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：1190200610
姓名： 张景阳

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

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

环境：语言不限，可以用 matlab，python。求解解析解时可以利用现成的矩阵求逆。梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不许用现成的平台，例如 pytorch，tensorflow 的自动微分工具。

三、设计思想（本程序中的用到的主要算法及数据结构）

1. 从朴素贝叶斯的推导过程

实验前提：

1） X 是每一维都是实数，且表示为<X1….Xn>的向量形式

2） Y 是一组布尔值

3） 假设 X 与 Y 独立同分布（很重要，这样我们才可以使用朴素贝叶斯）

4） 假设 P(X|Y = y)服从高斯分布，均值是 u，方差是 σ

5） Y 是伯努利分布，即两点分布

我们的思想是从朴素贝叶斯出发，直接求出后验分布

推导如下：

首先：

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

由于 P(X|Y = y)服从高斯分布，均值是 u，方差是 σ，即：

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

带入上面的公式我们可以得到:

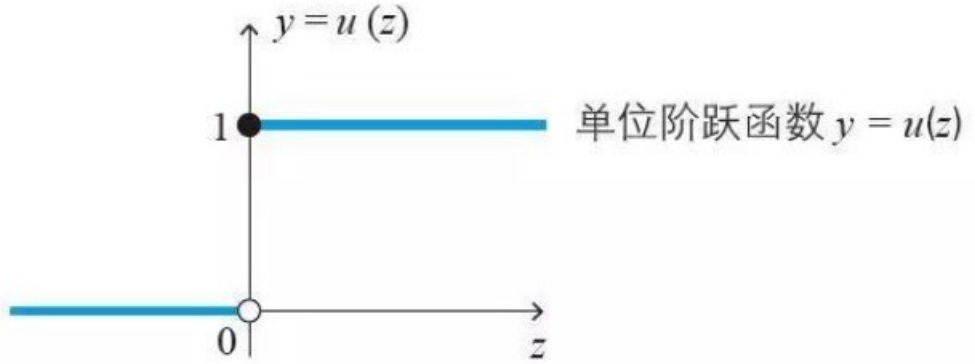
$$P(Y=1|X)=\frac{1}{1+exp(w_0+\sum_{i=1}^{n}w_iX_i)}$$
$$w_0=\sum_i\frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}+ln\frac{1-\pi}{\pi};w_i=\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}$$

综上：

$$P(Y=0|X)=\frac{exp(w_0+\sum_{i=1}^{n}w_iX_i)}{1+exp(w_0+\sum_{i=1}^{n}w_iX_i)}$$
$$ln\frac{P(Y=0|X)}{P(Y=1|X)}=ln(exp(w_0+\sum_{i=1}^{n}w_iX_i))=w_0+\sum_{i=1}^{n}w_iX_i$$

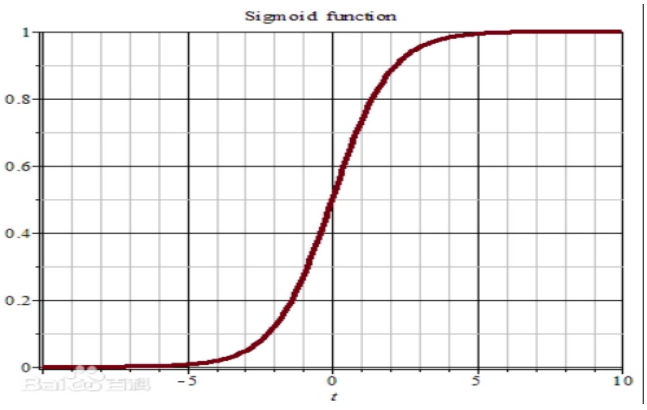
我们可以得到上面的式子线性回归的方程，但是我们要做的是一个分类问题，如何将线性的结果连接到分类的问题上哪？

这里引入一个分类的思想，考虑二分类任务，其输出标记 y∈{0，1}，而线性回归模型产生的预测值 z 是一个实值。我们需要将实值 z 转换为 0/1 值。最理想的是单位阶跃函数：



若预测值大于 0 就判为正例，小于 0 则判为反例，预测值为临界值 0 则可任意判别。

但是可惜的是，单位阶跃函数并不连续，因此不能直接作为线性转换为分类的函数。于是我们希望能够找到在一定程度上近似单位阶跃函数的“代替函数”，并且希望他单调可微。对数几率函数(Sigmoid)正是这样一个常用的代替函数：



注：sigmoid 函数也叫 Logistic，用于隐层神经元输出，取值范围为(0,1)，它可以将一个实数映射到(0,1)的区间，可以用来做二分类。在特征相差比较复杂或是相差不是特别大时效果比较好。Sigmoid 作为激活函数有以下优缺点：

优点：平滑、易于求导。

缺点：激活函数计算量大，反向传播求误差梯度时，求导涉及除法；反向传播时，很容易就会出现梯度消失的情况，从而无法完成深层网络的训练。

Sigmoid 函数：

$$S(x)=\frac{1}{1+e^{-x}}$$

综上，我们通过通过朴素贝叶斯将分类问题转化为线性求实值的问题，又引入 Sigmoid 函数来实现实值到 0/1 值的转换，因此我们的机器学习模型如下：

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

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

2. 梯度下降法

方法一：最大似然估计（MLE）

对于我们得到的线性分类，我们可以像线性回归一样运用最大似然估计来求解我们的参数 w。

我们这样我们就很熟悉了，其实公式都是没有变化的，只是其中的假设函数换成了 Sigmoid(h(x))，h(x)是我们的线性回归。

线性回归的模型如下：

$$h_\theta(x)=\theta^Tx$$

逻辑回归的模型定义：

$$g(x)=\frac{1}{1+\exp^{-x}}$$

我们得到的逻辑回归的最终模型：

$$h_\theta(x)=g(\theta^Tx)=\frac{1}{1+\exp^{(-\theta^Tx)}}$$

我们假设上式是等于正例的概率，因此我们可以得到：

$$\begin{cases}P(c=1|\,x;\theta)=h_\theta(x)\\P(c=0\,|\,x;\theta)=1-h_\theta(x)\end{cases}$$

通过合并我们可以将上面两个式子合并成一个公式：

$$P(c=y\mid x;\theta)=(h_\theta(x))^{y}(1-h_\theta(x))^{1-y}$$

根据最大似然法，我们的似然函数为：

$$L(\theta)=\prod_{i=1}^{m}(h_\theta(x))^{y}(1-h_\theta(x))^{1-y}$$

这样得到的公式又连乘，会有等于 0 的情况，因此我们取对数似然，将其变成连加：

$$\log(L(\theta))=\sum_{i=1}^{m}y(h_\theta(x))+(1-y)(1-h_\theta(x))$$

然后利用对数极大似然估计，即求上式的极大值。通常情况下，求极大值我们会将其转变成求极小值。因此，我们可以得到我们的代价函数：

$$J(\theta)=-\frac{1}{m}\log(L(\theta))$$

我们梯度下降的目的就是让代价函数尽可能小，利用我们梯度下降的思想，对代价函数求导，得到梯度，即总体在一个小区域内总体下降最快的方向：

$$\frac{\delta}{\delta\theta_j}J(\theta)=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x)-y_i)\theta_i^{j}$$

接下来的步骤就是对参数进行梯度更新，知道代价函数小到达到我们设定的要求即可。

方法二：最大后验估计（MAP）

我们通过先验概率估计，从而使得 P(w)P(Y｜X,w)最大，即：

$$W\leftarrow\arg\max_{W}\ \ln[P(W)\prod_{l}P(Y^{l}|X^{l},W)]$$

我们也跟上面的方法相同，也是求解似然函数，之后转化为对数似然，但是对于结果我们可以和上面的 MLE 对比一下可以得到：

$$w_i\leftarrow w_i-\eta\lambda w_i+\eta\sum X_i^{l}(Y^{l}-\hat{P}(Y^{l}=1|X^{l},W))$$

本质上就是，MLE 仅用似然来估计概率，而 MAP 是将先验概率加上之后再估计概率。这样的区别体现在最后的方程中就是在 MLE 的基础上加上了线性回归的正则项。这样的好处是在训练集较小的时候会矫正过拟合。

但是，随着样本数据越来越多，MAP 会逐步逼近 MLE。

3. 牛顿法

原因：梯度下降法由于处理的数据有不同的量纲和量纲单位，导致不同维度的数据之间尺度差异很大，目标函数的等高线是椭圆形的。这样在通过最小化目标函数寻找最优解的过程中，梯度下降法所走的路线是锯齿状的，需要经过的迭代次数过多，严重影响了算法的效率。虽然可以通过数据的归一化来提高算法的

效率，这里我们介绍一个数值优化的方法：牛顿法

对于我们呢求解到的代价函数，我们可以用数值优化的方法：牛顿法。不同通过梯度下降，而是通过函数的二阶泰勒展开去估计曲线，然后用二阶泰勒展开的函数的极值点去估计函数的极值点。重复迭代直到找到极值点。

假设，代价函数的一阶导为：

$$\frac{\partial J}{\partial \theta} = [\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \cdots, \frac{\partial J}{\partial \theta_n}]$$

二阶导数（Hessian 矩阵）为：

$$H(\mathrm{J}) = \begin{bmatrix} \frac{\partial^2 J}{\partial \theta_1^2} & \frac{\partial^2 J}{\partial \theta_1 \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_1 \theta_n} \\ \frac{\partial^2 J}{\partial \theta_2 \theta_1} & \frac{\partial^2 J}{\partial \theta_2^2} & \cdots & \frac{\partial^2 J}{\partial \theta_2 \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial \theta_n \theta_1} & \frac{\partial^2 J}{\partial \theta_n \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_n^2} \end{bmatrix}$$

代价函数包含二阶导数的泰勒展开式为：

$$J(\theta + \Delta\theta) = J(\theta) + \Delta\theta^T \frac{\partial J(\theta)}{\partial \theta} + \frac{1}{2}\Delta\theta^T \frac{\partial^2 J(\theta)}{\partial \theta^2}\Delta\theta$$

将上式看做 Δθ 的函数，其最小值应该在其偏导数等于 0 处取得：

$$\Delta\theta = -(\frac{\partial^2 J(\theta)}{\partial \theta^2})^{-1}\frac{\partial J(\theta)}{\partial \theta} = -H(\theta)^{-1}\frac{\partial J(\theta)}{\partial \theta}$$

θ+Δθ 是对目标函数取得最小值时参数的一个较好的估计，在牛顿法中会在 ΔθΔθ 的基础上乘以一个步长 α（取值小于 1，比如 0.001）。 使用牛顿法迭代过程为：

$$\theta = \theta + \alpha\Delta\theta = \theta - \alpha H(\theta)^{-1}\frac{\partial J(\theta)}{\partial \theta}$$

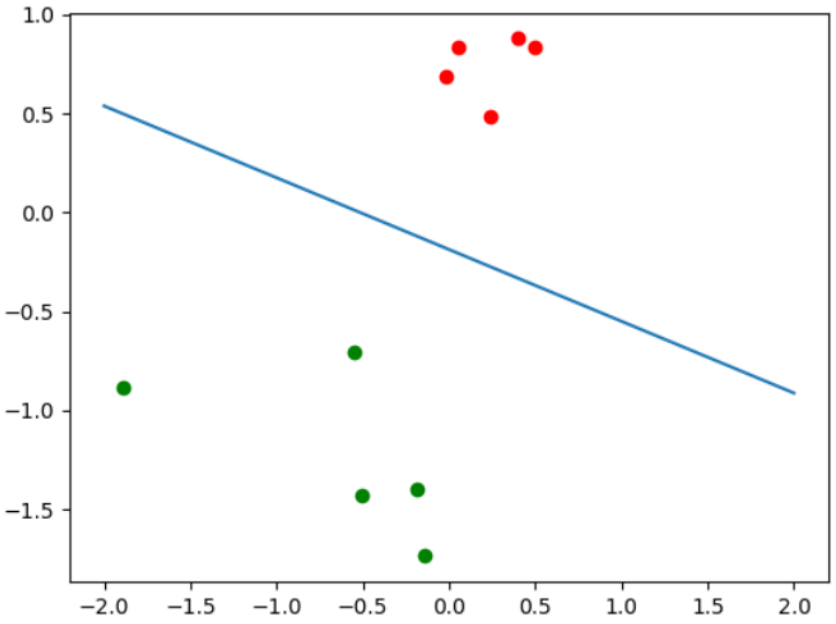
在我们的逻辑回归中，我们应用牛顿法可以得到：（正则项也考虑在内）

$$\theta = \theta - \alpha(H(\theta)^{-1}\frac{\partial J(\theta)}{\partial \theta} + \lambda\theta)$$

四、实验结果与分析

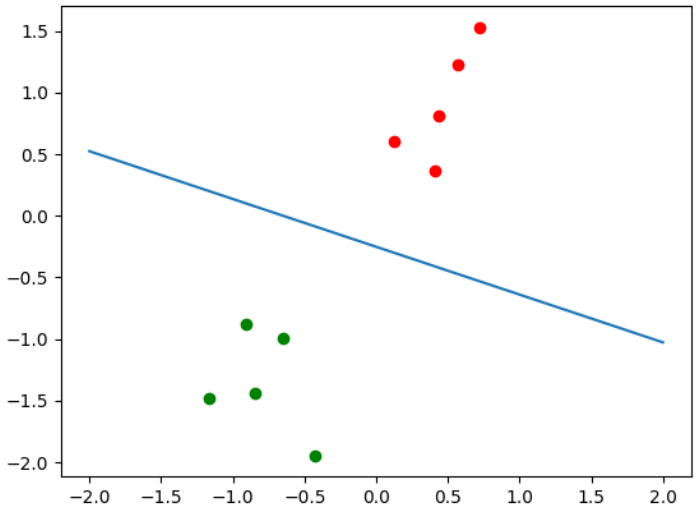
1. 梯度下降法：

1) 我通过一个 threshold，即两次代价函数的差小于 threshold 时，即达到我们的学习效果。当样本数目为 10，学习率为 1e-2，threshold 为 1e-8，两个特征满足朴素贝叶斯的情况下，我们学习情况如下：



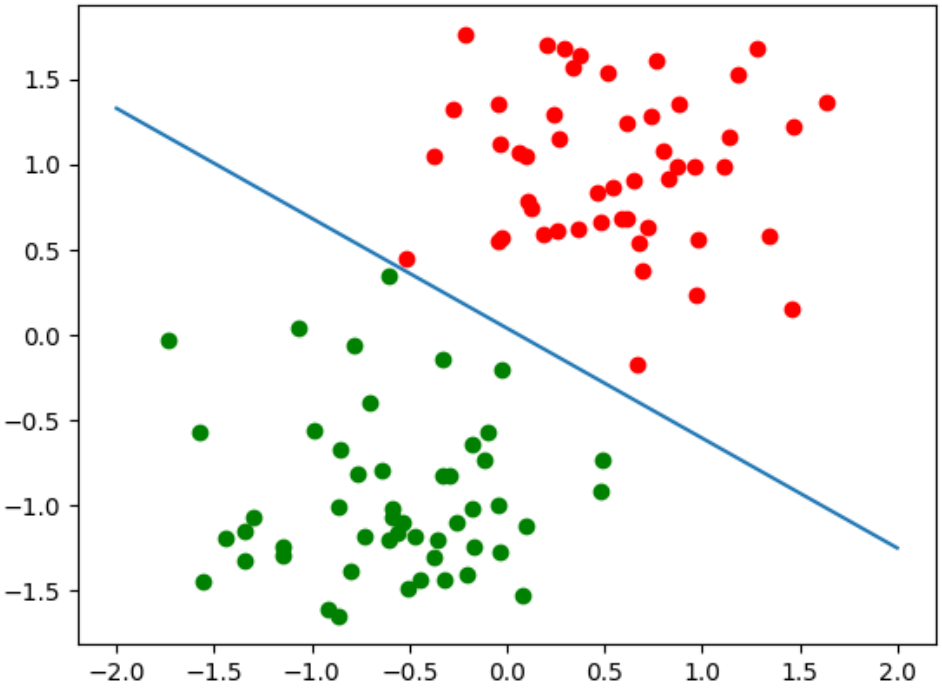
发现效果不是很好，我们通过添加正则项看看会有什么效果：

2) 当 lm=1e-4，学习率为 1e-3，threshold=1e-8 时，我们可以得到：



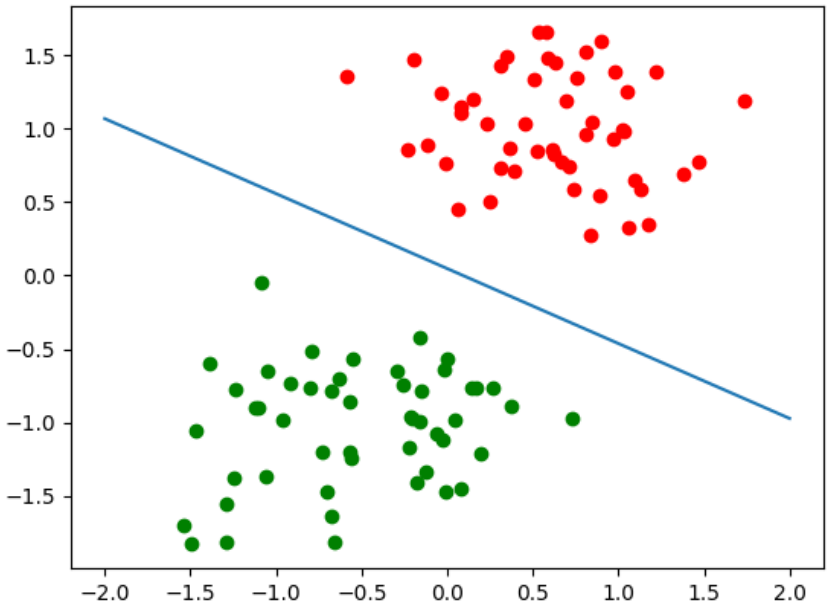
感觉有所改善，我们继续增加样本个数

3) M = 100 时，无正则项



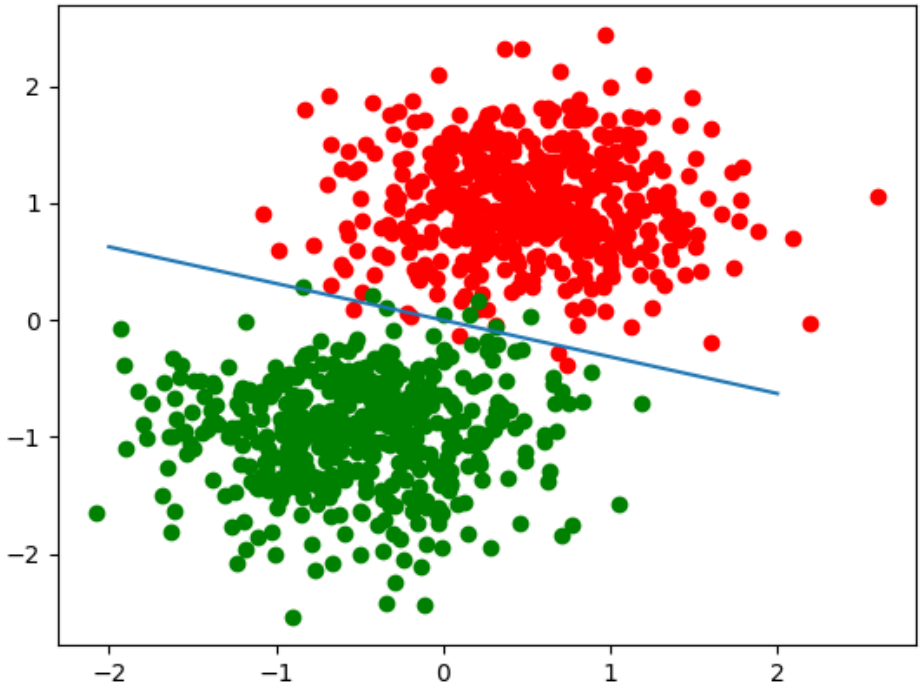
学习情况有所改善

4) M = 100 时，有正则项，lm=1e-4

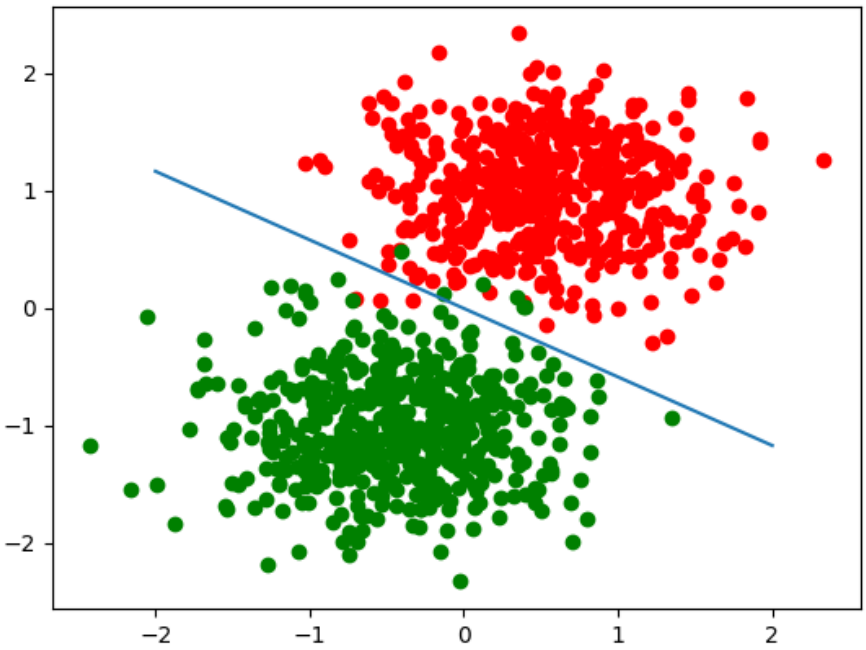


感觉学习情况并没有什么差别，但是斜率有所变化。正则项会改变求得的 w 从而导致分界线的斜率和截距变化，具有更好的泛化性能。

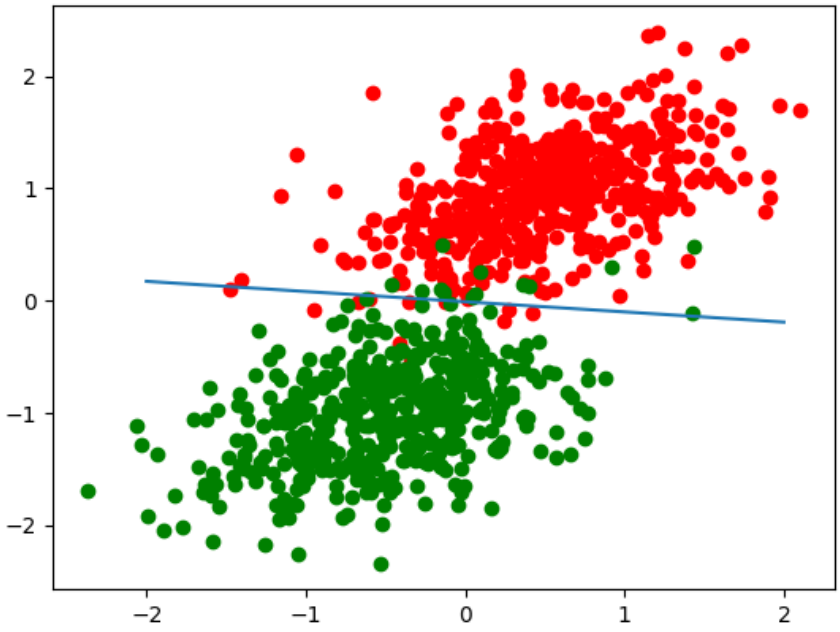
5) 当 M = 1000 无正则项



6）当 M = 1000 有正则项



7）当 M = 1000 不满足朴素贝叶斯分布

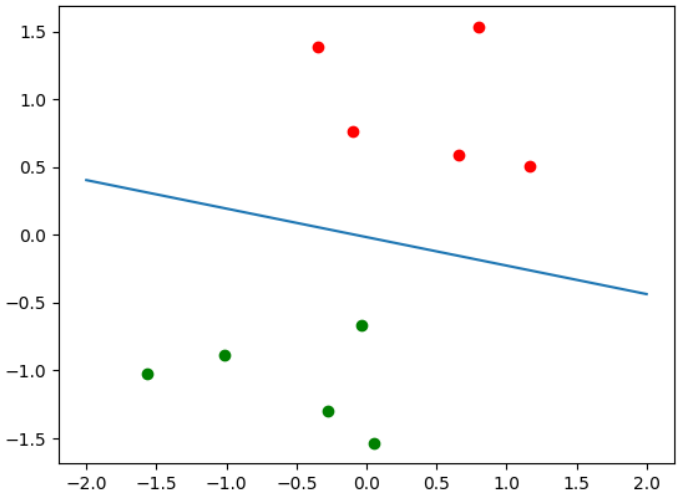


由于不满足朴素贝叶斯分布，所以正类与父类在图像上并不是完全独立的，两个样本看起来有所关联。因为我们逻辑回归梯度下降的公式是基于朴素贝叶斯，即两个特征值独立同分布。所以我们根据这个公式的学习结果没有上面的满足朴素贝叶斯的情况好。

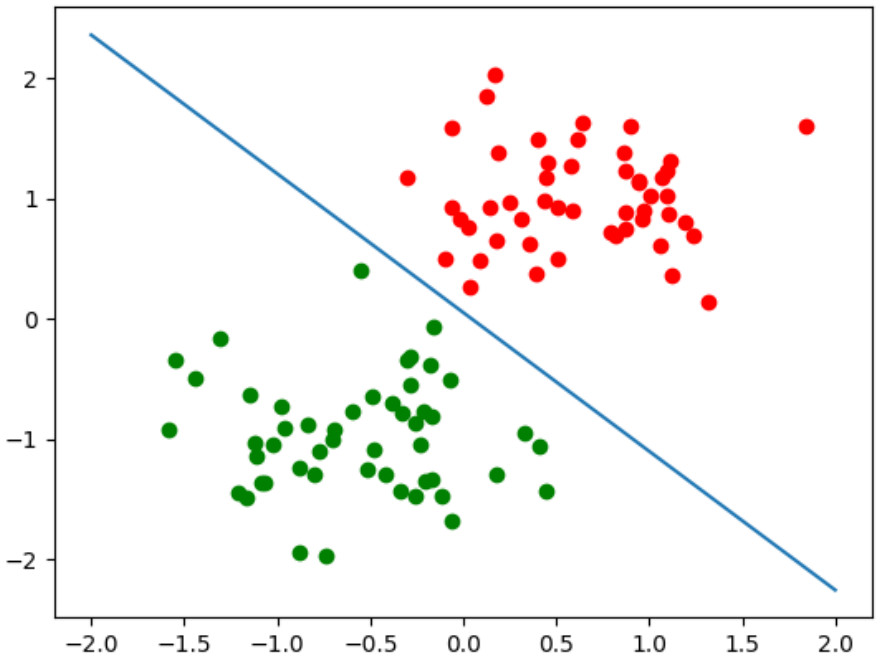
2. 牛顿法

同上面的情况，当样本个数为 10，100，1000 时我们使用牛顿法的分类情况如下：

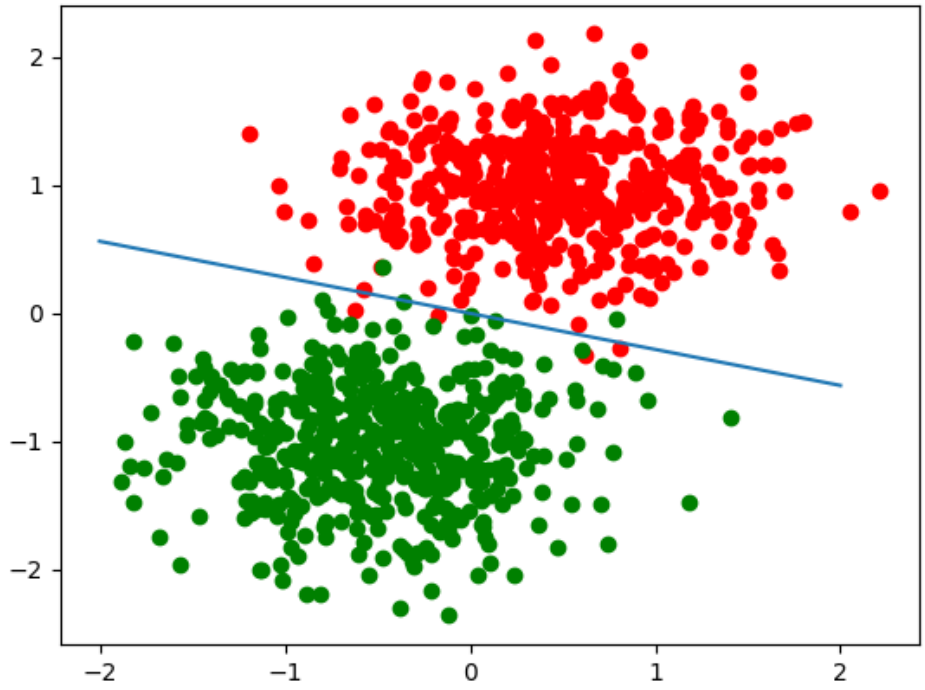
M = 10



M = 100



M = 1000



3. UCI 数据集

在 UCI 机器学习库中，选取了自 2007 年以来点击量第五的心脏病预测数据样本，链接：http://archive.ics.uci.edu/ml/datasets/Heart+Disease

我选取的样本是来自 V.A.长滩和克利夫兰诊所基金会医疗中心：Robert Detrano 博士提供的数据，一共有 76 个属性，但是在已发表的实验中涉及使用 14 个数据的子集。具体属性请参照数据库的说明文档

通过 Pandas 处理数据，在调用我们上面的，带有正则项的梯度下降法，在学习率 alpha=1e-5，threshold=1e-8 的情况下，在惩罚项 lm= 1e-2 1e-3 1e-4 1e-5 1e-6 五种情况下的学习情况：

1) 1e-2

```
共迭代0次
[[-0.37897419 -0.29579697  0.50295372  1.4790188  -0.08469478  0.82524311
   0.59262722  1.59874825  0.01830696  0.1429092   0.67334097 -1.7766117
   0.7269846   0.77860397]]
正确率为0.64
```

根本没有迭代，正确率在 60%左右

2）1e-3

```
共迭代316057次
[[ 0.55188279 -0.15501694  0.58198407  0.69308019  0.98915815  0.5469595
  -0.31158934  1.34721346 -0.24155849  0.67363017  2.11417458  0.70769172
   0.94041239  2.28218609]]
正确率为0.8533333333333334
```

这种情况下，学习率还可以

3）1e-4

```
共迭代475399次
[[ 1.57445215 -0.19600463  0.2221927   0.78419388  0.02323568  1.10319787
   0.1069689   2.04465984 -1.68492249  1.25155075  2.55730601  0.91536884
   2.22450229  2.07233624]]
正确率为0.7466666666666667
```

4) 1e-5

```
共迭代296824次
[[ 1.9395734   0.06095027 -0.52311699  1.10521356  1.01106194 -1.3007105
   0.71849527  0.27344113 -0.59548795  0.57560639  1.09246866 -0.28765459
   1.68273687  1.83624617]]
正确率为0.7066666666666667
```

5) 1e-6

```
共迭代342963次
[[ 1.93437014 -1.10053544 -0.09784211  1.16754343  1.21456045  0.01192689
  -1.19328436  0.71336073 -0.36266394  2.04447275  2.77462613 -0.02661131
   1.8249729   1.93685536]]
正确率为0.8266666666666667
```

发现学习率过高和过低都会导致正确率的下降

五、结论

1. 在样本数量很少时，加入正则项可以有效减轻过拟合问题。但随着样本数量的增加，区别不是特别明显
2. 在样本分布不满足朴素贝叶斯的条件下，逻辑回归效果会稍微差一点
3. 对于牛顿法来说，收敛速度较快
4. MLE 与 MAP 在数据量大时差别不大（对应第一条）
5.

六、参考文献

周志华 著. 机器学习, 北京: 清华大学出版社, 2016.1
李航 著. 统计学习方法, 北京: 清华大学出版社, 2019.5
吴恩达 Bilibili 网课

七、附录：源代码（带注释）

1. logistic_regression 逻辑回归

```
1.  import matplotlib
2. import numpy as np
3. import matplotlib.pyplot as plt
4. matplotlib.use("WebAgg")
5.
6.
7. def create_data_naive_bayesian(m, is_naive):
8.     positive_mean = [0.5, 1]
9.     positive_rate = 0.5
10.     negative_mean = [-0.5, -1]
11.     negative_rate = 1 - positive_rate
12.     X = np.zeros((m, 2))
13.     y = np.zeros(m)
14.     cov_11 = 0.3
15.     cov_22 = 0.2
16.     if is_naive == 1:
17.         cov_12 = cov_21 = 0
18.     else:
19.         cov_12 = cov_21 = 0.1
20.     cov = [[cov_11, cov_12], [cov_21, cov_22]]      # X 的两个维度
    的协方差矩阵
21.     positive_num = np.ceil(positive_rate * m).astype(np.int32)
22.     negative_num = np.ceil(negative_rate * m).astype(np.int32)
```

```
    X[:positive_num, :] = np.random.multivariate_normal(positive_mean, cov, size=positive_num)      # 根据协方差矩阵生成正态分布
    X[positive_num:, :] = np.random.multivariate_normal(negative_mean, cov, size=negative_num)      # 将正类反类区分开
    y[:positive_num] = 1
    y[positive_num:] = 0
    plt.scatter(X[:positive_num, 0], X[:positive_num, 1], c='r')
    plt.scatter(X[positive_num:, 0], X[positive_num:, 1], c='g')

    return X, y


def sigmoid(X):
    return 1 / (1 + np.exp(-X))


def hypothesis(theta, X):
    return sigmoid(X @ theta.T)


def cost_function(X, y, theta):
    log_likelihood = np.sum(y.reshape(1, -1) @ np.log(hypothesis(theta, X)))
    return -(log_likelihood / X.shape[0])


def cost_function_reg(X, y, theta, lm):
    log_likelihood = np.sum(y.reshape(1, -1) @ np.log(hypothesis(theta, X)))
    reg = np.sum(theta @ theta.T)
    log_likelihood = log_likelihood - (lm / 2) * reg
    return -(log_likelihood / X.shape[0])


def gradient(X, y, theta):
    return (hypothesis(theta, X).reshape(-1, 1) - y.reshape(-1, 1)).T @ X / X.shape[0]


def hessian(X, theta):
    h = hypothesis(theta, X)
    value = np.sum(h.T @ h)
```

```
60.     hessian_matrix = value * X.T @ X
61.     return hessian_matrix / X.shape[0]
62.
63.
64. def gradient_reg(X, y, theta, lm):
65.     grad = (hypothesis(theta, X).reshape(-1, 1) - y.reshape(-1,
    1)).T @ X / X.shape[0]
66.     return grad + (lm / X.shape[0]) * theta
67.
68.
69. def lr_grad_descent(X, y, alpha, threshold, is_reg, lm=1e-4):
70.     X = np.insert(X, 0, np.ones(X.shape[0]), axis=1)
71.     theta = np.random.randn(X.shape[1])
72.     cost_last = cost_function(X, y, theta)
73.     cost = [cost_last]
74.     iter_num = 0
75.     while True:
76.         if is_reg == 0:
77.             grad = gradient(X, y, theta)
78.             theta = theta - alpha * grad
79.             cost_now = cost_function(X, y, theta)
80.             if cost_last - cost_now < threshold:
81.                 break
82.             print(cost_now)
83.             cost.append(cost_now)
84.             cost_last = cost_now
85.             iter_num = iter_num + 1
86.         else:
87.             grad = gradient_reg(X, y, theta, lm)
88.             theta = theta - alpha * grad
89.             cost_now = cost_function_reg(X, y, theta, lm)
90.             if cost_last - cost_now < threshold:
91.                 break
92.             print(cost_now)
93.             cost.append(cost_now)
94.             cost_last = cost_now
95.             iter_num = iter_num + 1
96.     print("共迭代" + str(iter_num) + "次")
97.     return theta, cost, iter_num
98.
99.
100.    def lr_newton(X, y, threshold):
101.        X = np.insert(X, 0, np.ones(X.shape[0]), axis=1)
102.        theta = np.random.randn(X.shape[1]).reshape(1, -1)
```

```
        cost_last = cost_function(X, y, theta)
        cost = [cost_last]
        iter_num = 0
        while True:
            grad = gradient(X, y, theta)
            hessian_matrix = hessian(X, theta)
            theta = theta - grad @ np.linalg.pinv(hessian_matrix)
            cost_now = cost_function(X, y, theta)
            cost.append(cost_now)
            print(cost_now)
            iter_num = iter_num + 1
            if cost_last - cost_now < threshold:
                break
            cost_last = cost_now
        print("共迭代" + str(iter_num) + "次")
        return theta, cost, iter_num


    def draw(theta):
        test_data = np.linspace(-2, 2, 100)
        plt.figure(1)
        plt.plot(test_data, -(theta[:, 0] + theta[:, 1] * test_data)
    / theta[:, 2])
        plt.show()


    def main():
        X, y = create_data_naive_bayesian(1000, 1)
        alpha = 1e-5
        threshold = 1e-8
        theta, cost, iter_num = lr_grad_descent(X, y, alpha,
    threshold, 0)        # 梯度下降法，带正则项
        theta, cost, iter_num = lr_newton(X, y, threshold)
    # 牛顿法
        draw(theta)


    if __name__ == '__main__':
        main()

```

2. heart_disease_predict 心脏病预测

```
1. from sklearn.linear_model import LogisticRegression
2. from sklearn.model_selection import train_test_split
3. from sklearn.preprocessing import StandardScaler
4. import logistic_regression as lr
5. import pandas as pd
6. import numpy as np
7.
8.
9. def logistic_UCI_data():
10.     column_1 = ["age", "sex", "cp", "trestbps", "chol", "fbs",
   "restecg"]
11.     column_2 = ["thalach", "exang", "oldpeak", "slope", "ca",
   "thal", "num"]
12.     column = column_1 + column_2
13.     data = pd.read_csv("data/processed.cleveland.csv",
   engine='python', names=column)
14.     data = (data.replace(to_replace="?", value=np.nan)).dropna()
15.     X_train, X_test, y_train, y_test =
   train_test_split(data[column[0:13]], data[column[13]],
   test_size=0.25)
16.     std = StandardScaler()
17.
18.     X_train = std.fit_transform(X_train)
19.     X_test = std.transform(X_test)
20.     theta, cost, iter_num = lr.lr_grad_descent(X_train,
   y_train.array.to_numpy(), 1e-5, 1e-8, 1, lm=1e-6)
21.     print(theta)
22.     # lg = LogisticRegression(C=1.0)
23.     # lg.fit(X_train, y_train)
24.     # y_predict = lg.predict(X_test)
25.     y_predict = fit(theta, X_test)
26.     rate = cal_rate(y_predict, y_test)
27.     fit(theta, X_test)
28.     print("正确率为" + str(rate))
29.
30.
31. def fit(theta, X_test):
32.     X_test = np.insert(X_test, 0, np.ones(X_test.shape[0]),
   axis=1)
33.     y = lr.sigmoid(theta @ X_test.T)
34.     y_predict = np.zeros(y.shape[1])
35.     for i in range(y.shape[1]):
```

```
36.         if np.sum(y[:, i]) >= 0.5:
37.             y_predict[i] = 1
38.         else:
39.             y_predict[i] = 0
40.     return y_predict
41.
42.
43. def cal_rate(y_predict, y_test):
44.     n = y_predict.size
45.     count = 0
46.     for i in range(n):
47.         if y_predict[i] == y_test.array[i] and y_predict[i] == 0:
48.             count = count + 1
49.         if y_predict[i] > 0 and y_test.array[i] > 0:
50.             count = count + 1
51.     return count / n
52.
53.
54. def main():
55.     logistic_UCI_data()
56.
57.
58. if __name__ == '__main__':
59.     main()
60.
```