机器学习实验二——逻辑回归 实验报告

组员：张继伟 谢正宇 樊佳婷 孙晶铭 刘怡聪

题目

假设你是某大学招生主管，你想根据两次考试的结果决定每个申请者的录取机会。现有以往申请者的历史数据，可以此作为训练集建立逻辑回归模型，并用其预测某学生能否被大学录取。请按要求完成实验。建议使用 python 编程实现。

数据集

文件 ex2data1.txt 为该实验的数据集，第一列、第二列分别表示申请者两次考试的成绩，第三列表示录取结果（1 表示录取，0 表示不录取）。

步骤与要求

1）请导入数据并进行数据可视化，观察数据分布特征。（建议用 python 的matplotlib）
2）将逻辑回归参数初始化为 0，然后计算代价函数（cost function）并求出初始值。
3）选择一种优化方法求解逻辑回归参数。
4）某学生两次考试成绩分别为 42、85，预测其被录取的概率。
5）画出分类边界。（选做）

实验过程与结果分析

1. **对数据集进行观察，使用matplotlib将数据集绘制出散点图**

   首先对测试集数据进行预处理，将两次成绩与是否通过测试的标记分别生成矩阵，并将标记的矩阵转置。

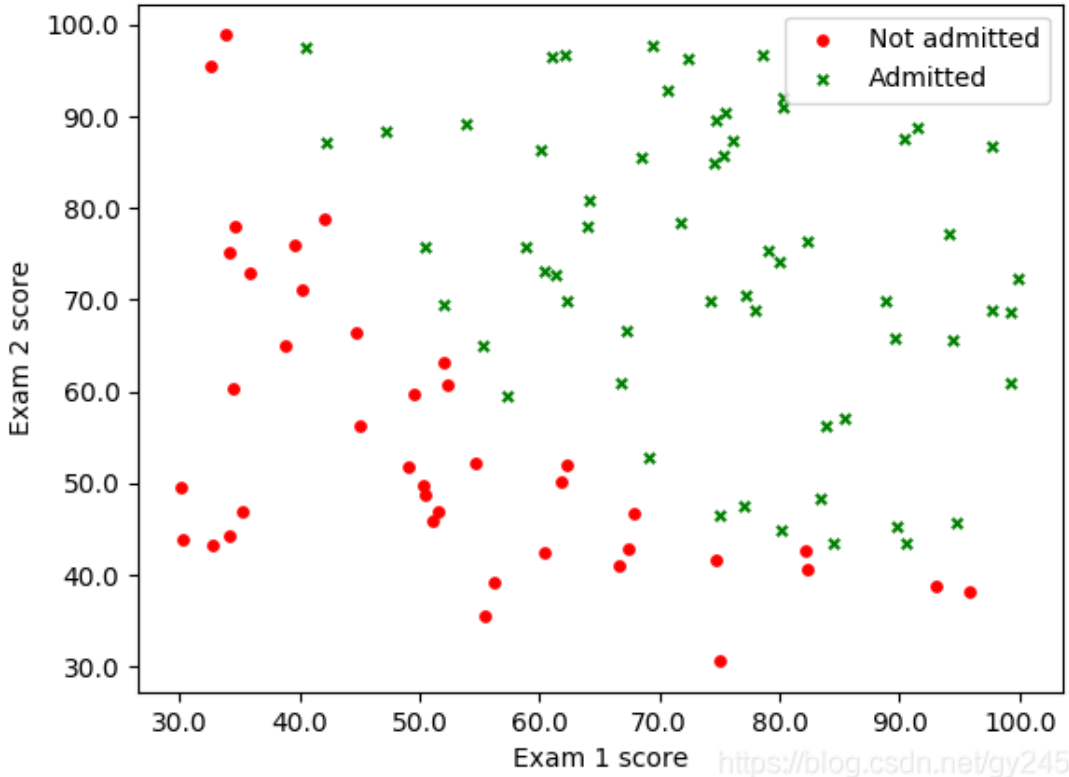
```
def init_data():
    # 两次成绩对应的特征矩阵
    data = []
    # 标记对应的矩阵
    label = []

    # 读取文件
    train_data = open("ex2data1.txt")
    lines = train_data.readlines()
    for line in lines:
        scores = line.split(",")
        # 去除标记后面的换行符
        isQualified = scores[2].replace("\n", "")
        # 添加特征x0，设置为1
        data.append([1, float(scores[0]), float(scores[1])])
        label.append(int(isQualified))

    # 标记矩阵转置，返回特征矩阵和标记矩阵
    return np.array(data), np.array(label).transpose()
```

   使用matplotlib库将数据集绘制成散点图：

```
def draw():
 # 定义x y数据 x1 y1:未通过 x2 y2:通过
 x1 = []
 y1 = []
 x2 = []
 y2 = []
 # 导入训练数据
 train_data = open("ex2data1.txt")
 lines = train_data.readlines()
 for line in lines:
     scores = line.split(",")
     # 去除标记后面的换行符
     isQualified = scores[2].replace("\n", "")
     # 根据标记将两次成绩放到对应的数组
     if isQualified == "0":
         x1.append(float(scores[0]))
         y1.append(float(scores[1]))
     else:
         x2.append(float(scores[0]))
         y2.append(float(scores[1]))
 # 设置标题和横纵坐标的标注
 plt.xlabel("Exam 1 score")
 plt.ylabel("Exam 2 score")
 # 设置通过测试和不通过测试数据的样式。其中x y为两次的成绩，marker:记号形状 color:颜色
s:点的大小 label:标注
 plt.scatter(x1, y1, marker='o', color='red', s=15, label='Not admitted')
 plt.scatter(x2, y2, marker='x', color='green', s=15, label='Admitted')
 # 标注[即上两行中的label]的显示位置：右上角
 plt.legend(loc='upper right')
 # 设置坐标轴上刻度的精度为一位小数。因训练数据中的分数的小数点太多，若不限制坐标轴上刻度显
示的精度，影响最终散点图的美观度
 plt.gca().xaxis.set_major_formatter(ticker.FormatStrFormatter('%.1f'))
 plt.gca().yaxis.set_major_formatter(ticker.FormatStrFormatter('%.1f'))
 # 显示
 plt.show()
```



运行上述代码得到上图，即测试数据集的散点分布图。

2. **观察测试数据集的图像，定义预测函数h(x)，定义代价函数，定义梯度下降函数**

观察散点图，推测对测试数据集进行分类需一条直线，因为有两个输入参数所以设：

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 = \theta^T X$$

其中x0均为1。 因此 sigmoid 函数为

$$g(z) = g(h\theta(x)) = \frac{1}{1 + e^{-h\theta(x)}}$$

编写如下h(x)和sigmoid函数：

```
*# 定义h(x)预测函数:theat为转置后的矩阵*
def **hypothesis**(theta, x):
  return np.dot(x, theta)

*# 定义sigmoid函数*
def **sigmoid**(theta, x):
  z = hypothesis(theta, x)
  return 1.0 / (1 + exp(-z))
```

逻辑回归的代价函数为：

$$J(\theta) = \frac{1}{M}\sum_{i=1}^{m}[-y^{(i)}log(\theta_0 + \theta_1 x_1^i + \theta_2 x_2^i) - (1 - y^{(i)})log(1 - \theta_0 - \theta_1 x_1^i - \theta_2 x_2^i)]$$

本实验中采用矩阵运算，所以需要对代价函数进行向量化，结果为：

$$J(\theta) = \frac{1}{m}[-y^T log(h) - (1 - y)^T log(1 - h)]$$

多个特征向量化，可以加快运算速度，尤其是对于特征量大的数据集效果更明显。代价函数的定义方式决定了下一步是采用梯度下降算法还是梯度上升算法，梯度上升和梯度下降都是求预测值和真实值最接近时theta的值，不同的是梯度上升用于求最大值，梯度下降用于求最小值。这里采用批量梯度下降法，对应的向量化为：

$$\frac{1}{m}X^T(Sigmoid(X\theta)-y)$$

定义如下代码：

```
*# 定义代价函数*
def **cost**(theta, X, y):
  return
np.mean(-y * np.log(sigmoid(theta, X)) - (1 - y) * np.log(1 - sigmoid(theta,
X)))

*# 梯度下降函数*
def **gradient**(theta, X, y):
  return (1 / **len**(X)) * X.T @ (sigmoid(theta, X) - y)
```

3. **求解最优解**

程序main方法定义如下：

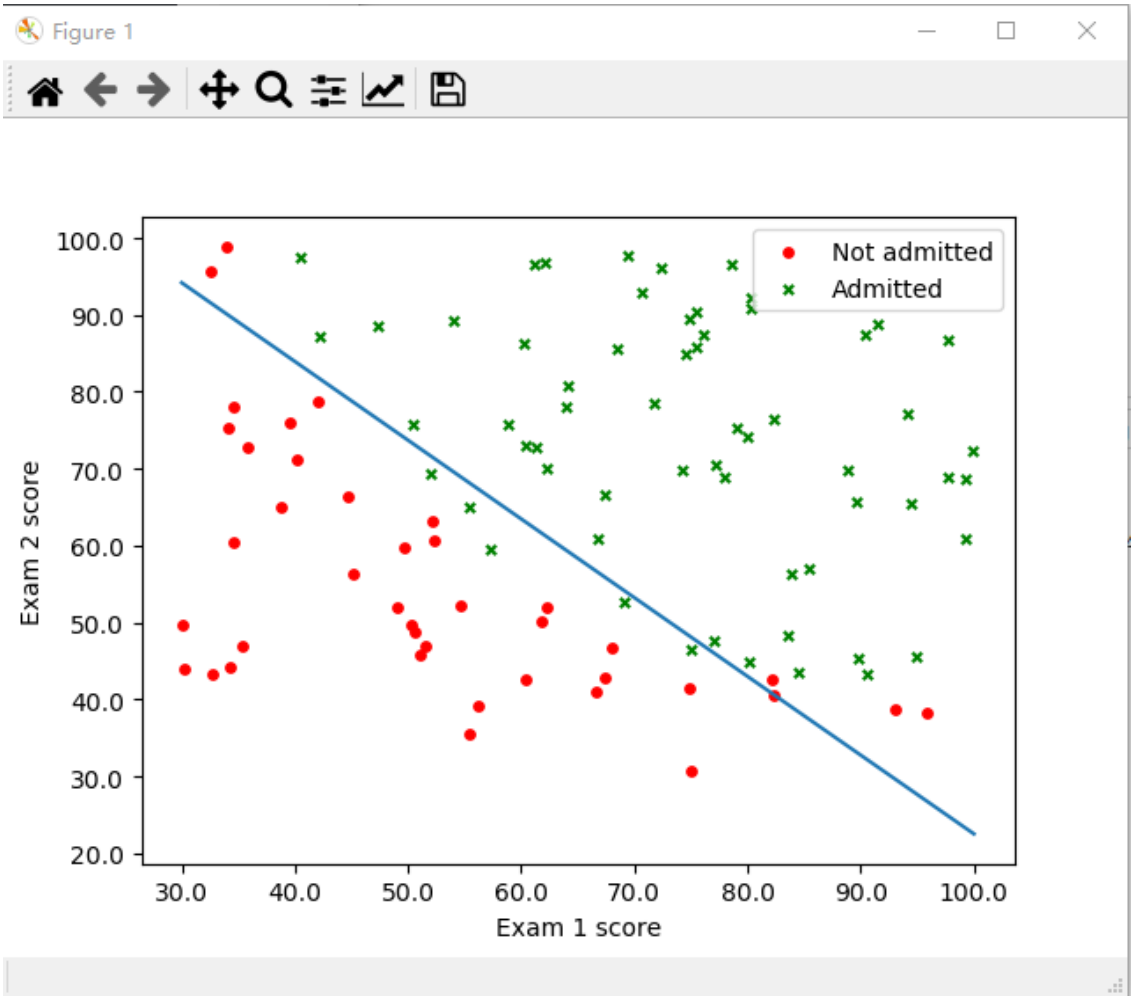
```
if __name__ == '__main__':
  *# 初始化数据*
  X, y = init_data()
  *# 初始化theta:三行一列的0矩阵*
  theta = np.zeros((3, 1))
  *# 使用minimize函数求解*
  result = opt.minimize(fun=cost, x0=theta, args=(X, y), method='Newton-CG',
jac=gradient)
  *# 绘图*
  draw()
```

这里直接使用**scipy.optimize.minimize()** 方法求解代价函数的最优解。其中cost为代价函数，theta为初始值，一般设置为0，args传入的X，y分别是训练数据的特征和标志位，method是指定优化算法。返回的result中x就是最终求得的theta值。

根据minimize方法求得的theta值计算预测函数，绘制预测函数图像，并预测成绩为42，85时的录取结果：

```
*# 设置训练得到的模型对应的直线，即h(x)对应的直线*
  *# 设置x的取值范围：[30, 110]步长为10*
  x = np.arange(30, 110, 10)
  y = (-result.x[0] - result.x[1] * x) / result.x[2]
  predict=(-result.x[0]-result.x[1]*42)/result.x[2]
  print("当第一次考试成绩为42时，最小可被接收的第二次成绩为",predict)
  if predict-85<0:
    print("这个学生可以被接收！")
  else:
    print("这个学生不能被接收！")
  plt.plot(x, y)
```

最终效果图为：



第一次成绩为42时，预测函数的Y值约为81.89，故第二次成绩为85的学生预测结果为被录取：

```
swag@LAPTOP-3C8MIQO9 MINGW64 /d/VSCode/PyCode
$  /usr/bin/env D:\\Anaconda\\python.exe c:\\Users\\swag\\.vscode\\extensions'
Learing1.py
当第一次考试成绩为42时，最小可被接收的第二次成绩为 81.89528963471702
这个学生可以被接收!
```

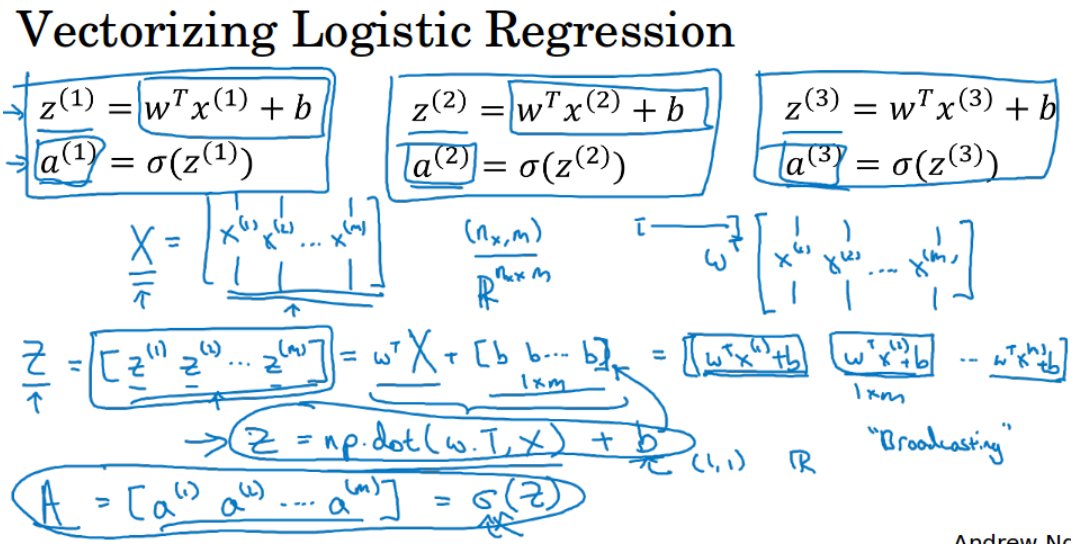
实验收获

1. **在定义预测函数h(x)的代码中，实现时我们的特征矩阵是n+1个，多出来的一个特征就是x0，并且x0都为1，之所以添加一个x0的原因是在代码中进行计算时是以矩阵的形式进行的，因此θ矩阵转置后的行数必须与特征矩阵X的列数一致，如果这两个数不一致则无法进行矩阵运算。**

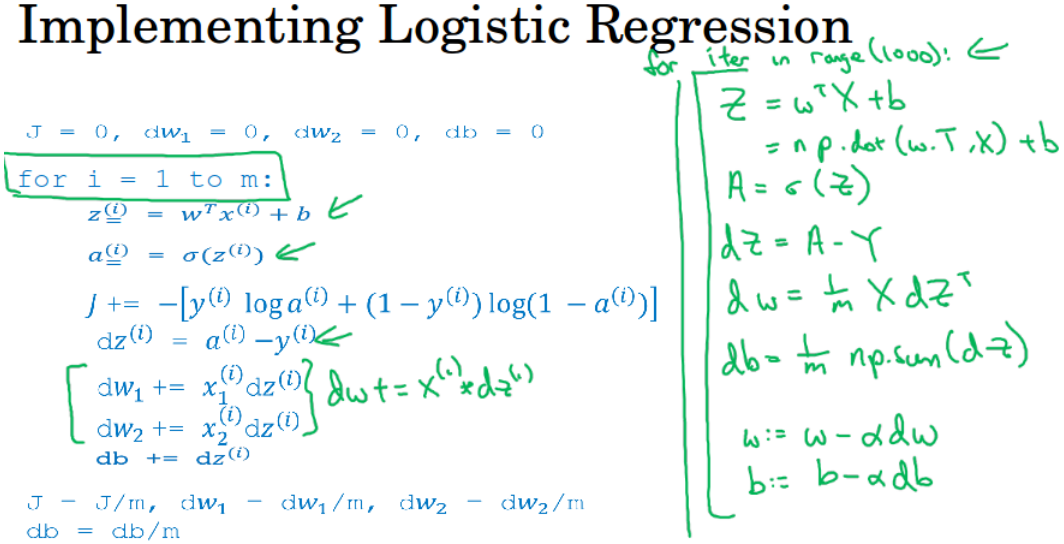
2. **实现代价函数时，可以采用for循环代替矩阵，但使用矩阵运算的速度远高于for循环。**

   在逻辑回归中，以计算z为例，求两个100万长的一维向量的内积，**用向量化花了1.5毫秒，而用for循环计算花了400多毫秒。**CPU和GPU都有并行化的指令，有时候叫SIMD(single instruction multiple data)。如果使用内置函数，比如np.function，python的numpy能充分利用并行化去更快的计算。

   图中给出了向量化的过程。Z的计算的向量化形式是z=np.dot(w.T,x)+bz=np.dot(w.T,x)+b，其中b在这里是一个实数，python在向量和实数相加时，会自动把实数变成一个相同维度的向量再相加。其中w是n * 1的列向量，w.T是1 * n的列向量，X是n * m的矩阵，结果就是1 * m的向量，最后加上1 * m的b向量，得到1 * m的Z。最后通过sigmoid得到预测值A。



同时还可以利用向量化计算m个数据的梯度，注意是同时计算。下图左边是for循环的实现，右边是向量化的实现。这里dz是代价函数对z变量的导数，之前推导过等于预测值减去实际值a - y。dw是代价函数对w的导数，db是代价函数对b的导数。



虽然要尽量使用向量化，但是在进行多次梯度下降的迭代还是要用到for循环，这个不可避免。