数据基础上，通过算法产生新生数据。

　　算法思想：基于距离度量的方式计算两个或多个稀有类样本之间的相似性。

　　　　　　　然后选择其中的一个样本作为基础样本，

　　　　　　　再在邻居样本中随机选取一定数量的样本对那个基础样本的一个属性进行噪声。每次处理一个属性，通过这样的方式产生新生数据。

**二、使用多个分类器进行分类。**

　　方法一中介绍的过采样，欠采样，都存在相应的问题。

　　过采样：可能会存在过拟合问题。（可以使用SMOTE算法，增加随机的噪声的方式来改善这个问题）

　　欠采样：可能会存在信息减少的问题。因为只是利用了一部分数据，所以模型只是学习到了一部分模型。

　　有以下两种方法可以解决欠采样所带来的问题。

　　方法一：模型融合 （bagging的思想 ）

　　思路：从丰富类样本中随机的选取（有放回的选取）和稀有类等量样本的数据。和稀有类样本组合成新的训练集。这样我们就产生了多个训练集，并且是互相独立的，然后训练得到多个分类器。

　　　　　若是分类问题，就把多个分类器投票的结果（少数服从多数）作为分类结果。

　　　　　若是回归问题，就将均值作为最后结果。

　　方法二：增量模型 （boosting的思想）

　　思路：使用全部的样本作为训练集，得到分类器L1

　　　　　从L1正确分类的样本中和错误分类的样本中各抽取50%的数据，即循环的一边采样一个。此时训练样本是平衡的。训练得到的分类器作为L2.

　　　　　从L1和L2分类结果中，选取结果不一致的样本作为训练集得到分类器L3.

　　　　　最后投票L1,L2,L3结果得到最后的分类结果。

**三、将二分类问题转换成其他问题。**

　　可以将不平衡的二分类问题转换成异常点检测，或者一分类问题（可使用one-class svm建模）

**四、改变正负类别样本在模型中的权重。**

　　使用代价函数学习得到每个类的权值，大类的权值小，小类的权值大。刚开始，可以设置每个类别的权值与样本个数比例的倒数，然后可以使用过采样进行调优。

**五、注意点：**

**1.不平衡问题的评价指标**

　　准确度这个评价指标在类别不均衡的分类任务中并不能work。几个比传统的准确度更有效的评价指标：

　　混淆矩阵(Confusion Matrix)：使用一个表格对分类器所预测的类别与其真实的类别的样本统计，分别为：TP、FN、FP与TN。  
　　精确度(Precision)  
　　召回率(Recall)  
　　F1得分(F1 Score)：精确度与找召回率的加权平均。  
  特别是：

　　Kappa (Cohen kappa)  
　　ROC曲线(ROC Curves)：见Assessing and Comparing Classifier Performance with ROC Curves

**2.交叉验证**

　　在K-Fold 校验中，每一份数据集中原则上应该保持类别样本比例一样或者近似，如果每份数据集中小类样本数目过少，那么应该降低K的值，知道小类样本的个数足够。

7. sklearn参数

*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*)

参数：

　　=> **penalty** : str, ‘l1’ or ‘l2’

　　　　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。

　　=> **dual** : bool

　　　　对偶或者原始方法。Dual只适用于正则化相为l2 liblinear的情况，通常样本数大于特征数的情况下，默认为False

　　=> **tol** : float, optional

　　迭代终止判据的误差范围。

　　=> **C** : float, default: 1.0

　　　　C为正则化系数λ的倒数，通常默认为1。设置越小则对应越强的正则化。

　　=> **fit\_intercept** : bool, default: True

　　　　是否存在截距，默认存在

　　=> **intercept\_scaling** : float, default 1.

　　　　仅在正则化项为"liblinear"，且fit\_intercept设置为True时有用。

　　=> **class\_weight** : dict or ‘balanced’, default: None

　　　　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。

　　　　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]  0,1分别出现2次和三次

**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。

　　=> **random\_state** : int, RandomState instance or None, optional, default: None

　　　　随机数种子，默认为无，仅在正则化优化算法为sag,liblinear时有用。

　　=> **solver** : {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}

　　　　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，SAG是一种线性收敛算法，这个速度远比SGD快。

　　　　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则既可以用L1正则化也可以用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

　　=> **max\_iter** : int, optional

　　　　仅在正则化优化算法为newton-cg, sag and lbfgs 才有用，算法收敛的最大迭代次数。

　　=> **multi\_class** : str, {‘ovr’, ‘multinomial’}, default: ‘ovr’

 　  　　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** : int, default: 0

　　=> **warm\_start** : bool, default: False

　　=> **n\_jobs** : int, default: 1

　　　　如果multi\_class ='ovr'“，并行数等于CPU内核数量。当“solver”设置为“liblinear”时，无论是否指定“multi\_class”，该参数将被忽略。如果给定值-1，则使用所有内核。