统计计算期末报告

于越

December 31, 2022

Contents

List of Figures

List of Tables

1 第一问

1.1 简单概述

第一问要求采用梯度下降法、随机梯度下降法以及坐标下降法求解参数 $\boldsymbol{beta}$ 的参数估计。本人先给出算法的相关推导过程，再详细介绍代码和算法实现，最后展示一下本人所写算法在两个数据集——鸢尾花数据集和统计计算第五次作业的生成数据集上面的结果表现。

1.2 迭代公式的推导

本人首先进行迭代公式的推导，并且给出算法的伪代码。

我们都知道，对于二元逻辑回归问题而言，其具有如下所示的结果表达形式：

$$\begin{aligned} P(y=1|x;\boldsymbol{\beta}) &= h_{\boldsymbol{\beta}}(x) = \frac{1}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}, \\ P(y=0|x;\boldsymbol{\beta}) &= 1-h_{\boldsymbol{\beta}}(x) = \frac{\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}. \end{aligned} \tag{1}$$

该式可以写为类似伯努利分布通式的形式：

$$P(y|x;\boldsymbol{\beta}) = [h_{\boldsymbol{\beta}}(x)]^{y}[1-h_{\boldsymbol{\beta}}(x)]^{1-y}. \tag{2}$$

首先，本人假定得到的数据集的样本量为 n，特征数为 p，由此可得数据 $X = [\boldsymbol{x_1}^{\mathrm{T}}, \boldsymbol{x_2}^{\mathrm{T}}, \cdots, \boldsymbol{x_n}^{\mathrm{T}}]^{\mathrm{T}}$ 是一个 $n \times p$ 的矩阵形式。而标签向量为 $\boldsymbol{y}$，其中 $y_i \in \{0,1\}, i = 1,2,\cdots,n$。由此，本人的算法目标是得到 p 维向量 $\boldsymbol{\beta}$ 的参数估计。

首先假定 logit 函数为

$$h_{\boldsymbol{\beta}}(\boldsymbol{x_i}) = g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i}) = \frac{1}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})}. \tag{3}$$

因为对于标签 $\boldsymbol{y}$ 而言，$y_i \in \{0,1\}, i = 1,2,\cdots,n$，故该估计问题可以被抽象成为一个伯努利分布有关的问题。故由最大似然估计理论，可以得到关于参数向量 $\boldsymbol{\beta}$ 的似然函数为

$$Like(\boldsymbol{\beta}) = \prod_{i=1}^{n} P(y_i|\boldsymbol{x_i},\boldsymbol{\beta}) = \prod_{i=1}^{n} [h_{\boldsymbol{\beta}}(\boldsymbol{x_i})]^{y_i}[1-h_{\boldsymbol{\beta}}(\boldsymbol{x_i})]^{1-y_i}. \tag{4}$$

对该似然函数取对数并取均值（除以样本量 n）后，便可得到类似损失的对数似然函数的形式。在该式的前面添加负号，使其成为一个下凸函数的形式，这样就可以采用常用的梯度下降方法求解。最终得到的损失函数形式为

$$L(\boldsymbol{\beta}) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h_{\boldsymbol{\beta}}(\boldsymbol{x_i}) + (1-y_i)\log[1-h_{\boldsymbol{\beta}}(\boldsymbol{x_i})]]. \tag{5}$$

而本题要求的是带 L_1 正则项的 Lasso 逻辑回归的参数估计，故损失函数为

$$\begin{aligned} L(\boldsymbol{\beta}) &= -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h_{\boldsymbol{\beta}}(\boldsymbol{x_i}) + (1-y_i)\log[1-h_{\boldsymbol{\beta}}(\boldsymbol{x_i})]] + \lambda\sum_{j=1}^{p}|\beta_j| \\ &= \frac{1}{n}\sum_{i=1}^{n}[-y_i\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i} + \log(1+\exp(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i}))] + \lambda\sum_{j=1}^{p}|\beta_j|. \end{aligned} \tag{6}$$

梯度下降法和坐标下降法的核心都是需要求解损失函数的导数。首先对函数 $h_{\boldsymbol{\beta}}(\boldsymbol{x_i})$ 的 $\boldsymbol{\beta}$ 的分量求导，可得

$$\frac{\partial h_{\boldsymbol{\beta}}(\boldsymbol{x_i})}{\partial \beta_j} = \frac{\exp(-\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})}{(1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i}))^2}\frac{\partial \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i}}{\partial \beta_j} = g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})[1-g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})]\frac{\partial \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i}}{\partial \beta_j} \tag{7}$$

故

$$\begin{aligned} \frac{\partial L(\boldsymbol{\beta})}{\partial \beta_j} &= -\frac{1}{n}\sum_{i=1}^{n}\left[\frac{y_i}{g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})} - \frac{1-y_i}{1-g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})}\right]\frac{g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})}{\partial \beta_j} + \lambda sign(\beta_j) \\ &= -\frac{1}{n}\sum_{i=1}^{n}\left[\frac{y_i}{g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})} - \frac{1-y_i}{1-g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})}\right]g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})[1-g(\boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i})]\frac{\partial \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x_i}}{\partial \beta_j} + \lambda sign(\beta_j) \\ &= \frac{1}{n}\sum_{i=1}^{n}[h_{\boldsymbol{\beta}}(\boldsymbol{x_i}) - y_i]x_{ij} + \lambda sign(\beta_j), \quad i = 1,2,\cdots,n \quad j = 1,2,\cdots,p. \end{aligned} \tag{8}$$

由此可得梯度下降法的每个参数分量 β_j 的迭代公式为：

$$\beta_j^{(t+1)} = \beta_j^{(t)} - \alpha\frac{1}{n}\sum_{i=1}^{n}[h_{\boldsymbol{\beta}}(\boldsymbol{x_i}) - y_i]x_{ij} - \lambda sign(\beta_j), \quad i = 1,2,\cdots,n \quad j = 1,2,\cdots,p. \tag{9}$$

其中 $sign$ 为符号函数，即

$$sign(x) = \begin{cases} 1, & x > 0 \\ -1, & x < 0 \\ 0, & x = 0. \end{cases} \tag{10}$$

该式也为坐标下降法的任意选得的坐标分量的下降迭代公式。

1.3 近端梯度下降方法

原本在得知梯度下降的迭代计算公式之后，事实上就可以进行计算。但是这里的逻辑回归是一个带 L_1 正则项的 Lasso 逻辑回归。带 L_1 正则项的损失函数常常采用近段梯度下降的算法进行求解，在这里本人将这一算法应用到所写的代码当中，以尽可能保证计算的准确性。

不妨假定待优化损失函数为 $f(\boldsymbol{x})$。故带 L_1 正则项的损失函数优化目标为

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \lambda||\boldsymbol{x}||_1. \tag{11}$$

若 $f(\boldsymbol{x})$ 可导，且导数 ∇f 满足 Lipschitz 条件，即存在常数 $L>0$，对 $\forall \boldsymbol{x}, \boldsymbol{x'}$ 有

$$||\nabla f(\boldsymbol{x'}) - \nabla f(\boldsymbol{x})||_2^2 \leq L||\boldsymbol{x'} - \boldsymbol{x}||_2^2. \tag{12}$$

则在 $\boldsymbol{x^{(t)}}$ 附近可将 $f(\boldsymbol{x})$ 通过二阶 Taylor 展开近似为

$$\begin{aligned} \hat{f}(\boldsymbol{x}) \approx & f(\boldsymbol{x^{(t)}}) + \nabla f(\boldsymbol{x^{(t)}})(\boldsymbol{x} - \boldsymbol{x^{(t)}}) + \frac{L}{2}||\boldsymbol{x} - \boldsymbol{x^{(t)}}||^2 \\ = & \frac{L}{2}||\boldsymbol{x} - (\boldsymbol{x^{(t)}} - \frac{1}{L}\nabla f(\boldsymbol{x^{(t)}}))||_2^2 + const. \end{aligned} \tag{13}$$

显然上式的最小值在

$$\boldsymbol{x^{(t+1)}} = \boldsymbol{x^{(t)}} - \frac{1}{L}\nabla f(\boldsymbol{x^{(t)}}) \tag{14}$$

时取到。

故通过梯度下降法对 $f(\boldsymbol{x})$ 进行最小化求解，每一步梯度下降的迭代实际上等价于最小化二次函数 $\hat{f}(\boldsymbol{x})$。则可以得到对于 (11) 式的近段梯度下降迭代应该写作

$$\boldsymbol{x^{(t+1)}} = \arg\min_{\boldsymbol{x}} \frac{L}{2}||\boldsymbol{x} - (\boldsymbol{x^{(t)}} - \frac{1}{L}\nabla f(\boldsymbol{x^{(t)}}))||_2^2 + \lambda||\boldsymbol{x}||_1. \tag{15}$$

对于上式，可以先计算 $\boldsymbol{z} = \boldsymbol{x^{(t)}} - \frac{1}{L}\nabla f(\boldsymbol{x^{(t)}})$，再求解

$$\boldsymbol{x^{(t+1)}} = \arg\min_{\boldsymbol{x}} \frac{L}{2}||\boldsymbol{x} - \boldsymbol{z}||_2^2 + \lambda||\boldsymbol{x}||_1. \tag{16}$$

设 $x_i^{(t)}$ 为 $\boldsymbol{x^{(t)}}$ 的第 i 个分量，因为 (16) 式中并不存在 $x_i^{(t)}x_j^{(t)}(i \neq j)$ 这样的项，即 $\boldsymbol{x^{(t)}}$ 的各分量之间互不影响，故 (16) 式有如下形式的解:

$$x_i^{(t+1)} = \begin{cases} z_i - \frac{\lambda}{L}, & \frac{\lambda}{L} < z_i \\ 0, & |z_i| \leq \frac{\lambda}{L} \\ z_i + \frac{\lambda}{L}, & z_i < -\frac{\lambda}{L}. \end{cases} \tag{17}$$

近段梯度下降方法是用于求解 Lasso 的一个经典方法。具体的方法讲解可以参考周志华老师的《机器学习》（西瓜书）第 252-253 页。在这里，本人把这一方法用于 Lasso 逻辑回归的梯度下降和坐标下降方法当中。

1.4 算法代码讲解

先给出梯度下降法和坐标下降法的算法如下 Algorithm1和3所示：

Algorithm 1: 梯度下降算法和随机梯度下降算法

1: 初始化：设置学习率 α，迭代初值 $\boldsymbol{\beta^{(0)}}$，迭代终止阈值 ε，正则化超参数 λ，用于近段梯度下降控制 Lipschitz 条件的 L，最大迭代次数 max_iter 以及一次迭代所用的一批样本的数量 $batch\_n$。

2: 这里的 $batch\_n$ 用于控制一次迭代过程中所用的样本数量。如果设置 $batch\_n = 1$，即每次迭代只随机使用一个样本，这样算法就是随机梯度下降；如果设置 $1 < batch\_n < n$，即一次迭代所用的一批样本量大于 1 小于全部样本数 n，即为小批量随机梯度下降；若设置 $batch\_n = -1$，即表示采用全部的样本，此时即为经典的梯度下降算法。

3: 对需要估计的参数向量 $\boldsymbol{\beta}$ 进行迭代计算，其每个分量的迭代公式为 (9) 式。在迭代过程中，每个分量采用 (15)-(17) 式的近段梯度下降算法进行计算，这样通过 $\boldsymbol{\beta^t}$ 即可求得下一步的算法迭代结果 $\boldsymbol{\beta^{t+1}}$。

4: 对于第三步，若迭代过程中的参数改变量 $||\boldsymbol{\beta^{t+1}} - \boldsymbol{\beta^t}||_2 < \varepsilon$，即参数改变量小于设置的终止阈值，或者达到了设置的最大迭代次数 max_iter，则终止算法，输出最终参数估计结果 $\hat{\boldsymbol{\beta}}$；否则继续重复第三步的迭代计算，直到算法收敛为止。

Algorithm 2: 坐标下降算法

1: 初始化：设置学习率 α，迭代初值 $\boldsymbol{\beta^{(0)}}$，迭代终止阈值 ε，正则化超参数 λ，用于近段梯度下降控制 Lipschitz 条件的 L，最大迭代次数 max_iter，一次迭代所用的一批样本的数量 $batch\_n$ 以及坐标选择的准则 $rand$。

2: 这里的 $batch\_n$ 默认设置为-1，即默认采用全部样本。$rand$ 是用于控制坐标下降的过程中是否采用随机选择坐标下降。若为 True 则为随机选择任一坐标维度下降，为 False 则为按照参数的顺序循环选择坐标维度下降。

3: 对需要估计的参数向量 $\boldsymbol{\beta}$ 进行迭代计算，其单个坐标维度下降的迭代公式为 (9) 式。在迭代过程中，每个分量采用 (15)-(17) 式的近段梯度下降算法进行计算，这样通过 β_i^t 即可求得下一步的算法迭代结果 β_i^{t+1}。

4: 对于第三步，若迭代过程中的参数改变量 $||\beta_i^{t+1} - \beta_i^t||_2 < \varepsilon$，即参数改变量小于设置的终止阈值，或者达到了设置的最大迭代次数 max_iter，则终止算法，输出最终参数估计结果 $\hat{\boldsymbol{\beta}}$；否则继续重复第三步的迭代计算，直到算法收敛为止。

在算法1当中，本人采用一个 $batch\_n$ 参数控制算法是采用梯度下降法还是随机梯度下降法（小批量或是单个样本），以此将 2 个算法整合到了一起。本人的代码和 sklearn 库当中的算法实现类似，也是通过先定义一个函数方法类的方式，在每一个类当中写入需要的函数。在实际调用方法时也和 sklearn 库的使用方法类似，需要先对方法类实例化，实例化完成后再调用方法类当中的函数，从而对数据进行拟合和预测。函数类中的

fit 函数用于对训练集数据进行训练，迭代计算得到 Lasso 逻辑回归的参数估计。predict 函数用于代入参数估计对测试集数据进行测试，得到测试集上预测的标签结果。

1.5 实际数据集上效果展示

在这里，本人采用两个数据集——鸢尾花数据集和统计计算第五次作业的数据集来测试一下本人所写的 Lasso 逻辑回归算法效果。

1.5.1 鸢尾花数据集

鸢尾花数据集是一个经典的分类数据集，该数据集共计 150 条样本，其具有花萼长度（Sepal Length）、花萼宽度（Sepal Width）、花瓣长度（Petal Length）、花瓣宽度（Petal Width）四个属性。标签 0、1、2 分别表示山鸢尾（Setosa）、变色鸢尾（Versicolor）、维吉尼亚鸢尾（Virginical）三种不同类型。在这里，因为本人实现的二分类的 Lasso 逻辑回归算法，因此仅仅选取其中的标签为 0 和 1 的山鸢尾与变色鸢尾样本进行试验。

在筛选出全部标签为 0 和 1 的样本后，以 7:3 的比例划分训练集与测试集；本人设置的实例初始化参数如下表1所示：

Table 1: Lasso 逻辑回归梯度下降参数设置（鸢尾花）

参数设置	参数值
学习率 α	0.01
初始参数向量 $\boldsymbol{\beta^{(0)}}$	$[1,1,1,1]^{\mathrm{T}}$
迭代终止阈值 ε	0.0001
正则化超参数 λ	0.01
近端梯度 Lipschitz 控制参数 L	1
最大迭代次数	1000

本人先设置 $batch\_n = -1$，即用上所用的训练集数据进行梯度下降算法的试验。将在训练集上训练后得到的参数估计作用于测试集，将预测结果与真实测试集结果比较，得到的预测准确率、召回率、查准率以及 f1-score 如下图1所示：可以看到，对于测试集

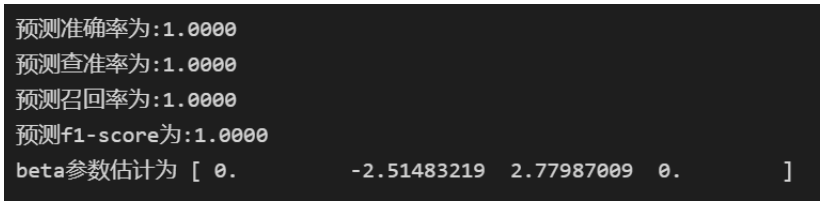


Figure 1: 梯度下降算法作用于鸢尾花数据集的测试结果

上所用的样本预测结果均正确；值得注意的是，从最后得到的参数估计可以发现，花萼

长度与花萼宽度两个特征对应的参数为 0。这很好地体现了 Lasso 的效果，通过压缩的方式，将一些特征对应参数变为 0，从而起到稀疏化与特征选择的效果。

接着设置表1当中的参数不变，修改 $batch\_n = 20$，即每次迭代采用 20 个样本的小批量随机梯度下降算法进行参数估计，并作用于测试集，将预测结果与真实测试集结果比较，得到的预测准确率、召回率、查准率以及 f1-score 如下图2所示：可以看到，测试

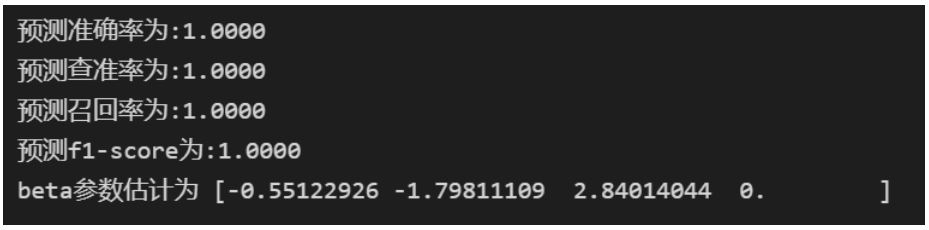


Figure 2: 小批量随机梯度下降算法作用于鸢尾花数据集的测试结果

集上的样本预测结果还是完全正确的。只是此时仅有花萼宽度这一个特征前面的参数对应为 0，Lasso 的特征选择特性此时变弱了一些。事实上这是可以预料的，因为此时并没有采用全部的样本，可能会导致一定程度上的信息丢失，因而特征选择机制也变弱了一些。

随机梯度下降的特性在于算法速度明显加快。采用全部样本的梯度下降算法需要 0.6 秒进行计算，而采用 20 个样本的小批量梯度下降算法仅仅需要 0.2 秒即可。

对于坐标下降算法而言，本人设置的参数和表1的完全不变，对于坐标维度选择参数 rand 本人设置为 False，即并不使用随机选择迭代下降坐标维度的形式，而是按照顺序循环坐标维度下降。对于训练集数据训练后，作用于测试集，得到的结果如下图3所示：从结果上来看，分类结果完全正确；但是由于坐标下降算法的原因，梯度的下降是

Figure 3: 坐标下降算法作用于鸢尾花数据集的测试结果

一个维度一个维度依次进行的，因而在算法执行过程中势必会发生信息丢失的情况，这里的 Lasso 特征选择效应并不明显。

与此同时，本人采用 sklearn 库当中的逻辑回归算法进行拟合，设置逻辑回归惩罚项为 L_1 惩罚，且采用线性逻辑回归，得到的测试集指标以及参数估计结果如下图11所示：

比较图片11与1的结果，发现参数估计的结果十分接近，且花萼长度和花萼宽度的对应参数都为 0。这说明本人的 Lasso 逻辑回归梯度下降算法效果比较可观，对于参数估计的较为准确。

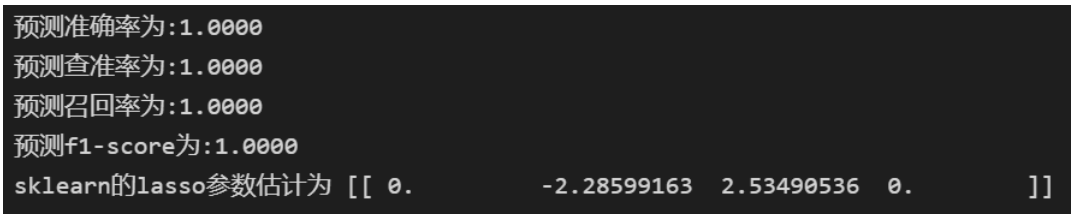


Figure 4: sklearnL_1 逻辑回归作用于鸢尾花数据集的测试结果

1.5.2 统计计算第五次作业数据集

从之前的鸢尾花数据集结果来看，无论采用何种算法，分类的准确率均为 100%。这可能和本人恰好选择了鸢尾花数据集上相距较远的两个数据集有关，这使得最后的结果不那么具有代表性和说服力。因此在这里，本人重新使用统计计算第五次作业的数据集进行算法的测试。该算法为助教通过逻辑回归的表达式手动生成。本人将所有的样本以 8:2 的比例划分训练集与测试集。

本人设置的实例初始化参数如下表2所示：

Table 2: Lasso 逻辑回归梯度下降参数设置

参数设置	参数值
学习率 α	0.1
初始参数向量 $\boldsymbol{\beta}^{(\mathbf{0})}$	$[1,1,1,1,1]^{\mathrm{T}}$
迭代终止阈值 ε	0.0001
正则化超参数 λ	0.01
近端梯度 Lipschitz 控制参数 L	2
最大迭代次数	1000

本人先设置 $batch\_n = -1$，即用上所用的训练集数据进行梯度下降算法的试验。将在训练集上训练后得到的参数估计作用于测试集，将预测结果与真实测试集结果比较，得到的预测准确率、召回率、查准率以及 f1-score 如下图5所示：

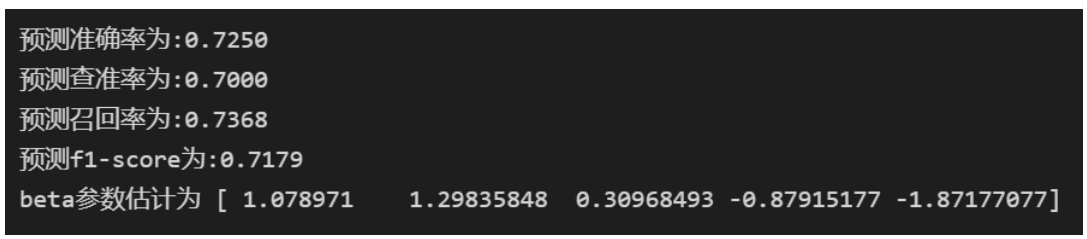


Figure 5: 梯度下降算法作用于第五次作业数据集的测试结果

可以看到此时的算法估计测试准确率大约在 72.50% 左右。但是直接看看不出结果好坏，本人还是采用 sklearn 库当中的逻辑回归算法进行拟合，设置逻辑回归惩罚项为 L_1 惩罚，得到的测试集指标以及参数估计结果如下图15所示：

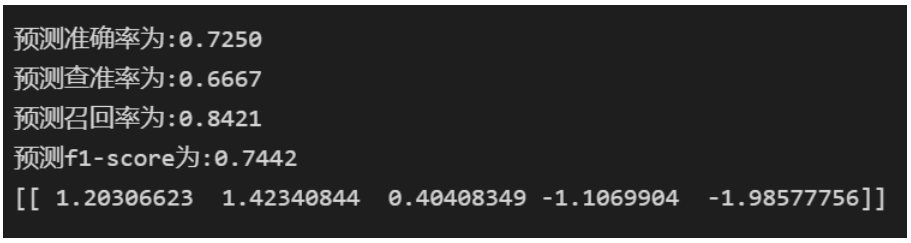


Figure 6: sklearnL_1 逻辑回归作用于第五次作业数据集的测试结果

将本人实现的算法和 sklearn 库的官方参数估计结果比较，发现预测准确率均为 72.50%，而本人实现的算法查准率更高，sklearn 库的官方算法召回率更高，可以说相比之下平分秋色，本人算法的 f1-score 略逊于官方 sklearn 库的结果。而比较 $\boldsymbol{\beta}$ 的参数估计结果，发现二者差别也并不大。由此说明本人实现的 Lasso 逻辑回归的梯度下降算法是比较有效地。

与此同时，本人设置算法 $batch\_n = 20$，通过小批量随机梯度下降方法看一下效果，结果如下图7所示：

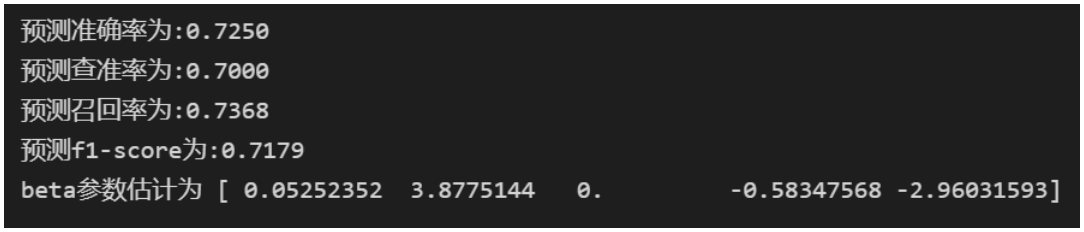


Figure 7: 小批量随机梯度下降算法作用于第五次作业数据集的测试结果

可以看到，算法的预测效果和梯度下降法的效果完全一致。在参数估计上，小批量随机梯度下降方法没有那么“保守”，特别是对于第三个特征而言，其直接稀疏化为 0，这一定程度上体现了 Lasso 逻辑回归的特征选择机制。

对于坐标下降算法，本人设置坐标维度选择参数 rand 为 False，采用循环选择坐标维度进行梯度下降，并展示结果如下图8所示：

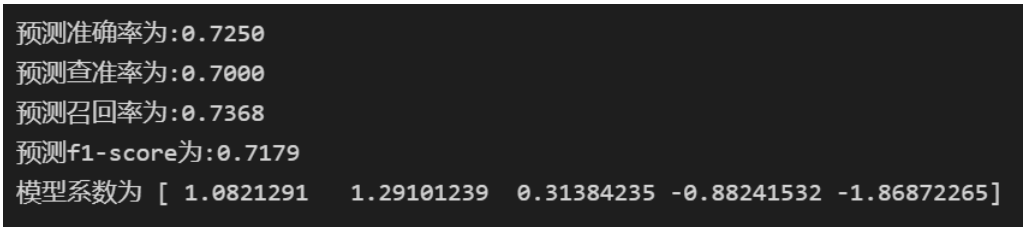


Figure 8: 坐标下降算法作用于第五次作业数据集的测试结果

该结果同样和梯度下降算法类似，且参数估计结果也近乎一致。鉴于坐标下降算法一次只用采用一个维度进行计算，因而运算速度会更快一些，这是坐标梯度下降算法的优势。

1.6 正则超参数 λ 的交叉验证展示

在 Lasso 逻辑回归当中，因为涉及 L_1 正则化，正则化超参数 λ 的选择就比较重要。在这里，本人采用经典的 K 折交叉验证的方法选择超参数 λ。在这里，本人给出程序代码，并围绕程序代码进行讲解。

```
from sklearn.model_selection import KFold # 从sklearn导入KFold包

#输入数据推荐使用numpy数组，使用list格式输入会报错
def K_Fold_spilt(K,fold,data,label):
    '''
    :param K: 要把数据集分成的份数。如十次十折取K=10
    :param fold: 要取第几折的数据。如要取第5折则fold=4(第一折为0)
    :param data: 需要分块的数据
    :param label: 对应的需要分块标签
    :return: 对应折的训练集、测试集和对应的标签
    '''
    split_list = []
    kf = KFold(n_splits=K,shuffle=True,random_state=55) #shuffle=True:乱序分配
    for train, test in kf.split(data):
    split_list.append(train.tolist())
    split_list.append(test.tolist())
    train,test=split_list[2 * fold],split_list[2 * fold + 1]
    return data[train], data[test], label[train], label[test] #已经分好块的数据集
```

首先定义一个 K_Fold_split 函数，用于将数据集分成 K 份。将数据分为 K 份后，再从中依次取出每一份作为测试集，余下的作为训练集，且将特征与标签分开，就完成了 K 折交叉验证的划分。

```
def plot_curve(x_list,y_list,label,color,ylabel,title):
    plt.figure(figsize=(12,8))
    plt.plot(x_list,y_list,color=color,label=label)
    plt.xlabel('参数的选取范围')
    plt.ylabel(ylabel)
    plt.xticks(rotation=45)
    plt.title(title)
    plt.legend()
    plt.show()
```

接着定义一个绘制不同 λ 上指标的函数 plot_curve。本人在这里对于不同的正则化超参数 λ，仍旧和之前一样，采用测试集上的预测结果的预测准确率、召回率、查准

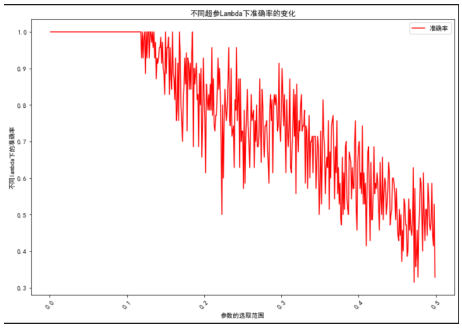
率以及 f1-score 作为评判指标，选择指标更高的 λ 作为 lasso 逻辑回归的超参数。

```
lamb_list = np.arange(0.001,0.5,0.001)
acc_lambda_list = []
rec_lambda_list = []
pre_lambda_list = []
f1_lambda_list = []
for lamb in lamb_list:
  K = 10
  acc_list = []
  rec_list = []
  pre_list = []
  f1_list = []
  for fold in range(K):
    fold_res = K_Fold_spilt(K,fold,train_x,train_y)
    x_train = fold_res[0]
    x_val = fold_res[1]
    y_train = fold_res[2]
    y_val = fold_res[3]

    ini_theta = np.array([1]*x_train.shape[1],dtype=float)
    l=LogisticRegression_L1(alpha=0.01,theta=ini_theta,valve=0.0001,
    lamb=lamb,L=1,batch_n=-1,max_iter=1000)
    m = l.fit(x_train,y_train)
    pred_yval = m.predict(x_val)

    acc = accuracy_score(y_val,pred_yval)
    rec = recall_score(y_val,pred_yval)
    pre = precision_score(y_val,pred_yval)
    f1 = f1_score(y_val,pred_yval)
    acc_list.append(acc)
    rec_list.append(rec)
    pre_list.append(pre)
    f1_list.append(f1)

    acc_lambda_list.append(np.mean(acc_list))
    rec_lambda_list.append(np.mean(rec_list))
    pre_lambda_list.append(np.mean(pre_list))
    f1_lambda_list.append(np.mean(f1_list))
```

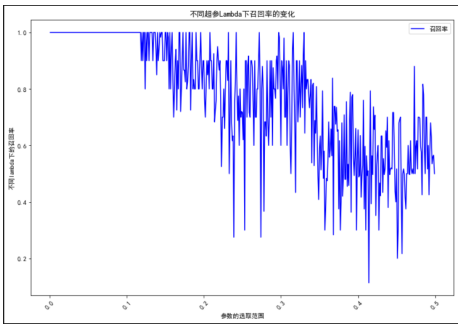
```
plot_curve(lamb_list,acc_lambda_list,'准确率','red',
'不同lambda下的准确率','不同超参Lambda下准确率的变化')
plot_curve(lamb_list,rec_lambda_list,'召回率','blue',
'不同lambda下的召回率','不同超参Lambda下召回率的变化')
plot_curve(lamb_list,pre_lambda_list,'查准率','green',
'不同lambda下的查准率','不同超参Lambda下查准率的变化')
plot_curve(lamb_list,f1_lambda_list,'f1值','purple',
'不同lambda下f1值','不同超参Lambda下f1值的变化')
```

最后设定 K 折交叉验证的 K，作用于数据集，计算测试集上的各项指标，将 K 折交叉验证的一个 λ 上的指标均值作为确定一个 λ 情况下的真正指标值，将这些指标值依次存储，再绘制该指标值随超参数 λ 变化的学习曲线，依据该学习曲线就可以选择对应的最优超参数 λ。

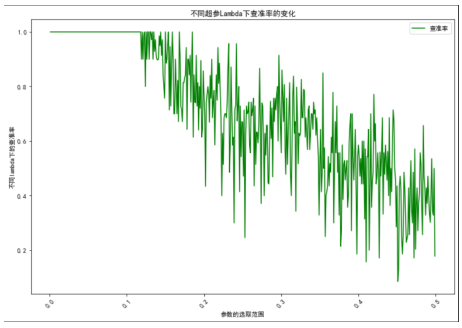
在这里，本人以鸢尾花数据集为例，绘制出测试集上预测准确率、召回率、查准率以及 f1-score 随 λ 变化的学习曲线，如下图所示：



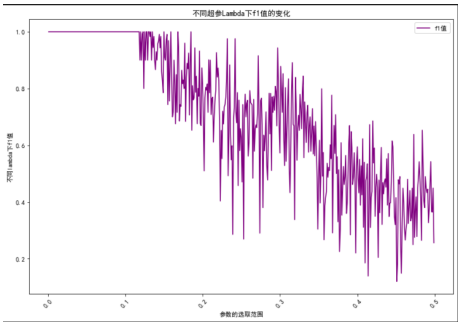
(a) 预测准确率学习曲线



(b) 召回率学习曲线



(c) 查准率学习曲线



(d) f1-score 学习曲线

Figure 9: 超参数 λ 学习曲线

从学习曲线上可以看到，当 $\lambda < 0.1$ 时，预测的结果较好，而当 $\lambda > 0.1$ 时，预测准确率发生波动且呈缓慢下降趋势。在这里可以看到曲线的走势较为一致，这体现了对于鸢尾花数据集，选择的正则化超参数较大时，Lasso 逻辑回归的分类效果会变差。因

此对于该数据集选择 $\lambda < 0.1$ 即可。

该部分的其他代码可以参见附录部分。

2 第二问

2.1 简单概述

第二问要求采用 MCMC 算法实现贝叶斯逻辑回归算法，即给定先验情况下 $\boldsymbol{\beta}$ 的后验估计。在这里本人还是先给出算法推导过程，随后基于算法的推导过程介绍算法的实现过程，最后采用鸢尾花数据集以及统计计算第五次作业的数据集对算法进行测试。

2.2 算法推导

对于二元逻辑回归问题而言，其具有如下所示的结果表达形式：

$$\begin{aligned} P(y=1|x;\boldsymbol{\beta}) &= h_{\boldsymbol{\beta}}(x) = \frac{1}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}, \\ P(y=0|x;\boldsymbol{\beta}) &= 1-h_{\boldsymbol{\beta}}(x) = \frac{\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}. \end{aligned} \tag{18}$$

该式可以写为类似伯努利分布通式的形式：

$$P(y|x;\boldsymbol{\beta}) = [h_{\boldsymbol{\beta}}(x)]^{y}[1-h_{\boldsymbol{\beta}}(x)]^{1-y}. \tag{19}$$

在这里，本人采用了两个不同形式的先验：Laplace 先验以及正态分布先验，作为 β_j 的先验。为了便于先验分布的生成，在这里本人假定 $\boldsymbol{\beta}$ 的各分量之间没有相关关系，即每个分量的先验分布都是独立生成的。

当采用正态分布作为先验，即 $\beta_j \sim N(\mu_j, \sigma_j^2), j = 1, 2, \cdots, p$ 时，鉴于本人假定 β_j 独立同分布，此时的后验分布可以写成如下形式：

$$\begin{aligned} &p(\boldsymbol{\beta}|X,\boldsymbol{y}) \propto \pi(\boldsymbol{\beta})f(\boldsymbol{y}|\boldsymbol{\beta}) \\ &= \prod_{i=1}^{n}\left(\frac{1}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}\right)^{y_i}\left(\frac{\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}\right)^{1-y_i}\prod_{j=1}^{p}\frac{1}{\sqrt{2\pi}\sigma_j}\exp\left[-\frac{(\beta_j-\mu_j)^2}{2\sigma_j^2}\right]. \end{aligned} \tag{20}$$

事实上，对于该式取对数，可以得到和第一问当中类似的 Lasso 逻辑回归的形式，只是此时的正则项为 2-范数，即取正态分布作为先验，得到的后验分布实际上是基于 L_2 正则的逻辑回归形式。

而当采用 Laplace 分布作为先验时，即 $\beta_j \sim La(\mu_j, \sigma_j), j = 1, 2, \cdots, p$ 时，有

$$\pi(\beta_j) = \frac{1}{2\sigma_j}\exp\left(-\frac{|\beta_j-\mu_j|}{\sigma_j}\right). \tag{21}$$

此时的后验分布可以写成如下形式：

$$\begin{aligned} &p(\boldsymbol{\beta}|X,\boldsymbol{y}) \propto \pi(\boldsymbol{\beta})f(\boldsymbol{y}|\boldsymbol{\beta}) \\ &= \prod_{i=1}^{n}\left(\frac{1}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}\right)^{y_i}\left(\frac{\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}{1+\exp(-\boldsymbol{\beta}^{\mathrm{T}}x)}\right)^{1-y_i}\prod_{j=1}^{p}\frac{1}{2\sigma_j}\exp\left(-\frac{|\beta_j-\mu_j|}{\sigma_j}\right). \end{aligned} \tag{22}$$

若同样对该式取对数，就可以得到第一问的 Lasso 逻辑回归形式，正则项为 $\boldsymbol{\beta}$ 的 1-范数的形式。说明若取 Laplace 分布作为先验，得到的后验分布实际上是基于 L_1 正则的逻辑回归形式。

但是观察 (20) 式与 (22) 式的形式，不难看出，想要采用传统的算法对参数 $\boldsymbol{\beta}$ 是较为困难的。因此需要采用 MCMC 算法对参数估计进行模拟，直到参数的迭代收敛，再得到参数 $\boldsymbol{\beta}$ 的估计。

2.3 算法代码讲解

本人采用的 MCMC 算法是随机游走算法。在这里，本人还是将算法写成方法类的形式，在使用之前，先传入参数对该类进行实例化，接着方可调用方法类当中的函数进行参数估计。

具体的算法流程如下所示：

Algorithm 3: 随机游走算法

1: 初始化：设置迭代初值 $\boldsymbol{\beta^{(0)}}$，用于生成多元对称噪声的多元正态分布的协方差矩阵 Σ，先验分布的位置参数向量 $\boldsymbol{\mu}$，先验分布的尺度参数向量 $\boldsymbol{\sigma}$，最大迭代次数 max_iter。用这些参数对方法类进行实例化。

2: 依据此前设置的位置参数向量 $\boldsymbol{\mu}$ 与尺度参数向量 $\boldsymbol{\sigma}$，构造先验分布 $\pi(\boldsymbol{\beta})$。

3: 依据生成的先验分布 $\pi(\boldsymbol{\beta})$ 以及已有的数据特征及标签，计算出 (20) 式与 (22) 式的后验分布形式 $p(\boldsymbol{\beta}|X, \boldsymbol{y})$.

4: 在得到第 t 步迭代的参数估计 $\boldsymbol{\beta^t}$ 的情形下，依据对称分布 $N(\mathbf{0}, \Sigma)$ 生成噪声 $\boldsymbol{\epsilon}$，同时令 $\boldsymbol{\theta} = \boldsymbol{\beta^t} + \boldsymbol{\epsilon}$.

5: 通过均匀分布生成 $u \sim U(0, 1)$。与此同时令

$$\boldsymbol{\beta^{t+1}} = \begin{cases} \boldsymbol{\theta}, & u \leq \alpha(\boldsymbol{\beta^t}, \boldsymbol{\theta}) \\ \boldsymbol{\beta^t}, & u > \alpha(\boldsymbol{\beta^t}, \boldsymbol{\theta}). \end{cases} \tag{23}$$

其中

$$\alpha(\boldsymbol{\beta^t}, \boldsymbol{\theta}) = \min\{1, \frac{p(\boldsymbol{\theta}|X, \boldsymbol{y})}{p(\boldsymbol{\beta^t}|X, \boldsymbol{y})}\} \tag{24}$$

.

6: 重复步骤 4-5，直到达到最大迭代次数 max_iter。

在方法类当中，本人主要设置了 fit 与 predict 两个函数。fit 函数用于对训练集数据进行训练，得到基于训练集数据的 $\boldsymbol{\beta}$ 的参数估计。对于得到的参数估计结果，利用 predict 函数用于测试集上的标签预测。

值得一提的是，本人在 fit 函数当中设置了一个 method 参数，改参数的可选值为'l1'与'l2'。若选择 method='l1'，则先验通过 Laplace 分布生成，此时得到的参数估计可以看成是基于 L_1 正则的逻辑回归参数结果；相对应的，若选择 method='l2'，则先验通过正态分布生成，此时得到的参数估计可以看成是基于 L_2 正则的逻辑回归参数结果。

2.4 实际数据集上的效果展示

在此处和之前类似，本人同样采用鸢尾花数据集和统计计算第五次作业生成的逻辑回归数据集来进行试验。

2.4.1 鸢尾花数据集

在鸢尾花数据集上，对于此处的二分类问题，同样选择标签为 0 和 1 的样本，并以 7:3 的比例划分训练集与测试集。

本人设定的实例初始化参数如下表所示：

Table 3: 逻辑回归 MCMC 参数设置（鸢尾花）

参数设置	参数值
初始参数向量 $\boldsymbol{\beta^{(0)}}$	$[1,1,1,1]^{\mathrm{T}}$
协方差矩阵 Σ	单位矩阵
位置参数向量 $\boldsymbol{\mu}$	$[0,0,0,0]^{\mathrm{T}}$
尺度参数向量 $\boldsymbol{\sigma}$	$[1,1,1,1]^{\mathrm{T}}$
最大迭代次数	10000

本人对于 Laplace 先验与正态分布先验都进行了试验，与此前第一问类似，同样比较测试集上的各项指标，并且输出最终的参数估计结果。

先给出以 Laplace 分布作为先验的参数估计结果，以及该参数估计作用于测试集的各项指标值，将该结果与 sklearn 库当中的 L_1 正则逻辑回归结果比较，相关结果如下图10与11所示：

```
预测准确率为:1.0000
预测查准率为:1.0000
预测召回率为:1.0000
预测f1-score为:1.0000
模型系数结果beta为 [-0.97252456 -1.06642838  3.81230857 -0.64347327]
```

Figure 10: Laplace 分布先验 MCMC 算法作用于鸢尾花数据集结果

```
预测准确率为:1.0000
预测查准率为:1.0000
预测召回率为:1.0000
预测f1-score为:1.0000
[[ 0.         -2.27827083  2.56153167  0.        ]]
```

Figure 11: sklearnL_1 逻辑回归作用于鸢尾花数据集结果

同样地以正态分布作为先验进行试验，将试验结果与 sklearn 库当中的 L_2 正则逻辑回归结果比较，相关结果如下图12与13所示：

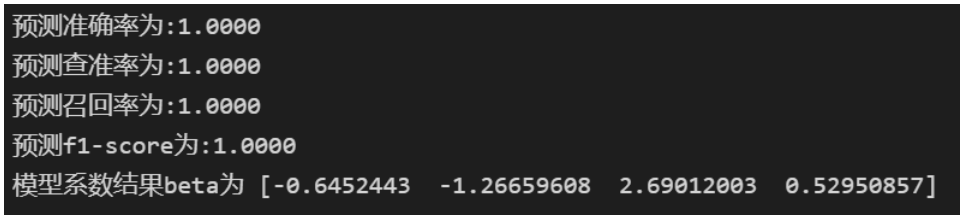


Figure 12: 正态分布先验 MCMC 算法作用于鸢尾花数据集结果

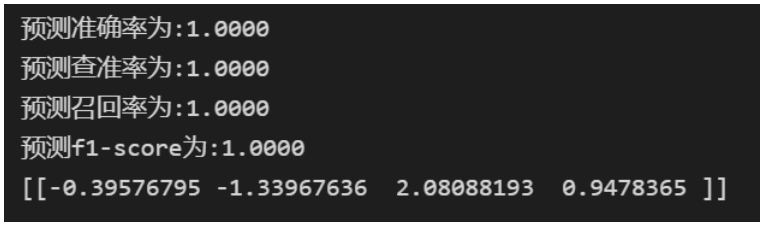


Figure 13: sklearnL_2 逻辑回归作用于鸢尾花数据集结果

从结果上可以看出以正态分布作为先验的 MCMC 算法得到的参数估计结果与 sklearn 库当中 L_2 正则下的逻辑回归的参数估计结果相对接近一些；而以 Laplace 分布作为先验的 MCMC 算法却很难达到 L_1 正则下的逻辑回归的特征选择效果。

与此前类似，选用鸢尾花数据集较难比较出测试集上分类效果的好坏。故本人还采用第五次作业的逻辑回归数据集进行试验。

2.4.2 第五次作业的逻辑回归数据集

和之前一样，本人还是以 8:2 的比例划分训练集与测试集。在此处，本人设置的参数和表3的大相径庭。只是由于该数据集的分类并没有此前的鸢尾花那么容易，因此本人设置最大迭代次数为 30000 次，在尽可能保证 MCMC 链完全收敛的情况下作出参数估计。

对于 Laplace 分布先验与正态分布先验，本人都进行了试验。且和之前类似，本人将 Laplace 分布先验与 sklearn 库的当中的 L_1 正则下的逻辑回归进行对比，而正态分布先验就对应地与 sklearn 库的当中的 L_2 正则下的逻辑回归进行对比。

先给出以 Laplace 分布作为先验的参数估计结果，将该结果与 sklearn 库当中的 L_1 正则逻辑回归结果比较，相关结果如下图14与15所示：

从图中结果可以看出，采用 Laplace 分布作为先验的 MCMC 算法的分类准确率稍逊于 sklearn 当中的 Lasso 逻辑回归算法。这一点主要是体现在了召回率上，即 MCMC 算法更容易将标签为 0 的样本误分为标签为 1 的样本。而从模型的参数估计方面，可以看到第一维和第五维的参数估计结果较为接近，主要的差异体现在中间三个维度的参数

```
预测准确率为:0.7000
预测查准率为:0.6667
预测召回率为:0.7368
预测f1-score为:0.7000
模型系数结果beta为 [ 1.00319243  2.2100314   1.02455745 -2.01142384 -2.04800168]
```

Figure 14: Laplace 分布先验 MCMC 算法作用于第五次作业数据集结果

```
预测准确率为:0.7250
预测查准率为:0.6667
预测召回率为:0.8421
预测f1-score为:0.7442
[[ 1.20252945  1.42441513  0.40335074 -1.10683269 -1.9855369 ]]
```

Figure 15: sklearnL_1 逻辑回归作用于第五次作业数据集结果

上，相比之下，采用 Laplace 分布作为先验的 MCMC 算法的参数估计没有 sklearn 当中的 Lasso 逻辑回归算法这么保守，其估计参数的绝对值会更大一些。

类似地，以正态分布作为先验进行试验，将试验结果与 sklearn 库当中的 L_2 正则逻辑回归结果比较，相关结果如下图16与17所示：

```
预测准确率为:0.7750
预测查准率为:0.7273
预测召回率为:0.8421
预测f1-score为:0.7805
模型系数结果beta为 [ 1.14342195  1.31200235  0.02890504 -0.55291776 -1.95797937]
```

Figure 16: 正态分布先验 MCMC 算法作用于第五次作业数据集结果

```
预测准确率为:0.7000
预测查准率为:0.6522
预测召回率为:0.7895
预测f1-score为:0.7143
[[ 1.18490769  1.2798258   0.47955201 -1.17427586 -1.82474141]]
```

Figure 17: sklearnL_2 逻辑回归作用于第五次作业数据集结果

从这边的结果可以明显看出，以正态分布为先验的 MCMC 算法的分类效果甚至要好于 sklearn 库当中的 L_2 正则下的逻辑回归分类效果。无论是分类准确率、查准率、召回率还是 f1-score，以正态分布为先验的 MCMC 算法都要更为领先一些；而从参数估计的结果来看，可以看到第一维、第二维和第五维的参数估计结果更为接近，主要的差异体现在中间的第三维与第四维上，相对地，以正态分布为先验的 MCMC 算法看上去

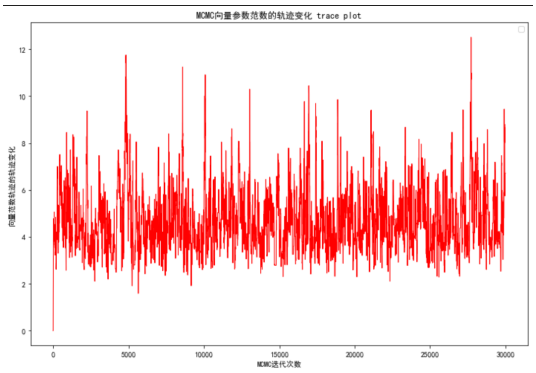
估计更为保守一些，尤其是第三维度的参数估计，十分接近于 0。从这里的测试集上分类结果看来，对于该分类问题，采用以正态分布为先验的 MCMC 算法是一个较好地选择，其效果甚至要好于官方的 sklearn 库的结果，这也变相说明了本人的 MCMC 算法也基本达到了细致平衡的稳定条件。

2.4.3 MCMC 的收敛性分析

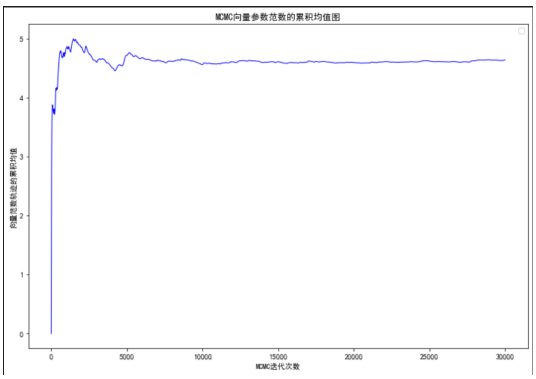
此前本人给出了两个数据集上的测试集分类效果以及算法的参数估计，但是事实上对于算法的收敛性并没有进行分析，因而算法的估计是否稳健，这一点是无法保证的。在这一部分，本人针对此前的两种不同先验分布下的 MCMC 算法，进行收敛性分析。

本人采用的方法为绘制样本路径图与遍历均值图，以此观察 MCMC 算法是否收敛。在算法实现的过程中，本人计算每次迭代得到的参数估计的二范数值，并进行记录，由每次迭代的二范数值绘制样本路径图与遍历均值图。在这里，本人都进行了 30000 次迭代，观察两图的效果。

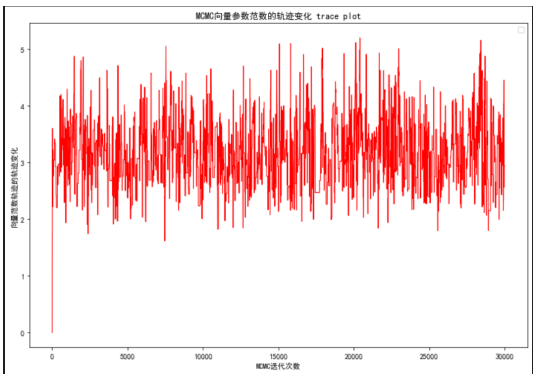
在鸢尾花数据集上，基于 Laplace 分布先验与正态分布先验的样本路径图与遍历均值图分别如下图所示：



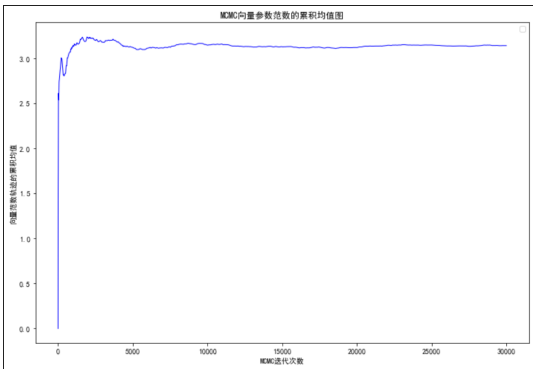
(a) Laplace 先验样本路径图



(b) Laplace 先验遍历均值图



(c) 正态先验样本路径图



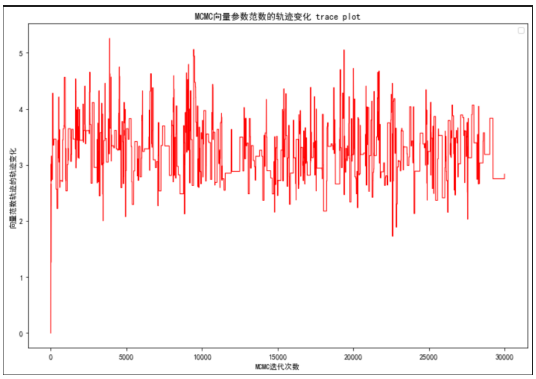
(d) 正态先验遍历均值图

Figure 18: 鸢尾花数据集不同先验下的样本路径图与遍历均值图

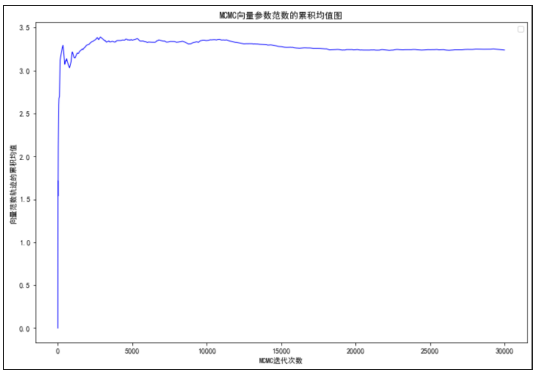
从图上结果可以看出，基本上当迭代次数达到 5000 次，累积的遍历均值基本趋于

稳定，而在迭代次数达到 10000 次时，累积均值不再发生大的波动。这说明，本人的 MCMC 算法最后对于鸢尾花数据集可以达到稳定，最后得到的参数估计也是稳健的。

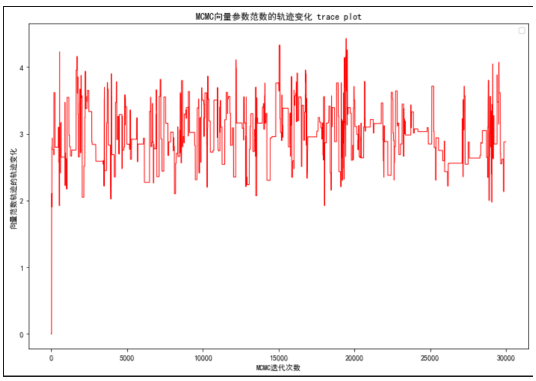
而在第五次作业的逻辑回归数据集上，本人也进行了同样的操作，得到的基于 Laplace 分布先验与正态分布先验的样本路径图与遍历均值图如下所示：



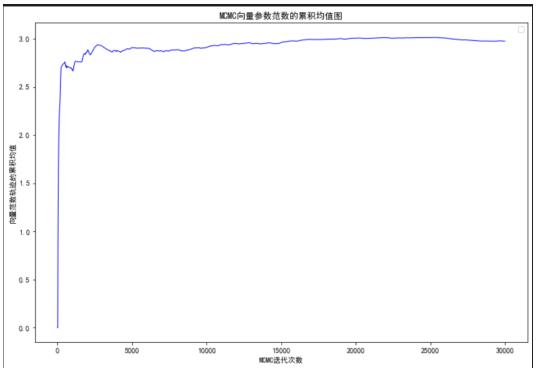
(a) Laplace 先验样本路径图



(b) Laplace 先验遍历均值图



(c) 正态先验样本路径图



(d) 正态先验遍历均值图

Figure 19: 第五次作业数据集不同先验下的样本路径图与遍历均值图

由于该数据集总体的区分度没有此前的鸢尾花数据集那么大，因此在迭代过程中的参数估计的二范数的稳健性不如此前的鸢尾花数据集。但是算法还是基本上可以在 15000 次迭代之后达到基本稳定。结合此前正态分布先验下的 MCMC 算法得到的参数估计在测试集上的分类效果甚至要好于 sklearn 库中的逻辑回归，有理由相信此处得到的参数估计是稳健的，采用 MCMC 算法是可靠的。

3 第三问

3.1 简要概述

第三问涉及一个真实场景——客户是否会放弃银行信用卡场景下的分类问题。本人首先基于数据集对预处理工作进行介绍，接着采用 Lasso 逻辑回归与 MCMC 算法对这一场景下的逻辑回归参数进行估计，然后在测试集上进行测试，报告测试集上的分类结果，最后对分类结果进行解释和分析。

3.2 数据预处理与特征筛选

在数据预处理方面，首先去掉数据当中的'ID' 一列，并重设数据索引，使得数据集从索引 0 开始。接着对于所有的标签，将其编码化。对于标签'Existing Customer'，本人将其重设为 0；而对于标签'Attrited Customer'，本人将其重设为 1。这是对于二分类问题的经典做法。

接下来着手处理数据的特征。数据的特征还是以数值的 float 类型居多，但是也存在着一些非数值型的变量，故需要着重对这些变量进行处理。

首先是性别。性别只有男女之分，对于这一二值变量，本人将男性以 0 进行编码，而女性以 1 进行编码。对于受教育程度，其从低到高分为未受教育到博士毕业共 7 个不同等级，包含了未知教育类型的'Unknown'。在这里，本人认为受教育程度的从高到低是存在顺序关系的变量，因此对这一特征进行 OrdinalEncoder 的顺序变量即可，无需采用 One-Hot 编码消除特征之间存在的顺序关系。因此对于从未受过教育到博士毕业的这 6 个等级，本人分别以 1-6 进行编码，而对于'Unknown'，本人直接设为 0。收入情况这一特征和受教育程度类似，同样是从高到低存在不同等级，本人认为为了凸显这一特征当中存在的顺序关系，同样采用 OrdinalEncoder 的顺序变量是较为稳妥的做法，无需采用 One-Hot 编码消除特征之间的顺序关系，因此本人对于‘收入小于 40000 美金’到‘收入大于 120000 美金’依次以 1-5 进行编码，而未知收入直接编码为 0。

而对于婚姻状况与信用卡类型这两个特征，个人认为其不同值之间不存在顺序关系。以婚姻状况举例，显然离婚与已婚以及单身之间不存在明显的数值上的顺序关系，因而使用 One-Hot 编码忽略掉数值层面的顺序关系是较为合理的做法。信用卡类型也同理。经过这一系列的编码操作，本人将所有的特征都变为了数值型，特征数扩增为了 25 个。接下来只需要继续对这一数值矩阵进行进一步处理即可。

个人认为 25 个特征还是过多，事实上没必要采用这么多的特征。故对特征进行筛选是必要的操作。对于分类问题而言，采用特征假设检验的方式对特征与标签之间的相关性进行检验是有效的过滤特征的操作。本人就采用这样的做法对变量进行筛选。

变量筛选的相关性检验主要有 3 种，分别为 F 检验、卡方检验和互信息检验。本人对这三种方法都进行了尝试。F 检验，又称 ANOVA，方差齐性检验，是用来捕捉每个特征与标签之间的线性关系的过滤方法。F 检验的本质是寻找两组数据之间的线性关

系，其原假设是“数据不存在显著的线性关系”，它返回 F 值和 p 值两个统计量，我们希望选取 p 值小于 0.05 或 0.01 的特征，这些特征与标签时显著线性相关的，而 p 值大于 0.05 或 0.01 的特征则被我们认为是和标签没有显著线性关系的特征，应该被删除。卡方检验的做法与 F 检验类似。

而互信息法是用来捕捉每个特征与标签之间的任意关系（包括线性和非线性关系）的过滤方法。互信息法的特性在于可以返回不只是线性关系的任意关系有关的变量，这是互信息法的强大之处。与 F 检验及卡方检验不同，互信息法不返回 p 值或 F 值类似的统计量，它返回“每个特征与目标之间的互信息量的估计”，这个估计量在 [0,1] 之间取值，为 0 则表示两个变量独立，为 1 则表示两个变量完全相关。

在这里，通过卡方检验、F 检验以及互信息法得到的过滤后的特征维度和特征名分别如下图所示：

```
[ 0 13 14 15 16 18 19 20 21 22 23 24]
Index(['Customer_Age', 'Months_on_book', 'Total_Relationship_Count',
       'Months_Inactive_12_mon', 'Contacts_Count_12_mon',
       'Total_Revolving_Bal', 'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1',
       'Total_Trans_Amt', 'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1',
       'Avg_Utilization_Ratio'],
      dtype='object')
```

Figure 20: F 检验特征筛选结果

```
[ 0 13 14 15 16 17 18 19 21 22 23 24]
Index(['Customer_Age', 'Months_on_book', 'Total_Relationship_Count',
       'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Credit_Limit',
       'Total_Revolving_Bal', 'Avg_Open_To_Buy', 'Total_Trans_Amt',
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'],
      dtype='object')
```

Figure 21: 卡方检验特征筛选结果

```
[ 0  4 12 14 15 16 17 18 19 20 21 22 23 24]
Index(['Customer_Age', 'Unknown', 'Platinum', 'Total_Relationship_Count',
       'Months_Inactive_12_mon', 'Contacts_Count_12_mon', 'Credit_Limit',
       'Total_Revolving_Bal', 'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1',
       'Total_Trans_Amt', 'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1',
       'Avg_Utilization_Ratio'],
      dtype='object')
```

Figure 22: 互信息检验特征筛选结果

从特征筛选结果看来，结果比较类似。值得一提的是，互信息检验方法在该数据集上的表现不是非常稳定。本人的设定是筛选出所有信息相关度大于 0.005 的特征，但是

互信息检验法很容易受到这一相关度阈值的影响，这一阈值设置得稍大一些，就会有很多特征被筛掉。相比之下，F 检验直接采用统计学中常用的 95% 置信度即可，筛选出检验 p 值小于 0.05 的特征即可，这一做法也比较方便。从结果上来看，F 检验筛选出的特征和卡方检验类似，基本上特征集中于持卡人的年龄、持卡人的消费习惯以及其他一些银行卡业务的参与程度和购买记录这些，这一筛选结果也相对比较合理。因此本人最后采用 F 检验的特征筛选结果，使用筛出的这 12 个特征进行逻辑回归模型的建立。

3.3 模型建立与结果比较

在这部分，本人将 Lasso 逻辑回归与 MCMC 方法比较的同时，将自己写的函数包和 sklearn 库的逻辑回归方法进行了比较，以选择出最佳的结果。

在进行试验之前，需要将筛选出的对应特征进行归一化。本人选定的归一化范围为 [-1,1]。这一做法主要是为了防止迭代过程中出现参数估计过大的问题（因为信用卡限额、平均开户购买力以及总账户转换金额这些特征的数值较大），从而导致参数估计结果受限于这些参数而过于离谱。

本人对自己写的 Lasso 逻辑回归方法类与 MCMC 方法类都进行了实例化（MCMC 方法 Laplace 先验和正态分布先验都进行实例化），采用的参数和之前第一问与第二问的参数几乎一致。

采用 Lasso 逻辑回归得到的参数估计结果与测试集上分类指标如下图23所示：

```
预测准确率为:0.8500
预测查准率为:0.8000
预测召回率为:0.3810
预测f1-score为:0.5161
beta为 [ 0.          0.         -0.75364091  1.40509936  0.69904226 -0.77588457
  0.51087514  0.          1.73278387 -3.36044005  0.77450009  0.19835793]
```

Figure 23: Lasso 逻辑回归结果

从参数估计结果来看，特征‘消费者年龄’、‘信用卡使用月数’和‘交易数量变动’是 Lasso 逻辑回归当中被认为不重要的变量，故参数直接估计为 0。从估计结果上看来，估计的查准率较高而召回率较高，即 Lasso 逻辑回归比较容易将真实标签为 1 的样本判别为 0。

采用以 Laplace 分布作为先验的 MCMC 算法，本人设置迭代总次数为 10000 次，以零向量作为先验的位置参数，以全 1 向量作为先验的尺度参数，得到的参数估计结果与测试集上分类指标如下图24所示：

从结果上来看，模型的参数估计结果显示各个特征均有一定的贡献，测试集上预测的各项指标都要高于此前 Lasso 逻辑回归的结果。且从参数估计的结果看来，各个特征的参数估计结果没有此前那么保守，尤其是‘总交易量’、‘总交易单量’这两个特征，在逻辑回归的最后判别当中起到了较为关键的作用。总体上看来，采用以 Laplace 分布

```
预测准确率为:0.8900
预测查准率为:0.8571
预测召回率为:0.5714
预测f1-score为:0.6857
模型系数结果beta为 [ 0.58320774 -0.47376855 -1.60839047  1.46712068  1.8304139  -1.72916858
  0.2889358   0.39297838  4.24754505 -7.02536244 -0.72291608  1.17394652]
```

Figure 24: Laplace 分布作为先验的 MCMC 算法结果

作为先验的 MCMC 算法，基本上可以获得一个较为稳健的估计，且预测的效果要好于 Lasso 逻辑回归。

采用以正态分布作为先验的 MCMC 算法，本人设置的各项函数参数与 Laplace 分布作为先验的情况完全一致，同样迭代 10000 次，得到的参数估计结果与测试集上分类指标如下图25所示：

```
预测准确率为:0.8800
预测查准率为:0.8462
预测召回率为:0.5238
预测f1-score为:0.6471
模型系数结果beta为 [ 0.95318444 -0.08937795 -0.92381952  1.73699133  1.47181748 -1.50250845
  0.69479624 -0.70708536  3.46643698 -5.15280412 -0.26861872  0.77135319]
```

Figure 25: 正态分布作为先验的 MCMC 算法结果

总体而言，采用正态分布作为先验的 MCMC 算法在测试集上的预测效果稍逊于以 Laplace 分布作为先验的 MCMC 算法。相比之下，参数估计也相对保守一些。

本人同样采用 sklearn 库当中官方的逻辑回归算法，分别设置惩罚项为 L_1 正则与 L_2 正则，将结果与两种不同先验情况下的 MCMC 算法进行比较，得到的结果分别如下图26与27所示：

```
预测准确率为:0.8600
预测查准率为:0.7917
预测召回率为:0.4524
预测f1-score为:0.5758
[[ 6.38399291e-03  2.32286277e-02 -4.90578128e-01  7.81019308e-01
   2.98314751e-01 -9.50225344e-04  1.58676496e-05 -5.62525040e-02
   3.07140020e-04 -1.03909036e-01 -2.94175241e+00 -3.91636295e-02]]
```

Figure 26: sklearn 库 L_1 正则逻辑回归结果

从总体结果上来看，sklearn 库当中的逻辑回归算法效果不如两种不同先验情形下的 MCMC 算法。算法预测的召回率较低，这导致算法容易在预测时将所有样本有偏向地预测为一类，这连带着导致算法的 f1-score 值也比较低，说明测试集上的算法预测效

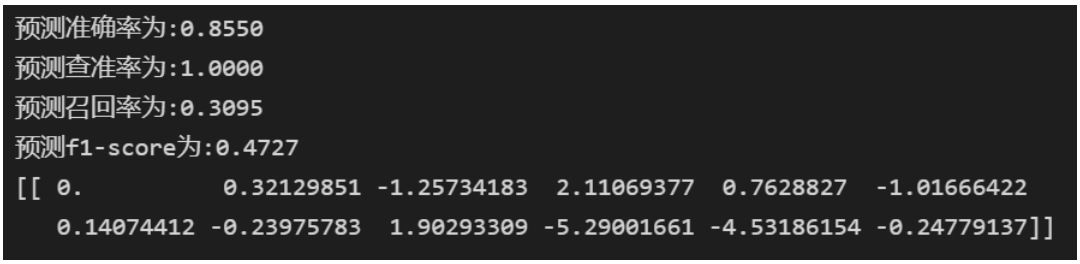


Figure 27: sklearn 库 L_2 正则逻辑回归结果

果并不好。而 L_2 惩罚下的逻辑回归查准率为 100% 但召回率却很低，这说明该算法将测试集上样本基本上都归位了一类，并没有很好地找出要放弃信用卡业务的客户，这反映了这一算法的效果非常不好。

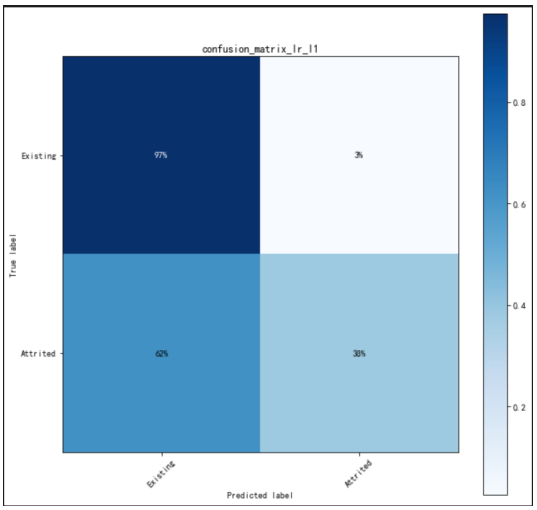
将所有的结果进行汇总，可以得到如下表格4：

Table 4: 各算法作用于信用卡数据集的结果汇总

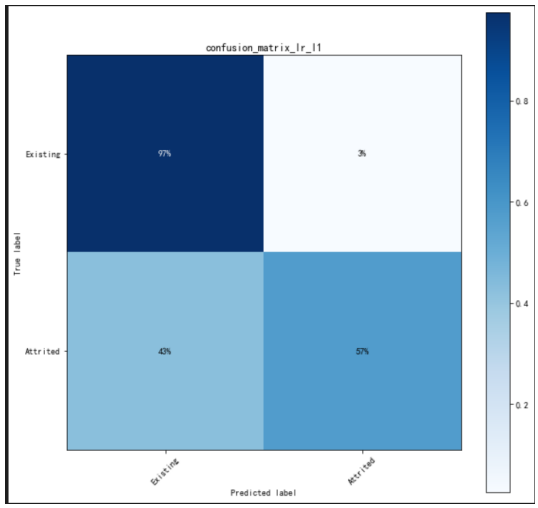
算法	预测准确率	查准率	召回率	f1 得分
Lasso 逻辑回归	85.00%	80.00%	38.10%	0.5161
Laplace 先验 MCMC	89.00%	85.71%	57.14%	0.6857
正态先验 MCMC	88.00%	84.62%	52.38%	0.6471
sklearnL_1 逻辑回归	86.00%	79.17%	45.24%	0.5758
sklearnL_2 逻辑回归	85.50%	100.00%	30.95%	0.4727

与此同时给出所有算法的混淆矩阵，如下图所示：

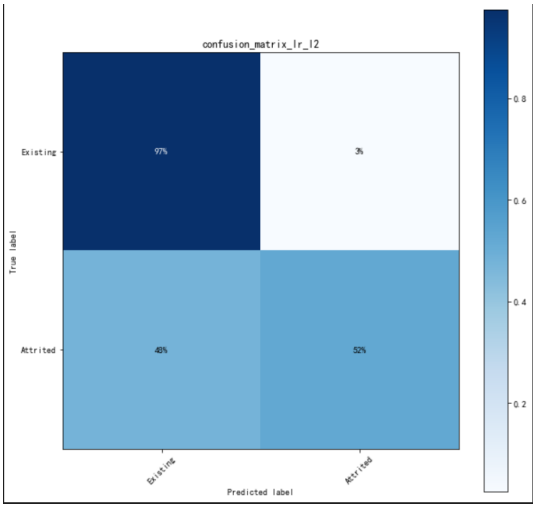
从汇总结果和混淆矩阵看来，MCMC 算法是对于该分类问题较好的参数估计方法，无论是采用 Laplace 先验还是正态先验，其分类的召回率都相对较高，可以在测试集上较好地找出要退卡的客户，相对地 f1 得分也更高。Lasso 逻辑回归方法的预测效果可能并不是特别好，但是该方法的主要效果体现在样本的特征选择上。Lasso 逻辑回归方法认为在该问题当中，‘消费者年龄’，‘信用卡使用时长’和‘交易数量变动量’是不重要的变量，并不会对预测的结果造成实质性影响。因而这一方法可以协助银行经理将工作重心投入到信用卡业务的其他特征，例如‘交易总量’、‘交易总单量’这些特征的研究，这也可以一定程度上帮助信用卡业务经理找到对于信用卡业务而言重要的指标，从而协助其挽留更多的信用卡客户。



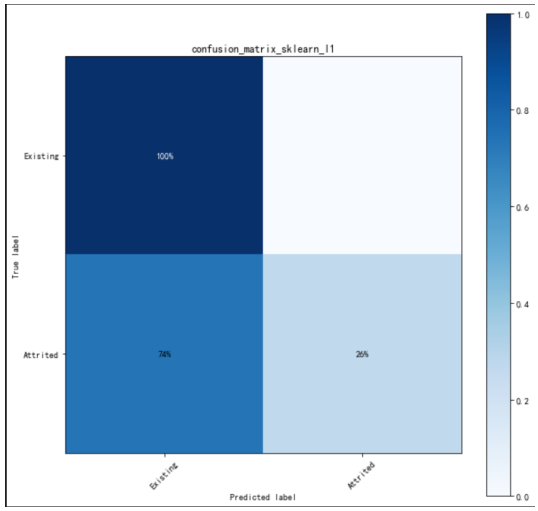
(a) Lasso 逻辑回归混淆矩阵



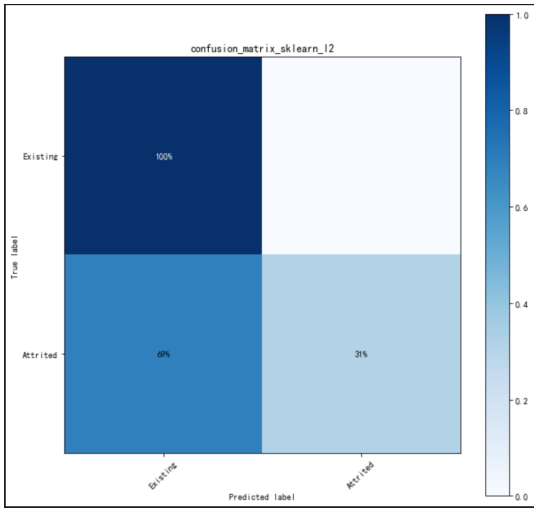
(b) Laplace 先验 MCMC 混淆矩阵



(c) 正态先验 MCMC 混淆矩阵



(d) sklearnL_1 逻辑回归混淆矩阵



(e) sklearnL_2 逻辑回归混淆矩阵

Figure 28: 第五次作业数据集不同先验下的样本路径图与遍历均值图

4 报告总结

该报告着重研究了逻辑回归的参数估计问题，从两个截然不同的视角看待这一问题，并且给出了完全不同的解决方案。

在 Lasso 逻辑回归部分，着重从经典统计的视角看待这一问题，在原有的损失函数的基础上加上了 L_1 正则项惩罚，这在一定程度上起到了特征选择的效果。在损失函数的求解方面，采用的是经典的梯度下降算法与坐标下降算法。在算法实现上，本人通过实例化过程中新设置的 batch_n 样本批量大小参数，将梯度下降算法与小批量随机梯度下降算法结合，以此达到参数估计准确性与算法速度二者之间的平衡。而在坐标下降算法的实现方面，本人同样设置了两种不同的坐标维度选取方式（按顺序循环选取与全随机选取），以此完成了经典的 L_1 惩罚下的逻辑回归参数估计问题，并且使用鸢尾花数据集和第五次作业的逻辑回归数据集进行了试验，测试了算法效果。

而在 MCMC 算法部分，着重从贝叶斯统计的视角看待这一问题，给参数提前设置先验，力求在具有先验分布的基础上，得到参数的后验估计，再通过后验参数估计建立模型。由于后验分布的形式较为复杂，通过经典的梯度下降算法无法进行求解，因而考虑采用 MCMC 算法进行模拟，得到参数估计。本人采用的是 MCMC 算法当中经典的随机游走算法，与此同时，使用了两种不同的先验——Laplace 分布先验与正态分布先验，分别对应逻辑回归中的 L_1 惩罚与 L_2 惩罚，对算法进行多次迭代后求得参数的后验估计。在这里，本人还是通过鸢尾花数据集与第五次作业的数据集，同时使用 sklearn 库当中的逻辑回归算法与之比较，测试了 MCMC 算法的效果。

最后，着重将这两个实现的算法作用于一个真实场景——银行信用卡客户是否会放弃银行卡业务这一问题，采用这两种截然不同的算法给出参数估计，并且将参数结果作用于测试集建模，比较了模型效果。最后发现，对于这一二分类问题，采用以 Laplace 分布作为先验的 MCMC 算法是求解该逻辑回归二分类问题参数估计的较好的选择，因为这一方法在测试集上具有较高的召回率和 f1 得分，分类效果相对较好一些。

逻辑回归算法作为机器学习当中的经典算法，具有形式简单，易于计算且应用场景广泛的特点，值得进一步深入的研究与改进。