江苏科技大学

JIANGSU UNIVERSITY OF SCIENCE AND TECHNOLOGY

《最优化理论与算法》

学习心得

班级： 1922107141

学号： 192210714116

姓名： 芦 星 宇

日期： 2021.4

目录

梯度类算法

芦星宇

April 2021

目录

1 LASSO 问题

对于 LASSO 问题

$$\min_x \frac{1}{2}\|Ax-b\|_2^2+\mu\|x\|_1.$$

此问题为回归问题，而对于回归问题其实本质就是一个函数拟合的过程，模型不能太过复杂，否则很容易发生过拟合现象，所以我们就要加入正则化项，而对于 LASSO 问题，采用 L1 正则化，会使得部分学习到的特征权值为 0, 从而达到稀疏化和特征选择的目的。

为什么 L1 正则更容易导致稀疏解?

假设只有一个参数 w, 损失函数为 $L(w)$, 加上 L 1 正则项后有:

$$J_{L1}(w)=L(w)+\lambda|w|_1$$

假设 $L(w)$ 在 0 处的导数为 d_0, 即

$$\frac{\partial L(w)}{\partial w}=d_0$$

则可以推导使用 L 1 正则的导数

$$\frac{\partial J_{L1}(w)}{\partial w}|_{w=0^-}=d_0-\lambda$$

$$\frac{\partial J_{L1}(w)}{\partial w}|_{w=0^+}=d_0+\lambda$$

引入 L 1 正则后, 代价函数在 0 处的导数有一个突变, 从 $d_0+\lambda$ 到 $d_0-\lambda$, 若 $d_0+\lambda$ 和 $d_0-\lambda$ 异号，则在 0 处会是一个极小值点。因此，优化时，可能优化到该极小值点上，即 $w=0$ 处。

2 LASSO 问题的梯度下降

2.1 LASSO 问题的连续化策略

目的：寻找一个合适的 μ_t, 求解相应的 LASSO 问题。

方法：从较大的 μ_t 逐渐减小到 μ_0。

代码解析：

1) 更新梯度阈值，他们随着外层迭代的进行逐渐减小，对子问题求解的精度逐渐提高。

2) 当内层循环达到收敛条件退出时，缩减正则化系数 μ_t，并判断收敛。

3) 外层循环收敛的条件：当 μ 已经减小到与 μ_0 相同并且函数值或梯度值满足收敛条件。

2.2 BB 步长梯度下降法

对于可微的目标函数 $f(x)$，梯度下降法通过使用如下重复迭代格式

$$x^{k+1} = x^k - \alpha \bigtriangledown f(x^k)$$

求解 $f(x)$ 的最小值，其中 a_k 为第 k 步的步长。

令 $s^k = x^{k+1} - x^k, y^k = \nabla f(x^{k+1}) - \nabla f(x^k)$，定义两种 BB 步长，$\dfrac{(s^k)^\top s^k}{(s^k)^\top y^k}$ 和 $\dfrac{(s^k)^\top y^k}{(y^k)^\top y^k}$。

理论解释：

如果我们记 $g^k = \bigtriangledown f(x^{(k)})$ 和 $F^k = \bigtriangledown^2 f(x^{(k)})$，那么**一阶方法**就是 $x^{k+1} = x^k - \alpha_k g(x^{(k)})$，其中步长 α_k 是固定的，也可以使线搜索获得的，一阶方法简单但是收敛速度慢，** 牛顿方法 ** 就是 $x^{(k+1)} = x^{(k)} - (F^{(k)})^{-1} g^{(k)}$，其收敛速度更快，但是海森矩阵计算代价较大，而 **BB 方法**就是用 $\alpha_k g^{(k)}$ 来近似 $(F^{(k)})^{-1} g^{(k)}$。

定义 $s^k = x^{k+1} - x^k$ 和 $y^{(k-1)} = g^{(k)} - g^{(k-1)}$，那么海森矩阵实际上就是

$$F^{(k)} s^{(k-1)} = y^{(k-1)}$$

用 $(a_k I)^{-1}$ 来近似 $F^{(k)}$，那么就应该有

$$(\alpha_k I)^{-1} s^{(k-1)} = y^{(k-1)}$$

利用最小二乘法：

$$\alpha_k^{-1} = \arg\min_\beta \frac{1}{2} ||s^{(k-1)}\beta - y^{(k-1)}||^2 => \alpha_k^1 = \frac{(s^{k-1})^\top s^{k-1}}{(s^{k-1})^\top y^{k-1}}$$

$$\alpha_k = \arg\min_\alpha \frac{1}{2} ||s^{(k-1)}\beta - y^{(k-1)}\alpha||^2 => \alpha_k^2 = \frac{(s^{k-1})^\top y^{k-1}}{(y^{k-1})^\top y^{k-1}}$$

BB 方法特点：

1) 几乎不需要额外的计算，但是往往会带来极大的性能收益

2) 实际应用中两个表达式都可以用，甚至可以交换使用，但是优劣需结合具体问题

3) 收敛性很难证明。

2.3 Huber 光滑梯度法

将 LASSO 问题转化为光滑函数，

$$\ell_\sigma(x) = \begin{cases} \frac{1}{2\sigma} x^2, & |x| < \sigma; \\ |x| - \frac{\sigma}{2}, & \text{otherwise.} \end{cases}$$

使用 $L_\sigma(x)=\sum_{i=1}^{n}\ell_\sigma(x_i)$ 替代 $||x||_1$，得到如下优化问题:

$$\min_x f(x) := \frac{1}{2}\|Ax-b\|_2^2+\mu L_\sigma(x).$$

在 x 点处 f 的梯度为:

$$\nabla f(x)=A^\top(Ax-b)+\mu\nabla L_\sigma(x),$$

其中

$$(\nabla L_\sigma(x))_i=\begin{cases}\text{sign}(x_i), & |x_i|>\sigma;\\ \frac{x_i}{\sigma}, & |x_i|\le\sigma.\end{cases}$$

代码解析:

1) 针对光滑化之后的函数进行梯度下降。

2) 内层循环的收敛条件：当当前梯度小于阈值或者目标函数比那化小于阈值，内层迭代终止

3) 采用线搜索循环选择合适步长并更新 x。在步长不符合线搜索条件的情况下，对当前步长以 η 进行衰减，线搜索次数加 1。

2.4 线搜索方法 (LineSearch)

线搜索的迭代过程是 $x_{k+1}=x_k+\alpha_k p_k$, 其中 α_k 和 p_k 分别表示搜索步长和搜索方向。

p_k 是一个下降方向，满足 $\bigtriangledown f_k p_k\le 0$，则 $p_k=-B_k^{-1}\bigtriangledown f_k$, B 为对称非奇异矩阵，根据 B_k 的选择会产生以下几个方向:

1) $B_k=I$ 时，搜索方向为负梯度方向，该方法为最速下降方向。

2) $B_k=\bigtriangledown^2 f_k$ 时，该方法为牛顿方法。

3) 需要满足对称正定矩阵，该方法为拟牛顿方法。

3 总结

对 Matlab 的代码采用 Python 重构，对算法的流程有了比较深入的了解，但是对示例代码进行运行时，生成的迭代次数-函数值图像，相比与网站中给出的同等 Matlab 生成图像更快的收敛。

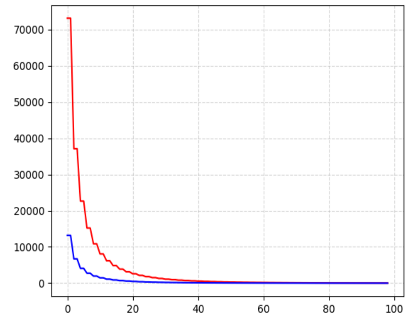

图 1: 利用梯度法解决 LASSO 问题

Github 地址：

https://github.com/Xingyu-Romantic/Machine-learning

参考文献

[1]. 为什么 L 1 正则化导致稀疏解 - CSDN

[2]. 线搜索方法 - CSDN

[3]. 凸优化笔记 15：梯度下降法 - 知乎

次梯度算法

芦星宇

April 2021

目录

1 次梯度方法

对于可导的凸函数，我们通常采用常规的梯度下降法处理，但当目标函数不可导（在某些点上导数不存在）时，我们没法采用常规的梯度下降法处理。于是引入次梯度 (Subgradient) 用于解决此类目标函数并不总是处处可导的问题。

1.1 次梯度定义

对于凸函数 f, 如果它可导，那么对 $\forall x, y \in dom f$，都有

$$f(y) \geq f(x) + \bigtriangledown f(x)^T(y-x)$$

此即凸函数的一阶条件，就是对于凸函数，其切线总是在函数的下方。

类比上式，给定函数 f, 对 $\forall y$，如果满足

$$f(y) \geq f(x) + g^T(y-x)$$

则称 g 是函数 f 在点 x 处的 ** 次梯度 (Subgradient)**。

将 f 在 x 处所有次梯度构成的集合称为 f 在 x 处的 ** 次微分 (Subdifferential)**，记作 $\partial f(x)$。

设函数 f 在 x_0 处不一定可导，以一维情况为例，

f 在 x_0 处的左导数：

$$a = \lim_{x \to x_0^-} \frac{f(x) - f(x_0)}{x - x_0}$$

f 在 x_0 处的右导数：

$$b = \lim_{x \to x_0^+} \frac{f(x) - f(x_0)}{x - x_0}$$

凸函数 f 的次微分区间 $[a, b]$ 中任何一个取值都是次梯度。

如果凸函数 f 在点 x 处是可导的，即 $a = b$，次微分中只有一个元素，此时次梯度就是梯度，即 g 就等于 $\bigtriangledown f(x)$；

如果凸函数 f 在点 x 处是不可导的，即 $a \neq b$，此时次梯度是次微分中的任意一个取值，它是不唯一的。

1.2 次梯度迭代算法

类似于梯度下降算法，只是将梯度更换称了次梯度，初始化 $x^{(0)}$，重复：

$$x^{(k)} = x^{(k-1)} - t_k g^{(k-1)}, k = 1, 2, 3...$$

其中 t_k 为步长，$g^{(k-1)} \in \partial f(x^{(k-1)})$, 即从次微分中随机选择一个次梯度作为梯度。

为了使更新的参数呈递减的趋势，对第 k 次的参数更新同步使用如下策略：

$$f(x_{best}^{(k)}) = \min_{i=0,\ldots,k} f(x^{(i)})$$

次梯度算法使用如下步长选择准则:

1) 固定的步长 $t_k = t, k = 1, 2, 3, ..., k$

2) 衰减的步长 t_k 满足如下条件:

$$\sum_{k=1}^{\infty} < \infty, \sum_{k=1}^{\infty} = \infty$$

例如, 取 $t_k = \frac{1}{k}, k = 1, 2, ..., \infty$, 则 $\sum_{k=1}^{\infty} t_k^2 = \sum_{k=1}^{\infty} t_k = \frac{1}{k^2}$ 收敛且 $\sum_{k=1}^{\infty} t_k = \sum_{k=1}^{\infty} t_k = \frac{1}{k}$（为调和级数，即 p=1）发散。

衰减的步长能够保证步长在逐步减小趋近于 0 的同时变化幅度不会太大。

只要选择的步长合适，算法总会收敛，只是算法收敛速度慢。

1.3 次梯度法的一般方法

1) $t = 1$ 选择有限的正的迭代步长 $(a_t)_{t=1}^{\infty}$

2) 计算一个次梯度 $g = \partial f(x^t)$

3) 更新 $x^{t+1} = x^t - \alpha_t g^t$

4) 若是算法没有收敛，则 $t = t + 1$ 返回第二步继续计算。

2 总结

采用 Python 重构次梯度算法问题，与 Matlab 生成图片有所偏差，在 1000 轮的时候梯度已不再变化，对于消失步长的线条，在 10 轮的时候会有突变。

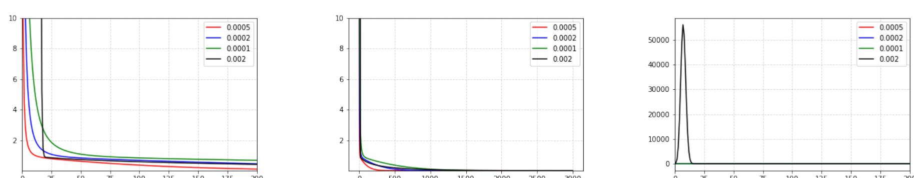

图 1: 次梯度算法解决 LASSO 问题

参考文献

[1]. [次梯度方法（Subgradient method）- 知乎

牛顿类算法

芦星宇

April 2021

目录

1 牛顿-共轭梯度法

1.1 牛顿法

牛顿法是梯度下降的进一步发展，梯度下降法利用了目标函数的一阶导数信息，以负梯度方向为搜索方向，只考虑目标函数在迭代点的局部性质，而牛顿法为代表的二阶近似方法除了一阶导数信息外，还利用二阶导数信息掌握了梯度变化的趋势，因而能够确定更合适的搜索方向加快收敛，具有二阶收敛速度。

1. 牛顿法对目标函数有比较高的要求，必须一阶二阶可导，海森矩阵必须**正定**
2. 计算量比较大，除了计算梯度外，还要计算海森矩阵及其逆矩阵
3. 当目标函数不是完全的凸函数时，容易陷入**鞍点**，导致更新朝着错误的方向移动。

1.2 共轭梯度法

共轭梯度法介于梯度下降法和牛顿法之间。共轭梯度法把共轭性与最速下降法相结合，利用迭代点处的梯度构造一组共轭方向，并沿共轭方向进行搜索，当前方向上的极小值搜索不会影响已经搜索过的方向的极值，因此共轭梯度法就有二次终止性。

什么是共轭?

H 是海森矩阵，如果满足 $d_t^T H d_{t-1} = 0$，则两个方向 d_t 和 d_{t-1} 共轭。

在应用共轭梯度法的时候，我们寻求一个和先前性搜索方向共轭的搜索方向，即不会撤销该方向上的进展，在训练迭代 t 时，下一步的搜索方向 d_t 的形式如下：

$$d_t = \bigtriangledown_\theta J(\theta) + \beta_t d_{t-1}$$

其中，β_t 控制我们应沿 d_{t-1} 加回多少到当前搜索方向上。适应共轭的直接方法会涉及到 H 特征向量的计算来选择 β_t，计算量是非常大的。为了避免这些计算，产生了以 FR 为代表的两种流行的算法来计算 β_t。

神经网络及其他深度学习模型的目标函数比二次函数复杂得多，但经验上，共轭梯度法在这种情况下仍然适用。当目标函数是高于二次的连续函数 (即目标函数的梯度存在) 时，其对应的解方程是 ** 非线性 ** 方程，非线性问题的目标函数可能存在局部极值，并且破坏了二次截止性，共轭梯度法需要在两个方面加以改进后，仍然可以用于实际的反演计算，但共轭梯度法不能确保收敛到全局极值。

1) 首先是共轭梯度法不能在 n 维空间内依靠 n 步搜索到达极值点，需要重启共轭梯度法，继续迭代，以完成搜索极值点的工作。

2) 在目标函数复杂，在计算时，由于需要局部线性化，需计算 Hessian 矩阵，且计算工作量比较大，矩阵 A 也有可能是病态的。Fletcher-Reeves 算法最为常用，抛弃了矩阵的计算。

共轭梯度法仅需一阶导数信息，但克服了最速下降法收敛慢的缺点，又避免了牛顿法需要存储和计算 Hesse 矩阵并求逆的缺点，共轭梯度法不仅是解决大型线性方程组最有用的方法之一，也是解大型非线性最优化最有效的算法之一。

对于规模较大的问题，精确求解牛顿方程组的代价比较高。事实上，牛顿方程求解等于无约束二次优化问题：

$$\min_{d^k} \frac{1}{2}(d^k)^\top \nabla^2 f(x^k) d^k + (\nabla f(x^k))^\top d^k,$$

其可以通过共轭梯度法来进行求解。

2 逻辑回归问题

如果面试官问你熟悉哪个机器学习模型，可以说 SVM，但千万别说 LR，因为细节真的太多。

Logistic Regression 虽然被成为回归，但其实际上是分类模型，并常用于二分类。Logistic Regression 因其简单、可并行化，可解释性强深受工业界喜爱。

本质： 假设数据服从这个分布，然后使用极大似然估计做参数的估计。

2.1 Logistic 分布

Logistic 分布是一种连续型的概率分布，其 ** 分布函数 ** 和 ** 密度函数 ** 分别为：

$$F(x) = P(X \le x) = \frac{1}{1+e^{-(x-\mu)/\gamma}} f(x) = F'(X \le x) = \frac{e^{-(x-\mu)/\gamma}}{\gamma(1+e^{-(x-\mu)/\gamma})^2}$$

其中 μ 表示位置参数, $\gamma > 0$ 为形状参数。

Logistic 分布是由其位置和尺度参数定义的连续分布。Logistic 分布的形状与正态分布的形状相似，但是 Logistic 分布的尾部更长，所以我们可以使用 Logistic 分布来建模比正态分布具有更长尾部和更高波峰的数据分布。在深度学习中常用到的 Sigmoid 函数就是 Logistic 的分布函数在 $\mu = 0, \gamma = 1$ 的特殊形式。

2.2 Logistic 回归

以二分类为例，对于所给数据集假设存在一条直线可以将数据完成线性可分。

有时候我们只要得到一个类别的概率，那么我们需要一种能输出 [0, 1] 区间的值的函数，考虑二分类模型，我们利用判别模型，希望对 $p(C|x)$ 建模，利用贝叶斯定理：

$$p(C_1|x) = \frac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)}$$

取 $a=\ln\frac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$，于是：

$$p(C_1|x)=\frac{1}{1+\exp(-a)}$$

上面的式子叫 Logistic Sigmoid 函数，其参数表示了两类联合概率比值的对数。在判别式中，不关心这个参数的具体值，模型假设直接对 a 进行。

Logistic 回归的模型假设是：

$$a=w^Tx$$

于是，通过寻找 w 的最佳值可以得到在这个模型假设下的最佳模型。概率判别模型常用最大似然估计的方式来确定参数。

对于一次观测，获得分类 y 的概率为（假定 $C_1=1,C_2=0$）：

$$p(y|x)=p_1^yp_0^{1-y}$$

那么对于 N 次独立全同的观测 MLE 为：

$$\hat{w}=\underset{w}{argmax}\,J(w)=argmax*w\sum*i=1^N(y_i\log p_1+(1-y_i)\log p_0)$$

注意到，这个表达式是交叉熵表达式的相反数乘 N，MLE 中的对数也保证了可以和指数函数相匹配，从而在大的区间汇总获取稳定的梯度。

对这个函数求导数，注意到：

$$p_1'=(\frac{1}{1+\exp(-a)})'=p_1(1-p_1)$$

则：

$$J'(w)=\sum_{i=1}^Ny_i(1-p_1)x_i-p_1x_i+y_ip_1x_i=\sum_{i=1}^N(y_i-p_1)x_i$$

由于概率值的非线性，放在求和符号中时，这个式子无法直接求解。于是在实际训练的时候，和感知机类似，也可以使用不同大小的批量随机梯度上升（对于最小化就是梯度下降）来获得这个函数的极大值。

3 总结

牛顿法使用了二阶梯度，梯度下降仅仅是一阶梯度，对于梯度下降而言，把它比做下山问题，那么梯度下降站在当前的位置要找到梯度最大的点，这样也就是坡度最大下山最快，但是牛顿法他不仅要找当前下降最快的方向，还要确保下下步的坡度更大，下山更快。

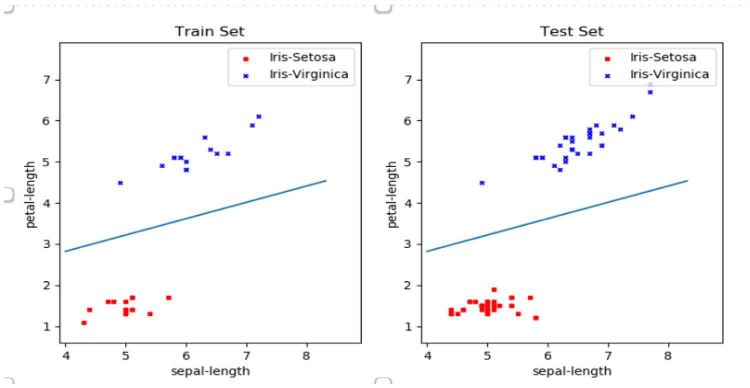

图 1: 牛顿法解决逻辑回归问题 (Iris 数据集)

参考文献

[1]. 拟牛顿法、共轭梯度法 - 知乎

[2]. 逻辑回归问题

[3]. 【机器学习】【白板推导系列】shuhuai008

[4]. 《Deep Learning》