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：1190201421

姓名：张瑞

**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。

1. **实验要求及实验环境**

**（一）实验要求**

1. 实现两种损失函数的参数估计（无惩罚项和带惩罚项）。

2. 采用梯度下降或者牛顿法求解。

3. 手工生成两个分别类别数据（可以用高斯分布），验证算法。

4. 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。

5. 逻辑回归有广泛的用处，例如广告预测。可以到UCI网站上，找实际数据加以测试。

**（二）实验环境**

Windows 10; PyCharm Community Edition 2021.2;Python 3.6

**三、设计思想**

给定训练集，其中。逻辑回归模型能从该训练集中学习得到一个分类器，以便对新的数据进行分类。

假设中每一维都是独立同分布的，且所有的为高斯分布，为伯努利分布，则对于二分类问题，根据贝叶斯公式有：

其中经计算可以表示为，则等式可进一步转换为：

进而得到：

若将视为样本作为第类的可能性，则是其为第类的可能性，两者的比值称为“几率”，反映了作为第类的相对可能性，对几率取对数则得到“对数几率”，记为。

将上述推导所得代入可得，若，则将分到第类，若，则将分到第类。此时，得到决策边界为。

当然，为了简化运算，在对样本分类的时候，也可以直接计算函数值，若的值大于等于，则将其分到第类，否则为第类。

若要求解参数，用最大似然估计（MLE）得到式子如下：

采用最大条件似然估计（MCLE）则能将式子写为：

为了防止较小数值连乘带来的下溢问题，可将上式取对数后再求最大值，即：

再将上式添加一个负号，即可定义代价函数：

则当有最小值时，对应的即为我们所求参数，为了求解该最小值，我们可以采用梯度下降的方法。首先对求导：

然后更新值（其中为步长，下同）：

如果考虑在代价函数中加入惩罚项，则有（其中为惩罚项比重，下同）：

求导得：

更新值：

1. **实验结果与分析**

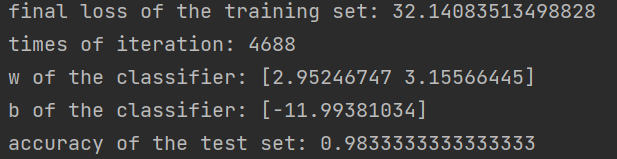
**（一）手动生成数据**

为了方便画图展示，将手动生成的数据维度定为2，这样就可以在二维平面直角坐标系下进行展示。

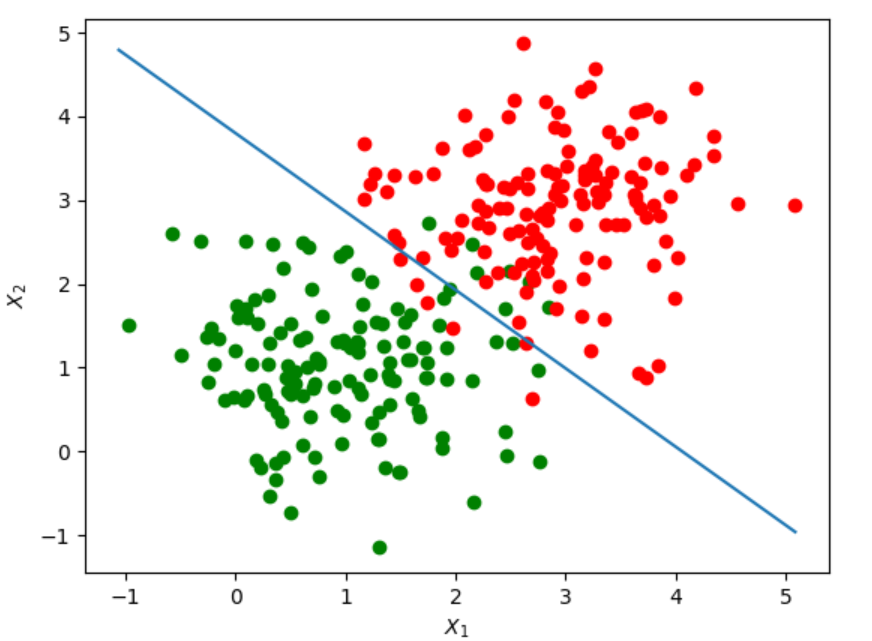
对于满足朴素贝叶斯假设的数据，其两个属性之间没有相关关系，协方差为0；对于不满足朴素贝叶斯假设的数据，其两个属性之间没有相关关系，协方差应该满足。

对于下列手动生成的数据，类0中的数据均值为，类1中的数据均值为。各类数据中各维的方差都为0.6，不满足朴素贝叶斯假设时，协方差满足。两个类的训练集大小均为140，测试集大小均为60，梯度下降求解时的步长均为0.01，惩罚项比重为1e-8，终止迭代的精度要求为为1e-5。

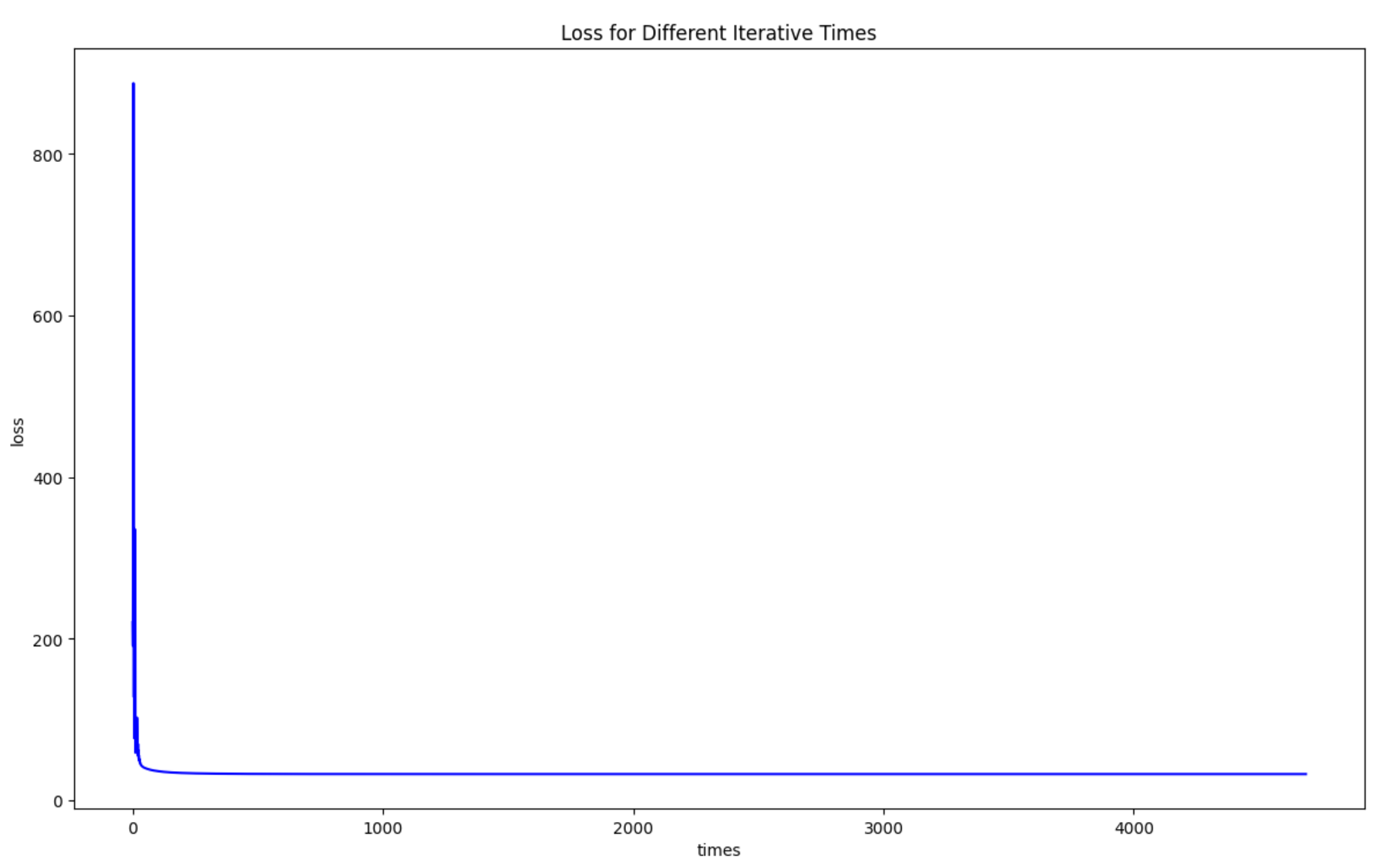
**1.满足朴素贝叶斯假设，无惩罚项**



可见，迭代次数4688次，最终的代价函数值为32.14，在测试集上的准确率为98.3%。决策边界及训练集如下：

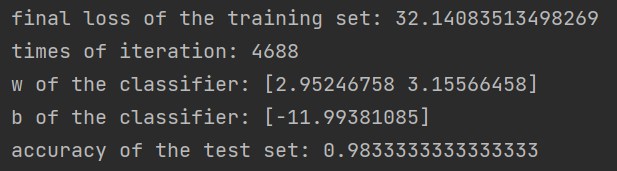


代价函数值随迭代次数变化的情况如下：

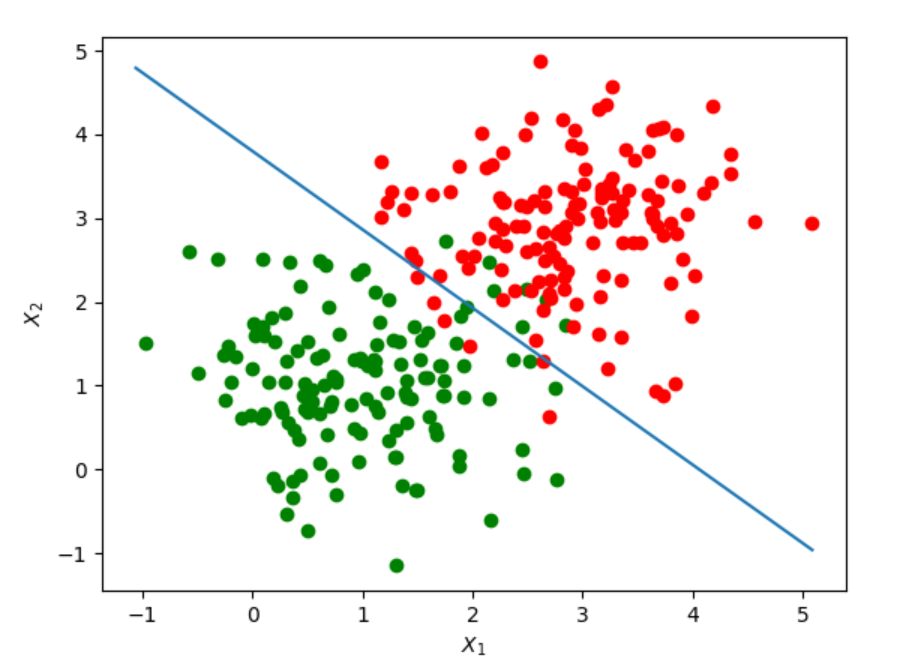


可以看到，由于两类数据分散得较为明显，所以逻辑回归模型得到的分类结果是很理想的，在测试集上的正确率很高。代价函数也能在迭代过程当中迅速下降到一个较小值，虽然后期收敛到的值依然较大，但这是由于两类数据的方差较大导致的，若将方差值减小，则会得到更小的代价函数值。

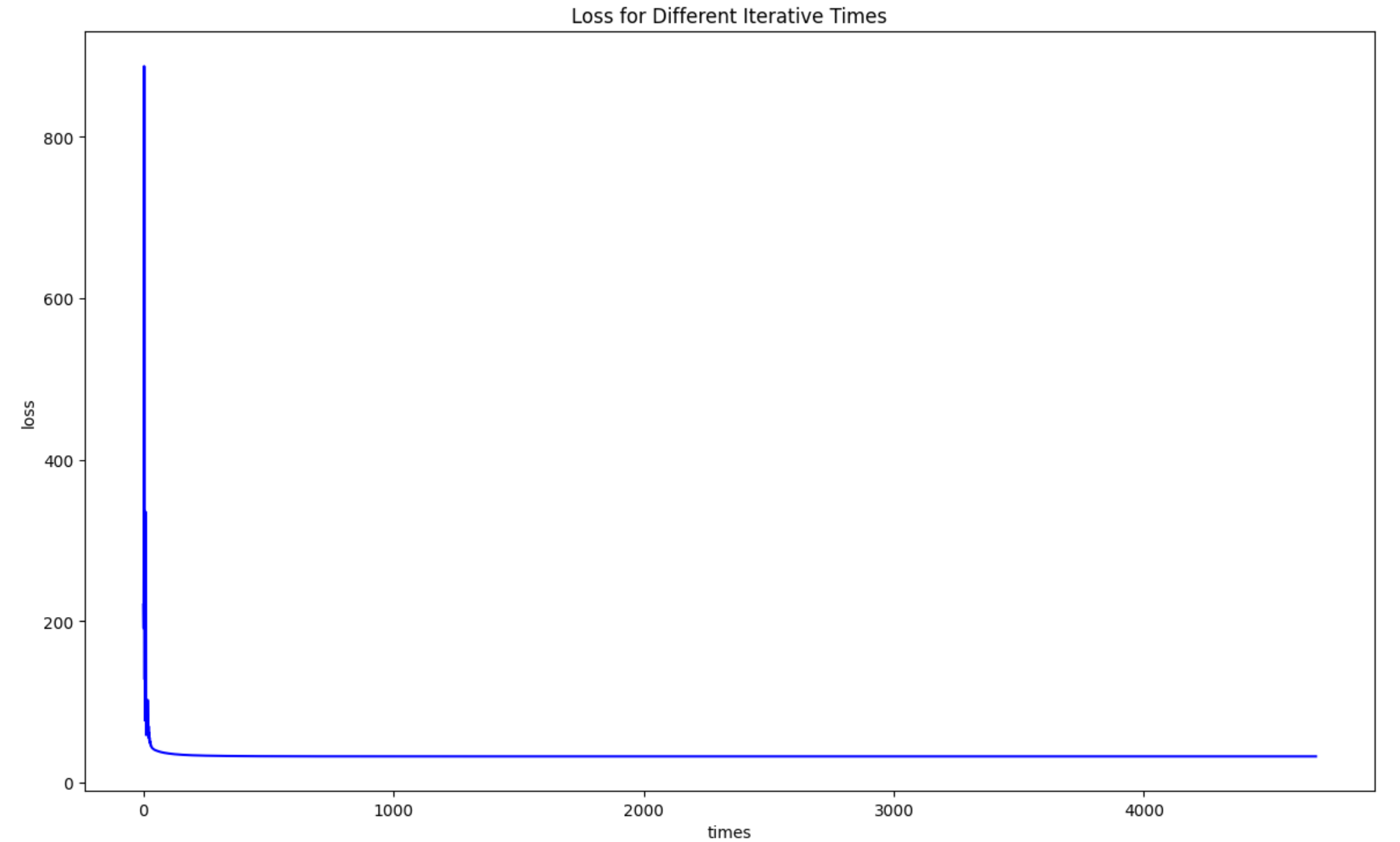
**2.满足朴素贝叶斯假设，带惩罚项**



可见，与无惩罚项极其相似，迭代次数4688次，最终的代价函数值为32.14，在测试集上的准确率为98.3%。决策边界及训练集如下：

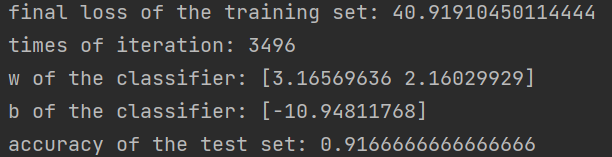


代价函数值随迭代次数变化的情况如下：

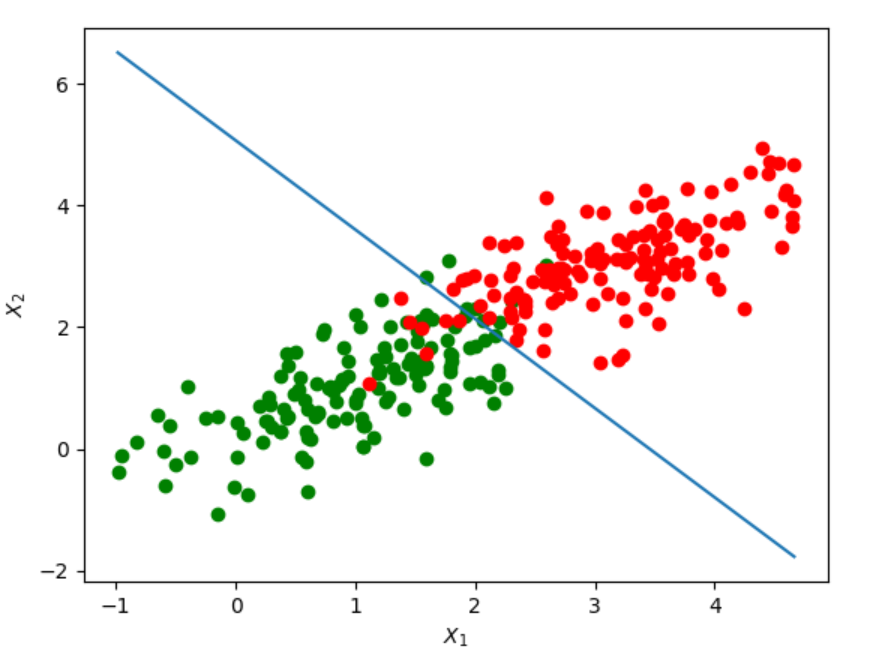


可以发现，加上惩罚项之后并无太大变化，这应该是数据集较大引起的，大量的数据和加入惩罚项能起到相同的作用。

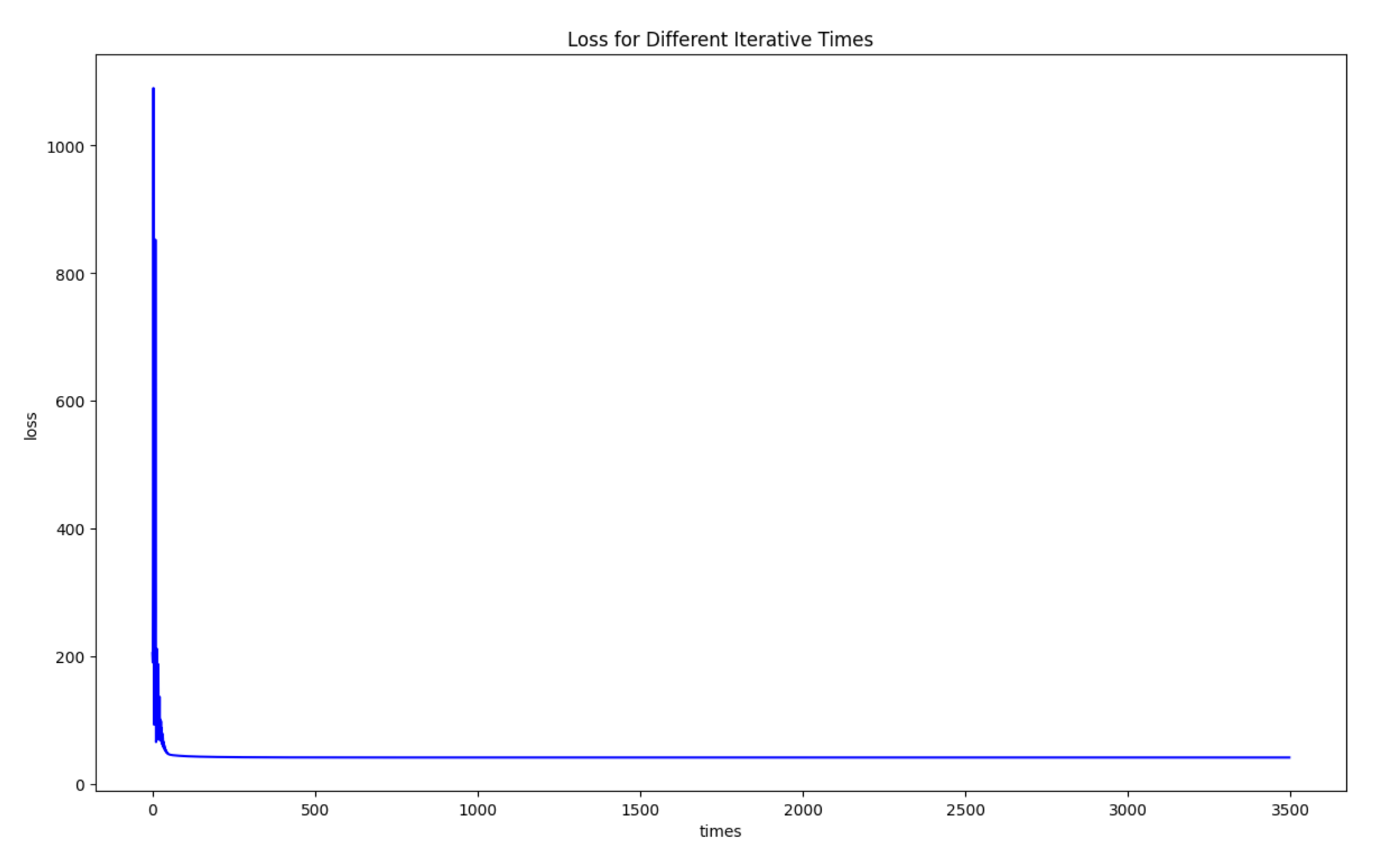
**3.不满足朴素贝叶斯假设，无惩罚项**



可见，迭代次数3496次，最终的代价函数值为40.92，在测试集上的准确率为91.7%。决策边界及训练集如下：

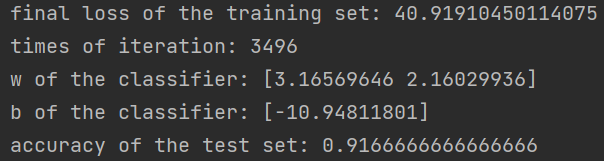


代价函数值随迭代次数变化的情况如下：

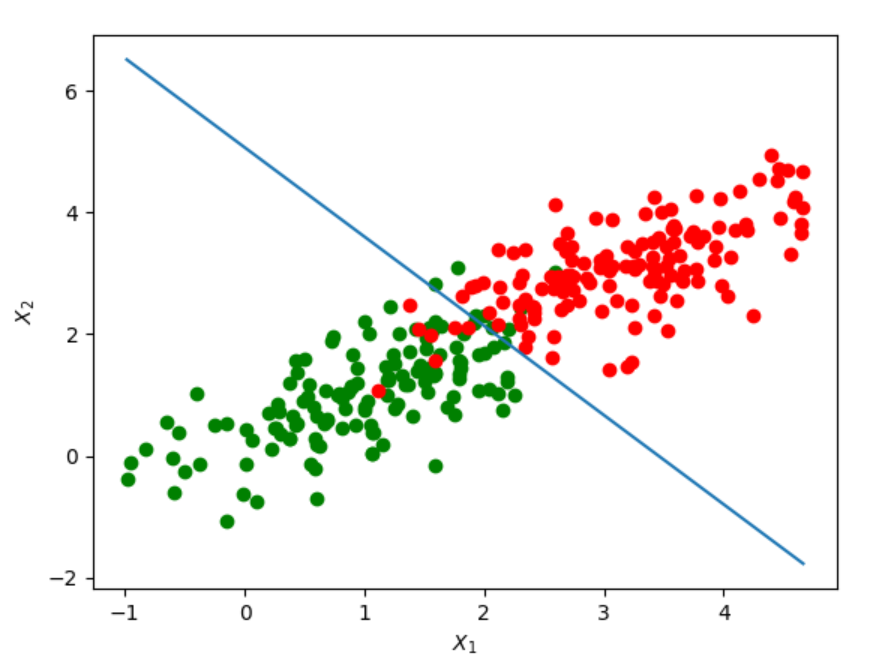


可以看到，相比于满足朴素贝叶斯假设的数据，测试集上的正确率有所下降。

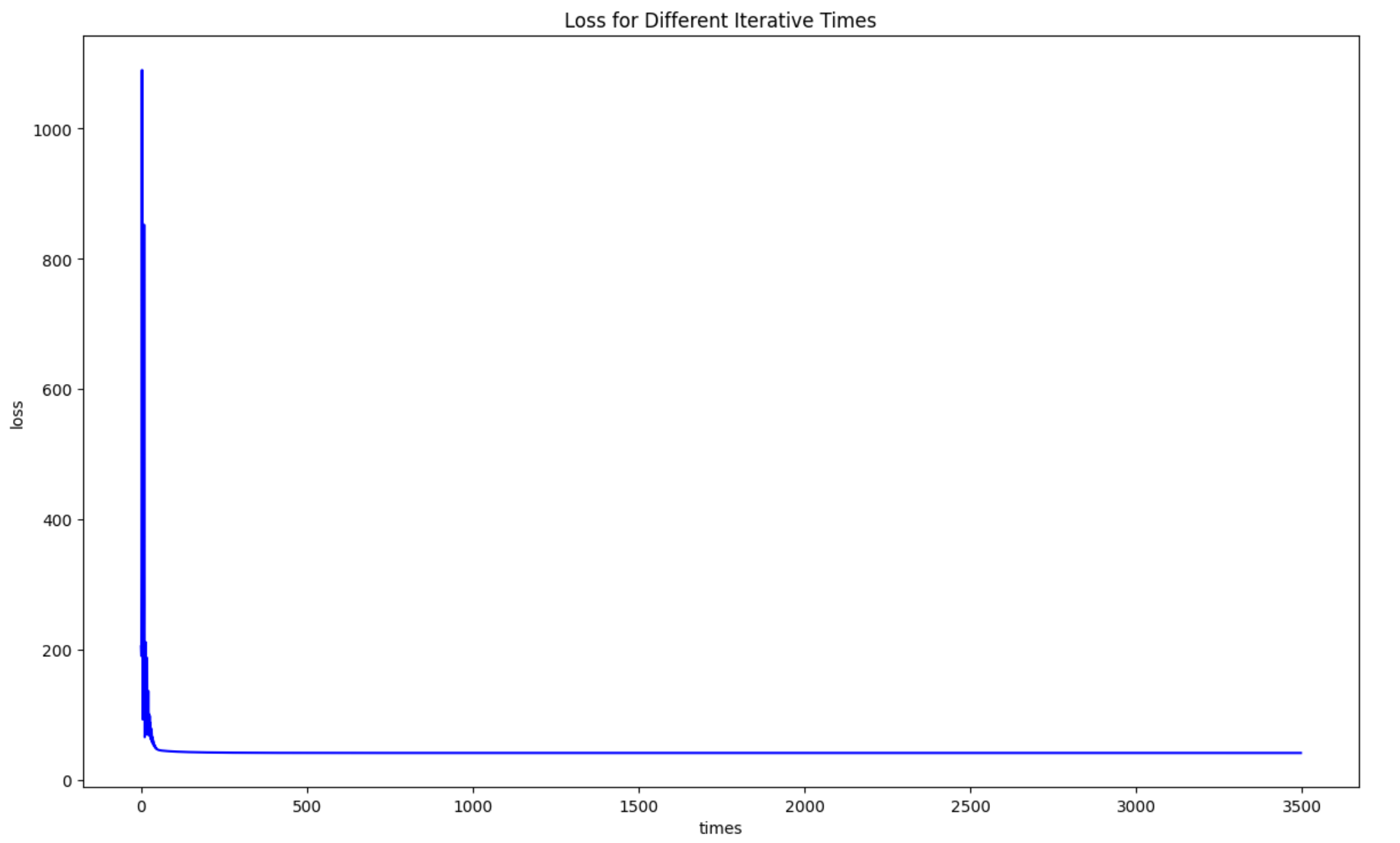
**4.不满足朴素贝叶斯假设，带惩罚项**



可见，与无惩罚项极其相似，迭代次数3496次，最终的代价函数值为40.92，在测试集上的准确率为91.7%。决策边界及训练集如下：



代价函数值随迭代次数变化的情况如下：

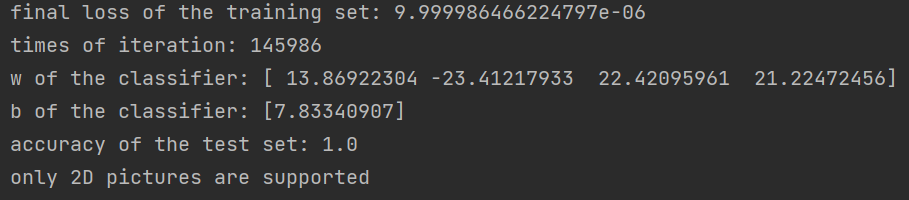


可以发现，同满足朴素贝叶斯假设的数据集一样，加上惩罚项之后也并无太大变化。

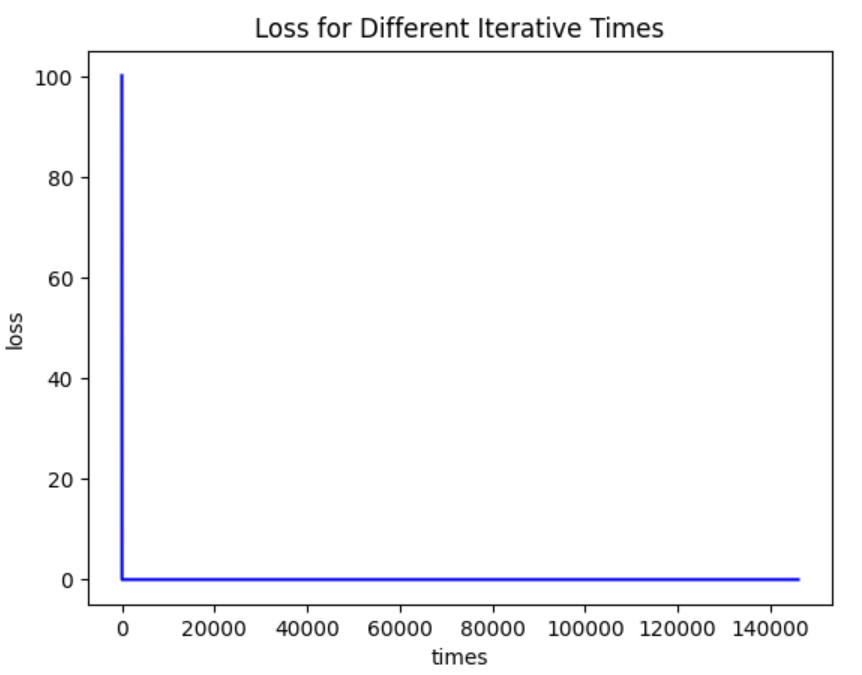
**（二）UCI数据**

首先在UCI官网上下载一个数据集iris.data，发现该数据集有三个标签：'Iris-setosa'，'Iris-versicolor' 和'Iris-virginica'，但由于上述实验实现的是二分类，所以将第一个分为0类，后两个分为1类。将数据导入后即可使用前述模型求解。梯度下降求解时的步长为0.48，惩罚项比重为1e-8，终止迭代的精度要求为1e-5。

**1.无惩罚项**

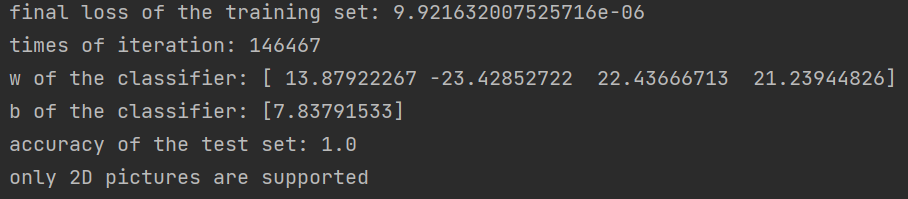


可见，迭代次数145986次，最终的代价函数值为9.999986466224797e-06，在测试集上的准确率为100%。由于数据维度为4，决策边界及训练集无法画图体现，但代价函数值随迭代次数变化的情况如下：

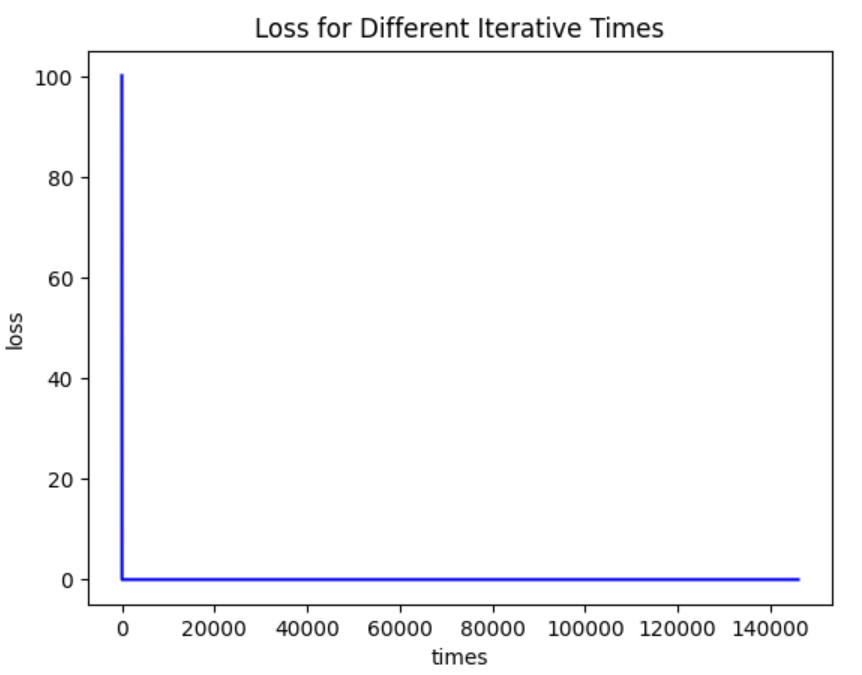


可以看到，迭代次数非常多，这是由于停止迭代的精度要求较高引起的，在这种情况下，即使代价函数值已经迅速降到极低了，仍然会继续迭代下去。同时，由于所获取的数据集本身就线性可分，所以模型训练得到的效果十分理想，测试集上的正确率达到100%。

**2.带惩罚项**



可见，与无惩罚项非常接近，迭代次数146467次，最终的代价函数值为9.921632007525716e-06，在测试集上的准确率为100%。由于数据维度为4，决策边界及训练集无法画图体现，但代价函数值随迭代次数变化的情况如下：



和前面一样，加入惩罚项依然没有太大变化。

1. **结论**

可以使用逻辑回归对数据进行二分类。

对于满足朴素贝叶斯假设的数据集，其分类的正确率会略高于不满足朴素贝叶斯假设的数据集。在待分类数据集较大时，向代价函数中加入惩罚项，对分类结果的影响并不显著。与多项式拟合正弦函数中的梯度下降法一样，该方法需要的迭代次数较多，而且当要求的精度高时，迭代次数会更多。

注意，由于逻辑回归模型并不需要知道数据的具体分布，而是直接根据已有的数据求解决策边界，该求解结果会受数据集的影响。

1. **参考文献**

周志华.机器学习[M].北京：清华大学出版社，2016

**七、附录：源代码（带注释）**

**1.main.py**

import numpy as np  
  
import mytool as mt  
import logisticregression as lr  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 np.random.seed(0)  
  
 class\_0\_number = 200 # 类0的数据集大小（含训练集与测试集）  
 class\_1\_number = 200 # 类1的数据集大小（含训练集与测试集）  
 eta = 0.48 # 梯度下降求解时的步长  
 train\_rate = 0.7 # 训练集大小占数据集大小的比例  
 times = 150000 # 梯度下降求解时的迭代次数上限  
  
 # 满足朴素贝叶斯的数据集  
 # x\_train, y\_train, x\_test, y\_test = mt.generate\_data(1, 3, 0.6, class\_0\_number, class\_1\_number, train\_rate)  
 # 不满足朴素贝叶斯的数据集  
 # x\_train, y\_train, x\_test, y\_test = mt.generate\_data(1, 3, 0.6, class\_0\_number, class\_1\_number, train\_rate, cov=0.4)  
 # 使用UCI上的数据  
 x\_train, y\_train, x\_test, y\_test = mt.load\_data("iris.data", train\_rate)  
  
 # 初始化逻辑回归分类器  
 classifier = lr.LogisticRegression(len(x\_train[0]))  
  
 # 损失函数无惩罚项  
 # loss\_list, t = classifier.solve(x\_train, y\_train, eta, times)  
 # 损失函数带惩罚项  
 loss\_list, t = classifier.solve(x\_train, y\_train, eta, times, lam=1e-8)  
  
 # 展示梯度下降法求得的w和b，最终结果下训练集的代价函数值，以及该分类器在测试集上的准确率  
 print("w of the classifier:", classifier.w)  
 print("b of the classifier:", classifier.b)  
 print("accuracy of the test set:", classifier.accuracy(x\_test, y\_test))  
  
 # 画出决策边界与测试集分布情况  
 x\_all = np.concatenate([x\_train[:, 0], x\_test[:, 0]])  
 classifier.draw\_border(min(x\_all), max(x\_all))  
  
 # 画出loss随迭代次数的变化情况  
 mt.draw\_loss\_line(t, loss\_list)

**2.logisticregression.py**

import numpy as np  
from matplotlib import pyplot as plt  
  
  
class LogisticRegression(object):  
  
 def \_\_init\_\_(self, dimension):  
 *"""  
 初始化逻辑回归分类器的参数* ***:param*** *dimension: 数据的维度  
 """* self.w = np.random.randn(dimension)  
 self.b = np.random.randn(1)  
  
 def sigmoid(self, x):  
 *"""  
 将数据值映射到sigmoid函数上* ***:param*** *x: 数据* ***:return****: 数据在当前参数取值下，映射到的sigmoid函数值  
 """* return 1 / (1 + np.e\*\*(-(x.dot(self.w) + self.b)))  
  
 def predict(self, x):  
 *"""  
 预测输入的数据属于哪一个类（二分类）* ***:param*** *x: 数据集* ***:return****: 数据所属类别，0为0类，1为1类  
 """* result = self.sigmoid(x)  
 result = np.where(result >= 0.5, 1, 0)  
 return result  
  
 def cal\_loss(self, x, y):  
 *"""  
 计算代价函数* ***:param*** *x: 数据集* ***:param*** *y: 各个数据的分类情况* ***:return****: 该组数据的代价函数值  
 """* return -sum(y \* (x.dot(self.w) + self.b) - np.log(1 + np.e\*\*(x.dot(self.w) + self.b)))  
  
 def cal\_gradient(self, x, y, lam):  
 *"""  
 计算w和b的梯度* ***:param*** *x: 数据集* ***:param*** *y: 各个数据的分类情况* ***:param*** *lam: 惩罚项比重* ***:return****: w和b的梯度  
 """* sig = self.sigmoid(x)  
 w\_gradient = -((y - sig).dot(x) + lam \* self.w)  
 b\_gradient = -(np.sum(y - sig) + lam \* self.b)  
 return w\_gradient, b\_gradient  
  
 def solve(self, x, y, eta, times, lam=0.0):  
 *"""  
 梯度下降法求解w和b* ***:param*** *x: 数据集* ***:param*** *y: 各个数据的分类情况* ***:param*** *eta: 迭代步长* ***:param*** *times: 迭代次数上限* ***:param*** *lam: 惩罚项比重* ***:return****: loss\_list, range(t + 1): 代价函数与迭代次数  
 """* loss\_list = [self.cal\_loss(x, y)]  
 w\_gradient, b\_gradient = self.cal\_gradient(x, y, lam)  
 t = 0  
 while not (np.all(np.absolute(w\_gradient) <= 1e-5) and np.all(np.absolute(b\_gradient) <= 1e-5)):  
 if t >= times:  
 break  
 self.w -= eta \* w\_gradient  
 self.b -= eta \* b\_gradient  
 loss\_list.append(self.cal\_loss(x, y))  
 w\_gradient, b\_gradient = self.cal\_gradient(x, y, lam)  
 t += 1  
 print("final loss of the training set:", self.cal\_loss(x, y))  
 print("times of iteration:", t)  
 return loss\_list, range(t + 1)  
  
 def accuracy(self, x, y):  
 *"""* ***:param*** *x: 数据集* ***:param*** *y: 各个数据的分类情况* ***:return****: 逻辑回归分类器在该组数据上的正确率  
 """* y\_pre = self.predict(x)  
 count = 0  
 for i in range(len(y)):  
 if y[i] == y\_pre[i]:  
 count += 1  
 return count / len(y)  
  
 def draw\_border(self, low, high):  
 *"""  
 画出逻辑回归分类器的决策边界（仅支持二维图）* ***:param*** *low: 横坐标最小值* ***:param*** *high: 横坐标最大值  
 """* if len(self.w) != 2:  
 print("only 2D pictures are supported")  
 return  
 x = np.linspace(low, high, 1000)  
 y = (-self.b - self.w[0] \* x) / self.w[1]  
 plt.plot(x, y)  
 plt.show()

**3.mytool.py**

import numpy as np  
from matplotlib import pyplot as plt  
from sklearn.utils import shuffle  
  
  
def generate\_data(mu\_0, mu\_1, sigma, n\_0, n\_1, train\_rate, cov=0.0):  
 *"""  
 生成两个类别的数据（高斯分布），默认生成的是满足朴素贝叶斯的数据（cov=0.0），若要生成不满足朴素贝叶斯的数据，则需传入另外的cov值* ***:param*** *mu\_0: 类别0中数据的均值（默认各维度均值相等）* ***:param*** *mu\_1: 类别1中数据的均值（默认各维度均值相等）* ***:param*** *sigma: 两个类别中数据的标准差（默认各维度标准差相等）* ***:param*** *n\_0: 类别0中数据量* ***:param*** *n\_1: 类别1中数据量* ***:param*** *train\_rate: 每个类别中训练集数据占总数据的比例（剩下部分为测试集）* ***:param*** *cov: 数据两个维度(x和y)的协方差，即cov(x,y)和cov(y,x)，默认情况下为零，独立，满足朴素贝叶斯* ***:return****: x\_train,y\_train,x\_test,y\_test: 生成的训练集与测试集  
 """* # 分别生成两个类别的数据  
 data\_1 = np.random.multivariate\_normal((mu\_0, mu\_0), [[sigma, cov], [cov, sigma]], n\_0)  
 data\_2 = np.random.multivariate\_normal((mu\_1, mu\_1), [[sigma, cov], [cov, sigma]], n\_1)  
 # 将训练集与测试集划分开  
 data\_sep\_1 = int(train\_rate \* n\_0)  
 data\_sep\_2 = int(train\_rate \* n\_1)  
 data\_1\_train, data\_1\_test = data\_1[:data\_sep\_1], data\_1[data\_sep\_1:]  
 data\_2\_train, data\_2\_test = data\_2[:data\_sep\_2], data\_2[data\_sep\_2:]  
 # 将两个类别的数据都画到图上  
 plt.scatter(data\_1\_train.T[0], data\_1\_train.T[1], color='g')  
 plt.xlabel('$X\_{1}$')  
 plt.scatter(data\_2\_train.T[0], data\_2\_train.T[1], color='r')  
 plt.ylabel('$X\_{2}$')  
 # 将两个类别中的训练集数据与测试集数据分别合在一起  
 x\_train = np.concatenate([data\_1\_train, data\_2\_train])  
 y\_train = np.concatenate([np.zeros(data\_1\_train.shape[0]), np.ones(data\_2\_train.shape[0])])  
 x\_test = np.concatenate([data\_1\_test, data\_2\_test])  
 y\_test = np.concatenate([np.zeros(data\_1\_test.shape[0]), np.ones(data\_2\_test.shape[0])])  
 # 将训练集中数据打乱  
 x\_train, y\_train = shuffle(x\_train, y\_train)  
 return x\_train, y\_train, x\_test, y\_test  
  
  
def load\_data(path, train\_rate):  
 *"""  
 使用从UCI获取的数据集* ***:param*** *path: 数据集所在路径* ***:param*** *train\_rate: 每个类别中训练集数据占总数据的比例（剩下部分为测试集）* ***:return****: x\_train,y\_train,x\_test,y\_test: 获取的训练集与测试集  
 """* data = []  
 file = open(path, encoding='utf-8')  
 for line in file:  
 data.append(line.strip('\n').split(sep=','))  
 # 将数据分为0类和1类  
 all\_data = np.array(data)  
 all\_data = np.where(all\_data == 'Iris-setosa', 0, all\_data)  
 all\_data = np.where(all\_data == 'Iris-versicolor', 1, all\_data)  
 all\_data = np.where(all\_data == 'Iris-virginica', 1, all\_data)  
 # 将数据与类别标签分开  
 x = all\_data[:, :len(all\_data[0]) - 1]  
 y = all\_data[:, -1]  
 x = np.array(x, dtype=np.float32)  
 y = np.array(y, dtype=int)  
 # 把数据集进行归一化处理  
 x = (x-np.mean(x, axis=0))/(np.std(x, axis=0))  
 # 将数据集打乱并分为训练集和测试集  
 data\_sep = int(train\_rate \* len(all\_data))  
 x, y = shuffle(x, y)  
 x\_train = x[:data\_sep, :]  
 x\_test = x[data\_sep:, :]  
 y\_train = y[:data\_sep]  
 y\_test = y[data\_sep:]  
 return x\_train, y\_train, x\_test, y\_test  
  
  
def draw\_loss\_line(t, loss\_list):  
 *"""  
 画出loss随迭代次数的变化情况* ***:param*** *t: 横坐标，迭代次数* ***:param*** *loss\_list: 纵坐标，代价函数值  
 """* fig, axes = plt.subplots()  
 axes.plot(t, loss\_list, 'b')  
 # 设置图名  
 title = "Loss for Different Iterative Times"  
 props = {'title': title, 'xlabel': 'times', 'ylabel': 'loss'}  
 axes.set(\*\*props)  
 plt.show()