

哈尔滨工业大学计算机科学与技术学院

## 实验报告

课程名称： 机器学习

课程类型： 专业限选

实验题目： 逻辑回归

学号： 7203610121

姓名： 刘天瑞

## 一、实验目的

理解逻辑回归模型，掌握逻辑回归模型的参数估计算法。

## 二、实验要求及实验环境

### 1.实验要求

实现两种损失函数的参数估计（1，无惩罚项；2.加入对参数的惩罚），可以采用梯度下降、共轭梯度或者牛顿法等。

实验验证：

1. 可以手工生成两个分别类别数据（可以用高斯分布），验证你的算法。考察类条件分布不满足朴素 Bayes 假设，会得到什么样的结果。
2. 逻辑回归有广泛的用处，例如广告预测。可以到 UCI 网站上，找一实际数据加以测试。

### 2.实验环境

Visual Studio 2022 ； Python 3.10.7 ； numpy 库和 matplotlib 库里的 pyplot 以及 mplot3d

## 三、设计思想（本程序中的用到的主要算法及数据结构）

### 1.主要算法及其原理

分类器做分类问题实质上是预测一个已知样本的位置标签，即  $P(Y=1|X)$ ，根据该概率值来判断样本的类别属性。按照朴素 Bayes 的方法，可以用 Bayes 概率公式将其转化为类条件概率(似然)和类概率的乘积。本实验是直接求该概率。以下为 Bayes 概率公式的推导：

考虑最基本的 0/1 二分类模型  $f: X \rightarrow Y$ ，其中  $X$  为实数向量  $X = \langle X_1, \dots, X_n \rangle$ ， $Y \in \{0, 1\}$ ，且根据朴素 Bayes 的假设，有所有  $X_i$  在给定  $Y$  的情况下条件独立。另外，有  $P(X_i|Y=y_k) \sim N(\mu_{ik}, \sigma_i)$ ， $P(Y) \sim B(\pi)$  成立。则基于以上这些假设，并将  $\pi = P(Y=1)$ ， $1 - \pi = P(Y=0)$  代入，用朴素 Bayes 将其展开，则有推导如下所示：

$$\begin{aligned}
P(Y=1|X) &= \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1) + P(Y=0)P(X|Y=0)} \\
&= \frac{1}{1 + \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}} \\
&= \frac{1}{1 + \exp(\ln \frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)})} \\
&= \frac{1}{1 + \exp(\ln \frac{1-\pi}{\pi} + \sum_i \ln \frac{P(X_i|Y=0)}{P(X_i|Y=1)})}
\end{aligned}$$

因为各个维度的条件概率分布符合高斯分布,所以可以将各个维度的高斯分布函数  $P(X_i|Y=y_k) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left(-\frac{(x - \mu_{ik})^2}{2\sigma_i^2}\right)$  代入可得:

$$\begin{aligned}
&= \frac{1}{1 + \exp(\ln(\frac{1-\pi}{\pi}) + \sum_i \ln(P(X_i|Y=0) - P(X_i|Y=1)))} \\
&= \frac{1}{1 + \exp(\ln(\frac{1-\pi}{\pi}) + \sum_i (-\ln\sigma_i \sqrt{2\pi} - \frac{(x_i - \mu_{i0})^2}{2\sigma_i^2} - (-\ln\sigma_i \sqrt{2\pi} - \frac{(x_i - \mu_{i1})^2}{2\sigma_i^2})))} \\
&= \frac{1}{1 + \exp(\ln(\frac{1-\pi}{\pi}) + \sum_i (\frac{(x_i - \mu_{i1})^2}{2\sigma_i^2} - \frac{(x_i - \mu_{i0})^2}{2\sigma_i^2}))} \\
&= \frac{1}{1 + \exp(\ln(\frac{1-\pi}{\pi}) + \sum_i (\frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2} x_i + \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}))}
\end{aligned}$$

然后再令  $w_0 = \ln \frac{1-\pi}{\pi} + \sum_i \frac{\mu_{i1}^2 - \mu_{i0}^2}{2\sigma_i^2}$ ,  $w_i = \frac{\mu_{i0} - \mu_{i1}}{\sigma_i^2}$ ,  $X = \begin{bmatrix} 1 \\ X_1 \\ \vdots \\ X_n \end{bmatrix}$ ,  $w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$ , 则有:

$$P(Y=1|X) = \frac{1}{1 + \exp(w_0 + \sum_{i=1}^n w_i X_i)}$$

再将其转化为矩阵形式:

$$P(Y=1|X) = \frac{1}{1 + \exp(w^T X)}$$

将形如  $y = \frac{1}{1 + e^{-z}}$  的函数称为 *sigmoid* 函数的基本形式,它具有将实值映射到

0 到 1 之间的某个值的特性。观察其函数图像可知,  $\text{sigmoid}(0) = 0.5$ , 且从  $x = 0$  出发, 无论是向右或向左,  $y$  值都以很快的速度向 1 和 0 逼近。因此可以看作这个函数将线性模型预测的连续值映射到 0 到 1 的概率上, 从而得到 0/1 标签离散值。

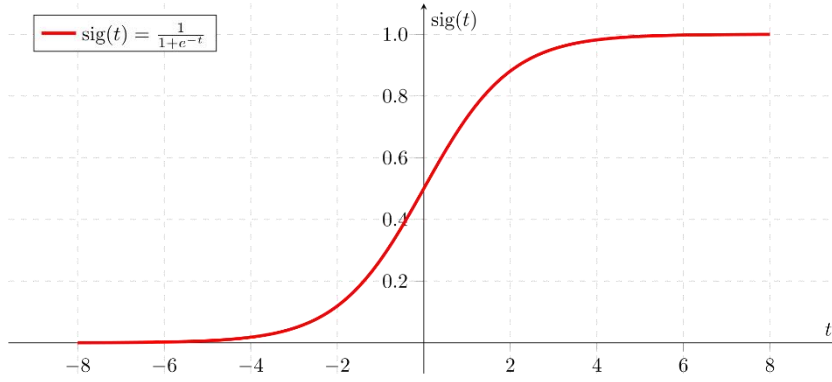


图 1 *sigmoid* 函数图像

而对于逻辑回归问题，参数即为 $w$ 。设损失函数为 $l(w)$ ，要计算 $l(w)$ ，因为需要计算 $P(<X, Y>|w)$ ，所以通过极大似然估计（MLE）难以解决，但是可以将其转化为计算极大似然条件估计（MCLE），此时只需计算 $P(X|Y, w)$ ，可得：

$$P(Y = 0|X, W) = \frac{1}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

$$P(Y = 1|X, W) = \frac{\exp(w_0 + \sum_i w_i X_i)}{1 + \exp(w_0 + \sum_i w_i X_i)}$$

为下述推导方便且不失一般性，可将式子转化为：

$$\begin{aligned} l(W) &= \sum_l Y^l \ln P(Y^l = 1|X^l, W) + (1 - Y^l) \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l \ln \frac{P(Y^l = 1|X^l, W)}{P(Y^l = 0|X^l, W)} + \ln P(Y^l = 0|X^l, W) \\ &= \sum_l Y^l (w_0 + \sum_i w_i X_i^l) - \ln(1 + \exp(w_0 + \sum_i w_i X_i^l)) \end{aligned}$$

其中 $n$ 为样本维度，在二维情况下为2，之后标记为 $d$ 。 $l$ 代表当前样本点的组数。共有 $\{<X_1^1, X_2^1, \dots, X_d^1>, <X_1^2, X_2^2, \dots, X_d^2>, \dots, <X_1^n, X_2^n, \dots, X_d^n>\}$ 组样本点，这其中 $X_d^n$ 代表第 $n$ 组样本点的第 $d$ 个坐标。

用梯度下降法求得 $w = \operatorname{argmax}_w l(w)$ ，习惯将 $-l(w)$ 作为损失函数，使其尽量保持和梯度下降算法的一致性，等价于求其最小值：

$$\begin{aligned} \frac{\partial l(w)}{\partial w_i} &= \sum_l X_i^l (Y^l - \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i^l))}) \\ w_i &\leftarrow w_i + \eta \sum_l X_i^l (Y^l - \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i^l))}) \end{aligned}$$

控制过拟合问题需要加入正则项。在逻辑回归分类问题中，决策面是线性的，有无数个线性决策面可以有效地区分出两类数据，而要求所选取的决策面，可以使“最短距离”（两类中距离线性决策面最近的距离之和）最大化，以达到最好的分类效果。

记线性决策面方程为  $w^T x + b = 0$ ，设常数  $c$ ，有  $(w^T x_i + b)y_i > c$ 。其中，对于  $\forall x_i \in$  第一类， $y_i = 1$ ， $\forall x_i \in$  第二类， $y_i = -1$ 。

其中因为  $w^T x$  为点到线性决策面的法向量的投影，所以可以记  $x_i^*, y_j^*$  分别为类别内距离决策面的最近点，记  $m$  为  $x_i^*, y_j^*$  到单位长的  $\vec{w}$  的投影值的差值，则有：

$$m = \frac{w^T}{\|w\|} (x_i^* - x_j^*) = \frac{2c}{\|w\|}$$

最后优化问题转换成在约束条件  $y_i(w^T x_i + b) \geq 1, \forall i$  下，求  $\min w^T w$ 。所以加上正则项的梯度下降为：

$$w_i \leftarrow w_i - \eta \lambda w_i + \eta \sum_l X_i^l (Y^l - \frac{1}{1 + \exp(-(w_0 + \sum_i w_i X_i^l))})$$

## 2.实验具体操作步骤

### 1.手工生成数据测试

若要生成类条件分布满足朴素 Bayes 假设的数据，那么就对每一个类别的每一个维度都用一个独立的高斯分布生成。若要生成类条件分布不满足朴素 Bayes 假设的数据，那么就对每一个类别的两个维度用一个二维高斯分布生成。

值得一提的是，由于高斯分布具有的特性，多维高斯分布不相关可以推出独立性，因此，可以用二维高斯分布生成数据，如果是满足朴素 Bayes 假设的，那么协方差矩阵的非对角线元素均为 0，如果是不满足朴素 Bayes 假设的，那么协方差矩阵的非对角线元素不为 0（协方差矩阵应该是对称阵）。

### 2.梯度下降法进行参数估计

按照前述算法设计梯度下降法的具体实现，通过三个函数构成：其中 `gradient_descent()` 函数是梯度下降法的主要函数，其通过调节参数可以实现带惩罚项与不带惩罚项的迭代功能。而 `gradient()` 函数是按照迭代算法来计算梯度。`loss()` 函数则计算损失函数。

以下是梯度下降算法部分的核心函数具体代码：

```
63 def gradient(X, train_y, w):#计算梯度
64     grad = np.zeros((1, X.shape[0]))
65     for i in range(0, X.shape[1]):
66         wXT = w @ X[:, i]
67         grad += X[:, i].T * (train_y[i] - sigmoid(wXT))
68     return grad
```

图 2 gradient()函数运行代码

```
70 def loss(X, train_y, w):#损失函数: 计算优化式子, 归一化
71     loss = 0
72     size = float(train_y.shape[0])
73     for i in range(0, X.shape[1]):
74         wXT = w @ X[:, i]
75         loss += (train_y[i] * wXT / size - np.log(1 + np.exp(wXT)) / size)
76     return loss
```

图 3 loss()函数运行代码

```
77 def gradient_descent(train_x, train_y):#梯度下降算法主函数部分
78     eta_temp = eta
79     w = np.ones((1, train_x.shape[1] + 1))#(1,3), 第一列全部置为1
80     X = generate_X(train_x)
81     grad = gradient(X, train_y, w)
82     loss_list = []
83     loss_0 = 0
84     loss_1 = loss(X, train_y, w)
85     count = 0
86
87     while count < epoch:#设定误差值, 当小于该误差值时则停止迭代
88         count += 1
89         w = w - eta_temp * lamda * w + eta_temp * grad
90         loss_0 = loss_1
91         loss_1 = loss(X, train_y, w)
92
93         if(np.abs(loss_1) > np.abs(loss_0)):#loss不降反增, 则学习率减半, 防止迭代过程中步幅过大
94             eta_temp *= 0.5
95         grad = gradient(X, train_y, w)
96         loss_list.append(np.abs(loss_0))
97         print(count)
98
99     return w, count, loss_list
```

图 4 gradient\_descent()函数运行代码

### 3.对 UCI 数据集进行分类实验并做可视化展示

选取 UCI 数据集网站里的 **Skin Segmentation Data Set**, 其中包含有超过 200,000 个三维分类指标与对应的标签, 接着将建立好的模型应用于这组数据进行分类实验: 其中抽取 80%的数据作为训练集 (train set), 20%的数据作为测试集 (test set)。具体实现也通过三个函数构成: `skin_read()`函数读入数据集, `skin_plot_show()`函数对结果进行可视化展示, `skin_show()`函数则作为主函数调用。

由于样本数量过大导致难以进行训练, 所以不妨按照步长为 20 取样本进行训练, 训练详细结果在下一章节。

## 四、实验结果与分析

### 1.带入正则项影响

设置生成的数据方差为 0.3, 两类的均值分别为[-0.7, -0.3]与[0.7, 0.3], 学习率为 0.01, 超参数 $\lambda$ 为 0.001, 训练次数为 10。设置分布标签 `naiveFlag = True`, 即满足 Bayes 假设。

(i)不带正则项, 小样本:

设置正则项  $\lambda = 0$ ，样本点个数为 100，训练次数为 10，计算分类准确率为 97%，结果如下图所示：

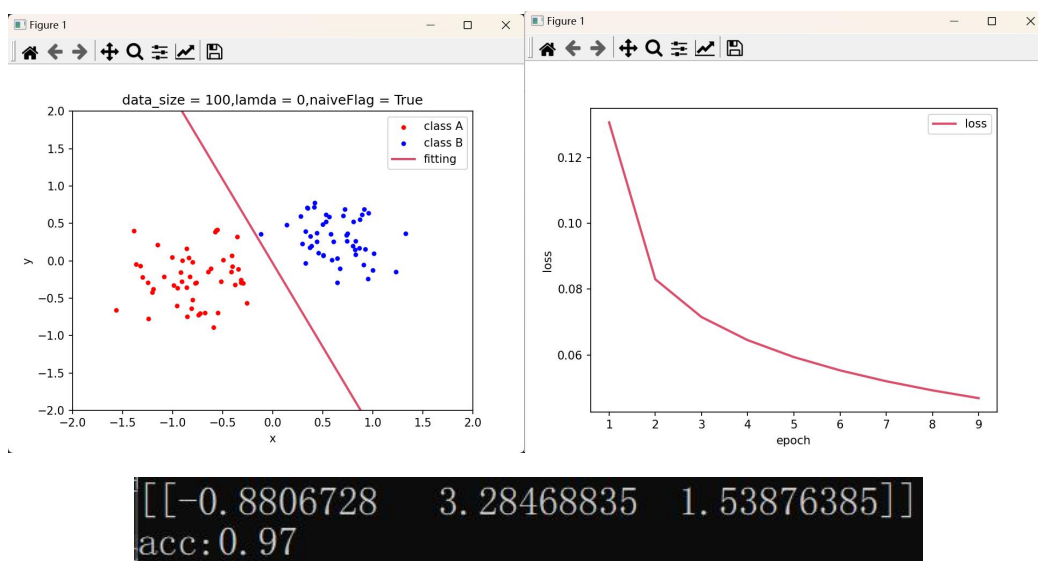


图 5 小样本不带正则项分类结果

(ii)带正则项，小样本：

设置正则项  $\lambda = 0.001$ ，样本点个数为 100，训练次数为 10，计算分类准确率为 98%，结果如下图所示：

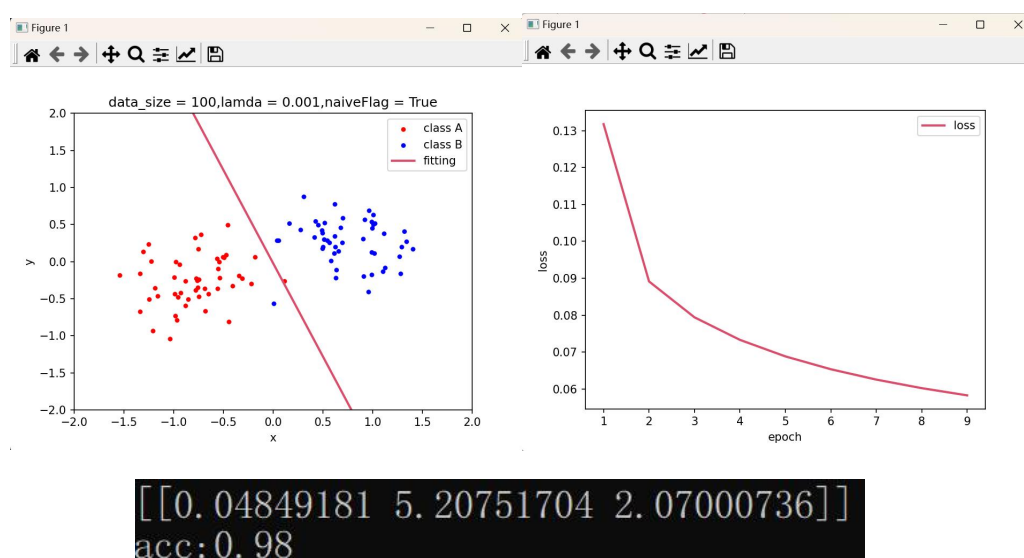


图 6 小样本带正则项分类结果

(iii)不带正则项，大样本：

设置正则项  $\lambda = 0$ ，样本点个数为 1000，训练次数为 5000，计算分类准确率为 99.1%，结果如下图所示：

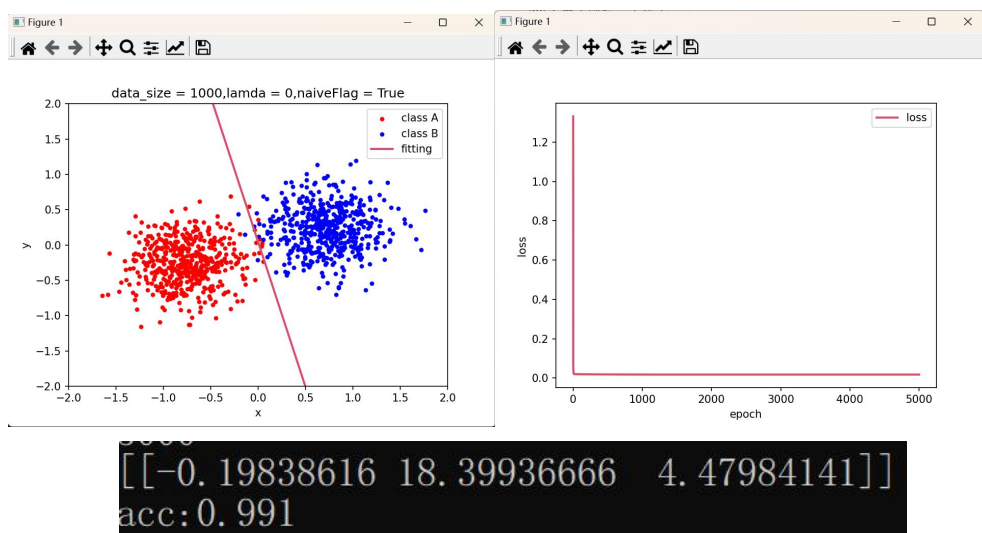


图 7 大样本不带正则项分类结果

(iv)带正则项，大样本：

设置正则项  $\lambda = 0.001$ ，样本点个数为 1000，训练次数为 5000，计算分类准确率为 99.7%，结果如下图所示：

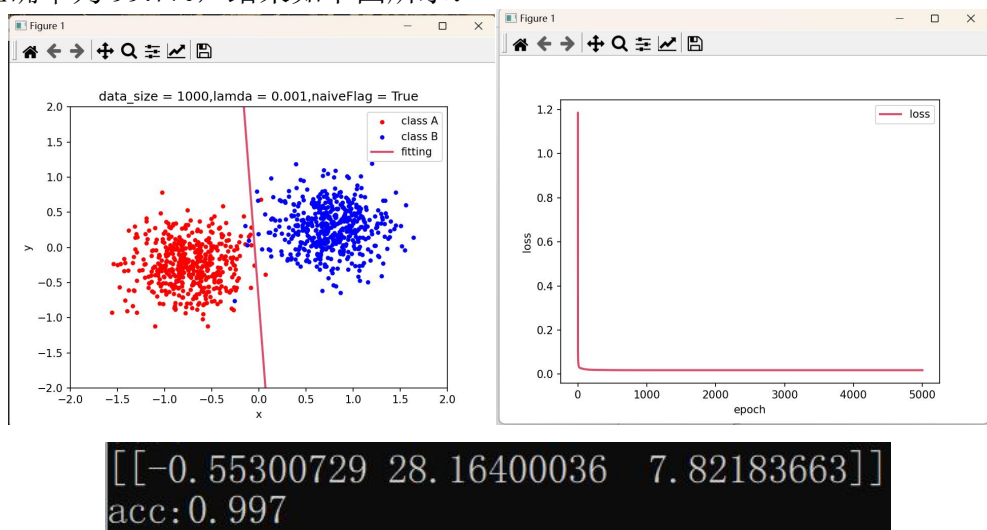


图 8 大样本带正则项分类结果

可以发现当数据量比较大的时候由于分类平面只是直线，因此正则项并没有起较大作用，这一点也是比较好解释的，由于使用的是线性分类器，而且次数仅为 1，学习能力并不强，因此在数据集上表现出过拟合的可能性较小。而可以发现测试集上的结果都比训练集略差，这也是符合一般认识的。

综上所述，对于生成测试数据，控制变量研究正则项的影响，可以得出如下结论：

- 1.在大样本和小样本的情况下，加上正则项计算出的 loss 值都比不加正则项要低，所以加上正则项可以得到更为准确的结果；
- 2.在小样本的情况下，不加正则项容易产生过拟合的问题，从而使得准确率较低，加上正则项可以有效缓解这个问题；
- 3.逻辑回归可以很好地解决线性分类问题，而且收敛速度较快，在生成测试



数据下往往能很快便使损失降低到一个可以接受的值，从而迭代得到结果更早；

4.在样本数量较多的情况下，加不加惩罚项都能得到比较好的分类结果，即在数据量较大时，正则项对结果的影响不大。

## 2.是否满足朴素 Bayes 假设条件的影响

设置生成的数据方差为 0.1，两类的均值分别为 $[-0.75, -0.25]$ 与 $[0.75, 0.25]$ ，学习率为 0.01,超参数 $\lambda$ 为 0.001，样本数为 1000，训练次数为 5000。若不满足朴素 Bayes 假设，则协方差矩阵的值为 0.08。

(i)满足朴素 Bayes 假设

设置分布标签 `naiveFlag = True`，计算分类准确率为 99.5%，训练结果如下图所示：

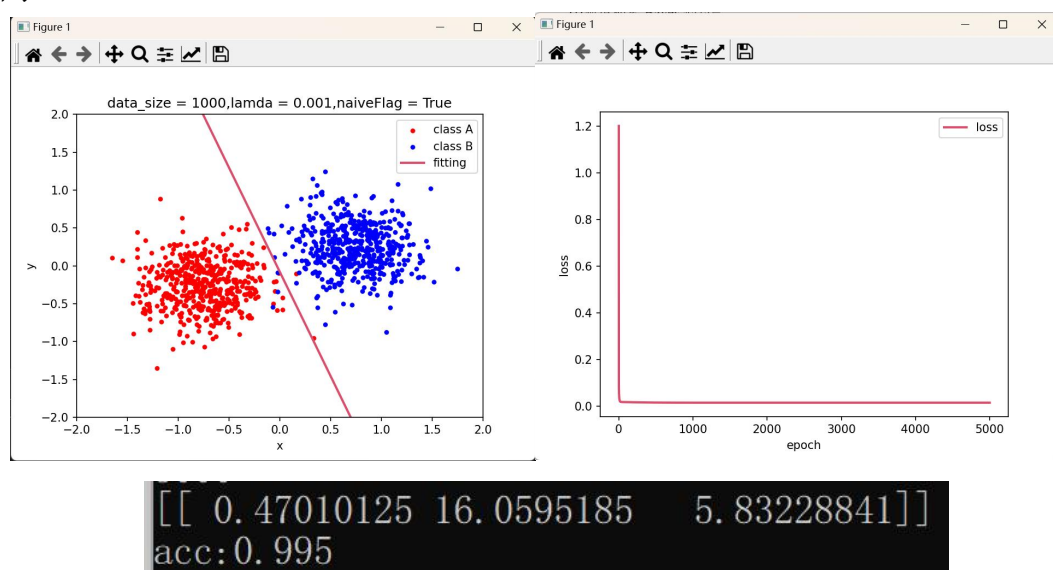
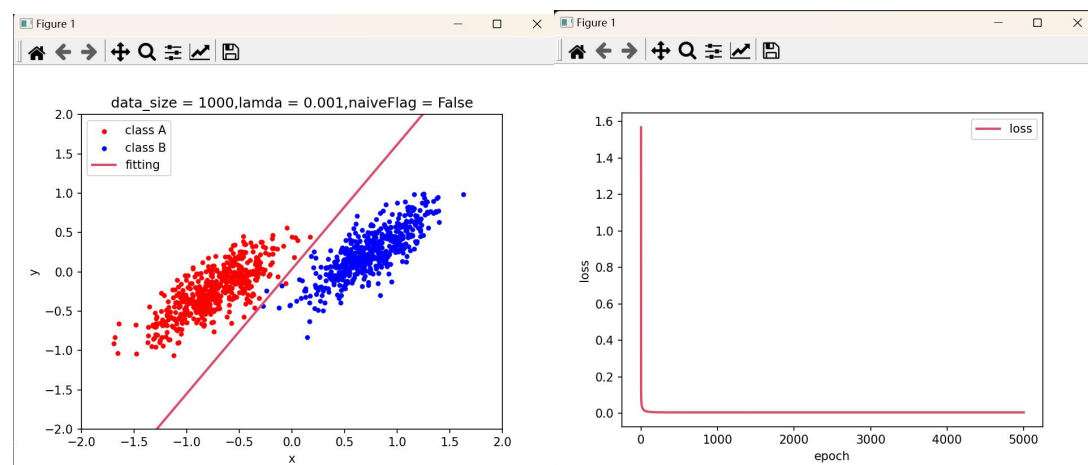


图 9 满足朴素 Bayes 假设的分类结果

(ii)不满足朴素 Bayes 假设

设置分布标签 `naiveFlag = False`，计算分类准确率为 99.8%，训练结果如下图所示：



```
[[ 0.57574513  26.57373153 -16.7728544 ]]  
acc:0.998
```

图 10 不满足朴素 Bayes 假设的分类结果

与符合朴素 Bayes 的结果相比可以发现,当分类数据不符合朴素 Bayes 的条件时,即类条件独立时分类的结果明显下降,但同时可以发现虽然精度下降却仍然保持在一个较高的水平,即若类条件之间的协方差并不是很大的时候,将其近似为类条件独立进行分类仍然可以得到较好结果。


综上所述,对于生成测试数据,控制变量研究是否满足朴素 Bayes 假设条件的影 响,可以得出如下结论:

1.可以明显发现不满足朴素 Bayes 假设时,数据点呈现出细长条形状,这表明数据的 2 个维度之间存在线性相关关系,符合预期。

2.虽然数据分布有所不同,但设计出的逻辑回归模型仍然可以很好地对其进行线性分类,但收敛速度相较于满足朴素 Bayes 假设时有所下降。


### 3.采用 UCI 数据集

选取 UCI 数据集网站的 Skin Segmentation Data Set 进行分类测试。



**UCI**

[About](#) | 
 [Contact Policy](#) | 
 [Privacy Policy](#) | 
 [Data Set Contact](#)



## Machine Learning Repository

Center for Machine Learning and Intelligent Systems

---

Check out the beta version of the new UCI Machine Learning Repository we are currently testing! Contact us if you have any issues, questions, or concerns. Click here to try out the new site.

## Skin Segmentation Data Set

Download [Data Folder](#) [Data Set Description](#)

Abstract: This Skin Segmentation dataset is constructed from the S, R, G color space. Data sets for each gender are generated using skin lesions from face images of diversity of age, gender, and race people.

Data Set Characteristics:	Instances	Number of Instances:	Size(s)	Area:	Complete
Attribute Characteristics:	Real	Number of Attributes:	4	Date Generated:	20/02/17
Associated Tasks:	Classification	Missing Values %	N/A	Number of Web Hits:	24/04/1

**Source:**

Kojan Bhatt, Aniruddh Chakr, [kj.bhatt@gmail.com](mailto:kj.bhatt@gmail.com), IIT Delhi

**Data Set Information:**

The skin dataset is collected by randomly sampling 5 GLS values from face images of various age groups (young, middle, and old), race groups (white, black, and asian), and genders obtained from FERECIT database and PIML databases. Total learning sample size is 240007, out of which 10000 is test set and 130000 is training samples. Color (RGB) Image Extension: [jpeg](#), [tiff](#). Face Extension from Procedure (Age, Gender, Race): [dataset](#), [train](#).

**Attribute Information:**

The dataset has 6 dimensions 210007 \* 6 where first three columns are B,G,R (x1,x2, and x3 features) value and fourth column is (z) of the class label (decision variable).

**Relevant Papers:**

1. Kojan B Bhatt, Arunima Ghosh, Aniruddh Chakr, Ranvira Chaudhary, "An Efficient skin lesion segmentation using low complexity fuzzy decision tree model", ICSPR/ICCVIS 2016, Dec 16-18, Hyderabad, India, pp. 1-4.
2. Aniruddh Chakr, Sourav Sharma, Deep Datta, Shubham Kulkarni, Pratik, "Multi-Scale Colour Map-based...", In Proc. of International Symposium on Visual Computing (VSVC 2016), Nov 20-24, Dec 1C, Las Vegas, Nevada, USA, Lecture Notes in Computer Science, Vol. 5076, pp. 726-735.

图 11 CI 数据集网站的 Skin Segmentation Data Set

设置训练参数为：学习率为 0.1，超参数 $\lambda$ 为 0.001，训练次数为 5000。计算分类准确率为 90.22%，训练结果如下图所示：

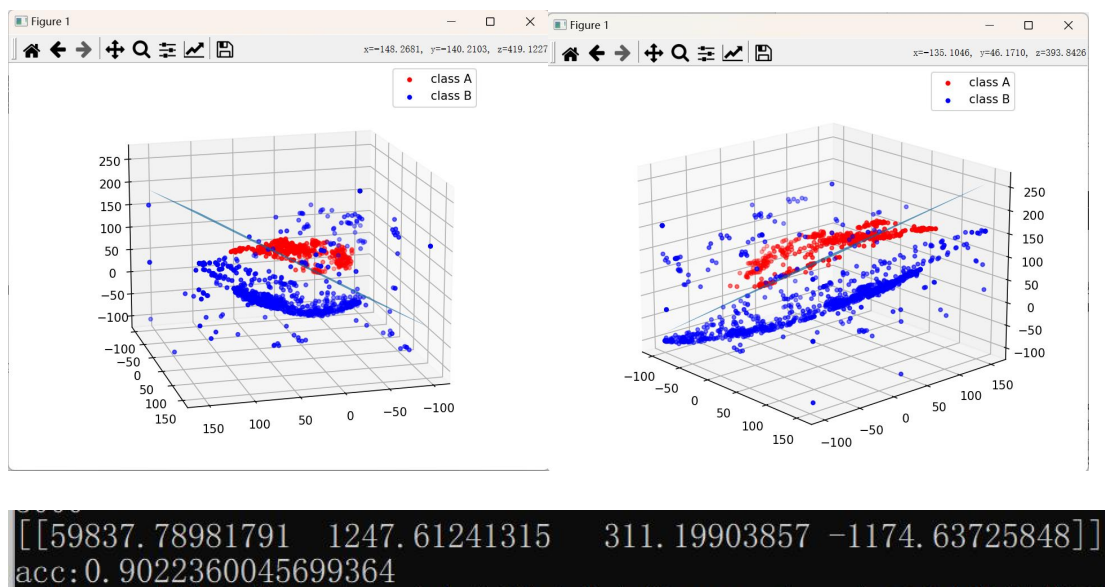


图 12 采用 UCI 数据集的三维分类结果

可以看出在经过足够多次迭代之后，训练出的模型在测试集(test set)上得到了 90%左右的分类准确率，说明在该问题上使用逻辑回归学习的得到的分化器性能和泛化能力都不错。

## 五、结论

1.在以梯度下降法完成的 Logistic 回归实验中，通过生成数据来研究正则项与数据集是否满足朴素 Bayes 分布这两个因素对分类结果造成的影响。逻辑回归的公式是基于数据的类条件分布独立，也就是满足朴素 Bayes 的条件推出的，因此针对不满足这一条件的数据的分类效果将会下降。

2.在样本数量较小时，加入正则项可以有效避免过拟合的情况，对分类结果准确率有一定的提升，在样本数量较大时，是否加入正则项对分类准确率的影响较小。而逻辑回归对于线性分类问题能够给出一个相对好的分类器，能较好解决问题。

3.是否满足朴素 Bayes 分布决定样本数据的分布，若不满足朴素 Bayes 分布，则样本数据间存在着某种线性关系，但这对分类结果的影响不大。

4.通过 UCI 的 Skin Segmentation Data Set 进行分类测试来验证模型。在足够多轮的训练之后，模型在 UCI 数据集上表现出 90%左右的准确率，表现结果较好。

5.在使用 sigmoid 函数的时候如果不进行一定的处理可能出现数值的上溢或者下溢，因此需要适当的防止溢出操作。

## 六、参考文献

[1] 周志华,《机器学习》,清华大学出版社,2016

## 七、附录：源代码（带注释）

```

# -*- coding: gbk -*-

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# 下列为定义好的全局变量
sigma = 0.1#生成数据的标准差
n = 100#生成样本点的数量
naiveFlag = False#if true, 生成的数据满足朴素Bayes; 否则不满足朴素Bayes
epsilon = 1e-5#允许或者接受范围内的误差值
eta = 1e-1#学习率
lamda = 0#超参数  $\lambda$ 
epoch = 5000#训练次数

def generate_Data():#在范围区间[0, 1]内生成二维样本点数据
    a = n // 2#类别A的样本点个数
    b = n - a#类别B的样本点个数
    cov_xy = 0.08#两个维度的协方差（若不满足朴素Bayes的假设）
    x_mean_1 = [-0.75, -0.25]#类别1的均值
    x_mean_2 = [0.75, 0.25]#类别2的均值
    train_x = np.zeros((n, 2))#train_x中保存点的坐标(data)
    train_y = np.zeros(n)#train_y中保存点的类别标签(label)

    if naiveFlag:#若满足朴素Bayes的假设
        train_x[:a, :] = np.random.multivariate_normal(x_mean_1, [[sigma, 0], [0, sigma]],
size = a)
        train_x[a:, :] = np.random.multivariate_normal(x_mean_2, [[sigma, 0], [0, sigma]],
size = b)
        train_y[:a] = 0
        train_y[a:] = 1
    else:#若不满足朴素Bayes的假设
        train_x[:a, :] = np.random.multivariate_normal(x_mean_1, [[sigma, cov_xy], [cov_xy,
sigma]], size = a)
        train_x[a:, :] = np.random.multivariate_normal(x_mean_2, [[sigma, cov_xy], [cov_xy,
sigma]], size = b)
        train_y[:a] = 0
        train_y[a:] = 1

    return train_x, train_y

def generate_X(train_x):#生成X矩阵
    raw_X = np.ones((train_x.shape[0], 1))

```

```

for i in range(0, train_x.shape[1]):
    j = train_x[:, i]
    j = j.reshape((train_x.shape[0], 1))
    raw_X = np.hstack((raw_X, j))

X = raw_X.T
assert X.shape[0] == train_x.shape[1] + 1
assert X.shape[1] == train_x.shape[0]

return X # 生成两矩阵: (3,500) (4, 49011)

def sigmoid(x):#构建sigmoid () 函数: 为减缓溢出进行优化操作
    if x >= 0: # 对sigmoid () 函数进行优化, 避免出现过大无法弥补的数据溢出
        return 1.0 / (1 + np.exp(-x))
    else:
        return np.exp(x) / (1 + np.exp(x))

def gradient(X, train_y, w):#计算梯度
    grad = np.zeros((1, X.shape[0]))

    for i in range(0, X.shape[1]):
        wXT = w @ X[:, i]
        grad += X[:, i].T * (train_y[i] - sigmoid(wXT))

    return grad

def loss(X, train_y, w):#损失函数: 计算优化式子, 归一化
    loss = 0
    size = float(train_y.shape[0])

    for i in range(0, X.shape[1]):
        wXT = w @ X[:, i]
        loss += (train_y[i] * wXT / size - np.log(1 + np.exp(wXT)) / size)

    return loss

def gradient_descent(train_x, train_y):#梯度下降算法主函数部分
    eta_temp = eta
    w = np.ones((1, train_x.shape[1] + 1))#(1,3), 第一列全部置为1
    X = generate_X(train_x)
    grad = gradient(X, train_y, w)
    loss_list = []
    loss_0 = 0

```

```

loss_1 = loss(X, train_y, w)
count = 0

while count < epoch:#设定误差值，当小于该误差值时则停止迭代
    count += 1
    w = w - eta_temp * lamda * w + eta_temp * grad
    loss_0 = loss_1
    loss_1 = loss(X, train_y, w)

    if(np.abs(loss_1) > np.abs(loss_0)):#loss不降反增，则学习率减半，防止迭代过程中
步幅过大
        eta_temp *= 0.5
    grad = gradient(X, train_y, w)
    loss_list.append(np.abs(loss_0))
    print(count)

return w, count, loss_list

def loss_show(loss_list):#展示loss函数的变化情况
    epoch_list = [x for x in range(1, epoch, 1)]
    loss_list.remove(loss_list[0])
    plt.plot(epoch_list, loss_list, c='#DB4D6D', linewidth=2, label='loss')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend()
    plt.show()

    return 0

def accuracy(X, train_y, w):#计算分类准确率
    wrong = 0
    total = train_y.shape[0]

    for i in range(0, train_y.shape[0]):
        flag = train_y[i]
        value = w @ X[:, i]

        if((flag == 1 and value < 0) or (flag != 1 and value > 0)):
            wrong += 1
    temp = float(wrong) / total
    acc = temp if temp > 0.5 else 1 - temp

    return acc

```

```

def class_show(train_x, train_y, w, count):#绘制分类效果图
    xa = []
    xb = []
    ya = []
    yb = []

    for i in range(0, train_x.shape[0]):
        if train_y[i] == 0:
            xa.append(train_x[i][0])
            ya.append(train_x[i][1])
        elif train_y[i] == 1:
            xb.append(train_x[i][0])
            yb.append(train_x[i][1])

    plt.scatter(xa, ya, c='red', s=10, label='class A')
    plt.scatter(xb, yb, c='blue', s=10, label='class B')#绘制散点图，按照不同标签赋予颜色

    x_real = np.arange(-2, 2, 0.01)
    w0 = w[0][0]
    w1 = w[0][1]
    w2 = w[0][2]
    y_real = -(w1 / w2) * x_real - w0 / w2

    plt.plot(x_real, y_real, c='#DB4D6D', linewidth=2, label='fitting')
    plt.xlim(-2, 2)
    plt.ylim(-2, 2)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.title('data_size = ' + str(n) + ', ' + 'lamda = ' + str(lamda) + ', ' + 'naiveFlag'
= ' + str(naiveFlag))#梯度下降
    plt.show()

    return 0

def gradient_descent_show():#对梯度下降法结果进行展示
    train_x, train_y = generate_Data()
    #print(train_x, train_y)
    #print(train_x.shape, train_y.shape)
    X = generate_X(train_x)
    w, count, loss_list = gradient_descent(train_x, train_y)
    acc = accuracy(X, train_y, w)

```

```

print(w)
print("acc:" + str(acc))
class_show(train_x, train_y, w, count)
loss_show(loss_list)

return 0

def skin_read():#读取UCI Skin Segmentation数据集
    skin_data = np.loadtxt('./Skin_NonSkin.txt', dtype=np.int32)
    np.random.shuffle(skin_data)#打乱读入的数据，以便分成训练集和测试集
    test_rate = 0.25#测试集比例
    train_rate = 0.75#训练集比例
    step = 20#抽取部分样本的步长
    #print(skin_data.shape)#(245057, 4)
    #print(skin_data)
    new_data = skin_data[::step]
    n = new_data.shape[0]#样本数量
    raw_train_data = new_data[:int(test_rate * n), : ]#训练集
    raw_test_data = new_data[int(test_rate * n):, : ]#测试集
    train_x = raw_train_data[:, :-1] - 100
    train_y = raw_train_data[:, -1:] - 1
    test_x = raw_test_data[:, :-1] - 100
    test_y = raw_test_data[:, -1:] - 1
    #print(train_x.shape)#(980, 3)
    #print(train_y.shape)#(980, 1)

    return train_x, train_y, test_x, test_y

def skin_plot_show(train_x, train_y, w):#对Skin Segmentation数据集的运行结果进行三维展示
    xa = []
    xb = []
    ya = []
    yb = []
    za = []
    zb = []
    w0 = w[0][0]
    w1 = w[0][1]
    w2 = w[0][2]
    w3 = w[0][3]

    for i in range(0, train_x.shape[0]):
        if train_y[i] == 0:
            xa.append(train_x[i][0])
            ya.append(train_x[i][1])

```



```

        za.append(train_x[i][2])
    elif train_y[i] == 1:
        xb.append(train_x[i][0])
        yb.append(train_x[i][1])
        zb.append(train_x[i][2])

fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(xa, ya, za, c='red', s=10, label='class A')
ax.scatter(xb, yb, zb, c='blue', s=10, label='class B')#绘制散点图，按照不同标签赋予
颜色
real_x = np.arange(np.min(train_x[:,0]), np.max(train_x[:,0]), 1)
real_y = np.arange(np.min(train_x[:,1]), np.max(train_x[:,1]), 1)
real_X, real_Y = np.meshgrid(real_x, real_y)
real_z = - w0 / w3 - (w1 / w3) * real_X - (w2 / w3) * real_Y
ax.plot_surface(real_X, real_Y, real_z)
ax.legend(loc='best')

plt.title("Skin Dataset Classification")
plt.show()

return 0

def skin_show():#对skin数据集的训练结果进行展示
    train_x, train_y, test_x, test_y = skin_read()
    X = generate_X(train_x)
    w, count, loss_list = gradient_descent(train_x, train_y)
    test_X = generate_X(test_x)
    acc = accuracy(test_X, test_y, w)
    print(w)
    print("acc:" + str(acc))
    skin_plot_show(train_x, train_y, w)

    return w, count, loss_list

#主函数部分
method = 1
if method == 1:
    skin_show()
elif method == 2:
    gradient_descent_show()

```