山东大学 计算机科学与技术 学院

机器学习 课程实验报告

学号:201900130059	姓名: 孙奇	班级: 2019 级 1 班
实验题目: Regularization		
实验学时: 2	实验日期: 2021/10/19	

实验目的:

1. 在线性回归和逻辑回归中运用正则化，观察正则化对于回归结果的影响;

硬件环境:

CPU: Intel i5-9300H

GPU: UHD630

软件环境:

Python3.8

PyCharm CE

实验步骤与内容:

1. Linear Regression with Regularization:

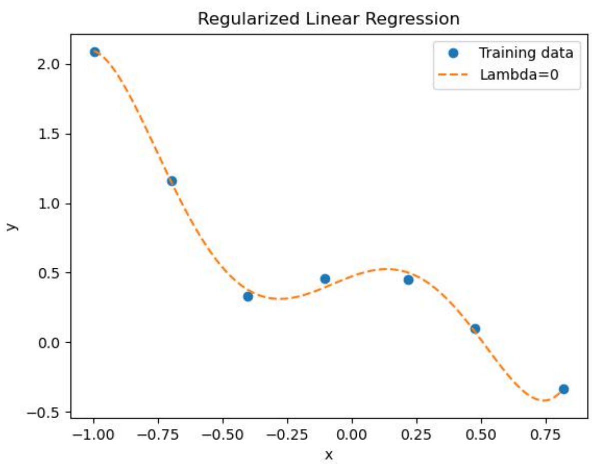
（1） 五阶多项式回归模型, hypothesis:

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

（2） Loss function:

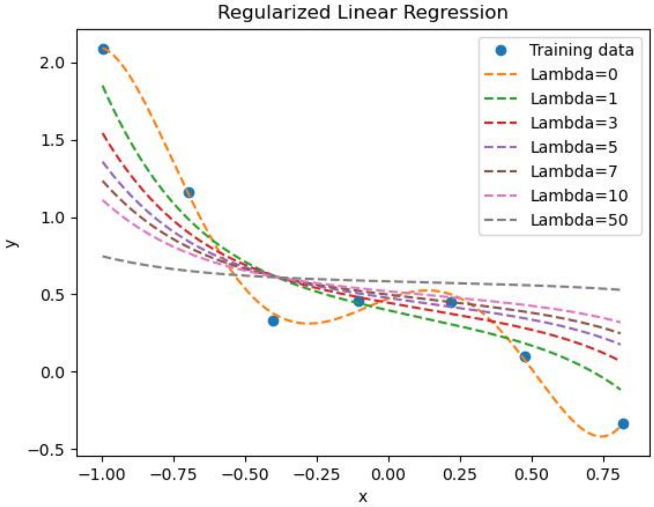
$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

（3） 在不使用正则化（即λ = 0）的情况下，绘制训练数据的散点图和五阶多项式回归结果:



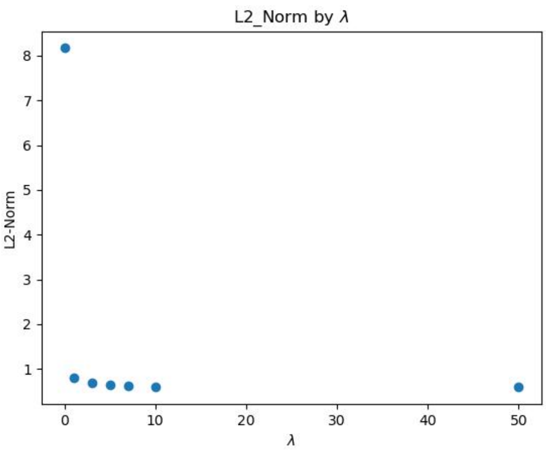
（4） 训练数据只有 7 个，这种情况下模型回归过拟合情况严重，为了使模型泛化，更具普遍性，加入惩罚项进行正则化;

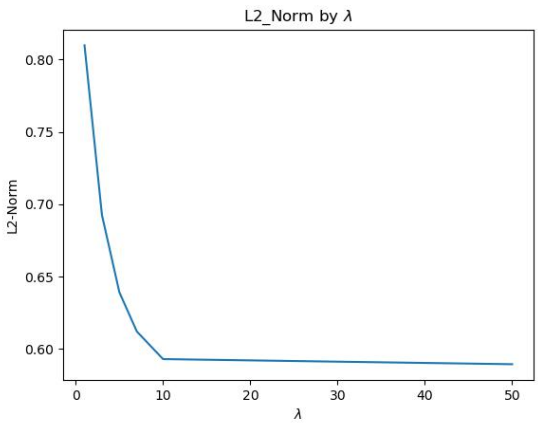
（5） 加入惩罚项 $\lambda\sum_{j=1}^{n}\theta_j^2$ ，在 λ = [1, 3, 5, 7, 10, 50] 的情况下观察模型回归效果：



当λ = 0 时，曲线过拟合，当λ比较小时，模型回归效果不错，同时也避免了过拟合，当λ比较大时，模型虽然避免了过拟合，但是回归效果很差，无法用来预测；

（6） 查看λ对 L2-Norm(θ)大小的影响：





发现λ作为惩罚项系数，在λ > 10 之后整个模型的复杂性变得很低，θ项对模型的影响力急剧缩小，同时，只要施加了惩罚项，整个模型中θ的影响力就被大幅缩小了；

2. Logistic Regression with Regularization：

（1） Hypothesis function：

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

（2） 为了使模型具有普适性，将 x 赋值为全部六阶单项式组成的向量，即

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

(3)　Loss function：

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

(4)　使用牛顿法求解该逻辑回归问题：

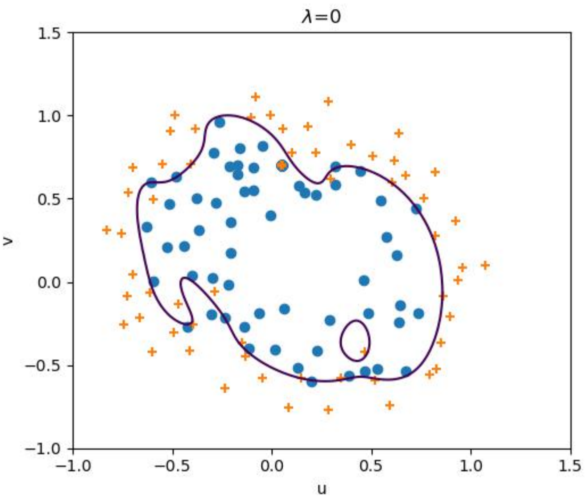
$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla_\theta J$$

(5)　带惩罚项的 Hessian 矩阵为：

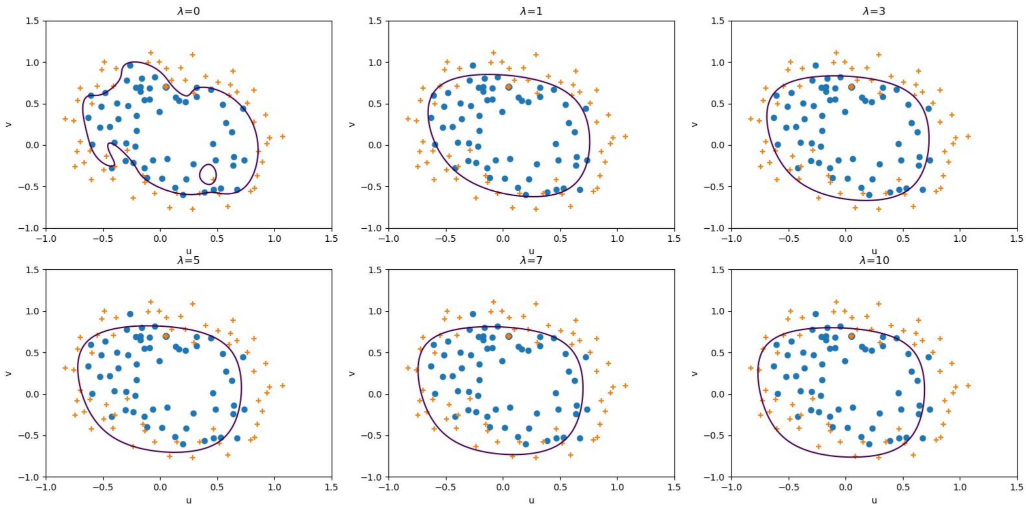
$$\nabla_\theta J = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_0^{(i)} \\ \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_1^{(i)} + \frac{\lambda}{m}\theta_1 \\ \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_2^{(i)} + \frac{\lambda}{m}\theta_2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

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

(6)　在不使用正则化（即λ ＝ 0）的情况下，绘制训练数据的散点图和决策边界如图：

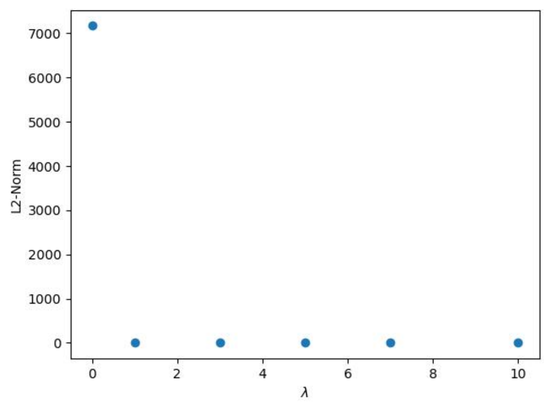


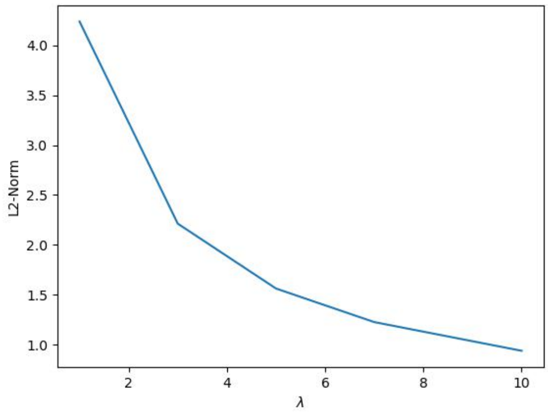
(7)　加入惩罚项 $\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$ ，在 λ ＝ [1, 3, 5, 7, 10] 的情况下观察模型回归效果：



当$\lambda = 0$时，模型过拟合，决策边界对于训练数据的划分过于具体；当λ比较小时，模型回归效果不错，同时泛化了决策边界，避免了过拟合，当λ比较大时，模型虽然避免了过拟合，泛化了决策边界，但是决策边界对于训练数据的划分偏差较大，回归效果差无法用来预测；

（8） 查看λ对 L2-Norm(θ)大小的影响：





发现λ作为惩罚项系数，在$\lambda > 10$之后整个模型的复杂性变得很低，θ项对模型的影响力急剧缩小，同时，只要施加了惩罚项，整个模型中θ的影响力就被大幅缩小了；

结论分析与体会：

1. 训练数据较少时，正则化对于线性回归和逻辑回归都有很好的避免过拟合的效果；
2. 在正则化的过程中，如果超参数λ选取合适，能够很好地泛化模型，对于过拟合起到很好的抑制作用，但是如果λ取值不当，如果太小，起不到抑制过拟合的作用；如果太大，虽然避免了模型的过拟合，但是同时导致θ的大小对惩罚项的控制力度太大，使模型产生偏差，回归效果较差，因此正则化超参数λ的选取是重要的；

附录：程序源代码

1. map_feature.py:

```
import numpy as np
```

```
def map_feature(feat1, feat2):
    degree = 6
    out = np.ones(feat1.size)

    for i in range(1, degree + 1):
        for j in range(i + 1):
            out = np.column_stack( (out, (feat1 ** (i - j)) * (feat2 ** j)) )

    return out
```

2. regularized_linear_regression.py:

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

class RegularizedLinearRegression:

    def __init__(self, x, y, lr=0.3, r_lambda=0):
        self.x = x
        self.y = y
        self.lr = lr
        self.r_lambda = r_lambda
        self.expand = lambda x, k: np.array([x ** i for i in range(2, k+1)]).T

    def train(self):
        # expand x
        m = self.x.shape[0]
        x_t = np.column_stack( (np.ones(m), self.x) )
        self.x = np.concatenate( (x_t, self.expand(self.x, 5)), axis=1 )
        x_t = self.x
        # Matrix following lambda
        n = x_t.shape[1]
        L = np.identity(n)
        L[0][0] = 0
        # solve theta
        theta = np.dot( np.linalg.inv( np.dot(x_t.T, x_t) + self.r_lambda * L ),
np.dot( x_t.T, self.y ) )

        return theta

    def plot_fit_curve(self, theta):
        x_hat = np.arange(np.min(self.x[:, 1]), np.max(self.x[:, 1]), 0.01)
        x = np.column_stack( (np.ones(len(x_hat)), x_hat) )
        x = np.concatenate( (x, self.expand(x_hat, 5)), axis=1 )
        y_hat = np.dot(x, theta)
```

```
        plt.plot(x_hat, y_hat, '--', label=f'Lambda={self.r_lambda}')

if __name__ == "__main__":
    x = np.loadtxt("data3/ex3Linx.dat")
    y = np.loadtxt("data3/ex3Liny.dat")
    plt.figure(1)
    plt.plot(x, y, 'o', label='Training data')
    lambdas = [0]
    thetas = []
    for r_lambda in lambdas:
        regLinear = RegularizedLinearRegression(x, y, r_lambda=r_lambda)
        theta = regLinear.train()
        thetas.append(theta)
        print(theta)
        regLinear.plot_fit_curve(theta)

    plt.title("Regularized Linear Regression")
    plt.legend()
    plt.xlabel("x")
    plt.ylabel("y")
    # l2_norm changes by lambda
    l2_norms = [np.linalg.norm(theta) for theta in thetas]
    plt.figure(2)
    plt.plot(lambdas, l2_norms, 'o')
    plt.title(r"L2_Norm by $\lambda$")
    plt.xlabel(r"$\lambda$")
    plt.ylabel("L2-Norm")
    # easy to observe
    plt.figure(3)
    plt.plot(lambdas[1:], l2_norms[1:])
    plt.title(r"L2_Norm by $\lambda$")
    plt.xlabel(r"$\lambda$")
    plt.ylabel("L2-Norm")
    plt.show()
```

3. regularized_logistic_regression.py:

```
import numpy as np
import matplotlib.pyplot as plt
from map_feature import map_feature

def Sigmoid(z):
    return 1 / (1 + np.exp(-z))

class RegularizedLogisticRegression:
```

```
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def Newton(self, r_lambda=0):
        m, n = self.x.shape
        theta = np.zeros((n, 1))
        # matrix L
        L = np.identity(n)
        L[0][0] = 0
        # loop
        loop_max = 100
        loop = 0
        pre_loss = 0
        loss_list = []

        print(f'---------------------Lambda={r_lambda}-------------------')
        for i in range(loop_max):
            h = Sigmoid(np.dot(self.x, theta))
            # loss
            loss = -1/m * np.sum( (self.y * np.log(h) + (1 - self.y) * np.log(1 - h)) ) + r_lambda/(2 * m) * np.sum(theta[1:] ** 2)
            print(f'Loss = {loss}')
            loss_list.append(loss)
            if abs(loss - pre_loss) < 1e-6:
                break
            pre_loss = loss
            # gradient: 28 * 1
            grad = 1/m * np.dot(self.x.T, (h - self.y))
            for i in range(1, n):
                grad[i] += r_lambda/m * theta[i]
            # update theta
            # hessain: 28 * 28
            hessian = 0
            for i in range(m):
                hessian += h[i] * (1 - h[i]) * np.dot(self.x.T[:, i].reshape(-1, 1), self.x[i, :].reshape(1, -1))
            hessian /= m
            hessian += r_lambda/m * L
            # theta: 28 * 1
            theta -= np.dot(np.linalg.inv(hessian), grad)
            loop += 1

        return theta
```

```
if __name__ == "__main__":
    # load and scatter
    x = np.loadtxt("data3/ex3Logx.dat", delimiter=',')
    y = np.loadtxt("data3/ex3Logy.dat", delimiter=',').reshape(-1, 1)
    pos = np.where(y == 1)
    neg = np.where(y == 0)
    # train
    regLogistic = RegularizedLogisticRegression(map_feature(x[:, 0], x[:, 1]), y)
    lambdas = [0, 1, 3, 5, 7, 10]
    thetas = []
    # boundary
    plt.figure(1)
    rows = cols = int(np.sqrt(len(lambdas)))
    if rows * cols < len(lambdas):
        cols += 1
    for k in range(len(lambdas)):
        plt.subplot(rows, cols, k + 1)
        plt.scatter(x[pos, 0], x[pos, 1], marker='o')
        plt.scatter(x[neg, 0], x[neg, 1], marker='+')
        # solve theta
        theta = regLogistic.Newton(r_lambda=lambdas[k])
        thetas.append(theta)
        # plot boundary
        u = np.linspace(-1, 1.5, 200)
        v = np.linspace(-1, 1.5, 200)
        z = np.zeros((len(u), len(v)))

        for i in range(len(u)):
            for j in range(len(v)):
                z[i, j] = np.dot(map_feature(u[i], v[j]), theta)

        plt.contour(u, v, z.T, [0])
        plt.title(f'$\lambda$={lambdas[k]}')
        plt.xlabel('u')
        plt.ylabel('v')
    # lambda affects results
    plt.figure(2)
    l2_norms = [np.linalg.norm(theta) for theta in thetas]
    plt.plot(lambdas, l2_norms, 'o')
    plt.xlabel(r'$\lambda$')
    plt.ylabel('L2-Norm')
    # easy to observe
    plt.figure(3)
    plt.plot(lambdas[1:], l2_norms[1:])
    plt.xlabel(r'$\lambda$')
```

```
plt.ylabel('L2-Norm')
plt.show()
```