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：必修

实验题目：逻辑回归

学号：1190202105

姓名：傅浩东

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

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贝叶斯假设，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。

实验环境：Windows 11; Visual Studio Code; python 3.9.7

三. 设计实验（主要算法和数据结构）

$$
\begin{aligned}
p(y=1 \mid \boldsymbol{x}) &= \frac{e^{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b}}{1+e^{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b}} \\
p(y=0 \mid \boldsymbol{x}) &= \frac{1}{1+e^{\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}+b}}
\end{aligned}
$$

可以通过 “极大似然法” (maximum likelihood method)来估计 $\boldsymbol{w}$ 和 b:

$$
\mathbf{w}_{MCLE} = \arg\max_{\mathbf{w}} \prod_{i=1}^{m} p\left(y_i \mid \boldsymbol{x}_i ; \boldsymbol{w}, b\right)
$$

给定数据集 $\left\{\left(\boldsymbol{x}_i, y_i\right)\right\}_{i=1}^m$, 对率回归模型最大化 “对数似然”，即上式左右两边同时取对数：

$$
\ell(\boldsymbol{w}, b) = \sum_{i=1}^{m} \ln p\left(y_i \mid \boldsymbol{x}_i ; \boldsymbol{w}, b\right)
$$

即让每个样本属于真实标记的概率越大越好。令 $\boldsymbol{\beta} = (\boldsymbol{w}; b)$, $\hat{\boldsymbol{x}} = (\boldsymbol{x}; 1)$, 则 $\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b$ 可简写为 $\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}$. 再令 $p_1(\hat{\boldsymbol{x}}; \boldsymbol{\beta}) = p(y=1 \mid \hat{\boldsymbol{x}}; \boldsymbol{\beta})$, $p_0(\hat{\boldsymbol{x}}; \boldsymbol{\beta}) = p(y=0 \mid \hat{\boldsymbol{x}}; \boldsymbol{\beta}) = 1 - p_1(\hat{\boldsymbol{x}}; \boldsymbol{\beta})$, 则上式中的似然项可重写为：

$$
p\left(y_i \mid \boldsymbol{x}_i ; \boldsymbol{w}, b\right) = y_i p_1\left(\hat{\boldsymbol{x}}_i ; \boldsymbol{\beta}\right) + \left(1-y_i\right) p_0\left(\hat{\boldsymbol{x}}_i ; \boldsymbol{\beta}\right)
$$

代入 $\ell(\boldsymbol{w}, b)$ 取反，则接下来应该最小化下式：

$$
\ell(\boldsymbol{\beta}) = \sum_{i=1}^{m}\left(-y_i \boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i + \ln\left(1+e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}\right)\right)
$$

此时为了避免过拟合加入惩罚项，从高斯分布考虑可以得到惩罚项应该为二次正则项：

$$
\ell(\boldsymbol{\beta}) = \frac{\lambda}{2} \beta^T \beta + \sum_{i=1}^{m}\left(-y_i \boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i + \ln\left(1+e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}\right)\right)
$$

1. 数据生成与基本操作

(1) 数据的随机生成

`numpy.random.multivariate_normal()` 函数是从多元正态分布中随机抽取样本的函数。多元正态分布、多重正态分布或高斯分布它是一维正态分布向更高维度的推广。这种分布由其均值和协方差矩阵来表示，在本次实验中，我随机生成的是二维正太函数，其中均值 `mean` 是一个包含两个元素的矩阵分别表示生成数据的 `x` 方向的均值和 `y` 方向的均值，然后是协方差 `cov` 如下所示，若满足朴素贝叶斯假设则 `b` 与 `c` 为0，即 `x` 与 `y` 独立，为对角矩阵；否则的话 `b` 与 `c` 不为0，那么就不满足朴素贝叶斯条件。

$$cov = \begin{Bmatrix} a, b \\ c, d \end{Bmatrix}$$

(2) Sigmoid函数

初始态Sigmoid函数设置为如下形态，但当 $x > 0$ 时，e^x 增长过快很容易就导致数据过大，造成溢出，所以修改为当 $x > 0$ 时利用形态 $e^{-x}/(1 + e^{-x})$ 来进行计算，就有效地避免了数据溢出。

```python
def sigmoid(x):
     return 1.0 / (1.0 + np.exp(x))

def sigmoid(x):
    if x <= 0:
        return 1.0 / (1.0 + np.exp(x))
    else:
        z = np.exp(-x)
        return z / (1.0 + z)
```

(3) 将数据分为训练集和测试集

给定训练样本比例和总体数据集，按照标签将正例和负例都均匀地分成成比例的训练集、测试集，再将正负例的训练测试样本分别合起来返回给调用方。在这里主要调用了两个函数 `numpy.vstack` 和 `numpy.concatenate` 分别将两个矩阵在列和行上合并起来。

(4) 计算准确度

计算准确度即验证测试集成功分类的比例，按照划分结果，将 $\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i > 0$，即 $sigmoid(\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i) < 0.5$ 且标签为 1 或者 $sigmoid(\boldsymbol{\beta}^{\mathrm{T}}\hat{\boldsymbol{x}}_i) > 0.5$ 且标签为 0 的测试结果视为正确分类，由此计算出测试集的分类准确度。

2. 梯度下降法

在这里，梯度下降法可以基于实验一的梯度下降法进行改动，基本上只需要改动损失函数 `loss` 和一阶导数 `derivative` 。对于一阶导数有：

$$\frac{\partial \ell(\beta)}{\partial \beta} = -\sum_{i=1}^{m} \hat{\boldsymbol{x}}_i \left(y_i - p_1\left(\hat{\boldsymbol{x}}_i; \boldsymbol{\beta}\right)\right) + \lambda\beta$$

其中当 λ 为0时，则认为没有添加惩罚项。

3. 牛顿法

从最小化一个标量变量的函数的最简单情况开始，比如$f(w)$，期望找到全局最小值$f(x*)$的自变量 $w*$ 的位置。假设 f 是平滑的，并且 $w*$是内部最小值，这意味着 $w*$ 处的导数为零，二阶导数为正，当x逐渐接近最小值，可以进行泰勒展开，其中后半部分是泰勒级数展开余数的拉格朗日形式，如下：

$$f(w) \approx f\left(w^*\right)+\frac{1}{2}\left(w-w^*\right)^2 \left.\frac{d^2 f}{d w^2}\right|_{w=w^*}$$

这里应该确保二阶导数必须是正的，以保证 $f(w)>f(w0*)$。换句话说，$f(w)$ 在最小值附近接近二次。 牛顿法就是最小化我们真正感兴趣的函数的二次近似值。假设一个初始点 w_0，如果这接近最小值，我们可以在 w_0 附近进行二阶泰勒展开，仍然有：

$$f(w) \approx f\left(w_0\right)+\left(w-w_0\right)\left.\frac{d f}{d w}\right|_{w=w_0}+\frac{1}{2}\left(w-w_0\right)^2 \left.\frac{d^2 f}{d w^2}\right|_{w=w_0}$$

假设：

$$\left.\frac{d f}{d w}\right|_{w=w_0}=f^{\prime}\left(w_0\right),\left.\frac{d^2 f}{d w^2}\right|_{w=w_0}=f^{\prime \prime}\left(w_0\right)$$

取关于 w 的导数，并假设在 w_1 点处将其设置为零，那么有：

$$\begin{aligned} 0&=f^{\prime}\left(w_0\right)+\frac{1}{2} f^{\prime \prime}\left(w_0\right) 2\left(w_1-w_0\right) \\ w_1&=w_0-\frac{f^{\prime}\left(w_0\right)}{f^{\prime \prime}\left(w_0\right)}\end{aligned}$$

与初始值 w_0 相比，值 w_1 应该更接近于最小值 w^* 。 因此，如果使用它对 f 进行二次近似，将会获得更好的近似值，因此我们可以迭代此过程，最小化一个近似值，然后使用它来获得新的近似值：

$$w^{n+1}=w^n-\frac{f^{\prime}\left(w^n\right)}{f^{\prime \prime}\left(w^n\right)}$$

真正的最小值 w^* 是一个不动点：在收敛过程中，如果碰巧落在它上面，即 $f^{\prime}(w^*)=0$ 则迭代结束；对于一般情况，如果 w_0 足够接近 w^*，则 $w_n \rightarrow w^*$ ，并且有 $|w_n-w^*|=O(n^{-2})$，收敛速度非常快。接下来，对于目标函数f，假设它是具有多个参数的函数$f(w_1, w_2, \cdots, w_n)$，那么函数更新为：

$$w^{n+1}=w^n-H^{-1}\left(w^n\right) \nabla f\left(w^n\right)$$

将 w 替换为β ，f 替换为 l 有，β 与 l 具体含义见前面表述：

$$\boldsymbol{\beta}^{t+1}=\boldsymbol{\beta}^t-\left(\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}}\right)^{-1} \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$$

关于一阶导数 ∇l 和二阶导数（海森矩阵）H^{-1}有：

$$\begin{aligned} \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}&=-\sum_{i=1}^m \hat{\boldsymbol{x}}_i\left(y_i-p_1\left(\hat{\boldsymbol{x}}_i ; \boldsymbol{\beta}\right)\right)+\lambda \beta \\ \frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta} \partial \boldsymbol{\beta}^{\mathrm{T}}}&=\lambda+\sum_{i=1}^m \hat{\boldsymbol{x}}_i \hat{\boldsymbol{x}}_i^{\mathrm{T}} p_1\left(\hat{\boldsymbol{x}}_i ; \boldsymbol{\beta}\right)\left(1-p_1\left(\hat{\boldsymbol{x}}_i ; \boldsymbol{\beta}\right)\right)\end{aligned}$$

其中从前文可知：

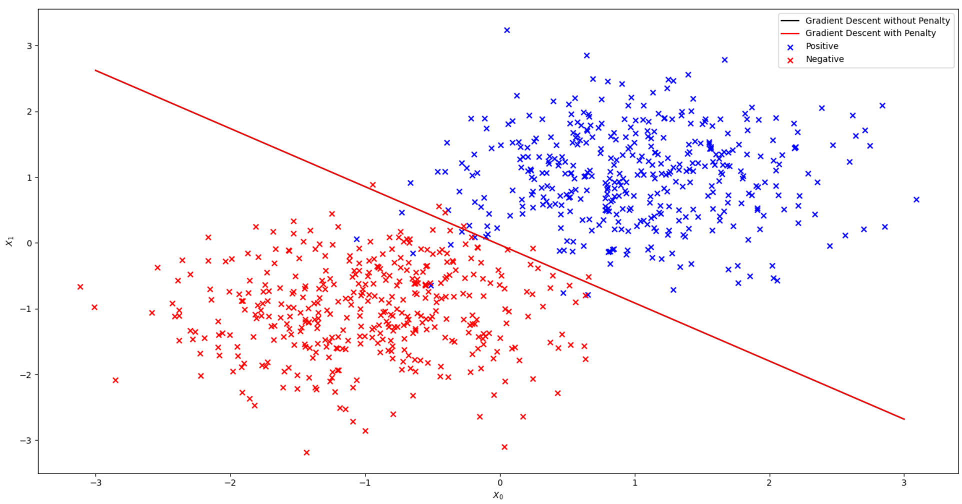
$$p_1\left(\hat{\boldsymbol{x}}_i ; \boldsymbol{\beta}\right)=\frac{e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}}{1+e^{\boldsymbol{\beta}^{\mathrm{T}} \hat{\boldsymbol{x}}_i}}$$

对于一般情况，按照此迭代，假设一个忍耐值`tolerance`当一阶导数小于这个值时就认为函数收敛；而当迭代次数超过`max.iter`则认为不收敛。当然这里还暂时忽略了一些潜在的问题，例如：当收敛时落在一个点使得$\ell''(\beta)=0$，或者当出现$\ell(\beta^{n+1})>\ell(\beta^{n})$等各种特殊情况的处理。

四. 实验结果与分析

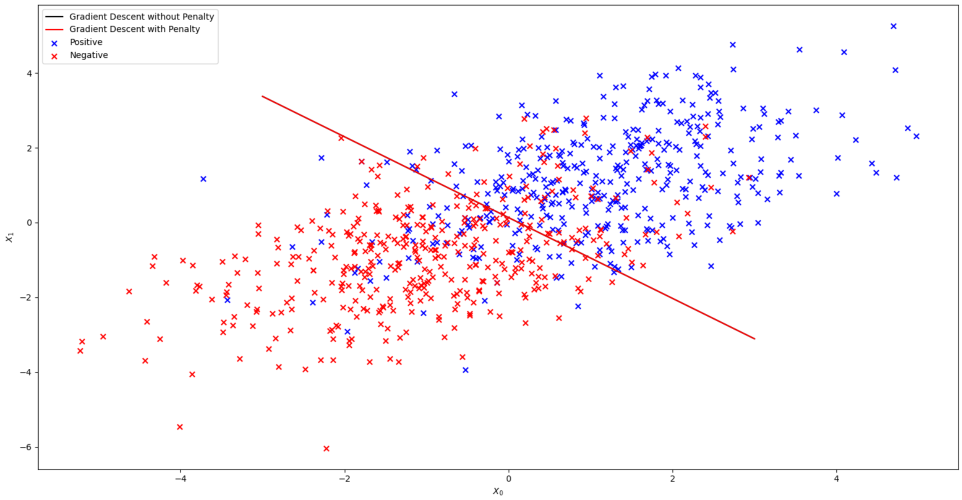
设置正例500、反例500，按照训练样本比例0.8生成分开二维特征数据，分别用梯度下降法和牛顿法进行训练，得到的结果与准确度如下所示。

1. 梯度下降法



Methods	Number of iterations	$beta(b, w_0, w_1)$	Accuracy
Gradient Descent without Penalty	3266	[0.11219175 3.49767623 3.95767663]	0.96
Gradient Descent with Penalty	3259	[0.11190345 3.49486324 3.95409359]	0.96

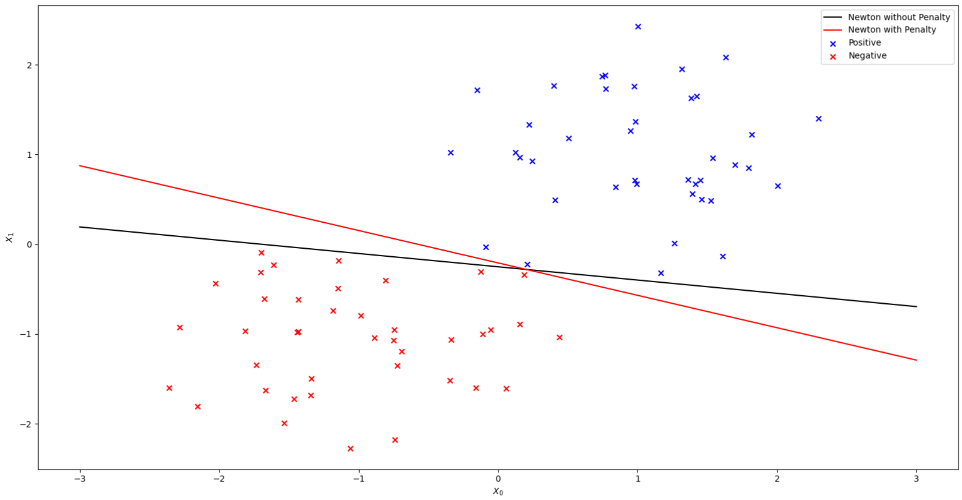
以上是两维独立数据的结果，可以看出惩罚项对拟合影响不太大，并且拟合次数、准确度等结果很相似。接下来考虑破坏各个维度之间的条件独立性，将协方差矩阵进行设置，使其不为对角阵，从结果来看不仅迭代次数降低了，同时影响到了准确度，这是因为非独立的条件使得正负例产生的交集变多。



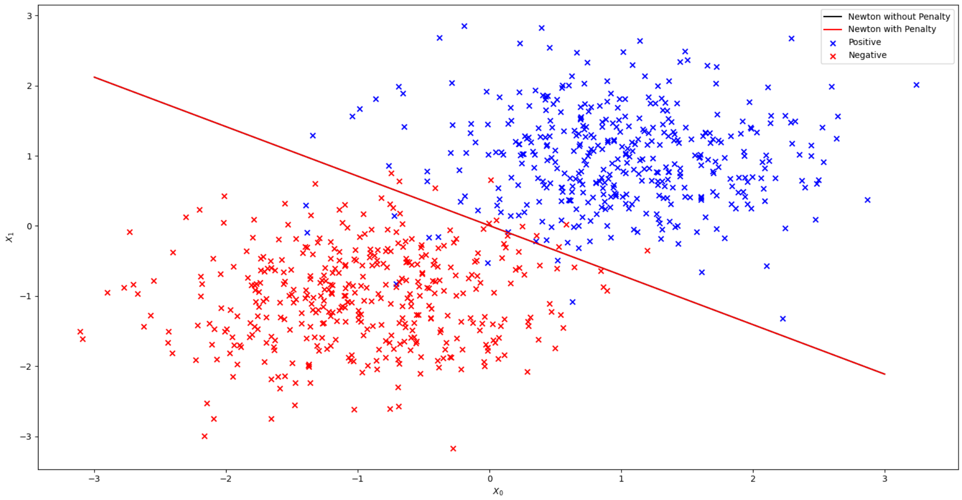
Methods	Number of iterations	$beta(b, w_0, w_1)$	Accuracy
Gradient Descent without Penalty	154	[-0.08929811 0.72853495 0.67408851]	0.79
Gradient Descent with Penalty	154	[-0.08929617 0.72852688 0.67408119]	0.79

2. 牛顿法

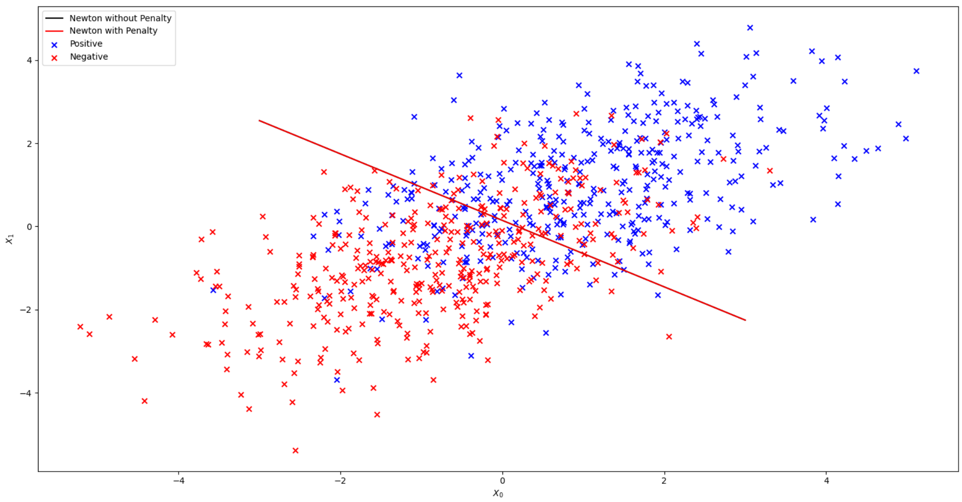
同上述表示，牛顿法运行结果如下，逻辑回归分类器在在满足朴素贝叶斯假设时分类良好，在不满足朴素贝叶斯假设时分类效果有较大影响，这是由于数据生成过时交集范围变大。在二维条件下，是否有惩罚项对分类效果影响也不大，但当减小训练集时，所得到的判别函数确实存在过拟合现象，加入正则项可以预防此现象的发生。如下，先是正负例分别只有50的牛顿法分类结果，再是与上述梯度下降相同数据集的分类结果。可以清晰地看到此时惩罚项对分类结果的影响。



Methods	Number of iterations	$beta(b, w_0, w_1)$	Accuracy
Newton Method without Penalty	20	[51.86001313 30.60262642 206.83661069]	0.95
Newton Method with Penalty	12	[3.74227712 6.5071256 18.01859339]	0.95



Methods	Number of iterations	$beta(b, w_0, w_1)$	Accuracy
Newton Method without Penalty	9	[-0.01531409 2.90311124 4.11291967]	0.97
Newton Method with Penalty	9	[-0.01532468 2.90128253 4.10963984]	0.97



Methods	Number of iterations	$beta(b, w_0, w_1)$	Accuracy
Newton Method without Penalty	6	[0.01268956 0.74630746 0.69016628]	0.81
Newton Method with Penalty	6	[0.01268935 0.74629886 0.69015811]	0.81

3. UCI实验数据

(1) Cervical Cancer Behavior Risk Data Set

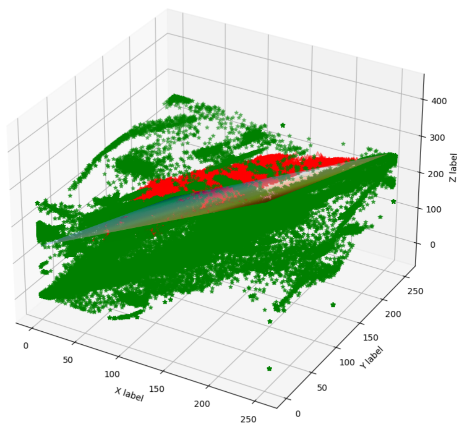
该数据集包含 19 个关于宫颈癌行为风险的属性，类标签为**ca_cervix**，值为 1 和 0，分别表示有和没有宫颈癌的受访者。该数据集包括18个属性，来自八个变量，分别是：**behavior** (sexualRisk，eating，personalHygine); **intention**(aggregation，commitment); **attitude** (consistency, spontaneity); **norm** (significantPerson, fulfillment); **perception** (vulnerability, severity); **motivation** (strength, willingness); **socialSupport** (emotionality, appreciation, instrumental); **empowerment** (knowledge, abilities, desires)。训练结果与准确度如下：

Methods	Iterations	$beta(b, w)$	Accuracy
Newton Method without Penalty	21	[65.28172977 -4.97324863 4.40331064 -2.15781802 -1.1690409 -1.89339197 6.68911897 3.01668879 2.22863228 -2.75609484 3.44211169 -9.4991092 -2.87701841 4.40045041 -2.43220015 -4.18802602 2.6645648 -3.45533425 0.5878001 -3.28302278]	1.0
Newton Method with Penalty	14	[0.26003302 -0.02594259 2.16159946 -0.77710536 -1.0931356 -0.54213705 1.45851991 2.51249146 0.85542644 -0.94760722 -0.29091714 -2.72642556 -0.0612657 0.93096871 -0.49151131 -1.51004841 1.4617037 -1.85228891 -0.36097548 -1.28503274]	0.92857
Gradient Descent without Penalty	7171	[0.12610426 0.14312901 1.17905612 -0.41636272 -0.59202367 -0.28563031 0.658336 1.33356021 0.28689991 -0.48189438 -0.29346668 -1.32757603 -0.03234814 0.4270257 -0.1347684 -0.68640519 0.77571812 -0.8883878 -0.37785854 -0.77284104]	0.92857
Gradient Descent with Penalty	7112	[0.12560722 0.14273005 1.17487291 -0.41462575 -0.58962038 -0.28452748 0.65589416 1.32822214 0.28549159 -0.48020473 -0.29259047 -1.32230554 -0.03250953 0.42533177 -0.13403129 -0.68333912 0.77277757 -0.88480299 -0.37688107 -0.77029605]	0.92857

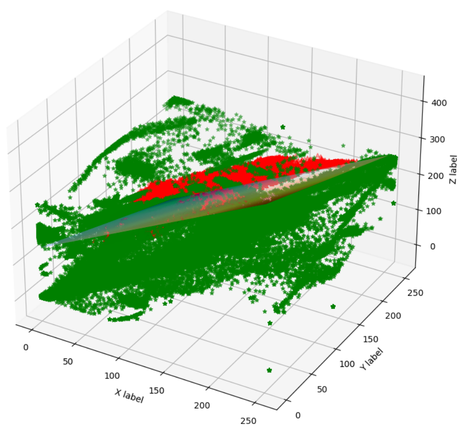
(2) Skin Segmentation Data Set

皮肤分割数据集是在 **B、G、R** 颜色空间上构建的。皮肤和非皮肤数据集是使用来自不同年龄、性别和种族的人脸图像的皮肤纹理生成的。皮肤数据集是通过从 FERET 数据库和 PAL 数据库获得的各个年龄组（年轻、中年和老年）、种族组（白人、黑人和亚洲人）和性别的人脸图像中随机采样 B、G、R 值来收集的。总学习样本数量为 245057；其中50859个是皮肤样本，194198个是非皮肤样本。对应到数据文件，前三列是B、G、H特征值，第四列是类标签。

此处考虑到数据集的庞大，首先是梯度下降法，设`tolerance` 为 e^{-5}，即当`loss`小于这个值时就认为函数收敛，按照此处一步步地梯度下降计算，运行将近一个小时，由之前牛顿法的表述可以看出当数据很大时，惩罚项对分类结果影响不大，所以在此时先暂时忽略，那么有以下结果。首先给出梯度下降运行结果，准确度大约达到 **92%**：



然后是牛顿法运行结果，大约迭代 **7**次，准确度约为 **92%**，与梯度下降法很相近：



五. 结论

- 牛顿法循环迭代的时间代价相比梯度下降法很低，主要是由于梯度下降迭代次数很高，而牛顿法一般只需要30次以内的迭代次数就能找到最小值；
- 牛顿法的计算过程中涉及求海森矩阵的逆，如果矩阵奇异，则不再适用，除此以外还有许多特殊情况没有进一步处理，例如当收敛时落在一个点使得$\ell''(\beta)=0$，或者当出现$\ell(\beta^{n+1})>\ell(\beta^n)$等各种特殊情况；

- 总体来说，牛顿法与梯度下降法都可以得到很好的结果，但是梯度下降法的迭代收敛的速度更慢。
- 是否满足朴素贝叶斯对结果有一定的影响，当然这取决于正负例的均值中心的临近程度，如果很相近就算是满足朴素贝叶斯准确率也会很低，所以在看准确率时不能只看是否满足朴素贝叶斯的条件，这是一个较为复杂的事情。
- 当数据集很大时，是否有惩罚项对分类效果影响不大，但当减小训练集时，所得到的判别函数确实存在过拟合现象，加入正则项可以预防此现象的发生。

六. 参考文献

[1] https://numpy.org/devdocs/reference/random/generated/numpy.random.multivariate_normal.html

[2] https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization

[3] https://en.wikipedia.org/wiki/Newton%27s_method#Code

[4] http://www.stat.cmu.edu/~cshalizi/350/lectures/26/lecture-26.pdf

[5] https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation

[6] https://archive.ics.uci.edu/ml/datasets/Cervical+Cancer+Behavior+Risk

七. 附录：源代码（带注释）

数据处理 与基本操作 operation.py

```
import math
import numpy as np

def sigmoid(x):
    "Numerically stable sigmoid function."
    if x <= 0:
        return 1.0 / (1.0 + np.exp(x))
    else:
        z = np.exp(-x)
        return z / (1.0 + z)

# def sigmoid(x):
#     return 1.0 / (1.0 + np.exp(x))


def Data(naive=True, N_1=500, N_0=500):
    """随机生成一组正例和一组负例，特征是二维的，可以选择条件是否满足朴素贝叶斯
    Args:
        naive (bool, optional): 是否满足朴素贝叶斯条件，True表示满足（即条件独立）.
Defaults to True.
        N_1 (int, optional): 正例数量. Defaults to 500.
        N_0 (int, optional): 负例数量. Defaults to 500.
    Returns:
        array: 返回生成的特征数组x和标签y（1或0）
    """
    mean_1 = [1, 1]
    mean_0 = [-1, -1]
    cov_naive = [[0.5, 0], [0, 0.5]]
    cov_NOT_naive = [[2, 1], [1, 2]]
    y = np.zeros(N_1+N_0).astype(np.int32)
    # 满足朴素贝叶斯假设
```

```python
    if naive:
        cov = cov_naive
    # 不满足朴素贝叶斯假设
    else:
        cov = cov_NOT_naive
    x_1 = np.random.multivariate_normal(mean_1, cov, size=N_1)
    x_0 = np.random.multivariate_normal(mean_0, cov, size=N_0)
    x = np.vstack((x_1, x_0))
    y = np.zeros(N_1+N_0).astype(np.int32)
    y[:N_1] = 1
    y[N_1:] = 0
    return x, y

def SplitData(x, y, trainRate=0.8):
    """划分测试集和训练集
    Args:
        x (array): 全部数据的特征集
        y (array): 一维数组，全部数据的标签，1或0
        trainRate (float, optional): 训练样本要占全部数据的比例. Defaults to 0.8.
    Returns:
        array: 训练样本和测试样本的特征和标签矩阵
    """
    N_1 = x[y == 1].shape[0]
    trainNum_1 = int(math.ceil(N_1 * trainRate))
    N_0 = x[y == 0].shape[0]
    trainNum_0 = int(math.ceil(N_0 * trainRate))
    # 训练集
    Train_x = np.vstack((x[:trainNum_1], x[N_1:N_1+trainNum_0]))
    Train_y = np.concatenate((y[:trainNum_1], y[N_1:N_1+trainNum_0]))
    # 测试集
    Test_x = np.vstack((x[trainNum_1:N_1], x[N_1+trainNum_0:]))
    Test_y = np.concatenate((y[trainNum_1:N_1], y[N_1+trainNum_0:]))

    return Train_x, Train_y, Test_x, Test_y

def x2xPlus(x):
    """在初始数据集之前加上一列1，使其符合beta的计算要求
    Args:
        x (array): 从数据中直接获取的特征，每一行都代表一个数据的各个特征
    Returns:
        array: 相较于x在前面多出了一列1
    """
    xPlus = np.column_stack((np.ones(x.shape[0]).T, x))
    return xPlus


def accuracy(Test_x, Test_y, beta):
    """ 计算训练结果的准确度，查看测试数据在分类面的哪一边与标签是否符合
    Args:
        Test_x (array): 特征数据，在这里Test_x的第一列包含1，这主要是为了配合beta的计算而
设置的
        Test_y (array): 一维标签，正例为1，负例为0
        beta (array): 一维数组，(b, w0, w1, ... ,wn)
    Returns:
        [float]: 测试集中分类成功比例，即准确度
    """
    columns = len(Test_x)
    count = 0
```

```
    for i in range(columns):
        if sigmoid(beta @ Test_x[i]) < 0.5 and Test_y[i] == 1:
            count += 1
        elif sigmoid(beta @ Test_x[i]) > 0.5 and Test_y[i] == 0:
            count += 1
    return count / columns
```

牛顿法 Newton.py

```
import numpy as np
from numpy.matrixlib.defmatrix import matrix
from operation import *
import matplotlib.pyplot as plt
import prettytable as pt

# 牛顿法
class Newton(object):
    def __init__(self, x, y, beta_0, hyper=0, tolerance=1e-6, max_iter = 50):
        """牛顿法初始化变量
        Args:
            x (array): 在原始特征array前加了一列的数组
            y (array): 训练样本的标签
            beta_0 (array): 初始化beta，一般是0
            hyper (int, optional): 惩罚项的系数，即lambda. Defaults to 0.
            tolerance (float, optional): 容忍度，即当一阶导数均小于这个值时认为收敛.
Defaults to 1e-6.
            max_iter (int, optional): 最多迭代次数，超过这个值认为不收敛. Defaults to
50.
        """
        self.x = x
        self.y = y
        self.beta_0 = beta_0
        self.hyper = hyper
        self.tolerance = tolerance
        self.max_iter = max_iter
        self.__row = len(x)
        self.__col = len(x.T)

    def __derivative(self, beta):
        """求一阶导数
        Args:
            beta (array): 特征值的系数
        Returns:
            array: 根据beta求出的导数值
        """
        ans = np.zeros(self.__col)
        for i in range(self.__row):
            ans += (self.x[i] * (self.y[i] - sigmoid( - beta @ self.x[i].T)))
        return - ans + self.hyper * beta

    def __hessian(self, beta):
        """求二阶导数，即海森矩阵
        Args:
            beta (array): 特征值系数
        Returns:
```

```
            array: 根据beta得到的二阶导数
        """
        ans = np.eye(self.__col) * self.hyper
        for i in range(self.__row):
            temp = sigmoid(beta @ self.x[i].T)
            m = np.mat(self.x[i]).T
            ans += np.array(m * m.T) * temp * (1 - temp)
        return ans

    def fit(self):
        k = 0
        beta = self.beta_0
        while k <= self.max_iter:
            gradient = self.__derivative(beta)
            if np.linalg.norm(gradient) < self.tolerance:
                break
            hess = self.__hessian(beta)
            beta_t = beta - np.linalg.inv(hess) @ gradient
            beta = beta_t
            k += 1
        return k, beta

if __name__ == '__main__':
    # x, y = Data(naive=False)
    x, y = Data()
    xPlus = x2xPlus(x)
    Train_x, Train_y, Test_x, Test_y = SplitData(xPlus, y)
    beta_0 = np.zeros(xPlus.shape[1])

    hyper = np.exp(-6)
    # 无惩罚项（正则项）的牛顿法
    newton = Newton(Train_x, Train_y, beta_0)
    k_newton, beta_newton = newton.fit()
    accuracy_newton = accuracy(Test_x, Test_y, beta_newton)
    # 带惩罚项（正则项）的牛顿法
    newton_penalty = Newton(Train_x, Train_y, beta_0, hyper=hyper)
    k_newton_penalty, beta_newton_penalty = newton_penalty.fit()
    accuracy_newton_penalty = accuracy(Test_x, Test_y, beta_newton_penalty)

    # 训练样本
    type1_x = Train_x[Train_y==1][:,1]
    type1_y = Train_x[Train_y==1][:,2]
    type0_x = Train_x[Train_y==0][:,1]
    type0_y = Train_x[Train_y==0][:,2]
    plt.scatter(type1_x, type1_y, marker="x", c="b", label="Positive")
    plt.scatter(type0_x, type0_y, marker="x", c="r", label="Negative")

    # 无惩罚项的结果图
    x_results = np.linspace(-3, 3)
    y_results = - (beta_newton[0] +beta_newton[1] * x_results) / beta_newton[2]
    plt.plot(x_results, y_results, color="k", label='Newton without Penalty')
    # 带惩罚项的结果图
    y_results_penalty = - (beta_newton_penalty[0] +beta_newton_penalty[1] *
x_results) / beta_newton_penalty[2]
    plt.plot(x_results, y_results_penalty, color="r", label='Newton with
Penalty')

    plt.xlabel("$X_0$")
```

```
    plt.ylabel("$X_1$")
    plt.legend(loc='best')
    plt.show()

    tb = pt.PrettyTable()
    tb.field_names = ["Methods", "Number of iterations", "beta(b, w0, w1)",
"Accuracy"]
    tb.add_row(["Newton Method without Penalty", k_newton, beta_newton,
accuracy_newton])
    tb.add_row(["Newton Method with Penalty", k_newton_penalty,
beta_newton_penalty, accuracy_newton_penalty])
    print(tb)
```

梯度下降法 gradient_descent.py

```
import numpy as np
from operation import *
import matplotlib.pyplot as plt
import prettytable as pt

# 梯度下降
class GradientDescent(object):
    def __init__(self, x, y, beta_0, hyper=0, rate=0.1, tolerance=1e-6):
        """梯度下降法初始化数据

        Args:
            x (array): 在原始特征array前加了一列的数组
            y (array): 训练样本的标签
            beta_0 (array): 初始化beta，一般是0
            hyper (int, optional): 惩罚项的系数，即lambda. Defaults to 0.
            rate (float, optional): 学习率，即梯度下降步长. Defaults to 0.1.
            tolerance (float, optional): 容忍度，即当一阶导数均小于这个值时认为收敛.
Defaults to 1e-6.
        """
        self.x = x
        self.y = y
        self.beta_0 = beta_0
        self.hyper = hyper
        self.rate = rate
        self.tolerance = tolerance
        self.__row = len(x)
        self.__col = len(x.T)

    def __loss(self, beta):
        ans = 0.5 * self.hyper * beta @ beta.T
        for i in range(self.__row):
            ans -= self.y[i] * beta @ self.x[i].T
            ans += np.log(1 + np.exp(beta @ self.x[i].T))
        return ans / self.__row

    def __derivative(self, beta):
        ans = np.zeros(self.__col)
        for i in range(self.__row):
            ans += self.x[i] * (self.y[i] - (1.0 - sigmoid(beta @ self.x[i].T)))
        return (-1 * ans + self.hyper * beta) / self.__row
```

```
    def fit(self):
        losses = []
        loss_0 = self.__loss(self.beta_0)
        losses.append(loss_0)
        k = 0
        beta = self.beta_0
        while True:
            der = self.__derivative(beta)
            beta_t = beta - self.rate * der
            loss = self.__loss(beta_t)
            losses.append(loss)
            if np.abs(loss - loss_0) < self.tolerance:
                break
            else:
                k += 1
                if loss > loss_0:
                    self.rate *= 0.5
                loss_0 = loss
                beta = beta_t
        return k, beta, losses


if __name__ == '__main__':
    # 获取数据，默认为正例500，负例500，且满足朴素贝叶斯假设（互不相关）
    # x, y = Data()
    x, y = Data(naive=False)
    xPlus = x2xPlus(x)
    Train_x, Train_y, Test_x, Test_y = SplitData(xPlus, y)
    beta_0 = np.zeros(xPlus.shape[1])

    hyper = np.exp(-6)
    # 无惩罚项（正则项）的梯度下降法
    gradient_descent = GradientDescent(Train_x, Train_y, beta_0)
    k_gradient, beta_gradient, losses_gradient = gradient_descent.fit()
    accuracy_gradient = accuracy(Test_x, Test_y, beta_gradient)
    # 带惩罚项（正则项）的梯度下降法
    gradient_descent_penalty = GradientDescent(Train_x, Train_y, beta_0,
hyper=hyper)
    k_gradient_penalty, beta_gradient_penalty, losses_penalty =
gradient_descent_penalty.fit()
    accuracy_gradient_penalty = accuracy(Test_x, Test_y, beta_gradient_penalty)

    # 画出二维参数的样本
    type1_x = Train_x[Train_y==1][:,1]
    type1_y = Train_x[Train_y==1][:,2]
    type0_x = Train_x[Train_y==0][:,1]
    type0_y = Train_x[Train_y==0][:,2]

    plt.scatter(type1_x, type1_y, marker="x", c="b", label="Positive")
    plt.scatter(type0_x, type0_y, marker="x", c="r", label="Negative")

    # 无惩罚项的结果图
    x_results = np.linspace(-3, 3)
    y_results = - (beta_gradient[0] +beta_gradient[1] * x_results) /
beta_gradient[2]
    plt.plot(x_results, y_results, color="k", label='Gradient Descent without
Penalty')
```

```
    # 带惩罚项的结果图
    y_results_penalty = - (beta_gradient_penalty[0] +beta_gradient_penalty[1] *
x_results) / beta_gradient_penalty[2]
    plt.plot(x_results, y_results_penalty, color="r", label='Gradient Descent
with Penalty')

    plt.xlabel("$X_0$")
    plt.ylabel("$X_1$")
    plt.legend(loc='best')
    plt.show()

    tb = pt.PrettyTable()
    tb.field_names = ["Methods", "Number of iterations", "beta(b, w0, w1)",
"Accuracy"]
    tb.add_row(["Gradient Descent without Penalty", k_gradient, beta_gradient,
accuracy_gradient])
    tb.add_row(["Gradient Descent with Penalty", k_gradient_penalty,
beta_gradient_penalty, accuracy_gradient_penalty])
    print(tb)
```

UCI皮肤实例 skin.py

```
import numpy as np
import pandas as pd
from operation import *
from newton import *
from gradient_descent import *

def GetData():
    data_set = pd.read_csv("./data/Skin_NonSkin.csv")
    x = data_set.drop('y', axis=1)
    y = data_set['y']
    new_y = np.copy(y)
    new_y[y==2] = 0
    return x, new_y

if __name__ == '__main__':
    x, y = GetData()
    xPlus = x2xPlus(x)
    Train_x, Train_y, Test_x, Test_y = SplitData(xPlus, y)
    # 无惩罚项（正则项）的梯度下降
    beta_0 = np.zeros(xPlus.shape[1])
    # newton = Newton(Train_x, Train_y, beta_0, hyper=np.exp(-6))
    # k, beta = newton.fit()
    # print(accuracy(Test_x, Test_y, beta))
    # 带惩罚项（正则项）的梯度下降
    # gradient_descent = GradientDescent(Train_x, Train_y, beta_0,
hyper=np.exp(-6))
    # k, beta, losses = gradient_descent.fit()
    # print(k, beta)
    # accuracy_gradient = accuracy(Test_x, Test_y, beta)
    # 0.9199959192001632

    newton = Newton(Train_x, Train_y, beta_0)
    k, beta = newton.fit()
```

```
    accuracy_newton = accuracy(Test_x, Test_y, beta)
    print(k, accuracy_newton)

    x_results = np.arange(0, 250, 0.25)
    y_results = np.arange(0, 250, 0.25)
    x_results, y_results = np.meshgrid(x_results, y_results)
    z_results = - (beta[0] + beta[1] * x_results + beta[2] * y_results) /
beta[3]

    data_1 = x[y==1]
    data_0 = x[y==0]
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(data_1['B'],data_1['G'],data_1['R'],c='r',marker='^')
    ax.scatter(data_0['B'],data_0['G'],data_0['R'],c='g',marker='*')

    ax.plot_surface(x_results, y_results, z_results, rstride = 1, cstride = 1,
cmap = plt.get_cmap('coolwarm'))

    ax.set_xlabel('X label')
    ax.set_ylabel('Y label')
    ax.set_zlabel('Z label')
    # plt.savefig('./skin.jpg')
    plt.show()
```

宫颈癌实例 sobar.py

```
import numpy as np
import pandas as pd
from operation import *
from newton import *
from gradient_descent import *
import prettytable as pt

def GetData():
    data_set = pd.read_csv("./data/sobar-72.csv")
    x = data_set.drop('ca_cervix', axis=1)
    y = data_set['ca_cervix']
    return x, y

if __name__ == '__main__':
    x, y = GetData()
    xPlus = x2xPlus(x)
    Train_x, Train_y, Test_x, Test_y = SplitData(xPlus, y)
    beta_0 = np.zeros(xPlus.shape[1])

    hyper = np.exp(-6)
    # 无惩罚项（正则项）的牛顿法
    newton = Newton(Train_x, Train_y, beta_0)
    k_newton, beta_newton = newton.fit()
    accuracy_newton = accuracy(Test_x, Test_y, beta_newton)
    # 带惩罚项（正则项）的牛顿法
    newton_penalty = Newton(Train_x, Train_y, beta_0, hyper=hyper)
    k_newton_penalty, beta_newton_penalty = newton_penalty.fit()
```

```
    accuracy_newton_penalty = accuracy(Test_x, Test_y, beta_newton_penalty)

    # 无惩罚项（正则项）的梯度下降法
    gradient_descent = GradientDescent(Train_x, Train_y, beta_0)
    k_gradient, beta_gradient, losses_gradient = gradient_descent.fit()
    accuracy_gradient = accuracy(Test_x, Test_y, beta_gradient)
    # 带惩罚项（正则项）的梯度下降法
    gradient_descent_penalty = GradientDescent(Train_x, Train_y, beta_0,
hyper=hyper)
    k_gradient_penalty, beta_gradient_penalty, losses_penalty =
gradient_descent_penalty.fit()
    accuracy_gradient_penalty = accuracy(Test_x, Test_y, beta_gradient_penalty)

    tb = pt.PrettyTable()
    tb.field_names = ["Methods", "Number of iterations", "beta(b, w0, w1)",
"Accuracy"]
    tb.add_row(["Newton Method without Penalty", k_newton, beta_newton,
accuracy_newton])
    tb.add_row(["Newton Method with Penalty", k_newton_penalty,
beta_newton_penalty, accuracy_newton_penalty])
    tb.add_row(["Gradient Descent without Penalty", k_gradient, beta_gradient,
accuracy_gradient])
    tb.add_row(["Gradient Descent with Penalty", k_gradient_penalty,
beta_gradient_penalty, accuracy_gradient_penalty])
    print(tb)
```