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

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：硬件环境：x64CPU

软件环境：pycharm

库版本号：anaconda python==3.6, numpy==1.17.0

三、设计思想（本程序中的用到的主要算法及数据结构）

3.1 逻辑回归的表达式推导

在二分类的情况下，推导逻辑回归表达式。令 $X=<X_1,X_2,...,X_n>, Y\in\{0,1\}$，则有：

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

如果符合朴素贝叶斯假设，X 的每个维度 $X_1,X_2,...,X_n$ 之间相互独立，则有

$$
P(X|Y=0)=\prod_{i=1}^{n}P(X=X_i|Y=0)
$$

$$
P(X|Y=1)=\prod_{i=1}^{n}P(X=X_i|Y=1)
$$

代入得

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

化简得

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

若 $X_1,X_2,...,X_n$ 符合高斯分布 $N(\mu_{ik},\sigma_i)$，Y 符合伯努利分布，即：

$$P(X_i|Y=0)=\frac{e^{-\frac{(X_i-\mu_{i0})^2}{2\sigma_i^2}}}{\sqrt{2\pi}\sigma_i}$$

$$P(X_i|Y=1)=\frac{e^{-\frac{(X_i-\mu_{i1})^2}{2\sigma_i^2}}}{\sqrt{2\pi}\sigma_i}$$

$$P(Y=1)=\pi$$

$$P(Y=0)=1-\pi$$

代入原式得，

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

提出常数项后，上式可表示为这样的形式：

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

因此有

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

逻辑回归的基本思想即为通过上面两个公式求算分类概率，基本方法即为通过训练数据 X 估计系数 $w_i(i=0,1,...,n)$，从而对测试数据进行分类。

3.2 多分类的逻辑回归

对于多分类任务，逻辑回归同样适用。令 $X=<X_1,X_2,...,X_n>,Y\in\{y_1,y_2,...,y_R\}$，则估计分类概率的公式可以表达为：

$$P(y=y_k \mid X)=\frac{\exp(\sum_{i=1}^{n} w_{ki}X_{ki}+w_{k0})}{1+\sum_{j}^{R-1}\exp(\sum_{i=1}^{n} w_{ji}X_i+w_{j0})} \quad (k<R)$$

$$P(y=y_k \mid X)=\frac{1}{1+\sum_{j}^{R-1}\exp(\sum_{i=1}^{n} w_{ji}X_i+w_{j0})} \quad (k=R)$$

与二分类的不同之处在于，系数矩阵 $\mathbf{w}$ 的维度由 $(1,n+1)$ 变为 $(R-1,n+1)$。

3.3 逻辑回归的训练过程

如果采用最大似然（MLE）方法估计参数 $\mathbf{w}$，则

$$\begin{aligned}\mathbf{w}_{MLE} &= \arg\max P(Y,X \mid \mathbf{w}) \\ &= \arg\max \prod_{l}^{N} P(Y^l,X^l \mid \mathbf{w})\end{aligned}$$

但是由于每个样本 Y^l,X^l 的联合分布难以估计，实际中并未采用 MLE 方法进行估计，反而将 X^l 后置，用最大条件似然（MCLE）估计参数 $\mathbf{w}$：

$$\begin{aligned}\mathbf{w}_{MCLE} &= \arg\max P(Y \mid X,\mathbf{w}) \\ &= \arg\max \prod_{l}^{N} P(Y^l \mid X^l,\mathbf{w}) \\ &= \arg\max \sum_{l}^{N} \ln P(Y^l \mid X^l,\mathbf{w})\end{aligned}$$

令

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

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

因此，损失函数设定为：

$$
\begin{aligned}
l(\mathbf{w}) &= -\sum_{l}^{N} \ln P(Y^l \mid X^l, \mathbf{w}) \\
&= -\sum_{l}^{N} (Y^l \ln P(Y^l = 1 \mid X^l, \mathbf{w}) + (1 - Y^l) \ln P(Y^l = 0 \mid X^l, \mathbf{w})) \\
&= -\sum_{l}^{N} (Y^l \ln \frac{P(Y^l = 1 \mid X^l, \mathbf{w})}{P(Y^l = 0 \mid X^l, \mathbf{w})} + \ln P(Y^l = 0 \mid X^l, \mathbf{w})) \\
&= -\sum_{l}^{N} (Y^l (\sum_{i=1}^{n} w_i X_i + w_0) - \ln(1 + \exp(\sum_{i=1}^{n} w_i X_i + w_0))
\end{aligned}
$$

对参数 $\mathbf{w}$ 求导得到，

$$
\frac{\partial l(\mathbf{w})}{\partial w_i} = \sum_{l} X_i^l (Y^l - \hat{P}(Y^l = 1 \mid X^l, W)
$$

若采用带有正则化项的损失函数，则

$$
l(\mathbf{w}) = -\sum_{l}^{N} (Y^l (\sum_{i=1}^{n} w_i X_i + w_0) - \ln(1 + \exp(\sum_{i=1}^{n} w_i X_i + w_0)) + \frac{\lambda}{2} \| \mathbf{w}^T \mathbf{w} \|
$$

$$
\frac{\partial l(\mathbf{w})}{\partial w_i} = \sum_{l} X_i^l (Y^l - \hat{P}(Y^l = 1 \mid X^l, W) + \lambda w_i
$$

使用梯度下降进行更新：

$$
\mathbf{w} = \mathbf{w} - \alpha \frac{\partial l(\mathbf{w})}{\partial \mathbf{w}}
$$

四、实验结果与分析

4.1 生成数据

4.1.1 满足朴素贝叶斯假设的数据

朴素贝叶斯假设对于一个样本 $X =< X_1, X_2, ..., X_n >$，其不同维度 $X_1, X_2, ..., X_n$ 之间互相独立。为了方便结果呈现，在实验中取维度 $n = 2$。

```python
def generate_independent_data(mu, sigma, num_sample):
    """
    Generate data of (X, y) pairs, in which every dimension of X satisfies an independent Gaussian Distribution,
    and y satisfies a Bernoulli Distribution with the number of classes at 2.
    :param mu: a matrix of (2, x_dim)
    :param sigma: a matrix of (2, x_dim)
    :param num_sample: the number of samples to generate
    :return:
    """
    assert len(mu) == len(sigma)
    assert len(mu[0]) == 2
    assert len(mu[1]) == 2
    x_dim = len(mu)

    num_class_1, num_class_2 = generate_bernoulli_data(num_sample)

    data = []
    for i in range(num_class_1):
        x = generate_gauss_data(mu[0], sigma[0], x_dim)
        data.append([x, [0]])

    for i in range(num_class_2):
        x = generate_gauss_data(mu[1], sigma[1], x_dim)
        data.append([x, [1]])
    return data
```

图 1 生成符合朴素贝叶斯假设数据的代码

在具体实现代码中，对于 X，基于高斯分布生成随机数据，其中 X_1, X_2 两个维度的数据是独立生成的，分别符合不同的高斯分布，因而可以保证它们是相互独立的。然后以随机的比例将这些数据分为两类，标记为 0 和 1。

4.1.2 不满足朴素贝叶斯假设的数据

不满足朴素贝叶斯假设意味着对于一个样本 $X =< X_1, X_2, ..., X_n >$，其不同维度 $X_1, X_2, ..., X_n$ 之间存在任意两个维度不独立。当维度 $n = 2$ 时，代表 X_1, X_2 不独立。

```python
def generate_correlated_data(mu, cov, num_sample):
    """
    Generate data of (X, y) pairs, in which X satisfies a multi-variate Gaussian Distribution,
    and y satisfies a Bernoulli Distribution with the number of classes at 2.
    :param mu: the mean of multi-variate Gaussian Distribution
    :param cov: the covariance matrix of multi-variate Gaussian Distribution
    :param num_sample: the number of samples to generate
    :return: data
    """
    assert len(mu) == 2
    assert len(cov) == 2

    num_class_1, num_class_2 = generate_bernoulli_data(num_sample)
    data = []
    for i in range(num_class_1):
        x = generate_multi_variate_gauss_data(mu[0], cov[0])
        data.append([x, [0]])

    for i in range(num_class_2):
        x = generate_multi_variate_gauss_data(mu[1], cov[1])
        data.append([x, [1]])

    return data
```

图 2 生成不符合朴素贝叶斯假设数据的代码

在具体代码实现中，对于 X，基于二维高斯分布生成随机数据。为了保证 X_1, X_2 不独立，需要保证协方差矩阵的反对角线上两个元素相等且不能为 0。

4.1.3 UCI 数据集的导入

实验中，选择了 2 个三分类数据集对具有多分类功能的逻辑回归分类器进行测试，这两个数据集分别是鸢尾花数据集（iris）和种子数据集（seed）。其中鸢尾花数据集每个样本具有 4 个特征，种子数据集每个样本具有 7 个特征。

首先读入数据并转换数据集格式。

```python
def load_iris():
    data = []
    with open('uci_data/iris.data') as f:
        for line in f:
            l = line.strip('\n').split(',')
            x = [float(scalar) for scalar in l[:-1]]
            y = [0, 0, 0]
            if l[-1] == 'Iris-setosa':
                y[0] = 1
            elif l[-1] == 'Iris-versicolor':
                y[1] = 1
            elif l[-1] == 'Iris-virginica':
                y[2] = 1
            data.append([x, y])
    f.close()
    return data
```

```python
def load_seeds():
    data = []
    with open('uci_data/seeds_dataset.txt') as f:
        for line in f:
            l = re.split(r"\t+", line.strip("\n"))
            x = [float(scalar) for scalar in l[:-1]]
            y = [0, 0, 0]
            if l[-1] == '1':
                y[0] = 1
            elif l[-1] == '2':
                y[1] = 1
            elif l[-1] == '3':
                y[2] = 1
            data.append([x, y])
    f.close()
    return data
```

图 3 加载数据集的代码

接下来按一定比例将数据集分为训练集与测试集（实验中设定的比例为 9：1），训练集用于模型调参，测试集用于测试结果。

```python
def split_data(data):
    total = len(data)
    train_data = []
    test_data = []
    group = total // 10
    for g in range(group):
        for i in range(9):
            if g * 10 + i < total:
                train_data.append(data[g * 10 + i])
            else:
                break
        if g * 10 + 9 < total:
            test_data.append(data[g * 10 + 9])
    return train_data, test_data
```

图 4 切分数据集的代码

最后对数据的不同维度分别进行 z-score 归一化。这是由于不同特征的量纲不同，可能对分析结果产生影响；同时，统一的量纲可以加速梯度下降的收敛，也可以减少数据溢出的错误。

```python
def normalize(data):
    mean = np.mean(data, axis=0)
    std = np.std(data, axis=0)
    data = (data - mean) / std
    return data
```

图 5 正则化的代码

4.2 结果分析

4.2.1 数据是否线性可分对结果的影响

当数据线性可分时，应满足的条件是，不同类别、相同特征符合的高斯分布均值可以不同，但方差必须相同，且取值适当，这时得到的决策面一定是线性的。

生成共 50 个样本点，正例样本，取 $E(X_1)=1, E(X_2)=1; D(X_1)=0.5, D(X_2)=0.3$；负例样本，取 $E(X_1)=3, E(X_2)=2; D(X_1)=0.5, D(X_2)=0.3$，梯度下降训练 100w 轮次，得到的结果如下所示：

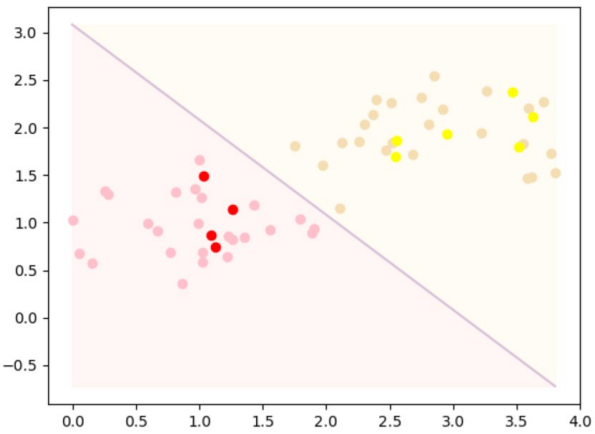

图 6 线性可分数据分类结果

最终的损失和梯度如下所示：

```
 96%|██████████| 959831/1000000 [03:48<00:09, 4311.53it/s]
MCLE Loss: 0.19591188734851644
gradient: [-0.00377811 -0.00387145  0.01158409]
 97%|██████████| 969791/1000000 [03:50<00:07, 4299.49it/s]
MCLE Loss: 0.19429160261041364
gradient: [-0.0037445  -0.00383982  0.01148416]
 98%|██████████| 979718/1000000 [03:53<00:04, 4502.28it/s]
MCLE Loss: 0.19269896816381277
gradient: [-0.00371149 -0.00380875  0.01138601]
 99%|██████████| 989673/1000000 [03:55<00:02, 4532.27it/s]
MCLE Loss: 0.19113326213421822
gradient: [-0.00367907 -0.00377821  0.01128959]
100%|██████████| 1000000/1000000 [03:57<00:00, 4206.52it/s]
```

图 7 线性可分数据 loss 和 gradient

可以看出，逻辑回归能够很好地对线性可分数据进行分类，最终梯度无限接近于 0，loss 也能够收敛到很小的值。

当数据线性不可分时，数据并不能被线性决策面完全分开，在二维空间内的表现是，不存在一条直线能够将两类数据完美分开，使得测试集（图中颜色较深的点测试集中的数据）准确率达到 100%。

生成共 50 个样本点，正例样本，取 $E(X_1)=1, E(X_2)=1; D(X_1)=3, D(X_2)=5$；负例样本取 $E(X_1)=3, E(X_2)=2; D(X_1)=3, D(X_2)=5$，梯度下降训练 5w 轮次，得到的结果如下所示：

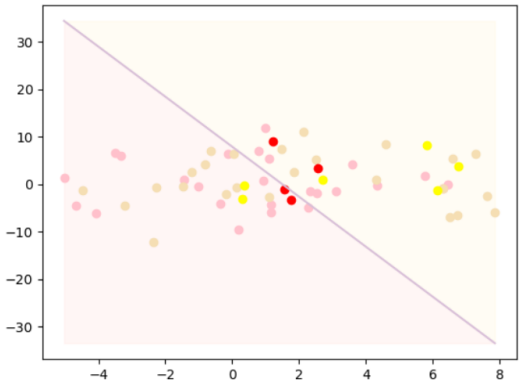

图 8 非线性可分数据分类结果

可以看出此时训练数据和测试数据都不能被得到的决策面很好地分开。损失、梯度和最终的参数如下所示：

```
 19%|█▉        | 9706/50000 [00:01<00:08, 5036.58it/s]
MCLE Loss: 33.808627482695655
gradient: [-2.95608023e-15  1.37929471e-15  1.34052591e-14]
 39%|███▉      | 19679/50000 [00:04<00:06, 4528.23it/s]
MCLE Loss: 33.808627482695655
gradient: [-2.95608023e-15  1.37929471e-15  1.34052591e-14]
 60%|█████▉    | 29826/50000 [00:06<00:04, 4795.61it/s]
MCLE Loss: 33.808627482695655
gradient: [-2.95608023e-15  1.37929471e-15  1.34052591e-14]
 80%|███████▉  | 40163/50000 [00:08<00:01, 4989.70it/s]
MCLE Loss: 33.808627482695655
gradient: [-2.95608023e-15  1.37929471e-15  1.34052591e-14]
100%|██████████| 50000/50000 [00:10<00:00, 4816.36it/s]
param: [ 0.10210773  0.01938071 -0.15343909]
```

图 9 非线性可分数据 loss 和 gradient(epoch=5w)

可以发现损失函数较大，并且从 1000 轮开始，损失函数发生的变化就微乎其微，但梯度已经十分接近于 0。增加训练次数到 10w 轮，结果如下所示：

```
 70%|███████   | 69867/100000 [00:15<00:07, 3921.21it/s]
MCLE Loss: 33.808627482695655
gradient: [-2.95608023e-15  1.37929471e-15  1.34052591e-14]
 80%|████████  | 80001/100000 [00:18<00:05, 3928.91it/s]
MCLE Loss: 33.808627482695655
gradient: [-2.95608023e-15  1.37929471e-15  1.34052591e-14]
 90%|█████████ | 89711/100000 [00:20<00:02, 4574.28it/s]
MCLE Loss: 33.808627482695655
gradient: [-2.95608023e-15  1.37929471e-15  1.34052591e-14]
100%|██████████| 100000/100000 [00:22<00:00, 4417.78it/s]
param: [ 0.10210773  0.01938071 -0.15343909]
```

图 10 非线性可分数据 loss 和 gradient(epoch=10w)

发现参数和损失都没有发生改变，因而可以得出结论，在数据线性不可分时，基于逻辑回归的分类器并不能做出很好的分类，只能找到一个线性决策面，使得两类数据尽可能被准确分类。但由于数据本身的性质使得其并不能被线性分开，结果并不理想。

4.2.2 数据是否符合朴素贝叶斯假设对结果的影响

生成 6 组符合朴素贝叶斯假设的数据，观察其结果如下：

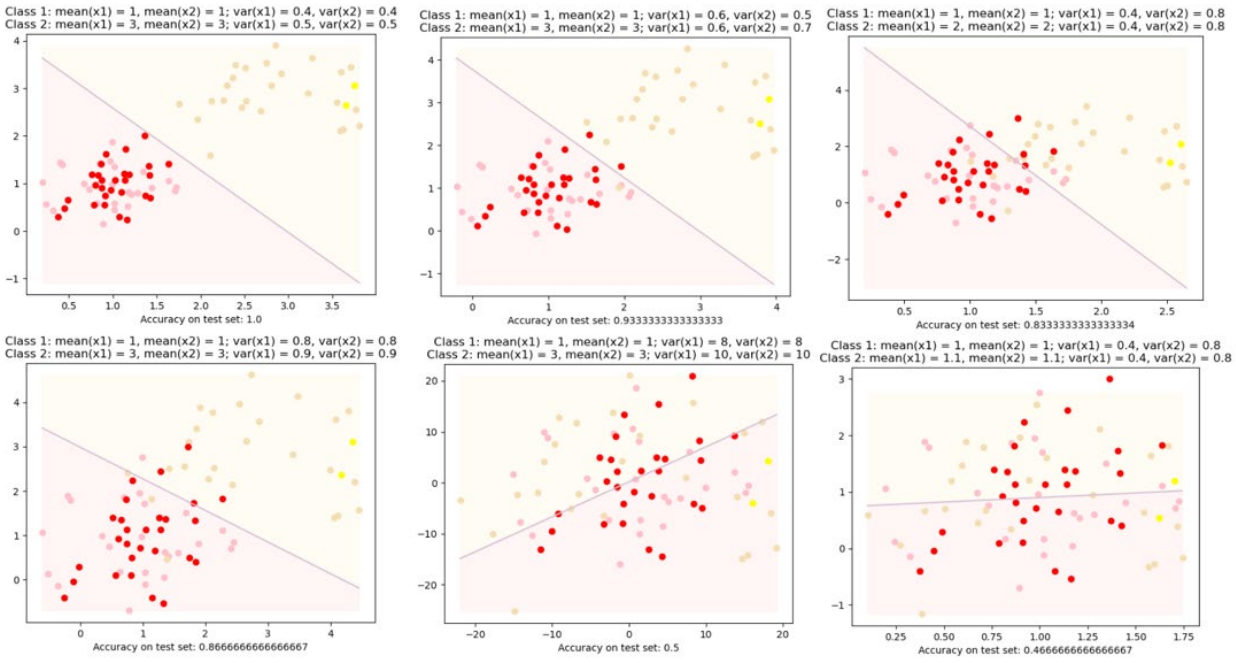

图 11 符合朴素贝叶斯假设数据的分类结果

在生成数据符合朴素贝叶斯假设的条件下，如果两类样本点分布的均值差异过小，方差过大，也会造成生成数据线性不可分，使得逻辑回归分类器准确下降。比较极端的情况下，当正、负例的样本均值为 1 和 3，方差分别 8 和 10 时，准确率仅为 0.5；当正、负例样本方差为 0.4、0.8，均值分别为 1，1.1 时，准确率仅为 0.47。只有当样本点分布均值差异以及方差均在合适的范围内时，才能保证数据是线性可分的，这时逻辑回归的准确率可以达到 1.0。

生成不符合朴素贝叶斯假设的数据，观察结果如下：

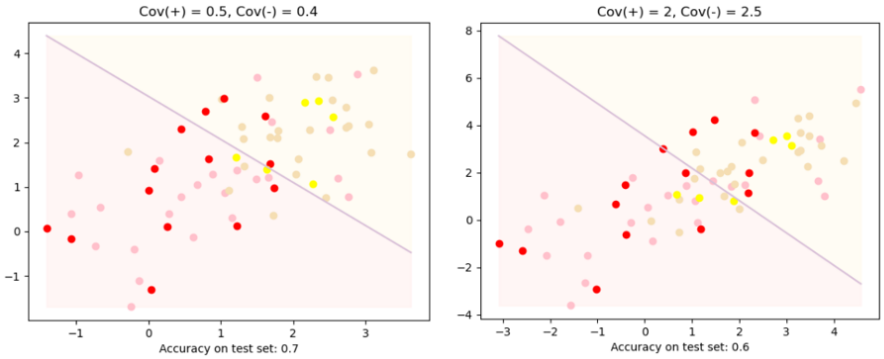

图 12 不符合朴素贝叶斯假设数据的分类结果

在生成数据不满足朴素贝叶斯假设的条件下，逻辑回归也只能找到使得损失最小的一个线性决策面，尽可能地将每个样本点分到正确的类别，但是即使在协方差取值并不大的情况下，分类结果也并不是很理想。

由此可见，无论是否符合朴素贝叶斯假设，对于线性可分的数据集，逻辑回归可以找到一个完美的决策面进行分类；对于非线性可分的数据集，逻辑回归只能找出一个使损失尽可能小的一个线性决策面，并不能很好地处理这种数据集。

4.2.3 正则化项的作用

生成数据集满足以下条件：对于正例 $E(X_1)=1, E(X_2)=1; D(X_1)=0.4, D(X_2)=0.4;$ 对于负例：$E(X_1)=2, E(X_2)=2; D(X_1)=0.4, D(X_2)=0.4$，训练数据仅有 5 个样本点，测试数据包含 100 个样本点。分别在有正则化项和无正则化项下进行实验，结果如下图所示。

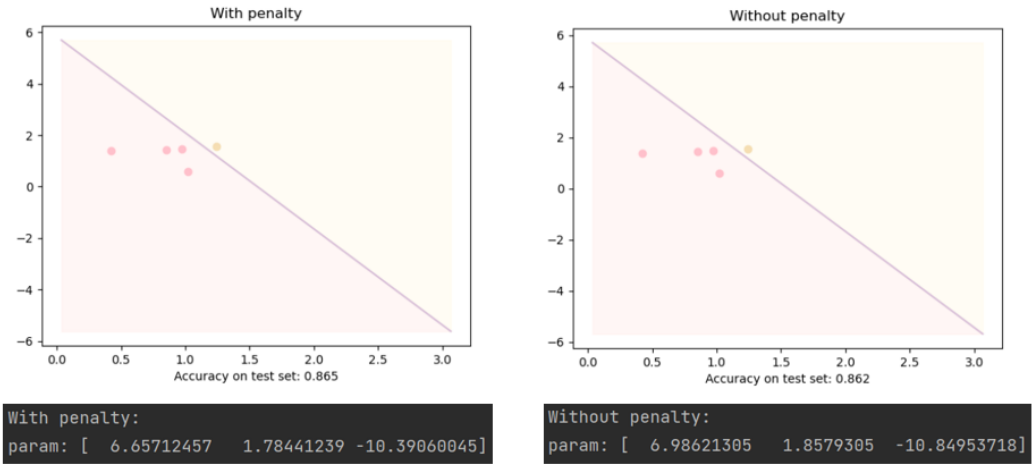

图 13 是否有正则化项对于分类结果的对比-1

可以看出，添加惩罚项使得准确率提高了 0.3%，并且观察参数可以发现，正则化项使参数值减少，模型复杂度降低，对于减缓模型的过拟合有一定的作用。

然而，经过多次实验发现，即使训练数据仅有 5 个样本，逻辑回归模型的过拟合问题也不严重，正则化项的优化作用微乎其微，同时由于参数原本量级就较小，正则化项对参数的影响也很小，多数情况下有无正则化项都不会影响决策面的选择和分类的准确率。随着训练数据的增加，更难产生过拟合现象，正则化项的影响更弱。

训练数据分别为 5 和 10 的实验结果如下图，在逻辑回归中当训练数据达到 10 就已经

很难发生过拟合现象了：

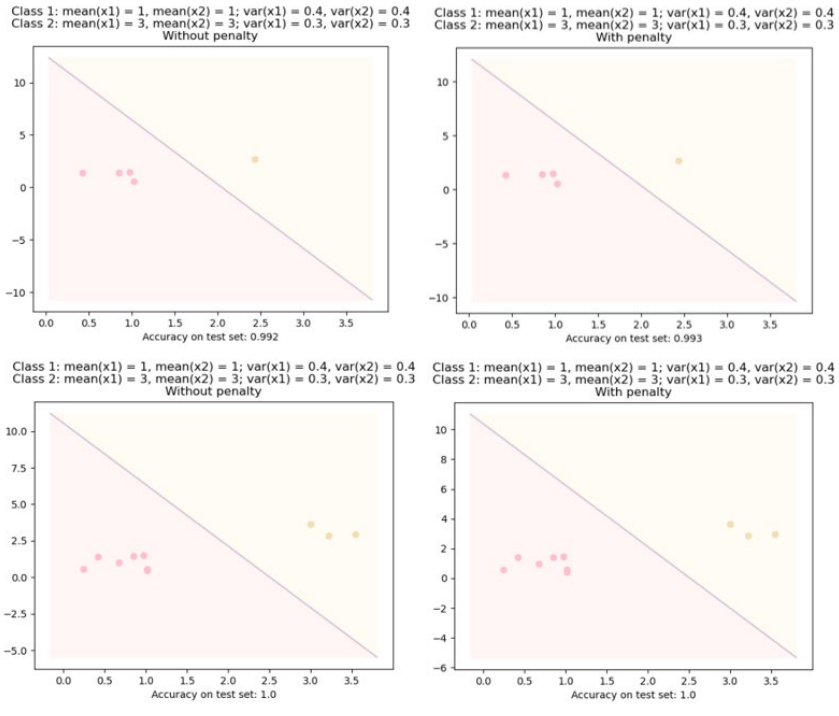

图 14 是否有正则化项对于分类结果的对比-2

4.2.4 数据溢出的解决方法

当数据量级过大，在 sigmoid 函数中取 e 的指数操作很容易导致数据溢出。经查阅相关资料和进行数学推导，发现可以通过改变 sigmoid 函数的计算方式来防止数据溢出。

$$sigmoid(x)=\begin{cases}\dfrac{1}{1+e^{-x}}, x\geq 0\\ \dfrac{e^{x}}{1+e^{x}}, x<0\end{cases}$$

这样无论数据的绝对值量级多大，都可以避免可能导致溢出的指数运算。以计算梯度为例，代码实现如下：

```
def gradient(self, X, Y):
    linear = self.b + np.dot(X, self.W.T)
    sigmoid_linear = self.sigmoid(linear)
    w_grad = - (np.sum((Y - sigmoid_linear) * X.T, axis=1))
    b_grad = - (np.sum(Y - sigmoid_linear))
    grad = np.concatenate((w_grad, [b_grad]))
    # print(grad)
    return grad
```

图 15 避免数据溢出的方法

4.2.5 UCI 数据集上的测试结果

实验中使用了鸢尾花数据集和种子数据集。

鸢尾花数据集中，数据特征为4个维度，数据类别有3个，分别是Iris-setosa，Iris-versicolor和 Iris-virginica。种子数据集中，数据特征为 7 个维度，数据类别有 3 个。为了与三分类对应，编写了可以实现多分类功能的逻辑回归分类器：

初始化参数:

```
class multi_logistic_regression_classifier:
    def __init__(self, x_dim, n_class):
        self.W = np.random.randn(n_class - 1, x_dim)
        self.b = np.random.randn(n_class - 1)
```

计算损失：

```
def mcle_loss(self, X, Y):
    tmp = np.c_[np.exp(np.dot(X, self.W.T) + self.b), np.ones(X.shape[0])]
    sum_exp = np.sum(tmp, axis=1)
    sum_exp = np.reshape(sum_exp, (sum_exp.shape[0], 1))
    loss = - np.sum(Y * np.log(tmp / sum_exp)) / X.shape[0]
    return loss
```

计算梯度：

```
def gradient(self, X, Y):
    exp = np.exp(np.dot(X, self.W.T) + self.b)
    sum_exp = np.sum(exp, axis=1) + 1
    sum_exp = np.reshape(sum_exp, (sum_exp.shape[0], 1))
    X = np.expand_dims(np.c_[X, np.ones(X.shape[0])], axis=1)
    X = np.repeat(X, Y.shape[1] - 1, axis=1)
    tmp = np.expand_dims(Y.T[:-1].T, axis=-1) * X
    ratio = np.expand_dims(exp / sum_exp, axis=-1)
    grad = - np.sum(tmp - ratio * X, axis=0)
    return grad
```

更新参数：

```
def train_gradient_descent(self, X, Y, lr=0.001, epoch=1000, log=True):
    for e in trange(epoch):
        param = self.param
        grad = self.gradient(X, Y)
        param = optimizer.gradient_descent(param, grad, lr)
        self.W = param.T[:-1].T
        self.b = param.T[-1]

        if log and e % 100 == 0:
            print(self.mcle_loss(X, Y))
```

预测结果：

```
def predict(self, X):
    logits = np.exp(np.dot(X, self.W.T) + self.b)
    logits = np.c_[logits, np.ones(logits.shape[0])]
    label = np.argmax(logits, axis=1)
    return label
```

实现多分类器后，就可以在数据集上进行测试。其中对数据的预处理也十分关键，需要在使用数据前将其随机打乱并统一量纲。最终在鸢尾花数据集和种子数据集上的结果如下所示：

鸢尾花数据集：

```
Accuracy on train set:
0.9777777777777777
Accuracy on test set:
1.0
```

图 16 鸢尾花数据集分类结果

种子数据集：

```
Accuracy on train set:
0.9894179894179894
Accuracy on test set:
0.8095238095238095
```

图 17 种子数据集分类结果

在鸢尾花数据集上的结果明显优于种子数据集。查阅资料得知，鸢尾花数据集线性可分，但种子数据集非线性可分，该分类结果的优劣情况也与之前对数据集是否线性可分的分析相契合。同时，分类结果也与特征的选取有关，通过对实际情况的进一步调研，得到与分类结果更加相关的数据特征，也可以使分类效果得到提升。

五、结论

对于线性可分的数据集，逻辑回归可以找到一个完美的决策面进行分类；对于非线性可分的数据集，逻辑回归只能找出一个使损失尽可能小的一个线性决策面，并不能很好地处理这种数据集。事实上，是否符合朴素贝叶斯假设对分类结果的影响也是基于其是否线性可分而言的。如果数据并非线性可分，即使符合朴素贝叶斯假设，也不能被逻辑回归完美分类。但逻辑回归总会尽可能找到与训练数据更接近的决策面，使更多的点被正确分类。

在数据量很小时，逻辑回归也会遭受轻微的过拟合现象，可以通过添加正则化项加以缓解。但是逻辑回归的过拟合现象并不严重，只要数据集稍微增加，就基本不存在过拟合现象，正则化项的影响也十分微小。

六、参考文献

[1] Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg.

[2] 周志华. 2016. 机器学习.

[3] Strang, G. 2006. Linear algebra and its applications. Belmont, CA: Thomson, Brooks/Cole.

[4] https://timvieira.github.io/blog/post/2014/02/11/exp-normalize-trick/

七、附录：源代码（带注释）

Utils.py:

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

random.seed(1219)
np.random.seed(1219)


def generate_bernoulli_data(num_sample):
    """
    generate data that satisfy bernoulli distribution
    :param num_sample: the total number of samples to generate
```

```
    :return: the number of samples in two classes
    """
    num_class_1 = random.randint(1, num_sample - 1)
    num_class_2 = num_sample - num_class_1
    return num_class_1, num_class_2


def generate_gauss_data(mu, sigma, dim):
    """
    generate data that satisfy gauss distribution
    :param mu: mean
    :param sigma: variance
    :param dim: dimension of parameter x
    :return: generated data x
    """
    x = []
    for j in range(dim):
        x.append(random.gauss(mu[j], sigma[j]))
    return x


def generate_independent_data(mu, sigma, num_sample):
    """
    Generate data of (X, y) pairs, in which every dimension of X satisfies an independent
    and y satisfies a Bernoulli Distribution with the number of classes at 2.
    :param mu: a matrix of (2, x_dim)
    :param sigma: a matrix of (2, x_dim)
    :param num_sample: the number of samples to generate
    :return: data
    """
    assert len(mu) == len(sigma)
    assert len(mu[0]) == 2
    assert len(mu[1]) == 2
    x_dim = len(mu)

    num_class_1, num_class_2 = generate_bernoulli_data(num_sample)
    # num_class_1, num_class_2 = int(0.6 * num_sample), int(0.4 * num_sample)

    data = []
    for i in range(num_class_1):
        x = generate_gauss_data(mu[0], sigma[0], x_dim)
        data.append([x, [0]])

    for i in range(num_class_2):
```

```
57          x = generate_gauss_data(mu[1], sigma[1], x_dim)
58          data.append([x, [1]])
59      return data
60
61
62  def generate_multi_variate_gauss_data(mu, cov):
63      """
64      generate data that satisfy multi-variate gauss
65      :param mu: mean
66      :param cov: variance
67      :return: generated data
68      """
69      assert len(mu) == len(cov)
70      assert len(cov) == len(cov[0])
71      x_dim = len(mu)
72
73      x = np.random.multivariate_normal(mu, cov).tolist()
74      return x
75
76
77  def generate_correlated_data(mu, cov, num_sample):
78      """
79      Generate data of (X, y) pairs, in which X satisfies a multi-variate Gaussian Distribu
80      and y satisfies a Bernoulli Distribution with the number of classes at 2.
81      :param mu: the mean of multi-variate Gaussian Distribution
82      :param cov: the covariance matrix of multi-variate Gaussian Distribution
83      :param num_sample: the number of samples to generate
84      :return: data
85      """
86      assert len(mu) == 2
87      assert len(cov) == 2
88
89      num_class_1, num_class_2 = generate_bernoulli_data(num_sample)
90      data = []
91      for i in range(num_class_1):
92          x = generate_multi_variate_gauss_data(mu[0], cov[0])
93          data.append([x, [0]])
94
95      for i in range(num_class_2):
96          x = generate_multi_variate_gauss_data(mu[1], cov[1])
97          data.append([x, [1]])
98
99      return data
100
```

```python

def shuffle_data(data, shuffle=True):
    """
    shuffle data randomly
    :param data: input data
    :param shuffle: conduct shuffle operation or not
    :return: processed data
    """
    if shuffle:
        random.shuffle(data)
    X = []
    Y = []
    for d in data:
        X.append(d[0])
        Y.append(d[1][0])
    return np.array(X), np.array(Y)


# def shuffle_data_multi(data):
#     random.shuffle(data)
#     X = []
#     Y = []
#     for d in data:
#         X.append(d[0])
#         if d[1][0] == 0:
#             Y.append([1, 0])
#         else:
#             Y.append([0, 1])
#     return np.array(X), np.array(Y)


def visualize_data(data, mode='train'):
    x_class_1, y_class_1, x_class_2, y_class_2 = [], [], [], []
    for d in data:
        if d[1][0] == 0:
            x_class_1.append(d[0][0])
            y_class_1.append(d[0][1])
        if d[1][0] == 1:
            x_class_2.append(d[0][0])
            y_class_2.append(d[0][1])
    if mode == 'train':
        plt.plot(x_class_1, y_class_1, 'ro', color='pink')
        plt.plot(x_class_2, y_class_2, 'ro', color='wheat')
    if mode == 'test':
```

```
        plt.plot(x_class_1, y_class_1, 'ro', color='red')
        plt.plot(x_class_2, y_class_2, 'ro', color='yellow')


def visualize_result_2d(param, x_min, x_max, y_min, y_max):
    w = param[:-1]
    b = param[-1]
    x = np.arange(x_min, x_max, 0.01)
    y = -(b + w[0] * x) / w[1]
    y_min = min(y_min, y[0], y[-1])
    y_max = max(y_max, y[0], y[-1])
    plt.plot(x, y, color='thistle')
    plt.fill_between(x, y, y_max, color='cornsilk', alpha=0.3)
    plt.fill_between(x, y_min, y, color='mistyrose', alpha=0.3)


# def visualize_result_2d_multi(param, x_min, x_max, y_min, y_max):
#     w = param.T[:-1].T
#     b = param.T[-1]
#     x = np.arange(x_min, x_max, 0.01)
#     y = -(b + w[0][0] * x) / w[0][1]
#     y_min = min(y_min, y[0], y[-1])
#     y_max = max(y_max, y[0], y[-1])
#     plt.plot(x, y, color='thistle')
#     plt.fill_between(x, y, y_max, color='mistyrose', alpha=0.3)
#     plt.fill_between(x, y_min, y, color='cornsilk', alpha=0.3)


def evaluate(labels, logits):
    """
    evaluate the accuracy
    :param labels: true
    :param logits: prediction
    :return: accuracy
    """
    total = len(labels)
    correct = 0
    for i in range(total):
        correct += (labels[i] == logits[i])
    acc = correct / total
    return acc


if __name__ == '__main__':
```

```
    # ----------------------- test -----------------------
    mu = [[0, 1], [0.3, 0.1]]
    sigma = [[0.1, 0.2], [0.1, 0.2]]
    # cov = [[[1, 1], [1, 2]], [[1, 1], [1, 2]]]
    data_ind = generate_independent_data(mu, sigma, 100)
    # data_cov = generate_correlated_data(mu, cov, 100)
    # visualize_data(data_ind)
    # plt.show()
    X = [1, 2, 3]
    Y = [1, 3, 3]
    print(len([x for x in X if x in Y]))
```

Load.py:

```
import numpy as np
import random
import re

random.seed(0)
np.random.seed(0)


def load_iris():
    data = []
    with open('uci_data/iris.data') as f:
        for line in f:
            l = line.strip('\n').split(',')
            x = [float(scalar) for scalar in l[:-1]]
            y = [0, 0, 0]
            if l[-1] == 'Iris-setosa':
                y[0] = 1
            elif l[-1] == 'Iris-versicolor':
                y[1] = 1
            elif l[-1] == 'Iris-virginica':
                y[2] = 1
            data.append([x, y])
    f.close()
    return data


def load_seeds():
    data = []
    with open('uci_data/seeds_dataset.txt') as f:
        for line in f:
            l = re.split(r"\t+", line.strip("\n"))
```

```
            x = [float(scalar) for scalar in l[:-1]]
            y = [0, 0, 0]
            if l[-1] == '1':
                y[0] = 1
            elif l[-1] == '2':
                y[1] = 1
            elif l[-1] == '3':
                y[2] = 1
            data.append([x, y])
    f.close()
    return data


def split_data(data):
    total = len(data)
    train_data = []
    test_data = []
    group = total // 10
    for g in range(group):
        for i in range(9):
            if g * 10 + i < total:
                train_data.append(data[g * 10 + i])
            else:
                break
        if g * 10 + 9 < total:
            test_data.append(data[g * 10 + 9])
    return train_data, test_data


def process_train(train):
    random.shuffle(train)
    X = []
    Y = []
    for d in train:
        X.append(d[0])
        Y.append(d[1])
    X = normalize(np.array(X))
    return X, np.array(Y)


def process_test(test):
    random.shuffle(test)
    X = []
    target = []
```

```
    for d in test:
        X.append(d[0])
        target.append(np.argmax(d[1]))
    X = normalize(np.array(X))
    return X, np.array(target)


def normalize(data):
    mean = np.mean(data, axis=0)
    std = np.std(data, axis=0)
    data = (data - mean) / std
    return data


if __name__ == '__main__':
    data = load_iris()
    train, test = split_data(data)
    print(train)
```

Logistic_regression.py:

```
import numpy as np
from tqdm import trange
import optimizer


class logistic_regression_classifier:
    def __init__(self, x_dim):
        self.W = np.random.randn(x_dim)
        self.b = np.random.randn(1)

    @property
    def param(self):
        param = np.concatenate((self.W, self.b))
        return param

    def mcle_loss(self, X, Y):
        linear = self.b + np.dot(X, self.W.T)
        loss = - np.sum(Y * linear - np.log(1 + np.exp(linear)))
        return loss

    def mcle_loss_with_penalty(self, X, Y, lambd):
        linear = self.b + np.dot(X, self.W.T)
        loss = - np.sum(Y * linear - np.log(1 + np.exp(linear))) + lambd / 2 * np.dot(sel
        return loss
```

```
25
26      def gradient(self, X, Y):
27          linear = self.b + np.dot(X, self.W.T)
28          sigmoid_linear = self.sigmoid(linear)
29          w_grad = - (np.sum((Y - sigmoid_linear) * X.T, axis=1))
30          b_grad = - (np.sum(Y - sigmoid_linear))
31          grad = np.concatenate((w_grad, [b_grad]))
32          # print(grad)
33          return grad
34
35      def train_gradient_descent(self, X, Y, lr=0.001, epoch=1000, log=False, penalty=False
36          loss = 0
37          for e in trange(epoch):
38              param = self.param
39              if penalty and lambd is not None:
40                  grad = self.gradient_with_penalty(X, Y, lambd)
41              else:
42                  grad = self.gradient(X, Y)
43              param = optimizer.gradient_descent(param, grad, lr)
44              self.W, self.b = param[:-1], [param[-1]]
45
46              if penalty and lambd is not None:
47                  loss_new = self.mcle_loss_with_penalty(X, Y, lambd)
48              else:
49                  loss_new = self.mcle_loss(X, Y)
50
51              # if np.abs(loss_new - loss) < 1e-5:
52              #     lr /= 2
53              # loss = loss_new
54
55              if log and e % 10000 == 0:
56                  print('\nMCLE Loss: ' + str(loss_new))
57                  print('gradient: ' + str(grad))
58
59      @staticmethod
60      def sigmoid(X):
61          res = np.zeros(len(X))
62          for i in range(len(X)):
63              if X[i] >= 0:
64                  res[i] = 1 / (1 + np.exp(-X[i]))
65              else:
66                  res[i] = np.exp(X[i]) / (1 + np.exp(X[i]))
67          return res
68
```

```
    # with penalty
    def gradient_with_penalty(self, X, Y, lambd):
        linear = self.b + np.dot(X, self.W.T)
        sigmoid_linear = self.sigmoid(linear)
        w_grad = - (np.sum((Y - sigmoid_linear) * X.T, axis=1)) + lambd * self.W
        b_grad = - (np.sum(Y - sigmoid_linear)) + lambd * self.b[0]
        grad = np.concatenate((w_grad, [b_grad]))
        # print(grad)
        return grad

    def predict(self, X):
        logits = np.exp(np.dot(X, self.W.T) + self.b)
        logits = np.c_[logits, np.ones(logits.shape[0])]
        label = np.argmin(logits, axis=1)
        return label


class multi_logistic_regression_classifier:
    def __init__(self, x_dim, n_class):
        self.W = np.random.randn(n_class - 1, x_dim)
        self.b = np.random.randn(n_class - 1)

    def show_w(self):
        print(self.W)

    def show_b(self):
        print(self.b)

    @property
    def param(self):
        param = np.c_[self.W, self.b]
        return param

    def mcle_loss(self, X, Y):
        tmp = np.c_[np.exp(np.dot(X, self.W.T) + self.b), np.ones(X.shape[0])]
        sum_exp = np.sum(tmp, axis=1)
        sum_exp = np.reshape(sum_exp, (sum_exp.shape[0], 1))
        loss = - np.sum(Y * np.log(tmp / sum_exp)) / X.shape[0]
        return loss

    def gradient(self, X, Y):
        exp = np.exp(np.dot(X, self.W.T) + self.b)
        sum_exp = np.sum(exp, axis=1) + 1
        sum_exp = np.reshape(sum_exp, (sum_exp.shape[0], 1))
```

```
        X = np.expand_dims(np.c_[X, np.ones(X.shape[0])], axis=1)
        X = np.repeat(X, Y.shape[1] - 1, axis=1)
        tmp = np.expand_dims(Y.T[:-1].T, axis=-1) * X
        ratio = np.expand_dims(exp / sum_exp, axis=-1)
        grad = - np.sum(tmp - ratio * X, axis=0)
        return grad

    def train_gradient_descent(self, X, Y, lr=0.001, epoch=1000, log=True):
        for e in trange(epoch):
            param = self.param
            grad = self.gradient(X, Y)
            param = optimizer.gradient_descent(param, grad, lr)
            self.W = param.T[:-1].T
            self.b = param.T[-1]

            if log and e % 100 == 0:
                print(self.mcle_loss(X, Y))

    def predict(self, X):
        logits = np.exp(np.dot(X, self.W.T) + self.b)
        logits = np.c_[logits, np.ones(logits.shape[0])]
        label = np.argmax(logits, axis=1)
        return label


if __name__ == '__main__':
    x = np.array([[1, 2], [1, 2]])
    y = np.array([1, 0])
    print(x * y)
```

Optimizer.py:

```
def gradient_descent(param, grad, lr=0.000001):
    param -= lr * grad
    return param
```

Main.py:

```
import matplotlib.pyplot as plt

from logistic_regression import logistic_regression_classifier
from logistic_regression import multi_logistic_regression_classifier
import utils
import numpy as np

import load
```

```python


def test_independent_data(mu, sigma, x_dim):
    data_ind = utils.generate_independent_data(mu, sigma, 10)
    data_test = utils.generate_independent_data(mu, sigma, 1000)
    plt.title('Class 1: mean(x1) = ' + str(mu[0][0]) + ', mean(x2) = ' + str(mu[0][1]) + '
              + ', var(x2) = ' + str(sigma[0][1]) + '\nClass 2: mean(x1) = ' + str(mu[1][0
              + str(sigma[1][0]) + ', var(x2) = ' + str(sigma[1][1]) +
              '\nWith penalty')

    utils.visualize_data(data_ind)
    # utils.visualize_data(data_test, mode='test')
    # print(data_ind)
    X, Y = utils.shuffle_data(data_ind)
    X_test, Y_test = utils.shuffle_data(data_test, shuffle=False)
    x_min, x_max, y_min, y_max = min(X_test.T[0]), max(X_test.T[0]), min(X_test.T[1]), max
    classifier = logistic_regression_classifier(x_dim)
    # classifier.train_gradient_descent(X, Y, epoch=100000)
    classifier.train_gradient_descent(X, Y, epoch=100000, penalty=True, lambd=1e-3)

    print('Without penalty:')
    print('param: ' + str(classifier.param))

    predict = classifier.predict(X_test)
    acc = utils.evaluate(Y_test, predict)
    plt.xlabel('Accuracy on test set: ' + str(acc))
    # print("Accuracy on test set: ")
    # print(utils.evaluate(Y_test, predict))

    utils.visualize_result_2d(classifier.param, x_min, x_max, y_min, y_max)

    plt.show()


def test_correlated_data(mu, cov, x_dim):
    plt.title('Cov(+) = ' + str(cov[0][0][1]) + ', Cov(-) = ' + str(cov[1][0][1]))
    data_cov = utils.generate_correlated_data(mu, cov, 5)
    utils.visualize_data(data_cov)
    data_test = utils.generate_correlated_data(mu, cov, 50)
    utils.visualize_data(data_test, mode='test')
    # print(data_ind)
    X, Y = utils.shuffle_data(data_cov)
    X_test, Y_test = utils.shuffle_data(data_test, shuffle=False)
    x_min, x_max, y_min, y_max = min(X_test.T[0]), max(X.T[0]), min(X.T[1]), max(X.T[1])
```

```
    classifier = logistic_regression_classifier(x_dim)

    classifier.train_gradient_descent(X, Y, epoch=100000)
    # classifier.train_gradient_descent(X, Y, epoch=100000, penalty=True, lambd=np.exp(-18
    utils.visualize_result_2d(classifier.param, x_min, x_max, y_min, y_max)

    predict = classifier.predict(X_test)
    acc = utils.evaluate(Y_test, predict)
    plt.xlabel('Accuracy on test set: ' + str(acc))

    plt.show()


def test_uci(dataset='iris'):
    if dataset == 'iris':
        data = load.load_iris()
    elif dataset == 'seeds':
        data = load.load_seeds()
    train, test = load.split_data(data)
    X_train, Y = load.process_train(train)
    X_test, target = load.process_test(test)
    _, train_target = load.process_test(train)
    classifier = multi_logistic_regression_classifier(X_train.shape[1], Y.shape[1])
    classifier.train_gradient_descent(X_train, Y, epoch=10000, log=True)

    X_valid, target_train = load.process_test(train)
    predict_train = classifier.predict(X_valid)
    print("Accuracy on train set: ")
    print(utils.evaluate(target_train, predict_train))
    predict = classifier.predict(X_test)
    print("Accuracy on test set: ")
    print(utils.evaluate(target, predict))


if __name__ == '__main__':
    mu = [[1, 1], [3, 3]]
    # sigma = [[0.8, 0.8], [0.9, 0.9]]
    sigma = [[0.4, 0.4], [0.3, 0.3]]
    # test_independent_data(mu, sigma, 2)
    # cov = [[[3, 2], [2, 3]], [[3, 2.5], [2.5, 3]]]
    # test_correlated_data(mu, cov, 2)

    test_uci('seeds')
```