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:	必修
实验题目:	逻辑斯蒂回归

姓名:	黄海
学号:	1160300329

一、实验目的

机器学习第二次实验 逻辑斯蒂回归

通过多种途径实现对满足伯努利分布的样本进行简单的线性分类。

实现功能：

- [x] 1. 无惩罚项梯度下降
- [x] 2. 无惩罚项牛顿法
- [x] 3. 有惩罚项梯度下降
- [x] 4. 有惩罚项牛顿法

二、实验要求及实验环境

实验要求

- 不能使用现有的逻辑回归第三方库

实验环境

- macOS 10.13.6
- python 3.7.0 64-bit
- Visual Studio Code 12.8.2

三、设计思想

1.算法原理

逻辑斯蒂回归的主要实现方式有两种，一是梯度下降算法，二是牛顿法。后者相对于前者的主要优势是速度快，迭代次数少。

梯度下降的主要算法依旧是对于多项式的梯度计算，然后在梯度方向来进行下降以找到更好的结果（即得到的函数对于所有样本点而言进行了较好的划分）。

对于得到的划分我们可以表示为

$$0 = w_0 + w_1x_1 + w_2x_2 + \cdots + w_mx_m \tag{1}$$

这在二维上表现为直线。

特别的对于二维特征，我们可以表示为

$$Data=\begin{bmatrix}x_{11} & x_{12}\\ x_{21} & x_{22}\\ x_{31} & x_{32}\\ \vdots & \vdots\\ x_{n1} & x_{n2}\end{bmatrix},Result=\begin{bmatrix}y_1\\ y_2\\ y_3\\ \vdots\\ y_n\end{bmatrix} \tag{2}$$

其中$Data$为二维特征，$Result$为每个样本点的类型，后者只有0和1之分（严格的伯努利分布）。

从概率上讲，一件事发生的概率为p的话，那么不发生的概率就是$1-p$，对于样本而言，属于第一类的概率符合这个理论，只不过这里的概率非0即1。我们定义这件事发生的几率，并给出表达

$$Logit(p)=\frac{p}{1-p} \tag{3}$$

那么我们将此带入(1)中有

$$\begin{aligned}Logit(P(y=1|x))&=w_1x_1+w_2x_2+w_3x_3+\cdots+w_nx_n\\&=\sum_{i=1}^{n}w_ix_i\\&=W^TX\end{aligned}$$

其中

$$W=\begin{bmatrix}w_1\\ w_2\\ w_3\\ \vdots\\ w_n\end{bmatrix},X=Data$$

输入所有的样本之后为了实现划分，我们其实都设置了一个限制，即超过多少的概率时为1反之为0。我们有以下函数计算概率

$$\phi(z)=\frac{1}{1+e^z} \tag{4}$$

称之为$sigmoid$函数。

致此我们终于可以通过函数来计算概率并且进行判断。

在线性感知器中，我们通过梯度下降算法来使得预测值与实际值的误差的平方和最小，来求得权重和阈值。而在逻辑斯蒂回归中所定义的代价函数就是使得该件事情发生的几率最大，也就是某个样本属于其真实标记样本的概率越大越好。这里我们使用梯度上升。

我们有最大似然函数

$$L(\omega)=P(y|x;\omega)=\prod_{i=1}^{n}P(y^{(i)}|x^{(i)};\omega)=(\phi(z^{(i)}))^{y^{(i)}}(1-\phi(z^{(i)}))^{1-y^{(i)}}$$

为了防止溢出取对数

$$l(\omega) = L(\omega) = P(y|x;\omega) = \sum_{i=1}^{n}(y^{(i)}log(\phi(z^{(i)})) + (1-y^{(i)})log(1-\phi(z^{(i)})) \tag{5}$$

为了判断是否足够收敛，我们定义$loss$

$$J(\phi(z), y; \omega) = \begin{cases} log(\phi(z)), & y = 1, \\ log(1-\phi(z)), & y = 0. \end{cases} \tag{6}$$

式(5)与式(6)相等。

在进行了最初的计算之后我们需要对权重W进行更新。

我们对似然函数求偏导数得到

$$\frac{\partial}{\partial \omega_j} l(\omega) = (y - \phi(z))x_j$$

这便是梯度，更新后的W为

$$W = W - \frac{1}{m}(Data^T \cdot (Result - \phi(z)))$$

添加正则化惩罚之后变为

$$J(\omega) = l(\omega) + \frac{\lambda}{2m}||\omega||_2^2$$
$$W = W - \frac{1}{m}(Data^T \cdot (Result - \phi(z))) - \frac{\lambda}{m}W$$

以上所有均适用于

$$z = X^T \cdot W$$

对于牛顿法，主要是求得二阶导数（$Hessian$）来确定函数下一个下降的点是哪里，使得越来越靠近最优结果。

同样的对于权重的更新我们有

$$W = W - H^{-1} \bigtriangledown_\theta J$$

$$H = Data^T \cdot diag(\phi(z)) \cdot diag(1 - \phi(z)) \cdot Data$$

$$\bigtriangledown_\theta J = Data^T \cdot (1 - \phi(z))$$

其中

$$diag((a, b)) = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

同样的加入正则后

$$\bigtriangledown_J = Data^T \cdot (1 - \phi(z)) - \frac{\lambda}{m}W$$

$$H = H = Data^T \cdot diag(\phi(z)) \cdot diag(1-\phi(z)) \cdot Data + \frac{\lambda}{m}\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

至此所有公式完成。

2.算法的实现

这里只展现加入正则后的所有函数即代码，我们约定$data_mat$为参数矩阵，$result$为类型矩阵。

对于加入正则的梯度下降，在计算出加入正则的权重更新函数之后，必须对于loss也加入正则项使得在必要的地方停止迭代。

梯度下降

```
def logistic_regression_Gradient_Descent_with_lambda(data_mat, result
, w = WN, Lambda=LAMBDA_G):
    """
    logistic_regression_Gradient_Descent_with_lambda 正则化梯度下降实现
逻辑斯蒂回归

    [description]

    Args:
        data_mat (np.mat): 样本点矩阵 每一行为一个样本的数据
        result (np.mat): 列矩阵 每个样本点的类别 只有01
        w (int, optional): Defaults to WN. 参数维度 两项特征则使用三个参数
 类推
        Lambda (double, optional): Defaults to LAMBDA_G. Lambda正则化
惩罚参数

    Returns:
        np.mat, int, np.mat: 分别是参数列矩阵，迭代次数，loss
    """
    re = 10
    count = 0
    size_y = np.size(data_mat, 0)
    y0 = np.ones((size_y, 1))
    X = np.ones((size_y, 1))
    X = np.hstack((X, data_mat))
    y = result
    W = np.zeros((w, 1))

    while abs(re) > EPSILON and count < LOOP_MAX:
        re = 0
```

```
        C = X * W
        Y_W = 1.0 / (1 + np.exp(-C))
        y_1 = y.T * np.log2(Y_W)
        y_2 = (y0 - y).T * np.log2(y0 - Y_W)
        lam = Lambda / (2 * w) * np.sum(np.power(W, 2))
        l = y_1 + y_2 + lam
        W = W - (ALPHA / w) * (X.T * (Y_W - y) + Lambda * W)
        re = l
        count += 1
        # print(W, re, count)
    print('logistic_regression_Gradient_Descent_with_lambda:\n', W, r
e, count)
    return W, count, re
```

牛顿法

```
def logistic_regression_Newton_methon_with_lambda(data_mat, result, w
 = WN, Lambda=LAMBDA_N):
    """
    logistic_regression_Newton_methon_with_lambda 正则化牛顿法实现逻辑斯
蒂回归

    [description]

        Args:
        data_mat (np.mat): 样本点矩阵 每一行为一个样本的数据
        result (np.mat): 列矩阵 每个样本点的类别 只有01
        w (int, optional): Defaults to WN. 参数维度 两项特征则使用三个参数
 类推
        Lambda (double, optional): Defaults to LAMBDA_N. Lambda正则化
惩罚参数

    Returns:
        np.mat, int, np.mat: 分别是参数列矩阵，迭代次数，loss
    """
    re = 10
    count = 0
    size_y = np.size(data_mat, 0)
    y0 = np.ones((size_y, 1))
    X = np.ones((size_y, 1))
    X = np.hstack((X, data_mat))
    y = result
    W = np.zeros((w, 1))
    A = np.eye(w)
    A[0][0] = 0

    while abs(re) > EPSILON and count < LOOP_MAX:
        re = 0
```

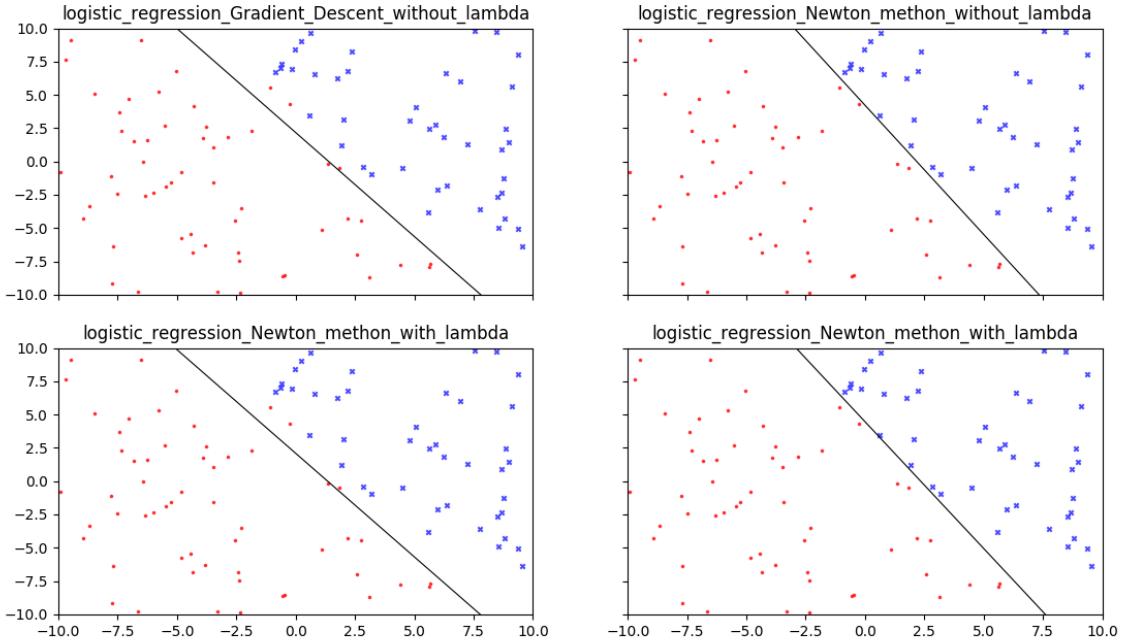
```
            C = X * W
            Y_W = 1.0 / (1 + np.exp(-C))
            y_1 = y.T * np.log2(Y_W)
            y_2 = (y0 - y).T * np.log2(y0 - Y_W)
            H = X.T * np.mat(np.diag(Y_W.T.tolist()[0])) * np.mat(np.diag
((y0 - Y_W).T.tolist()[0])) * X + Lambda / w * A
            S = X.T * (Y_W - y) - Lambda / w * W
            l = y_1 + y_2
            W = W - (H.I * S)
            re = l
            count += 1
            # print(W, re, count)
        print('logistic_regression_Newton_methon_with_lambda:\n', W, re,
count)
        return W, count, re
```

四、实验结果分析

以下所有的结果参数

- 样本个数 100
- 梯度下降Lambda 0.5
- 牛顿法Lambda 0.5
- 学习速度 0.001
- 样本生成直线$y = -2x + 4$

以下为结果



很明显可以看出，在梯度下降中，迭代多次的结果依旧有错误产生。而牛顿法在经历个位数迭代之后便给出了全正确结果。可见起迭代速度之快。

五、结论

在不考虑迭代时间的情况下，梯度下降和牛顿法都可以得出很完美的结果。但是在资源有限的情况下，后者的迭代速度超乎想象，并且效果很好。

在大型数据测试中，由于存在噪声，正确度只能达到97%左右。

六、参考文献

[1]M.JordanPattern recognition and machine learning[M]
[2]https://blog.csdn.net/sinat_29957455/article/details/78944939
[3]https://en.wikipedia.org/wiki/Logistic_regression

七、附录:源代码(带注释)

```
# -*- coding: utf-8 -*-
"""
逻辑斯蒂回归
生成两个类别的数据进行实验

- [x] 1. 无惩罚项梯度下降
- [x] 2. 无惩罚项牛顿法
- [x] 3. 有惩罚项梯度下降
- [x] 4. 有惩罚项牛顿法
- [x] 5. UCI数据验证
"""

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


# 数据点起点终点
DATA_HEAD_X = -10
DATA_END_X = 10
DATA_HEAD_Y = -10
DATA_END_Y = 10

# 直线参数
A = -2
B = 4
```

```
# 参数维度
WN = 3

# 数据点个数
DATA_NUM = 100

# 惩罚项
LAMBDA_G = 0.5
LAMBDA_N = 0.5

# 最小误差
EPSILON = 1

# 最大迭代次数
LOOP_MAX = 1000

# 学习速度
ALPHA = 0.001

def original_data(data_head_x=DATA_HEAD_X, data_end_x=DATA_END_X, dat
a_head_y=DATA_HEAD_Y, data_end_y=DATA_END_Y, line_a = A, line_b = B,
data_num=DATA_NUM):
    """
    original_data 根据某条直线严格的生成具体的两类数据

    [description]
        data_head_x (int, optional): Defaults to DATA_HEAD_X. 从此处开
始生成数据
        data_end_x (int, optional): Defaults to DATA_END_X. 到此处结束
生成数据
        data_head_y (int, optional): Defaults to DATA_HEAD_Y. 从此处开
始生成数据
        data_end_y (int, optional): Defaults to DATA_END_Y. 到此处结束
生成数据
        line_a (double, optional): Defaults to A. 生成样本依据的直线的一
次项参数
        line_b (double, optional): Defaults to B. 生成样本依据的直线的二
次项参数
        data_num (int, optional): Defaults to DATA_NUM. 生成的样本点个
数

    Returns:
        list, list, mat, mat: 生成的数据分别是x的数据点，y的数据点，xy组成的
DATA_NUM*2矩阵，类别矩阵。
    """
    list_x = []
    list_y = []
    for i in range(0, data_num):
        list_x.append(random.uniform(data_head_x, data_end_y))
        list_y.append(random.uniform(data_head_y, data_end_y))
    result = []
```

```
    for i in range(data_num):
        if(list_x[i] * line_a + line_b >= list_y[i]):
            result.append(1)
        else:
            result.append(0)
    data_mat = np.vstack((np.mat(list_x), np.mat(list_y)))
    result = np.mat(result)
    return list_x, list_y, data_mat.T, result.T


def logistic_regression_Gradient_Descent_without_lambda(data_mat, res
ult, w = WN):
    """
    logistic_regression_Gradient_Descent_without_lambda 无正则化的梯度
下降实现逻辑斯蒂回归

    [description]

    Args:
        data_mat (np.mat): 样本点矩阵 每一行为一个样本的数据
        result (np.mat): 列矩阵 每个样本点的类别 只有01
        w (int, optional): Defaults to WN. 参数维度 两项特征则使用三个参数
 类推

    Returns:
        np.mat, int, np.mat: 分别是参数列矩阵，迭代次数，loss
    """
    re = 10
    count = 0
    size_y = np.size(data_mat, 0)
    y0 = np.ones((size_y, 1))
    X = np.ones((size_y, 1))
    X = np.hstack((X, data_mat))
    y = result
    W = np.zeros((w, 1))

    while abs(re) > EPSILON and count < LOOP_MAX:
        re = 0
        C = X * W
        Y_W = 1.0 / (1 + np.exp(-C))
        y_1 = y.T * np.log2(Y_W)
        y_2 = (y0 - y).T * np.log2(y0 - Y_W)
        l = y_1 + y_2
        W = W - (ALPHA / w) * X.T * (Y_W - y)
        re = l
        count += 1
        # print(W, re, count)
    print('logistic_regression_Gradient_Descent_without_lambda:\n', W
, re, count)
    return W, count, re
```

```
def logistic_regression_Newton_methon_without_lambda(data_mat, result
, w = WN):
    """
    logistic_regression_Newton_methon_without_lambda 无正则化牛顿法实现
逻辑斯蒂回归

    [description]

    Args:
        data_mat (np.mat): 样本点矩阵 每一行为一个样本的数据
        result (np.mat): 列矩阵 每个样本点的类别 只有01
        w (int, optional): Defaults to WN. 参数维度 两项特征则使用三个参数
 类推

    Returns:
        np.mat, int, np.mat: 分别是参数列矩阵，迭代次数，loss
    """
    re = 10
    count = 0
    size_y = np.size(data_mat, 0)
    y0 = np.ones((size_y, 1))
    X = np.ones((size_y, 1))
    X = np.hstack((X, data_mat))
    y = result
    W = np.zeros((w, 1))

    while abs(re) > EPSILON and count < LOOP_MAX:
        re = 0
        C = X * W
        Y_W = 1.0 / (1 + np.exp(-C))
        y_1 = y.T * np.log2(Y_W)
        y_2 = (y0 - y).T * np.log2(y0 - Y_W)
        # s = 1.0 / w
        H = X.T * np.mat(np.diag(Y_W.T.tolist()[0])) * np.mat(np.diag
((y0 - Y_W).T.tolist()[0])) * X
        S = X.T * (Y_W - y)
        l = y_1 + y_2
        W = W - (H.I * S)
        re = l
        count += 1
        # print(W, re, count)
    print('logistic_regression_Newton_methon_without_lambda:\n', W, r
e, count)
    return W, count, re


def logistic_regression_Gradient_Descent_with_lambda(data_mat, result
, w = WN, Lambda=LAMBDA_G):
    """
    logistic_regression_Gradient_Descent_with_lambda 正则化梯度下降实现
```

```
逻辑斯蒂回归

    [description]

    Args:
        data_mat (np.mat): 样本点矩阵 每一行为一个样本的数据
        result (np.mat): 列矩阵 每个样本点的类别 只有01
        w (int, optional): Defaults to WN. 参数维度 两项特征则使用三个参数
 类推
        Lambda (double, optional): Defaults to LAMBDA_G. Lambda正则化
惩罚参数

    Returns:
        np.mat, int, np.mat: 分别是参数列矩阵，迭代次数，loss
    """
    re = 10
    count = 0
    size_y = np.size(data_mat, 0)
    y0 = np.ones((size_y, 1))
    X = np.ones((size_y, 1))
    X = np.hstack((X, data_mat))
    y = result
    W = np.zeros((w, 1))

    while abs(re) > EPSILON and count < LOOP_MAX:
        re = 0
        C = X * W
        Y_W = 1.0 / (1 + np.exp(-C))
        y_1 = y.T * np.log2(Y_W)
        y_2 = (y0 - y).T * np.log2(y0 - Y_W)
        lam = Lambda / (2 * w) * np.sum(np.power(W, 2))
        l = y_1 + y_2 + lam
        W = W - (ALPHA / w) * (X.T * (Y_W - y) + Lambda * W)
        re = l
        count += 1
        # print(W, re, count)
    print('logistic_regression_Gradient_Descent_with_lambda:\n', W, r
e, count)
    return W, count, re


def logistic_regression_Newton_methon_with_lambda(data_mat, result, w
 = WN, Lambda=LAMBDA_N):
    """
    logistic_regression_Newton_methon_with_lambda 正则化牛顿法实现逻辑斯
蒂回归

    [description]

        Args:
        data_mat (np.mat): 样本点矩阵 每一行为一个样本的数据
```

```
        result (np.mat): 列矩阵 每个样本点的类别 只有01
        w (int, optional): Defaults to WN. 参数维度 两项特征则使用三个参数
 类推
        Lambda (double, optional): Defaults to LAMBDA_N. Lambda正则化
惩罚参数

    Returns:
        np.mat, int, np.mat: 分别是参数列矩阵，迭代次数，loss
    """
    re = 10
    count = 0
    size_y = np.size(data_mat, 0)
    y0 = np.ones((size_y, 1))
    X = np.ones((size_y, 1))
    X = np.hstack((X, data_mat))
    y = result
    W = np.zeros((w, 1))
    A = np.eye(w)
    A[0][0] = 0

    while abs(re) > EPSILON and count < LOOP_MAX:
        re = 0
        C = X * W
        Y_W = 1.0 / (1 + np.exp(-C))
        y_1 = y.T * np.log2(Y_W)
        y_2 = (y0 - y).T * np.log2(y0 - Y_W)
        H = X.T * np.mat(np.diag(Y_W.T.tolist()[0])) * np.mat(np.diag
((y0 - Y_W).T.tolist()[0])) * X + Lambda / w * A
        S = X.T * (Y_W - y) - Lambda / w * W
        l = y_1 + y_2
        W = W - (H.I * S)
        re = l
        count += 1
        # print(W, re, count)
    print('logistic_regression_Newton_methon_with_lambda:\n', W, re,
count)
    return W, count, re


def draw(ax, name, list_x, list_y, W, data_num = DATA_NUM, line_a = A
, line_b = B, data_head_x = DATA_HEAD_X, data_end_x = DATA_END_X):

    for i in range(0, data_num):
        if A * list_x[i] + B > list_y[i]:
            ax.scatter(list_x[i], list_y[i], label='1', s=10, color='
red', marker='.', alpha=0.7)
        else:
            ax.scatter(list_x[i], list_y[i], label='2', s=10, color='
blue', marker='x', alpha=0.7)
    line_x = np.linspace(data_head_x, data_end_x, 300)
    line_y = line_a * line_x + line_b
```

```
    ax.plot(line_x, line_y, label='line', color='black', linewidth=0.
8, alpha=1)
    ax.set_title(name)
#     plt.legend()


def main():
    fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, figsize=
(8, 6))
    plt.xlim(DATA_HEAD_X, DATA_END_X)
    plt.ylim(DATA_HEAD_Y, DATA_END_Y)
    ax1 = axes[0,0]
    ax2 = axes[0,1]
    ax3 = axes[1,0]
    ax4 = axes[1,1]

    list_x, list_y, data_mat, result = original_data()
    W1, count1, re1 = logistic_regression_Gradient_Descent_without_la
mbda(data_mat, result)
    W2, count2, re2 = logistic_regression_Newton_methon_without_lambd
a(data_mat, result)
    W3, count3, re3 = logistic_regression_Gradient_Descent_with_lambd
a(data_mat, result)
    W4, count4, re4 = logistic_regression_Newton_methon_with_lambda(d
ata_mat, result)
    draw(ax1, "logistic_regression_Gradient_Descent_without_lambda",
list_x, list_y, W1, line_a=-W1.tolist()[1][0]/W1.tolist()[2][0], line
_b=-W1.tolist()[0][0]/W1.tolist()[2][0])
    draw(ax2, "logistic_regression_Newton_methon_without_lambda", lis
t_x, list_y, W2, line_a=-W2.tolist()[1][0]/W2.tolist()[2][0], line_b=
-W2.tolist()[0][0]/W2.tolist()[2][0])
    draw(ax3, "logistic_regression_Newton_methon_with_lambda", list_x
, list_y, W3, line_a=-W3.tolist()[1][0]/W3.tolist()[2][0], line_b=-W3
.tolist()[0][0]/W3.tolist()[2][0])
    draw(ax4, "logistic_regression_Newton_methon_with_lambda", list_x
, list_y, W4, line_a=-W4.tolist()[1][0]/W4.tolist()[2][0], line_b=-W4
.tolist()[0][0]/W4.tolist()[2][0])
    print(-W1.tolist()[1][0]/W1.tolist()[2][0], -W1.tolist()[0][0]/W1
.tolist()[2][0], re1, count1)
    print(-W2.tolist()[1][0]/W2.tolist()[2][0], -W2.tolist()[0][0]/W2
.tolist()[2][0], re2, count2)
    print(-W3.tolist()[1][0]/W3.tolist()[2][0], -W3.tolist()[0][0]/W3
.tolist()[2][0], re3, count3)
    print(-W4.tolist()[1][0]/W4.tolist()[2][0], -W4.tolist()[0][0]/W4
.tolist()[2][0], re4, count4)
    plt.show()


if __name__ == '__main__':
    main()
```