

第15课：逻辑回归——用来做分类的回归模型

现在我们回到 LR 模型本身。

### 回归模型做分类

从前面关于分类与回归的定义来看，分类模型和回归模型似乎是泾渭分流的。输出离散结果的就是用来做分类的，而输出连续结果的，就用来做回归。

我们前面讲的两个模型：线性回归的预测结果是一个连续值域上的任意值，而朴素贝叶斯分类模型的预测结果则是一个离散值。

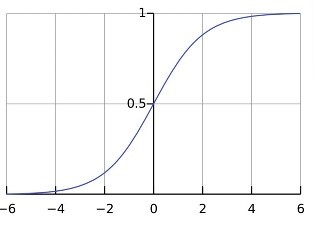
但 LR 却是用来做分类的。它的模型函为：

hθ(x)=11+e−θTxhθ(x)=11+e−θTx

设 z=θTxz=θTx，则

h(z)=11+e−zh(z)=11+e−z

在二维坐标中形成 S 形曲线：



上图中，zz 是自变量（横轴），最终计算出的因变量 yy（纵轴），则是一个 [0,1] 区间之内的实数值。

一般而言，当 y>0.5y>0.5 时，zz 被归类为真（True）或阳性（Positive），否则当 y<=0.5y<=0.5 时，zz 被归类为假（False）或阴性（Negative）。

所以，在模型输出预测结果时，不必输出 yy 的具体取值，而是根据上述判别标准，输出1（真）或0（假）。

因此，LR 典型的应用是二分类问题上，也就是说，把所有的数据只分为两个类。

****注意：****当然，这并不是说 LR 不能处理多分类问题，它当然可以处理，具体方法稍后讲。我们先来看 LR 本身。

看到此处，大家是不是会有点担心，如果大量的输入得到的结果都在 y=0.5y=0.5 附近，那岂不是很容易分错？

说得极端一点，如果所有的输入数据得出的结果都在 y=0.5y=0.5 附近，那岂不是没有什么分类意义了，和随机乱归类结果差不多？

这样的担心其实是不必要的。此模型函数在 y=0.5y=0.5 附近非常敏感，自变量取值稍有不同，因变量取值就会有很大差异，所以不用担心出现大量因细微特征差异而被归错类的情况——这也正是逻辑回归的“神奇”之处。

### 逻辑回归的目标函数

有了模型函数，来看看逻辑回归的目标函数。

逻辑函数 h(x)h(x) 是我们要通过训练得出来的最终结果。在最开始的时候，我们不知道其中的参数 θθ 的取值，我们所有的只是若干的 xx 和与其对应的 yy（训练集合）。训练 LR 的过程，就是求 θθ 的过程。

首先要设定一个目标：我们希望这个最终得出的 θθ 达到一个什么样的效果——我们当然是希望得出来的这个 θθ，能够让训练数据中被归为阳性的数据预测结果都为阳，本来被分为阴性的预测结果都为阴。

而从公式本身的角度来看，h(x)h(x) 实际上是 xx 为阳性的分布概率，所以，才会在 h(x)>0.5h(x)>0.5 时将 xx 归于阳性。也就是说 h(x)=P(y=1)h(x)=P(y=1)。反之，样例是阴性的概率 P(y=0)=1−h(x)P(y=0)=1−h(x)。

当我们把测试数据带入其中的时候，P(y=1)P(y=1) 和 P(y=0)P(y=0) 就都有了先决条件，它们为训练数据的 xx 所限定。因此：

P(y=1|x)=h(x);P(y=0|x)=1−h(x)P(y=1|x)=h(x);P(y=0|x)=1−h(x)。

根据****二项分布****公式，可得出 P(y|x)=h(x)y(1−h(x))(1−y)P(y|x)=h(x)y(1−h(x))(1−y)。

假设我们的训练集一共有 m 个数据，那么这 m 个数据的联合概率就是：

L(θ)=∏mi=1P(y(i)|x(i);θ)=∏mi=1(hθ(x(i)))y(i)(1−hθ(x(i)))(1−y(i))L(θ)=∏i=1mP(y(i)|x(i);θ)=∏i=1m(hθ(x(i)))y(i)(1−hθ(x(i)))(1−y(i))

我们求取 θθ 的结果，就是让这个 L(θ)L(θ) 达到最大。

还记得我们之前在朴素贝叶斯分类器中讲到的****极大似然估计****吗？其实此处 LR 目标函数的构建过程也是依据极大似然估计。

L(θ)L(θ) 就是 LR 的****似然函数****。我们要让它达到最大，也就是对其进行“****极大估计****”。因此，求解 LR 目标函数的过程，就是对 LR 模型函数进行极大似然估计的过程。

为了好计算，我们对它求对数。得到****对数似然函数****：

l(θ)=log(L(θ))=∑mi=1[y(i)log(hθ(x(i)))+(1−y(i))log(1−hθ(x(i)))]l(θ)=log⁡(L(θ))=∑i=1m[y(i)log⁡(hθ(x(i)))+(1−y(i))log⁡(1−hθ(x(i)))]

我们要求出让 l(θ)l(θ) 能够得到最大值的 θθ。

l(θ)l(θ) 其实可以作为 LR 的目标函数。前面讲过，我们需要目标函数是一个凸函数，具备最小值。因此我们设定：J(θ)=−l(θ)J(θ)=−l(θ)。

J(θ)=−log(L(θ))=−∑mi=1[y(i)log(hθ(x(i)))+(1−y(i))log(1−hθ(x(i)))]J(θ)=−log⁡(L(θ))=−∑i=1m[y(i)log⁡(hθ(x(i)))+(1−y(i))log⁡(1−hθ(x(i)))]

这样，求 l(θ)l(θ) 的最大值就成了求 J(θ)J(θ) 的最小值。J(θ)J(θ) 又叫做****负对数似然函数****。它就是 ****LR 的目标函数****。

#### 优化算法

我们已经得到了 ****LR 的目标函数****J(θ)J(θ)，并且****优化目标是最小化它****。

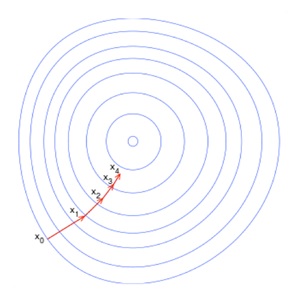
如何求解 θθ 呢？具体方法其实有很多。此处我们仍然运用之前已经学习过的，最常见最基础的梯度下降算法。

基本步骤如下：

• 通过对 J(θ)J(θ) 求导获得下降方向—— J′(θ)J′(θ)；

• 根据预设的步长 αα，更新参数 θ:=θ−αJ′(θ)θ:=θ−αJ′(θ)；

• 重复以上两步直到逼近最优值，满足终止条件。



既然知道了方法，我们就来计算一下。

已知：

J(θ)=−log(L(θ))=−∑mi=1[y(i)log(hθ(x(i)))+(1−y(i))log(1−hθ(x(i)))]J(θ)=−log⁡(L(θ))=−∑i=1m[y(i)log⁡(hθ(x(i)))+(1−y(i))log⁡(1−hθ(x(i)))]

J(θ)J(θ) 对 θθ 求导：

∂J(θ)∂θ=−∑mi=1[y(i)h′θ(x(i))hθ(x(i))−(1−y(i))h′θ(x(i))(1−hθ(x(i)))]=∑mi=1[(−y(i))h′θ(x(i))hθ(x(i))+(1−y(i))h′θ(x(i))(1−hθ(x(i)))]∂J(θ)∂θ=−∑i=1m[y(i)hθ′(x(i))hθ(x(i))−(1−y(i))hθ′(x(i))(1−hθ(x(i)))]=∑i=1m[(−y(i))hθ′(x(i))hθ(x(i))+(1−y(i))hθ′(x(i))(1−hθ(x(i)))]

因为有：

h′(z)=d(11+e−z)dz=−(−e−z(1+e−z)2)=e−z1+e−z11+e−z=(1−11+e−z)(11+e−z)=h(z)(1−h(z))h′(z)=d(11+e−z)dz=−(−e−z(1+e−z)2)=e−z1+e−z11+e−z=(1−11+e−z)(11+e−z)=h(z)(1−h(z))

同时，运用链式法则，有：

∂hθ(x)∂θ=∂hθ(x)∂(θx)x=hθ(x)(1−hθ(x))x∂hθ(x)∂θ=∂hθ(x)∂(θx)x=hθ(x)(1−hθ(x))x

将上式带入上面的 J(θ)J(θ) 求导式子里，有：

∂J(θ)∂θ=∑mi=1[(−y(i))hθ(x(i))(1−hθ(x(i)))x(i)hθ(x(i))+(1−y(i))hθ(x(i))(1−hθ(x(i)))x(i)(1−hθ(x(i)))]=∑mi=1[−y(i)+y(i)hθ(x(i))+hθ(x(i))−y(i)hθ(x(i))]x(i)=∑mi=1[hθ(x(i))−y(i)]x(i)∂J(θ)∂θ=∑i=1m[(−y(i))hθ(x(i))(1−hθ(x(i)))x(i)hθ(x(i))+(1−y(i))hθ(x(i))(1−hθ(x(i)))x(i)(1−hθ(x(i)))]=∑i=1m[−y(i)+y(i)hθ(x(i))+hθ(x(i))−y(i)hθ(x(i))]x(i)=∑i=1m[hθ(x(i))−y(i)]x(i)

当 xx 为多维的时候（设 xx 有 nn 维），则在对 z=θxz=θx 求导的时候，要对 xx 的每一个维度求导。

又因为 θθ 和 xx 维度相同，所以当 xx 有 nn 维的时候，θθ 同样是有 nn 维的。则 J(θ)J(θ)的求导也变成了对 θθ 的每一个维度求导：

∂J(θ)∂θj=∑mi=1[hθ(x(i))−y(i)]x(i)j;j=1,2,...,n∂J(θ)∂θj=∑i=1m[hθ(x(i))−y(i)]xj(i);j=1,2,...,n

因此，优化算法伪代码为：

Set initial value: θ0,αθ0,α

while (not convergence)

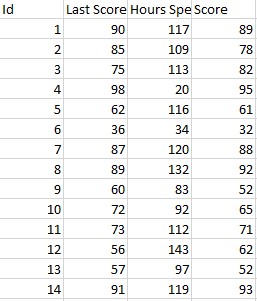
{

θj:=θj+α∑mi=1(y(i)−hθ(x(i)))x(i)jθj:=θj+α∑i=1m(y(i)−hθ(x(i)))xj(i)

}

### 实例及代码实现

我们来看一个例子，比如某位老师想用学生上学期考试的成绩（Last Score）和本学期在学习上花费的时间（Hours Spent）来预期本学期的成绩：



面对这样一个需求，我们可能首先想到的是线性回归，毕竟，要做的是预测本次的成绩。那样的话，我们取 X = [“Last Score”, “Hours Spent”]，y = “Score”。

用线性回归实现代码如下：

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.linear\_model **import** LinearRegression

**import** pandas **as** pd

*# Importing dataset*

data = pd.read\_csv('quiz.csv', delimiter=',')

used\_features = ["Last Score", "Hours Spent"]

X = data[used\_features].values

scores = data["Score"].values

X\_train = X[:11]

X\_test = X[11:]

*# Linear Regression - Regression*

y\_train = scores[:11]

y\_test = scores[11:]

regr = LinearRegression()

regr.fit(X\_train, y\_train)

y\_predict = regr.predict(X\_test)

**print**(y\_predict)

我们把前11个样本作为训练集，最后3个样本作为测试集。

这样训练出来之后，得到的预测结果为：[55.33375602 54.29040467 90.76185124]，也就说 id 为 12-14 的三个同学的预测分数为55，54和91。

第一个差别比较大，id 为12的同学，明明考及格了，却被预测为不及格。

这是为什么呢？大家注意 id 为4的同学，这是一位学霸，他只用了20小时在学习上，却考出了第一名的好成绩。

回想一下线性回归的目标函数，我们不难发现，所有训练样本对于目标的贡献是平均的，因此，4号同学这种超常学霸的出现，在数据量本身就小的情况下，有可能影响整个模型。

这还是幸亏我们有历史记录，知道上次考试的成绩，如果 X 只包含“Hours Spent”，学霸同学根本就会带偏大多数的预测结果（自变量只有“Hours Spent”的线性回归模型会是什么样的？这个问题留给同学们自己去实践）。

那么我们看看用逻辑回归如何。用逻辑回归的时候，我们就不再是预测具体分数，而是预测这个学生本次能否及格了。

这样我们就需要对数据先做一下转换，把具体分数转变成是否合格，合格标志为1，不合格为0，然后再进行逻辑回归：

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.linear\_model **import** LinearRegression

**import** pandas **as** pd

*# Importing dataset*

data = pd.read\_csv('quiz.csv', delimiter=',')

used\_features = [ "Last Score", "Hours Spent"]

X = data[used\_features].values

scores = data["Score"].values

X\_train = X[:11]

X\_test = X[11:]

*# Logistic Regression – Binary Classification*

passed = []

**for** i **in** range(len(scores)):

**if**(scores[i] >= 60):

passed.append(1)

**else**:

passed.append(0)

y\_train = passed[:11]

y\_test = passed[11:]

classifier = LogisticRegression(C=1e5)

classifier.fit(X\_train, y\_train)

y\_predict = classifier.predict(X\_test)

print(y\_predict)

这次的输出就是[1 0 1]，对12、13、14号同学能否通过本次考试的判断是正确的。

### LR 处理多分类问题

LR 是用来做二分类的，但是如果我们面对的是多分类问题：样本标签的枚举值多于2个，还能用 LR 吗？

当然是可以的。我们可以把二分类问题分成多次来做。

假设你一共有 n 个标签（类别），也就是说可能的分类一共有 n 个。那么就构造 n 个 LR 分类模型，第一个模型用来区分 label\_1和 non-label \_1（即所有不属于 label\_1 的都归属到一类），第二个模型用来区分 label\_2 和 non-label \_2……, 第 n 个模型用来区分 label\_n 和 non-label \_n。

使用的时候，每一个输入数据都被这 n 个模型同时预测。最后哪个模型得出了 Positive 结果，就是该数据最终的结果。

如果有多个模型都得出了 Positive，那也没有关系。因为 LR 是一个回归模型，它直接预测的输出不仅是一个标签，还包括该标签正确的概率。那么对比几个 Positive 结果的概率，选最高的一个就是了。

例如，有一个数据，第一和第二个模型都给出了 Positive 结果，不过 label\_1模型的预测值是0.95，而 label\_2 的结果是0.78，那么当然是选高的，结果就是 label\_1。

说起原理来好像挺麻烦，好在 sklearn 已经为我们处理了多分类问题，我们用 sklearn 来做多分类的时候，只是需要把 y 准备好，其他的，都和做二分类一样就可以了。

比如还是上面的例子，现在我们需要区分：学生的本次成绩是优秀（>=85），及格，还是不及格。我们就在处理 y 的时候给它设置三个值：0 （不及格），1（及格）和2（优秀），然后再做 LR 分类就可以了。代码如下：

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.linear\_model **import** LinearRegression

**import** pandas **as** pd

*# Importing dataset*

data = pd.read\_csv('quiz.csv', delimiter=',')

used\_features = [ "Last Score", "Hours Spent"]

X = data[used\_features].values

scores = data["Score"].values

X\_train = X[:11]

X\_test = X[11:]

*# Logistic Regression - Multiple Classification*

level = []

**for** i **in** range(len(scores)):

**if**(scores[i] >= 85):

level.append(2)

**elif**(scores[i] >= 60):

level.append(1)

**else**:

level.append(0)

y\_train = level[:11]

y\_test = level[11:]

classifier = LogisticRegression(C=1e5)

classifier.fit(X\_train, y\_train)

y\_predict = classifier.predict(X\_test)

print(y\_predict)

测试集的输出是：[1 0 2] —— 12号及格，13号不及格，14号优秀，还是蛮准的。

### 附录

quiz.csv 文件：

Id,Last Score,Hours Spent,Score

1,90,117,89

2,85,109,78

3,75,113,82

4,98,20,95

5,62,116,61

6,36,34,32

7,87,120,88

8,89,132,92

9,60,83,52

10,72,92,65

11,73,112,71

12,56,143,62

13,57,97,52

14,91,119,93

[邀请好友一起学，获得 25% 返现奖励](https://gitbook.cn/m/mazi/columns/5bc6ac7442d7d32f50f19a98/topics/5bf28232fd72950cafdccd93)

[IMG_260](https://gitbook.cn/m/mazi/comp/column?columnId=5bc6ac7442d7d32f50f19a98%26tag=2#catalog)