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：201705130113	姓名：黄瑞哲	班级：计科 17.3
实验题目：正则化		
实验学时：2	实验日期：　2019.10.18	
实验目的： 掌握线性回归和逻辑回归中的正则化，防止过拟合现象的发生。		
硬件环境： Intel Core i5-8300H @ 2.3GHz		
软件环境： Windows10 Pro 1903 Python 3.7 Visual Studio Code 1.38.1		

实验步骤与内容：

线性回归

1. 读取数据并绘制散点图
2. 假定 H 函数为五次多项式

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

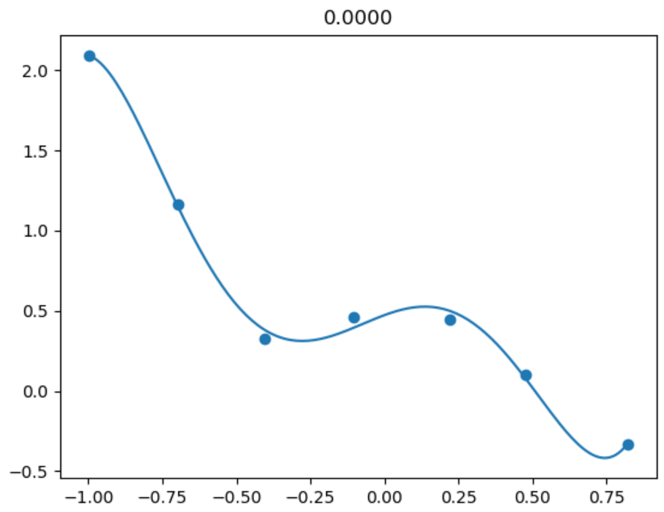
损失函数在 L2 正则化下为

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

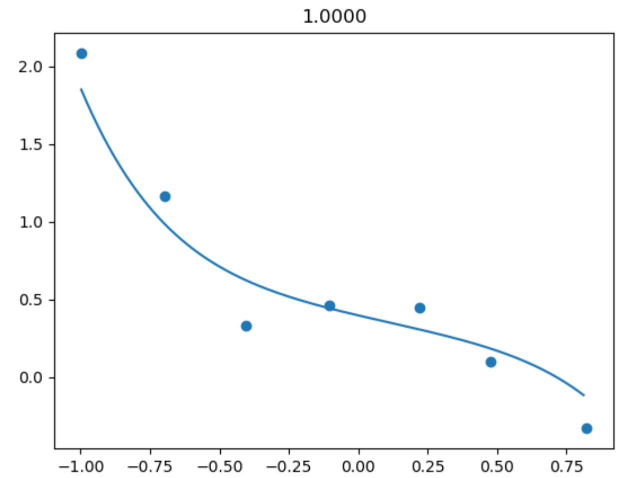
3. 在不同 lambda 下利用公式求出最优的 theta

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

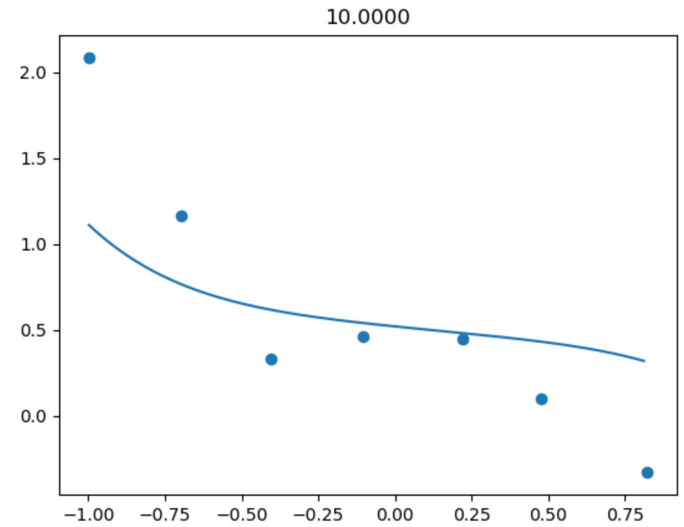
然后对与 lambda=0,1,10 分别画出对应的函数图像



当不做正则化时会发生过拟合现象



当 lambda 为 1 时适合做预测



而当 lambda 为 10 时发生了欠拟合

逻辑回归

1. 读数据画散点图
2. 定义 feature 向量是训练数据每一项的单项式组合

$$x = [1, u, v, u^2, uv, v^2, \dots, v^6]^T$$

对应的在 L2 正则下损失函数表达式为

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

3. 利用牛顿迭代法优化 theta，其中梯度表达式为

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

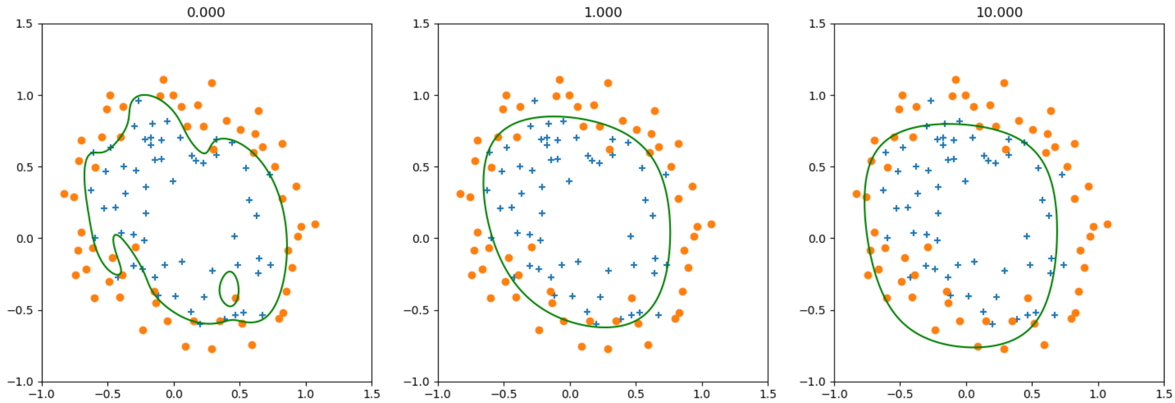
比之前不做正则化时多了一个 theta 项

同时海森矩阵变为

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

多了一个对角矩阵

4. 求出 theta 后再绘制出决策边界



可以看出当 lambda 为 0 时发生了过拟合，右下角单独出现了一个区域。

当 lambda 为 1 时适合分类

当 lambda 为 10 时决策边界已经发生了偏移，为欠拟合。

结论分析与体会：
在逻辑回归中，增加惩罚项可以看出迭代次数明显减少。
当 lambda 为 0 时迭代了 13 次，L2 范式为 7172.6662

```
iteration= 1 [[0.3490692]]
iteration= 2 [[0.30440592]]
iteration= 3 [[0.28694527]]
iteration= 4 [[0.26551459]]
iteration= 5 [[0.24963143]]
iteration= 6 [[0.24025865]]
iteration= 7 [[0.22877771]]
iteration= 8 [[0.20727845]]
iteration= 9 [[0.20160922]]
iteration= 10 [[0.20030067]]
iteration= 11 [[0.19987132]]
iteration= 12 [[0.19983777]]
iteration= 13 [[0.1998375]]
```

而到了 lambda 为 10 时仅迭代了 3 次

```
iteration= 1 [[0.64760211]]
iteration= 2 [[0.64758396]]
iteration= 3 [[0.64758396]]
```

同时对应的 L2 范式仅为 0.9384

惩罚项的设置，明显地限制了 theta 的变化，因此可以避免抖动，避免过拟合的发生，但是当 lambda 设置不当时会导致欠拟合的发生，训练出的模型没有使用价值。因此，选择合适的参数，能够维持模型的鲁棒性。

附录：程序源代码

```
# linear_regularized.py
import numpy as np

def load_data():
    feature = []
    with open("exp3/data/ex3Linx.dat") as f:
        for each_line in f.readlines():
            feature_tmp = []
            for data in each_line.strip().split():
                for i in range(6):
                    feature_tmp.append(float(data) ** i)
            feature.append(feature_tmp)
    label = []
    with open("exp3/data/ex3Liny.dat") as f:
        for each_line in f.readlines():
            label_tmp = []
            for data in each_line.strip().split():
```

```
            label_tmp.append(float(data))
        label.append(label_tmp)
    return np.mat(feature), np.mat(label)

def plt_linear(feature, label, theta, lamb):
    import matplotlib.pyplot as plt
    plt.figure()
    plt.title("%.4f" % lamb)
    plt.scatter(feature[:, 1].tolist(), label.tolist(), marker='o')
    x = np.arange(min(feature[:, 1]), max(feature[:, 1]), 0.01)
    y = []
    for i in x:
        now = 0
        for j in range(6):
            now += theta[j, 0] * (i ** j)
        y.append(now)
    plt.plot(x, y)
    plt.show()


if __name__ == '__main__':
    feature, label = load_data()
    m, n = np.shape(feature)
    lamd = [0, 1, 10]
    for lam in lamd:
        E = np.eye(n)
        E[0, 0] = 0
        theta = (feature.T * feature + lam * E).I * feature.T * label
        print("lambda=", lam)
        print(theta.T)
        plt_linear(feature, label, theta, lam)


# logistic_regularized.py
import numpy as np
import matplotlib.pyplot as plt

def sig(x): return 1. / (1. + np.exp(-x))


def load_data():
    feature = []
    with open("exp3/data/ex3Logx.dat") as f:
        for each_line in f.readlines():
            feature_tmp = [1]
```

```
        for data in each_line.strip().split(','):
            feature_tmp.append(float(data))
        feature.append(feature_tmp)
    label = []
    with open("exp3/data/ex3Logy.dat") as f:
        for each_line in f.readlines():
            label_tmp = []
            for data in each_line.strip().split():
                label_tmp.append(float(data))
            label.append(label_tmp)
    return np.mat(feature), np.mat(label)


def map_feature(feature1, feature2):
    x = []
    for i in range(7):
        for j in range(i + 1):
            x.append((feature1 ** (i - j)) * (feature2 ** j))
    return x


def cost(feature, label, theta, lamb):
    ret = 0
    m, n = np.shape(feature)
    for i in range(m):
        h = sig(feature[i] * theta)
        ret += -label[i, 0] * np.log(h) - (1 - label[i, 0]) * np.log(1 - h)
    for i in range(1, n):
        ret += lamb / 2 * (theta[i, 0] ** 2)
    return ret / m


def gradient(feature, label, theta, lamb):
    m, n = np.shape(feature)
    err = feature.T * (sig(feature * theta) - label)
    for j in range(1, n):
        err[j, 0] += lamb * theta[j, 0]
    return err / m


def hessian(feature, label, theta, lamb):
    m, n = np.shape(feature)
    H = np.mat(np.zeros((n, n)))
    for i in range(m):
        h = sig(feature[i] * theta)[0, 0]
```

```python
        H += (h * (1 - h)) * feature[i].T * feature[i]
    E = np.eye(n)
    E[0, 0] = 0
    H += lamb * E
    return H / m


def newton(feature, label, lamb, epsilon=1e-6):
    val = 0
    iteration = 0
    n = np.shape(feature)[1]
    w = np.mat(np.zeros((n, 1)))
    while True:
        iteration += 1
        H = hessian(feature, label, w, lamb)
        dJ = gradient(feature, label, w, lamb)
        w = w - H.I * dJ
        last, val = val, cost(feature, label, w, lamb)
        print("iteration=", iteration, val)
        if abs(last - val) <= epsilon:
            break
    return w


def plt_logistic(feature, label, theta):
    # divide pos and neg
    x_pos = []
    x_neg = []
    y_pos = []
    y_neg = []
    for (x, y) in zip(feature, label):
        if y[0] == 0:
            x_neg.append(x[0, 1])
            y_neg.append(x[0, 2])
        else:
            x_pos.append(x[0, 1])
            y_pos.append(x[0, 2])
    plt.scatter(x_pos, y_pos, marker='+')
    plt.scatter(x_neg, y_neg, marker='o')
    # plot contour
    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[j, i] = map_feature(u[i], v[j]) * theta
    plt.contour(u, v, z, [0], colors='g')


if __name__ == '__main__':
    feature, label = load_data()
    # all monomials
    _feature = []
    for i in range(len(feature)):
        _feature.append(map_feature(feature[i, 1], feature[i, 2]))
    _feature = np.mat(_feature)

    plt.figure()
    # we calculate theta for each lambda
    lamb = [0, 1, 10]
    for i in range(len(lamb)):
        theta = newton(_feature, label, lamb[i])
        print("theta=", theta.T)
        print("norm=", np.linalg.norm(theta))
        ax = plt.subplot(1, len(lamb), i + 1)
        ax.set_title("%.3f" % lamb[i])
        plt_logistic(feature, label, theta)
    plt.show()
```