机器学习实战

Stephen CUI[1]

February 17, 2023

[1]cuixuanStephen@gmail.com

Contents

Part I

分类

前两部分主要探讨监督学习（supervised learning）。在监督学习的过程中，我们只需要给定输入样本集，机器就可以从中推演出指定目标变量的可能结果。

监督学习一般使用两种类型的目标变量：标称型和数值型。标称型目标变量的结果只在有限目标集中取值，如真与假、动物分类集合 { 爬行类、鱼类、哺乳类、两栖类 } ；数值型目标变量则可以从无限的数值集合中取值，如 0.100、42.001、1000.743 等。数值型目标变量主要用于回归分析。

Chapter 1

Logistic回归

假设现在有一些数据点，我们用一条直线对这些点进行拟合（该线称为最佳拟合直线），这个拟合过程就称作回归。利用Logistic回归进行分类的主要思想是：根据现有数据对分类边界线建立回归公式，以此进行分类。这里的“回归”一词源于最佳拟合，表示要找到最佳拟合参数集。

1.1 基于 Logistic 回归和 Sigmoid 函数的分类

Logistic回归

优点：计算代价不高，易于理解和实现。
缺点：容易欠拟合，分类精度可能不高。
适用数据类型：数值型和标称型数据。

我们想要的函数应该是，能接受所有的输入然后预测出类别。例如，在两个类的情况下，上述函数输出0或1。或许你之前接触过具有这种性质的函数，该函数称为**海维塞德阶跃函数**（Heaviside step function），或者直接称为单位阶跃函数。然而，海维塞德阶跃函数的问题在于：该函数在跳跃点上从0瞬间跳跃到1，这个瞬间跳跃过程有时很难处理。幸好，另一个函数也有类似的性质[1]，且数学上更易处理，这就是Sigmoid函数。Sigmoid函数具体的计算公式如下：

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

为了实现Logistic回归分类器，我们可以在每个特征上都乘以一个回归系数，然后把所有的结果值相加，将这个总和代入Sigmoid函数中，进而得到一个范围在0 1之间的数值。任何大于0.5的数据被分入1类，小于0.5即被归入0类。所以，Logistic回归也可以被看成是一种概率估计。

1.2 基于最优化方法的最佳回归系数确定

Sigmoid函数的输入记为z，由下面公式得出：

$$z = w_0x_0 + w_1x_1 + w_2x_3 + \cdots + w_nx_n$$

[1]Sigmoid函数是一种阶跃函数（step function）。在数学中，如果实数域上的某个函数可以用半开区间上的指示函数的有限次线性组合来表示，那么这个函数就是阶跃函数。而数学中指示函数（indicator function）是定义在某集合X上的函数，表示其中有哪些元素属于某一子集A。

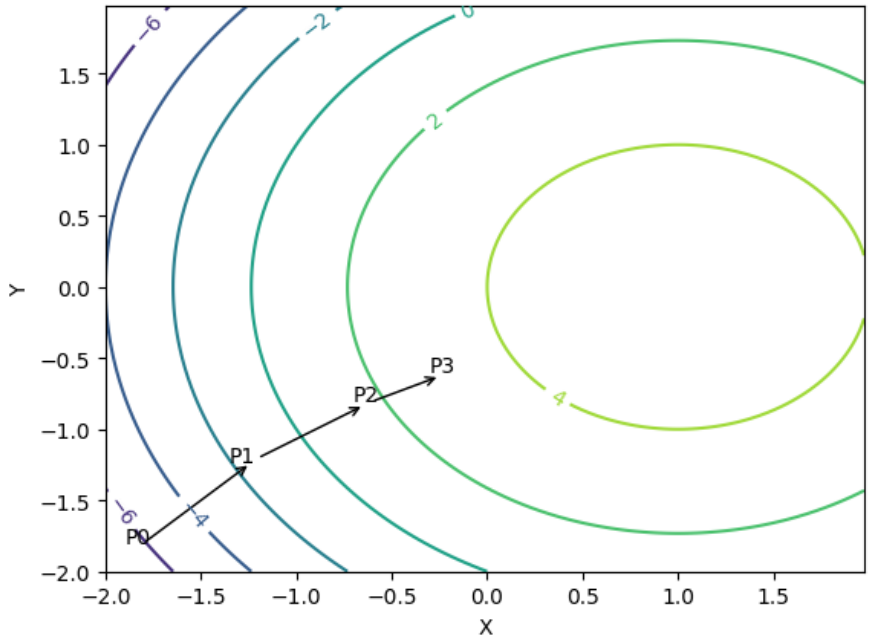

Figure 1.1: Gradient descent

如果采用向量的写法，上述公式可以写成$z = \boldsymbol{w}^T\boldsymbol{x}$，它表示将这两个数值向量对应元素相乘然后全部加起来即得到z值。其中的向量$\boldsymbol{x}$是分类器的输入数据，向量$\boldsymbol{w}$也就是我们要找到的最佳参数（系数），从而使得分类器尽可能地精确。

1.2.1 梯度上升法

梯度上升法基于的思想是：要找到某函数的最大值，最好的方法是沿着该函数的梯度方向探寻。如果梯度记为∇，则函数$f(x,y)$的梯度由下式表示：

$$\nabla f(x,y) = \begin{pmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{pmatrix}$$

这个梯度意味着要沿x的方向移动$\frac{\partial f(x,y)}{\partial x}$，沿$y$的方向移动$\frac{\partial f(x,y)}{\partial y}$。其中，函数$f(x,y)$必须要在待计算的点上有定义并且可微。

Figure 1.1中的梯度上升算法沿梯度方向移动了一步。可以看到，梯度算子总是指向函数值增长最快的方向。这里所说的是移动方向，而未提到移动量的大小。该量值称为步长，记做α。用向量来表示的话，梯度上升算法的迭代公式如下：

$$\boldsymbol{w} := \boldsymbol{w} + \alpha\nabla_{\boldsymbol{w}} f(w)$$

梯度下降

梯度下降算法，它与这里的梯度上升算法是一样的，只是公式中的加法需要变成减法。因此，对应的公式可以写成

$$\boldsymbol{w} := \boldsymbol{w} - \alpha\nabla_{\boldsymbol{w}} f(w)$$

梯度上升算法用来求函数的最大值，而梯度下降算法用来求函数的最小值。

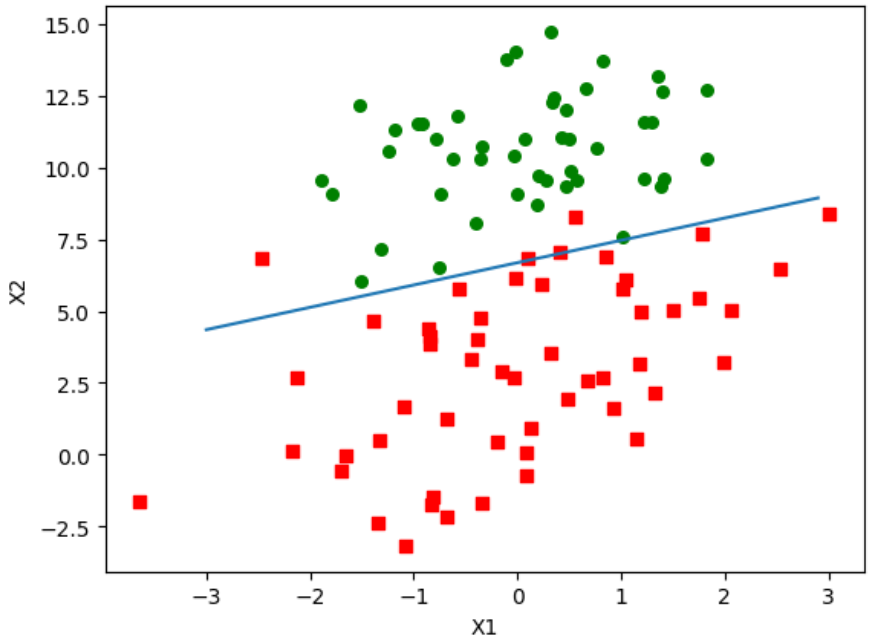

Figure 1.2: The logistic regression best-fit line after 500 cycles of gradient ascent

1.2.2 训练算法：使用梯度上升找到最佳参数

梯度上升法的伪代码如下：

Algorithm 1: 梯度上升法

每个回归系数初始化为1;
repeat
  计算整个数据集的梯度;
  使用alpha × gradient更新回归系数的向量;
until *R*次;
return 回归系数

1.2.3 分析数据：画出决策边界

设置了sigmoid函数为0。0是两个分类（类别1和类别0）的分界处。因此，我们设定 $0 = w_0x_0 + w_1x_1 + w_2x_2$，然后解出X2和X1的关系式（即分隔线的方程，注意$x_0 = 1$）。

1.2.4 训练算法：随机梯度上升

梯度上升算法在每次更新回归系数时都需要遍历整个数据集，该方法在处理100个左右的数据集时尚可，但如果有数十亿样本和成千上万的特征，那么该方法的计算复杂度就太高了。一种改进方法是一次仅用一个样本点来更新回归系数，该方法称为随机梯度上升算法。由于可以在新样本到来时对分类器进行增量式更新，因而随机梯度上升算法是一个在线学习算法。与“在线学习”相对应，一次处理所有数据被称作是“批处理”。

随机梯度上升算法可以写成如下的伪代码：

Algorithm 2: 随机梯度上升

所有回归系数初始化为1;
for 数据集中每个样本 **do**
  | 计算该样本的梯度使用alpha × gradient更新回归系数值
end
return返回回归系数值

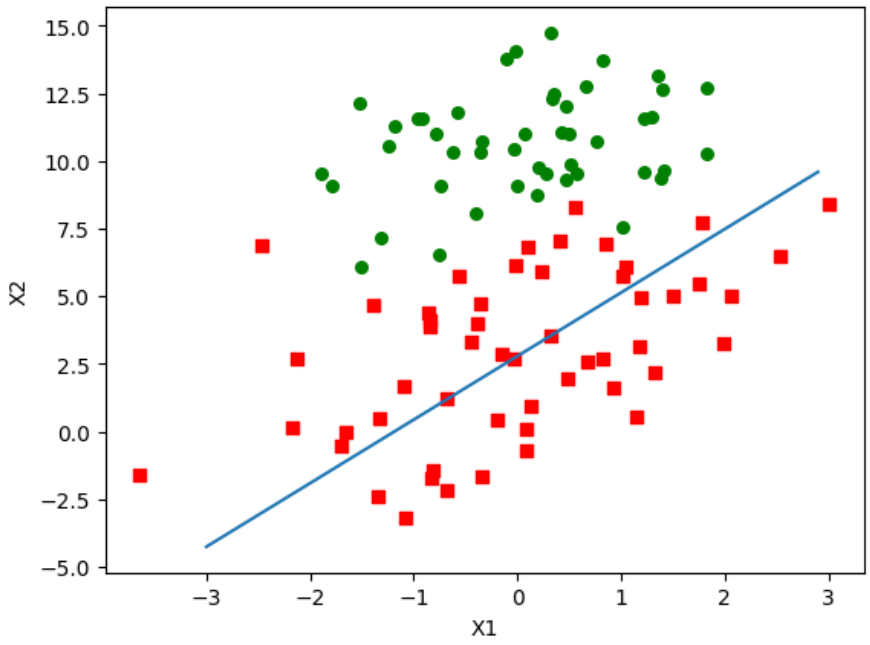

Figure 1.3: Our simple dataset with solution from stochastic gradient ascent

随机梯度上升算法与梯度上升算法在代码上很相似，但也有一些区别：第一，后者的变量h和误差error都是向量，而前者则全是数值；第二，前者没有矩阵的转换过程，所有变量的数据类型都是NumPy数组。

直接比较Figure 1.3和Figure 1.2的代码结果是不公平的，后者的结果是在整个数据集上迭代了500次才得到的。一个判断优化算法优劣的可靠方法是看它是否收敛，也就是说参数是否达到了稳定值，是否还会不断地变化？

Figure 1.4展示了随机梯度上升算法在200次迭代过程中回归系数的变化情况。在200次迭代后没有出现收敛的信息，产生这种现象的原因是存在一些不能正确分类的样本点（数据集并非线性可分），在每次迭代时会引发系数的剧烈改变。我们期望算法能避免来回波动，从而收敛到某个值。另外，收敛速度也需要加快。对于Figure 1.4存在的问题，可以通过改进的随机梯度上升算法来解决。

一方面，alpha在每次迭代的时候都会调整，这会缓解图5-6上的数据波动或者高频波动。另外，虽然alpha会随着迭代次数不断减小，但永远不会减小到0，这是因为中还存在一个常数项。必须这样做的原因是为了保证在多次迭代之后新数据仍然具有一定的影响。如果要处理的问题是动态变化的，那么可以适当加大上述常数项，来确保新的值获得更大的回归系数。另一点值得注意的是，在降低alpha的函数中，alpha每次减少$1/(j+i)$，其中j是迭代次数，i是样本点的下标。这样当$j << max(i)$时，alpha就不是严格下降的。避免参数的严格下降也常见于模拟退火算法等其他优化算法中。

第二个改进的地方在处，这里通过随机选取样本来更新回归系数。这种方法将减少周期性的波动

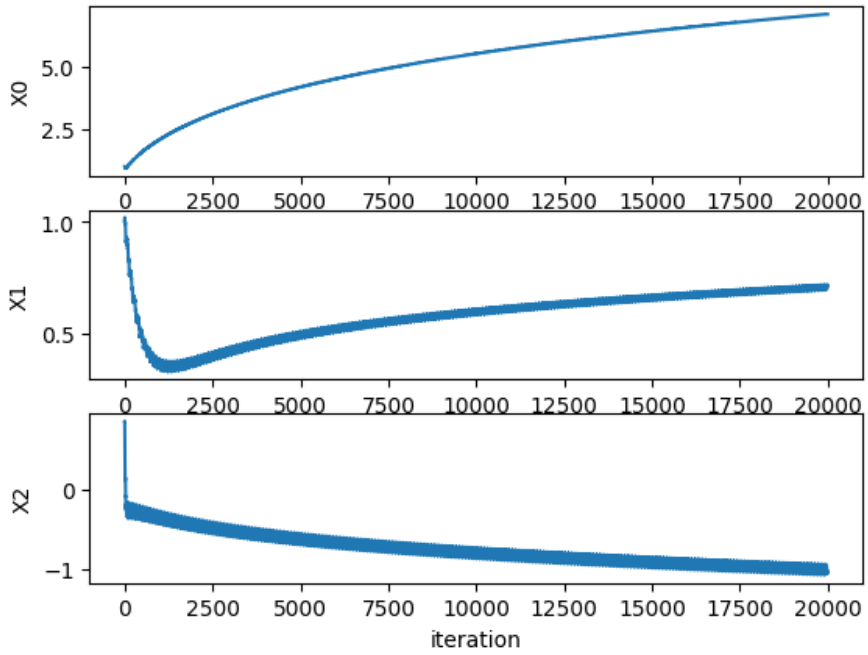

Figure 1.4: Weights versus iteration number for one pass through the dataset

（如Figure 1.4中的波动）。

改进算法还增加了一个迭代次数作为第3个参数。如果该参数没有给定的话，算法将默认迭代150次。如果给定，那么算法将按照新的参数值进行迭代。

比较Figure 1.5和Figure 1.4可以看到两点不同。第一点是，图Figure 1.5中的系数没有像Figure 1.4里那样出现周期性的波动，这归功于stocGradAscent1()里的样本随机选择机制；第二点是，图Figure 1.5的水平轴比Figure 1.4短了很多，这是由于`stocGradAscent1()`可以收敛得更快。这次我们仅仅对数据集做了20次遍历，而之前的方法是200次。

下面看看在同一个数据集上的分类效果。

程序运行之后应该能看到类似Figure 1.6的结果图。该分隔线达到了与`GradientAscent()`差不多的效果，但是所使用的计算量更少。

1.3 示例：从疝气病症预测病马的死亡率

这里的数据[2]包含368个样本和28个特征，包含了医院检测马疝病的一些指标，有的指标比较主观，有的指标难以测量，例如马的疼痛级别。另外需要说明的是，除了部分指标主观和难以测量外，该数据还存在一个问题，数据集中有30%的值是缺失的。

1.3.1 准备数据：处理数据中的缺失值

数据中的缺失值是个非常棘手的问题，有很多文献都致力于解决这个问题。那么，数据缺失究竟带来了什么问题？假设有100个样本和20个特征，这些数据都是机器收集回来的。若机器上的某个传感器损坏导致一个特征无效时该怎么办？此时是否要扔掉整个数据？这种情况下，另外19个特征怎么办？它

[2]Dataset retrieved from UCI Machine Learning Repository on 11/1/2010 (http://archive.ics.uci.edu/ml/datasets/Horse+Colic). Data originally created by Mary McLeish and Matt Cecile, Department of Computer Science, University of Guelph, Guelph, Ontario, Canada N1G 2W1.

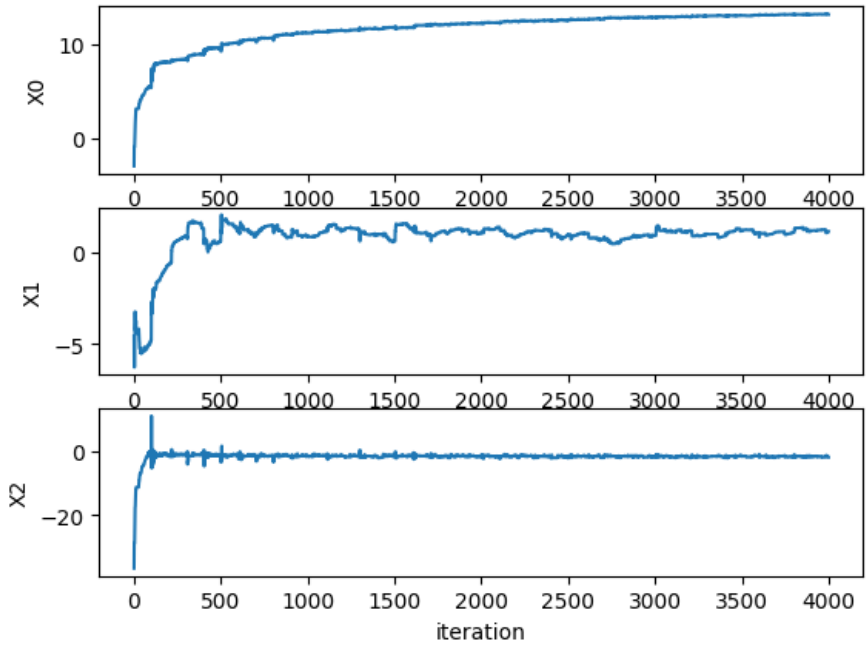

Figure 1.5: Coefficient convergence in with random vector selection and decreasing alpha

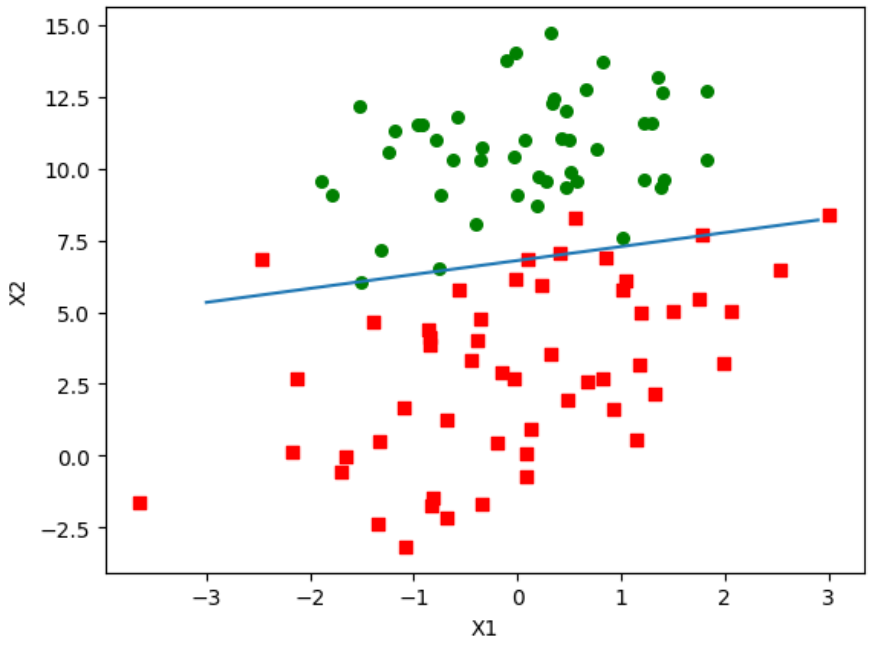

Figure 1.6: The coefficients with the improved stochastic gradient descent algorithm

们是否还可用？答案是肯定的。因为有时候数据相当昂贵，扔掉和重新获取都是不可取的，所以必须采用一些方法来解决这个问题。

下面给出了一些可选的做法：

- 使用可用特征的均值来填补缺失值；
- 使用特殊值来填补缺失值，如-1；
- 忽略有缺失值的样本；
- 使用相似样本的均值添补缺失值；
- 使用另外的机器学习算法预测缺失值。

在预处理阶段需要做两件事：第一，所有的缺失值必须用一个实数值来替换，因为我们使用的NumPy数据类型不允许包含缺失值。这里选择实数0来替换所有缺失值，恰好能适用于Logistic回归。这样做的直觉在于，我们需要的是一个在更新时不会影响系数的值。回归系数的更新公式如下：

```
weights = weights + alpha * error * dataMatrix[randIndex]
```

如果dataMatrix的某特征对应值为0，那么该特征的系数将不做更新，即：

```
weights = weights
```

另外，由于$sigmoid(0) = 0.5$，即它对结果的预测不具有任何倾向性，因此上述做法也不会对误差项造成任何影响。基于上述原因，将缺失值用0代替既可以保留现有数据，也不需要对优化算法进行修改。此外，该数据集中的特征取值一般不为0，因此在某种意义上说它也满足“特殊值”这个要求。

Chapter 2

树回归