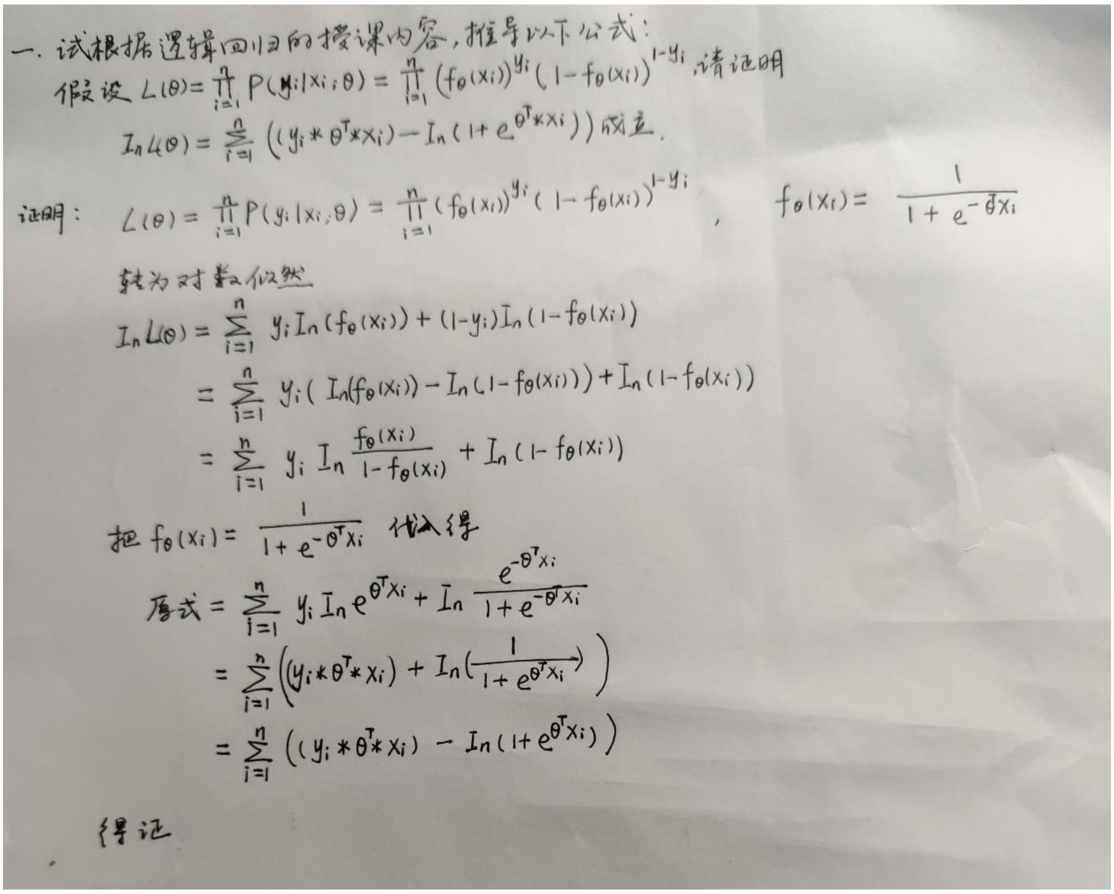
第一章

一、逻辑回归证明



一. 试根据逻辑回归的授课内容，推导以下公式：

假设 $L(\theta)=\prod_{i=1}^{n} P(y_i|x_i;\theta)=\prod_{i=1}^{n}(f_\theta(x_i))^{y_i}(1-f_\theta(x_i))^{1-y_i}$，请证明

$\ln L(\theta)=\sum_{i=1}^{n}((y_i*\theta^T*x_i)-\ln(1+e^{\theta^T*x_i}))$ 成立.

证明： $L(\theta)=\prod_{i=1}^{n} P(y_i|x_i;\theta)=\prod_{i=1}^{n}(f_\theta(x_i))^{y_i}(1-f_\theta(x_i))^{1-y_i}$ ， $f_\theta(x_i)=\frac{1}{1+e^{-\theta x_i}}$

转为对数似然

$$\ln L(\theta)=\sum_{i=1}^{n} y_i\ln(f_\theta(x_i))+(1-y_i)\ln(1-f_\theta(x_i))$$

$$=\sum_{i=1}^{n} y_i(\ln(f_\theta(x_i))-\ln(1-f_\theta(x_i)))+\ln(1-f_\theta(x_i))$$

$$=\sum_{i=1}^{n} y_i\ln\frac{f_\theta(x_i)}{1-f_\theta(x_i)}+\ln(1-f_\theta(x_i))$$

把 $f_\theta(x_i)=\frac{1}{1+e^{-\theta^T x_i}}$ 代入得

$$原式=\sum_{i=1}^{n} y_i\ln e^{\theta^T x_i}+\ln\frac{e^{-\theta^T x_i}}{1+e^{-\theta^T x_i}}$$

$$=\sum_{i=1}^{n}\left((y_i*\theta^T*x_i)+\ln\left(\frac{1}{1+e^{\theta^T x_i}}\right)\right)$$

$$=\sum_{i=1}^{n}\left((y_i*\theta^T*x_i)-\ln(1+e^{\theta^T x_i})\right)$$

得证

二、逻辑回归模型训练分类器

1、代码解析

（1）导包

```python
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np
import scipy.optimize as opt
from sklearn.metrics import classification_report
import pandas as pd
```

（2）读取文档“ex2data2.txt”中的数据

```python
# 读取于文档“ex2data2.txt”中的数据
def read_data(path):
    raw_data = pd.read_csv(path, header=None, names=['x1', 'x2', 'y'])
    return raw_data

```

（3）绘制原始数据散点图

```python
# 绘制原始数据散点图
def draw_scatter(data):
    # 将样本分为正负样本
    positive = data[data['y'].isin([1])]
    negative = data[data['y'].isin([0])]
    # 绘制 x1 和 x2 的散点图
    plt.scatter(positive['x1'], positive['x2'], s=50, c='green', marker='o', label='accepted')
    plt.scatter(negative['x1'], negative['x2'], s=50, c='red', marker='x', label='rejected')
    plt.xlabel('x1')
    plt.ylabel('x2')
    # 注释的显示位置：右上角
    plt.legend(loc='upper right')
    # 设置坐标轴上刻度的精度
    plt.gca().xaxis.set_major_formatter(ticker.FormatStrFormatter('%.1f'))
    plt.gca().yaxis.set_major_formatter(ticker.FormatStrFormatter('%.1f'))
    return plt

```

（4）将 x1 x2 原始一阶特征映射到 6 阶（多项式拟合曲线）

```python

# 特征映射 x1 x2 映射到 power 阶特征
def feature_mapping(x1, x2, power):
    data_map = {}
    for i in range(power+1):
        for j in range(i+1):
            data_map["f{}{}".format(j, i-j)] = np.power(x1, j)*np.power(x2, i-j)
    return pd.DataFrame(data_map)

```

（5）sigmoid 函数

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

```

（6）代价函数，防止过拟合添加惩罚项即正则化代价函数

```python
# 正则化代价函数
def regularized_cost_function(theta, x, y, lam):
    m = x.shape[0]  # m-样本数量
```

```
    # 使用交叉熵损失函数
    j = ((y.dot(np.log(sigmoid(x.dot(theta)))))+((1-y).dot(np.log(1-sigmoid(x.dot(theta))))))/-m
    # L2 正则项
    penalty = lam*(theta.dot(theta))/(2*m)
    return j+penalty

```

（7）梯度函数

```
# 梯度函数
def regularized_gradient_descent(theta, x, y, lam):
    m = x.shape[0]
    # 损失函数对 theta_j 求导
    partial_j = ((sigmoid(x.dot(theta))-y).T).dot(x)/m   # .T 表示转置
    partial_penalty = lam*theta/m
    # 不惩罚第一项
    partial_penalty[0] = 0
    return partial_j+partial_penalty

```

（8）预测函数，已求出 theta，验证分类效果

```
# 预测函数
def predict(theta, x):
    h = x.dot(theta)  # 矩阵相乘
    return [1 if x >= 0.5 else 0 for x in h]

```

（9）根据求出的 theta 绘制决策边界

```
# 绘制决策边界
def draw_boundary(theta, data):
    x = np.linspace(-1, 1.5, 200)
    x1, x2 = np.meshgrid(x, x)

    # 生成高维特征数据
    z = feature_mapping(x1.flatten(), x2.flatten(), 6).values  # flatten()展平
    z = z.dot(theta)
    # 保持维度一致
    z = z.reshape(x1.shape)
    # 绘制散点图
    plt = draw_scatter(data)
    # 绘制高度为 0 的等高线
```

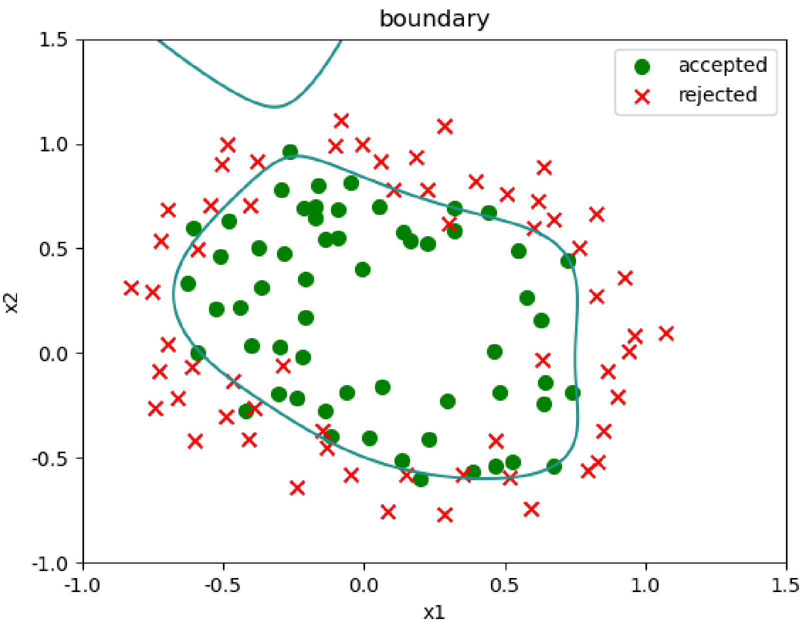
```
    plt.contour(x1, x2, z, 0)
    plt.title('boundary')
    plt.show()

```

（10）主函数调用

```
# 主函数
if __name__ == '__main__':
    # 读取原始数据
    raw_data = read_data('ex2data2.txt')
    # print(raw_data)
    # plt = draw_scatter(raw_data)
    # plt.show()

    # 由散点图可知决策边界非线性，正则化逻辑回归，采用多项式回归，6 阶
    # 构造从原始特征的多项式中得到的特征
    processed_data = feature_mapping(raw_data['x1'], raw_data['x2'], power=6)
    # print(processed_data)
    x = processed_data.values  # 118*28 矩阵
    print(x)
    y = raw_data['y']  # 118*1 label
    # print(y.shape)

    # 初始化 theta 矩阵   规格 28*1  0 填充
    theta = np.zeros(x.shape[1])

    # 设置正则化参数 lambda
    lam = 0.01

    print(regularized_cost_function(theta, x, y, lam))
    # 使用 minimize 函数求解
    theta = opt.minimize(fun=regularized_cost_function, x0=theta, args=(x, y, lam), method='tnc', jac=regularized_gradient_descent).x

    print(regularized_cost_function(theta, x, y, lam))

    # sklearn classification_report 方法 评估分类器性能
    print(classification_report(predict(theta, x), y))
    # 可视化决策边界
    draw_boundary(theta, raw_data)
```

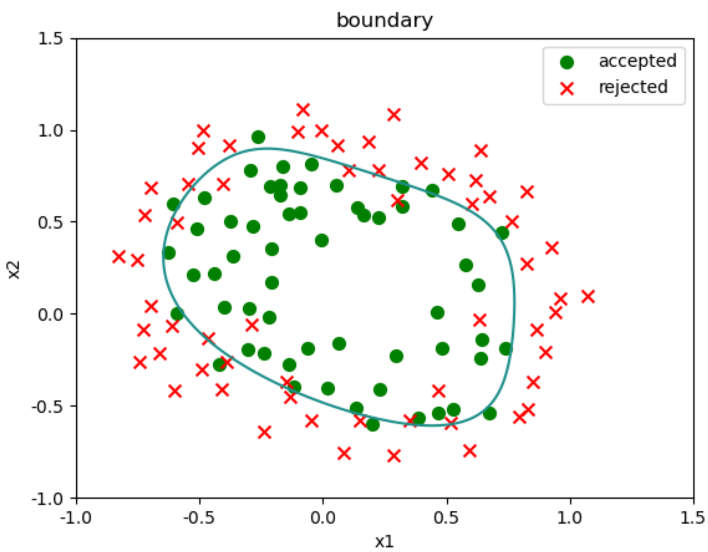
2、实验结果 使用 classification_report()评估分类器性能

①lambda=0.001 过拟合



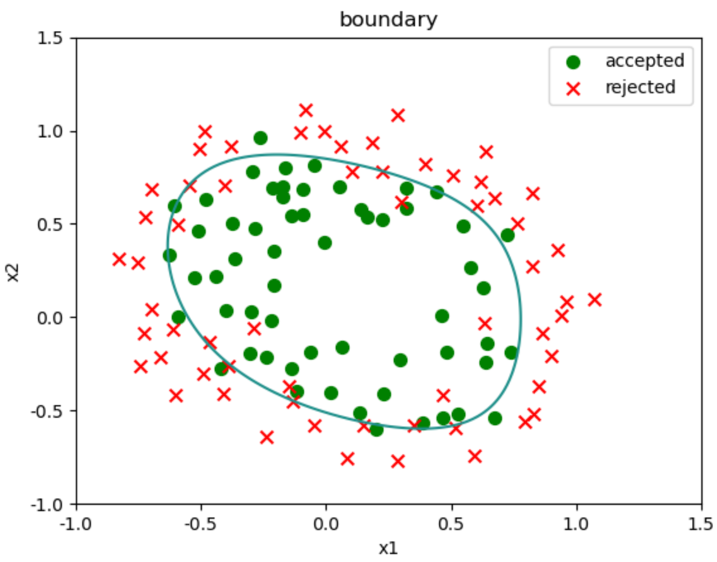
	precision	recall	f1-score	support
0	0.90	0.83	0.86	65
1	0.81	0.89	0.85	53
accuracy			0.86	118
macro avg	0.86	0.86	0.86	118
weighted avg	0.86	0.86	0.86	118

②lambda=0.01 拟合



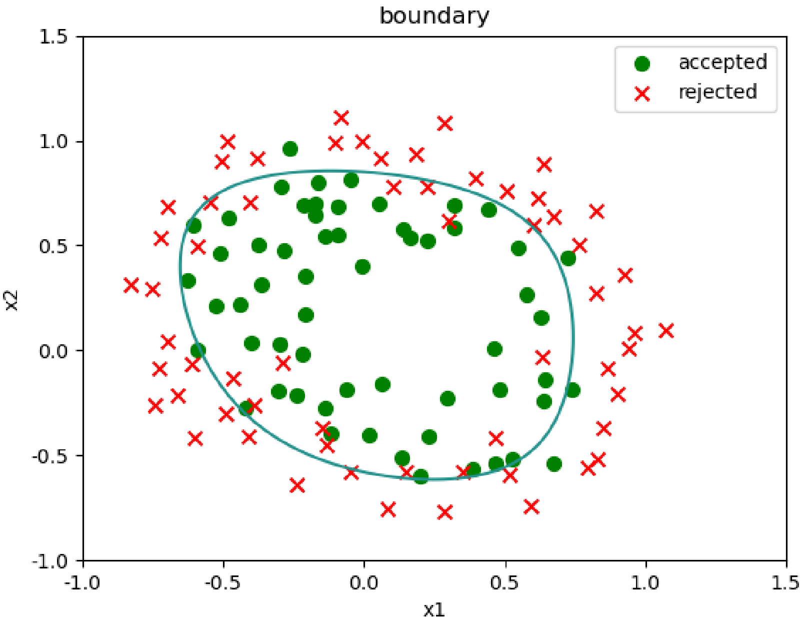
	precision	recall	f1-score	support
0	0.92	0.80	0.85	69
1	0.76	0.90	0.82	49
accuracy			0.84	118
macro avg	0.84	0.85	0.84	118
weighted avg	0.85	0.84	0.84	118

③lambda=0.1 拟合



	precision	recall	f1-score	support
0	0.88	0.78	0.83	68
1	0.74	0.86	0.80	50
accuracy			0.81	118
macro avg	0.81	0.82	0.81	118
weighted avg	0.82	0.81	0.81	118

④lambda=1 欠拟合



	precision	recall	f1-score	support
0	0.93	0.73	0.82	77
1	0.64	0.90	0.75	41
accuracy			0.79	118
macro avg	0.79	0.81	0.78	118
weighted avg	0.83	0.79	0.79	118