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：1190202107
姓名： 姚舜宇

# 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

使用不同的优化方法，实现对有无正则项的两种损失函数的参数估计。

环境：Windows10，python3.9，pycharm

# 三、设计思想（本程序中的用到的主要算法及数据结构）

定义数据集$\{(\boldsymbol{x}_i,y_i)\},i\in[1,N]$，$X=\begin{bmatrix}\boldsymbol{x}_1^T\\ \boldsymbol{x}_2^T\\ \vdots\\ \boldsymbol{x}_N^T\end{bmatrix}$，$Y=\begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_N\end{bmatrix}$，$\boldsymbol{x}_i\in R^p$，是$p\times1$列向量，$y_i\in\{0,1\}$。

在生成数据时，为方便图示，使用二维高斯分布进行生成。

根据逻辑回归模型，设$p_1=p(y=1\mid\boldsymbol{x})=\dfrac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}=\phi(\boldsymbol{x})$，$p_2=1-p_1=1-\phi(\boldsymbol{x})$。易得参数$\boldsymbol{w}$的似然为$p(y\mid X,\boldsymbol{w})=\prod_{i=1}^{N}p_1^{y}p_0^{1-y}=\prod_{i=1}^{N}\phi(\boldsymbol{x}_i)^{y_i}(1-\phi(\boldsymbol{x}_i))^{1-y_i}$。根据极大似然估计，可以写出：

$$\begin{aligned}\boldsymbol{w}^*&=arg\ \max_{\boldsymbol{w}}\ \log\prod_{i=1}^{N}\phi(\boldsymbol{x}_i)^{y_i}(1-\phi(\boldsymbol{x}_i))^{1-y_i}\\&=arg\ \max_{\boldsymbol{w}}\ \sum_{i=1}^{N}[y_i\log\phi(\boldsymbol{x}_i)+(1-y_i)\log(1-\phi(\boldsymbol{x}_i))]\\&=arg\ \max_{\boldsymbol{w}}\ \sum_{i=1}^{N}[(y_i-1)\boldsymbol{w}^T\boldsymbol{x}_i-\log(1+e^{-\boldsymbol{w}^T\boldsymbol{x}_i})]\end{aligned}$$

考虑到正则项，可以根据上式写出损失函数：

$$L=\sum_{i=1}^{N}[-y_i\boldsymbol{w}^T\boldsymbol{x}_i+\log(1+e^{\boldsymbol{w}^T\boldsymbol{x}_i})]+\lambda\boldsymbol{w}^T\boldsymbol{w}$$

为求得梯度，两边取微分：

$$\begin{aligned}dL&=\sum_{i=1}^{N}[-y_i(d\boldsymbol{w})^T\boldsymbol{x}_i+\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)(d\boldsymbol{w})^T\boldsymbol{x}_i]+\lambda d\boldsymbol{w}^T\boldsymbol{w}+\lambda\boldsymbol{w}^T d\boldsymbol{w}\\&=tr\left[\left(\sum_{i=1}^{N}(-y_i+\sigma(\boldsymbol{w}^T\boldsymbol{x}_i))\boldsymbol{x}_i^T+2\lambda\boldsymbol{w}^T\right)d\boldsymbol{w}\right]\end{aligned}$$

故梯度$\nabla_{\boldsymbol{w}}(L)=\sum_{i=1}^{N}(\sigma(\boldsymbol{w}^T\boldsymbol{x}_i)-y_i)\boldsymbol{x}_i+2\lambda\boldsymbol{w}$。为使用牛顿法进行优化，需要计算$Hessian$矩阵，为此需要对梯度进行求导。

为求二阶梯度，使用复合求导方法。令$t_i=\boldsymbol{w}^T\boldsymbol{x}_i$，容易得到$\dfrac{\partial t_i}{\partial\boldsymbol{w}}=\boldsymbol{x}_i$。只需求$\dfrac{\partial\nabla_{\boldsymbol{w}}(L)}{\partial t_i}$即

可。$d\nabla_{\boldsymbol{w}}(L)=\sigma'(t_i)dt_i\boldsymbol{x}_i$，得到$\frac{\partial\nabla_{\boldsymbol{w}}(L)}{\partial t_i}=\sigma'(t_i)\boldsymbol{x}_i^T$。

故$\frac{\partial\nabla_{\boldsymbol{w}}(L)}{\partial\boldsymbol{w}}=\sum_{i=1}^{N}\frac{\partial t_i}{\partial\boldsymbol{w}}\frac{\partial\nabla_{\boldsymbol{w}}(L)}{\partial t_i}=\sum_{i=1}^{N}[\boldsymbol{x}_i\sigma'(t_i)\boldsymbol{x}_i^T]$。由于正则项的存在，上式应当修改为

$\frac{\partial\nabla_{\boldsymbol{w}}(L)}{\partial\boldsymbol{w}}=\sum_{i=1}^{N}[\boldsymbol{x}_i\sigma'(t_i)\boldsymbol{x}_i^T]+2\lambda$。向量化得$\nabla_{\boldsymbol{w}}^2(L)=X^T\mathrm{diag}(\sigma'(X\boldsymbol{w}))X+2\lambda$，其中

$\mathrm{diag}(A)$表示用矩阵$A$的元素按列优先排成的对角阵。

计算出损失函数以及其梯度、*Hessian*矩阵后，即可通过梯度下降法和牛顿法进行优化。

### 梯度下降法

梯度下降法的迭代公式为：

$$\boldsymbol{w}_{k+1}=\boldsymbol{w}_k-\alpha\nabla_{\boldsymbol{w}}(L)$$

其中$\alpha$为学习率，为一个超参数。手动选择超参数学习率，惩罚率，迭代轮数，即可使用梯度下降法进行拟合。

### 牛顿法

牛顿法是一种数值优化方法，基本思想是用迭代点的梯度信息和二阶导数对目标函数进行二次函数逼近，然后把二次函数的极小值作为新的迭代点，不断重复这一过程直到求出极小点。在此问题中，迭代公式为：

$$\boldsymbol{w}_{k+1}=\boldsymbol{w}_k-\alpha\nabla_{\boldsymbol{w}}^2(L)\nabla_{\boldsymbol{w}}(L)$$

其中$\alpha$为学习率。

### 共轭梯度法

因为目标函数不是二次型，所以需要使用一般的共轭梯度法。算法流程如下：
初始条件：

$$d_1=r_1=-\nabla_{\boldsymbol{w}}(L)$$

迭代关系：

$$a_k=\frac{d_k^Tr_k}{d_k^THd_k}$$

$$\boldsymbol{w}_{k+1}=\boldsymbol{w}_k+a_kd_k$$

$$r_{k+1}=r_k-a_kHd_k$$

$$b_k=\frac{r_{k+1}^THd_k}{d_k^THd_k}$$

$$d_{k+1}=r_{k+1}-b_kd_k$$

其中 H 为黑塞矩阵。为了尽可能减少迭代次数，在实验中，当两次迭代的损失函数值差的绝对值小于一个较小数时，结束迭代。

通过上述的优化方法求得参数$\boldsymbol{w}^*$后，对于样本特征向量$\hat{\boldsymbol{x}}$，属于 1 类的概率$p_1=\frac{1}{1+e^{-\boldsymbol{w}^T\hat{\boldsymbol{x}}}}$，

属于 0 类的概率$p_0=1-p_1$。则预测此样本属于的类别为两概率较大者所对应的类别。

# 四、实验结果与分析

生成数据要求：不妨设两类数据的样本数量均为 100。需要分别实验满足朴素贝叶斯假设和不满足朴素贝叶斯假设，即生成高斯分布时协方差矩阵是否为对角阵。

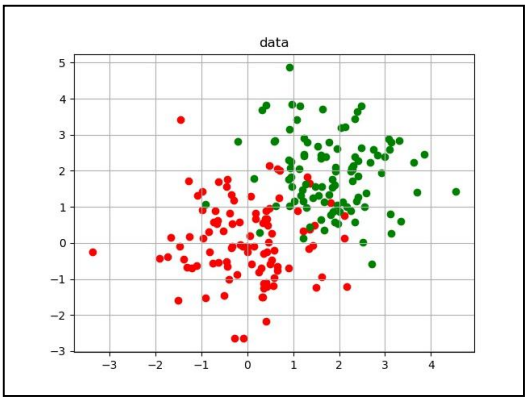
梯度下降法：尝试不同学习率、惩罚率进行测试。

牛顿法：尝试不同学习率、惩罚率进行测试。

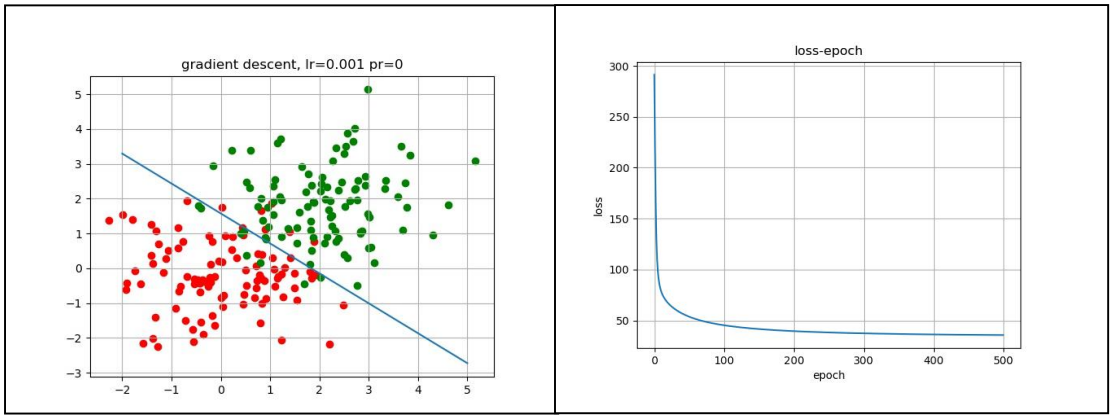
共轭梯度法：尝试不同的惩罚率进行测试。

使用 UCI 网站提供的白葡萄酒分类数据集进行验证。

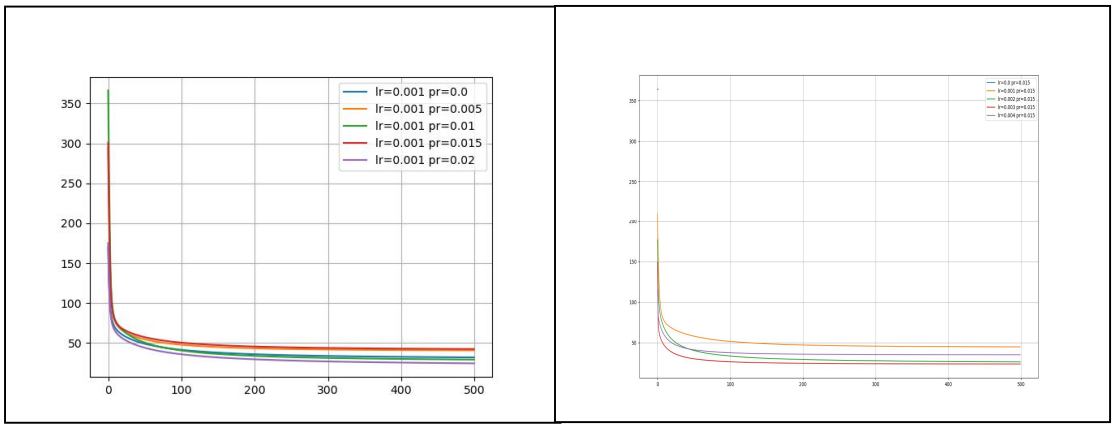
1. 生成两组数据，均服从二维高斯分布。均值分别为[0,0]，[2,2]，协方差矩阵均为单位对角阵，即服从朴素贝叶斯假设。数据点效果如下：




使用梯度下降法进行优化，学习率取 0.001，惩罚率取 0，进行 500 轮迭代。结果如下。



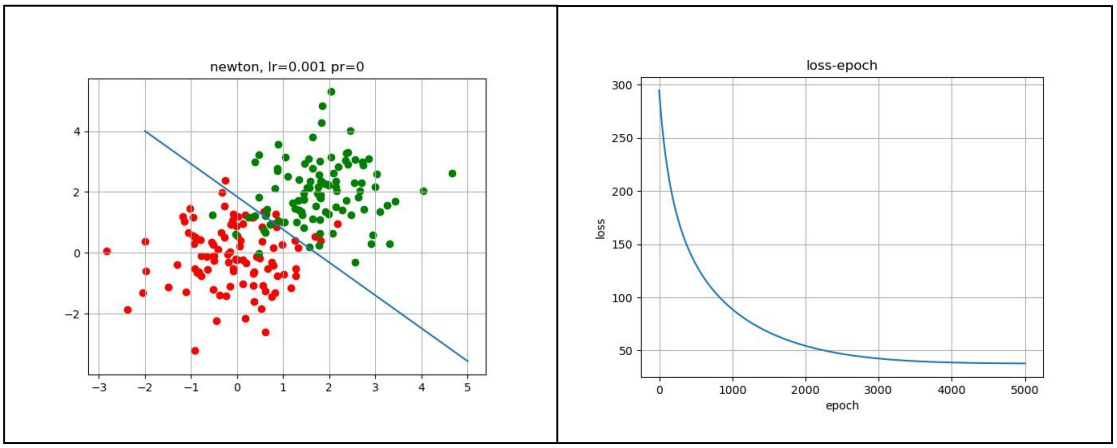

可以看出迭代次数超过 300 时，损失函数基本收敛。为探究不同超参数对学习结果的影响，分别取定学习率为 0.001，惩罚率从 0 变化到 0.02；惩罚率取 0.015，学习率从 0 变化到 0.004，作图如下。



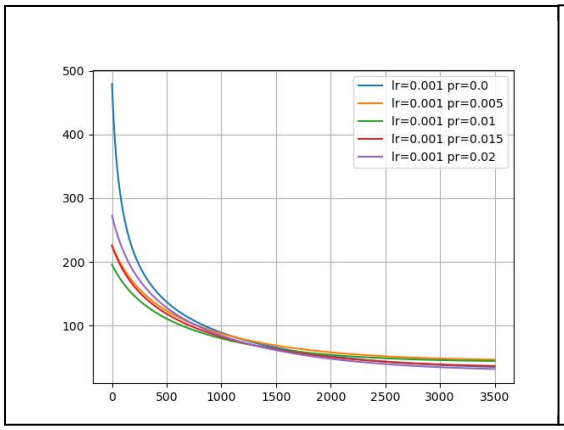

从变化曲线可以看出，当学习率取 0.003，惩罚率取 0.02 时损失函数最小，但差别并不明显。

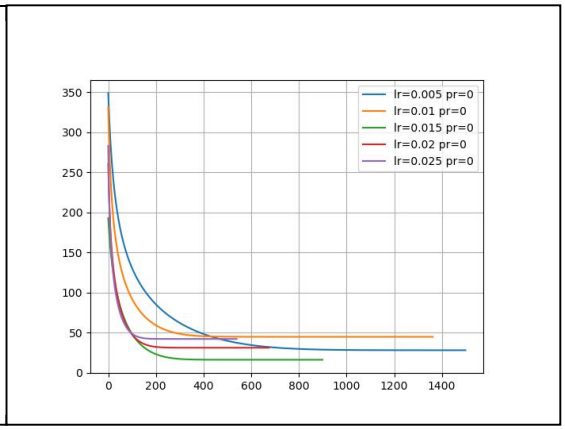
使用学习率为 0.003，惩罚率为 0.02，用梯度下降优化的逻辑回归模型，对同分布的 200 个样本分类，实验 5 次，正确率分别为：96%，91.5%，91.5%，94%，90.5%。可以看出，对上述符合朴素贝叶斯假设的二维高斯分布，使用梯度下降的逻辑分类效果较好。

使用牛顿法进行优化，学习率取 0.001，惩罚率取 0，由于学习率较小，故进行 5000 轮迭代。结果如下。




可以看出迭代次数超过 3000 时，损失函数基本收敛。为探究不同超参数对学习结果的影响，分别取定学习率为 0.001，惩罚率从 0 变化到 0.02；惩罚率取 0，学习率从 0.005 变化到 0.025，作图如下。

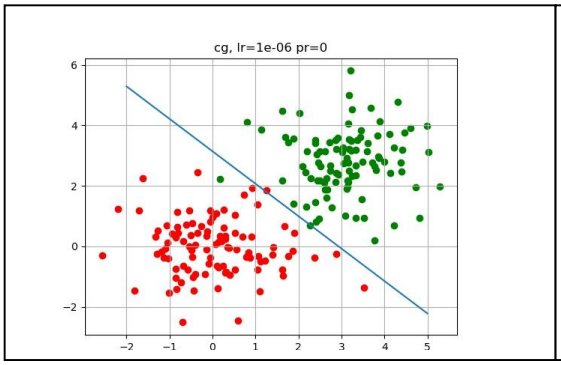


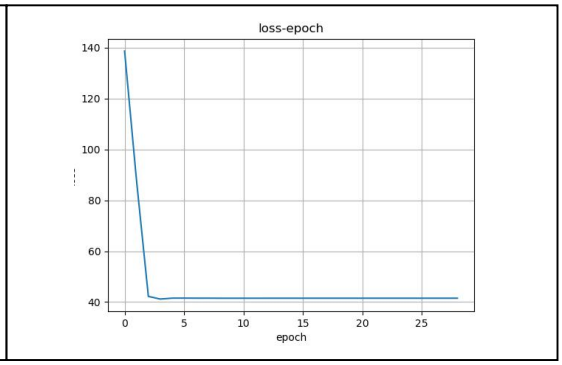




从变化曲线可以看出，惩罚率的变化对模型效果无太大影响；对学习率而言，取 0.015 时效果最佳，同时收敛速度较快。

使用学习率为 0.015，不加惩罚项，用牛顿法优化的逻辑回归模型，对同分布的 200 个样本进行分类，实验 5 次，正确率分别为：92.5%，96%，93%，93%，93%。可以看出，对上述符合朴素贝叶斯假设的二维高斯分布，使用牛顿法的逻辑分类的效果较好，且效果与使用梯度下降法优化相近。

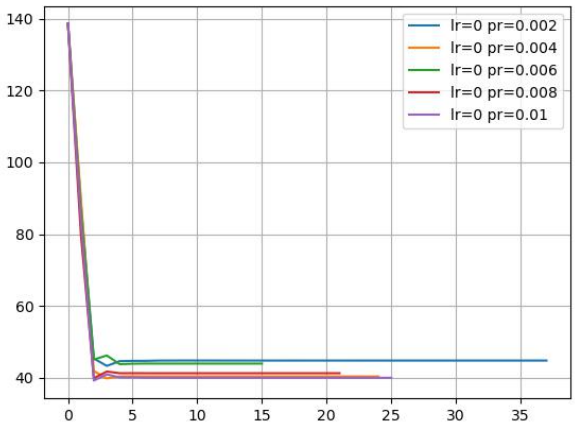
使用共轭梯度法进行优化。进行 50 次迭代，且当两次迭代损失函数差的绝对值不超过 10 的-7 次方时，结束迭代。首先惩罚率取 0。



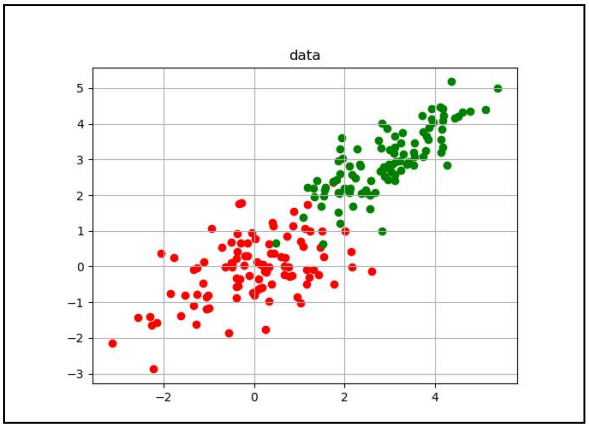

通过结果可以看出，共轭梯度法的迭代速度非常快，超过 3 到 4 次迭代之后损失函数就几乎没有变化。并且左图能够看出分类效果较优。

为探究不同超参数对学习结果的影响，取惩罚率从 0.002 变化到 0.01，绘制出损失函数随迭代次数变化曲线。

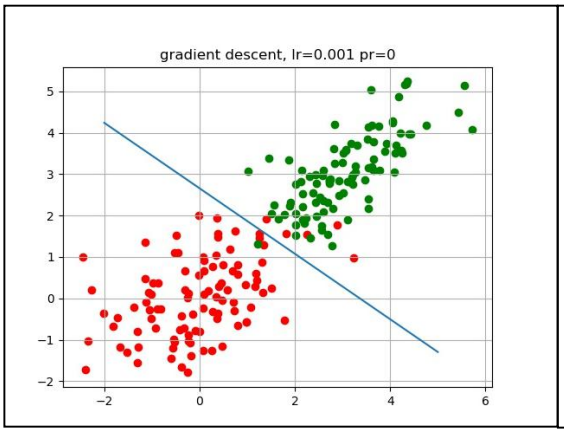


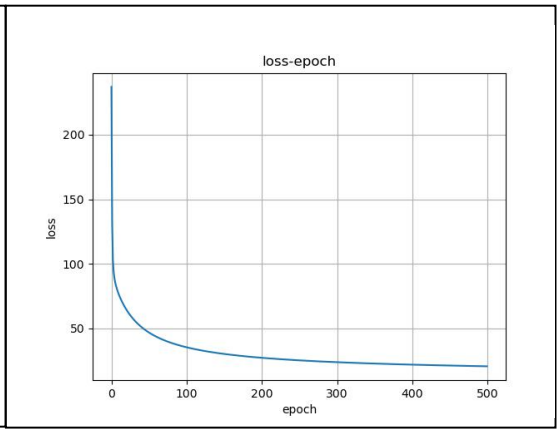
从图中可以看出，惩罚率为 0.01 时损失函数最小。经过多次实验，损失函数最小时对应的惩罚率并不相同，所以可以猜想共轭梯度法和惩罚项没有太大关系。
使用惩罚率为 0.005，用共轭梯度优化的逻辑回归模型，对同分布的 200 个样本分类，实验 5 次，正确率分别为：94.5%，91%，93%，92.5%，93%。可以看出，对上述符合朴素贝叶斯假设的二维高斯分布，使用共轭梯度的逻辑分类效果较好。

2. 生成两组数据，均服从二维高斯分布。均值分别为[0,0]，[3,3]，协方差矩阵均为 $\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$，即不服从朴素贝叶斯假设。数据点效果如下：

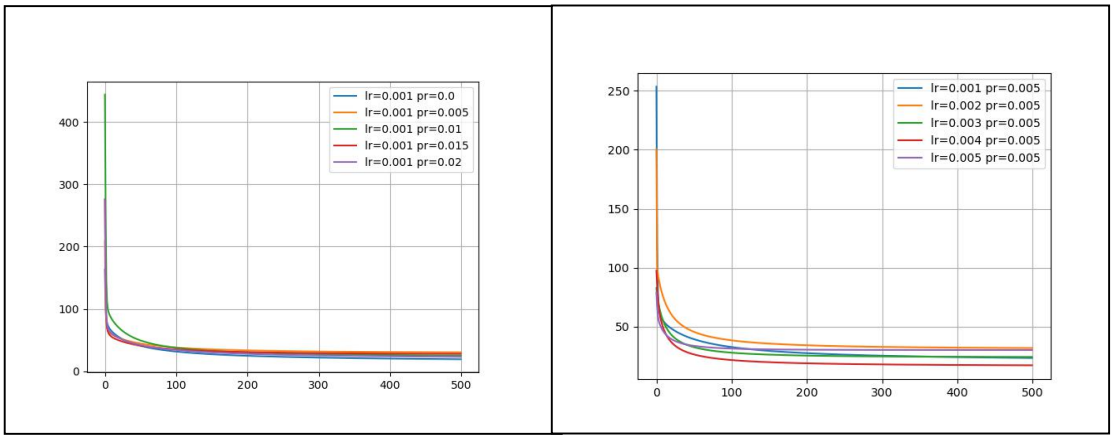



使用梯度下降法进行优化，学习率取 0.001，惩罚率取 0，进行 500 轮迭代。结果如下。



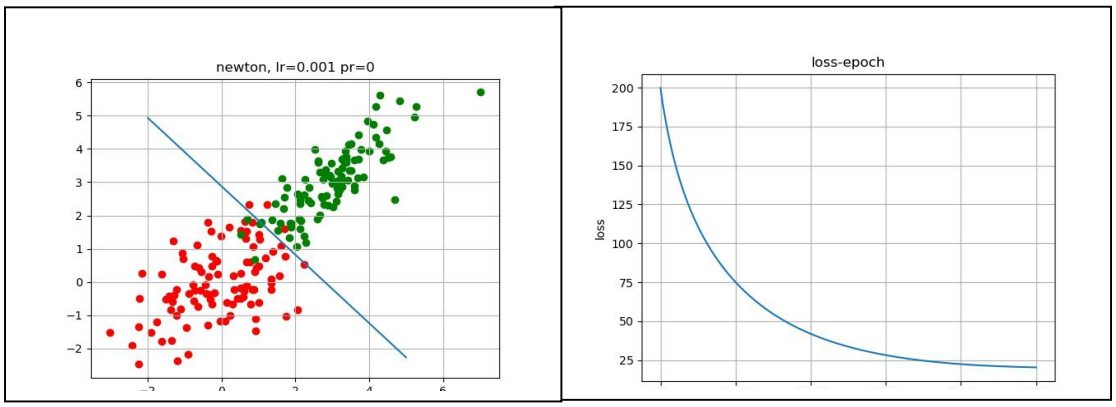

可以看出迭代次数约 500 时，损失函数基本收敛。为探究不同超参数对学习结果的影响，分别取定学习率为 0.001，惩罚率从 0 变化到 0.02；惩罚率取 0.005，学习率从 0 变化到 0.004，作图如下。



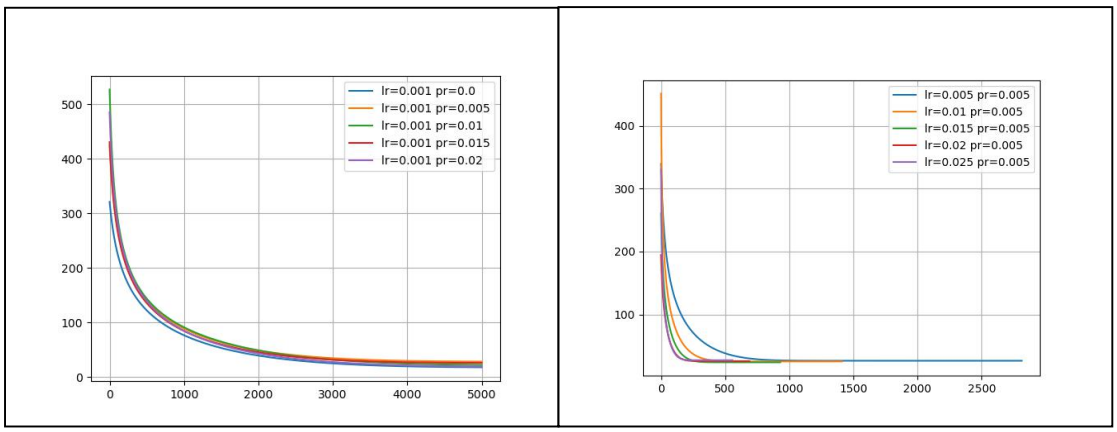

从变化曲线可以看出，当学习率取 0.004 时损失函数最小，惩罚率对损失函数影响较小。

使用学习率为 0.004，不加惩罚项，用梯度下降优化的逻辑回归模型，对同分布的 200 个样本分类，实验 5 次，正确率分别为：97%，97.5%，97%，95.5%，96.5%。可以看出，对上述符合朴素贝叶斯假设的二维高斯分布，使用梯度下降的逻辑分类效果较好。因为修改了两种样本的均值，所以无法和不符合朴素贝叶斯假设的二维高斯分布进行对比，但可以推断，在一定范围内，是否符合朴素贝叶斯假设对逻辑分类效果无明显影响。

使用牛顿法进行优化，学习率取 0.001，惩罚率取 0，进行 5000 轮迭代。结果如下。




可以看出迭代次数超过 4000 时，损失函数基本收敛。为探究不同超参数对学习结果的影响，分别取定学习率为 0.001，惩罚率从 0 变化到 0.02；惩罚率取 0.005，学习率从 0.005 变化到 0.025，作图如下。

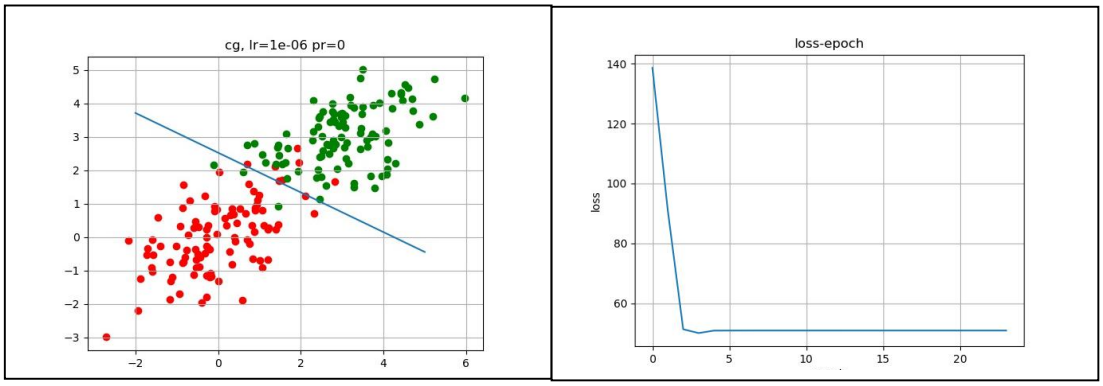



从变化曲线可以看出，惩罚率的变化对模型效果无太大影响；对学习率而言，取

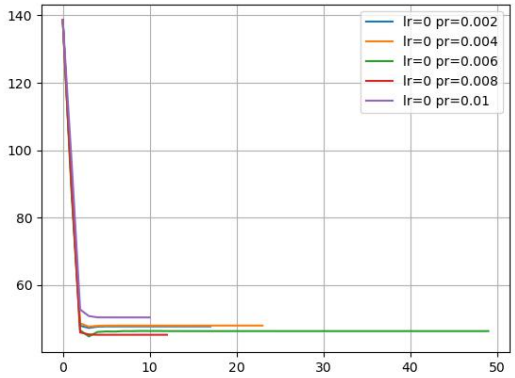
0.015 时效果最佳，同时收敛速度最快。

使用学习率为 0.015，不加惩罚项，用牛顿法优化的逻辑回归模型，对同分布的 200 个样本进行分类，实验 5 次，正确率分别为：96%，94.5%，94%，94%，94%。可以看出，对上述符合朴素贝叶斯假设的二维高斯分布，使用牛顿法的逻辑分类的效果较好，且效果与使用梯度下降法优化基本相近。

使用共轭梯度法进行优化，惩罚率取 0.005，进行 50 轮迭代。结果如下。




可以看出迭代次数到达 3 到 4 时，损失函数基本收敛。为探究不同超参数对学习结果的影响，取惩罚率从 0 变化到 0.01 作图如下。




从上图中来看，惩罚率取 0.008 时损失函数最小。但经过多次实验，损失函数最小时对应的惩罚率并不相同，所以可以猜想共轭梯度法和惩罚项没有太大关系。

使用惩罚率为 0.005，用共轭梯度优化的逻辑回归模型，对同分布的 200 个样本分类，实验 5 次，正确率分别为：95.5%，95.5%，95.5%，94%，97.5%。可以看出，对上述符合朴素贝叶斯假设的二维高斯分布，使用共轭梯度法的逻辑分类的效果较好，且效果与使用梯度下降法以及牛顿法优化基本相近。

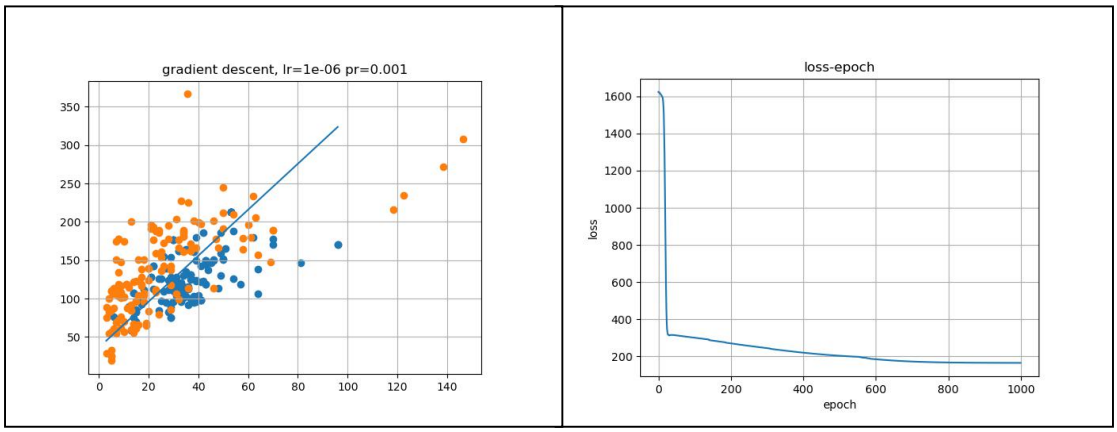
3. 使用 UCI 网站提供给数据集进行训练和测试。

选择 Wine Quality Data Set 中的 winequality-white.csv。由于此实验中的逻辑分类为二分类模型，所以首先对数据进行处理，选择两种类别的数据进行实验。实验中使用数据的前 75%进行训练，共 272 个样本。测试集为后 25%，共 91 个样本。实验发现，需要选择非常小的学习率，才可以有效使用梯度下降。将学习率定为 1e-7，惩罚率定为 0.001，进行 150000 次迭代，保存学习到的参数。使用后 25%的数据进行测试，正确率约为 66%。

结果分析：首先样本量略少，难以学习到一个更优的模型参数。然后在实验过程中，最

后一次迭代时损失函数仍然在下降，即参数没有下降到最优。考虑到时间原因，很难使用更多的迭代轮数，故学习率难以减小。并且根据多次实验过程，学习率更大时，容易在最优解附近跳动，无法收敛。另外，样本的特征数量为 11，用线性分类算法很难获得比较好的分类效果。因为特征维数原因，无法进行可视化。

为直观观测实验结果，我计算了样本各个维度的方差，取方差最大的两个特征保留，进行降维。迭代 1000 次，学习率定为 1e-6，惩罚率定为 0.001，进行实验。结果如下图。




从左图中可以看出，两类样本重合率较高，对于线性分类效果一般较差。经实验，使用测试集进行测试，预测准确率约为 65%，和没有进行降维时近似。所以认为该数据集不适合使用线性分类算法。

# 五、结论

逻辑回归是一种简单的分类算法，在假定$P(x|y_k)$服从高斯分布的情况下可以推导出逻辑回归的一般公式$p_1=p(y=1|\boldsymbol{x})=\frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}=\phi(\boldsymbol{x})$，$p_2=1-p_1=1-\phi(\boldsymbol{x})$。可以通过极大似然法估计出参数的表示形式，在使用解析解方法或其他优化算法学习到参数。对于给定的一个样本，计算它是某一类的概率，取最大者，认为该样本为这一类。

对于优化算法，发现效果相近，但共轭梯度法收敛速度最快。因为在每一轮迭代中，它能够求出每一维度的参数的最优值，从而理论上在线性与参数维度的时间内可以得到最优解。

# 六、参考文献

《统计学习方法》李航
一般目标函数的共轭梯度法 https://blog.csdn.net/tu__zi/article/details/105496968

# 七、附录：源代码（带注释）

```
1.  import numpy as np
2.  import matplotlib.pyplot as plt
3.  import pandas as pd
4.
5.  # the parameters
6.  # 第一类数据的生成参数
```

```
num1 = 100
mean1 = [0, 0]
cov1 = [[1, 0.5], [0.5, 1]]
# 第二类数据的生成参数
num2 = 100
mean2 = [3, 3]
cov2 = [[1, 0.5], [0.5, 1]]
start = -2
end = 5

epoch = 50
learning_rate = 0.000001  # 学习率
penalty_rate = 0.001  # 惩罚率
loss_list = []
epoch_list = []


def Generate_data(draw=True):
    """
    generate two types of data that obey Gaussian distribution
    :return:
    """
    data1 = np.random.multivariate_normal(mean1, cov1, num1)  # 生成高维高斯分布数据
    data1 = np.c_[data1, np.ones((num1, 1))]
    y1 = np.array([1] * num1)
    y1 = y1.reshape((num1, -1))
    data2 = np.random.multivariate_normal(mean2, cov2, num2)
    data2 = np.c_[data2, np.ones((num2, 1))]
    y2 = np.array([0] * num2)
    y2 = y2.reshape((num2, -1))
    if draw:  # 画出样本散点图
        plt.scatter(data1[:, 0], data1[:, 1], c='r')
        plt.scatter(data2[:, 0], data2[:, 1], c='g')
    # plt.grid()
    # plt.title('data')
    # plt.savefig('data.jpg')
    # plt.show()
    # exit(0)
    X = np.mat(np.vstack((data1, data2)))
    Y = np.mat(np.vstack((y1, y2)))
    return X, Y


```

```python
def sigmoid(z):
    """
    sigmoid function
    :param z: input
    :return: sigmoid(z)
    """
    # sigmoid 函数，当自变量过大或过小时，直接返回结果
    if z > 10:
        return 1 - 1e-5
    elif z < -10:
        return 1e-5
    return 1 / (1 + np.exp(- z))


def loss(X, Y, w, pr=float(0)):
    """
    calculate the loss function
    :param X: X
    :param Y: Y
    :param w: the parameters
    :param pr: penalty rate
    :return: the loss
    """
    res = 0
    for i in range(X.shape[0]):
        yi = np.array(Y[i]).flatten()[0]
        sig = np.array(sigmoid(w.T @ X[i].T)).flatten()[0]
        if yi == 1:
            res += - np.log(sig)
        else:
            res += - np.log(1 - sig)
    res += pr * np.array(w.T @ w).flatten()[0]
    return res


def grad(w, X, Y, pr):  # 计算梯度
    """
    calculate the gradient
    :param w: the parameters
    :param X: X
    :param Y: Y
    :param pr: penalty rate
    :return: the gradient matrix
    """
```

```
    sig = 1 / (1 + np.exp(- X @ w))
    g = X.T @ (sig - Y)
    return g + np.multiply(2 * pr, w)


def Hessian(w, X, pr):  # 计算海森矩阵
    """
    calculate the Hessian matrix
    :param w: the parameters
    :param X: X
    :param pr: penalty rate
    :return: Hessian matrix
    """
    Z = X @ w
    expZ = np.exp(Z)
    p = np.mat(np.ones((expZ.shape[0], expZ.shape[1]))) + expZ
    p = np.multiply(p, p)
    diff_sig = expZ / p
    diag = np.mat(np.identity(X.shape[0]))
    for j in range(0, diff_sig.shape[0]):
        diag[j, j] = diff_sig[j, 0]
    H = X.T @ diag @ X
    return H + np.multiply(2 * pr, w)


def Gradient_descent(X, Y, epoch=10000, lr=0.003, pr=0.001):
    """
    the gradient descent optimization
    :param X: X
    :param Y: Y
    :param epoch: epoch
    :param lr: learning rate
    :param pr: penalty rate
    :return: the parameters this algorithm learned
    """
    w = np.mat(np.random.random(size=(X.shape[1], 1)))
    eplison = 1e-7
    old_loss = 0
    epoch_list.clear()
    loss_list.clear()
    for i in range(epoch):
        w -= lr * grad(w, X, Y, pr)  # 迭代公式
        l = loss(X, Y, w, pr)
        loss_list.append(l)
```

```
        epoch_list.append(i)
        if abs(l - old_loss) < eplison:
            break
        old_loss = l
        print('epoch ' + str(i) + '  loss=' + str(l))
    return w


def Newton(X, Y, epoch=10000, lr=0.001, pr=0.001):
    """
    the newton optimization
    :param X: X
    :param Y: Y
    :param epoch: epoch
    :param lr: learning rate
    :param pr: penalty rate
    :return: the parameters this algorithm learned
    """
    w = np.mat(np.random.random(size=(X.shape[1], 1)))
    eplison = 1e-7
    epoch_list.clear()
    loss_list.clear()
    for i in range(epoch):
        g = grad(w, X, Y, pr)
        if np.sum(g.T @ g) < eplison:
            break
        w -= np.multiply(lr, Hessian(w, X, pr).I @ g)  # 迭代公式
        l = loss(X, Y, w, pr)
        loss_list.append(l)
        epoch_list.append(i)
        print('epoch ' + str(i) + '  loss=' + str(l))
    return w


def Conjugate_gradient(X, Y, epoch=10, pr=float(0)):
    """
    the conjugate gradient optimization
    :param X: X
    :param Y: Y
    :param epoch: epoch
    :param pr: penalty rate
    :return: the parameters this algorithm learned
    """
    w = np.mat(np.zeros((X.shape[1], 1)))
```

```
    g = grad(w, X, Y, pr)
    dk = rk = -g
    l = 0
    eplison = 1e-7
    epoch_list.clear()
    loss_list.clear()
    for i in range(epoch):
        H = Hessian(w, X, pr)
        ak = (dk.T @ rk).tolist()[0][0] / (dk.T @ H @ dk).tolist()[0][0]  # 计算步长
        new_w = w + np.multiply(ak, dk)  # 迭代公式
        new_loss = loss(X, Y, w, pr)
        if abs(new_loss - l) < eplison:
            w = new_w
            break
        l = new_loss
        loss_list.append(l)
        epoch_list.append(i)
        print('epoch ' + str(i) + '  loss=' + str(l))
        w = new_w
        rk = rk - np.multiply(ak, H @ dk)
        bk = (rk.T @ H @ dk).tolist()[0][0] / (dk.T @ H @ dk).tolist()[0][0]
        dk = rk - np.multiply(bk, dk)  # 计算方向
    return w


def draw_Classifier(w):  # 画出分类面
    """
    draw classifier curve
    :param w: the parameters
    :return: none
    """
    x = np.linspace(start, end, 1000)
    y = (-w[0] * x - w[2]) / w[1]
    plt.plot(x, y)


def train(method, dataset='my', lr=learning_rate, pr=penalty_rate, draw=True):
    """
    train a logistic regression model
    :param method: the optimization algorithm
    :param dataset: dataset
```

```
    :param lr: learning rate
    :param pr: penalty rate
    :param draw: if you want to draw a picture
    :return: the parameters this model learned
    """
    if dataset == 'my':
        X, Y = Generate_data()  # 生成画图的数据
        # X, Y = Generate_data(False)  # 生成不画图的数据
    else:  # 读入文件进行训练
        data = pd.read_csv(dataset)
        data_num = int(data.shape[0] * 0.75)
        dim = data.shape[1]
        X = np.mat(np.ones((data_num, dim)))
        Y = np.mat(np.ones((data_num, 1)))
        for i in range(data_num):
            for j in range(dim - 1):
                X[i, j] = data[data.columns[j]][i]
            Y[i] = data[data.columns[dim - 1]][i]
        if draw and dim == 3:
            x1 = []
            x2 = []
            y1 = []
            y2 = []
            for i in range(data_num):
                if Y[i] == 1:
                    x1.append(X[i, 0])
                    y1.append(X[i, 1])
                else:
                    x2.append(X[i, 0])
                    y2.append(X[i, 1])
            global start
            start = min(x1[:]) if min(x1[:]) < min(x2[:]) else min(x2[:])
            global end
            end = max(x1[:]) if max(x1[:]) < max(x2[:]) else max(x2[:])
            plt.scatter(x1, y1)
            plt.scatter(x2, y2)
    w = 0
    if method == 'gd':
        w = Gradient_descent(X, Y, epoch=epoch, lr=lr, pr=pr)  # 使用梯度下
降法优化
        method = 'gradient descent'
    elif method == 'nt':
        w = Newton(X, Y, epoch=epoch, lr=lr, pr=pr)  # 使用牛顿法优化
        method = 'newton'
```

```
    elif method == 'cg':
        w = Conjugate_gradient(X, Y, epoch=epoch, pr=pr)  # 使用共轭梯度法优化
        method = 'conjugate gradient'
    else:
        print('wrong')
        exit(0)
    w = np.array(w).flatten()
    f = open('parameter.txt', 'a')
    f.write(str(method) + ', epoch=' + str(epoch) + ' lr=' + str(lr) + ' pr=' + str(
        pr) + ' w=' + str(w) + '\n')
    if not draw:
        return w

    draw_Classifier(w)
    plt.grid()
    plt.title(str(method) + ', lr=' + str(lr) + ' pr=' + str(pr))
    plt.savefig(str(method) + '.jpg')
    plt.show()

    plt.plot(epoch_list, loss_list, label='lr=' + str(lr) + ' pr=' + str(pr))  # 画出损失函数随迭代次数变化的曲线

    plt.grid()
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.title('loss-epoch')
    plt.show()
    return w


def test(w, dataset='my'):
    """
    use test set data to test the model
    :param w: the parameters
    :param dataset: the dataset
    :return: none
    """
    w = np.mat(w).T
    cnt = 0
    if dataset == 'my':
        data_num = num1 + num2
        X, Y = Generate_data()
```

```
307.         for i in range(X.shape[0]):
308.             p1 = sigmoid(X[i] @ w)
309.             if p1 > 0.5 and Y[i] == 1:
310.                 cnt += 1
311.             if p1 < 0.5 and Y[i] == 0:
312.                 cnt += 1
313.     else:
314.         data = pd.read_csv(dataset)
315.         data_sum_num = data.shape[0]
316.         data_num = int(data.shape[0] * 0.25)
317.         dim = data.shape[1]
318.         X = np.mat(np.ones((data_num, dim)))
319.         Y = np.mat(np.ones((data_num, 1)))
320.         for i in range(data_num):
321.             for j in range(dim - 1):
322.                 X[i, j] = data[data.columns[j]][data_sum_num - data_num + i
    ]
323.             Y[i] = data[data.columns[dim - 1]][data_sum_num - data_num + i]

324.         for i in range(data_num):
325.             p1 = sigmoid(X[i] @ w)
326.             if p1 > 0.5 and Y[i] == 1:
327.                 cnt += 1
328.             if p1 < 0.5 and Y[i] == 0:
329.                 cnt += 1
330.
331.     print('correctness: ' + str(100 * cnt / data_num) + '%')
332.
333.
334. # w = train('gd', dataset='my', lr=0.004, pr=0)
335. # w = train('nt', dataset='my', lr=0.015, pr=0)
336. w = train('cg', dataset='my', pr=0, draw=True)
337.
338.
339. # for i in range(5):
340. #     train('cg', lr=0, pr=round(0.002*(i+1), 4))
341. # plt.grid()
342. # plt.legend()
343. # plt.show()
344.
345. # test(w)
346.
347.
348. # w = train('gd', dataset='white_wine.csv', draw=False)
```

```
349.
350. # w = [-0.6571559, 0.80784953, 0.38885334, 0.10746713, 0.80613137, 0.051226
    23,
351. #      -0.02438892, 0.52578626, 0.04262576, 0.47074137, 0.32694792, 0.85998
    691]
352. # test(w, 'white_wine.csv')
353.
354.
355. # w = train('gd', dataset='white_wine_dr.csv')
356. # test(w, 'white_wine_dr.csv')
```