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： 考查课

实验题目： 逻辑回归

学号：
姓名：

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。

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。

②实验环境:

Windows10; python 3.9.12; Visual Studio Code; Anaconda.

三、设计思想（本程序中的用到的主要算法及数据结构）

算法目标：

从训练集<X^i, Y^i>(其中 X 为向量,Y 为 0 或 1)中学习一个分类器 f : X →Y 以便预测一个新的样本 X 所属的类别 Y.

实验假设：

1. X 的每一维属性 X_i 都是实数，X 可视为 n 维向量。

2. Y 是 boolean 值，取值为 1 或 0。

3. X_i 关于 Y 条件独立。

4. $P(X_i|Y = yk) \sim N(\mu_{ik},\ \sigma_i)$（方差与 Y 的取值无关）

5. $P(Y) \sim B(\Pi)$

代价函数推导：

由贝叶斯公式，带入可知：

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

根据变量符合正态分布的假设

$$
P(X_i|Y=y_k) = \frac{1}{\sigma_i\sqrt{2\pi}}\exp(\frac{-(X_i-\mu_{ik})^2}{2\sigma_i^2})
$$

带入可得，

$$
\begin{aligned}
P(Y=1|X) &= \frac{1}{1+\exp(w_0+\sum_{i=1}^n w_i X_i)} \\
w_0 = \sum_i \frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2} &+ \ln\frac{1-\pi}{\pi};\ w_i = \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}
\end{aligned}
$$

因此，

$$
\begin{aligned}
P(Y=0|X) &= \frac{\exp(w_0+\sum_{i=1}^n w_i X_i)}{1+\exp(w_0+\sum_{i=1}^n w_i X_i)} \\
\ln\frac{P(Y=0|X)}{P(Y=1|X)} &= \ln(\exp(w_0+\sum_{i=1}^n w_i X_i)) = w_0+\sum_{i=1}^n w_i X_i
\end{aligned}
$$

在分类中，可以利用 odds 进行分类:

$$\mathrm{logit}(p) = \ln\frac{p}{1-p}$$

依据 logit(p)>0 还是<0，来判定事件发生还是不发生，这便是 odds 概念的作用。在逻辑回归问题中，带入得

$$\mathrm{logit}(Y = 0|X) = w_0 + \sum_{i=1}^{n} w_i X_i$$

logit(Y=0｜X)>0 则将 X 分到 Y=0 类，若 logit(Y=0｜X)<0 则将 X 分到 Y=1 类，在实际的计算中，可以给 X 向量扩展一个维度 1，w 向量扩展一个维度 w_0, 记作:

$$w^T X = 0$$
$$w = [w_0, w_1 \ldots w_n], X = [1, X_1, X_2 \ldots X_n]$$

其中，可以引入 sigmoid 函数。

$$\mathrm{sig}(t) = \frac{1}{1+e^{-t}}$$

并结合机器误差对计算方法进行优化，防止溢出。

```
#求sigmoid函数
def sigmoid(x):
    if x >= 0:
        return 1.0/(1 + np.exp(-x))
    else:
        return np.exp(x)/(1 + np.exp(x))
```

并结合机器误差对计算方法进行优化，防止溢出。

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

$$P(Y = 0|X) = \frac{\exp(w^T X)}{1+\exp(w^T X)} = \frac{1}{1+\exp(-w^T X)} = \mathrm{sigmoid}(w^T X)$$

前面我们已经得到了分类的界限 $w^T X=0$,接下来我们应通过使得代价函数最小来求解 w.

有两种方法：最大似然估计 MLE 和贝叶斯估计 MAP，两者在 loss 函数里面就分别代表了无正则项的 loss 函数和有正则项的 loss 函数。

①MCLE(条件最大似然估计)

如果直接使用最大似然估计 MLE 计算 P(<X,Y>｜w)，非常困难，因此可以将 MLE 转换为 MCLE，计算 P(Y|X，w)。

似然函数为：

$$
\begin{aligned}
L(w) &= \ln \prod_{l} P\left(Y^{l} \mid X^{l}, w\right) \\
&= \sum_{l}\left(Y^{l} w^{T} X^{l}-\ln\left(1+\exp\left(w^{T} X^{l}\right)\right)\right)
\end{aligned}
$$

令似然函数最大，可以转化为求代价函数最小，Loss(w) = -L(w)：

$$
\operatorname{loss}(w)=\sum_{l}\left(-Y^{l} w^{T} X^{l}+\ln\left(1+\exp\left(w^{T} X^{l}\right)\right)\right)
$$

②MAP（贝叶斯估计）

MAP 的核心思想是：w 是一个随机变量，符合一定的概率分布。所以我们的任务就是给 w 添加一个先验 P(w),然后使得 P(w)P(Y｜X,w)最大。

我们假设 $w_i \sim N(0, \sigma)$ ，则似然函数为

$$
L(w)=\sum_{l}\left(Y^{l} w^{T} X^{l}-\ln\left(1+\exp\left(w^{T} X^{l}\right)\right)\right)-\frac{w^{T} w}{2 \sigma^{2}}+\ln\left(\frac{1}{\sqrt{2 \pi} \sigma}\right)
$$

忽略常量，可以简化为：

$$
L(w)=\sum_{l}\left(Y^{l} w^{T} X^{l}-\ln\left(1+\exp\left(w^{T} X^{l}\right)\right)\right)-\frac{\lambda}{2} w^{T} w
$$

Loss 函数将最大化问题化为最小化问题：

$$\mathrm{loss}(w) = \sum_{l}(-Y^{l}w^{T}X^{l} + \ln(1+\exp(w^{T}X^{l}))) + \frac{\lambda}{2}w^{T}w$$

相当于在 MCLE 的基础上加了正则项。

基于牛顿法通过代价函数求解 w：

由于梯度下降法、共轭梯度法在实验 1 中已经有过学习和应用，因此实验 2 在求解 w 的过程中使用牛顿法。

设 f(x)有二阶连续偏导数，若第 k 次迭代值为 x^k，则可以将 f(x)在 x^k 附近进行二阶泰勒展开:

$$f(x) = f(x^k) + g_k^T(x - x^k) + \frac{1}{2}(x - x^k)^T H(x^k)(x - x^k)$$

其中，$g^k = \nabla f(x)$是梯度向量，H(x)是海森矩阵。

$$H(x) = [\frac{\partial^2 f}{\partial x_i \partial x_j}]_{n\times n}$$

f(x)在极值点处 $g^k=0$；特别的，当 H(x)为正定矩阵时， f(x)的极值是极小值。

为了得到 $g^k=0$ 的点，对 f(x)式求导。

$$\frac{df(x)}{dx} = g_k + H_k(x - x^k)$$

极值点处导数为 0，有迭代公式：

$$x^{k+1} = x^k - H_k^{-1}g_k$$

将其应用到我们的 loss 函数有：

$$w^{k+1} = w^k - (\frac{\partial^2 \mathrm{loss}(w)}{\partial w \partial w^T})^{-1}\frac{\partial \mathrm{loss}(w)}{\partial w}$$

其中：

$$\frac{\partial \mathrm{loss(w)}}{\partial \mathrm{w}} = -\sum_{l} \mathrm{x}^{l}(\mathrm{Y}^{l} - \mathrm{sigmoid}(\mathrm{w}^{T}\mathrm{X})) + \lambda \mathrm{w}$$

$$\frac{\partial^{2} \mathrm{loss(w)}}{\partial \mathrm{w} \partial \mathrm{w}^{T}} = \sum_{l} (\mathrm{X}\mathrm{X}^{T}\mathrm{sigmoid}(\mathrm{w}^{T}\mathrm{X})\mathrm{sigmoid}(-\mathrm{w}^{T}\mathrm{X})) + \lambda \mathrm{I}$$

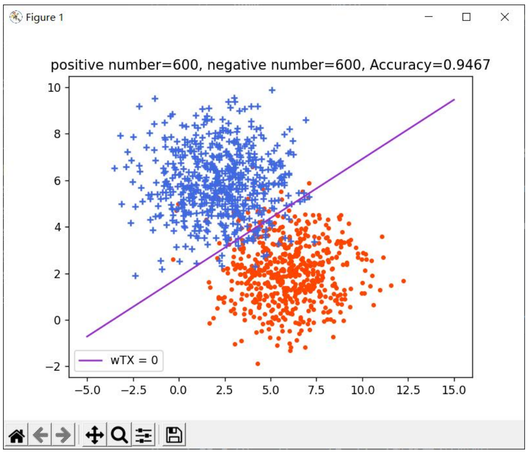
λ 为 0 是为 MCLE 的情况， λ 不为 0 为 MAP 的情况。

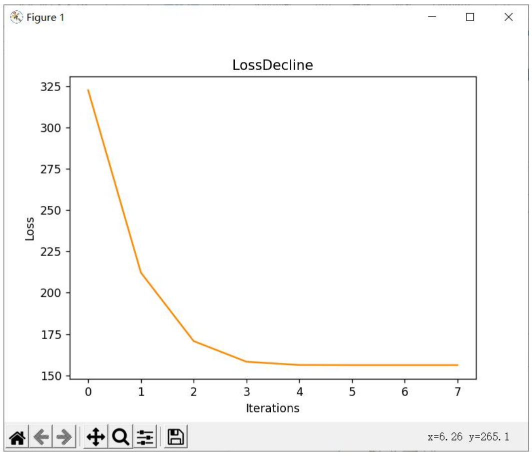
```
#求代价函数关于w向量的的一阶导数（结果为向量）,l为正则项系数
def derivative_1(trainset, w, l, dimension):
    result = np.zeros(dimension)
    #x = trainset[0] (不含常数项), Y = trainset[1]
    for i in range(len(trainset[0])):
        #X在x上加入常数项，方便与w相乘
        X = np.append(trainset[0][i], 1)
        result = result + X * (trainset[1][i] - sigmoid(np.dot(w, X)))
    return -result + l * w

#求代价函数关于w向量的的二阶导数（结果为矩阵）,l为正则项系数
def derivative_2(trainset, w, l, dimension):
    result = np.eye(dimension) * l

    for i in range(len(trainset[0])):
        X = np.append(trainset[0][i], 1)
        temp = sigmoid(np.dot(w, X))
        result = result + np.dot(X[:, None], X[None, :]) * temp * (1-temp)
    return result

def compute_loss(trainset, w, l, dimension):
    loss = 0
    for i in range(len(trainset[0])):
        X = np.append(trainset[0][i], 1)
        loss += -trainset[1][i] * np.dot(X, w) + np.log(1+np.exp(np.dot(X,
w)))
    return loss + l/2 * np.dot(w.T, w)

#牛顿迭代法求解w
def Newton(trainset, l, epsilon):
    #w的维数 = x（trainset[0]）的维数 + 1
    dimension = len(trainset[0][0]) + 1
    w = np.zeros(dimension)
    loss_lst = []
```

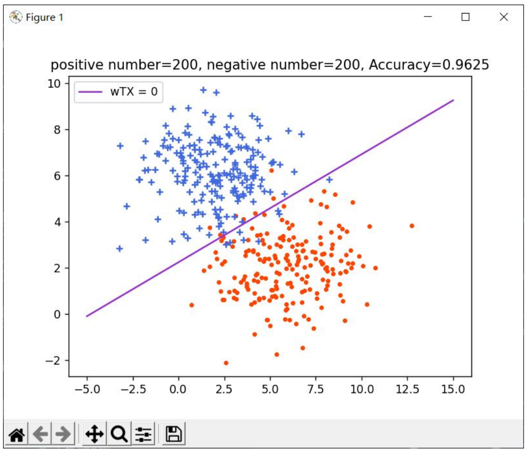
```python
    while True:
            gradient = derivative_1(trainset, w, l, dimension)
            # 满足精度要求(梯度的范数足够小时)，即可退出迭代
            if np.linalg.norm(gradient) <= epsilon:
                break
            # 使用迭代公式进行下一次迭代
            H_inv = np.linalg.inv(derivative_2(trainset, w, l, dimension))
            w = w - np.dot(H_inv, gradient)
            loss_lst.append(compute_loss(trainset, w, l, dimension))
    return [w, loss_lst]
```

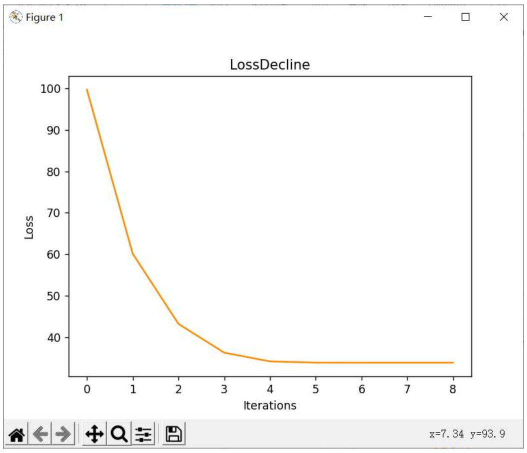
四、实验结果与分析

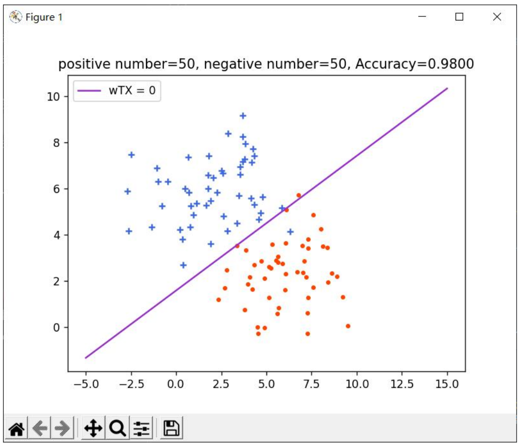
① 无正则项，满足朴素贝叶斯假设

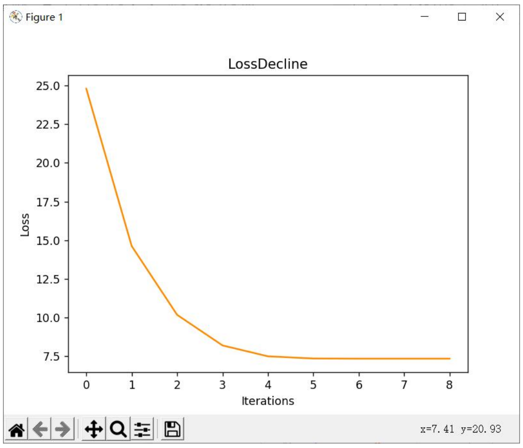


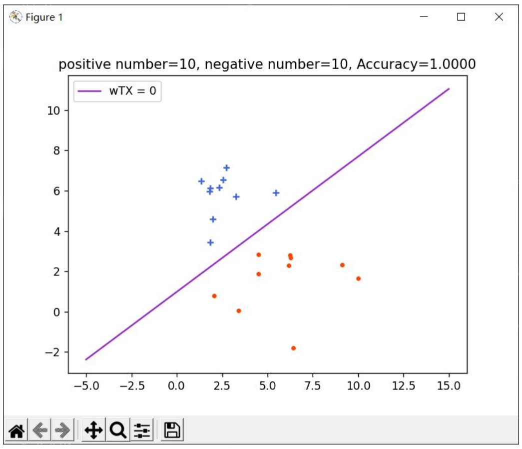


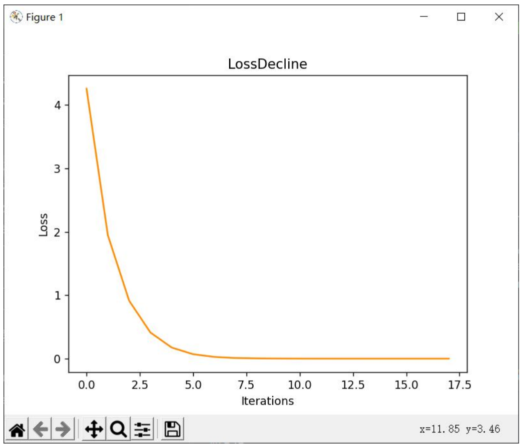






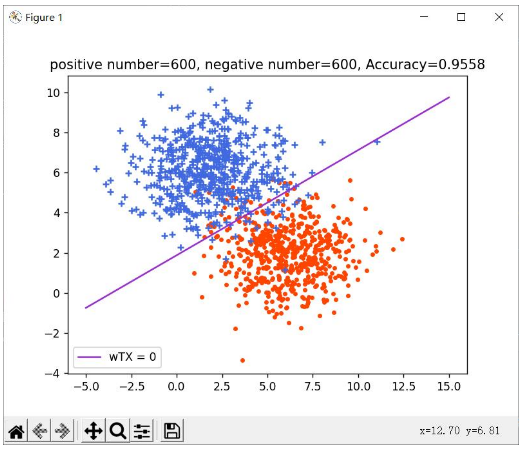


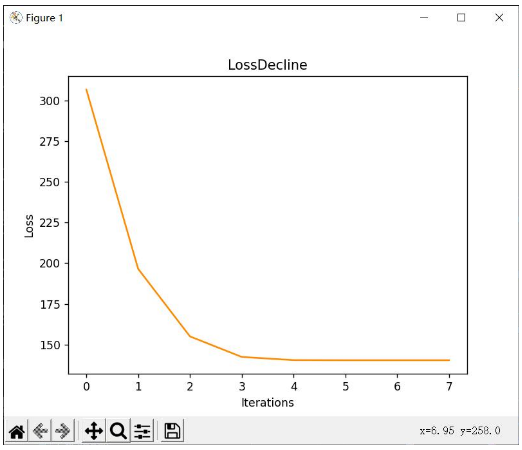


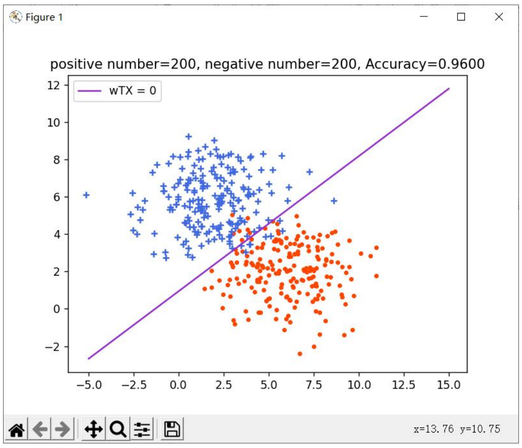


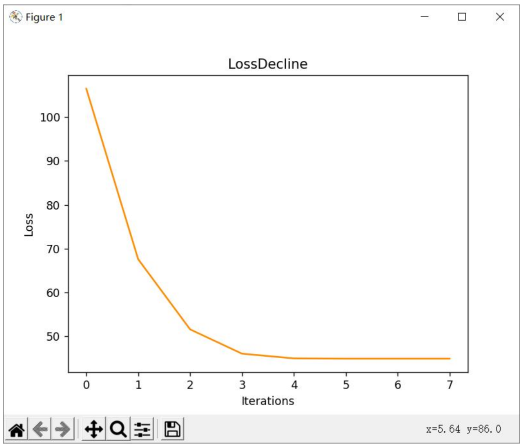
在训练集创建过程中，正例和反例数量基本一致。在无正则项，满足朴素贝叶斯假设的情况下，随着总样本数量从 1200、400、100、20 的变化过程当中，分类器变化不大，即使样本数量不大时的分类效果也相对较好，准确率均比较高。在损失函数的迭代次数上，样本量较小时迭代次数相对更多才能达到迭代终止条件，且牛顿迭代法迭代很快，迭代次数数量级很小。

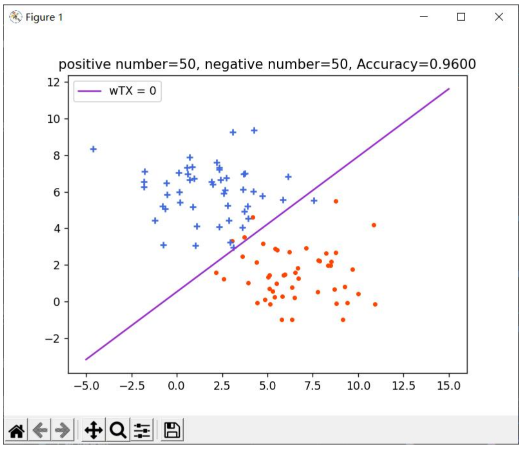
②正则项系数为 1，满足朴素贝叶斯假设

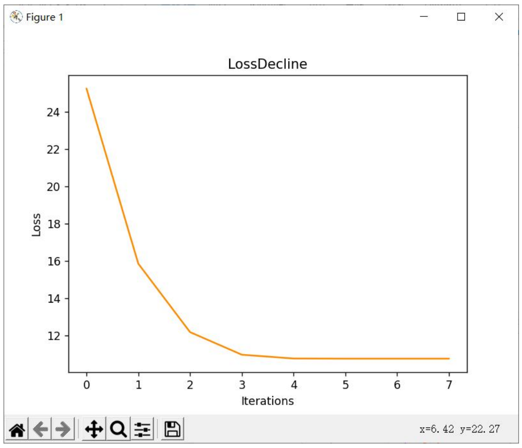


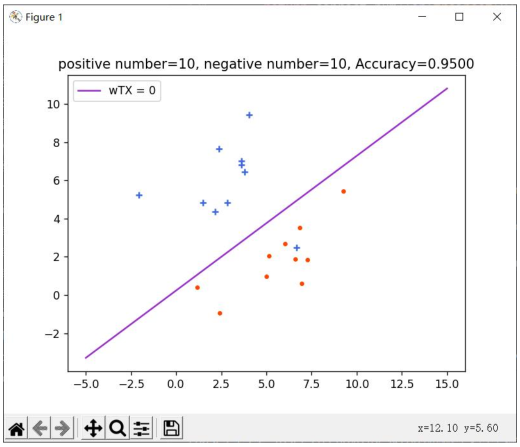


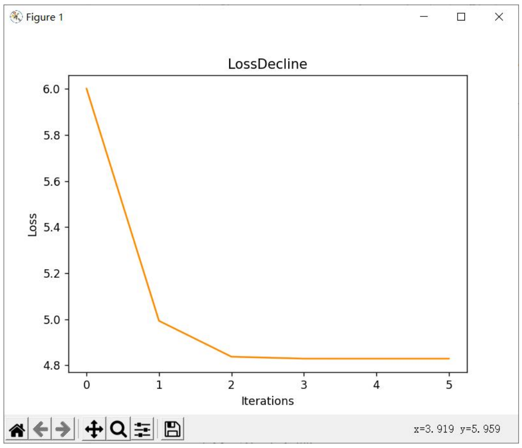






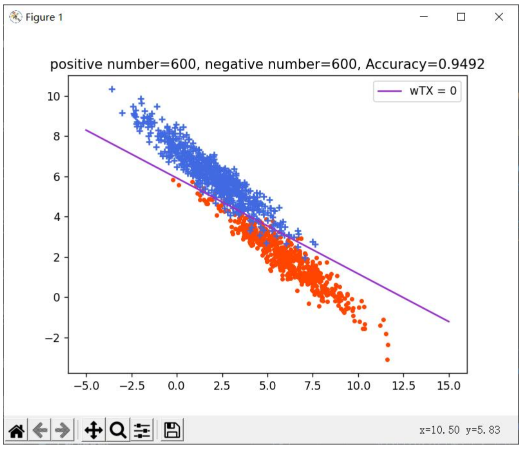


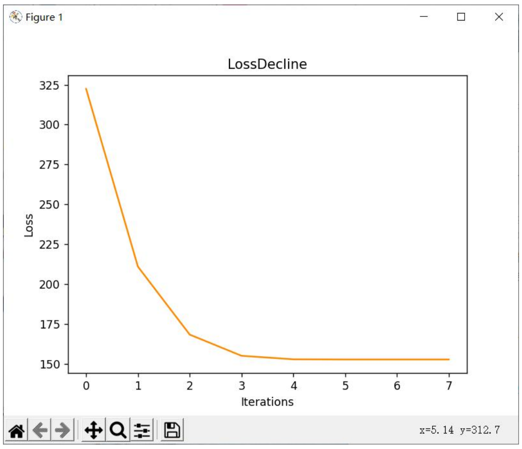


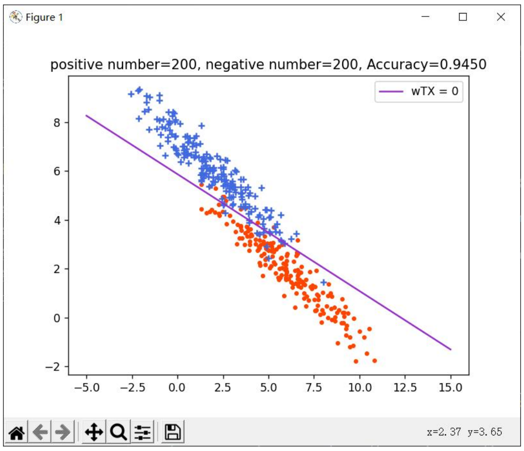


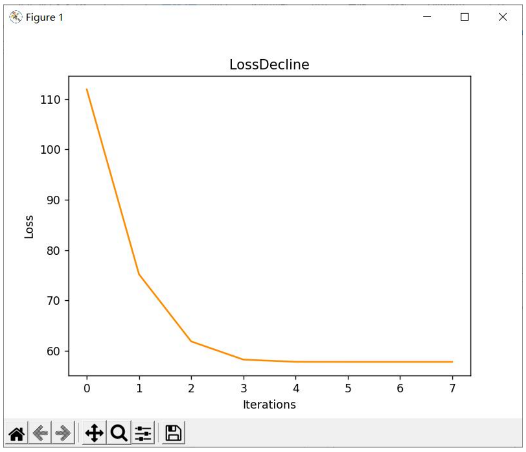
在正则项系数为 1，满足朴素贝叶斯假设的情况下，随着总样本数量从 1200、400、100、20 的变化过程当中，分类器变化不大，准确率均比较高，模型的复杂程度相对降低了（w 系数更接近 0）。在损失函数的迭代次数上，样本量较小时添加正则项可以减少迭代次数加速达到迭代终止条件。

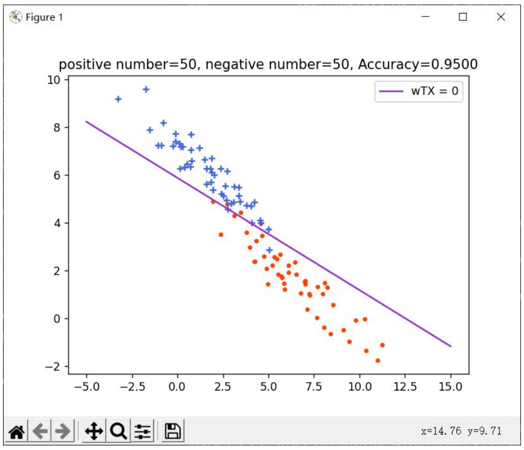
③无正则项，不满足朴素贝叶斯假设（协方差 Cov12 为-3）

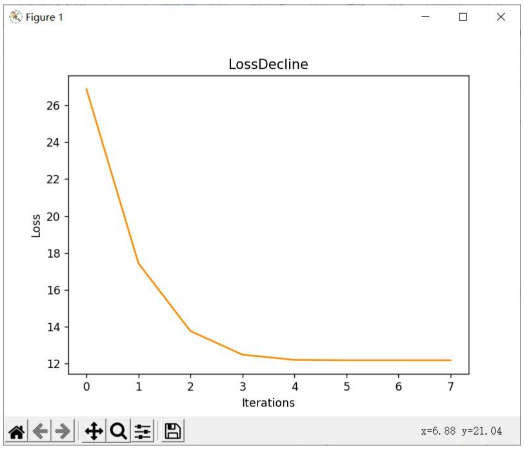


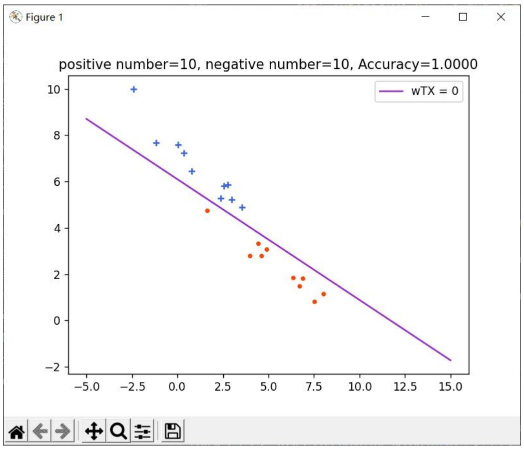


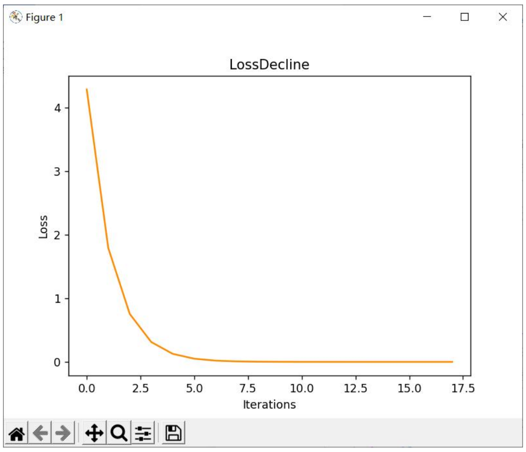






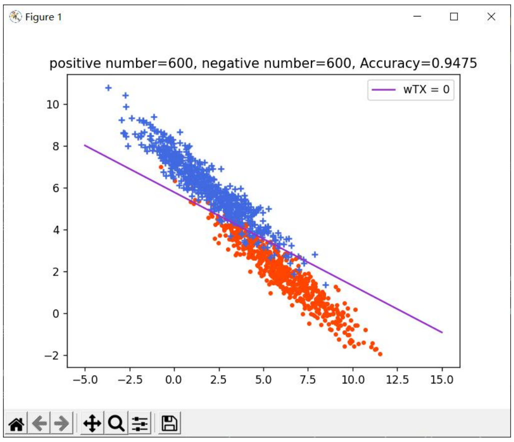


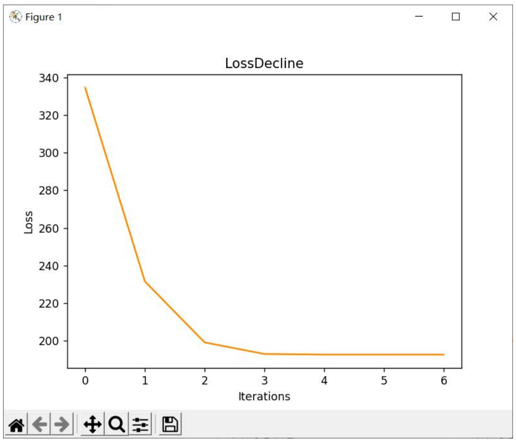


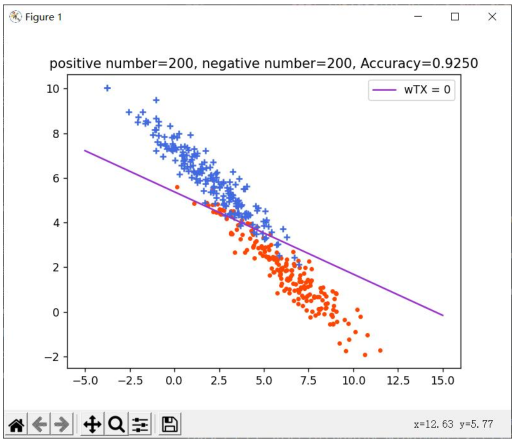


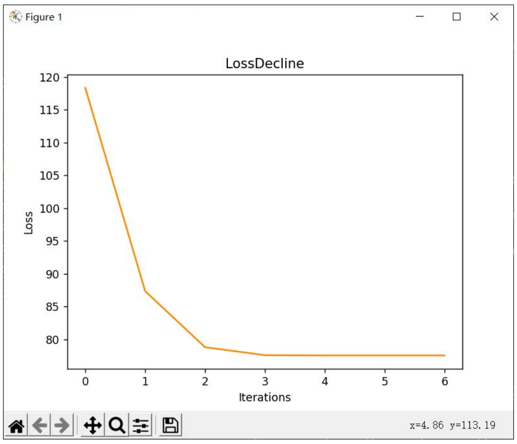
在无正则项，不满足朴素贝叶斯假设的情况下，分类器分类效果略有下降，但准确率依然较高。迭代次数与满足朴素贝叶斯假设的情况下差别不大。

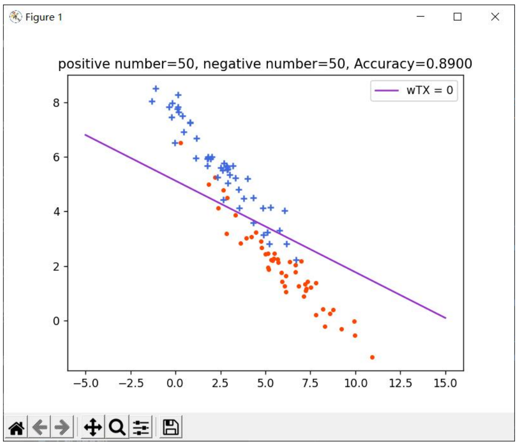
④正则项系数为 0.1，不满足朴素贝叶斯假设（协方差 Cov12 为-3）

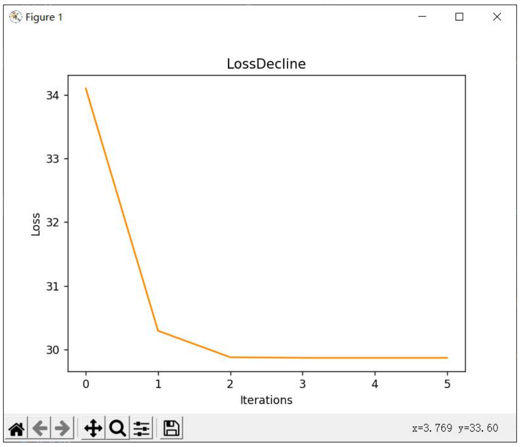


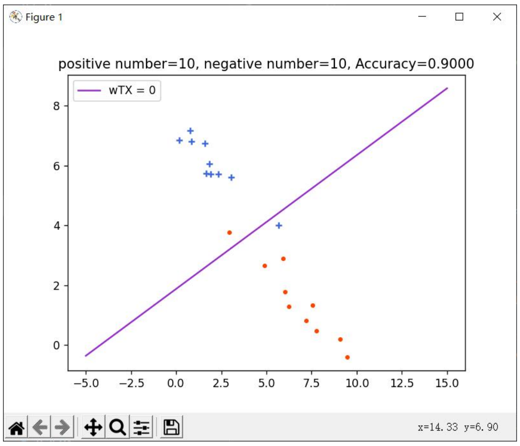


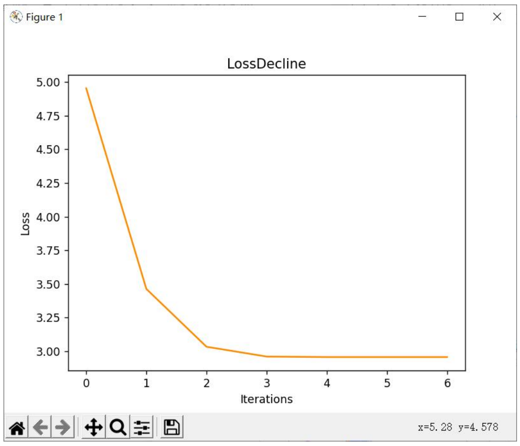






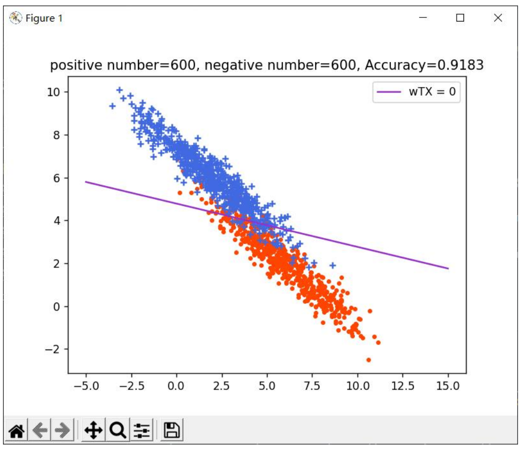


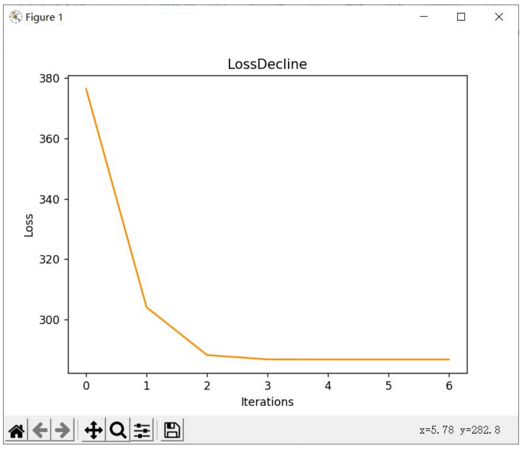


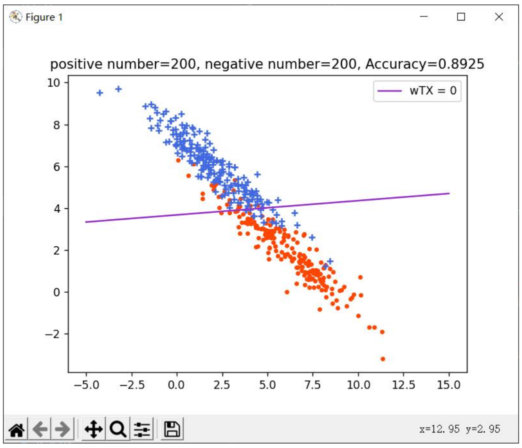


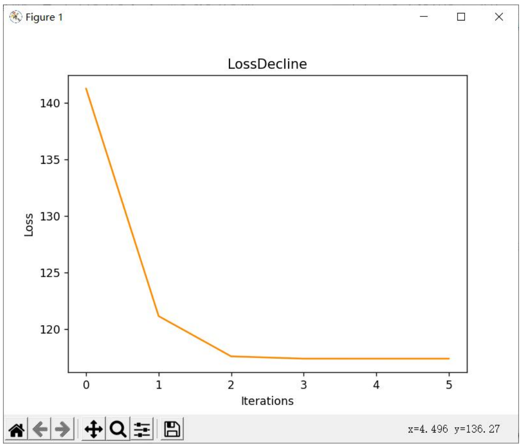
在正则项系数为 0.1，不满足朴素贝叶斯假设的情况下，分类器分类效果相比无正则项的情况略有下降，但准确率依然较高。在样本较小的情况下，模型迭代次数能够减小，但受到模型复杂程度降低的影响，和实际分布的分类偏差很大。

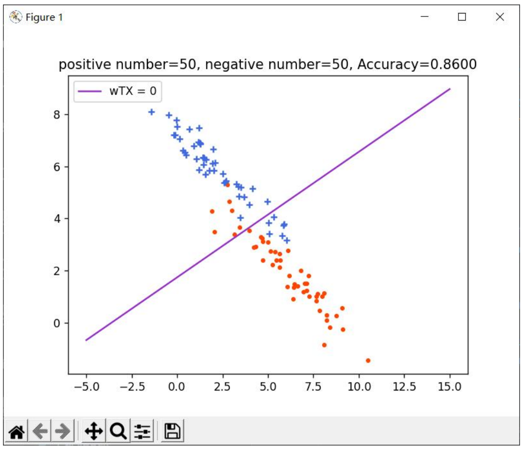
④正则项系数为 1，不满足朴素贝叶斯假设（协方差 Cov12 为-3）

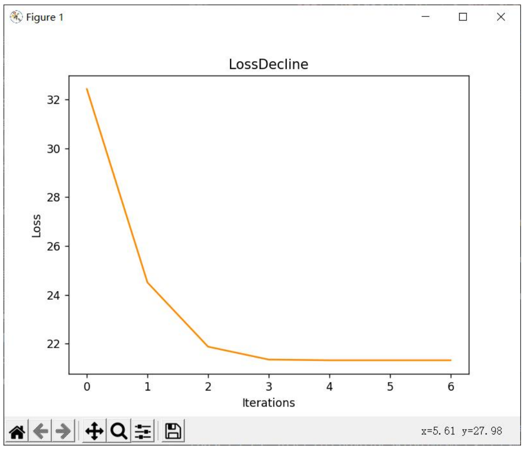


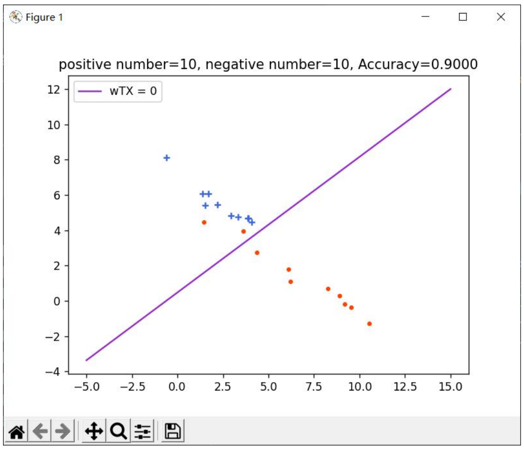


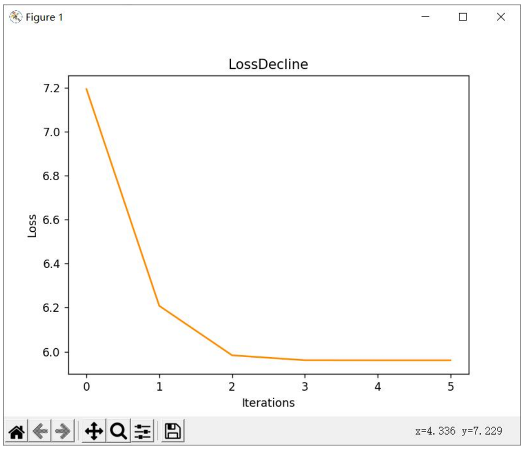












在正则项系数为 1，不满足朴素贝叶斯假设的情况下，分类器分类效果下降了一些，但准确率较高。添加较大的正则项，模型迭代次数能够减小，但受到模型复杂程度降低的影响，即便样本数据量很大也和实际分布的分类偏差很大，不应使用较大的正则项系数。

⑤ 对从 UCI 网站上下载的数据集做分类预测

数据集介绍：Somerville Happiness Survey Data Set

Abstract: A data extract of a non-federal dataset posted here

Data Set Characteristics:	N/A	**Number of Instances:**	143	**Area:**	Life
Attribute Characteristics:	Integer	**Number of Attributes:**	7	**Date Donated**	2018-05-24
Associated Tasks:	Classification	**Missing Values?**	N/A	**Number of Web Hits:**	35644

Source:
Waldemar W. Koczkodaj, wkoczkodaj@gmail, independent researcher.

Data Set Information:
It is a case of supervised learning with the use of Receiver Operating Characteristic (ROC) to select the minimal set of attributes preserving or increasing predictability of the data.

Attribute Information:
D = decision attribute (D) with values 0 (unhappy) and 1 (happy)
X1 = the availability of information about the city services
X2 = the cost of housing
X3 = the overall quality of public schools
X4 = your trust in the local police
X5 = the maintenance of streets and sidewalks
X6 = the availability of social community events
Attributes X1 to X6 have values 1 to 5.

部分数据展示：

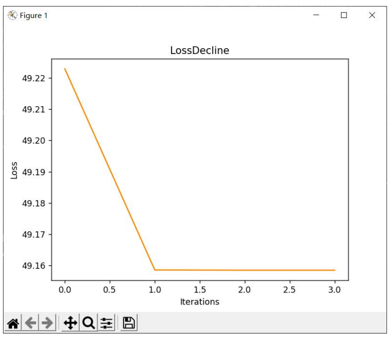
```
D,X1,X2,X3,X4,X5,X6
0,3,3,3,4,2,4
0,3,2,3,5,4,3
1,5,3,3,3,3,5
0,5,4,3,3,3,5
0,5,4,3,3,3,5
```

分类过程：

共 143 个样本，将前 80 个分成训练集，后 63 个分成测试集。对训练集经过与二维自制样本点相同方式、只是向量维数由 2 维上升到 5 维的分类预测之后，用测试集测试标签判断的准确率。

得到准确率：`accuracy = 0.5556`

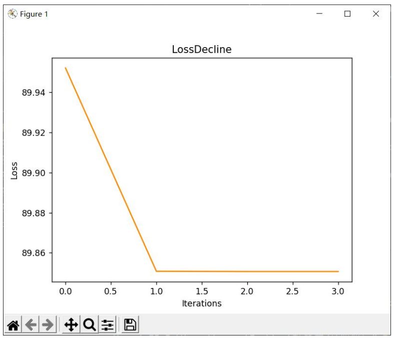
迭代收敛速度：



从中可以看出准确率较低。

即便用训练集做测试，准确率也不高：`accuracy = 0.6014`

迭代收敛速度：



这可能是因为：1.该数据集的研究设计存在结构性缺陷，以上 6 个变量无法概括性描述幸福感的决定因素；2.该数据集的实际概型场景不符合实验假设，各个变量不独立或不符合线性分类器生效的条件；3.该数据集样本量过小，容易出现过拟合，导致与实际的分布和分类器相差较大；等等原因。

五、结论

①类条件分布在满足朴素贝叶斯假设时的逻辑回归分类表现，要比不满足假设时略好，但不满足假设的数据条件下分类表现也不错，并可以通过旋转来满足朴素贝叶斯假设。

②对于在训练集样本数很少时，加入正则项可以有效解决过拟合问题，但过大的正则项可能会过于重视降低模型复杂程度而忽视逻辑回归分类表现。

③逻辑回归可以很好地解决简单的线性分类问题，但在样本数据量过小或者不满足朴素贝叶斯假设时分类效果很差。

④使用牛顿法求优化解时，收敛速度较快，只迭代几次就能满足对于梯度很小、接近极值点的要求，且样本点数目对它的收敛速度影响不大。

六、参考文献

[1] Snyman J A , Wilke D N . Practical Mathematical Optimization: Basic Optimization Theory and Gradient-Based Algorithms, Springer Optimization and Its Applications 133[M]. 2018.

[2] Guenin B , Knemann J , Tunel L . A Gentle Introduction to Optimization. 2018.

[3] 吴勃英. 数值分析原理(科学版)(研究生教学丛书)[M]. 科学出版社, 2006.

七、附录：源代码（带注释）

```
import numpy as np
import matplotlib.pyplot as plt

#准备：生成训练集，其中 covXY 为 0 时符合朴素贝叶斯假设
def get_trainset(pos_num, neg_num, covXY = 0):
    sigma = [4, 2]     #方差只与 X 向量的第几维有关，而与 Y 无关（不同标签的 sigma 都相等）
    pos_mean = [6, 2]     #正例 x,y 方向上的均值μ
    neg_mean = [2, 6]     #反例 x,y 方向上的均值μ
```

```
    x = np.zeros((pos_num + neg_num, 2))
    y = np.zeros(pos_num + neg_num)

    #在训练集数组中：正例在前，反(0)例在后
    x[:pos_num, :] = np.random.multivariate_normal(pos_mean, [[sigma[0], covXY],
                                                   [covXY, sigma[1]]], size=pos_num)
    x[pos_num:, :] = np.random.multivariate_normal(neg_mean, [[sigma[0], covXY],
                                                   [covXY, sigma[1]]], size=neg_num)
    y[:pos_num] = 1
    y[pos_num:] = 0
    #print (x)
    #print (y)
    return [x,y]
```

```
# 绘制训练集散点图
def draw_trainset2D(trainset, pos_num):
    for [x,y] in trainset[:pos_num]:
        plt.scatter(x, y, color = 'orangered', marker = '.')
    for [x,y] in trainset[pos_num:]:
        plt.scatter(x, y, color = 'royalblue', marker = '+')
```

```
def read_dataset(path):
    file = np.loadtxt(path, delimiter=',',
                         encoding = 'utf-16', dtype=int, skiprows=1)
    x = file[:,1:]
    y = file[:,0]
    #print (x)
    #print (y)
    return [x,y]
```

```
#求 sigmoid 函数
def sigmoid(x):
    if x >= 0:
        return 1.0/(1 + np.exp(-x))
    else:
        return np.exp(x)/(1 + np.exp(x))
```

```
#求代价函数关于 w 向量的的一阶导数（结果为向量）,l 为正则项系数
def derivative_1(trainset, w, l, dimension):
    result = np.zeros(dimension)
    #x = trainset[0] (不含常数项), Y = trainset[1]
    for i in range(len(trainset[0])):
        #X 在 x 上加入常数项，方便与 w 相乘
```

```python
        X = np.append(trainset[0][i], 1)
        result = result + X * (trainset[1][i] - sigmoid(np.dot(w, X)))

    return -result + l * w
```

```python
#求代价函数关于 w 向量的的二阶导数（结果为矩阵）,l 为正则项系数
def derivative_2(trainset, w, l, dimension):
    result = np.eye(dimension) * l

    for i in range(len(trainset[0])):
        X = np.append(trainset[0][i], 1)
        temp = sigmoid(np.dot(w, X))
        result = result + np.dot(X[:, None], X[None, :]) * temp * (1-temp)
        #print (np.dot(X[:, None], X[None, :]))
    return result
```

```python
def compute_loss(trainset, w, l, dimension):
    loss = 0
    for i in range(len(trainset[0])):
        X = np.append(trainset[0][i], 1)
        loss += -trainset[1][i] * np.dot(X, w) + np.log(1+np.exp(np.dot(X, w)))
    return loss + l/2 * np.dot(w.T, w)
```

```python
#牛顿迭代法求解 w
def Newton(trainset, l, epsilon):
    #w 的维数 = x（trainset[0]）的维数 + 1
    dimension = len(trainset[0][0]) + 1
    w = np.zeros(dimension)
    loss_lst = []
    while True:
            gradient = derivative_1(trainset, w, l, dimension)
            #print (gradient)
            # 满足精度要求(梯度的范数足够小时)，即可退出迭代
            if np.linalg.norm(gradient) <= epsilon:
                break
            # 使用迭代公式进行下一次迭代
            #print (derivative_2(trainset, w, l, dimension))
            H_inv = np.linalg.inv(derivative_2(trainset, w, l, dimension))
            w = w - np.dot(H_inv, gradient)
            #print(H_inv)
            loss_lst.append(compute_loss(trainset, w, l, dimension))
    return [w, loss_lst]
```

```python
def draw_classifier2D(w, pos_num, neg_num, Accuracy):
    x = np.linspace(-5,15,10)
    y = ( -w[2] - w[0] * x) / w[1]
    plt.plot(x, y, label = 'wTX = 0', color = 'darkorchid')
    plt.title('positive number=%d, negative number=%d, Accuracy=%.4f'
              %(pos_num, neg_num, Accuracy))
    plt.legend()
    plt.show()

def draw_loss(loss_lst):
    plt.plot(loss_lst, color = 'darkorange')
    plt.xlabel('Iterations')
    plt.ylabel('Loss')
    plt.title('LossDecline')
    plt.show()
```

```python
def get_accuracy(trainset, w):
    correct_num = 0
    for i in range(len(trainset[0])):
        X = np.append(trainset[0][i], 1)
        if (np.dot(X, w) > 0 and trainset[1][i] == 1):
            correct_num += 1
        elif (np.dot(X, w) < 0 and trainset[1][i] == 0):
            correct_num += 1
    return correct_num/len(trainset[0])
```

```python
if __name__ == '__main__':
    #self_made_data 变量指示：训练时使用自己编造的数据 or 从 UCI 下载的数据
    self_made_data = False

    #随机生成二维高斯分布的点作为数据集
    if self_made_data == True:
        pos_num = 600
        neg_num = 600
        trainset = get_trainset(pos_num, neg_num, covXY = -3)
        #trainset[0]是 X 部分，[1]是 Y 部分
        draw_trainset2D(trainset[0], pos_num)
        w, loss_lst = Newton(trainset, 0.1, 0.000001)
        draw_classifier2D(w, pos_num, neg_num, get_accuracy(trainset, w))
        draw_loss(loss_lst)
```

```python
    #下载 UCI 的幸福感测量作为数据集
    if self_made_data == False:
        data_set = read_dataset('实验 2\SomervilleHappinessSurvey2015.csv')
```

```python
split = 80
#前 split 组数据作为训练集
train_set = [data_set[0][:split], data_set[1][:split]]
#后(143-split)组数据作为测试集
test_set = [data_set[0][split:], data_set[1][split:]]
w, loss_lst = Newton(data_set, 0, 0.000001)
print ('accuracy = %.4f' %get_accuracy(data_set, w))
draw_loss(loss_lst)
```