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。

二、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$D=\{(\boldsymbol{X}_1,Y_1),(\boldsymbol{X}_2,Y_2),\dots,(\boldsymbol{X}_m,Y_m)\}$，其中$\boldsymbol{X}_i=(X_{i1};X_{i2};\dots;X_{id})$，$Y_i\in\boldsymbol{R}$。逻辑回归模型能从该训练集中学习得到一个分类器$f:\boldsymbol{X}\rightarrow Y$，以便对新的数据进行分类。

假设$\boldsymbol{X}=(X_1;X_2;\dots;X_d)$中每一维都是独立同分布的，且所有的$P(X_i|Y=Y_k)$为高斯分布$N(\mu_{ik},\sigma_i)$，$P(Y)$为伯努利分布$Bernoulli(\pi)$，则对于二分类问题，根据贝叶斯公式有：

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

其中$\sum_i ln\frac{P(X_i|Y=1)}{P(X_i|Y=0)}$经计算可以表示为$\sum_i(\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}x_i+\frac{\mu_{i0}^2-\mu_{i1}^2}{2\sigma_i^2})$,则等式可进一步转换为：

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

进而得到：

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

若将p视为样本$\boldsymbol{X}$作为第0类的可能性，则$1-p$是其为第1类的可能性，两者的比值$\frac{p}{1-p}$称为“几率”，反映了$\boldsymbol{X}$作为第0类的相对可能性，对几率取对数则得到“对数几率”$ln\frac{p}{1-p}$，记为$logit(p)$。

将上述推导所得$P(Y=1|\boldsymbol{X})$代入$logit(p)$可得$logit\big(P(Y=1|\boldsymbol{X})\big)=w_0+\sum_{i=1}^{n}w_iX_i$，若$logit\big(P(Y=1|\boldsymbol{X})\big)>0$，则将$\boldsymbol{X}$分到第1类，若$logit\big(P(Y=1|\boldsymbol{X})\big)<0$，则将$\boldsymbol{X}$分到第0类。此时，得到决策边界为$w_0+\sum_{i=1}^{n}w_iX_i=0$。

当然，为了简化运算，在对样本分类的时候，也可以直接计算$sigmoid$函数值，若$sigmoid$的值大于等于0.5，则将其分到第1类，否则为第0类。

若要求解参数$\boldsymbol{W}$，用最大似然估计（MLE）得到式子如下：

$$\boldsymbol{W}_{MLE}=arg\max_{\boldsymbol{W}}P(<\boldsymbol{X}^1,Y^1>,<\boldsymbol{X}^2,Y^2>,\dots,<\boldsymbol{X}^L,Y^L>|\boldsymbol{W})$$

$$=arg\max_{\boldsymbol{W}}\prod_{l}P(<\boldsymbol{X}^l,Y^l>|\boldsymbol{W})$$

采用最大条件似然估计（MCLE）则能将式子写为：

$$\boldsymbol{W}_{MCLE}=\arg\max_{\boldsymbol{W}}\prod_{l}P(Y^l|\boldsymbol{X}^l,\boldsymbol{W})$$

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：

$$\boldsymbol{W}_{MCLE}=\arg\max_{\boldsymbol{W}}\ln\prod_{l}P(Y^l|\boldsymbol{X}^l,\boldsymbol{W})$$

再将上式添加一个负号，即可定义代价函数：

$$l(\boldsymbol{W})=-\ln\prod_{l}P(Y^l|\boldsymbol{X}^l,\boldsymbol{W})$$

$$=-\sum_{l}\ln P(Y^l|\boldsymbol{X}^l,\boldsymbol{W})$$

$$=-\sum_{l}(Y^l\ln P(Y^l=1|\boldsymbol{X}^l,\boldsymbol{W})+(1-Y^l)\ln P(Y^l=0|\boldsymbol{X}^l,\boldsymbol{W}))$$

$$=-\sum_{l}(Y^l\ln\frac{P(Y^l=1|\boldsymbol{X}^l,\boldsymbol{W})}{P(Y^l=0|\boldsymbol{X}^l,\boldsymbol{W})}+\ln P(Y^l=0|\boldsymbol{X}^l,\boldsymbol{W}))$$

$$=-\sum_{l}(Y^l\left(w_0+\sum_{i}^{n}w_iX_i^l\right)-\ln(1+\exp(w_0+\sum_{i}^{n}w_iX_i^l)))$$

则当$l(\boldsymbol{W})$有最小值时，对应的$\boldsymbol{W}$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$l(\boldsymbol{W})$求导：

$$\frac{\partial l(\boldsymbol{W})}{\partial w_i} = -\sum_l X_i^l\left(Y^l - P(Y^l = 1|\boldsymbol{X}^l, \boldsymbol{W})\right)$$

然后更新w_i值（其中η为步长，下同）：

$$w_i \leftarrow w_i - \eta(-\sum_l X_i^l\left(Y^l - P(Y^l = 1|\boldsymbol{X}^l, \boldsymbol{W})\right))$$

如果考虑在代价函数中加入惩罚项，则有（其中λ为惩罚项比重，下同）：

$$l(\boldsymbol{W}) = -\sum_l\left(Y^l\left(w_0 + \sum_i^n w_i X_i^l\right) - \ln\left(1 + \exp\left(w_0 + \sum_i^n w_i X_i^l\right)\right)\right) + \frac{\lambda}{2}\boldsymbol{W}^T\boldsymbol{W}$$

求导得：

$$\frac{\partial l(\boldsymbol{W})}{\partial w_i} = -\sum_l X_i^l\left(Y^l - P(Y^l = 1|\boldsymbol{X}^l, \boldsymbol{W})\right) + \lambda w_i$$

更新w_i值：

$$w_i \leftarrow w_i - \eta(-\sum_l X_i^l\left(Y^l - P(Y^l = 1|\boldsymbol{X}^l, \boldsymbol{W})\right) + \lambda w_i)$$

四、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(X_1, X_2)就可以在二维平面直角坐标系下进行展示。

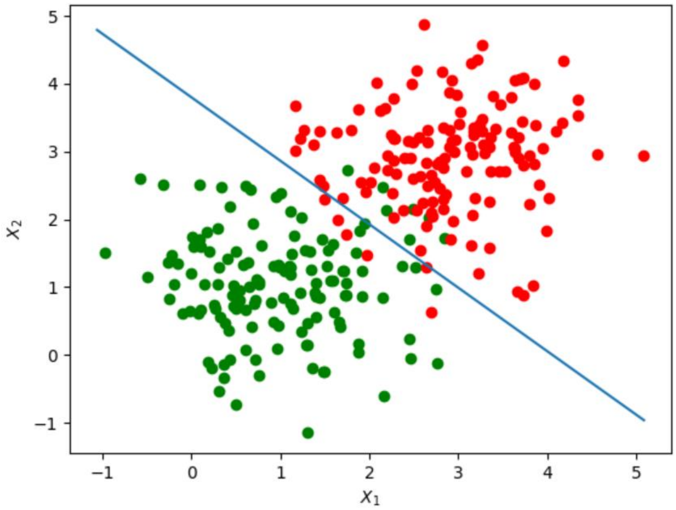
对于满足朴素贝叶斯假设的数据，其两个属性之间没有相关关系，协方差为 0；对于不满足朴素贝叶斯假设的数据，其两个属性之间没有相关关系，协方差应该满足$cov(X_1, X_2) = cov(X_2, X_1) \neq 0$。

对于下列手动生成的数据，类 0 中的数据均值为(1,1)，类 1 中的数据均值为(3,3)。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$cov(X_1, X_2) = cov(X_2, X_1) = 0.4$。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。

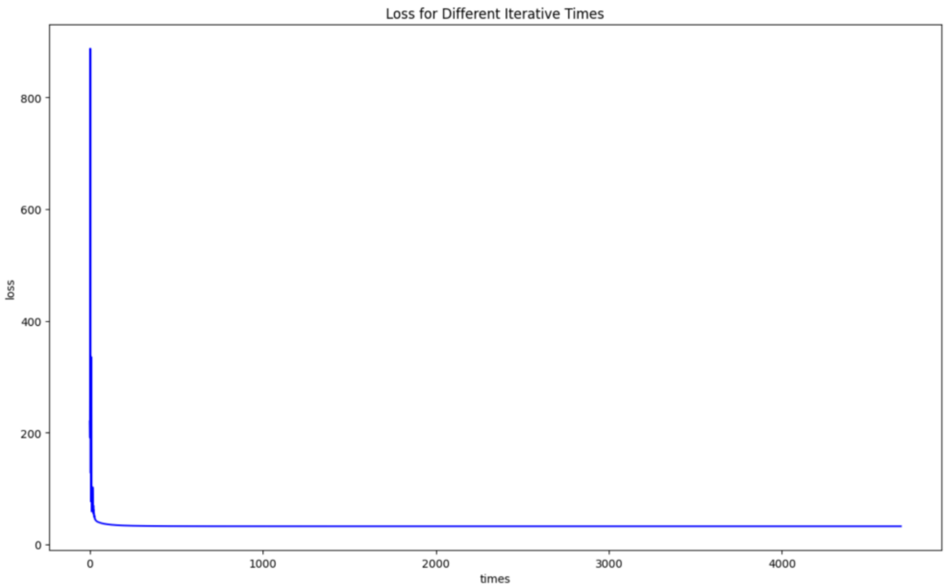
1.满足朴素贝叶斯假设，无惩罚项

```
final loss of the training set: 32.14083513498828
times of iteration: 4688
w of the classifier: [2.95246747 3.15566445]
b of the classifier: [-11.99381034]
accuracy of the test set: 0.9833333333333333
```

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：

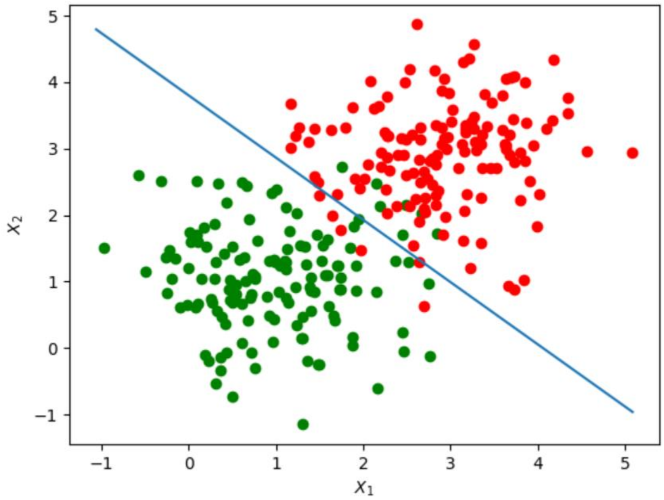


可以看到，由于两类数据分散得较为明显，所以逻辑回归模型得到的分类结果是很理想的，在测试集上的正确率很高。代价函数也能在迭代过程当中迅速下降到一个较小值，虽然后期收敛到的值依然较大，但这是由于两类数据的方差较大导致的，若将方差值减小，则会得到更小的代价函数值。

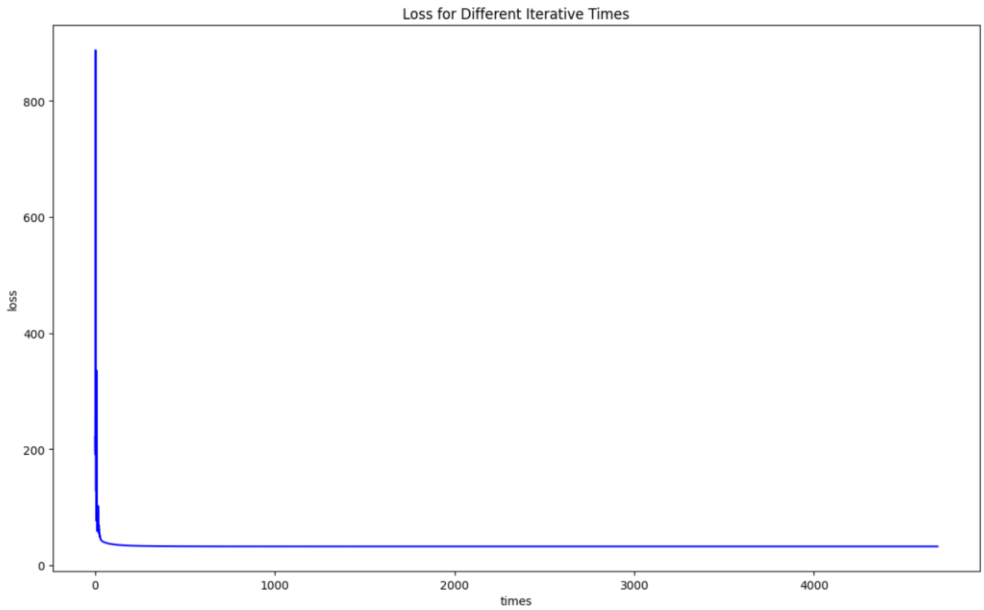
2.满足朴素贝叶斯假设，带惩罚项

```
final loss of the training set: 32.14083513498269
times of iteration: 4688
w of the classifier: [2.95246758 3.15566458]
b of the classifier: [-11.99381085]
accuracy of the test set: 0.9833333333333333
```

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：

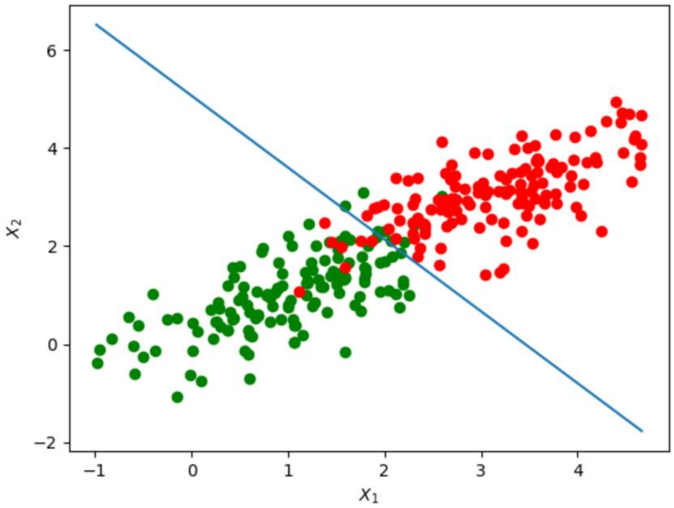


可以发现，加上惩罚项之后并无太大变化，这应该是数据集较大引起的，大量的数据和加入惩罚项能起到相同的作用。

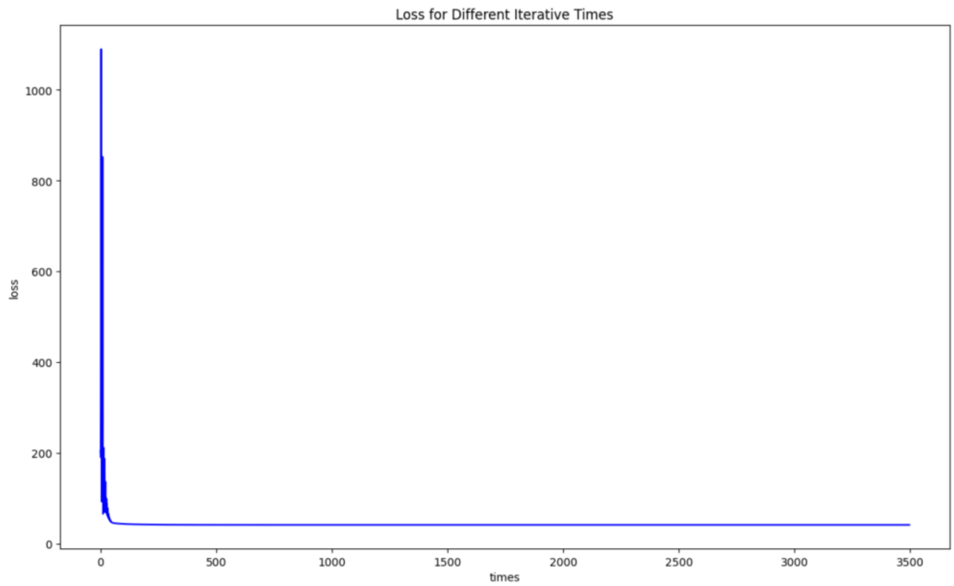
3.不满足朴素贝叶斯假设，无惩罚项

```
final loss of the training set: 40.91910450114444
times of iteration: 3496
w of the classifier: [3.16569636 2.16029929]
b of the classifier: [-10.94811768]
accuracy of the test set: 0.9166666666666666
```

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：

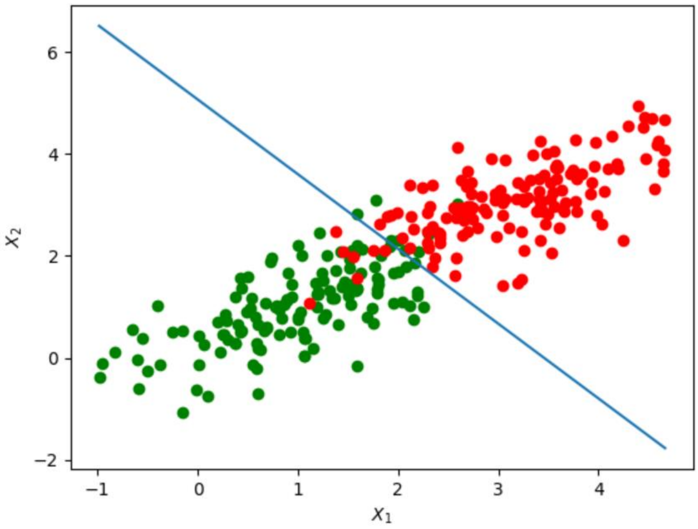


可以看到，相比于满足朴素贝叶斯假设的数据，测试集上的正确率有所下降。

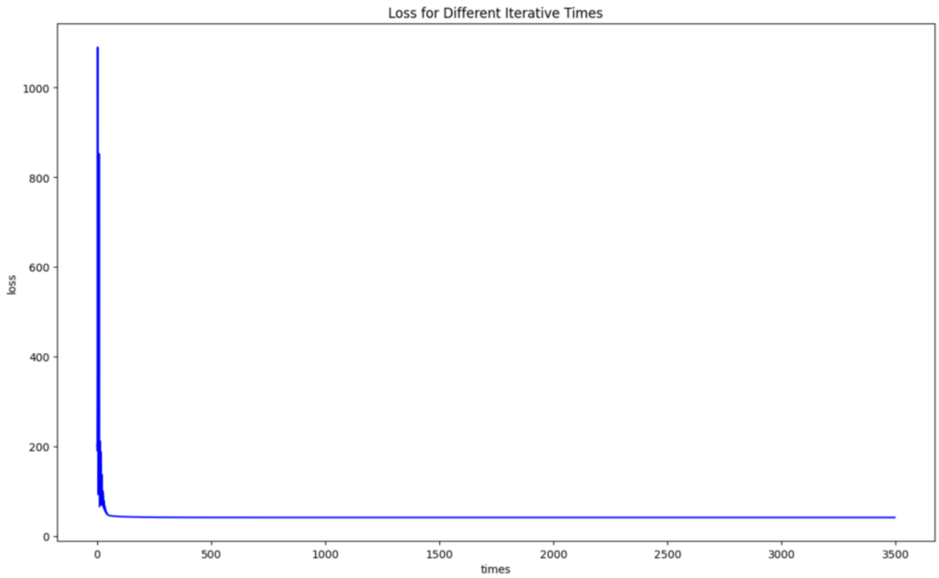
4.不满足朴素贝叶斯假设，带惩罚项

```
final loss of the training set: 40.91910450114075
times of iteration: 3496
w of the classifier: [3.16569646 2.16029936]
b of the classifier: [-10.94811801]
accuracy of the test set: 0.9166666666666666
```

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。

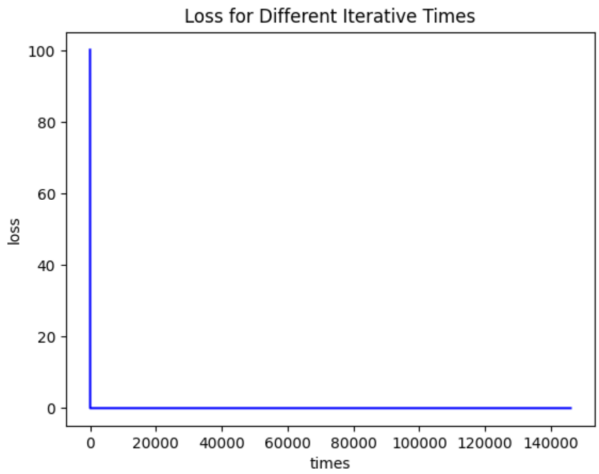
（二）UCI 数据

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

```
final loss of the training set: 9.999986466224797e-06
times of iteration: 145986
w of the classifier: [ 13.86922304 -23.41217933  22.42095961  21.22472456]
b of the classifier: [7.83340907]
accuracy of the test set: 1.0
only 2D pictures are supported
```

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：

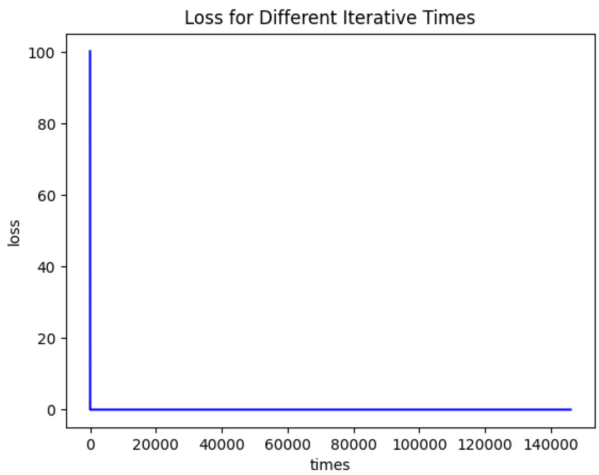


可以看到，迭代次数非常多，这是由于停止迭代的精度要求较高引起的，在这种情况下，即使代价函数值已经迅速降到极低了，仍然会继续迭代下去。同时，由于所获取的数据集本身就线性可分，所以模型训练得到的效果十分理想，测试集上的正确率达到 100%。

2.带惩罚项

```
final loss of the training set: 9.921632007525716e-06
times of iteration: 146467
w of the classifier: [ 13.87922267 -23.42852722  22.43666713  21.23944826]
b of the classifier: [7.83791533]
accuracy of the test set: 1.0
only 2D pictures are supported
```

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。

五、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。

六、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)
```

```python
    # 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

```python
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
```

```python
        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

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

```python
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
```

```python


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
```

```python
title = "Loss for Different Iterative Times"
props = {'title': title, 'xlabel': 'times', 'ylabel': 'loss'}
axes.set(**props)
plt.show()
```