机器学习算法系列（25）：最速下降法、牛顿法、拟牛顿法

一、最速下降法

最速下降法，又称为梯度下降法，是无约束最优化领域中最简单的算法，单独就这种算法来看，属于早就“过时”了的一种算法。但是，它的理念是其他某些算法的组成部分，或者说是在其他算法中，也有最速下降法的影子。它是一种迭代算法，每一步需要求解目标函数的梯度向量。

假设$f(x)$是R^n上具有一阶连续偏导的函数。要求解的无约束最优化问题是：

$$min_{x \in R^n} f(x)$$

梯度下降法是一种迭代算法。选取适当的初值$x^{(0)}$，不断迭代，更新x的值，进行目标函数的极小化，直到收敛。由于负梯度方向是使函数值下降最快的方向，在迭代的每一步，以负梯度方向更新x的值，从而达到减少函数值的目的。

由于$f(x)$具有一阶连续偏导数，若第k次迭代值为$x^{(k)}$，则可将$f(x)$在$x^{(k)}$附近进行一阶泰勒展开：

$$f(x) = f(x^{(k)}) + g_k^T(x - x^{(k)})$$

这里，$g_k = g(x^{(k)}) = \nabla f(x^{(k)})$为$f(x)$在$x^{(k)}$的梯度。

求出第$k+1$次迭代值$x^{(k+1)}$：

$$x^{(k+1)} \leftarrow x^{(k)} + \lambda_k p_k$$

其中p_k是搜索方向，取负梯度方向$p_k = -\nabla f(x^{(k)})$,λ_k是步长，由一维搜索确定，即λ_k使得

$$f(x^{(k)} + \lambda_k p_k) = min_{\lambda \geq 0} f(x^{(k)} + \lambda p_k)$$

算法步骤如下：

输入：目标函数$f(x)$，梯度函数$g(x) = \nabla f(x)$，计算精度ξ;

输出：$f(x)$的极小点x^*

1. 取初始值$x^{(0)} \in R^n$，置$k = 0$
2. 计算$f(x^{(k)})$
3. 计算梯度$g_k = g(x^{(k)})$，当$||g_k|| < \xi$时，停止迭代，令$x^* = x^{(k)}$；否则，令$p_k = -g(x^{(k)})$

，求λ_k，使

$$f(x^{(k)}+\lambda_k p_k)=min\lambda\geq 0 f(x^{(k)}+\lambda p_k)$$

4. 置$x^{(k+1)}=x^{(k)}+\lambda_k p_k$，计算$f(x^{(k+1)})$当$||f(x^{(k+1)})-f(x^{(k)})||<\xi$或$||x^{(k=1)}-x^{(k)}||<\xi$，停止迭代，令$x^*=x^{(k+1)}$
5. 否则，置$k=k+1$，转到步骤3。

当目标函数是凸函数时，梯度下降法的解释全局最优解。一般情况下，其解不保证是全局最优解。梯度下降法的收敛速度也未必是很快的。

二、牛顿法

考虑如下无约束的极小化问题

$$minX f(x)$$

其中$X=(x_1,x_2,x_3，\quad\cdots，\quad x_N)^T\in R^N$这里我们假定$f$为凸函数，且两阶连续可微。记$x^*$为目标函数的极小值。

为了简单起见，首先考虑$N=1$的简单情形（此时目标函数$f(X)$变为$f(x)$）。牛顿法的基本思想是：在现有极小值估计值的附近对$f(x)$作二阶泰勒展开，进而找极小点的下一个估计值。设x_k为当前的极小点估计值，则

$$f(x)=f(x_k)+f^{'}(x_k)(x-x_k)+\frac{1}{2}f^{''}(x_k)(x-x_k)^2$$

表示$f(x)$在x_k附近的二阶泰勒展开式（略去了关于$x-x_k$的高阶项）。由于求得是最值，由极值必要条件可知，$f(x)$应该满足

$$f^{'}(x)=0$$

，即

$$f^{'}(x_k)+f^{''}(x_k)(x-x_k)^2$$

从而求得

$$x=x_k-\frac{f^{'}(x_k)}{f^{''}(x_k)}$$

于是，若给定初始值x_0，则可以构造如下的迭代格式

$$x_{k+1} = x_k - \frac{f^{'}(x_k)}{f^{''}(x_k)}$$

于是，若给定初始值x_0，则可以构造如下的迭代格式

$$x_{k+1} = x_k - \frac{f^{'}(x_k)}{f^{''}(x_k)},\ k = 0, 1, \cdots$$

产生序列$\{x_k\}$来逼近$f(x)$的极小点。在一定条件下$\{x_k\}$可以收敛到$f(x)$的极小点。

对于$N > 1$的情形，二阶泰勒展开式可以做推广，此时

$$f(X) = f(X_k) + \nabla f(X_k) \cdot (X - X_k) + \frac{1}{2} \cdot (X - X_k)^T \cdot \nabla^2 f(X_k) \cdot (X - X_k)$$

其中∇f为f的梯度向量，$\nabla^2 f$为海森矩阵，其定义分别为

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \cdots \\ \frac{\partial f}{\partial x_N} \end{bmatrix}, nablaf^2 = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_N} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_N} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial^2 f}{\partial x_N \partial x_1} & \frac{\partial^2 f}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_N^2} \end{bmatrix}$$

注意，∇f和$\nabla^2 f$中的元素均为关于X的函数，以下分别将其简记为g和H。特别地，若f的混合偏导数可交换次序(即对$\forall\ i, j$，成立$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial^2 f}{\partial x_j \partial x_i}$)，则海森矩阵$H$为对称矩阵，而$\nabla f(X_k)$和$\nabla^2 f(X_k)$则表示将$X$取为$X_k$后得到的实值向量和矩阵，以下分别将其简记为$g_k$和$H_k$（这里字母g表示gradient，H表示Hessian）

同样地，由于是求极小点，极值必要条件要求它为$f(X)$的驻点，即

$$\nabla f(X) = 0$$

亦即对二阶泰勒展开作用一个梯度算子

$$g_k + H_k \cdot (X - X_k) = 0$$

进一步，若矩阵H_k非奇异，则可解得

$$X = X_k - H_k^{-1} \cdot g_k$$

于是，若给定初始值X_0，则同样可以构造出迭代格式

$$X_{k+1} = X_k - H_k^{-1} \cdot g_k$$

这就是原始的牛顿迭代法，其迭代格式中的搜索方向$d_k = -H_k^{-1} \cdot g_k$称为牛顿方向。下面给出牛顿法的完整算法描述：

1. 给定初值X_0和精度阀值ξ，并令$k:=0$
2. 计算g_k和H_k
3. 若$||g_k|| < \xi$，则停止迭代；否则确定搜索方向$d_k = -H_k^{-1} \cdot g_k$
4. 计算新的迭代点$X_{k+1}:=X_k + d_k$
5. 令k:=k+1，转至步2

当目标函数是二次函数时，由于二次泰勒展开函数与原目标函数不是近似而是完全相同的二次式，海森矩阵退化成一个常数矩阵，从任一初始点出发，秩序一步迭代即可达到$f(X)$的极小点X^*，因此牛顿法是一种具有二次收敛性的算法。对于非二次函数，若函数的二次形性态较强，或迭代点已进入极小点的领域，则其收敛速度也是很快的，这是牛顿法的主要优点。

但原始牛顿法由于迭代公式中没有步长因子，而是定步长迭代，对于非二次型目标函数，有时会使函数值上升，即出现$f(X_{k=1}) > f(X_k)$的情况，这表明原始牛顿法不能保证函数值稳定地下降，在严重的情况下甚至可能造成迭代点列$\{X_k\}$的发散而导致计算失败。

为了消除这个弊病，人们提出了“阻尼牛顿法”，阻尼牛顿法每次迭代的方向仍然采用d_k，但每次迭代需沿此方向作一维搜索（line search），寻求最优的步长因子λ_k，即

$$\lambda_k = argmin\lambda \in Rf(X_k + \lambda d_k)$$

下面给出阻尼牛顿法的完整算法描述：

1. 给定初值X_0和精度阀值ξ，并令$k:=0$
2. 计算g_k和H_k
3. 若$||g_k|| < \xi$，则停止迭代；否则确定搜索方向$d_k = -H_k^{-1} \cdot g_k$
4. 利用$\lambda_k = argmin\lambda \in Rf(X_k + \lambda d_k)$得到步长$\lambda_k$，计算新的迭代点$X_{k+1}:=X_k + d_k$
5. 令k:=k+1，转至步2

至此完成了牛顿法的算法介绍，接下来对其做个小结：

牛顿法是梯度下降法的进一步发展，梯度下降法利用目标函数的一阶偏导数信息、以负梯度方向

作为搜索方向，只考虑目标函数在迭代点的局部性质；而牛顿法不仅使用目标函数的一阶偏导数，还进一步利用了目标函数的二阶偏导数，这样就考虑了梯度变化的趋势，因而能更全面地确定合适的搜索方向加快收敛，它具二阶收敛速度。但牛顿法主要存在以下两个缺点：

1. 对目标函数有较严格的要求。函数必须具有连续的一、二阶偏导数，海森矩阵必须正定。
2. 极端相当复杂，除需要计算梯度以外，还需要计算二阶偏导数矩阵和它的逆矩阵。计算量、存储量均很大，且均以维数N的平方比增加，当N很大时这个问题更加突出。

三、拟牛顿法

牛顿法虽然收敛速度快，但是计算过程中需要计算目标函数的二阶偏导数，计算复杂度较大。而且有时目标函数的海森矩阵无法保持正定，从而使牛顿法失效。为了克服这两个问题，人们提出了拟牛顿法。这个方法的基本思想是：不用二阶偏导数而构造出可以近似海森矩阵或者海森矩阵的逆的正定对称阵，在拟牛顿的条件下优化目标函数。不同的构造方法就产生了不同的拟牛顿法。

也有人把“拟牛顿法”翻译成“准牛顿法”，其实都是表示“类似于牛顿法”的意思，因此只是对算法中用来计算搜索方向的海森矩阵（或海森矩阵的逆）作了近似计算罢了。

在介绍具体的拟牛顿法之前，我们先推到一个拟牛顿条件，或者叫拟牛顿方程，还有的叫做割线条件。因为对海森矩阵（或海森矩阵的逆）做近似总不能随便近似，也需要理论指导，而拟牛顿条件则是用来提供理论指导的，它指出了用来近似的矩阵应该满足的条件。

为明确起见，下文中用B表示对海森矩阵H本身的近似，而用D表示对海森矩阵的逆H^{-1}的近似，即$B \approx H, D \approx H^{-1}$

3.1 拟牛顿条件

设经过$k+1$次迭代后得到X_{k+1}，此时将目标函数$f(X)$在X_{k+1}附近作泰勒展开，取二阶近似，得到

$$f(X) \approx f(X_{k+1}) + \nabla f(X_{k+1}) \cdot (X - X_{k+1}) + \frac{1}{2} \cdot (X - X_{k+1})^T \cdot \nabla^2 f(X_{k+1}) \cdot (X - X_{k+1})$$

在两边同时作用一个梯度算子∇，可得

$$\nabla f(X) \approx \nabla f\left(X_{k+1}\right) + H_{k+1} \cdot (X - X_{k+1})$$

取$X = X_k$并整理，可得

$$g_{k+1} - g_k \approx H_{k+1} \cdot (X_{k+1} - X_k)$$

若引入记号$s_k = X_{k+1}$， $y_k = g_{k+1} - g_k$则可以改写成

$$y_k \approx H_{k+1} \cdot s_k$$

或者

$$s_k \approx H_{k+1}^{-1} \cdot y_k$$

这就是所谓的拟牛顿条件，它对迭代过程中的海森矩阵H_{k+1}作约束，因此，对H_{k+1}做近似的B_{k+1}，以及对H_{k+1}^{-1}做近似的D_{k+1}可以将

$$y_k \approx H_{k+1} \cdot s_k$$

或者

$$s_k \approx H_{k+1}^{-1} \cdot y_k$$

作为指导。

3.2 DFP算法

DFP算法是以William C.Davidon、Roger Fletcher、Michael J.D.Powell三个人的名字的首字母命名的，它由Davidon于1959年首先提出，是最早的拟牛顿法。该算法的核心是：通过迭代的方法，对H_{k+1}^{-1}做近似，迭代格式为

$$D_{k+1} = D_k + \Delta D_k, k = 0, 1, 2, \cdots$$

其中的D_0通常取为单位矩阵I。因此，关键是每一步的校正矩阵ΔD_k如何构造。

注意，我们猜想ΔD_k可能与s_k, y_k和D_k发生关联。这里，我们采用“待定法”，即首先将ΔD_k待定城某种形式，然后结合拟牛顿条件来进行推导。

那将ΔD_k待定成什么形式呢？说起来比较tricky，我们将其待定为

$$\Delta D_k = \alpha u u^T + \beta v v^T$$

其中α和β为待定向量。从形式上看，这种待定公式至少保证了矩阵ΔD_k的对称性（因为uu^T和vv^T均为对称矩阵）

将其代入迭代式，并结合拟牛顿指导条件，可得

$$s_k = D_k y_k + \alpha u u^T y_k + \beta v v^T y_k$$

将其改写一下

$$s_k = D_k y_k + u(\alpha u^T y_k) + v(\beta v^T y_k) = D_k y_k + (\alpha u^T y_k) u + (\beta v^T y_k) v$$

括号中为两个数，既然是数，我们不妨作如下简单赋值

$$\alpha u^T y_k = 1 \ , \quad \beta v^T y_k = -1$$

即

$$\alpha = \frac{1}{u^T y_k}, \beta = -\frac{1}{v^T y_k}$$

其中向量u, v仍有待确定。

我们把$s_k = D_k y_k + u - v$写作

$$u - v = s_k - D_k y_k$$

要上式成立，不妨直接取

$$u = s_k, v = D_k y_k$$

代入求α和β的式子，便得到

$$\alpha = \frac{1}{s_k^T y_k}, \beta = \frac{1}{(D_k y_k)^T y_k} = -\frac{1}{y_k^T D_k y_k}$$

其中第二个式子用到了D_k的对称性。至此，我们已经将校正矩阵ΔD_k构造出来了，我们就可以得到

$$\Delta D_k = \frac{s_k s_k^T}{s_k^T y_k} - \frac{D_k y_k y_k^T D_k}{y_k^T D_k y_k}$$

综上，我们给出DFP算法的一个完整的算法描述。

1. 给定初值X_0和精度阀值ξ，并令$k := 0$
2. 确定搜索方向$d_k = -D_k^{-1} \cdot g_k$
3. 利用$\lambda_k = argmin_{\lambda \in R} f(X_k + \lambda d_k)$得到步长$\lambda_k$，令$s_k = \lambda_k d_k$，计算新的迭代点$X_{k+1} := X_k + s_k$
4. 若$||g_{k=1}|| < \xi$，则算法结束
5. 计算$y_k = g_{k+1} - g_k$
6. 计算

$$D_{k+1} = D_k + \frac{s_k s_k^T}{s_k^T y_k} - \frac{D_k y_k y_k^T D_k}{y_k^T D_k y_k}$$

7. 令$k := k + 1$转至步骤2.

3.3 BFGS算法

BFGS算法是以其发明者Broyden、Fletcher、Goldfarb和Shanno四个人的名字的首字母命名的。与DFP算法相比，BFGS算法性能更加。目前它已成为求解无约束非线性优化问题最常用的方法之一。BFGS算法已有较完善的局部收敛理论，对其全局收敛的研究也取得了重要成果。

BFGS算法中核心公式的推导过程和DFP完全类似，只是互换了其中s_k和y_k的位置。需要注意的是，BFGS算法是直接逼近海森矩阵，即$B_k \approx H_k$,仍采用迭代方法，设迭代格式为

$$B_{k+1} = B_k + \Delta B_k, k = 0, 1, 2, \cdots$$

其中的B_0也常取为单位矩阵I。因此，关键是每一步的校正矩阵ΔB_k如何构造，同样，将其待定为

$$\Delta B_k = \alpha u u^T + \beta v v^T$$

将其代入上式，并结合指导条件$y_k \approx H_{k+1} \cdot s_k$，可得

$$y_k = B_k s_k + (a u^T s_k) u + (\beta v^T s_k) v$$

通过令$a u^T s_k = 1, \beta v^T s_k = -1$,以及

$$u = y_k, v = B_k s_k$$

可以算得

$$\alpha = \frac{1}{y_k^T s_k}, \beta = -\frac{1}{s_k^T B_k s_k}$$

综上，便得到了如下的校正矩阵ΔB_k的公式

$$\Delta B_k = \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

好了，现在把矩阵ΔB_k和ΔD_k拿出来对比一下，除了你将D换成B外，就是把s_k和y_k互换了一下位置。

最后，给出BFGS算法的一个完整算法描述：

1. 给定初值X_0和精度阀值ξ，并令$k := 0$
2. 确定搜索方向$d_k = -B_k^{-1} \cdot g_k$
3. 利用$\lambda_k = argmin\lambda \in Rf(X_k + \lambda d_k)$得到步长$\lambda_k$，令$s_k = \lambda_k d_k$，计算新的迭代点$X_{k+1} := X_k + s_k$
4. 若$||g_{k=1}|| < \xi$，则算法结束
5. 计算$y_k = g_{k+1} - g_k$

6. 计算

$$B_{k+1} = B_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k}$$

7. 令$k := k + 1$转至步骤2.

3.4 L-BFGS算法