Lab 5: 正则化线性回归 偏差与方差

介绍

在本次实验中，您将实现正则化线性回归，并使用它来研究具有不同偏差-方差属性的模型。

用到的数据

ex5.mat

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.io import loadmat
import scipy.optimize as opt
```

1 正则化线性回归

在练习的前半部分，您将实现正则化线性回归，利用水库水位的变化来预测从大坝流出的水量。在下半部分中，您将对调试学习算法进行一些诊断，并检查偏差和方差的影响。

1.1 可视化数据

首先，我们将可视化包含水位变化的历史记录的数据集，X 和从大坝流出的水量 y。该数据集可分为三个部分：

1. **训练集 X，y，用于训练模型；**
2. **验证集 Xval，yval，用于选择正则化参数；**
3. **测试集 Xtest，ytest，用于评估性能；**

接下来我们将绘制训练数据。

```
def load_mat(path):
    # 读取ex5.mat数据
    return X,y,Xval,yval,Xtest,ytest
```

```
def plotdata(X,y):
    # 绘制训练集X，y的图像
    # plt.grid(False)
```

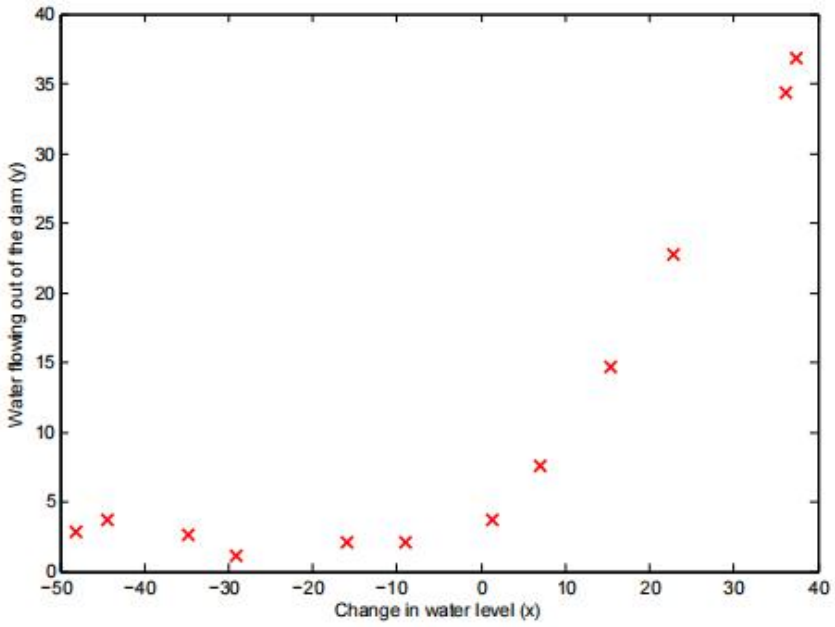

Figure 1: Data

1.2 正则化线性回归损失函数

回想一下，正则化的线性回归具有以下损失函数：

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

其中 λ 是控制正则化程度的正则化参数，因此可以用于防止过拟合，正则化项对总代价 J(θ)施加惩罚，随着模型参数 θj 的增大，惩罚也增大，需要注意的是不惩罚 θ0 项，即索引从 1 开始。

```
def costReg(theta,X,y,l):
    # 损失函数
    return cost
```

当 θ 的初始为[1;1]时，所得到的输出应该是 303.993。在这里一定要注意公式后半部分正则化处不包括 θ0 这一项，不然输出就变成 304.04，反复检查了好几遍才发现这里出了问题。

1.3 正则化线性回归梯度

正则化线性回归梯度公式定义如下：

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{for } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{for } j \geq 1$$

```
def gradientReg(theta,X,y,l):
    # 梯度函数
    return grad
```

当 θ 的初始为[1;1]时，梯度的计算结果应为[-15.30;598.250]。

1.4 拟合线性回归

在这一部分中，**我们将正则化参数λ设为零。因为我们目前的线性回归实现是试图拟合一个二维的θ，正则化对于如此低维的θ不会非常有用。**你将调用 scipy.optimize 中的 minimize 函数计算参数最优解，并将拟合函数与原始数据画在一起。

```
def TrainLinearReg(X,y,l):
    theta=np.zeros(X.shape[1])
    res=opt.minimize(fun=CostReg,x0=theta,args=(X,y,l),
                     method='TNC',jac=GradientReg)
    return res.x
```

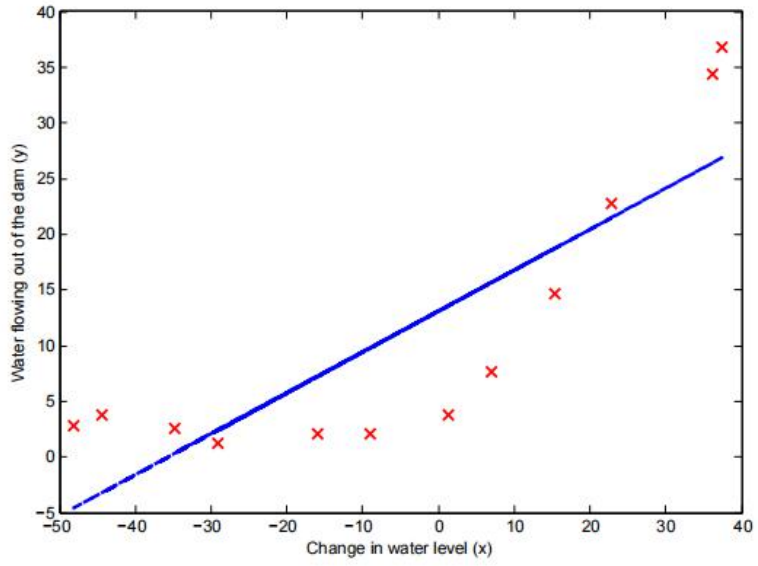

Figure 2: Linear Fit

最佳拟合直线告诉我们，**由于数据具有非线性，模型与数据的拟合不是很好。**虽然如图所示可视化最佳拟合是调试学习算法的一种可能方法，但可视化数据和模型并不总是那么容易。于是我们可以实现一个生成学习曲线的函数，即使数据

不容易可视化，它也可以帮助调试学习算法。

2 偏差-方差

机器学习中的一个重要概念是偏差-方差权衡。

偏差代表着模型与数据的拟合程度，高偏差的模型容易出现拟合不足。

方差代表着模型的泛化能力，高方差的模型对训练数据会出现过度拟合，即对新的样本泛化能力很差。

我们可以通过绘制训练集与验证集的误差，来判断偏差与方差问题。

2.1 学习曲线

为了绘制学习曲线，我们需要不同训练集大小的训练和交叉验证集误差。为了获得不同的训练集大小，你应该使用原始训练集 X 的不同子集。具体来说，对于一个训练集大小为 i 的集合，你应该使用前 i 个示例。

模型的最优参数可以通过前面提到的 minimize 函数进行求解，得出模型参数后开始计算训练集与验证机上的误差，数据集的训练错误定义如下：

$$J_{\text{train}}(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2\right].$$

特别需要注意的是，**训练误差不包括正则化项**。计算训练误差的一种方法是使用您现有的代价函数并设置 λ 为 0，只有当使用它来计算训练误差和交叉验证误差时。当计算训练集误差时，确保你是在训练子集上计算它(而不是整个训练集)。但是，对于交叉验证错误，需要在整个交叉验证集上计算它。最后将计算的误差存储在向量、误差序列和误差值中，以便于可视化。

```
def plot_learning_curve(X,y,Xval,yval,1):
    size = range(1, len(X)+1)
    error_train, error_val = [], []
    for i in size:
        i_theta = TrainLinearReg(X[:i, :], y[:i], 1)
        i_train_cost = costReg(i_theta, X[:i, :], y[:i], 0)
        i_val_cost = costReg(i_theta, Xval, yval)
        error_train.append(i_train_cost)
        error_val.append(i_val_cost)

    fix, ax = plt.subplots(figsize=(12, 8))
    ax.plot(size,error_train,label="Train",color="blue")
    ax.plot(size,error_val,label="Cross Validation",color="green
    ax.legend()
    ax.set_xlabel("Number of training examples")
    ax.set_ylabel("Error")
    ax.set_title("Learning curve for linear regression")
    ax.grid(False)
    plt.show()
plot_learning_curve(X, y, Xval, yval, 0)
```

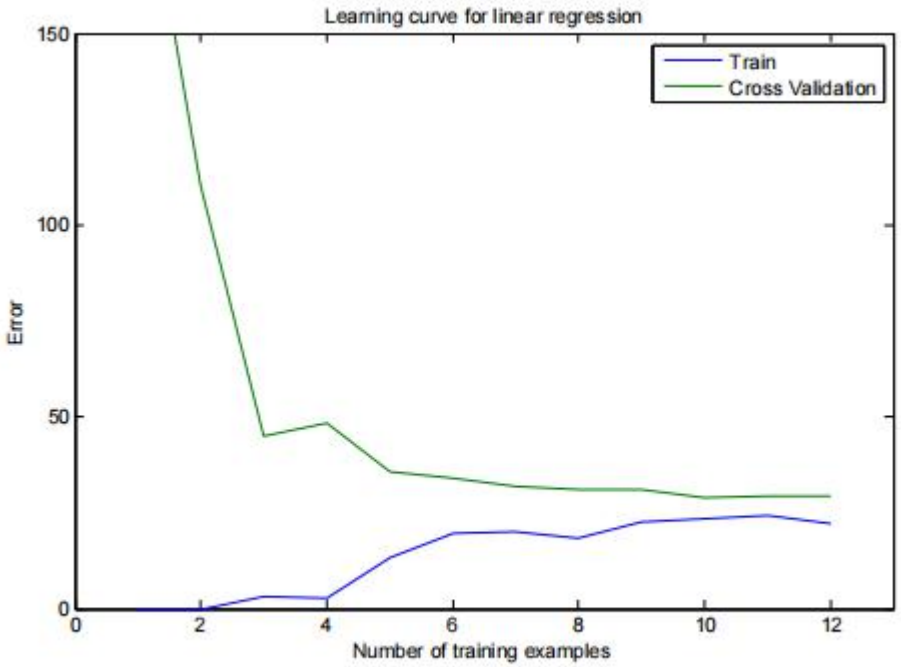

Figure 3: Linear regression learning curve

图中可知随着训练样本增加，两者误差都很高，说明线性回归不能很好拟合数据集，出现了**高偏差问题**。

3 多项式回归

在前面的模型中出现的问题是，它对数据过于简单，导致拟合很差，所以采用**添加更多特征**来解决此问题。**对于多项式回归，假设形式如下，增加的特征为原始值的各种幂次。**在这次实验中，增加的特征是原始值（水位）的各种幂。

$$h_\theta(x) = \theta_0 + \theta_1 * (\text{waterLevel}) + \theta_2 * (\text{waterLevel})^2 + \cdots + \theta_p * (\text{waterLevel})^p$$
$$= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p.$$

```
def poly_feature(X, power):
    # 特征映射，只有各次幂
    return Xpoly
```

3.1 学习多项式回归

增加了特征之后，由于增加的特征为各次幂，故数据差异很大，比如 x = 40，它的八次方就达到了 10 的 12 次方，所以我们需要对特征向量**进行特征标准化**。

```
def get_means_std(X):
    mean=np.mean(X,axis=0)
    std=np.std(X,axis=0,ddof=1)
    return mean,std
def featureNormalize(X, mean, std):
    X[:,1:]=X[:,1:]-mean[1:]
    X[:,1:]=X[:,1:]/std[1:]
    return X
```

```
# 数据处理
power = 8
train_means, train_stds = get_means_std(poly_features(X, power))
X_norm = featureNormalize(poly_features(X, power), train_means, train_stds)
Xval_norm = featureNormalize(poly_features(Xval, power), train_means, train_stds)
Xtest_norm = featureNormalize(poly_features(Xtest, power), train_means, train_stds)
```

在学习了参数 θ 之后，您应该会看到使用 λ = 0 为多项式回归生成的两个图（如下）。

```
def plot_fit(means,stds,l):
    # 画出拟合图像
    plt.show()
```

```
plot_fit(train_means, train_stds, 0)
plot_learning_curve(X_norm, y, Xval_norm, yval, 0)
```

从图 4 中，您应该可以看到，多项式拟合能够很好地跟踪数据点——因此，获得了一个较低的训练误差。然而，多项式拟合是非常复杂的，甚至在极端值下降。这表明多项式回归模型过拟合训练数据，不能很好地泛化。

为了更好地理解非正则化（λ = 0）模型的问题，您可以看到学习曲线（图 5）显示了相同的效果，但交叉验证错误高。训练错误和交叉验证错误之间存在差距，表明存在**高方差问题**。

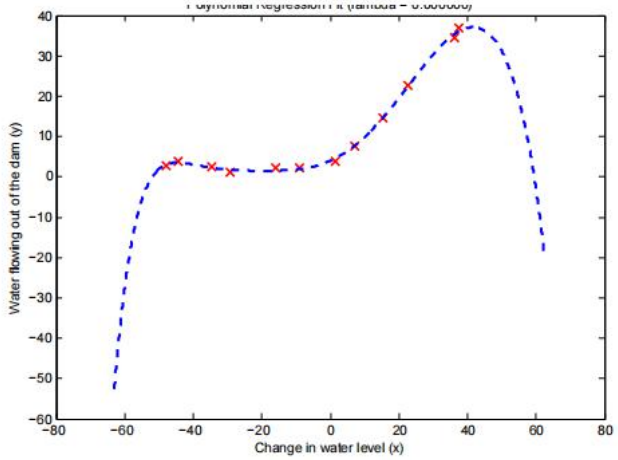

Figure 4: Polynomial fit, $\lambda = 0$

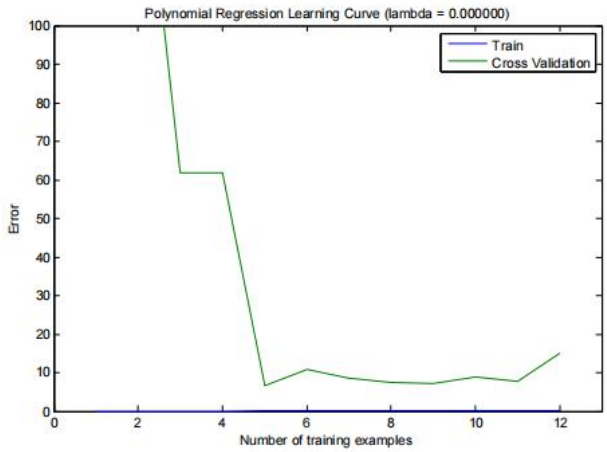

Figure 5: Polynomial learning curve, $\lambda = 0$

解决过拟合（高方差）问题的一种方法是在模型中添加正则化，在下一节中，您将尝试不同的λ参数，以了解正则化如何导致更好的模型。

3.2 调整正则化参数

在本节中，您将观察正则化参数如何影响正则化多项式回归的偏差-方差。

对于λ = 1，您应该可以看到一个很好地遵循数据趋势的多项式拟合（图 6）和一个学习曲线（图 7），这表明交叉验证和训练误差都收敛到一个相对较低的值。这表明λ = 1 正则化多项式回归模型不存在高偏差或高方差问题。实际上，它在偏差和方差之间实现了一个很好的权衡。

```
plot_fit(train_means, train_stds, 1)
plot_learning_curve(X_norm, y, Xval_norm, yval, 1)
```

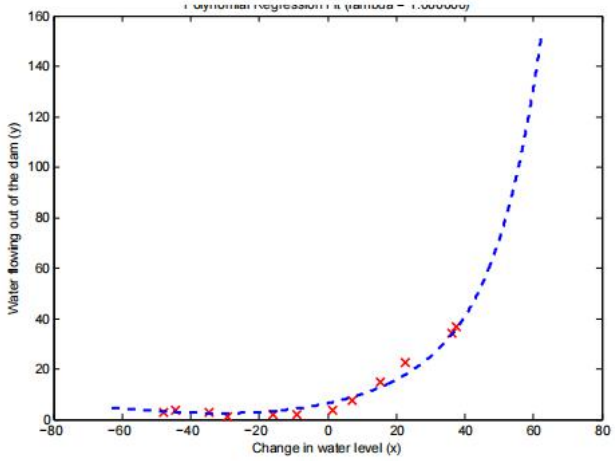

Figure 6: Polynomial fit, $\lambda = 1$

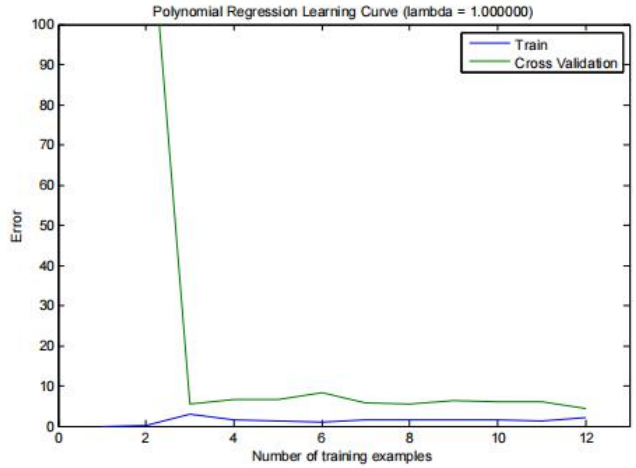

Figure 7: Polynomial learning curve, $\lambda = 1$

对于λ = 100，您应该会看到一个不能很好地跟踪数据的多项式拟合（图 8）。在这种情况下，存在太多的正则化，模型无法拟合训练数据。

```
plot_fit(train_means, train_stds, 100)
plot_learning_curve(X_norm, y, Xval_norm, yval, 100)
```

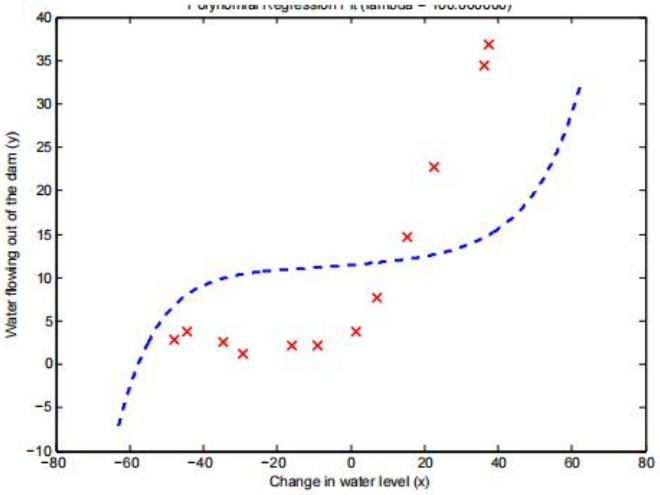

Figure 8: Polynomial fit, $\lambda = 100$

3.3 使用交叉验证集选择 λ

从练习的前一部分中，您观察到λ的值可以显著地影响训练集和交叉验证集上的正则化多项式回归的结果。特别是，没有正则化的模型（λ = 0）很好地适合训练集，但不能泛化。相反的是，正则化模型（ λ = 100）不能很好地适应训练集和测试集。一个好的选择的 λ（例如， λ = 1）可以提供一个很好的数据拟合。

在本节中，您将实现一个自动的方法来选择λ参数。具体地说，您将使用一个交叉验证集来评估每个λ值的好坏。在使用交叉验证集选择最佳λ值后，我们可

以在测试集上评估模型，以估计模型在实际看不见的数据上的表现。

具体来说，您应该使用不同的λ值来训练模型，并计算训练误差和交叉验证误差。您应该在以下范围内尝试**λ：{0、0.001、0.003、0.01、0.03、0.1、0.3、1、3、10}**。

```
lambdas = [0., 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1., 3., 10.]
errors_train, errors_val = [], []
for l in lambdas:
    theta=TrainLinearReg(X_norm, y, l)
    errors_train.append(costReg(theta, X_norm, y,0))
    errors_val.append(costReg(theta, Xval_norm, yval,0))
```

完成代码后，您应该会看到一个类似于图 9 的图。在这个图中，我们可以看到λ的最佳值在 3 左右。由于数据集的训练和验证分割的随机性，交叉验证误差有时会低于训练误差。

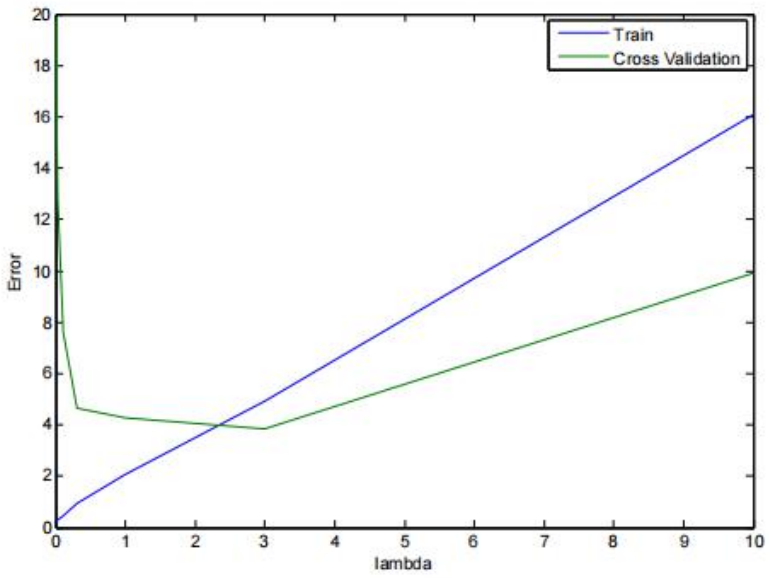

Figure 9: Selecting λ using a cross validation set

3.4 计算测试集误差

在练习的前一部分中，您实现了代码来计算正则化参数λ的各种值的交叉验证错误。然而，得到更好的指示模型的性能在现实世界中，重要的是评估“最终”模型测试集没有用于任何部分的训练（也就是说，它既不用于选择λ参数，也不是学习模型参数θ）。

您应该使用找到的最佳λ值计算测试错误。在我们的交叉验证中，**我们得到了λ = 3 的测试误差为 3.8599。**

```
1 theta=TrainLinearReg(X_norm,y,3)
2 print("l={}时,测试误差={}".format(3,CostReg(theta, Xtest_norm, ytest,0)))
```

```
l=3时,测试误差=3.859889027944475
```