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 2: Logistic Regression and Newton's Method		
实验学时：4	实验日期：　2022/10/5	

实验目的：

1. 实现实验指导书中逻辑回归及牛顿法的相关内容；
2. 学习使用 MATLAB、Python 等工具进行实验；
3. 理解体会逻辑回归、牛顿法等基本概念。

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

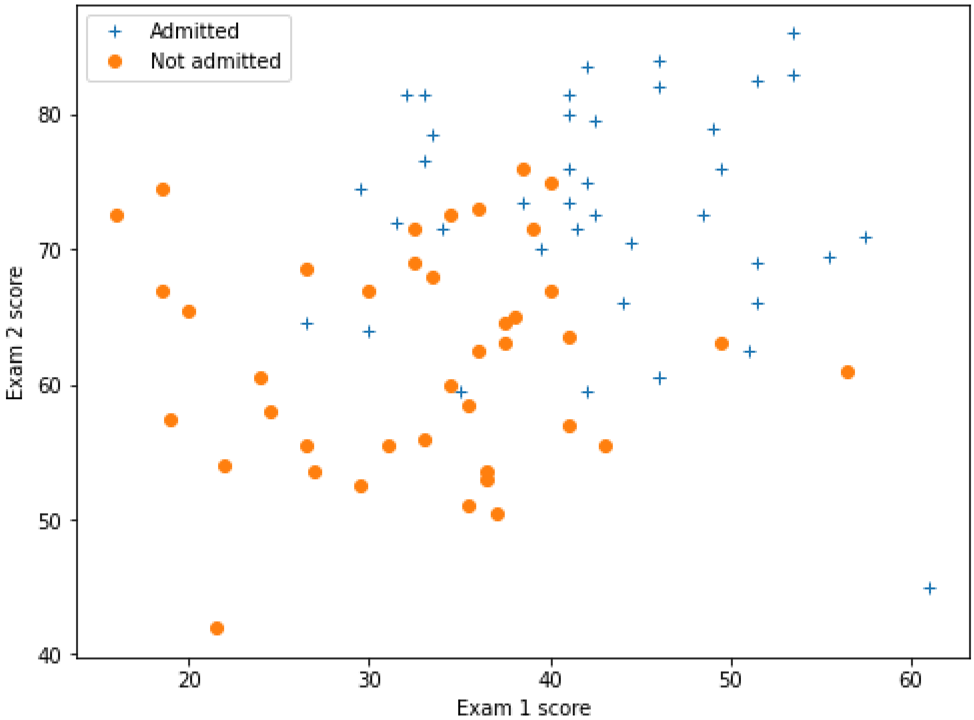
numpy 1.20.3

matplotlib 3.4.3

实验步骤与内容：

1. 数据集表示：

本次实验的数据集为多位学生在两次考试上的得分，来预测其是否被录取：



2. 逻辑回归的假设函数：

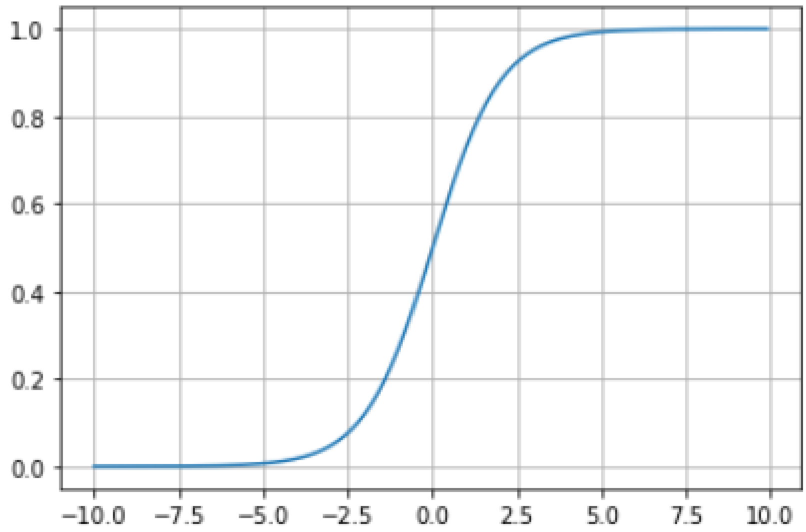
$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} = P(y = 1|x;\theta)$$

当假设函数的值大于等于 0.5 时，预测该学生会被录取，小于 0.5 时，预测该学生不会被录取。

其中 g(·) 为 sigmoid 函数，实现：

```python
def sigmoid(z):
    return 1. / (1. + np.exp(-z))
```

其函数图像：



由此得到假设函数：

```python
def h(theta, x):
    return sigmoid(np.dot(x, theta))
```

3. 本实验的似然函数：

$$J(\theta) = \prod_{i=1}^{m} (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

为便于优化，将其转换为对数似然函数：

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

要最大化上式，即最小化其负值：

$$\min_\theta L(\theta) = \frac{1}{m} \sum_{i=1}^{m} [-y^{(i)} log(h_\theta(x^{(i)})) - (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

该式的梯度：

$$\nabla_\theta L = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

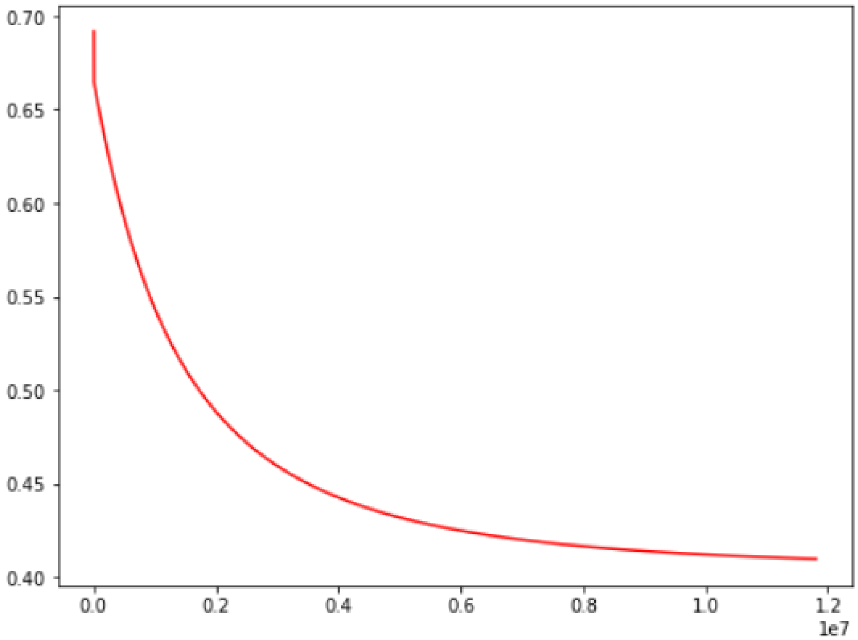
对数似然函数的负值形式的实现：

```
def L(theta, x, y):
    m = y.shape[0]
    return (1./m) * (np.dot(-y.T, np.log(h(theta, x))) - np.dot((1-y.T), np.log(1-h(theta, x))))
```

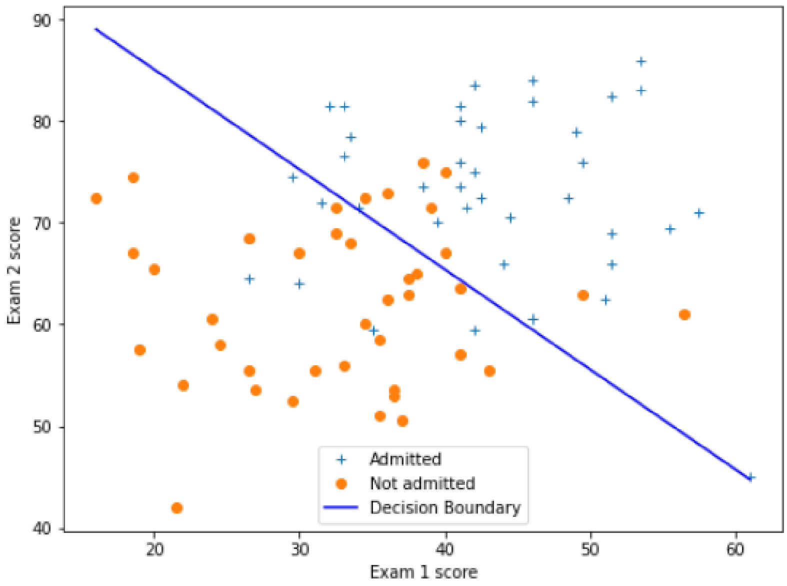
梯度下降：

```
def descendGradient(learning_rate, 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 - (learning_rate/m) * np.dot(X.T, (h(theta, X)-Y))
        cost = L(theta, X, Y)
        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
```

4. 设置学习率为 1e-4，初始 theta 全为 0，当两次对数似然函数的结果的差的绝对值小于 1e-9 时停止训练。
   绘制对数似然函数的下降曲线：



表示出预测边界：



在图中蓝线上方的点会被预测为 admitted，下方的点会被预测为 not admitted。

此处回答这部分的问题：

1）收敛需要的迭代次数：`11804233`；

2）收敛后的 theta：
```
array([[-13.52535717],
       [  0.127088   ],
       [  0.12908398]])
```
；

3）似然函数的变化过程已显示在上文中；

4）预测边界也显示在上文中；

5）对于在两次考试中分别得到 20、80 分的同学进行预测，其未被 admitted 的概率为：`array([0.6585588])`，该概率大于 0.5，预测结果为该同学不会被 admitted。

5. 以下使用牛顿法进行预测：

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

其中：

$$H = \frac{1}{m}\sum_{i=1}^{m}[h_\theta(x^{(i)})(1-h_\theta(x^{(i)}))x^{(i)}(x^{(i)})^T]$$

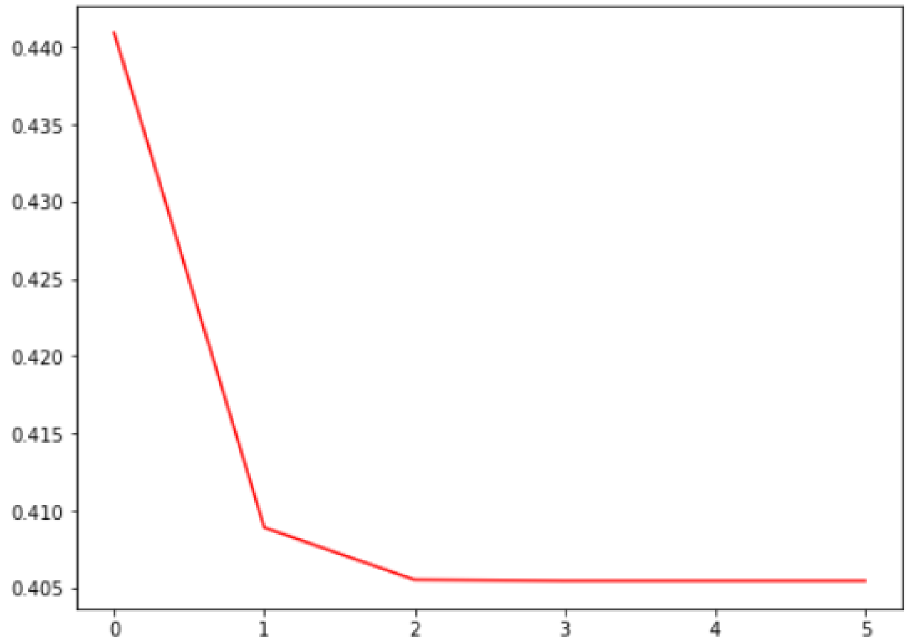
似然函数的实现：

```
def H(theta, x):
    m, n = x.shape
    temp = (x.T).dot(np.diag(h(theta, x).reshape(-1))).dot(np.diag(1-h(theta, x).reshape(-1))).dot(x)
    return (1./m) * temp
```

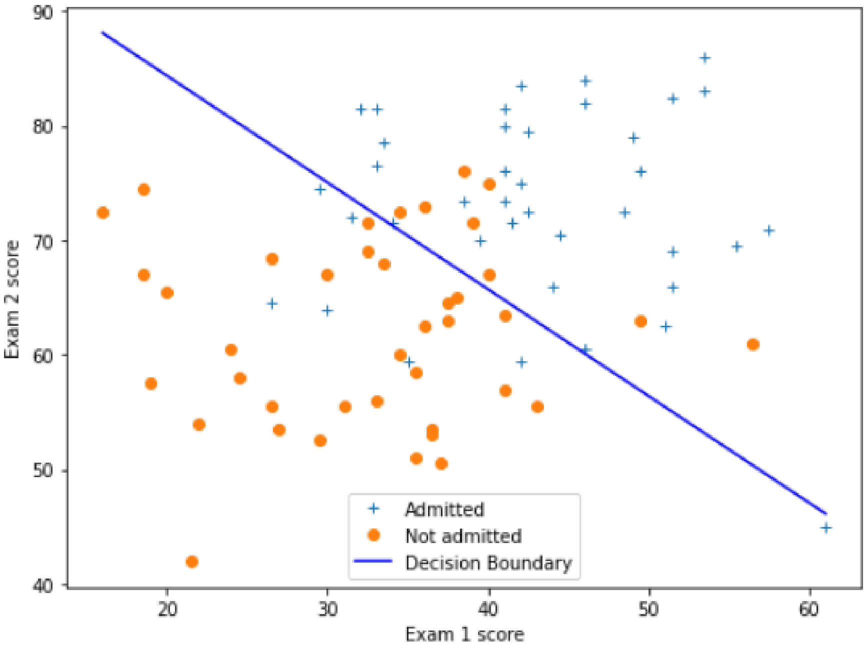
牛顿法的实现：

```python
def newton_method(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(theta, X)).dot((1./m)*(X.T).dot(h(theta, X)-Y))
        cost = L(theta, X, Y)
        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
```

同样设置 theta 全为 0，两次似然函数的差的绝对值小于 1e-9 时结束迭代，得到似然函数下降图像：



6. 绘制出预测边界：



同样，位于预测边界上方的点预测为 admitted，下方的预测为 not admitted。

此处回答牛顿法部分的问题：

1）达到收敛时 theta 的值：

```
array([[-16.37874341],
       [  0.14834077],
       [  0.15890845]])
```

2）L 的下降过程已显示在上文中；

3）决策边界也已显示在上文中；

4）对于在两次考试中分别得到 20、80 分的同学进行预测，其未被 admitted 的概率为：

```
array([0.66802186])
```

，该概率大于 0.5，故预测该同学不会被 admitted。

5）两种方法相比较，牛顿法的下降速度更快，不过相对来说计算量更大，因而在参数较少时可通过选用牛顿法来更快地得到最优结果，而在参数较多时应使用梯度下降法进行训练。

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. 逻辑回归、牛顿法，是机器学习中的基本概念，需要认真做好实验，理解体会其内容，为将来的学习打下基础。

附录：程序源代码

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

```python
# %%
x = np.loadtxt('data2/ex2x.dat')  # 载入数据
y = np.loadtxt('data2/ex2y.dat')

# %%
# 整理数据
m = y.shape[0]
x = np.hstack((np.ones((m, 1)), x.reshape(-1, 2)))
y = y.reshape(-1, 1)

# %% [markdown]
# 3. Plot the Data

# %%
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, 1], x[pos, 2], '+', label='Admitted')
plt.plot(x[neg, 1], x[neg, 2], 'o', label='Not admitted')
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend()
plt.show()

# %% [markdown]
# 4. Logistic Regression

# %% [markdown]
# 假设函数:
# $$
# h_\theta(x)=g(\theta^Tx)=\frac{1}{1+e^{-\theta^Tx}}=P(y=1|x;\theta)
# $$

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

# %%
temp = np.arange(-10, 10, 0.1)
plt.plot(temp, sigmoid(temp))
plt.grid()
```

```python
plt.show()

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

# %% [markdown]
# 似然函数:
# $$
# J(\theta)=\prod_{i=1}^{m}(h_\theta(x^{(i)}))^{y^{(i)}}(1-h_\theta(x^{(i)}))^{1-y^{(i)}}
# $$
# 对数似然函数:
# $$
# L(\theta)=\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x^{(i)}))+(1-y^{(i)})log(1-h_\theta(x^{(i)}))]
# $$
# 要最大化上式，即最小化其负值:
# $$
# \mathop{min}\limits_{\theta}L(\theta)=\frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}log(h_\theta(x^{(i)}))-(1-y^{(i)})log(1-h_\theta(x^{(i)}))]
# $$
# 该式的梯度:
# $$
# \nabla_\theta L=\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x^{(i)}
# $$

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

# %%
def descendGradient(learning_rate, 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 - (learning_rate/m) * np.dot(X.T, (h(theta, X)-Y))
```

```
        cost = L(theta, X, Y)
        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

# %%
learning_rate = 0.0001
theta = np.zeros((x.shape[1], 1))

# %%
theta_0, theta_record_0, L_record_0, iterations_0 =
descendGradient(learning_rate, theta, x, y)

# %%
iterations_0

# %%
theta_0

# %%
plt.figure(figsize=(8, 6))
plt.plot(range(0, len(L_record_0)), L_record_0, 'r')
plt.show()

# %%
boundary_xs = np.array([np.min(x[:, 1]), np.max(x[:, 1])])
boundary_ys = (-1./theta_0[2]) * (theta_0[0] + theta_0[1] * boundary_xs)

plt.figure(figsize=(8, 6))
plt.plot(x[pos, 1], x[pos, 2], '+', label='Admitted')
plt.plot(x[neg, 1], x[neg, 2], 'o', label='Not admitted')
plt.plot(boundary_xs, boundary_ys, 'b-', label='Decision Boundary')
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend()
plt.show()

# %%
1 - h(theta_0, np.array([1, 20, 80]))
```

```python
# %% [markdown]
# ### Newton's Method

# %% [markdown]
# $$
# \theta^{(t+1)}=\theta^{(t)}-H^{-1}\nabla_\theta L
# $$
# In logistic regression, the Hessian is:
# $$
# H = \frac{1}{m}\sum^{m}_{i=1}[h_\theta(x^{(i)})(1-
h_\theta(x^{(i)}))x^{(i)}(x^{(i)})^T]
# $$

# %%
def H(theta, x):
    m, n = x.shape
    temp = (x.T).dot(np.diag(h(theta, x).reshape(-1))).dot(np.diag(1-
h(theta, x).reshape(-1))).dot(x)
    return (1./m) * temp
    # temp = np.zeros((n, n))
    # for i in range(m):
    #     temp += h(theta, x[i]) * (1-h(theta, x[i])) * x[i].reshape(-1,
1).dot(x[i].reshape(1, -1))
    # return (1./m) * temp

# %%
def newton_method(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(theta,
X)).dot((1./m)*(X.T).dot(h(theta, X)-Y))
        cost = L(theta, X, Y)
        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))
```

```python
    L_record = np.array(L_record).reshape(-1)
    return theta, theta_record, L_record, iterations

# %%
theta = np.zeros((x.shape[1], 1))
theta_1, theta_record_1, L_record_1, iterations_1 = newton_method(theta,
x, y)

# %%
theta_1

# %%
plt.figure(figsize=(8, 6))
plt.plot(range(0, len(L_record_1)), L_record_1, 'r')
plt.show()

# %%
boundary_xs = np.array([np.min(x[:, 1]), np.max(x[:, 1])])
boundary_ys = (-1./theta_1[2]) * (theta_1[0] + theta_1[1] * boundary_xs)

plt.figure(figsize=(8, 6))
plt.plot(x[pos, 1], x[pos, 2], '+', label='Admitted')
plt.plot(x[neg, 1], x[neg, 2], 'o', label='Not admitted')
plt.plot(boundary_xs, boundary_ys, 'b-', label='Decision Boundary')
plt.xlabel('Exam 1 score')
plt.ylabel('Exam 2 score')
plt.legend()
plt.show()

# %%
1 - h(theta_1, np.array([1, 20, 80]))

# %%
```