《最优化方法》实验报告

实验一 近邻梯度方法与次梯度方法求解 Lasso 问题

实验报告

课　　程	机器学习
院（系）	计算机科学与技术学院
专　　业	人工智能
级　　别	2021 级
学　　号	2115102001
姓　　名	安炳旭
指导老师	彭佳林

2023 年 12 月 3 日

目 录

实验一 近邻梯度方法与次梯度方法
求解 Lasso 问题

1 实验目标

1. 理解 Lasso 问题。
2. 掌握求解 Lasso 问题的方法。
3. 理解近邻梯度法和迫近梯度法求解 Lasso 问题的方法。

2 实验内容

1. 使用近邻梯度法求解 Lasso 问题。
2. 使用迫近梯度法求解 Lasso 问题。

3 实验要求

1. 鲁棒、良好的算法实现
2. 接口友好、可复用程度高
3. 对比两种算法的优劣，并画出对比图
4. 研究 lambda 的影响
5. 研究不同的步长及影响
6. 实验报告应写出算法伪代码
7. 代码应有良好的注释

4 实验方案

4.1 实验概括

Lasso 问题是一种线性回归的变体，通过在损失函数中添加 L1 正则项，促使模型系数稀疏化，从而实现特征选择和降低模型复杂性。然而由于 L1 正则项无法直接进行求导，因此 Lasso 问题的求解并不容易。本实验对 Lasso 问题采取了两种求解策略，分别是**近邻梯度法和次梯度法**，并探讨了两种方法的优劣对比，发现**近邻梯度法在相同条件下比次梯度法性能更佳**。同时通过调整模型的步长和正则化系数等超参数，对两种方法进行了进一步研究，发**现当步长选定** $t=3\times10^{-4}$,**正则化系数**$\lambda=1$**时，在本实验的近邻梯度法取得最佳效果**。

4.2 实验平台

本实验选取 python 环境，版本 3.9，同时使用了 Jupyter Notebook 进行程序运行。

4.3 实验原理

➢ 4.3.1 Lasso 问题

LASSO 问题可以用如下的数学公式进行描述：

给定数据集$D(X,y)$，$X\in\mathbb{R}^{m\times n}$,$y\in\mathbb{R}^{n\times 1}$,问题是找到一个参数向量$\beta$,最小化以下目标函数，如(1)式所示:

$$f(\beta)=\frac{1}{2}\|y-X\beta\|_2^2+\lambda\|\beta\|_1 \tag{1}$$

其中，$\|y-X\beta\|_2^2$表示平方 2 范数，是平方损失；$\|\beta\|_1$表示 1 范数，是正则化项。λ是正则化系数，用于权衡平方损失和正则化项。

LASSO 问题的目标是寻找一个稀疏解，即让一些参数为零，从而实现特征选择的效果。

➢ 4.3.2 迫近梯度法(Proximal Gradient Descent)

迫近梯度法常用于求解以下形式的优化问题，如(2)式所示：

$$\min f(x)=g(x)+h(x) \tag{2}$$

其中$g(x)$是凸函数且可导，$h(x)$是凸函数但不一定要可导。使用迫近梯度法的过程是，首先对$g(x)$进行一次梯度下降，得到中间结果$x^{k+\frac{1}{2}}$,然后将这个中间结果带入到$h(x)$中求其近邻点的投影，即完成一次迭代。算法如(3)式所示：

$$\begin{cases} x^{k+\frac{1}{2}}=x^k-t\cdot\nabla g(x^k) \\ x^{k+1}=\arg\min\limits_x\{h(x)+\dfrac{1}{2t}||x-x^{k+\frac{1}{2}}||^2 \end{cases} \tag{3}$$

在本题中，$g(\beta)=\frac{1}{2}\|y-X\beta\|_2^2$，$h(\beta)=\lambda\|\beta\|_1$ 将它们带入到(3)式中，得到(4)式:

$$\begin{cases} \beta^{k+\frac{1}{2}} = \beta^k - t \cdot X^T(y - X\beta^k) \\ \beta^{k+1} = \arg\min\limits_x \{\lambda \| \beta \|_1 + \dfrac{1}{2t} \| \beta - \beta^{k+\frac{1}{2}} \|^2\} \end{cases} \tag{4}$$

求解上述迭代方程组第二行的$\arg\min\limits_{\beta}$时，用到了软阈值算法，求解可以得到β^{k+1}的形式如(5)式所示:

$$\beta^{k+1} = \begin{cases} \beta^{k+\frac{1}{2}} + \lambda, & \beta^{k+\frac{1}{2}} < -t\lambda \\ 0, & |\beta^{k+\frac{1}{2}}| \le t\lambda \\ \beta^{k+\frac{1}{2}} - \lambda, & \beta^{k+\frac{1}{2}} > t\lambda \end{cases} \tag{5}$$

至此，成功使用近邻梯度法求解了 Lasso 问题，并得到了其迭代式。

- 4.3.1 **次梯度法**(Subsgradient Descent)

用$g(x) \in \partial f(x)$来表示$f(x)$在x处的次梯度。所谓次梯度法，就是在不可微点处用次梯度代替梯度，然后使用梯度下降法。当函数可微时，与梯度下降法无异，梯度下降法的一般形式如(6)式所示。

$$x^{k+1} = x^k - t \cdot g(x^k) \tag{6}$$

次梯度法对于步长t的选取需满足三个条件之一:**(i)**固定步长$t^k = t$. **(ii)** 步长平方可加，但步长不可加，即$t^k \ge 0, \sum\limits_{k=1}^{\infty} t_k^2 < \infty, \sum\limits_{k=1}^{\infty} t_k = \infty$. **(iii)** 步长不可加但步长递减，即$\alpha_k \ge 0, \lim\limits_{k\to\infty} \alpha_k = 0, \sum\limits_{k=1}^{\infty} \alpha_k = \infty$.

在本问题中,令 Lasso 问题的目标函数为$f(\beta) = \frac{1}{2} \| y - X\beta \|_2^2 + \lambda \| \beta \|_1$则

$$\frac{\partial f(\beta)}{\partial \beta} = X^T(y - X\beta) + \lambda \frac{\partial \| \beta \|_1}{\partial \beta} \tag{7}$$

可以发现，(7)式中，$\frac{\partial \| \beta \|_1}{\partial \beta}$在 0 处不可微，因此其次梯度可以选取$[-1, 1]$之内的任何值，如(8)式所示：

$$\begin{cases} \left(\dfrac{\partial \| \beta \|_1}{\partial \beta}\right)_i = -1, & \beta_i < 0 \\ -1 \le \left(\dfrac{\partial \| \beta \|_1}{\partial \beta}\right)_i \le 1, & \beta_i = 0 \\ \left(\dfrac{\partial \| \beta \|_1}{\partial \beta}\right)_i = 1, & \beta_i > 0 \end{cases} \tag{8}$$

本实验中，对于步长选取，采取了第三种方式，选取$t^k = 0.001 \times \dfrac{1}{\sqrt{k}}$.

4.4 算法伪码

使用近端梯度下降法和次梯度法的伪代码分别如表 1 与表 2 所示

表 1 近端下降算法伪代码

Algorithm 1:近端梯度下降算法
Input: 数据集$D(X,y)$,初始参数β^0,最大迭代次数k,学习率t,正则化参数λ Output: 第k次迭代参数β^k
1. 计算函数$f(\beta)=g(\beta)+h(\beta)$ 2. **for** t=0,1,…,k-1 **do** 3. 计算梯度 $\nabla g(\beta)$ 4. 更新参数 $\beta^{k+\frac{1}{2}} = \beta^k - t\nabla g(\beta)$ 5. 近端映射 $\beta^{k+1} = \arg\min_x \{\lambda \| \beta \|_1 + \dfrac{1}{2t} \| \beta - \beta^{k+\frac{1}{2}} \|^2\}$ 6. **end** 7. **return** β^k

表 2 次梯度下降算法伪代码

Algorithm 2:次梯度法
Input: 数据集$D(X,y)$最大迭代次数k,学习率t,正则化参数λ Output: 第k次迭代参数β^k
1. ,初始参数β^0 2. **for** t=0,1,…,k-1 **do** 3. 计算次梯度 $g^k \in \partial f(x^k)$ 4. 计算步长 $t^k = 0.001 \times \dfrac{1}{\sqrt{k}}$

5. 更新参数 $\beta^{k+1}=\beta^k-t^k g^k$

6. **end**

7. **return** β^k

4.5 实验步骤

■ 4.5.1 数据生成

数据生成部分，本实验编写了 Generate_Data 函数实现这一功能。其中，函数可以生成指定维度的特征矩阵X,值得注意的是，为了提高模型的鲁棒性，本实验对观测向量y添加了噪声项$\epsilon\sim N(0,0.1)$. 详细代码如表 3 所示:

表 3 数据生成详细代码

```
def Generate_Data(X_dim, beta_dim, beta_sparsity, e_dim):
    # 生成具有稀疏特性的真实 beta 向量
    beta_nonzero_index = random.sample(range(beta_dim),
beta_sparsity)  # 从范围内随机选择非零元素的下标
    beta_nonzero_element = np.random.normal(0, 1, beta_sparsity)  #
生成非零元素的随机值
    beta_real = np.zeros(beta_dim)
    for i in range(len(beta_nonzero_index)):
        beta_real[beta_nonzero_index] = beta_nonzero_element
    # 生成测量矩阵 X 和噪声向量 e
    X = np.random.normal(0, 1, X_dim)  # 生成服从正态分布的测量矩阵 X
    e = np.random.normal(0, 0.1, e_dim)  # 生成服从正态分布的噪声向量
e
    # 生成观测向量 y，其中 y = X * beta_real + e
    y = X @ beta_real + e
    return beta_real, y, X
```

关于其参数的详细解释如下所示：

该函数接收如下几个参数：

- X_dim: 表示输入数据矩阵 X 的维度，即生成的数据中包含的特征或变量的数量。
- beta_dim: 表示真实系数向量 beta_real 的维度，即与 X 中的特征对应的系数数量。
- beta_sparsity: 确定真实系数向量 beta_real 的稀疏度，指定了在 beta_real 中非零元素的数量，这在只有部分特征对输出有显著影响的情况下很有用。
- e_dim: 表示噪声向量 e 的维度，指定了噪声向量中的元素数量，该噪声被添加到 X 与 beta_real 的乘积中，以模拟生成的数据中的噪声。

该函数提供以下几个返回值：

- beta_real: 真实系数向量，表示特征与输出之间的潜在关系。
- y: 输出向量，生成为 X 与 beta_real 的乘积，同时加入了噪声 e。
- X: 随机生成的输入数据矩阵。

■ 4.5.2 迫近梯度法

接下来，使用迫近梯度法来对生成的数据进行求解，根据 4.3 的分析以及表 1 的伪代码，编写的程序如表 4 所示：

表 4 迫近梯度法详细代码

```
def PGD(beta_dim, X, y, epoch=5000, t=0.0003, lamda=1, epsilon=1e-5):
    # 初始化系数向量
    beta_k = np.zeros(beta_dim)
    beta_k_old = np.zeros(beta_dim)
    # 记录每步计算结果的列表
    plot_iteration = []
    # 保存每一步的 f 值的列表
    f_values = []
    # 迭代次数
    k = 1
    while k < epoch:
        # 计算下降方向（负梯度）
        beta_k_temp = beta_k - t * X.T @ (X @ beta_k - y)
        # 临近点投影（软阈值）
```

```
        for i in range(beta_dim):
            if beta_k_temp[i] < -t * lamda:
                beta_k[i] = beta_k_temp[i] + t * lamda
            elif beta_k_temp[i] > t * lamda:
                beta_k[i] = beta_k_temp[i] - t * lamda
            else:
                beta_k[i] = 0
        # 记录每步计算结果
        plot_iteration.append(beta_k.copy())
        # 计算 f 值的变化
        f_k = np.linalg.norm(beta_k - beta_k_old)
        # 记录每一步的 f 值
        f_values.append(f_k)
        # 如果 f 值变化小于设定的阈值，提前结束迭代
        if f_k < epsilon:
            break
        else:
            # 更新旧的系数向量
            beta_k_old = beta_k.copy()
            k += 1
    # 最优解
    beta_optm = beta_k[:]
    # 计算最终 Lasso 函数值
    f_k = Lasso(y, X, beta_optm, lamda)
    print(f"||beta(k)-beta(k-1)||={f_k:.2f}")
    return beta_optm, plot_iteration, f_values
```

该函数接收如下几个参数：

- beta_dim: 系数向量的维度，即模型的参数个数。
- X: 输入数据的特征矩阵。

- y: 输入数据的标签向量。
- epoch: 最大迭代次数，默认为 5000。
- t(可选项): 学习率或步长，控制每次迭代更新的幅度，默认为 0.0003。
- lamda(可选项): L1 正则化参数，用于控制系数的稀疏性，默认为 1。
- epsilon(可选项): 迭代停止的阈值，当系数变化小于该值时停止迭代，默认为 1e-5。

该函数提供以下几个返回值：

- beta_optm: 求解得到的最优系数向量，即 Lasso 回归的结果。
- plot_iteration: 记录每步计算结果的列表，包含了每次迭代得到的系数向量。
- f_values: 保存当前β与上一步β差值的列表，反映了优化过程中目标函数的变化趋势。

通过上述函数，能够将迭代完毕后的β以列表形式进行返回，方便后续结果的比较。

■ 4.5.3 次梯度法

最后实现的是次梯度法，编写的函数如表 5 所示：

表 5 次梯度法详细代码

```
def SGD(beta_dim,X,y,epoch=5000,t_0=0.0003,lamda=1,epsilon=1e-5):
    beta_k = np.zeros(beta_dim)
    beta_k_old = np.zeros(beta_dim)
    f_0=Lasso(y,X,beta_k,lamda)
    plot_iteration = []  # 每步计算结果
    f_values = []  # 保存每一步的 f 值
    k = 1  # 迭代次数
    while k < epoch:
        t = t_0 / np.sqrt(k)  # 递减步长
        # 计算目标函数次梯度
        l1_subgradient = np.zeros(beta_dim)  # L1 范数的次梯度
        for i in range(len(beta_k)):
            if beta_k[i] != 0:
                l1_subgradient[i] = np.sign(l1_subgradient[i])
            else:
```

```
                l1_subgradient[i] = np.random.uniform(-1, 1)  # 随机取[-1,
1]内的值作为次梯度
        subgradient = X.T @ (X@beta_k - y) + lamda * l1_subgradient
        beta_k = beta_k - t * subgradient
        plot_iteration.append(beta_k.copy())  # 记录每步计算结果
        f_k = np.linalg.norm(beta_k - beta_k_old)
        f_values.append(f_k)  # 记录每一步的 f 值
        if f_k < epsilon:
            break
        else:
            beta_k_old = beta_k.copy()  # 深拷贝
            k += 1
    beta_optm = beta_k.copy()  # 最优解
    print(f"||beta(k)-beta(k-1)||={f_k:.2f}")
    return beta_optm,plot_iteration,f_values
```

这个函数是次梯度法的实现，以下是函数的参数和返回值的解释：

该函数接收如下几个参数：

- beta_dim: 系数向量的维度，即模型的参数个数。
- X: 输入数据的特征矩阵。
- y: 输入数据的标签向量。
- epoch: 最大迭代次数，默认为 5000。
- t_0: 初始学习率或步长，默认为 0.0003。
- lamda: L1 正则化参数，用于控制系数的稀疏性，默认为 1。
- epsilon: 迭代停止的阈值，当系数变化小于该值时，停止迭代，默认为 1e-5。

该函数提供以下几个返回值：

- beta_optm: 求解得到的最优系数向量，即 Lasso 回归的结果。
- plot_iteration: 记录每步计算结果的列表，包含了每次迭代得到的系数向量。
- f_values: 保存每一步的 Lasso 目标函数值的列表，反映了优化过程中目标函数的变化趋势。

5 实验结果与分析

本实验对生成的数据进行 Lasso 回归，并且分别使用迫近梯度法和次梯度法进行求解，作出的可视化图表如下所示：

首先来探讨迫近梯度法的性能，考虑**步长**对算法的影响：

实验选取了步长为1×10^{-4},5×10^{-5},1×10^{-5}三种不同的步长，固定正则化系数λ=10,得到的结果如图 1 所示：

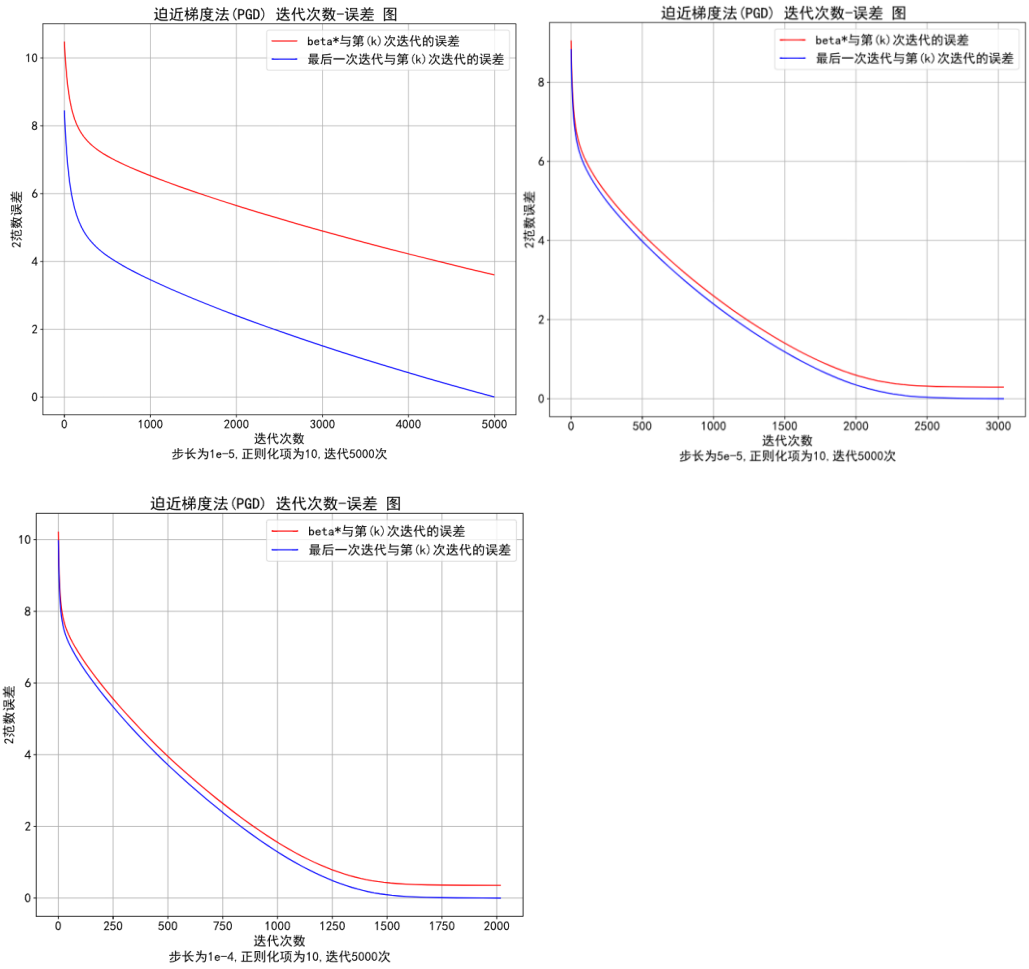


图 1 不同步长值对迫近梯度法性能的影响

图 1 蓝色曲线代表每次迭代的β^k与最后一次的β^K之间的二范数差值$||\beta^k-\beta^K||_2^2$,红色曲线代表每次迭代的β^k与真实值β^*之间的二范数差值，即$||\beta^k-\beta^*||_2^2$,二者越近，说明算法效果越好;二者接近的越早，说明算法越早收敛。

从图 1 中可以看到，当步长值为1×10^{-5}时，两条曲线之间的差值较大，说明此时的模型性能尚待提高，当选取更大的步长时，这一缺陷得以弥补，可以看到此时的模型性能较好，因此步长值对模型有着一定影响。

接下来考虑正则化项对模型的影响，实验固定步长为1×10^{-4}，选取不同的正则化系数为$0.1, 1, 10, 100$,得到的结果如图 2 所示：

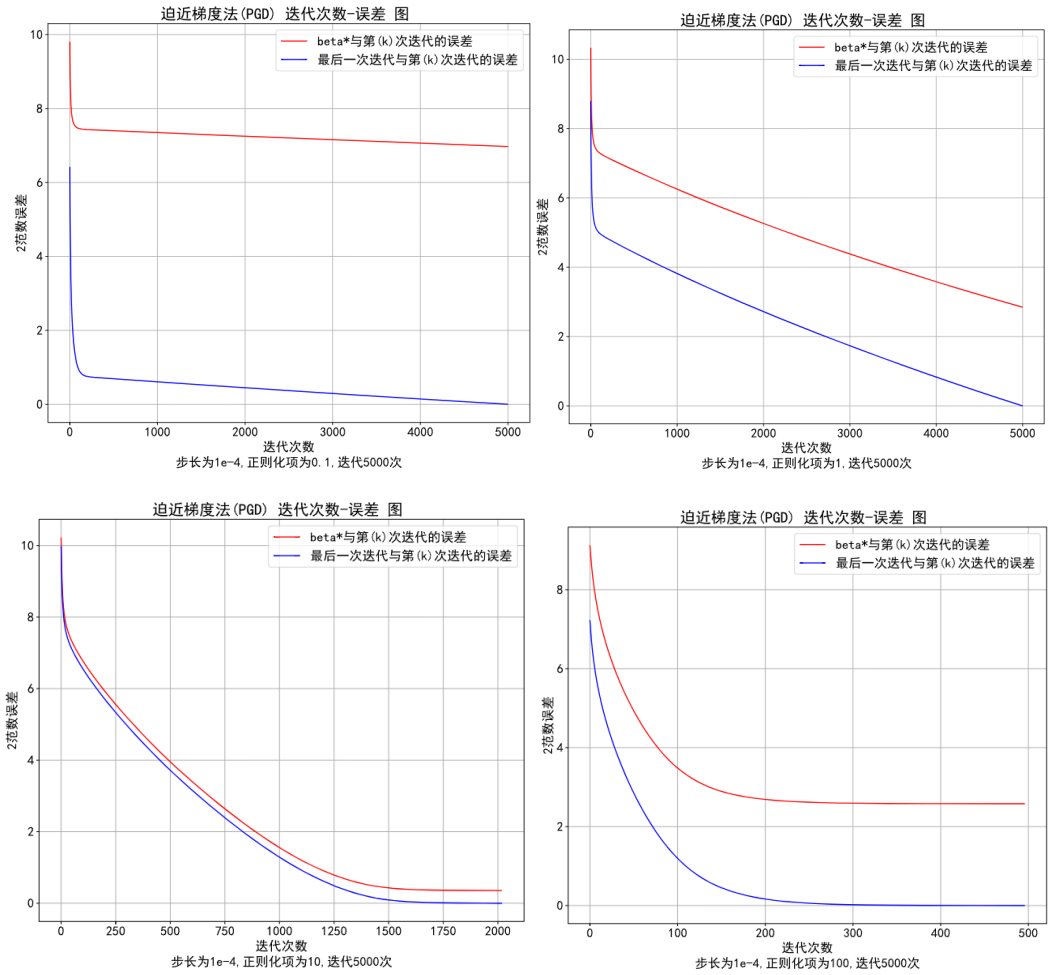


图 2 不同正则化项对迫近梯度法性能的影响

从图 2 中可以看出，当选取正则化项$\lambda=10$时，两条曲线最为接近，此时的模型效果最好，在其它的情况下，模型陷入了正则化项过大或过小的问题。

当$\lambda=0.1$或$\lambda=1$时，正则化项过小，此时函数会减弱对系数向量中的非零元素的惩罚，使得更多的特征可能被保留在模型中，导致系数向量不够稀疏。

当$\lambda=100$时，正则化项过大，此时函数会更强烈地惩罚系数向量中的非零元

素，从而促使更多的系数变为零。这种效果忽略了一些可能对模型性能有帮助的特征。

经过一系列调整参数，发现当步长值$t=3\times10^{-4}$,$\lambda=1$时，模型取得最好的性能，此时$||\beta^k-\beta^*||_2^2=0.12$,得到的结果如图 3 所示：

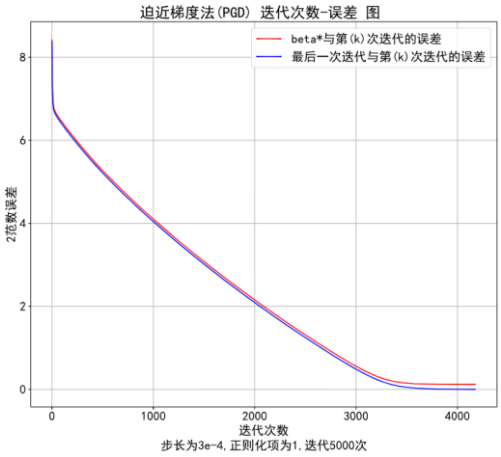
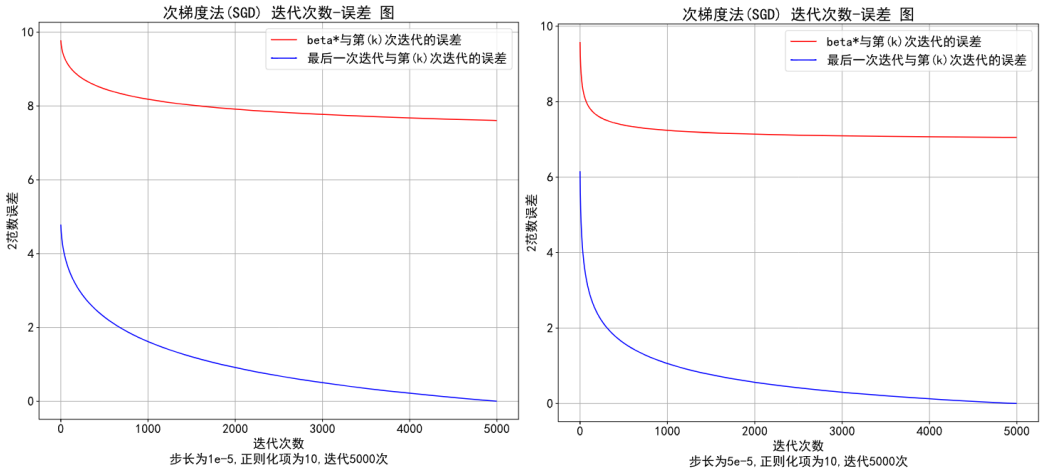


图 3 $t=3\times10^{-4}$, $\lambda=1$时模型结果

值得注意的是，当前实验结果相较于之前的尝试呈现了更好的性能，并且表现出更快的收敛速度。虽然我们在实验过程中尝试了多种超参数组合，**但未能找到比当前结果更优的组合**。这表明当前的超参数选择可能是相对较为合适的，使得算法在此配置下能够更有效地解决 Lasso 问题。这也进一步强调了超参数选择对于算法性能的关键影响，为进一步的实验和优化提供了有益的指导。

接下来讨论次梯度法的性能，考虑**步长**对算法的影响，如图 4 所示。



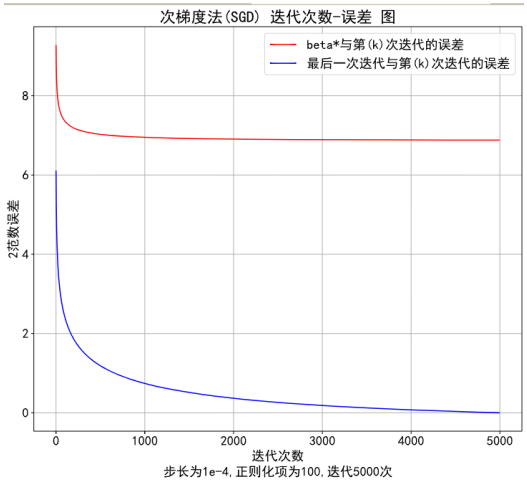
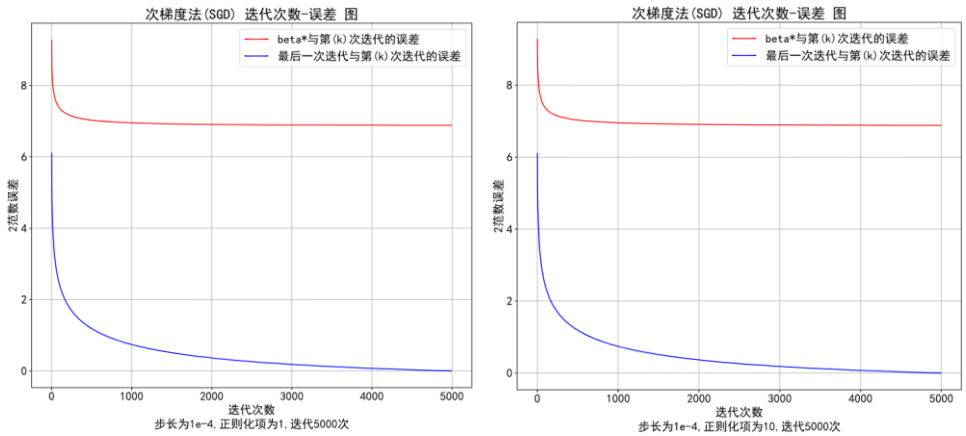


图 4 不同步长值对次梯度法性能的影响

通过观察图 4 的实验结果，我们可以得出次梯度算法相比于迫近梯度法在收敛性上并没有表现得更好。特别地，算法对步长值的改变似乎并不敏感，这可能意味着在一定范围内的步长变化对模型的性能提升不明显。

接下来考虑**正则化项**对算法的影响，实验固定步长为1×10^{-4}，选取不同的正则化系数为$0.1, 1, 10, 100$,得到的结果如图 5 所示：



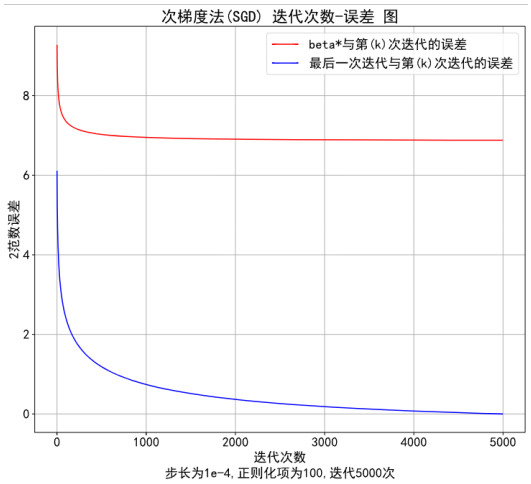


图 5 不同正则化项对次梯度法性能的影响

在图 5 中，我们观察到正则化项的改变对次梯度法的性能几乎没有明显的影响，变化微乎其微。相较之下，将次梯度法的图像与迫近梯度法进行对比，我们发现次梯度法在回归效果上并不理想。尽管次梯度法能够收敛到最优解，但最优解与真值之间存在较大差距，表明该方法在 Lasso 问题上的表现相对不佳。这进一步验证了在相同条件下，迫近梯度法相较于次梯度法更具优势，能够更好地逼近真实值，为 Lasso 问题的求解提供了更可靠的选择。

最后来查看二者的一个直观对比图，如图 6 所示：

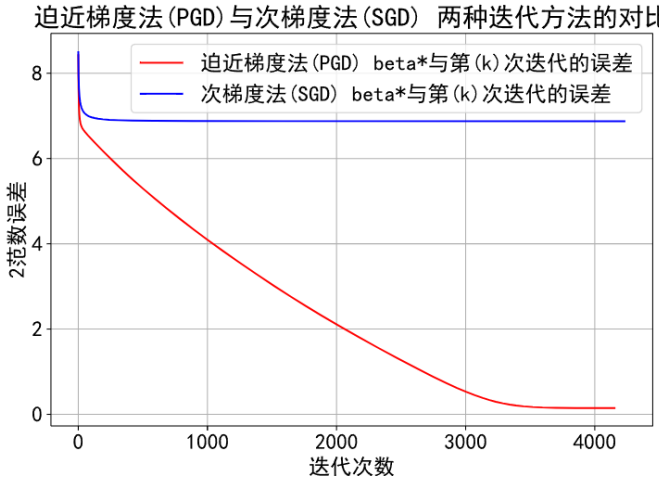


图 6 迫近梯度法与次梯度法的对比

从图中可以看到，迫近梯度法的算法能够收敛的很好，次梯度法仅在前几次迭代成功收敛，但效果并不理想。并且但从效果来说，迫近梯度法比次梯度法有着更加显著且优异的效果。

6 实验总结

总的来说，Lasso 问题作为线性回归的一种变体，通过引入 L1 正则项在模型中进行特征选择和降维，具有广泛的实际应用价值。在本次实验中，我们采用了近邻梯度法和次梯度法两种方法进行求解，通过对比两者性能表现，深入研究了 Lasso 问题的求解策略。

通过实验观察和对比分析，我们发现**近邻梯度法**在相同条件下表现更为出色，具有更好的收敛性和逼近真实值的能力。此外，通过调整超参数，特别是**步长和正则化系数**，我们发现这两个参数对模型性能有着显著的影响，为模型的调优提供了有益的指导。

通过这次实验，我们不仅深入理解了 Lasso 问题的求解方法，还在实际操作中掌握了两种方法的应用。这对于解决高维数据中的稀疏建模问题提供了深刻认识，为进一步的研究和应用奠定了基础。最终，我们得出结论，**近邻梯度法是一种可行的解决方案，而次梯度法在效果上略显不足**，需要更多的调优和改进。这些发现对于实际应用中的模型选择和参数调整提供了有益的经验和指导。