山东大学 计算机科学与技术 学院

机器学习（双语） 课程实验报告

学号：	姓名：	班级：
实验题目：Experiment 3: Regularization		
实验学时：4	实验日期： 2022/10/12	

实验目的：

1. 实现实验指导书中正则化的相关内容；
2. 学习使用 MATLAB、Python 等工具进行实验；
3. 根据实验中设置不同正则化系数 lambda 所得到的实验结果，理解体会正则化的作用。

硬件环境：

Inter（R） Core（TM） i7-8750H

RAM：16.0 GB

软件环境：

Visual Studio Code

版本: 1.67.2 (user setup)

OS: Windows_NT x64 10.0.19044

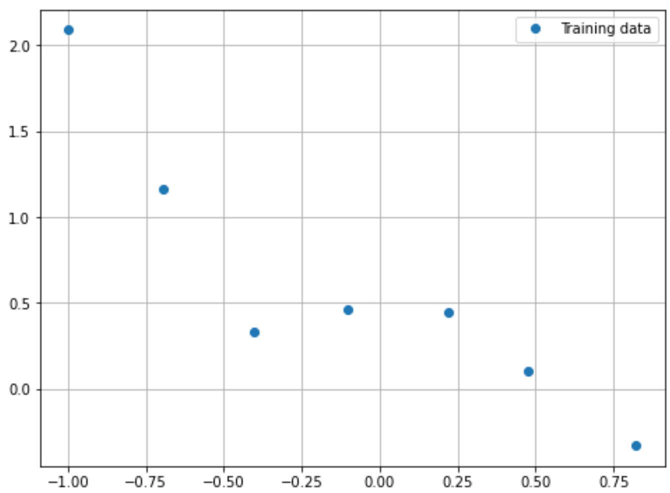
Python 3.9.7

    numpy 1.20.3

    matplotlib 3.4.3

实验步骤与内容：

1. 数据集表示：



本次实验的数据集如图所示，仅有 7 个数据，需要根据这些数据拟合出预测曲线。

2. 为显示正则化项的作用，需要生成高阶多项式来捕捉点的更多特征：

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$$

实现代码：

```
def gen_x(num, X):
    m = X.shape[0]
    x0 = np.hstack((np.ones((m, 1)), X))
    for i in range(2, num+1):
        x0 = np.hstack((x0, X**i))
    return x0
```

3. 代价函数：

$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2]$$

相比较一般的线性回归，该函数在后方多添加了一个正则化项：

```
def J(theta, l, X, Y):
    term1 = np.dot((h(theta, X)-Y).T, h(theta, X)-Y)
    term2 = l*np.sum(theta[1:])
    return (1./2*m)*(term1 + term2)
```

4. 由正则方程：

$$\theta = (X^TX + \lambda\begin{bmatrix}0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1\end{bmatrix})^{-1}X^T\vec{y}$$

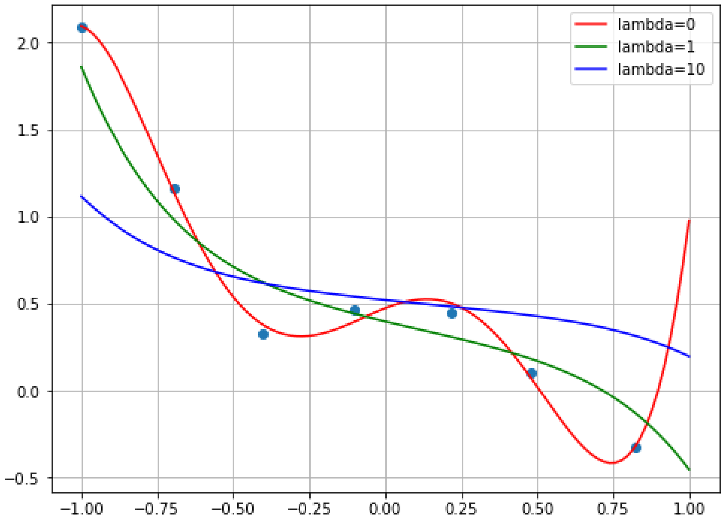
可直接得到 theta：

```
def gen_theta(l, X, Y):
    temp = np.eye(6) * l
    temp[0, 0] = 0
    return np.linalg.inv(X.T.dot(X) + temp).dot(X.T).dot(Y)
```

5. 分别设置正则化系数为 0，1，10，得到对应的 theta 值：

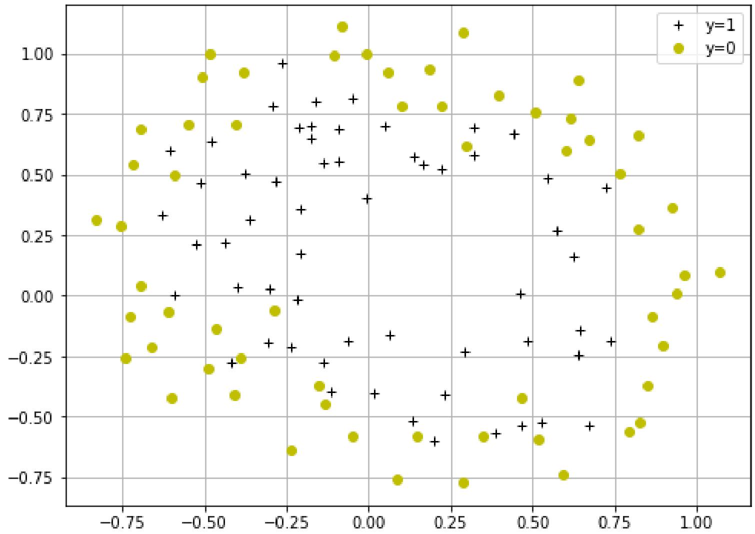
```
theta_0 = gen_theta(0, x, y)
theta_1 = gen_theta(1, x, y)
theta_2 = gen_theta(10, x, y)
```

6. 根据上方得到的 3 组 theta，拟合出对应的预测曲线：



可以看到，当 lambda 为 0，相当于不添加正则化项时，预测结果趋于过拟合，而当 lambda 为 10 时，预测结果趋于欠拟合，当 lambda 为 1 时，预测效果较好。

7. 在第二部分，需要实现逻辑回归的正则化：
8. 首先可视化数据：



9. 与线性回归类似，需要生成高阶数据来捕获高阶信息：

$$x = \begin{bmatrix} 1 \\ u \\ v \\ u^2 \\ uv \\ v^2 \\ u^3 \\ \vdots \\ uv^5 \\ v^6 \end{bmatrix}$$

```
def gen_x2(num, X):
    x0 = np.hstack((np.ones((X.shape[0], 1)), X))
    for i in range(2, num+1):
        for j in range(0, i+1):
            x0 = np.hstack((x0, ((X[:, 0]**(i-j))*(X[:, 1]**j)).reshape(-1, 1)))
    return x0
```

10. 正则化后的逻辑回归代价函数：

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x^{(i)})) + (1-y^{(i)})log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

梯度矩阵：

$$\nabla_\theta J = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)} + \frac{\lambda}{m}\theta_1 \\ \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)} + \frac{\lambda}{m}\theta_2 \\ \vdots \\ \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_n^{(i)} + \frac{\lambda}{m}\theta_n \end{bmatrix}$$

实现：

```
def func(l, theta, x, y):
    temp = []
    temp.append(((h2(theta, x2)-y) * x2[:, 0].reshape(-1, 1)).sum())
    for i in range(1, x.shape[1]):
        temp.append(((h2(theta, x2)-y) * x2[:, i].reshape(-1, 1)).sum() + l*theta[i][0])
    return (1./x.shape[0]) * np.array(temp).reshape(-1, 1)
```

海森矩阵：

$$H = \frac{1}{m}[\sum_{i=1}^{m} h_\theta(x^{(i)})(1 - h_\theta(x^{(i)}))x^{(i)}(x^{(i)})^T] + \frac{\lambda}{m}\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

实现：

```
def H(l, theta, x):
    term1 = (x.T).dot(np.diag(h2(theta, x).reshape(-1))).dot(np.diag(1-h2(theta, x).reshape(-1))).dot(x)
    term2 = np.eye(x.shape[1]) * l
    term2[0, 0] = 0
    return (1./x.shape[0]) * (term1 + term2)
```

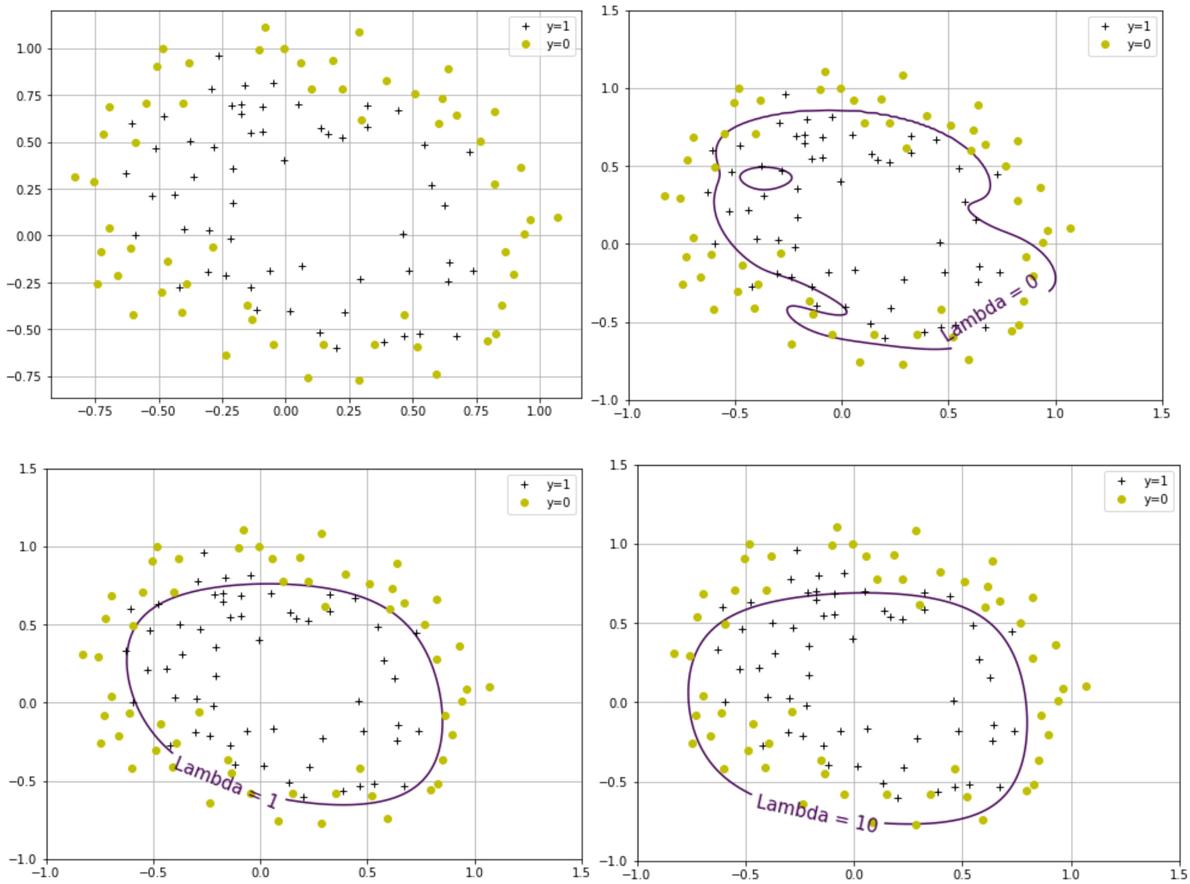
11. 使用牛顿法得到 theta：

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla_\theta J$$

```
def newton_method(l, theta, X, Y):
    theta_record = []   # 记录theta
    L_record = []       # 记录损失
    temp = L(theta, X, Y)
    iterations = 0
    while True:
        theta_record.append(theta.tolist())
        # print(theta_record)
        theta = theta - np.linalg.inv(H(l, theta, X)).dot(func(l, theta, X, Y))
        cost = L(theta, X, Y)
        # print(cost)
        L_record.append(cost.tolist())
        iterations += 1
        if abs(temp - cost) < 1e-9:
            break
        temp = cost
    theta_record = np.array(theta_record).reshape(-1, len(theta))
    L_record = np.array(L_record).reshape(-1)
    return theta, theta_record, L_record, iterations
```

12. 同样，绘制出 lambda 为 0，1，10 时的拟合边界：

13.



可见，当 lambda 为 0 时，过拟合严重，当 lambda 为 10 时，欠拟合，预测效果均较差。而当 lambda 为 1 时，拟合效果较好。

14. 计算 lambda 为 0，1，10 时所得到的三组 theta 的 2 范数，如图所示：

```
norm_1 = np.linalg.norm(theta_1)
norm_2 = np.linalg.norm(theta_2)
norm_3 = np.linalg.norm(theta_3)
```

```
print(norm_1, norm_2, norm_3)
```

```
7172.694617802267 4.240009281990332 0.9384184573785728
```

可见，当正则化系数 lambda 较小时，所得 theta 的 2 范数较大，而 lambda 较大时，所得 theta 的 2 范数较小，分别对应过拟合到欠拟合的过程，说明我们通过设置正则项限制高阶项的系数的目的成功达到。

结论分析与体会：

1. 在实验前，需要充分理解使用 matlab、python 等工具，才能更好地进行实验，实现实验中的各个步骤。
2. 在实验中，需要理解掌握课上所学知识，结合实验指导书，才能更好地完成实验；
3. 正则化在线性回归及逻辑回归中有着重要作用，可通过控制正则化系数来控制拟合程度，从而获得较好的预测效果。

附录：程序源代码

```python
# %%
import numpy as np
import matplotlib.pyplot as plt

# %% [markdown]
# ### 3 Regularized Linear Regression

# %%
x = np.loadtxt('data3/ex3Linx.dat')  # 载入数据
y = np.loadtxt('data3/ex3Liny.dat')

# %%
# 依据数据画出散点图
plt.figure(figsize=(8, 6))
plt.plot(x, y, 'o', label='Training data')
plt.grid()
plt.legend()
plt.show()

# %%
# 整理数据
m = y.shape[0]
x = x.reshape(-1, 1)
y = y.reshape(-1, 1)

# %%
def gen_x(num, X):
    m = X.shape[0]
    x0 = np.hstack((np.ones((m, 1)), X))
    for i in range(2, num+1):
        x0 = np.hstack((x0, X**i))
    return x0

# %%
x = gen_x(5, x)

# %% [markdown]
# 假设函数:
# $$
# h_\theta(x)=\theta_0+\theta_1x+\theta_2x^2+\theta_3x^3+\theta_4x^4+\theta_5x^5
```

```python
# $$

# %%
def h(theta, X):
    return np.dot(X, theta)

# %%
theta = np.zeros((6, 1))

# %% [markdown]
# 代价函数:
# $$
# J(\theta)=\frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2+\lambda\sum_{j=1}^{n}\theta_j^2]
# $$

# %%
def J(theta, l, X, Y):
    term1 = np.dot((h(theta, X)-Y).T, h(theta, X)-Y)
    term2 = l*np.sum(theta[1:])
    return (1./2*m)*(term1 + term2)

# %% [markdown]
# Normal Equation:
# $$
# \theta=(X^TX+\lambda
# \begin{bmatrix}
# 0 \\
# & 1 \\
# & \ & \ddots \\
# & \ & \ & 1
# \end{bmatrix}
# )^{-1}X^T\vec{y}
# $$

# %%
def gen_theta(l, X, Y):
    temp = np.eye(6) * l
    temp[0, 0] = 0
    return np.linalg.inv(X.T.dot(X) + temp).dot(X.T).dot(Y)

# %%
theta_0 = gen_theta(0, x, y)
theta_1 = gen_theta(1, x, y)
```

```python
theta_2 = gen_theta(10, x, y)

# %%
x_space = np.linspace(-1, 1, 100)
x_space = gen_x(5, x_space.reshape(-1, 1))

# %%
plt.figure(figsize=(8, 6))
plt.plot(x[:, 1], y, 'o')
plt.plot(x_space[:, 1], h(theta_0, x_space), 'r', label='lambda=0')
plt.plot(x_space[:, 1], h(theta_1, x_space), 'g', label='lambda=1')
plt.plot(x_space[:, 1], h(theta_2, x_space), 'b', label='lambda=10')
plt.grid()
plt.legend()
plt.show()

# %% [markdown]
# ### 4 Regularized Logistic Regression

# %%
x = np.loadtxt('data3/ex3Logx.dat', delimiter=',')  # 载入数据
y = np.loadtxt('data3/ex3Logy.dat', delimiter=',')

# %%
# 整理数据
m = y.shape[0]
y = y.reshape(-1, 1)

# %%
pos = [i for i in range(y.shape[0]) if y[i] == 1]
neg = [i for i in range(y.shape[0]) if y[i] == 0]

# %%
plt.figure(figsize=(8, 6))
plt.plot(x[pos, 0], x[pos, 1], 'k+', label='y=1')
plt.plot(x[neg, 0], x[neg, 1], 'yo', label='y=0')
plt.grid()
plt.legend()
plt.show()

# %% [markdown]
# $$
# x=
# \begin{bmatrix}
```

```
# 1 \\
# u \\
# v \\
# u^2 \\
# uv \\
# v^2 \\
# u^3 \\
# \vdots \\
# uv^5 \\
# v^6
# \end{bmatrix}
# $$

# %%
def gen_x2(num, X):
    x0 = np.hstack((np.ones((X.shape[0], 1)), X))
    for i in range(2, num+1):
        for j in range(0, i+1):
            x0 = np.hstack((x0, ((X[:, 0]**(i-j))*(X[:, 1]**j)).reshape(-
1, 1)))
    return x0

# %%
x2 = gen_x2(6, x)

# %%
def sigmoid(z):
    return 1. / (1. + np.exp(-z))

# %%
def h2(theta, x):
    return sigmoid(np.dot(x, theta))

# %%
def L(theta, x, y):
    return (1./x.shape[0]) * (np.dot(-y.T, np.log(h2(theta, x))) -
np.dot((1-y.T), np.log(1-h2(theta, x))))

# %% [markdown]
# Regularized Logistic Regression:
# $$
# J(\theta)=-\frac{1}{m}\sum^{m}_{i=1}[y^{(i)}log(h_\theta(x^{(i)}))+(1-
y^{(i)})log(1-
h_\theta(x^{(i)}))]+\frac{\lambda}{2m}\sum^{n}_{j=1}\theta_j^2
```

```
# $$
#
# Newton's Method:
# $$
# \theta^{(t+1)}=\theta^{(t)}-H^{-1}\nabla_\theta J
# $$
#
# The gradient $\nabla_\theta (J)$:
# $$
# \nabla_\theta J =
# \begin{bmatrix}
# \frac{1}{m}\sum^{m}_{i=1}(h_\theta(x^{(i)})-y^{(i)})x_0^{(i)} \\
# \frac{1}{m}\sum^{m}_{i=1}(h_\theta(x^{(i)})-
y^{(i)})x_1^{(i)}+\frac{\lambda}{m}\theta_1 \\
# \frac{1}{m}\sum^{m}_{i=1}(h_\theta(x^{(i)})-
y^{(i)})x_2^{(i)}+\frac{\lambda}{m}\theta_2 \\
# \vdots \\
# \frac{1}{m}\sum^{m}_{i=1}(h_\theta(x^{(i)})-
y^{(i)})x_n^{(i)}+\frac{\lambda}{m}\theta_n \\
# \end{bmatrix}
# $$
#
# Hessian:
# $$
# H=\frac{1}{m}[\sum^{m}_{i=1}h_\theta(x^{(i)})(1-
h_\theta(x^{(i)}))x^{(i)}(x^{(i)})^T]+\frac{\lambda}{m}
# \begin{bmatrix}
# 0 \\
# & 1 \\
# & \ & \ddots \\
# & \ & \ & 1
# \end{bmatrix}
# $$

# %%
def func(l, theta, x, y):
    temp = []
    temp.append(((h2(theta, x2)-y) * x2[:, 0].reshape(-1, 1)).sum())
    for i in range(1, x.shape[1]):
        temp.append(((h2(theta, x2)-y) * x2[:, i].reshape(-1, 1)).sum() +
l*theta[i][0])
    return (1./x.shape[0]) * np.array(temp).reshape(-1, 1)

# %%
```

```python
def H(l, theta, x):
    term1 = (x.T).dot(np.diag(h2(theta, x).reshape(-1))).dot(np.diag(1-
h2(theta, x).reshape(-1))).dot(x)
    term2 = np.eye(x.shape[1]) * l
    term2[0, 0] = 0
    return (1./x.shape[0]) * (term1 + term2)

# %%
def newton_method(L, theta, X, Y):
    theta_record = []   # 记录theta
    L_record = []       # 记录损失
    temp = L(theta, X, Y)
    iterations = 0
    while True:
        theta_record.append(theta.tolist())
        # print(theta_record)
        theta = theta - np.linalg.inv(H(l, theta, X)).dot(func(l, theta,
X, Y))
        cost = L(theta, X, Y)
        # print(cost)
        L_record.append(cost.tolist())
        iterations += 1
        if abs(temp - cost) < 1e-9:
            break
        temp = cost
    theta_record = np.array(theta_record).reshape(-1, len(theta))
    L_record = np.array(L_record).reshape(-1)
    return theta, theta_record, L_record, iterations

# %%
theta = np.zeros((x2.shape[1], 1))
theta_1, theta_record_1, L_record_1, iterations_1 = newton_method(0,
theta, x2, y)
theta_2, theta_record_2, L_record_2, iterations_2 = newton_method(1,
theta, x2, y)
theta_3, theta_record_3, L_record_3, iterations_3 = newton_method(10,
theta, x2, y)

# %%
def plotData(l, theta):
    u = np.linspace(-1, 1.5, 200)
    v = np.linspace(-1, 1.5, 200)
    z = np.zeros((len(u), len(v)))
```

```python
    for i in range(len(u)):
        for j in range(len(v)):
            z[i][j] = h2(theta, gen_x2(6, np.array((u[i],
v[j])).reshape(1, -1)))

    plt.figure(figsize=(8, 6))
    plt.plot(x[pos, 0], x[pos, 1], 'k+', label='y=1')
    plt.plot(x[neg, 0], x[neg, 1], 'yo', label='y=0')
    contour = plt.contour(u, v, z, [0.5])
    plt.clabel(contour, inline=1, fontsize=15, fmt='Lambda =
{0}'.format(l))
    plt.grid()
    plt.legend()
    plt.show()

# %%
plotData(0, theta_1)
plotData(1, theta_2)
plotData(10, theta_3)
```