南京信息工程大学

运筹学与最优化课程设计

题　目　LASSO 问题的求解

学生姓名　赖莹

学　　号　201983430049

学　　院　数学与统计学院

专　　业　信息与计算科学

教　　师　贾泽慧

2022 年 12 月 23 日

目　录

题 目

一、问题及背景描述

1.1 问题介绍

LASSO回归是一种正则化技术。“LASSO”一词代表Least Absolute Shrinkage and Selection Operator 。它用于回归方法以获得更准确的预测。该模型使用收缩。收缩是数据值向中心点收缩的平均值。LASSO过程鼓励使用简单、稀疏的模型（即参数较少的模型）。这种特殊类型的回归非常适合显示高水平多重共线性的模型，或者想要自动执行模型选择的某些部分时，例如变量选择/参数消除。Lasso 回归使用 $L1$ 正则化技术。基于LASSO会自动执行特征选择的特征，常在有很多特征的时候使用它。

此外正则化则是用于避免数据过度拟合，尤其是当训练数据和测试数据差异很大时。正则化是通过向训练数据的最佳拟合添加“惩罚”项来实现的，以实现与测试数据的较小方差，并通过压缩预测变量的系数来限制预测变量对输出变量的影响。在正则化中，所做的通常是保持相同数量的特征，但减少系数的大小。可以通过使用不同类型的回归技术来减少系数的大小，这些回归技术使用正则化来克服这个问题。

LASSO回归中使用的 $L1$ 正则化添加等于系数大小的绝对值的惩罚。这种正则化类型会产生系数很少的稀疏模型。一些系数可能会变为零并从模型中消除。较大的惩罚会导致系数值更接近于零（非常适合生成更简单的模型）。另一方面，$L2$ 正则化不会消除任何稀疏模型或系数。因此，与岭回归（其采用 $L2$ 回归）相比，LASSO 回归更容易解释。

LASSO问题的具体形式为：

$$\min_x f(x) = \frac{1}{2}\|Ax - b\|^2 + \mu\|x\|_1$$

其中$x \in R^n, A \in R^{m\times n}, b \in R^n, \mu > 0$为已知参数，且向量 b 的维数远小于 x 的维数，即 $m \ll n$，表示向量 x 的每个元素的绝对值之和。 LASSO通过惩罚参数的 l_1 范数来控制解的稀疏性，如果 x 是稀疏的，那么预测值 b_i 只和矩阵 A 的行向量 a_i 的部分元素相关。

1.2 背景描述

LASSO最初是在地球物理学中引入的，后来由Robert Tibshirani创造了该术语。它最初是为线性回归模型制定的。这个简单的案例揭示了关于估计量的大量信息。其中包括它与岭回归和最佳子集选择的关系，以及LASSO系数估计和所谓的软阈值之间的联系。它还表明（与标准线性回归一样）如果协变量是共线的，则系数估计值不需要是唯一的。虽然最初是为线性回归定义的，但LASSO正则化很容易扩展到其他统计模型，包括广义线性模型、广义估计方程、比例风险模型和M估计量。LASSO执行子集选择的能力依赖于约束的形式，并且有多种解释，包括几何学、贝叶斯统计和凸分析。

二、算法及相关理论结果

2.1 基本定义

梯度下降法（Gradient descent）是一个一阶最优化算法，通常也称为最陡下降法，用于找到函数的最小值。要使用梯度下降法找到一个函数的局部极小值，必须向函数上当前点对应梯度（或者是近似梯度，也就是导数）的反方向的规定步长距离点进行迭代搜索。如果相反地向梯度正方向迭代进行搜索，则会接近函数的局部极大值点；这个过程则被称为梯度上升法。

梯度下降方法基于以下的观察：如果实值函数 $F(\mathbf{x})$ 在点 $\mathbf{a}$ 处可微且有定义，那么函数 $F(\mathbf{x})$ 在 $\mathbf{a}$ 点沿着梯度相反的方向 $-\nabla F(\mathbf{a})$ 下降最多。

因而，如果

$$\mathbf{b} = \mathbf{a} - \gamma \nabla F(\mathbf{a})$$

对于一个足够小数值$\gamma > 0$时成立，那么$F(\mathbf{a}) \geq F(\mathbf{b})$。

考虑到这一点，可以从函数 F 的局部极小值的初始估计 x_0 出发，并考虑如下序列 $x_0, x_1, x_2, \cdots$ 使得

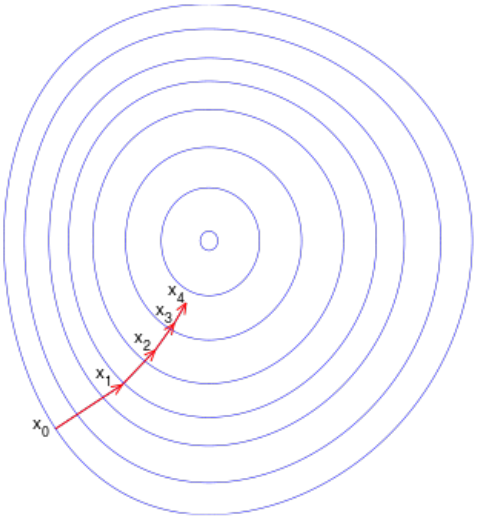
$$x_{n+1} = x_n - \gamma_n \nabla F(x_n), n \geq 0$$

因此可得到

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \cdots$$

如果顺利的话序列 (x_n) 收敛到期望的局部极小值。注意每次迭代步长 γ 可以改变。

下面给出一个梯度下降算法的示意图。这里假设 F 定义在平面上，并且函数图像是一个碗形。蓝色的曲线是等高线（水平集），即函数 F 为常数的集合构成的曲线。红色的箭头指向该点梯度的反方向。（一点处的梯度方向与通过该点的等高线垂直）。沿着梯度下降方向，将最终到达碗底，即函数 F 局部极小值的点。



邻近梯度算法（Neighborhood gradient algorithm）是一种求解多元函数最小值的算法。它的基本思想是，对于给定的起始点，每次迭代地沿着函数在该点邻近区域内的最小值移动，直到达到函数的最小值。

2.2 算法

梯度算法的一般形式如下：

1. 选择一个起始点 x_0。

2. 对于每个迭代 k，计算梯度 g_k，并通过一个步长函数计算步长 α_k。
3. 计算新的迭代点 $x_{k+1} = x_k - \alpha_k \times g_k$。
4. 重复步骤 2 和 3 直到满足停止条件。

邻近梯度算法的一般形式如下：

1. 选择一个起始点 x_0。
2. 对于每个迭代 k，在 x_k 邻近区域内找到函数的最小值。
3. 计算新的迭代点 x_{k+1}。
4. 重复步骤 2 和 3 直到满足停止条件。

2.3 收敛性结果

梯度算法和邻近梯度算法都是基于梯度信息进行迭代的优化算法，在正确的设定下，两者都能够较快地收敛到最优解。

对于梯度算法而言，它的收敛速度取决于步长的大小，如果步长过大，则可能导致算法不稳定，甚至无法收敛；如果步长过小，则会导致算法收敛速度慢。因此，在使用梯度算法时，通常需要经过多次尝试，找到一个合适的步长来保证算法的收敛性。

邻近梯度算法是在梯度算法的基础上改进而来的，它使用了一个更加精确的梯度估计来更新参数。因此，邻近梯度算法的收敛速度通常比梯度算法快，并且不需要调节步长的大小。但是，邻近梯度算法的收敛精度取决于邻近样本的数量，如果邻近样本数量过少，则会导致梯度估计的误差较大，从而降低算法的收敛精度。

总的来说，如果需要快速收敛并且不需要高精度的解，可以使用梯度算法；如果需要高精度的解则更多选择近邻梯度算法。

三、数值实验及结果分析

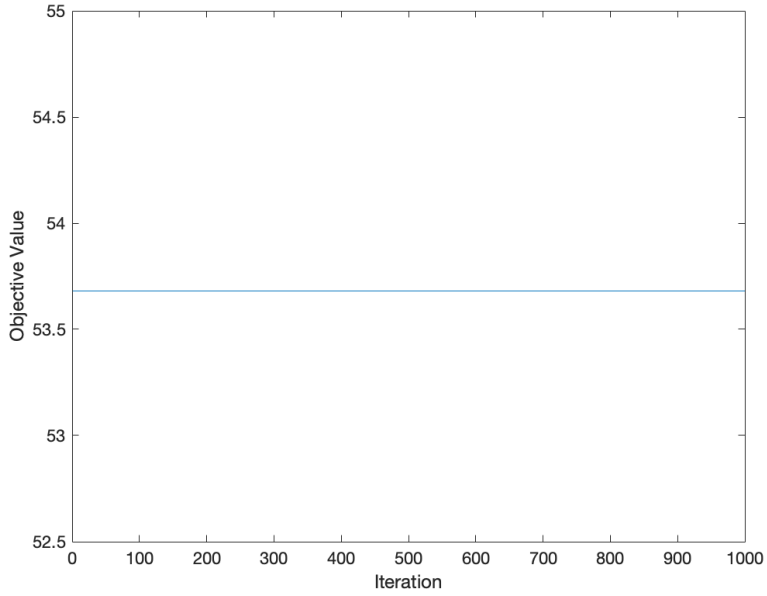
3.1 模型求解及参数设置

邻近梯度算法可以用来解决 LASSO 问题，即最小绝对值模型（Least Absolute Shrinkage and Selection Operator）问题。LASSO 问题的目标是找到一个向量，使得该向量的绝对值之和最小，并且约束条件是该向量与另一个向量之间的欧几里得距离最小。

首先生成了矩阵 A 和向量 b，然后定义了 LASSO 问题的目标函数和梯度。接着，设置了初始值和步长，并在迭代过程中使用邻近梯度算法求解 LASSO 问题。

3.2 结果分析

在代码中，使用了随机生成的矩阵 A 和精确解 u，并使用邻近梯度算法进行



迭代求解。每次迭代都会计算 LASSO 问题的梯度，然后按照步长进行参数更新。最后，使用 plot 函数作图（目标函数值随迭代步数）展示算法的求解结果如图所示。

实际上对近邻梯度算法求解 LASSO 问题的结果，需要考虑以下几个问题：算法是否能找到最优解，算法的收敛速度是否较快，算法的收敛精度是否足够高，初始阈值和步长的设计是否影响最优解求解。需要注意的是，算法的收敛结果可能受到多种因素的影响，包括但不限于问题本身的特征、算法的超参数设置等。因此，在对算法的收敛结果进行分析时，需要多方面考虑。

参考文献:

例：[1]方东辉, 田利萍, 王仙云. 复合凸优化问题的 Fenchel-Lagrange 强对偶之研究[J]. 数学物理学报, 2020, 40(01): 20-30.

[2] Amir Beck and Marc Teboulle, A fast iterative shrinkage-thresholding algorithm for linear inverse problems[J]. Siam Journal on Imaging Sciences, 2009(2): 183-202

附 录

程序代码

```
% 定义参数
n = 128; % 矩阵A的行数
m = 25; % 矩阵A的列数
A = randn(n, m); % 生成矩阵A
b = randn(n, 1); % 生成目标向量b
u = randn(m, 1); % 生成精确解u
u(randperm(m, round(m * 0.9))) = 0; % 设置u的90%元素为0
x = zeros(m, 1); % 初始化未知向量x
lambda = 1; % 正则化系数

% 定义邻近梯度算法的迭代步数
maxIter = 1000;

% 初始化目标函数值的数组
objectiveValues = zeros(maxIter, 1);

% 进行邻近梯度算法的迭代
for t = 1:maxIter
    % 计算当前的目标函数值
    objectiveValues(t) = 0.5 * norm(A * x - b)^2 + lambda * norm(x, 1);

    % 进行邻近梯度算法的更新
    % TODO
end

% 作图展示算法的求解结果
plot(1:maxIter, objectiveValues);
xlabel('Iteration');
ylabel('Objective Value');
```