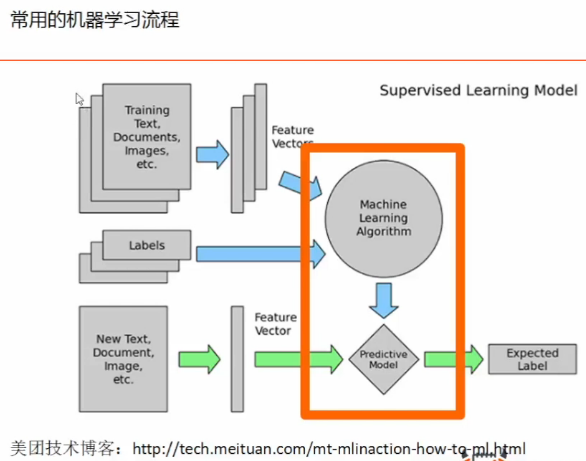
Logistic Regression



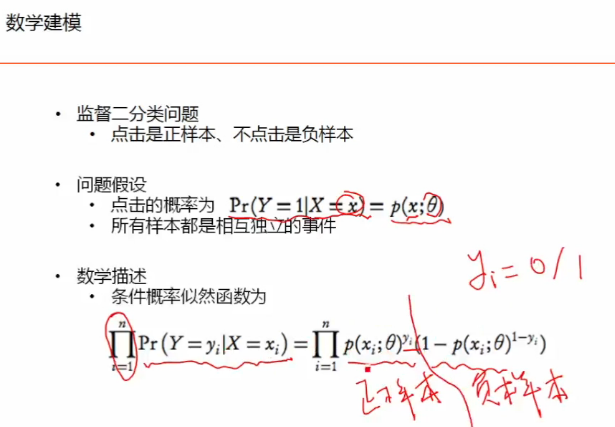
讲解红框内的内容！

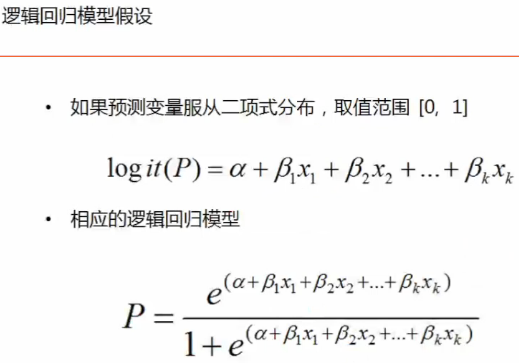




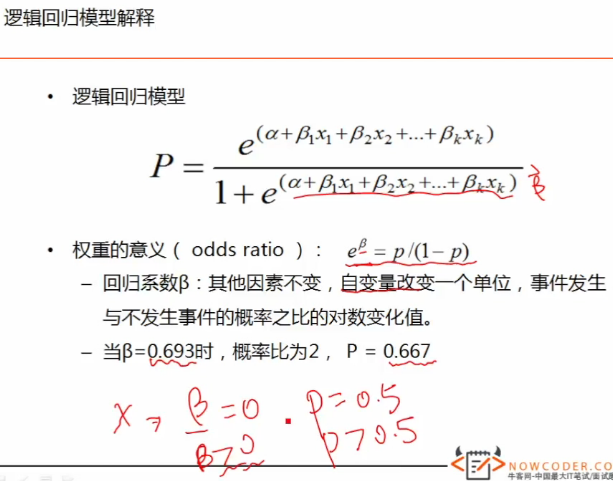


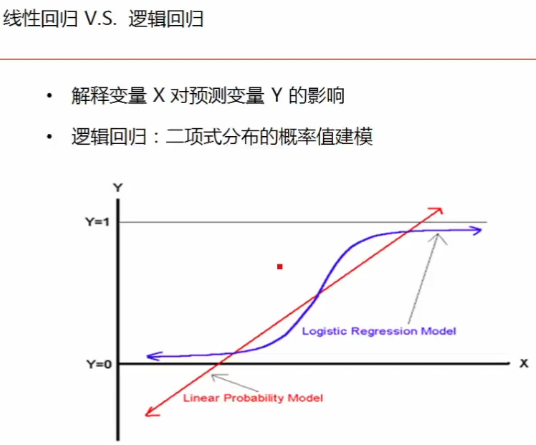


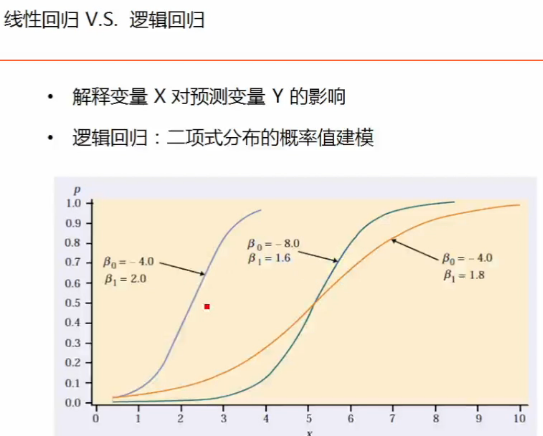




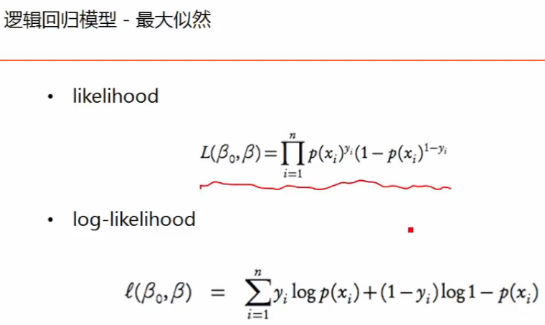
逻辑回归的模型假设：预测变量符合二项式分布，而且取值范围是再0-1之间，所有的特征变量都是线性变量特征的加权。



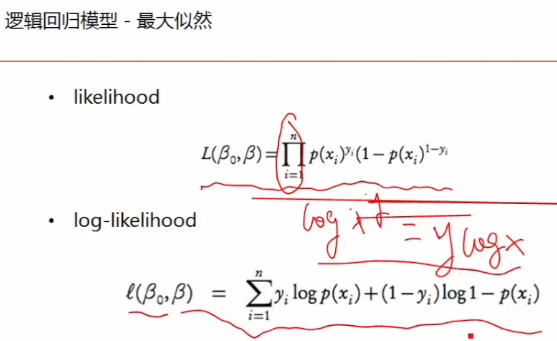




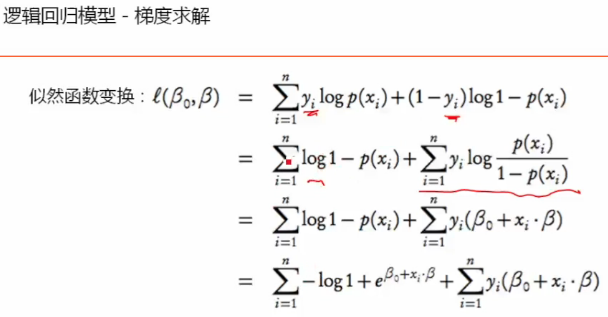
横轴是权重，对最后预估的值的贡献是越大的

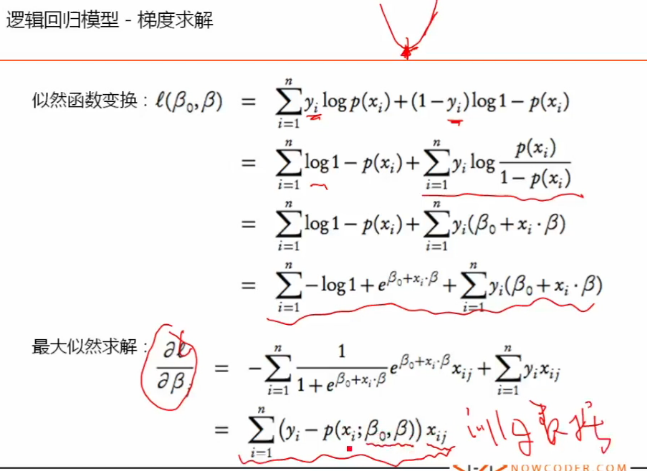


以逻辑回归为例，得到红线所示函数。一般我们使用log使其变成加号



我们目的是利用最大似然对对数似然函数做优化！逻辑回归取log因为逻辑回归模型是指数形式，取log的话指数没了，也可以用最大熵法。

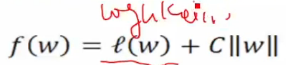






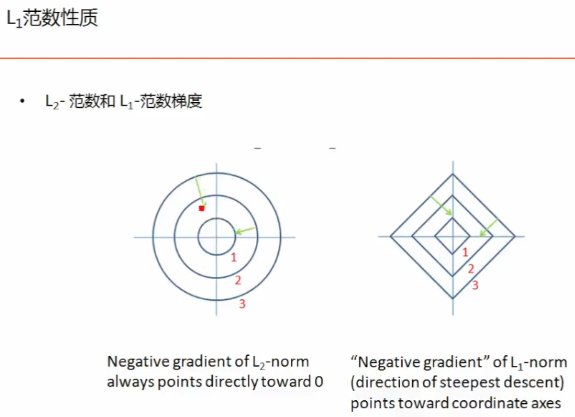
Xij是训练样本，xi和yi都是训练数据，贝塔0和贝塔是要求解的权重，

里边的xi是某一维具体的向量（是训练数据），根据这个公式我们可以进行梯度的求解，最后得到一个模型的估值

但是一般再进行模型求解的过程中，我们通常引入一个模型的正则化项，目的是抑制模型参数对数据的过拟合。比如红线部分是log-likelihood；C|w|是后边加的正则，目的是希望log-likelihood函数和模型中间取的一个balance，希望log-likelihood越小越好，加上惩罚项希望模型越简单越好，有的时候是log-likelihood越来越小的时候模型变得越来越复杂，所以希望再模型尽量简单的情况下，对已有的训练数据更好的拟合值，所以加上了一个惩罚项。一般是加入一个二范数或者一范数。二范数就是w的平方。一范数就是w的绝对值。

L2假设先验概率w符合正态分布就加入w的平方

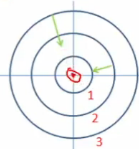
L1的线性分布是个拉普拉斯的分布，但是工业界常用L1范数，因为泛化性能强，模型参数比较少，再引擎里加载数据对内存的压力以及做预测时的速度都是很好的帮助。

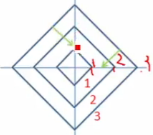


为什么L1或者L2能到来那样的效果？

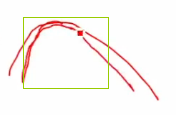
L2是平方所以画出来是个圆，L1是个绝对值所以画出来是方框

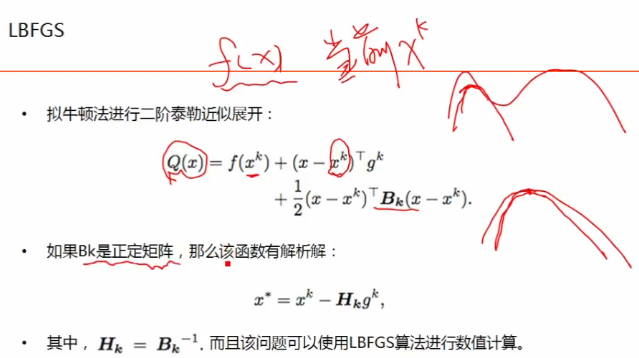
L1范数比较容易得到一个稀疏的模型，

L2会朝着中心点走，一般会落在中心点0点附近，最后w会很小，但是不等于0.

L1朝着某一维坐标进行下降，它比较容易落在一个维度上面，当它落在红点处的维度上时，另外一个维度就被正则化掉了。所以L1比较容易得到一个稀疏的模型。

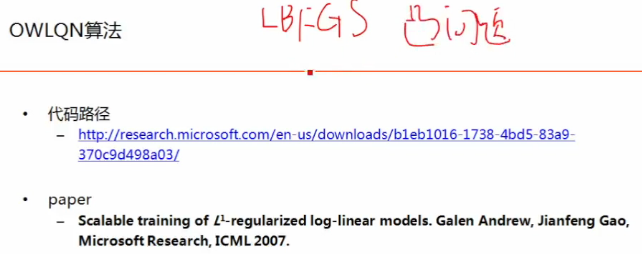
刚才讲到使用梯度下降进行求解，训练model。但是再工业界还是会使用二阶下降方法LBFGS，因为梯度下降在收敛附近容易出现抖动，它的naisearch不太好控制，它需要更多的迭代轮数，因为它是一阶收敛的。而LBFGS是二阶收敛的。

LBFGS核心思想：比如有一个函数f(x)，再当前x(k)这个地方做一个二阶的泰勒展开(对原始的f(x)的一个近似)，比如说有这么一个函数在某一点做个泰勒展开，得到方框中下方函数的结果，此函数是再该点的一个二阶近似。此函数在x(k)处和原函数是几乎相等的，但是再偏离x(k)点较远时，近似函数Q(x)就与原函数f(x)偏离的比较远。



Bk如果不是正定的，那么我们常常加一个正则上去就变成正定的了。就会得到最优解。之后呢，H(k)等于B(k)的导数，涉及到矩阵求逆。

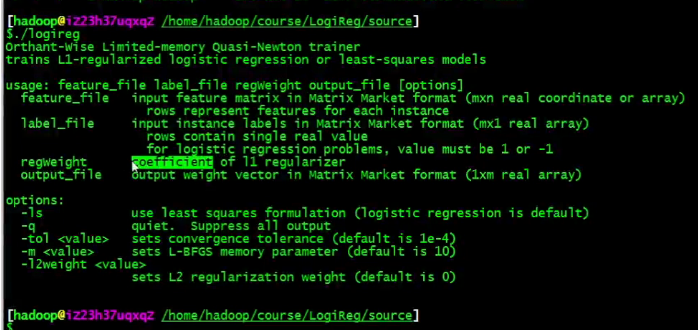
前提条件是它必须是个正定矩阵，所以一般我们会在这里边C|w|加一个正则项，例如加一个L2范数，那么就在原来凸问题上加了一个严格的凸问题，然后它就变成了一个正定矩阵。一般情况下，加了L2范数的话，用LBFGS是可以对它正常求解的。



LBFGS可以用来求解一个凸问题，再加上一个L1范数，对LBFGS做一个简单的变形，就变成OWLQN算法。

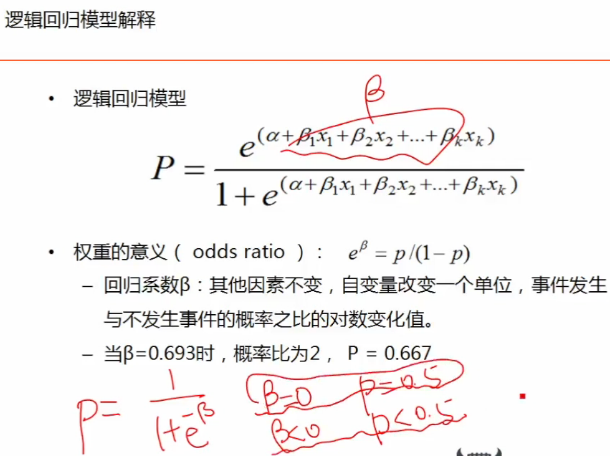
微软的高剑锋，Logistic Model+L1范数的求解方法

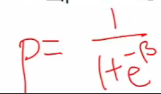
什么叫范数？L1或L2在英文中叫norm



我们可以修改L1或者L2正则项的系数

再分类时，负类的特征是不是就没有意义了？



将看成是贝塔，然后分子分母都除以贝塔，变成，当贝塔等于0时P是等于0.5的，当贝塔<0时，p是小于0.5的。等于0对你是没有帮助的，小于0说明是个负样本，帮你做了个决策。

LBFGS与梯度下降法求解模型的差别是什么？

梯度下降法是个一阶的求解方法，LBFGS是利用牛顿的二阶下降的特性(泰勒展开后变成了二阶下降)；比如一阶下降可能需要迭代100轮才能到下降点，二阶下降可能只需要迭代十轮就能到下降点(最优点)

逻辑回归对非数值型特征怎么处理？

使用训练化来进行处理，比如今天是星期六，就是非数值的，但是我们可以用一个id来表示它。

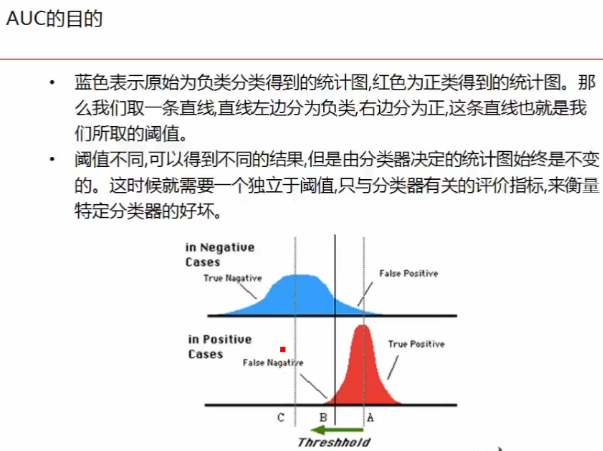
一般正则项选择L1还是L2有什么经验？

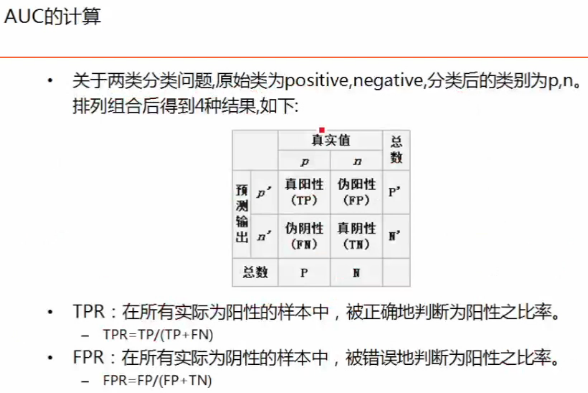
一般如果要稀疏model的话使用L1，范数越大模型越小。

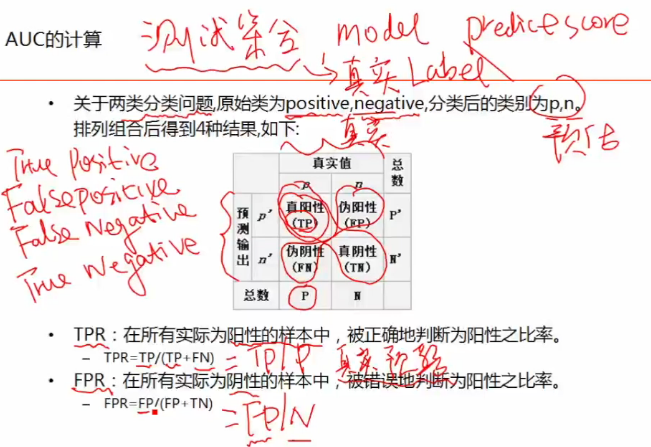
数值型的0或1有大小之分，类别型的0或1没有大小之分，这在逻辑回归当中有什么影响吗？

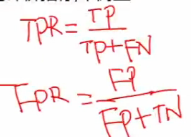
类别型0或1它的大小是没有意义的，而数值型是有大小之分的，代表强弱。

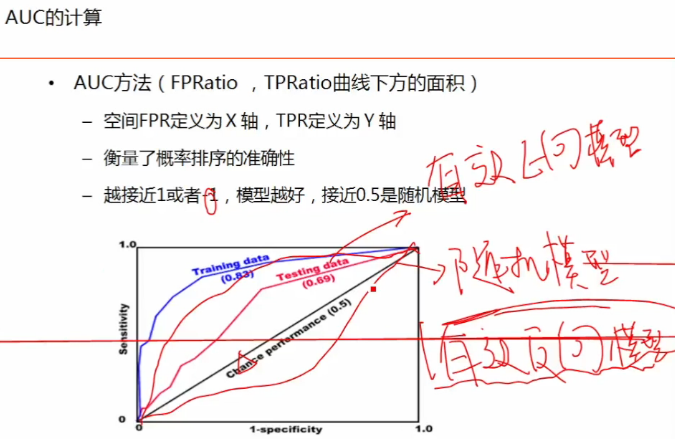
当我们有了一个模型，如何去评估一个模型的好坏？



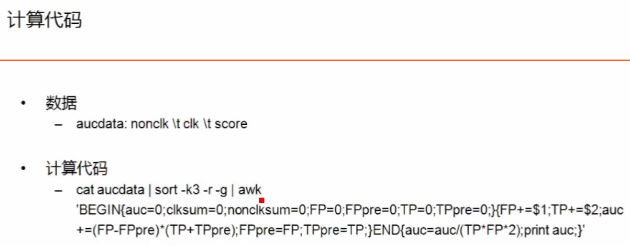


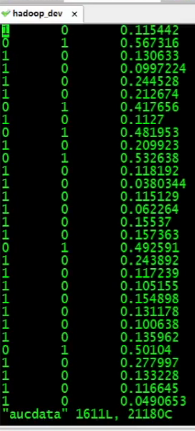






如何计算AUC面积大小？



第一列为1代表负样本，第二列是1代表正样本，第三列是预测score。sort –r 是倒排，-k3对第三列排序 –g用数值来倒排



AUC主要用来评估二分类的。

#-\*- coding:utf-8 -\*-

def auc():

    data = []

    count = [0,0]

    with open('aucdata') as f:

        for line in f.readlines():

            label1, lable2, score = line.strip().split('\t')

            label = 0 if label1 == '1' else 1

            score = float(score)

            count[label] += 1

            data.append([score, label])

    data = sorted(data, key = lambda x:x[0], reverse = True)

    auc = 0

    tp = 0

    fp = 0

    tppre = 0

    fppre = 0

    for sort\_data in data:

        if sort\_data[1] == 0:

            fp = fp + 1

        else:

            tp = tp + 1

        auc = auc + (fp - fppre) \* (tp + tppre)

        fppre = fp

        tppre = tp

    print 1.0 \* auc /(fp \* tp \* 2)

def auc1():

    data = []

    count = [0,0]

    with open('aucdata') as f:

        for line in f.readlines():

            label1, label2, score = line.strip().split('\t')

            label = 0 if label1 == '1' else 1

            score = float(score)

            count[label] += 1

            data.append([score, label])

        data = sorted(data,key=lambda x:x[0],reverse=True)

        n = len(data)

        k = 0

        for i in xrange(n):

            if data[i][1] == 1:

                k += n-i

        print (k-count[1]\*(count[1]+1)/2.0)/(count[0]\*count[1])

if \_\_name\_\_ == '\_\_main\_\_':

    auc()