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：选修

实验题目： 逻辑回归

学号：1173710204

姓名： 陈东鑫

# 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。

## 实验环境：

操作系统：Windows10
语言：python3
编程环境：jupyter notebook

# 三、设计思想（本程序中的用到的主要算法及数据结构）

## 1．算法原理

可相当于是使用 sigmod 函数去拟合数据集的 Y=f（X）（X 为样本点，f 是一个 0-1 分段函数，表示每个样本点的类别）。sigmod 函数表明对应样本点在参数 W 下归类为 1 的概率。其中 M 是样本点个数，N 为特征数

$$\text{sigmod} = \frac{1}{1+e^{-z}}$$

其中,x 的小标代表第 i 个特征，上标代表第 j 个样本

$$\text{z} = \begin{bmatrix} w_0 + w_1x_1^{(1)} + w_2x_2^{(1)} + \cdots + w_nx_n^{(1)} \\ w_0 + w_1x_1^{(j)} + w_2x_2^{(j)} + \cdots + w_nx_n^{(j)} \\ w_0 + w_1x_1^{(m)} + w_2x_2^{(m)} + \cdots + w_nx_n^{(m)} \end{bmatrix}^T$$

其矩阵表示为

$$\text{z} = W^TX$$

同时考虑其凹凸性，在 loss 函数上加上符号，故其 loss 函数应为

$$\text{loss} = -[sigmod^Y + (1-sigmod)^{(1-Y)}]$$

为了计算方便，对右式的每个加数取对数

$$\text{loss} = \frac{1}{M}\text{sum}(-[\text{Y}\ln sigmod + (1-Y)\ln(1-sigmod)])$$

作为这次学习的 loss 函数。
再计算其梯度

$$\text{grad} = \frac{1}{M}(\text{sigmod} - \text{Y}) \cdot \text{X}$$

在确定其学习率 lr 之后，即可进行梯度下降算法更新参数向量

$$\text{W} = \text{W} - \text{lr} \times \text{grad}$$

接下来关注共轭梯度法的算法原理：
共轭梯度法的原型函数为这样的一个线性方程组：

$$\mathrm{Ax} = \mathrm{b}$$

在这一次实验中，$X^T X = A$，$\mathrm{W} = \mathrm{x}$，故可得到 b。但此时的 b 并不是 Y，两者之间存在一个等式关系，该等式为：

$$\frac{1}{1+e^{-b}} = Y$$

为了做下一步的处理，先对 Y 进行一个近似处理，近似系数为 like（$0.5 < \mathrm{like} < 1$），ones 为形如 Y 的一个全 0 的矩阵：

$$\bar{Y} = \mathrm{like} \cdot \mathrm{Y} + (1 - \mathrm{like}) \cdot (\mathrm{ones} - \mathrm{Y})$$

由此可得：

$$\mathrm{b} = -\ln(\frac{ones}{\bar{Y}} - ones)$$

故共轭梯度法所使用的对象为：

$$X^T X \cdot \mathrm{W} = -\ln(\frac{ones}{\mathrm{like} \cdot \mathrm{Y} + (1 - \mathrm{like}) \cdot (\mathrm{ones} - \mathrm{Y})} - ones)$$

## 2. 算法的实现

### 1）生成数据

使用 np.random.multivariate_normal(mean, cov, size)函数生成二维高斯分布数据集。mean 代表均值，cov 代表协方差矩阵，size 代表样本点个数。其中包含两个二维高斯分布的数据，训练集中，其中一个分布的均值为[0.25, 0.25]，协方差矩阵为[[0.017, 0], [0, 0.017]]，样本点个数为 500。另一个分布的均值为[0.75, 0.75]，协方差矩阵为[[0.017, 0], [0, 0.017]]，样本点个数为 500。
协方差矩阵为对角阵即满足朴素贝叶斯假设。
测试集也遵循一样的分布，将所得到的数据进行归一化处理。得到样本点的分布如下图所示：

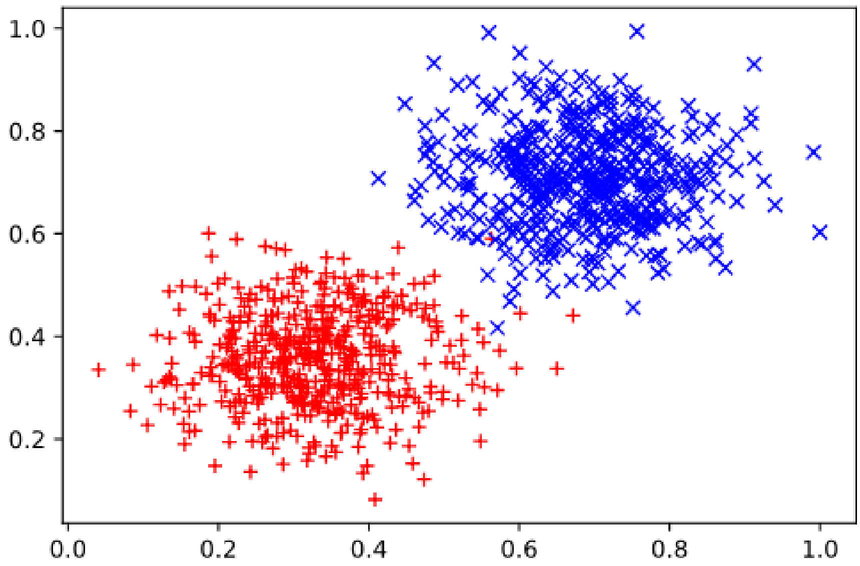



图 1 训练数据集

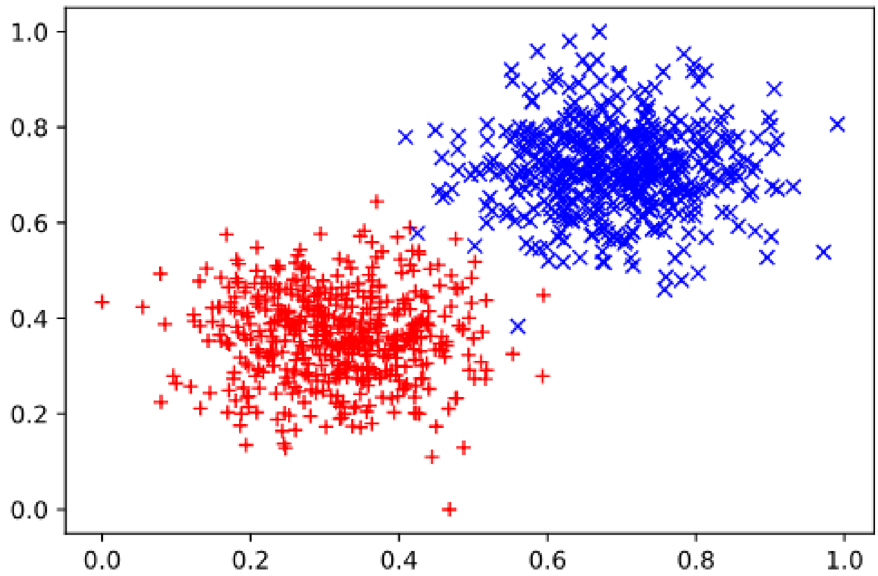



图 2 测试数据集

生成数据所需函数的 python 代码如下

```
# 生成数据函数
# mean 平均值，cov 协方差矩阵，M 样本点个数，为偶数
# 返回值的形状解释：
# data.shape = (3, M)
# X.shape = (2, M)
# Y.shape = (1, M)
# data1.shape = (3, M/2)
# data2.shape = (3, M/2)
def genData(mean1, cov1, mean2, cov2, M):
    # 训练集
```

```
    X1 = np.random.multivariate_normal(mean1, cov1, int(M / 2)).T
    X2 = np.random.multivariate_normal(mean2, cov2, int(M / 2)).T
    Y1 = np.zeros_like(X1[0]).reshape((1, int(M / 2)))
    Y2 = np.ones_like(X2[0]).reshape((1, int(M / 2)))
    X = np.c_[X1, X2]
    Y = np.c_[Y1, Y2]

    # 测试集
    XT1 = np.random.multivariate_normal(mean1, cov1, int(M / 2)).T
    XT2 = np.random.multivariate_normal(mean2, cov2, int(M / 2)).T
    YT1 = np.zeros_like(XT1[0]).reshape((1, int(M / 2)))
    YT2 = np.ones_like(XT2[0]).reshape((1, int(M / 2)))
    XT = np.c_[XT1, XT2]
    YT = np.c_[YT1, YT2]

    # 归一化
    theMax = np.max(np.c_[X, XT], axis=1).reshape((2, 1))
    theMin = np.min(np.c_[X, XT], axis=1).reshape((2, 1))
    X = X - theMin
    XT = XT - theMin
    X = X / (theMax - theMin)
    XT = XT / (theMax - theMin)

    # 训练集
    data1 = np.r_[X[:, 0:int(M / 2)], Y1]
    data2 = np.r_[X[:, int(M / 2):], Y2]
    data = np.c_[data1, data2]

    # 测试集
    dataT1 = np.r_[XT[:, 0:int(M / 2)], YT1]
    dataT2 = np.r_[XT[:, int(M / 2):], YT2]
    dataT = np.c_[dataT1, dataT2]

    return data, X, Y, data1, data2, dataT, XT, YT, dataT1, dataT2
```

生成数据的代码如下：

```
# 生成数据
mean1 = [0.25, 0.25]
cov1 = 0.017 * np.eye(2)

# 不满足朴素贝叶斯假设
# cov1[0][1] = cov1[0][0] * (-0.9)
# cov1[1][0] = cov1[0][1]
```

```
mean2 = [0.75, 0.75]
cov2 = cov1
M = 1000
data, X, Y, data1, data2, dataT, XT, YT, dataT1, dataT2 = genData(
    mean1, cov1, mean2, cov2, M)
```

## 2）训练模型

由算法原理处的论述，算法实现所需的函数代码如下

```
# 初始化 W 向量
# W.shape = (N, 1)
def param(N):
    if N < 3 or N % 2 != 1:
        print('the invaild N')
        return
    W = np.ones((N, 1))
    return W


# 初始化用于计算的 X（特征数的不同会有不同的 X）
# X.shape = (N, M)
def initX(X, N, M):
    temp = X
    n = int((N - 3) / 2)
    for i in range(n):
        X = np.r_[X, temp**(i + 2)]
    X = np.r_[np.ones((1, M)), X]
    return X


# 多项式函数
# W 是参数，X 是样本，N 是特征数与 W 中元素个数相等
# N 应为大于等于 3 的奇数
# res.shape = (1, M)
def H(W, X):
    res = np.dot(W.T, X)
    return res


# 加正则项的多项式函数
def RegH(W, X, lamda, M):
    reg = lamda / (2 * M) * np.dot(W.T, W)
```

```
    res = np.dot(W.T, X) + reg * np.ones((1, M))
    return res


# sigmod 函数
# sigmod.shape = (1, M)
def G(z):
    sigmod = np.exp(-1 * z) + np.ones_like(z)
    sigmod = 1 / sigmod
    return sigmod


# loss 函数
# type(sum) = <class 'numpy.float64'>
def loss(W, X, Y, M):
    sigmod = G(H(W, X))
    sum1 = np.log(sigmod) * Y
    sum2 = np.log(np.ones_like(sigmod) - sigmod) * (np.ones_like(Y) - Y)
    temp = sum1 + sum2
    sum = np.sum(temp)
    sum = -1 * sum
    sum = sum / M

    return sum


# 梯度
def grad(X_, Y, sigmod, M):
    dz = sigmod - Y
    grad = np.dot(X_, dz.T)
    grad = grad / M
    return grad


# 加正则项的梯度
def RegGrad(X_, Y, sigmod, M, lamda, W):
    dz = sigmod - Y
    grad = np.dot(X_, dz.T) + lamda * W
    grad = grad / M
    return grad


#共轭梯度法
def conGrad(X, Y, W, N, like):
```

```python
    # 对 Y 做一个近似以满足共轭梯度法的要求
    Y = like * Y + (1 - like) * (np.ones_like(Y) - Y)
    Y = -1 * np.log(1 / Y - np.ones_like(Y))

    b = np.dot(X, Y.T)
    X = np.dot(X, X.T)

    b = b.reshape((b.size, 1))
    x = W
    p = b - np.dot(X.T, W)
    r = p

    for k in range(N):
        if (r == np.zeros_like(r)).all():
            break
        alpha = ((np.dot(r.T, r)) / np.dot(np.dot(X.T, p).T, p))[0][0]
        x = x + alpha * p
        temp = r
        r = r - alpha * np.dot(X.T, p)
        belta = np.dot(r.T, r)[0][0] / np.dot(temp.T, temp)[0][0]
        p = r + belta * p
    return x
```

其中，

param(N)：用来初始化 W 向量（列向量），N 是 W 向量的行数，也代表特征数，要求 N 是大于等于 3 的奇数。

initX(X, N, M)：初始化用于计算的 X，根据 N 特征数的不同，生成不同的 X 矩阵，（比如当 N 为 5 时，则在 X 原有的（$1, x_1, x_2$）基础上加入$x_1^2, x_2^2$项）。最终返回矩阵 X_，是一个$\mathrm{N} \times \mathrm{M}$的矩阵

H(W, X)：多项式函数，返回$W^T X$（$1 \times \mathrm{M}$的矩阵），作为 sigmod 函数的输入，是 sigmod 函数中 e 的负指数。

RegH(W, X, lamda, M)：加入了 L2 正则项的多项式函数，返回$W^T X + \frac{1}{2M} W^T W$（$1 \times \mathrm{M}$的矩阵）。

G(z)：sigmod 函数，其中 z 为 H()或 regH()的返回值（取决于选用的方法）。计算$\mathrm{sigmod} = \frac{1}{1+e^{-z}}$并返回 sigmod（$1 \times \mathrm{M}$的矩阵）。

loss(W, X, Y, M)：计算$\mathrm{loss} = \frac{1}{M} \mathrm{sum}(-[\mathrm{Y} \ln sigmod + (1 - Y) \ln(1 - sigmod)])$并返回

grad(X_, Y, sigmod, M)：计算$\mathrm{grad} = \frac{1}{M} (\mathrm{sigmod} - \mathrm{Y}) \cdot \mathrm{X}$并返回之。

RegGrad(X_, Y, sigmod, M, lamda, W)：计算$\mathrm{grad} = \frac{1}{M} (\mathrm{sigmod} - \mathrm{Y}) \cdot \mathrm{X} + \mathrm{lamda} \cdot \mathrm{W}$并返

回之

conGrad(X, Y, W, N, like)：进行共轭梯度法的迭代，返回结果值为所求的参数。。

然后通过这样的函数调用实现梯度下降法，带正则项的梯度下降法同理：

```
# init
N = 11
W0 = param(N)
X_ = initX(X, N, M)
XT_ = initX(XT, N, M)

# epoch
epoch = 800
# learning rate
lr = 0.05
newLoss = loss(W0, X_, Y, M)
oldLoss = newLoss
sigmod = G(H(W0, X_))

for item in range(epoch):
    W0 = W0 - lr * grad(X_, Y, sigmod, M)
    oldLoss = newLoss
    newLoss = loss(W0, X_, Y, M)
    t = oldLoss - newLoss
    if t < 0:
        lr /= 1.5
```

共轭梯度法：

```
like = 0.999
N = 11
W2 = param(N)
X_ = initX(X, N, M)
XT_ = initX(XT, N, M)
W2 = conGrad(X_, Y, W2, N, like)
```

为了读取 UCI 数据，生成数据函数重新写了一个，可从 txt 文件中读取数据，生成 80%训练集和 20%测试集：

```
def readFile(filename):
    f = open(filename, 'r', encoding='utf-8')
    return f.readlines()


# M 为训练集样本点个数
```

```python
# O 为维度数
def genDataFromFile(filename):
    context = readFile(filename)

    for i in range(len(context)):
        context[i] = context[i].replace('\n', '')
        context[i] = context[i].replace(' ', '')
        context[i] = (context[i]).split(',')
    O = len(context[0]) - 1
    M = len(context)

    for i in range(M):
        for j in range(O + 1):
            if j != O:
                context[i][j] = float(context[i][j])
            else:
                context[i][j] = int(context[i][j])

    data = np.array(context).reshape((M, O + 1))
    data = np.random.permutation(data)

    # 归一化
    dataMax = np.max(data, axis=0)
    dataMin = np.min(data, axis=0)
    temp = dataMax - dataMin
    data = data - dataMin
    data = data / temp

    # data, dataT
    M_ = M
    M = int(M * 0.8)
    MT = M_ - M
    print("M =", M)
    print("MT =", MT)
    dataT = data.copy()[M:, :]
    data = data.copy()[0:M, :]

    # data1, data2
    flag1 = False
    flag2 = False
    for i in range(M):
        if data[i][O] == 0:
            if flag1:
                data1 = np.c_[data1, data[:][i].reshape((O + 1, 1))]
```

```
            else:
                data1 = data[:][i].reshape((O + 1, 1))
                flag1 = True
        if data[i][O] == 1:
            if flag2:
                data2 = np.c_[data2, data[:][i].reshape((O + 1, 1))]
            else:
                data2 = data[:][i].reshape((O + 1, 1))
                flag2 = True

    # dataT1, dataT2
    flag1 = False
    flag2 = False
    for i in range(MT):
        if dataT[i][O] == 0:
            if flag1:
                dataT1 = np.c_[dataT1, dataT[:][i].reshape((O + 1, 1))]
            else:
                dataT1 = dataT[:][i].reshape((O + 1, 1))
                flag1 = True
        if dataT[i][O] == 1:
            if flag2:
                dataT2 = np.c_[dataT2, dataT[:][i].reshape((O + 1, 1))]
            else:
                dataT2 = dataT[:][i].reshape((O + 1, 1))
                flag2 = True
    data = data.T
    dataT=dataT.T

    # X, Y, XT, YT
    X = data[0:O][:].copy()
    Y = data[O][:].copy().reshape((1, M))
    XT = dataT[0:O][:].copy()
    YT = dataT[O][:].copy().reshape((1, MT))

    return M, O, data, data1, data2, X, Y, MT, dataT, dataT1, dataT2, XT, YT
```

# 四、实验结果与分析

实验使用的参数为，初始 W 为全 1 的列向量，N 为 11 即特征数为 11，M 为 1000 即有 1000 个样本点。

当数据分布满足朴素贝叶斯假设时，结果如下:

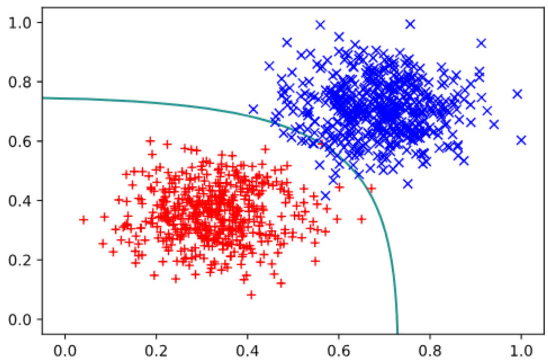
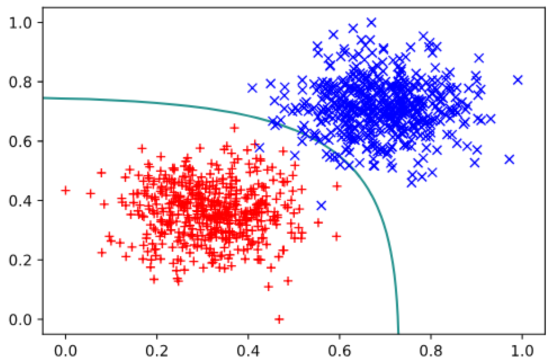
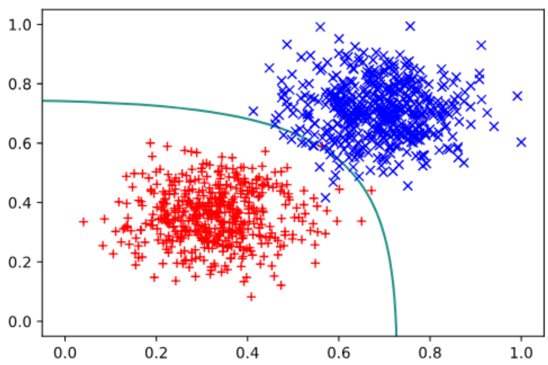
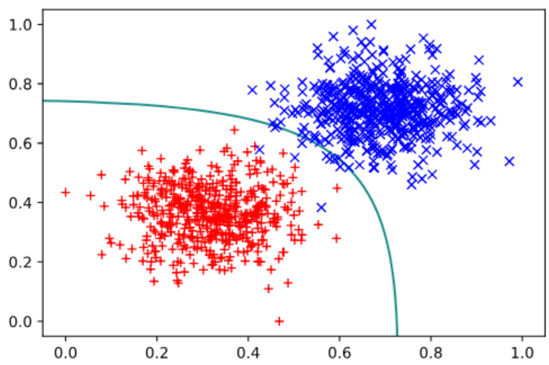
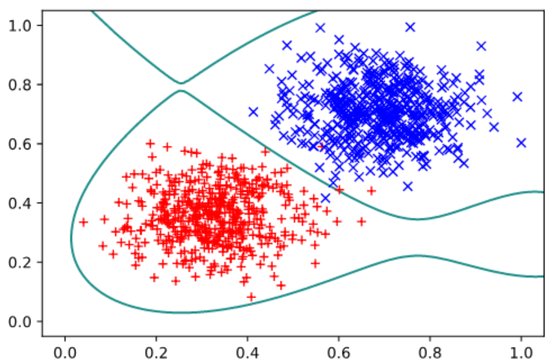
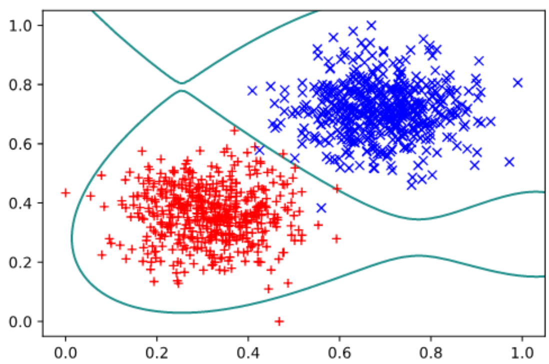
| | 训练 | 测试 | 测试集上的正确率 |
|---|---|---|---|
| 梯度下降法 |  |  | 98.90% |
| 带 L2 正则项的梯度下降法 |  |  | 99.10% |
| 共轭梯度法 |  |  | 99.40% |

表 1 数据分布满足朴素贝叶斯假设时结果一览

当数据分布不满足朴素贝叶斯假设时，，由于本次实验生成的两个分布之间相对位置的关系，当协方差矩阵控制分布的两个维度之间为正相关时，训练集上正确率变差，当协方差矩阵控制分布的两个维度之间为负相关时，训练集上正确率变高。从样本点在图上的分布来看，产生这一结果的原因在于，正相关时两个分类混杂的区域变大，该区域内的样本点无法在低维空间做到很好的划分，负相关时则正好相反。

正相关时，协方差矩阵为[[0.017, 0.0068], [0.0068, 0.017 ]]结果如下：

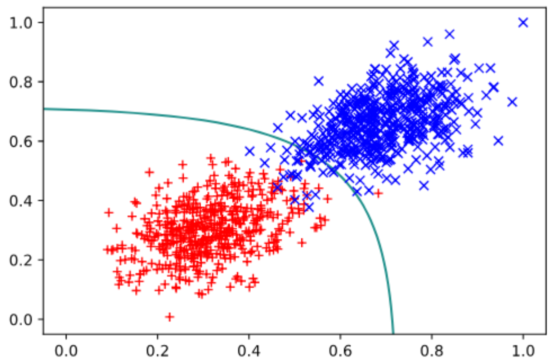
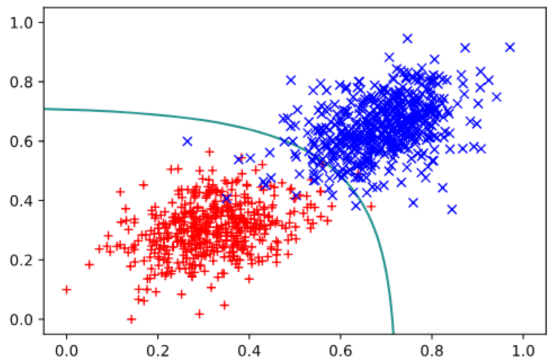
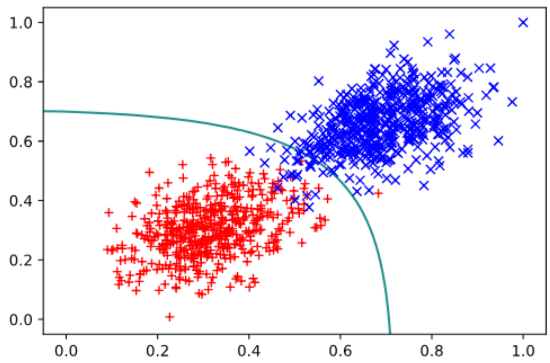
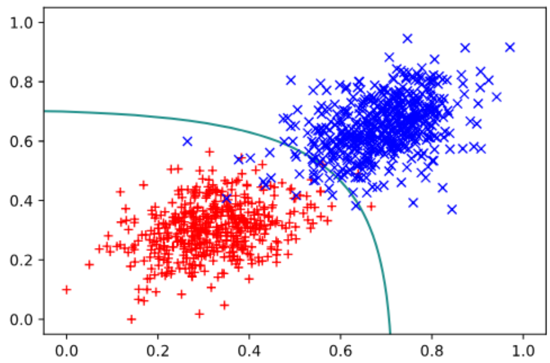
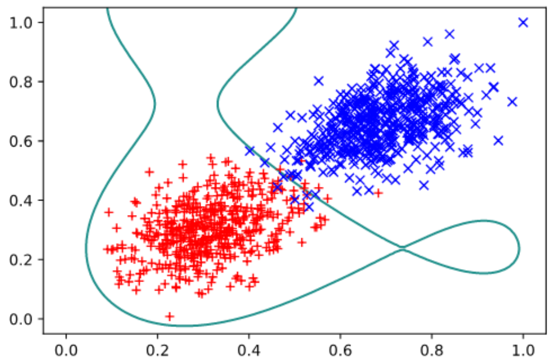
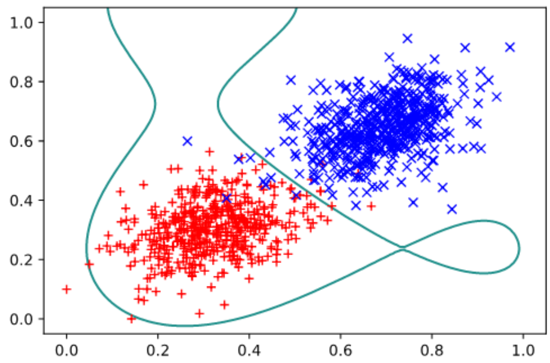
| | 训练 | 测试 | 测试集上的正确率 |
|---|---|---|---|
| 梯度下降法 |  |  | 96.70% |
| 带 L2 正则项的梯度下降法 |  |  | 97.30% |
| 共轭梯度法 |  |  | 98.30% |

表 2 正相关的协方差矩阵结果一览

负相关时，协方差矩阵为[[ 0.017, -0.0068], [-0.0068, 0.017 ]]，结果如下：

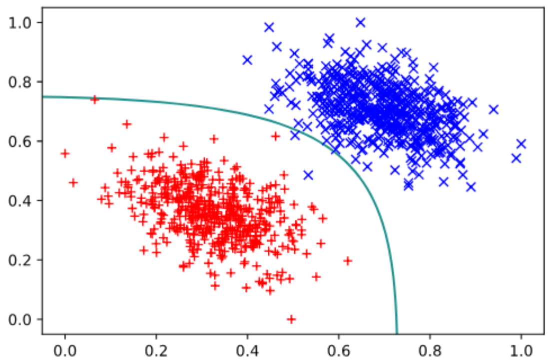
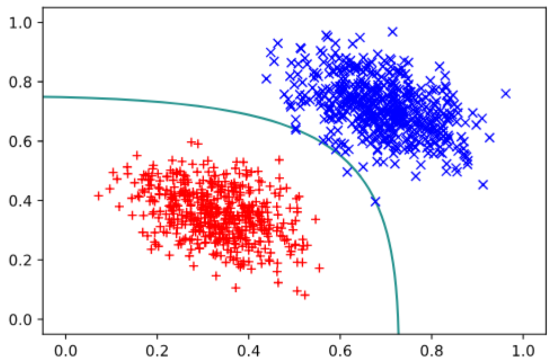
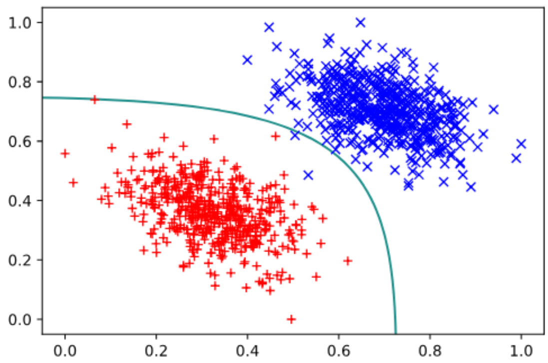
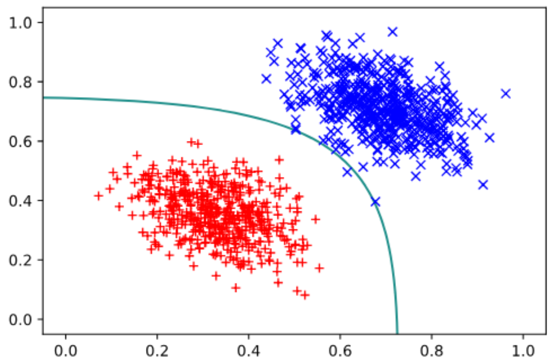
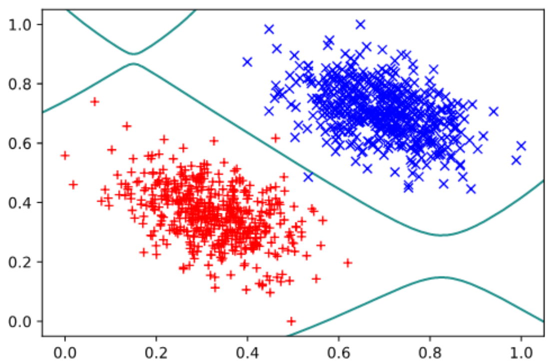
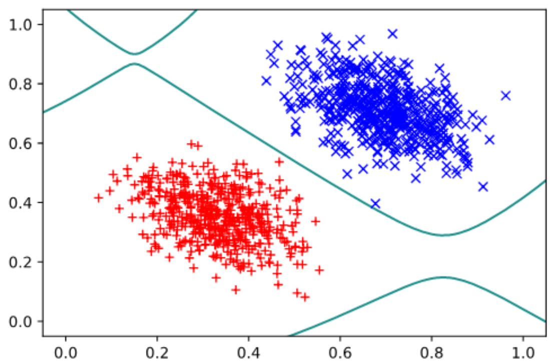
| | 训练 | 测试 | 测试集上的正确率 |
|---|---|---|---|
| 梯度下降法 |  |  | 99.70% |
| 带 L2 正则项的梯度下降法 |  |  | 99.90% |
| 共轭梯度法 |  |  | 100.00% |

表 3 负相关的协方差矩阵结果一览

可见，共轭梯度法生成的决策面是过拟合

使用在 UCI 上得到的数据集 Blood Transfusion Service Center Data Set 运行算法，当 N=25 时，得到的结果如下：

| 算法 | 正确率 |
|---|---|
| 梯度下降法 | 78.67% |
| 带 L2 正则项的梯度下降法 | 78.67% |
| 共轭梯度法 | 81.33% |

# 五、结论

1）正确率

梯度下降法 < 带 L2 正则项的梯度下降法 < 共轭梯度法

2）关于是否满足朴素贝叶斯假设的讨论。

前述：当数据分布不满足朴素贝叶斯假设时，，由于本次实验生成的两个分布之间相对位置的关系，当协方差矩阵控制分布的两个维度之间为正相关时，训练集上正确率变差，当协方差矩阵控制分布的两个维度之间为负相关时，训练集上正确率变高。从样本点在图上的分布来看，产生这一结果的原因，在数据分布图上看来，正相关时两个分类混杂的区域变大，该区域内的样本点无法在低维空间做到很好的划分，负相关时则正好相反。

由此可知，是否满足朴素贝叶斯假设对结果会产生一定影响，具体影响的方向要视实际样本的分布情况而定

# 六、参考文献

# 七、附录：源代码（带注释）

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

# 生成数据函数
# mean 平均值，cov 协方差矩阵，M 样本点个数，为偶数
# 返回值的形状解释：
# data.shape = (3, M)
# X.shape = (2, M)
# Y.shape = (1, M)
# data1.shape = (3, M/2)
# data2.shape = (3, M/2)
def genData(mean1, cov1, mean2, cov2, M):
    # 训练集
    X1 = np.random.multivariate_normal(mean1, cov1, int(M / 2)).T
    X2 = np.random.multivariate_normal(mean2, cov2, int(M / 2)).T
    Y1 = np.zeros_like(X1[0]).reshape((1, int(M / 2)))
    Y2 = np.ones_like(X2[0]).reshape((1, int(M / 2)))
    X = np.c_[X1, X2]
    Y = np.c_[Y1, Y2]

    # 测试集
    XT1 = np.random.multivariate_normal(mean1, cov1, int(M / 2)).T
    XT2 = np.random.multivariate_normal(mean2, cov2, int(M / 2)).T
    YT1 = np.zeros_like(XT1[0]).reshape((1, int(M / 2)))
```

```python
    YT2 = np.ones_like(XT2[0]).reshape((1, int(M / 2)))
    XT = np.c_[XT1, XT2]
    YT = np.c_[YT1, YT2]

    # 归一化
    theMax = np.max(np.c_[X, XT], axis=1).reshape((2, 1))
    theMin = np.min(np.c_[X, XT], axis=1).reshape((2, 1))
    X = X - theMin
    XT = XT - theMin
    X = X / (theMax - theMin)
    XT = XT / (theMax - theMin)

    # 训练集
    data1 = np.r_[X[:, 0:int(M / 2)], Y1]
    data2 = np.r_[X[:, int(M / 2):], Y2]
    data = np.c_[data1, data2]

    # 测试集
    dataT1 = np.r_[XT[:, 0:int(M / 2)], YT1]
    dataT2 = np.r_[XT[:, int(M / 2):], YT2]
    dataT = np.c_[dataT1, dataT2]

    print("data.shape:", data.shape)
    print("X.shape:", X.shape)
    print("Y.shape:", Y.shape)
    return data, X, Y, data1, data2, dataT, XT, YT, dataT1, dataT2


# 初始化 W 向量
# W.shape = (N, 1)
def param(N):
    if N < 3 or N % 2 != 1:
        print('the invaild N')
        return
    W = np.ones((N, 1))
    return W


# 初始化用于计算的 X（特征数的不同会有不同的 X）
# X.shape = (N, M)
def initX(X, N, M):
    temp = X
    n = int((N - 3) / 2)
    for i in range(n):
```

```python
        X = np.r_[X, temp**(i + 2)]
    X = np.r_[np.ones((1, M)), X]
    return X


# 多项式函数
# W 是参数，X 是样本，N 是特征数与 W 中元素个数相等
# N 应为大于等于 3 的奇数
# res.shape = (1, M)
def H(W, X):
    res = np.dot(W.T, X)
    return res


# 加正则项的多项式函数
def RegH(W, X, lamda, M):
    reg = lamda / (2 * M) * np.dot(W.T, W)
    res = np.dot(W.T, X) + reg * np.ones((1, M))
    return res


# sigmod 函数
# sigmod.shape = (1, M)
def G(z):
    sigmod = np.exp(-1 * z) + np.ones_like(z)
    sigmod = 1 / sigmod
    return sigmod


# loss 函数
# type(sum) = <class 'numpy.float64'>
def loss(W, X, Y, M):
    sigmod = G(H(W, X))
    sum1 = np.log(sigmod) * Y
    sum2 = np.log(np.ones_like(sigmod) - sigmod) * (np.ones_like(Y) - Y)
    temp = sum1 + sum2
    sum = np.sum(temp)
    sum = -1 * sum
    sum = sum / M

    return sum


# 梯度
```

```
def grad(X_, Y, sigmod, M):
    dz = sigmod - Y
    grad = np.dot(X_, dz.T)
    grad = grad / M
    return grad


# 加正则项的梯度
def RegGrad(X_, Y, sigmod, M, lamda, W):
    dz = sigmod - Y
    grad = np.dot(X_, dz.T) + lamda * W
    grad = grad / M
    return grad


#共轭梯度法
def conGrad(X, Y, W, N, like):
    # 对 Y 做一个近似以满足共轭梯度法的要求
    Y = like * Y + (1 - like) * (np.ones_like(Y) - Y)
    Y = -1 * np.log(1 / Y - np.ones_like(Y))

    b = np.dot(X, Y.T)
    X = np.dot(X, X.T)

    b = b.reshape((b.size, 1))
    x = W
    p = b - np.dot(X.T, W)
    r = p

    for k in range(N):
        if (r == np.zeros_like(r)).all():
            break
        alpha = ((np.dot(r.T, r)) / np.dot(np.dot(X.T, p).T, p))[0][0]
        x = x + alpha * p
        temp = r
        r = r - alpha * np.dot(X.T, p)
        belta = np.dot(r.T, r)[0][0] / np.dot(temp.T, temp)[0][0]
        p = r + belta * p
    return x

# 生成数据
mean1 = [0.25, 0.25]
cov1 = 0.017 * np.eye(2)

```

```python
# 不满足朴素贝叶斯假设
cov1[0][1] = cov1[0][0] * (-0.4)
cov1[1][0] = cov1[0][1]
print(cov1)

mean2 = [0.75, 0.75]
cov2 = cov1
M = 1000
data, X, Y, data1, data2, dataT, XT, YT, dataT1, dataT2 = genData(
    mean1, cov1, mean2, cov2, M)

# 画出样本点
plt.figure()
x1, y1 = data1[0:2, :]
x2, y2 = data2[0:2, :]
plt.plot(x1, y1, '+', color='red')
plt.plot(x2, y2, 'x', color='blue')
plt.show()

#画出测试集
plt.figure()
x1, y1 = dataT1[0:2, :]
x2, y2 = dataT2[0:2, :]
plt.plot(x1, y1, '+', color='red')
plt.plot(x2, y2, 'x', color='blue')
plt.show()

# init
N = 11
W0 = param(N)
X_ = initX(X, N, M)
XT_ = initX(XT, N, M)

# epoch
epoch = 800
# learning rate
lr = 0.05
newLoss = loss(W0, X_, Y, M)
oldLoss = newLoss
sigmod = G(H(W0, X_))

for item in range(epoch):
    W0 = W0 - lr * grad(X_, Y, sigmod, M)
    oldLoss = newLoss
```

```
    newLoss = loss(W0, X_, Y, M)
    t = oldLoss - newLoss
    if t < 0:
        lr /= 1.5
    # print('第', item, '次迭代:平方损失函数为', newLoss, '与上次相差', t)

# 画出结果
plt.figure()
x1, y1 = data1[0:2, :]
x2, y2 = data2[0:2, :]
plt.plot(x1, y1, '+', color='red')
plt.plot(x2, y2, 'x', color='blue')

x3 = np.arange(-2, 2, 0.01)
y3 = np.arange(-2, 2, 0.01)
x3, y3 = np.meshgrid(x3, y3)
z3 = W0[0] * np.ones_like(x3)
for i in range(1, N, 2):
    temp1 = W0[i] * np.power(x3, (i + 1) / 2)
    temp2 = W0[i + 1] * np.power(y3, (i + 1) / 2)
    z3 += temp1 + temp2
plt.contour(x3, y3, z3, 0)
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.show()

# 测试结果
plt.figure()
x1, y1 = dataT1[0:2, :]
x2, y2 = dataT2[0:2, :]
plt.plot(x1, y1, '+', color='red')
plt.plot(x2, y2, 'x', color='blue')

x3 = np.arange(-2, 2, 0.01)
y3 = np.arange(-2, 2, 0.01)
x3, y3 = np.meshgrid(x3, y3)
z3 = W0[0] * np.ones_like(x3)
for i in range(1, N, 2):
    temp1 = W0[i] * np.power(x3, (i + 1) / 2)
    temp2 = W0[i + 1] * np.power(y3, (i + 1) / 2)
    z3 += temp1 + temp2
plt.contour(x3, y3, z3, 0)
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
```

```
plt.show()

res = G(H(W0, XT_))

# print(res>0.5)
# print(res.shape)

res[res > 0.5] = 1
res[res <= 0.5] = 0

temp = res - YT
temp = np.abs(temp)
sum = np.sum(temp)
print(sum)

print("正确率为: %.2f%%" % (100 - 100 * sum / res.size))

W1 = param(N)
lamda = 10
# epoch
epoch = 800
# learning rate
lr = 0.05
newLoss = loss(W1, X_, Y, M)
oldLoss = newLoss
sigmod = G(RegH(W1, X_, lamda, M))

for item in range(epoch):
    W1 = W1 - lr * RegGrad(X_, Y, sigmod, M, lamda, W1)
    oldLoss = newLoss
    newLoss = loss(W1, X_, Y, M)
    t = oldLoss - newLoss
    if t <= 0:
        lr /= 1.5
    # print('第', item, '次迭代:平方损失函数为', newLoss, '与上次相差', t)

# 画出结果
plt.figure()
x1, y1 = data1[0:2, :]
x2, y2 = data2[0:2, :]
plt.plot(x1, y1, '+', color='red')
plt.plot(x2, y2, 'x', color='blue')

x3 = np.arange(-2, 2, 0.01)
```

```
y3 = np.arange(-2, 2, 0.01)
x3, y3 = np.meshgrid(x3, y3)
z3 = W1[0] * np.ones_like(x3)
for i in range(1, N, 2):
    temp1 = W1[i] * np.power(x3, (i + 1) / 2)
    temp2 = W1[i + 1] * np.power(y3, (i + 1) / 2)
    z3 += temp1 + temp2
plt.contour(x3, y3, z3, 0)
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.show()

# 测试结果
plt.figure()
x1, y1 = dataT1[0:2, :]
x2, y2 = dataT2[0:2, :]
plt.plot(x1, y1, '+', color='red')
plt.plot(x2, y2, 'x', color='blue')

x3 = np.arange(-2, 2, 0.01)
y3 = np.arange(-2, 2, 0.01)
x3, y3 = np.meshgrid(x3, y3)
z3 = W1[0] * np.ones_like(x3)
for i in range(1, N, 2):
    temp1 = W1[i] * np.power(x3, (i + 1) / 2)
    temp2 = W1[i + 1] * np.power(y3, (i + 1) / 2)
    z3 += temp1 + temp2
plt.contour(x3, y3, z3, 0)
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.show()

res = G(H(W1, XT_))
res[res > 0.5] = 1
res[res <= 0.5] = 0

temp = res - YT
temp = np.abs(temp)
sum = np.sum(temp)
print(sum)
print("正确率为: %.2f%%" % (100 - 100 * sum / res.size))

# init
like = 0.999
```

```
N = 11
W2 = param(N)
X_ = initX(X, N, M)
XT_ = initX(XT, N, M)
W2 = conGrad(X_, Y, W2, N, like)

# 画出结果
plt.figure()
x1, y1 = data1[0:2, :]
x2, y2 = data2[0:2, :]
plt.plot(x1, y1, '+', color='red')
plt.plot(x2, y2, 'x', color='blue')

x3 = np.arange(-2, 2, 0.01)
y3 = np.arange(-2, 2, 0.01)
x3, y3 = np.meshgrid(x3, y3)
z3 = W2[0] * np.ones_like(x3)
for i in range(1, N, 2):
    temp1 = W2[i] * np.power(x3, (i + 1) / 2)
    temp2 = W2[i + 1] * np.power(y3, (i + 1) / 2)
    z3 += temp1 + temp2
plt.contour(x3, y3, z3, 0)
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
plt.show()

# 测试结果
plt.figure()
x1, y1 = dataT1[0:2, :]
x2, y2 = dataT2[0:2, :]
plt.plot(x1, y1, '+', color='red')
plt.plot(x2, y2, 'x', color='blue')

x3 = np.arange(-2, 2, 0.01)
y3 = np.arange(-2, 2, 0.01)
x3, y3 = np.meshgrid(x3, y3)
z3 = W2[0] * np.ones_like(x3)
for i in range(1, N, 2):
    temp1 = W2[i] * np.power(x3, (i + 1) / 2)
    temp2 = W2[i + 1] * np.power(y3, (i + 1) / 2)
    z3 += temp1 + temp2
plt.contour(x3, y3, z3, 0)
plt.xlim((-0.05, 1.05))
plt.ylim((-0.05, 1.05))
```

```
plt.show()

res = G(H(W2, XT_))
res[res > 0.5] = 1
res[res <= 0.5] = 0
temp = res - YT
temp = np.abs(temp)
sum = np.sum(temp)
print(sum)
print("正确率为: %.2f%%" % (100 - 100 * sum / res.size))
```