逻辑回归与计算中的向量化思想

引子

- 这部分总结的有点晚了，不过在熟悉机器学习体系和几个其它模型后，再回过头来看逻辑回归，发现从原理到思想都有比较好的理解。
- 理论主要参考NG的斯坦福课程和李航老师的《统计学习方法》，代码实现参见《机器学习实战》。
- 向量化计算的思想在逻辑回归代码中体现的很好，利用矩阵计算上的便利性来解决复杂的循环过程。但由于《实战》书中的原理推导及其吝惜笔墨（与其简洁优美的代码对比鲜明），这里做一下补充。
- 感谢洞庭之子这篇博客，解决了我的很多疑问，很佩服作者的思路和严谨的推导,但这篇博客中的有些写法不规范，容易引起误解。

逻辑回归

思想

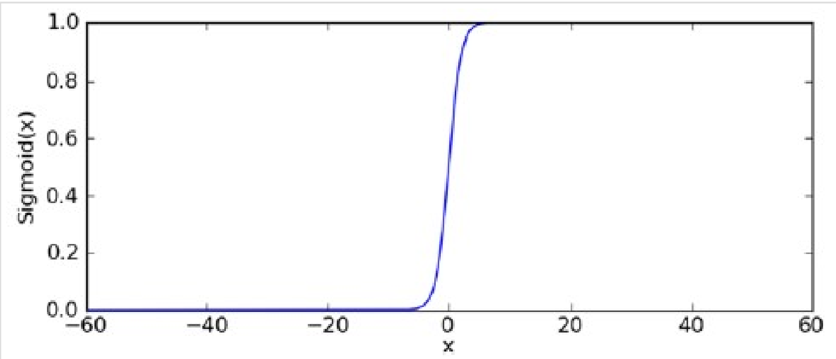
- **根据数据对分类边界线建立回归公式：**与感知机乃至SVM大同小异，都是寻找一个`超平面`将数据集分为两部分。基于如此，逻辑回归一般只能处理两分类问题，同时两个类别线性可分。对于`多分类问题`，还是老思想，化用二分类（目标类为一类，剩余唯一类），构建多个分类器，寻找概率最大的那个类作为分类结果。
- **通过分类函数（sigmoid函数）寻找分类超平面**： 具体sigmoid函数相关的内容下面有详细叙述.
- **判别模型的老思路**：假设特征系数θ，构造预测函数 ——> 构造损失函数 ——> 求解最优化问题：寻找使损失函数最小時的特征系数θ ——> 得到分类器（即超平面）。
- **优缺点：** 计算简单，训练分类器后计算量小;准确度有限，容易欠拟合，只针对二分类问题。

分类函数Sigmoid

- 逻辑回归选择`近似于阶越函数`的Sigmoid函数作为分类函数：

$$\sigma(z_i) = \frac{1}{1+e^{-z}} \quad , (z_i = \theta_0 x_i^{(0)} + \theta_1 x_i^{(1)} + .. + \theta_n x_i^{(m)})$$

- 函数图像：



- θ为特征系数向量, 每个特征都有一个特征系数θ_n。另外，$x_i^{(j)}$ 为输入向量x_i的第j个分量,
- 作用：1. 逻辑回归的分类函数。 2. 将样本映射到0-1区间，进而巧妙地将数据到分界线的距离转化为概率，然后通过最大斯然估计等方法求解，这一些后面会谈到。

问题求解

求解的过程这里简单讲讲，具体内容（比如阶梯求导结果等）不在赘述，可以参阅参考博客。
判别模型的基本套路：
预测函数 ——> 构造损失函数or最大斯然估计求发生概率 ——> 求解最优化问题：

- 1、预测函数：

$$h_\theta(x) = \sigma(z_i) = \frac{1}{1 + e^{-z}} \quad ,(z_i = \theta_0 x_i^{(0)} + \theta_1 x_i^{(1)} + ..+\theta_n x_i^{(m)})$$

sigmoid的函数值可以表示成分类概率：

$$P(y = 1|x, \theta) = h_\theta(x)$$
$$P(y = 0|x, \theta) = 1 - h_\theta(x)$$

- 2、通过最大斯然估计求发生概率，单个样本发生的概率为：

$$P(y|x, \theta) = (h_\theta(x))^y (1 - h_\theta(x))^{1-y}$$

最大似然估计，让所有样本的发生概率最大化，取似然函数：

$$L(\theta) = \prod_{i=1}^{n} P(y_i|x_i, \theta) = \prod_{i=1}^{n} (h_\theta(x_i))^{y_i} (1 - h_\theta(x_i))^{1-y_i}$$

取对数：

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{n} (y_i(\log h_\theta(x_i) + (1 - y_i) \log(1 - h_\theta(x_i)) \end{aligned}$$

到这里，我们就可以用最优化方法求解$l(\theta)$，本例用的是梯度上升法。
在斯坦福ML的课程中，取$J(\theta) = -\frac{1}{m} l(\theta)$,构建损失函数，然后利用梯度下降法求解$\theta$，本质是完全一样的。

- 3、借助梯度上升法求解最优值：

$$\begin{aligned} \theta_j &= \theta_j + \alpha \frac{\delta}{\delta\theta_j} J(\theta) \\ &= \theta_j + \alpha \sum_{i=1}^{n} (y_i - h_\theta(x_i))x_i^{(j)} \end{aligned}$$

（特征维数$j = 0...m$,、α为学习步长 ）

$$\begin{aligned} \frac{\delta}{\delta\theta_j} J(\theta) &= \sum_{i=1}^{n} [y_i \frac{1}{h_\theta(x_i)} \frac{\delta}{\delta\theta_j} h_\theta(x_i) - (1 - y_i) \frac{1}{1 - h_\theta(x_i)} \frac{\delta}{\delta\theta_j} h_\theta(x_i)] \\ &= ..... \\ &= \theta_j + \alpha \sum_{i=1}^{n} (y_i - h_\theta(x_i))x_i^{(j)} \end{aligned}$$

这里省去了大部分求偏导的过程，详细的求解步骤，可以参见参考博客。博文虽用的是梯度下降法，但每一步的求偏导数过程是完全一致的。

计算中的向量化思想

- 在求解最优化方法時，将数据向量化，用矩阵的方式计算，是一种很好的思想。
- 借助numpy等包，使得以前针对单个数值编写的方法，对矩阵也有了很好的支持度。
- 《机器学习实战》的实现代码中，使用梯度上升法求解θ時，就用了向量化思想，用矩阵乘法代替了循环。但书中直接给出了迭代求θ的公式，缺少了如上节类似的推导，初读还是有些令人费解。

上节已知求θ每一步的更新过程:

$$\begin{aligned}\theta_j &= \theta_j + \alpha \frac{\delta}{\delta\theta_j} J(\theta) \\ &= \theta_j + \alpha \sum_{i=1}^{n} (y_i - h_\theta(x_i)) x_i^{(j)}\end{aligned}$$

- 1、首先我们把特征系数θ用m×1的列向量表示。(假设有n个输入向量，特征共有m维,$\theta_0 x_0$当作常量偏移):

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_m \end{bmatrix}$$

输入数据用n×m的矩阵表示:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1^{(0)}, x_1^{(1)} \dots x_1^{(m)} \\ x_2^{(0)}, x_2^{(1)} \dots x_2^{(m)} \\ \dots \\ x_n^{(0)}, x_n^{(1)} \dots x_n^{(m)} \end{bmatrix}$$

所以$z_n = \theta_0 x_n^{(0)} + \theta_1 x_n^{(1)} + .. + \theta_n x_n^{(m)}$ 可以用$X \cdot \theta$得出的n×1列向量来表示:

$$Z = \begin{bmatrix} \theta_0 x_1^{(0)} + \theta_1 x_1^{(1)} + \dots + \theta_m x_1^{(m)} \\ \theta_0 x_2^{(0)} + \theta_1 x_2^{(1)} + \dots + \theta_m x_2^{(m)} \\ \dots \\ \theta_0 x_n^{(0)} + \theta_1 x_n^{(1)} + \dots + \theta_m x_n^{(m)} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \dots \\ z_n \end{bmatrix}$$

用矩阵乘法的列向量分解思想来说，$X \cdot \theta$表示X中的每一个列向量以系数θ进行 `线性组合`，符合公式的含义。

- 2、我们再用列向量$Y = (y_1, y_2 .. y_n)^T$ 表示输入数据的类别，然后我们对Z进行sigmoid函数运算，所以更新过程公式中$(y_i - h_\theta(x_i))$ 可以用n×1的列向量E表示:

$$E = \begin{bmatrix} y_1 - \sigma(z_1) \\ y_2 - \sigma(z_2) \\ \dots \\ y_n - \sigma(z_n) \end{bmatrix} = \begin{bmatrix} e_1 \\ e_2 \\ \dots \\ e_n \end{bmatrix}$$

(这里的E代表误差error)

- 3、最后我们求解整个式子$\theta_j = \theta_j + \alpha \sum_{i=1}^{n} (y_i - h_\theta(x_i))x_i^{(j)}$. 式中的连加同样可以通过矩阵乘法解决。
  将X 转置：

$$X^T = [x_1, x_2, \dots x_n] = \begin{bmatrix} x_1^{(0)}, x_2^{(0)} \dots x_n^{(0)} \\ x_1^{(1)}, x_2^{(1)} \dots x_n^{(1)} \\ \dots\dots \\ x_1^{(m)}, x_2^{(m)} \dots x_n^{(m)} \end{bmatrix}$$

转置后矩阵是m×n的，仔细观察一下转置后的X，每一个列向量是一条输入数据，而**每一个行向量是所有输入数据在某一个特征维度上的记录**，一共有m个行向量。
所以，我们同时用每一个行向量·列向量E ,用矩阵乘法即：

$$X^T \cdot E = \begin{bmatrix} x_1^{(0)} e_1 + x_2^{(0)} e_2 + \dots + x_n^{(0)} e_n \\ x_1^{(1)} e_1 + x_2^{(1)} e_2 + \dots + x_n^{(1)} e_n \\ \dots\dots \\ x_1^{(m)} e_1, x_2^{(m)} e_2 + \dots + x_n^{(m)} e_n \end{bmatrix}$$

(m*1)

- 4、已知增长系数α，然后我们就可以用矩阵计算进行一次梯度上升迭代：

$$\theta_{n+1} = \theta_n + \alpha \cdot X^T \cdot E = \begin{bmatrix} \theta_{n+1}^{(0)} \\ \theta_{n+1}^{(1)} \\ \dots \\ \theta_{n+1}^{(m)} \end{bmatrix}$$

1. 设置合适的迭代次数，求得最终的特征系数θ,我们就得到了训练好的判别函数。

小结：

- 看完我们会发现，逻辑回归也是基于简单的线性分类思想，只不过逻辑回归通过一个契合线性分类的sigmoid函数，将数据到分界限的距离巧妙地投影到了0-1区间，进而可以将距离转化为概率，通过最大斯然估计求解。
- 向量化的思想其实充斥着机器学习的各个角落。就像MIT线性代数公开课中说的，矩阵并不是生来存在的，而是人们后天发明用来方便计算的产物。将数据向量化，通过矩阵运算求解，是解决问题的必经之路。

这次公式比较多，处理markdown语法和$\LaTeX$ 的冲突時花了不少时间，不过也因此找到一个好方法，不用修改node.js的配置就可以处理好语法冲突，只需要在`公式的前后用raw标签注释就可以了`，原理应该是通过标签屏蔽了markdown语法的解释器，前端不是很懂，不过很有效~

来

源： http://robinzheng.me/2015/08/13/%E9%80%BB%E8%BE%91%E5%9B%9E%E5%BD%92%E4%B8%8E%E8%AE%A1%E7%AE%97%E4%B8%AD%E7%9A%84%E5%90%91%E9%87%8F%E5%8C%96%E6%80%9D%E6%83%B3/