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：　逻辑回归实验

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

2.1 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。

2.2 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。

2.3 实验环境：

PyCharm 2020.3.2 x64
Python 3.8　numpy1.20.3　matplotlib 3.4.2

三、设计思想（本程序中的用到的主要算法及数据结构）

3.1 理论推导：

在一般的假设条件下，类别C_1的后验概率可以写为作用在特征向量$\emptyset$的线性函数上的 logistics sigmoid 函数的形式，即

$$\mathrm{P}(C_1|\emptyset) = \mathrm{y}(\emptyset) = \sigma(w^T\emptyset) \tag{1}$$

且

$$\mathrm{P}(C_2|\emptyset) = 1 - \mathrm{P}(C_1|\emptyset) \tag{2}$$

由贝叶斯定理可得，X属于类C_1的后验概率为：

$$P(C_2|\mathrm{X}=\mathrm{x}) = \frac{P(C_2)P(X=x|C_2)}{P(C_1)P(X=x|C_1)+P(C_2)P(X=x|C_2)} \tag{3}$$

$$= \frac{1}{1+\frac{P(C_1)P(X=x|C_1)}{P(C_2)P(X=x|C_2)}} \tag{4}$$

$$= \frac{1}{1+e^{\ln\frac{P(C_1)P(X=x|C_1)}{P(C_2)P(X=x|C_2)}}} \tag{5}$$

并定义

$$\sigma(\mathrm{a}) = \frac{1}{1+e^{-a}} \tag{6}$$

$$P(C_2|\mathrm{X}=\mathrm{x})=\frac{1}{1+e^{-a}} \tag{7}$$

$$\mathrm{a}=\ln\frac{P(C_2)P(X=x|C_2)}{P(C_1)P(X=x|C_1)} \tag{8}$$

其反函数为

$$\mathrm{a}=\ln\left(\frac{\sigma}{1-\sigma}\right) \tag{9}$$

因此有

$$P(C_2|\mathrm{X})=\frac{1}{1+e^{\ln\frac{P(X=x|C_1)}{P(X=x|C_2)}+\ln\frac{P(C_1)}{P(C_2)}}} \tag{10}$$

对于离散特征值，由于特征值相互独立，可得类条件分布，

$$P(X=x|C_k)=P\left(X^{(1)}=x^{(1)},\dots,X^{(n)}=x^{(n)}\middle|C_k\right) \tag{11}$$

$$=\prod_{i=1}^{n}P(X^{(j)}=x^{(j)}|C_k) \tag{12}$$

假设类条件概率密度是高斯分布，并假定所有的类别的协方差矩阵相同，此时类别C_k的类条件概率为

$$P(X=x|C_k)=\frac{1}{(2\pi)^{\frac{n}{2}}}\frac{1}{|\Sigma|^{\frac{1}{2}}}e^{-\frac{1}{2}(x-\mu_k)^T\Sigma^{-1}(x-\mu_k)} \tag{13}$$

其中x表示维度为n的向量，μ_k是这些向量的平均值，Σ表示所有向量x的协方差矩阵。

因此有

$$\ln\frac{P(X=x|C_1)}{P(X=x|C_2)}=-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)+\frac{1}{2}(x-\mu_2)^T\Sigma^{-1}(x-\mu_2) \tag{14}$$

$$=-\frac{1}{2}(x^T\Sigma^{-1}x-2\mu_1^T\Sigma^{-1}x+\mu_1^T\Sigma^{-1}\mu_1)+\frac{1}{2}(x^T\Sigma^{-1}x-2\mu_2^T\Sigma^{-1}x+\mu_2^T\Sigma^{-1}\mu_2) \tag{15}$$

$$=(\mu_1^T-\mu_2^T)\Sigma^{-1}x-\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1+\frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 \tag{16}$$

代入公式(8)中，可得

$$P(C_1|\mathrm{X})=\frac{1}{1+e^{\ln\frac{P(X=x|C_1)}{P(X=x|C_2)}+\ln\frac{P(C_1)}{P(C_2)}}} \tag{17}$$

$$=\frac{1}{1+e^{(\mu_1^T-\mu_2^T)\Sigma^{-1}x-\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1+\frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2+\ln\frac{P(C_1)}{P(C_2)}}} \tag{18}$$

假设

$$\mathrm{w}=\Sigma^{-1}(\mu_1-\mu_2) \tag{19}$$

$$w_0=-\frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1+\frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2+\ln\frac{P(C_1)}{P(C_2)} \tag{20}$$

因此有

$$\ln\frac{P(C_1)P(X=x|C_1)}{P(C_2)P(X=x|C_2)}=\ln\frac{P(X=x|C_1)}{P(X=x|C_2)}+\ln\frac{P(C_1)}{P(C_2)} \tag{21}$$

$$=w^Tx+w_0 \tag{22}$$

代入公式(17)中，可得

$$P(C_1|\mathrm{X})=\frac{1}{1+e^{w^Tx+w_0}} \tag{23}$$

由于其归一化的特性，我们有

$$P(C_2|\mathrm{X})=1-P(C_1|\mathrm{X}) \tag{24}$$

$$=\frac{e^{w^Tx+w_0}}{1+e^{w^Tx+w_0}} \tag{25}$$

假设x各维度之间独立，则有各维度间的协方差为 0，因此有如下定义

$$\Sigma=\begin{bmatrix}\sigma(x_1,x_1) & \dots & \sigma(x_1,x_n)\\ \vdots & \ddots & \vdots\\ \sigma(x_n,x_1) & \dots & \sigma(x_n,x_n)\end{bmatrix} \tag{26}$$

$$=\begin{bmatrix}\sigma(x_1,x_1) & \dots & 0\\ \vdots & \ddots & \vdots\\ 0 & \dots & \sigma(x_n,x_n)\end{bmatrix} \tag{27}$$

因此很容易得到Σ矩阵的逆为

$$\varSigma^{-1}=\begin{bmatrix}\frac{1}{\sigma(x_1,x_1)} & \dots & 0\\ \vdots & \ddots & \vdots\\ 0 & \dots & \frac{1}{\sigma(x_n,x_n)}\end{bmatrix} \tag{28}$$

根据对μ的定义，可以展开写为下式

$$\mathrm{\mu}_i=\begin{bmatrix}\mu_{i1}\\ \vdots\\ \mu_{in}\end{bmatrix} \tag{29}$$

代入公式(20)，可得

$$w_0=-\frac{1}{2}\mu_1^T\varSigma^{-1}\mu_1+\frac{1}{2}\mu_2^T\varSigma^{-1}\mu_2+\ln\frac{P(C_1)}{P(C_2)} \tag{20}$$

$$=-\frac{1}{2}\begin{bmatrix}\mu_{11}\\ \vdots\\ \mu_{1n}\end{bmatrix}^T\begin{bmatrix}\frac{1}{\sigma(x_1,x_1)} & \dots & 0\\ \vdots & \ddots & \vdots\\ 0 & \dots & \frac{1}{\sigma(x_n,x_n)}\end{bmatrix}\begin{bmatrix}\mu_{11}\\ \vdots\\ \mu_{1n}\end{bmatrix}+\frac{1}{2}\begin{bmatrix}\mu_{21}\\ \vdots\\ \mu_{2n}\end{bmatrix}^T\begin{bmatrix}\frac{1}{\sigma(x_1,x_1)} & \dots & 0\\ \vdots & \ddots & \vdots\\ 0 & \dots & \frac{1}{\sigma(x_n,x_n)}\end{bmatrix}\begin{bmatrix}\mu_{21}\\ \vdots\\ \mu_{2n}\end{bmatrix}+\ln\frac{P(C_1)}{P(C_2)}$$

$$=\ln\frac{P(C_1)}{P(C_2)}+\sum_{i=1}^{n}\frac{\mu_{2i}^2-\mu_{1i}^2}{2\sigma_i^2} \tag{31}$$

代入公式(19)，可得

$$\mathrm{w}=\varSigma^{-1}(\mu_1-\mu_2) \tag{19}$$

$$=\begin{bmatrix}\frac{1}{\sigma(x_1,x_1)} & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & \frac{1}{\sigma(x_n,x_n)}\end{bmatrix}\begin{bmatrix}\mu_{11}-\mu_{21}\\ \vdots\\ \mu_{1n}-\mu_{2n}\end{bmatrix} \tag{32}$$

$$=\begin{bmatrix}\frac{\mu_{11}-\mu_{21}}{\sigma_1^2}\\ \vdots\\ \frac{\mu_{1n}-\mu_{2n}}{\sigma_n^2}\end{bmatrix} \tag{33}$$

此处对w扩充，将其表示为

$$\mathrm{w}=\begin{bmatrix}1\\ \frac{\mu_{11}-\mu_{21}}{\sigma_1^2}\\ \vdots\\ \frac{\mu_{1n}-\mu_{2n}}{\sigma_n^2}\end{bmatrix} \tag{34}$$

对 x 扩充，将其表示为

$$\mathrm{x}=\begin{bmatrix}1\\ x_1\\ \vdots\\ x_n\end{bmatrix} \tag{35}$$

由此，我们可以得到

$$P(C_1|\mathrm{X})=\frac{1}{1+e^{w^Tx}} \tag{36}$$

$$P(C_2|\mathrm{X})=\frac{e^{w^Tx}}{1+e^{w^Tx}} \tag{37}$$

因此

$$\frac{P(C_2|\mathrm{X})}{P(C_1|\mathrm{X})}=e^{w^Tx} \tag{38}$$

公式(6)的函数被称为sigmoid函数，它具有将参数映射到0,1之间某个值的性质，在$\mathrm{x}=0$的左右，y分别以很快的速度向0,1逼近，从而得到近似0,1标签的离散值。

根据我们的定义，我们将x分到后验概率最大的类中，因此若$\frac{P(C_2|\mathrm{X})}{P(C_1|\mathrm{X})}>1$，即$e^{w^Tx}>1$，则有$x$属于$C_2$类，否则有$x$属于$C_1$类。

假设当前的数据为$\{<X^1,Y^1>,<X^2,Y^2>,<X^3,Y^3>,\dots,<X^l,Y^l>\}$，其中$y^i$是$x^i$对应的分类$(1\le \mathrm{i}\le \mathrm{l})$。

要计算损失函数，使用极大似然估计(MLE)，计算$\mathrm{P}(<\mathrm{X},\mathrm{Y}>|\mathrm{W})$是难的问题，因此可以转化为计算极大条件似然估计(MCLE)，只需计算$\mathrm{P}(\mathrm{X}|\mathrm{Y},\mathrm{W})$。因此使用极大条件似然估计，对参数$w$估计。

$$w_{MCLE} = \arg\max_{w} \prod_{l} P(Y^l|X^l, w) \tag{38}$$

于是有似然函数

$$\mathrm{P(c, x|w)} = \prod_{i=1}^{l} P(Y^l|X^l, w) \tag{39}$$

对其取对数可得

$$\mathrm{l(w)} = \ln\left(\prod_{i=1}^{l} P(Y^l|X^l, w)\right) \tag{40}$$

$$= \sum_{i=1}^{l} \left(\ln\left(P(Y^l|X^l, w)\right)\right) \tag{41}$$

$$= \sum_{i=1}^{l} \left(Y^l \ln\left(P(Y^l = 1|X^l, w)\right) + (1 - Y^l) \ln\left(P(Y^l = 0|X^l, w)\right)\right) \tag{42}$$

$$= \sum_{i=1}^{l} \left(Y^l \ln\left(P(Y^l = 1|X^l, w)\right) - Y^l \ln\left(P(Y^l = 0|X^l, w)\right) + \ln\left(P(Y^l = 0|X^l, w)\right)\right) \tag{43}$$

$$= \sum_{i=1}^{l} \left(Y^l \ln\left(\frac{P(Y^l = 1|X^l, w)}{P(Y^l = 0|X^l, w)}\right) + \ln\left(P(Y^l = 0|X^l, w)\right)\right) \tag{44}$$

$$= \sum_{i=1}^{l} \left(Y^l \ln\left(e^{w^T \boldsymbol{x}}\right) + \ln\left(P(Y^l = 0|X^l, w)\right)\right) \tag{45}$$

$$= \sum_{i=1}^{l} \left(Y^l w^T \boldsymbol{x} + \ln\left(\frac{1}{1 + e^{w^T x}}\right)\right) \tag{46}$$

$$= \sum_{i=1}^{l} \left(Y^l w^T \boldsymbol{x} - \ln\left(1 + e^{w^T \boldsymbol{x}}\right)\right) \tag{47}$$

若要最大化公式(40)，只需取其相反数，将其转化为梯度下降，寻找极小值的问题。

我们令

$$\mathrm{L(w)} = \sum_{i=1}^{l} \left(-Y^l w^T \boldsymbol{x} + \ln\left(1 + e^{w^T \boldsymbol{x}}\right)\right) \tag{48}$$

为了避免过拟合，此处添加惩罚项

$$\mathrm{L(w)} = \sum_{i=1}^{l} \left(-Y^l w^T \boldsymbol{x} + \ln\left(1 + e^{w^T \boldsymbol{x}}\right)\right) + \frac{\lambda}{2} \|w\| \tag{50}$$

其中$\|w\| = w^T w$.

3.2 凸优化：

由于公式(50)是凸函数，因此可以找到其极小值。

(1)梯度下降法：

类比于lab1，我们可以使用梯度下降法，寻找使L(w)最小的w的值。

由公式(50)我们可以推出

$$\frac{\partial}{\partial \mathrm{w}}\mathrm{L}(\mathrm{w})=\sum_{i=1}^{l}\left(-Y^l\boldsymbol{x}+\frac{e^{w^T\boldsymbol{x}}}{1+e^{w^T\boldsymbol{x}}}\boldsymbol{x}\right) \tag{51}$$

实现梯度下降法：

$$\mathrm{w}=\mathrm{w}-\alpha\frac{\partial}{\partial \mathrm{W}}\mathrm{L}(\mathrm{w}) \tag{52}$$

对于公式(52)，选择合适的α，使$\mathrm{L}(\theta_{k+1})<\mathrm{L}(\theta_k)$。

加上惩罚项得到梯度下降迭代公式

$$\mathrm{w}=\mathrm{w}-\alpha\left(\sum_{i=1}^{l}\left(-Y^l\boldsymbol{x}+\frac{e^{w^T\boldsymbol{x}}}{1+e^{w^T\boldsymbol{x}}}\boldsymbol{x}\right)+\lambda w\right) \tag{53}$$

(2)牛顿迭代法：

规定

$$f(w)=\sum_{i=1}^{l}\left(-Y^l\boldsymbol{x}+\frac{e^{w^T\boldsymbol{x}}}{1+e^{w^T\boldsymbol{x}}}\boldsymbol{x}\right)+\lambda w \tag{54}$$

首先选择一个接近公式(51)零点的w_0，计算其对应的$f(w_0)$和切线斜率$f'(w_0)$，然后计算穿过点$(w_0,f(w_0))$并且斜率为$f'(w_0)$的直线和x轴的交点w的坐标，即对如下方程求解：

$$0=(\mathrm{w}-w_0)f'(w_0)+f(w_0) \tag{55}$$

我们新求得的w坐标命名为w_1，通常w_1会比w_0更接近方程$f(\mathrm{w})=0$的解，因此可以用w_1进行下一轮迭代。

迭代过程可以简化为如下过程：

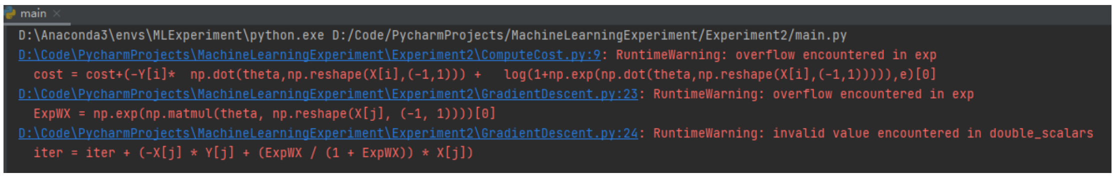
$$w_{n+1}=w_n-\frac{f(w_n)}{f'(w_n)} \tag{56}$$

因此我们可以得到：

$$f'(w)=\sum_{i=1}^{l}\left(\frac{\boldsymbol{x}e^{w^T\boldsymbol{x}}}{1+e^{w^T\boldsymbol{x}}}\boldsymbol{x}^T\right)+\lambda \tag{57}$$

3.3 溢出问题：

对于公式(50)，可能会引起溢出问题，如图。



```
main
D:\Anaconda3\envs\MLExperiment\python.exe D:/Code/PycharmProjects/MachineLearningExperiment/Experiment2/main.py
D:\Code\PycharmProjects\MachineLearningExperiment\Experiment2\ComputeCost.py:9: RuntimeWarning: overflow encountered in exp
  cost = cost+(-Y[i]*  np.dot(theta,np.reshape(X[i],(-1,1))) +   log(1+np.exp(np.dot(theta,np.reshape(X[i],(-1,1))))),e)[0]
D:\Code\PycharmProjects\MachineLearningExperiment\Experiment2\GradientDescent.py:23: RuntimeWarning: overflow encountered in exp
  ExpWX = np.exp(np.matmul(theta, np.reshape(X[j], (-1, 1))))[0]
D:\Code\PycharmProjects\MachineLearningExperiment\Experiment2\GradientDescent.py:24: RuntimeWarning: invalid value encountered in double_scalars
  iter = iter + (-X[j] * Y[j] + (ExpWX / (1 + ExpWX)) * X[j])
```

因此，我们需要解决以上问题。

$$\mathrm{w} = \mathrm{w} - \alpha\left(\sum_{i=1}^{l}\left(-Y^{l}\boldsymbol{x} + \frac{e^{w^{T}\boldsymbol{x}}}{1 + e^{w^{T}\boldsymbol{x}}}\boldsymbol{x}\right) + \lambda w\right) \tag{53}$$

我们有公式(53)的迭代公式，需要对公式(53)改进。

可能存在以下情况：

当$w^{T}\boldsymbol{x} > 0$且过大时，$\mathrm{np.exp}(w^{T}\boldsymbol{x})$会向上溢出，因此此时可以将公式 53 修改为如下公式。

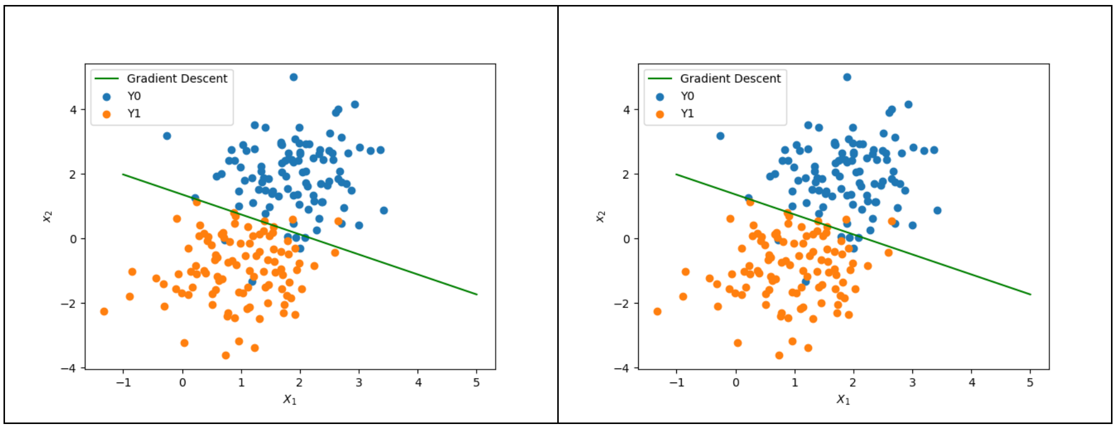
$$\mathrm{w} = \mathrm{w} - \alpha\left(\sum_{i=1}^{l}\left(-Y^{l}\boldsymbol{x} + \frac{1}{1 + e^{-w^{T}\boldsymbol{x}}}\boldsymbol{x}\right) + \lambda w\right) \tag{58}$$

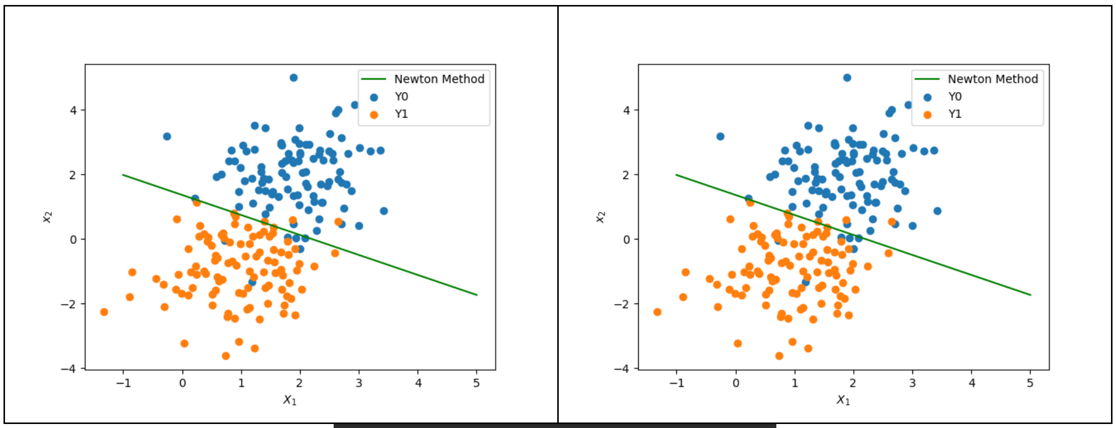
四、实验结果与分析

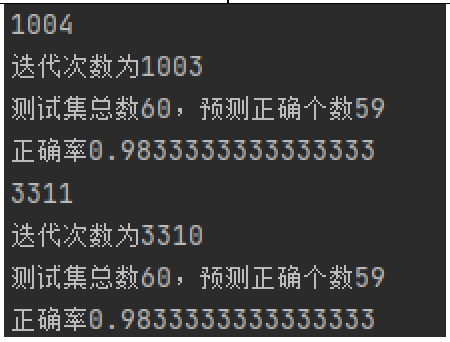
4.1 生成高斯分布的两类点，两类点中各特征相互独立：

设置训练集和测试集的比例为 7:3，生成特征相互独立的数据，每个类别 100 个数据。左侧使用梯度下降法，右侧使用牛顿迭代法。

绘图中样本点为训练集和测试集的合集。







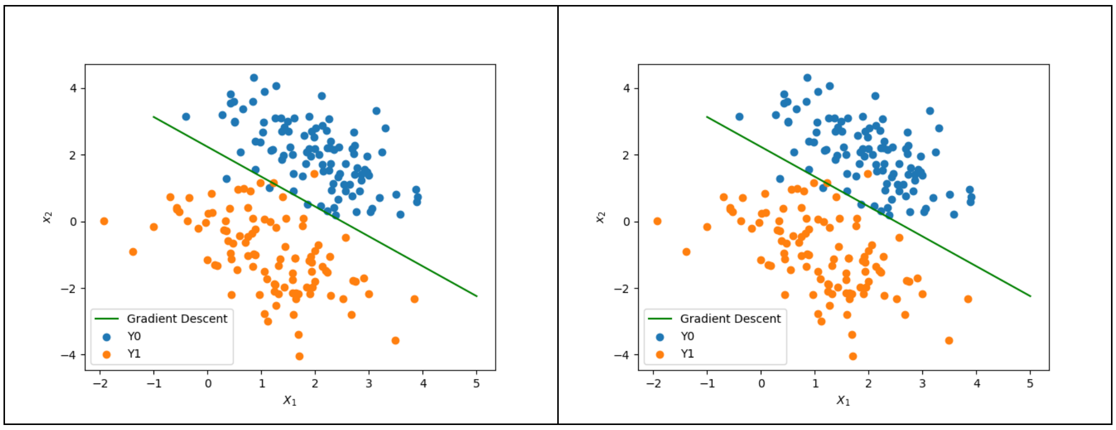
```
1004
迭代次数为1003
测试集总数60，预测正确个数59
正确率0.9833333333333333
3311
迭代次数为3310
测试集总数60，预测正确个数59
正确率0.9833333333333333
```

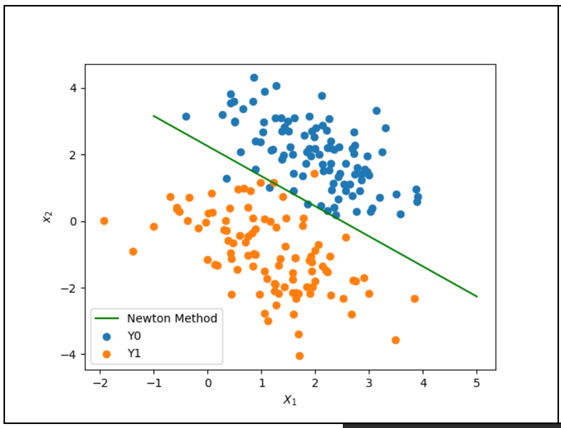
实验结果为：使用梯度下降法和使用牛顿迭代法得出曲线近乎一致，测试集共 60 个样本，预测正确 59 个。预测正确率为 98.3%。

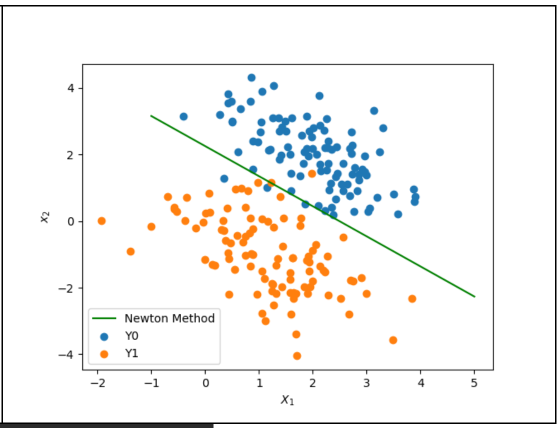
图 1 为添加惩罚项的梯度下降法，图 2 为添加惩罚项的牛顿迭代法，图 3 为添加惩罚项的梯度下降法，图 4 为添加惩罚项的牛顿迭代法。添加惩罚项对生成的数据集分类效果提升较小。

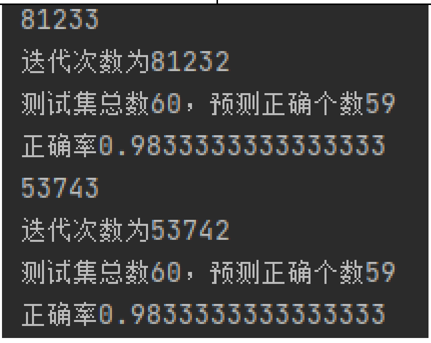
4.2 生成高斯分布的两类点，两类点中各特征不相互独立：

设置训练集和测试集的比例为 7:3，生成两类各特征不相互独立的点，每个类别 100 个数据。左侧使用梯度下降法，右侧使用牛顿迭代法。









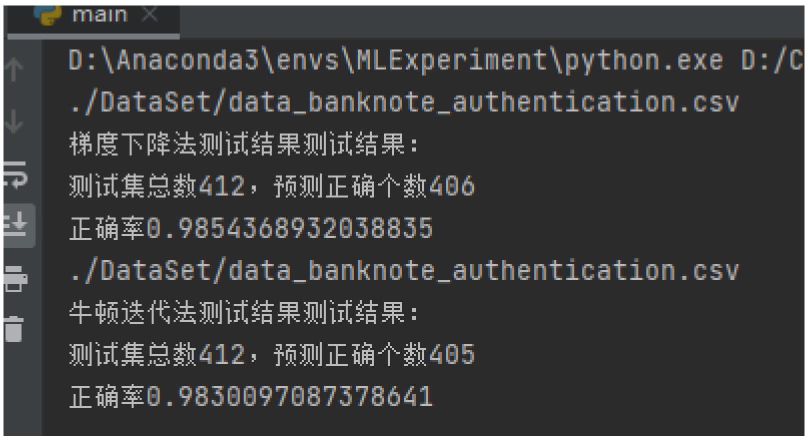
```
81233
迭代次数为81232
测试集总数60，预测正确个数59
正确率0.9833333333333333
53743
迭代次数为53742
测试集总数60，预测正确个数59
正确率0.9833333333333333
```

实验结果为：使用梯度下降法和使用牛顿迭代法得出曲线近乎一致，测试集共 60 个样本，预测正确 59 个。预测正确率为 98.3%。

图 1 为添加惩罚项的梯度下降法，图 2 为添加惩罚项的牛顿迭代法，图 3 为添加惩罚项的梯度下降法，图 4 为添加惩罚项的牛顿迭代法。添加惩罚项对生成的数据集分类效果提升较小。

4.3 使用uci上的数据集测试

(1) data_banknote_authentication数据集



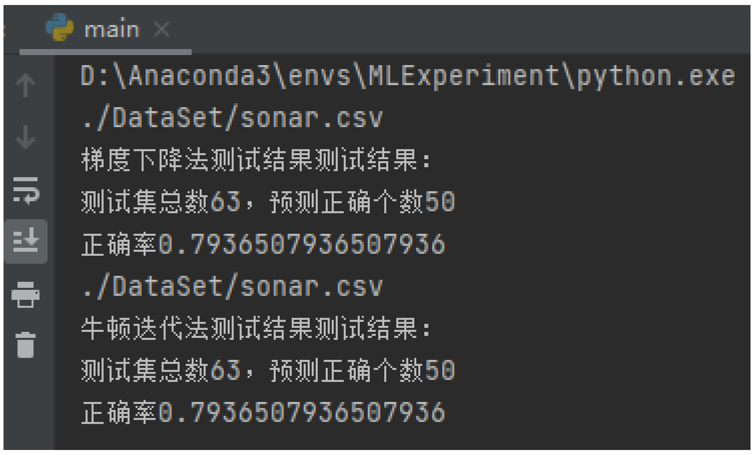
```
D:\Anaconda3\envs\MLExperiment\python.exe D:/C
./DataSet/data_banknote_authentication.csv
梯度下降法测试结果测试结果：
测试集总数412，预测正确个数406
正确率0.9854368932038835
./DataSet/data_banknote_authentication.csv
牛顿迭代法测试结果测试结果：
测试集总数412，预测正确个数405
正确率0.9830097087378641
```

实验数据：
第一列：图像经小波变换后的方差(variance)(连续值)；
第二列：图像经小波变换后的偏态(skewness)(连续值)；
第三列：图像经小波变换后的峰度(curtosis)(连续值)；
第四列：图像的熵(entropy)(连续值)；

第五列：钞票所属的类别(整数，0 或 1)。

实验结果为：使用梯度下降法和使用牛顿迭代法测试结果几乎一致，测试集共 412 个样本，使用梯度下降法预测正确 406 个，预测正确率为 98.54%；使用牛顿迭代法预测正确 405 个，预测正确率为 98.3%

(2) sonar数据集



```
D:\Anaconda3\envs\MLExperiment\python.exe
./DataSet/sonar.csv
梯度下降法测试结果测试结果:
测试集总数63，预测正确个数50
正确率0.7936507936507936
./DataSet/sonar.csv
牛顿迭代法测试结果测试结果:
测试集总数63，预测正确个数50
正确率0.7936507936507936
```

实验结果为：使用梯度下降法和使用牛顿迭代法测试结果一致，测试集共 63 个样本，使用梯度下降法预测正确 50 个，预测正确率为 79.36%；使用牛顿迭代法预测正确 50 个，预测正确率为 79.36%

五、结论

1. 使用梯度下降法和牛顿迭代法所得到分类结果近乎相同。
2. 选择不同的数据可能对分类结果有很不同的影响。

六、参考文献

[1] Pattern Recognition and Machine Learning
[2] 机器学习 周志华著 北京：清华大学出版社，2016 年 1 月.
[2] https://zhuanlan.zhihu.com/p/76639936

七、附录：源代码（带注释）

main.py　主程序

```python
from PredictResult import predictResult

if __name__ == '__main__':
    # 使用生成的数据点求解
    # 符合朴素贝叶斯分布的点求解
    # predictResult(0)
    # 不符合朴素贝叶斯分布的点求解
    # predictResult(1)
    # 使用梯度下降法对保存的文件 求解
    predictResult(2,"./DataSet/data_banknote_authentication.csv")
    # predictResult(2,"./DataSet/sonar.csv")
    # predictResult(2, "./DataSet/heart.csv")
    # 使用牛顿迭代法对保存的文件 求解
    predictResult(3, "./DataSet/data_banknote_authentication.csv")
    # predictResult(3, "./DataSet/sonar.csv")
    # predictResult(3, "./DataSet/heart.csv")
```

ComputeCost.py　计算代价值

```python
import numpy as np
from math import log
from math import e

def computeCost(X, Y, theta, Lambda):
    cost = 0
    for i in range(0, X.shape[1]):
        cost = cost + (
        -Y[i] * np.dot(theta, np.reshape(X[i], (-1, 1))) + log(1 + np.exp(np.dot(theta, np.reshape(X[i], (-1, 1))))),
        e)[0]
    cost = cost + Lambda / 2 * np.dot(theta, np.reshape(theta, (-1, 1)))
    return cost
```

DataFromFile.py　从文件中读取数据

```python
import pandas as pd
import numpy as np
from GenerateData import addUnitColumn


def readFromFile(DocumentName):
    df = pd.read_csv(DocumentName)
    df.rename(columns={df.columns.array[df.columns.shape[0] - 1]: 'Predict'}
, inplace=True)
    Y = df['Predict']
    X = df.drop('Predict', axis=1)
    X = np.array(X.values)
    X = np.reshape(X, (1, X.shape[0], X.shape[1]))
    X = addUnitColumn(X)
    Y = Y.values
    return X, Y
```

Draw.py 绘画曲线 绘点

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


def drawTheta(Start, End, theta, method):
    # number
    number = 1000
    X = np.linspace(start=Start, stop=End, num=number)
    X = np.reshape(X, (-1, 1))
    UnitMatrix = np.reshape(np.ones(number), (-1, 1))
    Y = (theta[0] * UnitMatrix + theta[1] * X) * (-1) * (1 / theta[2])
    if method == 0:
        plt.plot(X, Y, 'g', label="Gradient Descent")
    elif method == 1:
        plt.plot(X, Y, 'g', label="Newton Method")
    plt.xlabel('$X_{1}$', fontsize=10)
    plt.ylabel('$x_{2}$', fontsize=10)
    plt.legend()
    plt.show()


def plotScatter(X1Y0, X2Y0, X1Y1, X2Y1):
    plt.scatter(X1Y0, X2Y0, label='Y{}'.format(0))
    plt.scatter(X1Y1, X2Y1, label='Y{}'.format(1))
```

GenerateData.py

```python
import numpy as np
from sklearn.model_selection import train_test_split


```

```python
def generate2DimensionalData(Mu1, Mu2, cov11, cov12, cov21, cov22, Num, noiseSigma1, noiseSigma2):
    """
    生成二维高斯分布数据
    :param Mu1: 维度 1 的平均值
    :param Mu2: 维度 2 的平均值
    :param cov11: 维度 1 的方差
    :param cov12: 维度 12 的协方差
    :param cov21: 维度 21 的协方差
    :param cov22: 维度 2 的方差
    :param Num: 生成样本点的个数
    :param Class: 类别
    :param noiseSigma1: 维度 1 噪声的方差
    :param noiseSigma2: 维度 2 噪声的方差
    :return: 分别返回两个维度的样本点
    """
    mean = np.array([Mu1, Mu2])
    cov = np.array([[cov11, cov12], [cov21, cov22]])
    Data = np.random.multivariate_normal(mean, cov, Num)
    X1 = Data[:, 0]
    X2 = Data[:, 1]
    # 添加高斯噪声
    GuassNoise1 = np.random.normal(0, scale=noiseSigma1, size=Num)
    GuassNoise2 = np.random.normal(0, scale=noiseSigma2, size=Num)
    X1 = X1 + GuassNoise1
    X2 = X2 + GuassNoise2
    return X1, X2


def generateData(method):
    """
    生成两类数据并满足朴素贝叶斯分布
    :param method: 分别生成符合朴素贝叶斯分布和不符合朴素贝叶斯分布的数据
    :return: 生成数据
    """
    if method == 1:
        # Y0Mu1 = 2
        # Y0Mu2 = 2
        # Y0cov11 = 1
        # Y0cov12 = -0.5
        # Y0cov21 = -0.5
        # Y0cov22 = 1
        # Y0Num = 200
        # Y0noiseSigma1 = 0.2
        # Y0noiseSigma2 = 0.4
        # X1Y, X2Y = generate2DimensionalData(Y0Mu1, Y0Mu2, Y0cov11, Y0cov12, Y0cov21, Y0cov22, Y0Num,
        #                                       Y0noiseSigma1, Y0noiseSigma2)
        # X1Y0 = X1Y[0:np.int64(Y0Num/2)]
        # X1Y1 = X1Y[np.int64(Y0Num/2):np.int64(Y0Num)]
        # X2Y0 = X2Y[0:np.int64(Y0Num/2)]
        # X2Y1 = X2Y[np.int64(Y0Num/2):np.int64(Y0Num)]
        # XY0 = np.dstack((X1Y0, X2Y0))
        # XY1 = np.dstack((X1Y1, X2Y1))
        # XY0 = addUnitColumn(XY0)
        # XY1 = addUnitColumn(XY1)
        # X = np.vstack((XY0, XY1))
        # Y_zero = np.zeros([XY0.shape[0]])
        # Y_ones = np.ones([XY1.shape[0]])
        # Y_zero = np.reshape(Y_zero, (-1, 1))
        # Y_ones = np.reshape(Y_ones, (-1, 1))
        # Y = np.vstack((Y_zero, Y_ones))
        # Y = np.reshape(Y, (-1))
        # return X, Y, X1Y0, X2Y0, X1Y1, X2Y1

        # Y0 类生成数据的参数
        Y0Mu1 = 2
        Y0Mu2 = 2
        Y0cov11 = 1
        Y0cov12 = -0.5
        Y0cov21 = -0.5
        Y0cov22 = 0.7
        Y0Num = 100
        Y0noiseSigma1 = 0.2
        Y0noiseSigma2 = 0.4
        # Y1 类生成数据的参数
        Y1Mu1 = 1
        Y1Mu2 = -1
        Y1cov11 = 1
        Y1cov12 = -0.6
        Y1cov21 = -0.6
        Y1cov22 = 1.3
        Y1Num = 100
        Y1noiseSigma1 = 0.1
        Y1noiseSigma2 = 0.2
        X1Y0, X2Y0 = generate2DimensionalData(Y0Mu1, Y0Mu2, Y0cov11, Y0cov12, Y0cov21, Y0cov22, Y0Num,
                                              Y0noiseSigma1, Y0noiseSigma2)
        X1Y1, X2Y1 = generate2DimensionalData(Y1Mu1, Y1Mu2, Y1cov11, Y1cov12, Y1cov21, Y1cov22, Y1Num,
                                              Y1noiseSigma1, Y1noiseSigma2)
        XY0 = np.dstack((X1Y0, X2Y0))
        XY1 = np.dstack((X1Y1, X2Y1))
        XY0 = addUnitColumn(XY0)
        XY1 = addUnitColumn(XY1)
        X = np.vstack((XY0, XY1))
        Y_zero = np.zeros([XY0.shape[0]])
        Y_ones = np.ones([XY1.shape[0]])
        Y_zero = np.reshape(Y_zero, (-1, 1))
        Y_ones = np.reshape(Y_ones, (-1, 1))
        Y = np.vstack((Y_zero, Y_ones))
        Y = np.reshape(Y, (-1))
        return X, Y, X1Y0, X2Y0, X1Y1, X2Y1



    elif method == 0:
        # Y0 类生成数据的参数
        Y0Mu1 = 2
        Y0Mu2 = 2
        Y0cov11 = 0.5
        Y0cov12 = 0
        Y0cov21 = 0
        Y0cov22 = 1
        Y0Num = 100
        Y0noiseSigma1 = 0.2
        Y0noiseSigma2 = 0.4
        # Y1 类生成数据的参数
        Y1Mu1 = 1
        Y1Mu2 = -1
        Y1cov11 = 0.5
        Y1cov12 = 0
        Y1cov21 = 0
        Y1cov22 = 1
        Y1Num = 100
        Y1noiseSigma1 = 0.1
        Y1noiseSigma2 = 0.2
        X1Y0, X2Y0 = generate2DimensionalData(Y0Mu1, Y0Mu2, Y0cov11, Y0cov12, Y0cov21, Y0cov22, Y0Num,
                                              Y0noiseSigma1, Y0noiseSigma2)
        X1Y1, X2Y1 = generate2DimensionalData(Y1Mu1, Y1Mu2, Y1cov11, Y1cov12, Y1cov21, Y1cov22, Y1Num,
                                              Y1noiseSigma1, Y1noiseSigma2)
        XY0 = np.dstack((X1Y0, X2Y0))
        XY1 = np.dstack((X1Y1, X2Y1))
        XY0 = addUnitColumn(XY0)
        XY1 = addUnitColumn(XY1)
        X = np.vstack((XY0, XY1))
        Y_zero = np.zeros([XY0.shape[0]])
        Y_ones = np.ones([XY1.shape[0]])
        Y_zero = np.reshape(Y_zero, (-1, 1))
        Y_ones = np.reshape(Y_ones, (-1, 1))
        Y = np.vstack((Y_zero, Y_ones))
        Y = np.reshape(Y, (-1))
        return X, Y, X1Y0, X2Y0, X1Y1, X2Y1


def addUnitColumn(X):
    """
    为 x 添加全为 1 的列
    :param X: 待添加的矩阵
    :return: 添加后的矩阵
    """
    UnitMatrix = np.ones(X.shape[1])
    # UnitMatrix = np.reshape(UnitMatrix,(X.shape[0],-1))
    X = np.dstack((UnitMatrix, X))
    X = np.reshape(X, (X.shape[1], X.shape[2]))
    return X


def TrainTestSplit(X, Y):
    """
    将数据集分为训练集和测试集
    :param X: 训练特征数据
    :param Y: 训练数据对应分类
    :return: 分开后的数据集
    """
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.7)
    return X_train, X_test, Y_train, Y_test
```

GradientDescent.py 梯度下降法

```python
import numpy as np
from ComputeCost import computeCost


def gradientDescent(X, Y, alpha, Lambda, precision, iterNum):
    """
    使用梯度下降法进行迭代
    :param X: 数据特征
    :param Y: 数据分类
    :param alpha: 学习率
    :param Lambda: 惩罚项系数
    :param precision: 迭代精度
    :param iterNum: 迭代次数
    :return:
    """
    theta = np.zeros(X.shape[1])
    descentStore = np.zeros(iterNum + 1)
    iterStore = np.zeros(iterNum + 1)
    iterStore[0] = np.inf
    for i in range(1, iterNum):
        iter = 0
        for j in range(0, X.shape[0]):

            # WX = np.matmul(theta, np.reshape(X[j], (-1, 1)))
            # ExpWX = np.exp(WX)[0]
            # iter = iter + (-X[j] * Y[j] + (ExpWX/ (1 + ExpWX)) * X[j])

            WX = np.matmul(theta, np.reshape(X[j], (-1, 1)))
            ExpWX = np.exp(WX)[0]

            if ExpWX == np.inf:
                iter = iter + (-X[j] * Y[j] + X[j])
            else:
                iter = iter + (-
X[j] * Y[j] + (ExpWX / (1 + ExpWX)) * X[j])

        # if i % 1000 == 0:
        #     print("np.sum(iter)")
        #     print(abs(np.sum(iter)))
        iterStore[i + 1] = np.sum(iter)
        # if abs(iterStore[i+1]) > abs(iterStore[i]):
        #     alpha = alpha/2
        #     print("当前 alpha={}".format(alpha))
        # print("iterStore[i+1]")
        # print(iterStore[i+1])
        # print("iterStore[i+1]-iterStore[i]")
        # print(abs(iterStore[i+1]-iterStore[i]))
        # if abs(iterStore[i+1]-iterStore[i]) <= precision:
        #     # print(i)
        #     print("迭代次数为{}".format(i - 1))
        #     print(abs(descentStore[i]-descentStore[i-1]))
        #     break
        iter = iter + Lambda * theta
        theta = theta - alpha * iter
        descentStore[i] = computeCost(X, Y, theta, Lambda)
        # if abs(descentStore[i]-descentStore[i-1])<=precision:
        #     print(i)
        #     print("迭代次数为{}".format(i-1))
        #     # print("descentStore[i]-descentStore[i-1]")
        #     # print(abs(descentStore[i]-descentStore[i-1]))
        #     break
    return theta
```

NewtonMethod.py 牛顿迭代法

```python
import numpy as np


def newtonMethod(X, Y, Lambda, precision, iterNum):
    """
    使用牛顿迭代法进行迭代
    :param X: 数据特征
    :param Y: 数据分类
    :param Lambda: 惩罚项系数
    :param precision: 预测精度
    :param iterNum: 迭代次数
    :return: theta
    """
    theta = np.zeros(X.shape[1])
    iterStore = np.zeros(iterNum + 1)
    iterStore[0] = np.inf
    for i in range(0, iterNum):
        iterComputeDifferential = 0
        iterCompute = 0
        for j in range(0, X.shape[0]):
            ExpWX = np.exp(np.dot(theta, np.reshape(X[j], (-1, 1))))
            iterComputeDifferential = iterComputeDifferential + (ExpWX * np.dot(X[j], np.reshape(X[j], (-1, 1)))) / (
                        1 + ExpWX)

            ExpWX = np.exp(np.matmul(theta, np.reshape(X[j], (-1, 1))))[0]
            iterCompute = iterCompute + (-X[j] * Y[j] + (ExpWX / (1 + ExpWX)) * X[j])

        iterStore[i + 1] = np.sum(iterCompute)
        # if abs(iterStore[i+1]-iterStore[i]) <= precision:
        #     print(i)
        #     print("迭代次数为{}".format(i - 1))
        #     break

        if abs(iterStore[i + 1] - iterStore[i]) <= precision:
            print(i)
            print("迭代次数为{}".format(i - 1))
            break

        iterCompute = iterCompute + Lambda * theta
        iterComputeDifferential = iterComputeDifferential + Lambda
        theta = theta - iterCompute / iterComputeDifferential
    return theta
```

PredictResult.py 预测结果

```python
from GradientDescent import gradientDescent
from GenerateData import TrainTestSplit
from DataFromFile import readFromFile
from VerificationPredict import predictPrecision
from NewtonMethod import newtonMethod
from GenerateData import generateData
from Draw import drawTheta
from Draw import plotScatter
import numpy as np
import matplotlib.pyplot as plt


def predictResult(method, FileName=""):
    """
    从文件中读取数据并根据不同的 method 使用不同方法求得极小值
    并通过调用自己写的精度函数验证预测准确率
    :param method: 使用的方法，0:对于符合朴素贝叶斯分布的数据 1：对于不符合朴素贝叶斯分布的数据 2：梯度下降法 3：牛顿迭代法
    :param FileName: 文件名称
    """

    if method == 0:
        X, Y, X1Y0, X2Y0, X1Y1, X2Y1 = generateData(0)
        X_train, X_test, Y_train, Y_test = TrainTestSplit(X, Y)
        theta = gradientDescent(X_train, Y_train, 0.01, 1e-6, 1e-7, 20000)
        plotScatter(X1Y0, X2Y0, X1Y1, X2Y1)
        drawTheta(-1, 5, theta, 0)
        predictPrecision(theta, X_test, Y_test)
        theta = gradientDescent(X_train, Y_train, 0.01, 0, 1e-7, 20000)
        plotScatter(X1Y0, X2Y0, X1Y1, X2Y1)
        drawTheta(-1, 5, theta, 0)
        predictPrecision(theta, X_test, Y_test)
        theta = newtonMethod(X_train, Y_train, 1e-6, 1e-7, 20000)
        plotScatter(X1Y0, X2Y0, X1Y1, X2Y1)
        drawTheta(-1, 5, theta, 1)
        predictPrecision(theta, X_test, Y_test)
        theta = newtonMethod(X_train, Y_train, 0, 1e-7, 20000)
        plotScatter(X1Y0, X2Y0, X1Y1, X2Y1)
        drawTheta(-1, 5, theta, 1)
        predictPrecision(theta, X_test, Y_test)

    elif method == 1:
        X, Y, X1Y0, X2Y0, X1Y1, X2Y1 = generateData(1)
        X_train, X_test, Y_train, Y_test = TrainTestSplit(X, Y)
        theta = gradientDescent(X_train, Y_train, 0.001, 1e-7, 1e-7, 20000)
        plotScatter(X1Y0, X2Y0, X1Y1, X2Y1)
        drawTheta(-1, 5, theta, 0)
        predictPrecision(theta, X_test, Y_test)
        theta = gradientDescent(X_train, Y_train, 0.001, 0, 1e-7, 20000)
        plotScatter(X1Y0, X2Y0, X1Y1, X2Y1)
        drawTheta(-1, 5, theta, 0)
        predictPrecision(theta, X_test, Y_test)
        theta = newtonMethod(X_train, Y_train, 1e-7, 1e-7, 20000)
        plotScatter(X1Y0, X2Y0, X1Y1, X2Y1)
        drawTheta(-1, 5, theta, 1)
        predictPrecision(theta, X_test, Y_test)
        theta = newtonMethod(X_train, Y_train, 0, 1e-7, 20000)
        plotScatter(X1Y0, X2Y0, X1Y1, X2Y1)
        drawTheta(-1, 5, theta, 1)
        predictPrecision(theta, X_test, Y_test)


    elif method == 2:
        X, Y = readFromFile(FileName)
        X_train, X_test, Y_train, Y_test = TrainTestSplit(X, Y)
        theta = gradientDescent(X_train, Y_train, 0.0001, 1e-7, 1e-8, 50000)
        print(FileName)
        print("梯度下降法测试结果测试结果：")
        predictPrecision(theta, X_test, Y_test)
    elif method == 3:
        X, Y = readFromFile(FileName)
        X_train, X_test, Y_train, Y_test = TrainTestSplit(X, Y)
        theta = newtonMethod(X_train, Y_train, 1e-7, 1e-8, 50000)
        print(FileName)
        print("牛顿迭代法测试结果测试结果：")
        predictPrecision(theta, X_test, Y_test)
```

VerificationPredict.py 验证测试集结果

```python
import numpy as np


def predictPrecision(theta, X_test, Y_test):
    """
    计算预测的准确率
    :param theta:迭代出来的系数
    :param X_test: 待测试数据
    :param Y_test: 数据集所属真实分类
    """

    predict = np.dot(X_test, theta)
    accuracy = 0
    for i in range(0, predict.shape[0]):
        truth = Y_test[i]
        if (predict[i] >= 0 and truth == 1) or (predict[i] <= 0 and truth == 0):
            accuracy += 1
    print("测试集总数{}，预测正确个数{}".format(predict.shape[0], accuracy))
    print("正确率{}".format(accuracy / predict.shape[0]))
```