Logistic 回归总结

作者：靠谱哥

微博：洞庭之子-Bing

（2013 年 11 月）

出处：http://blog.csdn.net/dongtingzhizi/article/details/15962797

1 引言

看了 Stanford 的 Andrew Ng 老师的机器学习公开课中关于 Logistic Regression 的讲解，然后又看了《机器学习实战》中的 Logistic Regression 部分，写下此篇学习笔记总结一下。

首先说一下我的感受，《机器学习实战》一书在介绍原理的同时将全部的算法用源代码实现，非常具有操作性，可以加深对算法的理解，但是美中不足的是在原理上介绍的比较粗略，很多细节没有具体介绍。所以，对于没有基础的朋友（包括我）某些地方可能看的一头雾水，需要查阅相关资料进行了解。所以说，该书还是比较适合有基础的朋友。

本文主要介绍以下三个方面的内容：

（1） Logistic Regression 的基本原理，分布在第二章中；

（2） Logistic Regression 的具体过程，包括：选取预测函数，求解 Cost 函数和 $J(\theta)$，梯度下降法求 $J(\theta)$ 的最小值，以及递归下降过程的向量化（vectorization），分布在第三章中；

（3） 对《机器学习实战》中给出的实现代码进行了分析，对阅读该书 Logistic Regression 部分遇到的疑惑进行了解释。没有基础的朋友在阅读该书的 Logistic Regression 部分时可能会觉得一头雾水，书中给出的代码很简单，但是怎么也跟书中介绍的理论联系不起来。也会有很多的疑问，比如：一般都是用梯度下降法求损失函数的最小值，为何这里用梯度上升法呢？

书中说用梯度上升发，为何代码实现时没见到求梯度的代码呢？这些问题在第三章和第四章中都会得到解答。

文中参考或引用内容的出处列在最后的“参考文献”中。文中所阐述的内容仅仅是我个人的理解，如有错误或疏漏，欢迎大家批评指正。下面进入正题。

2 基本原理

Logistic Regression 和 Linear Regression 的原理是相似的，按照我自己的理解，可以简单的描述为这样的过程：

1. 找一个合适的预测函数（Andrew Ng 的公开课中称为 hypothesis），一般表示为 h 函数，该函数就是我们需要找的分类函数，它用来预测输入数据的判断结果。这个过程时非常关键的，需要对数据有一定的了解或分析，知道或者猜测预测函数的“大概”形式，比如是线性函数还是非线性函数。

2. 构造一个 Cost 函数（损失函数），该函数表示预测的输出（h）与训练数据类别（y）之间的偏差，可以是二者之间的差（$h-y$）或者是其他的形式。综合考虑所有训练数据的“损失”，将 Cost 求和或者求平均，记为 $J(\theta)$ 函数，表示所有训练数据预测值与实际类别的偏差。

3. 显然，$J(\theta)$ 函数的值越小表示预测函数越准确（即 h 函数越准确），所以这一步需要做的是找到 $J(\theta)$ 函数的最小值。找函数的最小值有不同的方法，Logistic Regression 实现时有的是梯度下降法（Gradient Descent）。

3 具体过程

3.1 构造预测函数

Logistic Regression 虽然名字里带“回归”，但是它实际上是一种分类方法，用于两分类问题（即输出只有两种）。根据第二章中的步骤，需要先找到一个预测函数（h），显然，该函数的输出必须是两个值（分别代表两个类别）

，所以利用了 Logistic 函数（或称为 Sigmoid 函数），函数形式为：

$$g(z)=\frac{1}{1+e^{-z}} \tag{1}$$

对应的函数图像是一个取值在 0 和 1 之间的 S 型曲线（图 1）。

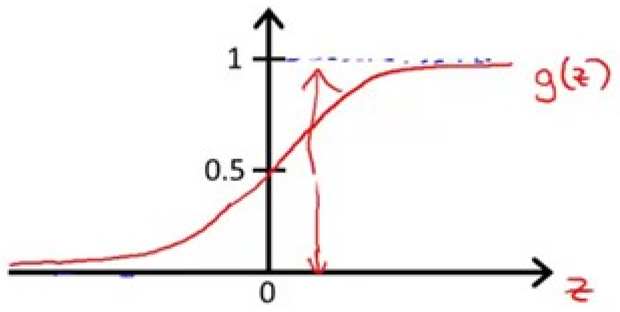

图 1

接下来需要确定数据划分的边界类型，对于图 2 和图 3 中的两种数据分布，显然图 2 需要一个线性的边界，而图 3 需要一个非线性的边界。接下来我们只讨论线性边界的情况。

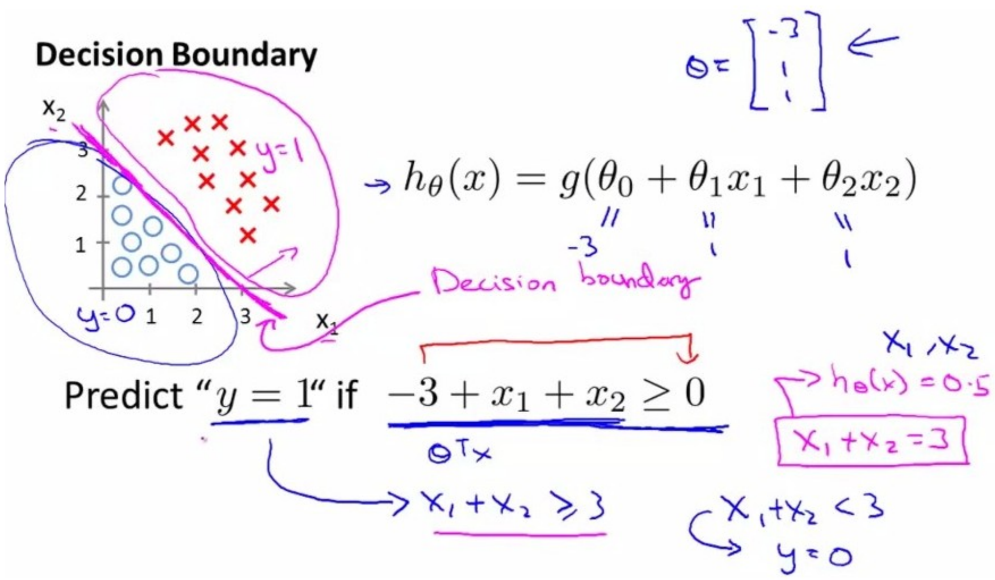

图 2

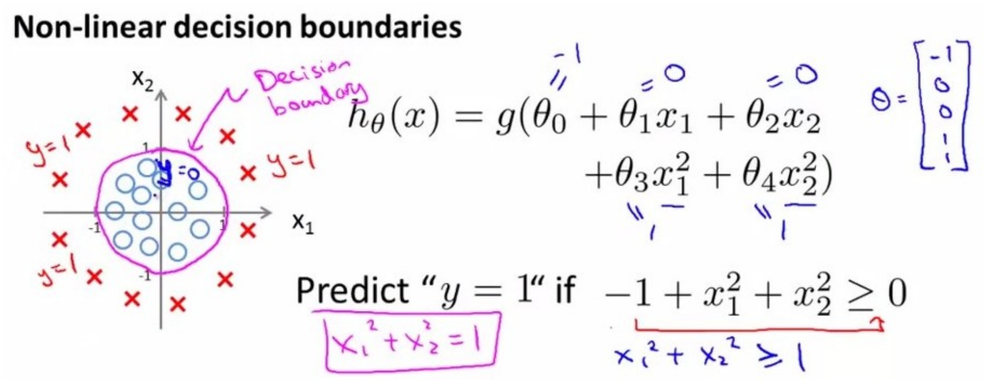

图 3

对于线性边界的情况，边界形式如下：

$$\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \sum_{i=0}^{n} \theta_i x_i = \theta^T x \qquad (2)$$

构造预测函数为：

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \qquad (3)$$

h_θ(x) 函数的值有特殊的含义，它表示结果取 1 的概率，因此对于输入 x 分类结果为类别 1 和类别 0 的概率分别为：

$$\begin{aligned} P(y=1 \mid x;\theta) &= h_\theta(x) \\ P(y=0 \mid x;\theta) &= 1 - h_\theta(x) \end{aligned} \qquad (4)$$

3.2 构造 Cost 函数

Andrew　Ng 在课程中直接给出了 Cost 函数及 $J(\theta)$ 函数如式（5）和（6），但是并没有给出具体的解释，只是说明了这个函数来衡量 h 函数预测的好坏是合理的。

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases} \qquad (5)$$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

（6）

实际上这里的 Cost 函数和$J(\theta)$函数是基于最大似然估计推导得到的。下面详细说明推导的过程。（4）式综合起来可以写成：

$$P(y \mid x;\theta) = (h_\theta(x))^{y}(1-h_\theta(x))^{1-y} \qquad (7)$$

取似然函数为：

$$L(\theta) = \prod_{i=1}^{m} P(y^{(\mathrm{i})} \mid x^{(\mathrm{i})};\theta)$$
$$= \prod_{i=1}^{m}(h_\theta(x^{(\mathrm{i})}))^{y^{(\mathrm{i})}}(1-h_\theta(x^{(\mathrm{i})}))^{1-y^{(\mathrm{i})}} \qquad (8)$$

对数似然函数为：

$$l(\theta) = \log L(\theta)$$
$$= \sum_{i=1}^{m}\left(y^{(i)}\log h_\theta(x^{(i)}) + \left(1-y^{(i)}\right)\log(1-h_\theta(x^{(i)}))\right) \qquad (9)$$

最大似然估计就是要求得使$l(\theta)$取最大值时的θ，其实这里可以使用梯度上升法求解，求得的θ就是要求的最佳参数。但是，在 Andrew Ng 的课程中将$J(\theta)$取为（6）式，即：

$$J(\theta) = -\frac{1}{m}l(\theta) \qquad (10)$$

因为乘了一个负的系数$-\frac{1}{m}$，所以$J(\theta)$取最小值时的θ为要求的最佳参数。

3.3 梯度下降法求$J(\theta)$的最小值

求$J(\theta)$的最小值可以使用梯度下降法，根据梯度下降法可得θ的更新过程：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta), \quad (j = 0 \ldots n) \tag{11}$$

式中α为学习步长，下面来求偏导：

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \frac{1}{h_\theta(\mathrm{x}^{(\mathrm{i})})} \frac{\partial}{\partial \theta_j} h_\theta(\mathrm{x}^{(\mathrm{i})}) - (1 - y^{(i)}) \frac{1}{1 - h_\theta(\mathrm{x}^{(\mathrm{i})})} \frac{\partial}{\partial \theta_j} h_\theta(\mathrm{x}^{(\mathrm{i})}) \right) \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \frac{1}{g(\theta^T x^{(\mathrm{i})})} - (1 - y^{(i)}) \frac{1}{1 - g(\theta^T x^{(\mathrm{i})})} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x^{(\mathrm{i})}) \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \frac{1}{g(\theta^T x^{(\mathrm{i})})} - (1 - y^{(i)}) \frac{1}{1 - g(\theta^T x^{(\mathrm{i})})} \right) g(\theta^T x^{(\mathrm{i})}) \left(1 - g(\theta^T x^{(\mathrm{i})})\right) \frac{\partial}{\partial \theta_j} \theta^T x^{(\mathrm{i})} \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} \left(1 - g(\theta^T x^{(\mathrm{i})})\right) - (1 - y^{(i)}) g(\theta^T x^{(\mathrm{i})}) \right) x_j^{(\mathrm{i})} \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{i} - g(\theta^T x^{(\mathrm{i})}) \right) x_j^{(\mathrm{i})} \\
&= -\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(\mathrm{x}^{(\mathrm{i})}) \right) x_j^{(\mathrm{i})} \\
&= \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(\mathrm{x}^{(\mathrm{i})}) - y^{(i)} \right) x_j^{(\mathrm{i})}
\end{aligned}$$

（12）

上式求解过程中用到如下的公式：

$$\begin{aligned}
f(x) &= \frac{1}{1 + e^{g(\mathrm{x})}} \\
\frac{\partial}{\partial x} f(x) &= \frac{1}{\left(1 + e^{g(\mathrm{x})}\right)^2} e^{g(\mathrm{x})} \frac{\partial}{\partial x} g(x) \\
&= \frac{1}{1 + e^{g(\mathrm{x})}} \frac{e^{g(\mathrm{x})}}{1 + e^{g(\mathrm{x})}} \frac{\partial}{\partial x} g(x) \\
&= f(x)\left(1 - f(x)\right) \frac{\partial}{\partial x} g(x)
\end{aligned} \tag{13}$$

因此，（11）式的更新过程可以写成：

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(\mathrm{x}^{(\mathrm{i})}) - y^{(i)} \right) x_j^{(\mathrm{i})}, \quad (j = 0 \ldots n) \qquad (14)$$

因为式中α本来为一常量，所以一般将$\frac{1}{m}$省略，所以最终的θ更新过程为：

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta(\mathrm{x}^{(\mathrm{i})}) - y^{(i)} \right) x_j^{(\mathrm{i})}, \quad (j = 0 \ldots n) \qquad (15)$$

另外，补充一下，3.2 节中提到求得$l(\theta)$取最大值时的θ也是一样的，用梯度上升法求（9）式的最大值，可得：

$$\begin{aligned} \theta_j &:= \theta_j + \alpha \frac{\partial}{\partial \theta_j} \ell(\theta) \\ &= \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(\mathrm{x}^{(\mathrm{i})}) \right) x_j^{(\mathrm{i})}, \end{aligned} \quad (j = 0 \ldots n) \qquad (16)$$

观察上式发现跟（14）是一样的，所以，采用梯度上升发和梯度下降法是完全一样的，这也是《机器学习实战》中采用梯度上升法的原因。

3.4 梯度下降过程向量化

关于θ更新过程的 vectorization，Andrew Ng 的课程中只是一带而过，没有具体的讲解。

《机器学习实战》连 Cost 函数及求梯度等都没有说明，所以更不可能说明 vectorization 了。但是，其中给出的实现代码确是实现了 vectorization 的，图 4 所示代码的 32 行中 weights（也就是θ）的更新只用了一行代码，直接通过矩阵或者向量计算更新，没有用 for 循环，说明确实实现了 vectorization，具体代码下一章分析。

文献[3]中也提到了 vectorization，但是也是比较粗略，很简单的给出 vectorization 的结果为：

$$\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(\mathrm{x}^{(\mathrm{i})}) - y^{(i)} \right) x^{(\mathrm{i})}, \quad (j = 0 \ldots n) \qquad (17)$$

且不论该更新公式正确与否，这里的$\sum_{i=1}^{m}(\ldots)$是一个求和的过程，显然需要一个 for 语句循环 m 次，所以根本没有完全的实现 vectorization，不像《机器学习实战》的代码中一条语句就可以完成θ的更新。

下面说明一下我理解《机器学习实战》中代码实现的 vectorization 过程。

约定训练数据的矩阵形式如下，x 的每一行为一条训练样本，而每一列为不同的特称取值：

$$x = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \ldots \\ x^{(\mathrm{m})} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \ldots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \ldots & x_n^{(2)} \\ \ldots & \ldots & \ldots & \ldots \\ x_0^{(\mathrm{m})} & x_1^{(\mathrm{m})} & \ldots & x_n^{(\mathrm{m})} \end{bmatrix} \quad , \quad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \ldots \\ y^{(\mathrm{m})} \end{bmatrix} \qquad (18)$$

约定待求的参数θ的矩阵形式为：

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \ldots \\ \theta_n \end{bmatrix} \qquad (19)$$

先求$x\cdot\theta$并记为A：

$$A = x\cdot\theta = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_n^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \dots & x_n^{(2)} \\ \dots & \dots & \dots & \dots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix} = \begin{bmatrix} \theta_0 x_0^{(1)} + \theta_1 x_1^{(1)} + \dots + \theta_n x_n^{(1)} \\ \theta_0 x_0^{(2)} + \theta_1 x_1^{(2)} + \dots + \theta_n x_n^{(2)} \\ \dots \\ \theta_0 x_0^{(m)} + \theta_1 x_1^{(m)} + \dots + \theta_n x_n^{(m)} \end{bmatrix}$$

（20）

求 $h_\theta(\mathrm{x}) - y$ 并记为 E：

$$E = h_\theta(\mathrm{x}) - y = \begin{bmatrix} g\left(A^{(1)}\right) - y^{(1)} \\ g\left(A^{(2)}\right) - y^{(2)} \\ \\ g\left(A^{(m)}\right) - y^{(m)} \end{bmatrix} = \begin{bmatrix} e^{(1)} \\ e^{(2)} \\ \dots \\ e^{(m)} \end{bmatrix} = g\left(A\right) - y \quad (21)$$

$g\left(A\right)$的参数 A 为一列向量，所以实现 g 函数时要支持列向量作为参数，并返回列向量。由上式可知 $h_\theta(\mathrm{x}) - y$ 可以由 $g\left(A\right) - y$ 一次计算求得。

再来看一下（15）式的 θ 更新过程，当 $j = 0$ 时：

$$\begin{aligned} \theta_0 &:= \theta_0 - \alpha \sum_{i=1}^{m} \left(h_\theta(\mathrm{x}^{(\mathrm{i})}) - y^{(i)}\right) x_0^{(\mathrm{i})} \\ &= \theta_0 - \alpha \sum_{i=1}^{m} e^{(\mathrm{i})} x_0^{(\mathrm{i})} \\ &= \theta_0 - \alpha \cdot \left(x_0^{(1)}, x_0^{(2)}, \dots, x_0^{(\mathrm{m})}\right) \cdot E \end{aligned} \quad (22)$$

同样的可以写出 θ_j，

$$\theta_j := \theta_j - \alpha \cdot \left(x_j^{(1)}, x_j^{(2)}, \dots, x_j^{(\mathrm{m})}\right) \cdot E \quad (23)$$

综合起来就是：

$$\begin{bmatrix}\theta_0\\ \theta_1\\ \cdots\\ \theta_n\end{bmatrix} := \begin{bmatrix}\theta_0\\ \theta_1\\ \cdots\\ \theta_n\end{bmatrix} - \alpha \cdot \begin{bmatrix} x_0^{(1)}, x_0^{(2)}, \ldots, x_0^{(m)} \\ x_1^{(1)}, x_1^{(2)}, \ldots, x_1^{(m)} \\ \\ x_n^{(1)}, x_n^{(2)}, \ldots, x_n^{(m)} \end{bmatrix} \cdot E \tag{24}$$

$$= \theta - \alpha \cdot x^T \cdot E$$

综上所述，vectorization 后 θ 更新的步骤如下：

（1） 求 $A = x \cdot \theta$；

（2） 求 $E = g\left(A\right) - y$；

（3） 求 $\theta := \theta - \alpha \cdot x^T \cdot E$。

4 代码分析

图 4 中是《机器学习实战》中给出的部分实现代码。

```
def sigmoid(inX):
    return 1.0/(1+exp(-inX))

def gradAscent(dataMatIn, classLabels):
    dataMatrix = mat(dataMatIn)             #convert to NumPy matrix
    labelMat = mat(classLabels).transpose() #convert to NumPy matrix
    m,n = shape(dataMatrix)
    alpha = 0.001
    maxCycles = 500
    weights = ones((n,1))
    for k in range(maxCycles):              #heavy on matrix operations
        h = sigmoid(dataMatrix*weights)     #matrix mult
        error = (labelMat - h)              #vector subtraction
        weights = weights + alpha * dataMatrix.transpose()* error #matrix mult
    return weights
```

图 4

sigmoid 函数就是前文中的 $g(z)$ 函数，参数 inX 可以是向量，因为程序中使用了 Python 的 numpy。

gradAscent 函数是梯度上升的实现函数，参数 dataMatin 和 classLabels 为训练数据，23 和 24 行对训练数据做了处理，转换成 numpy 的矩阵类型，同时将横向量的 classlabels 转换成列向量 labelMat，此时的 dataMatrix 和 labelMat 就是（18）式中的x和y。alpha 为学习步长，maxCycles 为迭代次数。weights 为 n 维（等于x的列数）列向量，就是（19）式中的θ。

29 行的 for 循环将更新θ的过程迭代 maxCycles 次，每循环一次更新一次θ。对比 3.4 节最后总结的向量化的θ更新步骤，30 行相当于求了$A = x\cdot\theta$和$g(A)$，31 行相当于求了$E = g(A) - y$，32 行相当于求$\theta := \theta - \alpha\cdot x^T\cdot E$。所以这三行代码实际上与向量化的$\theta$更新步骤是完全一致的。

总结一下，从上面代码分析可以看出，虽然只有十多行的代码，但是里面却隐含了太多的细节，如果没有相关基础确实是非常难以理解的。相信完整的阅读了本文，就应该没有问题了！^_^。

【参考文献】

[1] 《机器学习实战》——【美】Peter Harington

[2] Stanford 机器学习公开课（https://www.coursera.org/course/ml）

[3] http://blog.csdn.net/abcjennifer/article/details/7716281

[4] http://www.cnblogs.com/tornadomeet/p/3395593.html

[5] http://blog.csdn.net/moodytong/article/details/9731283

[6] http://blog.csdn.net/jackie_zhu/article/details/8895270