Regularized linear regression: bias and variance

The objective of this practice is to verify the effects of bias and variance. To do that it is needed to implement a regularized linear regression for a biased hypothesis. Later using a higher degree polynomial we will overfit a model to our training dataset.

First part- regularized linear regression

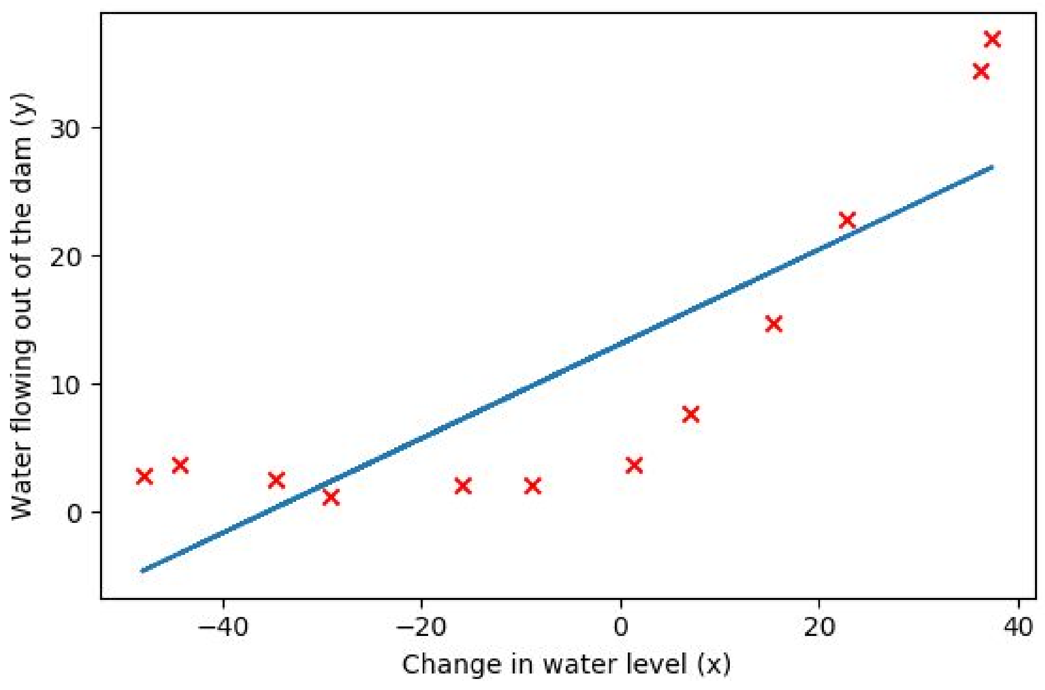
Our code:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize as opt
from scipy.io import loadmat
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import PolynomialFeatures

data = loadmat('ex5data1.mat')
#se pueden consultar las claves con data.keys( )
y = data ['y']
X = data ['X']
X_val = data ['Xval']
y_val = data ['yval']
X_test = data ['Xtest']
y_test = data ['ytest']

#almacena los datos leídos en X, y

m = np.shape(X)[0]
m_val = np.shape(X_val)[0]
n = np.shape(X)[1]
X = np.hstack([np.ones([len(X), 1]), X])
X_val = np.hstack([np.ones([len(X_val), 1]), X_val])
#lineal regression

def init_theta(X):
    return np.ones((X.shape[1], ))

def init_theta_zeros(X):
    return np.zeros((X.shape[1], ))

def hypothesis(X, Theta):
    return np.dot(X, Theta)

def cost(Theta,X, Y, reg):
    aux = (reg / (2 * m)) * np.sum(Theta[1:] ** 2)
    return 1/(2*len(X)) * (np.sum(np.square(hypothesis(X, Theta)-Y))) +
aux

def gradient(Theta, X, Y, reg):
    Y = Y.ravel()
    coste = cost(Theta, X, Y, reg)
    grad = 1 / len(X) * np.dot((hypothesis(X, Theta) - Y), X)
    grad[1:] += (reg / len(X)) * Theta[1:]
    return (coste, grad.ravel())

print(gradient(init_theta(X), X, y, 1))
```

Running the above code with **λ** = 1 and Theta = [1:1] we have obtained a cost of 303.9931922202643 and a gradient of [-15.30301567, 598.25074417].

```
def model_lineal_regression(X, y, reg):
    y = y.ravel()
    theta = init_theta(X)
    min = opt.minimize(gradient, theta, args=(X, y, reg), method='TNC',
jac=True)
    theta = min.x
    plt.scatter(X[:, 1:], y, c = 'red', marker = 'x')
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.plot(X[:, 1], hypothesis(X, theta))
    return theta

model_lineal_regression(X, y, 0)
```

Training the model using the above function generated us the image below.

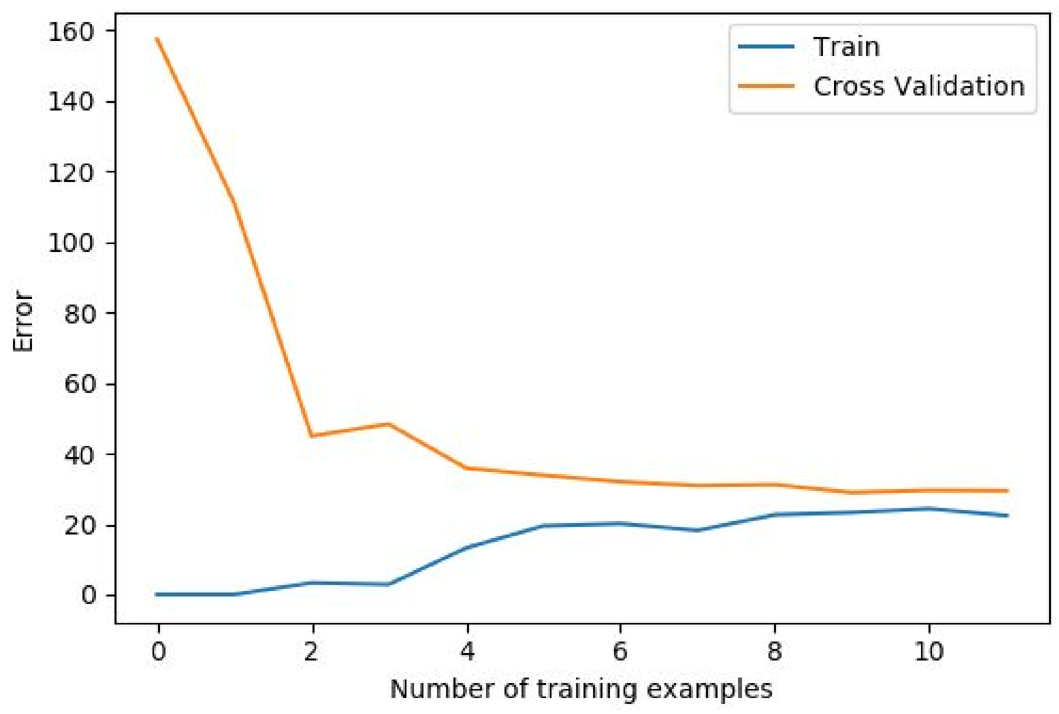


Second part- learning curves

Our code:

```
def curvas_aprendizaje(X, y, X_val, y_val, reg):
    dots = np.arange(m)
    error_dots = np.zeros(m)
    error_val = np.zeros(m)
    for i in range(1, m + 1):
        theta = model_lineal_regression(X[0 : i], y[0 : i], reg)
        error_dots[i - 1] = gradient(theta, X[0:i], y[0:i], reg)[0]
        error_val[i - 1] = gradient(theta, X_val, y_val, reg)[0]
    plt.plot(dots, error_dots, label='Train')
    plt.plot(dots, error_val,  label='Cross Validation')
    plt.xlabel("Number of training examples")
    plt.ylabel("Error")
    plt.legend(loc = 'upper right')

curvas_aprendizaje(X, y, X_val, y_val, 2)
```

In this part, we have trained our model using a different number of examples. After that, we have calculated the errors for both the training and the cross-validation set. The below graph represents our results.

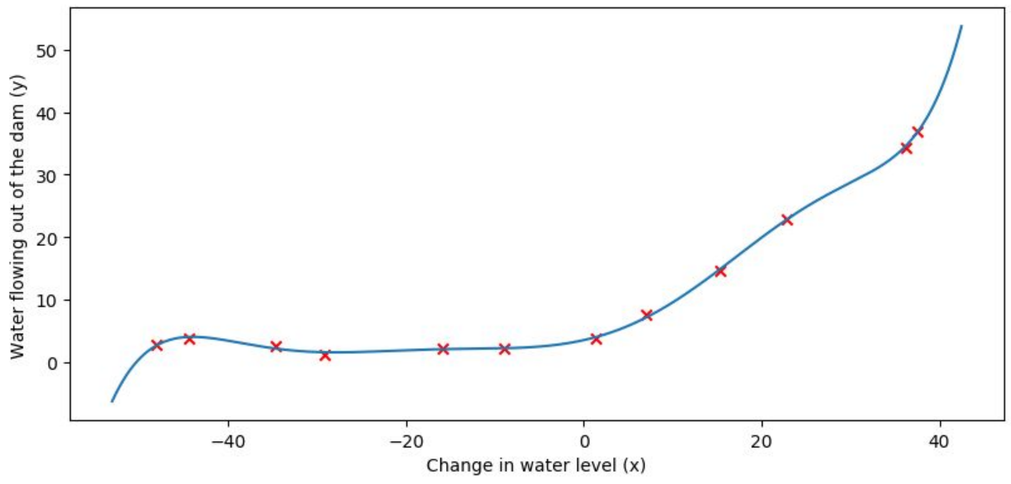


<u>Third part- polynomial regression</u>

<u>Our code:</u>

```
def modify_x(X, p):
    return PolynomialFeatures(p, include_bias = False).fit_transform(X)

def normalize(X):
    mean = np.mean(X, axis=0)
    std = np.std(X, axis=0)
    return ((X - mean) / std, mean, std)

def model_polinomial_regression(X, X_pol, norm, y, reg): #X[:, 1:]
expected
    y = y.ravel()
    theta = init_theta_zeros(X_pol)
    min = opt.minimize(gradient, theta, args=(X_pol, y, reg),
method='TNC', jac=True)
    theta = min.x

    dots  = np.arange(X.min() - 5, X.max() + 5, 0.05)
    dots_pol = modify_x(dots.reshape(-1,1), 8)
    dots_pol = (dots_pol-norm[1])/norm[2]
    dots_pol = np.hstack([np.ones([len(dots_pol), 1]), dots_pol])

    plt.scatter(X, y, c = 'red', marker = 'x')
    plt.plot(dots, hypothesis(dots_pol, theta))
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")

    return theta

X_pol = modify_x(X[:, 1:], 8)
norm = normalize(X_pol)
X_pol = norm[0]
X_pol = np.hstack([np.ones([len(X), 1]), X_pol])

X_val_pol = modify_x(X_val[:, 1:], 8)
X_val_pol = (X_val_pol - norm[1]) / norm[2]
X_val_pol = np.hstack([np.ones([len(X_val), 1]), X_val_pol])

model_polinomial_regression(X[:, 1:], X_pol, norm, y, 0)
```

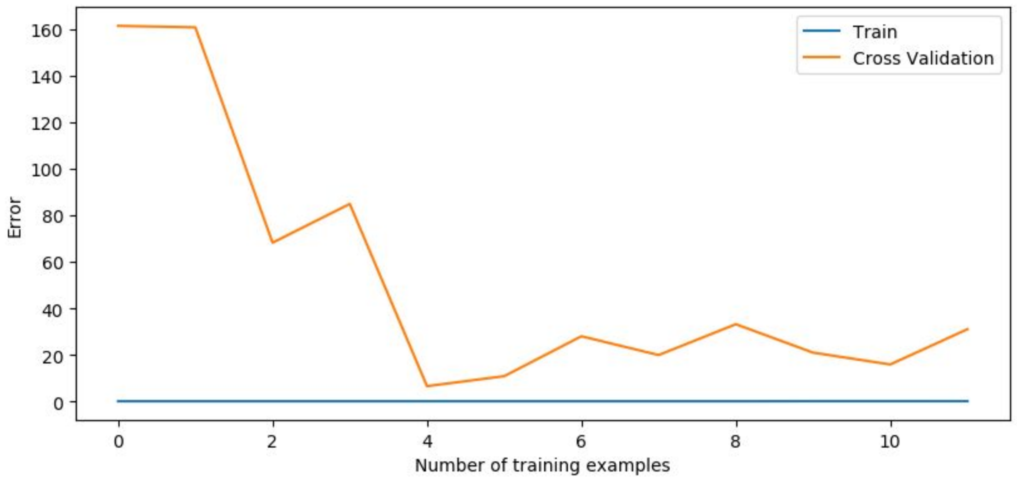
Executing the polynomial regression with **λ** = 0 we have obtained the graph below.



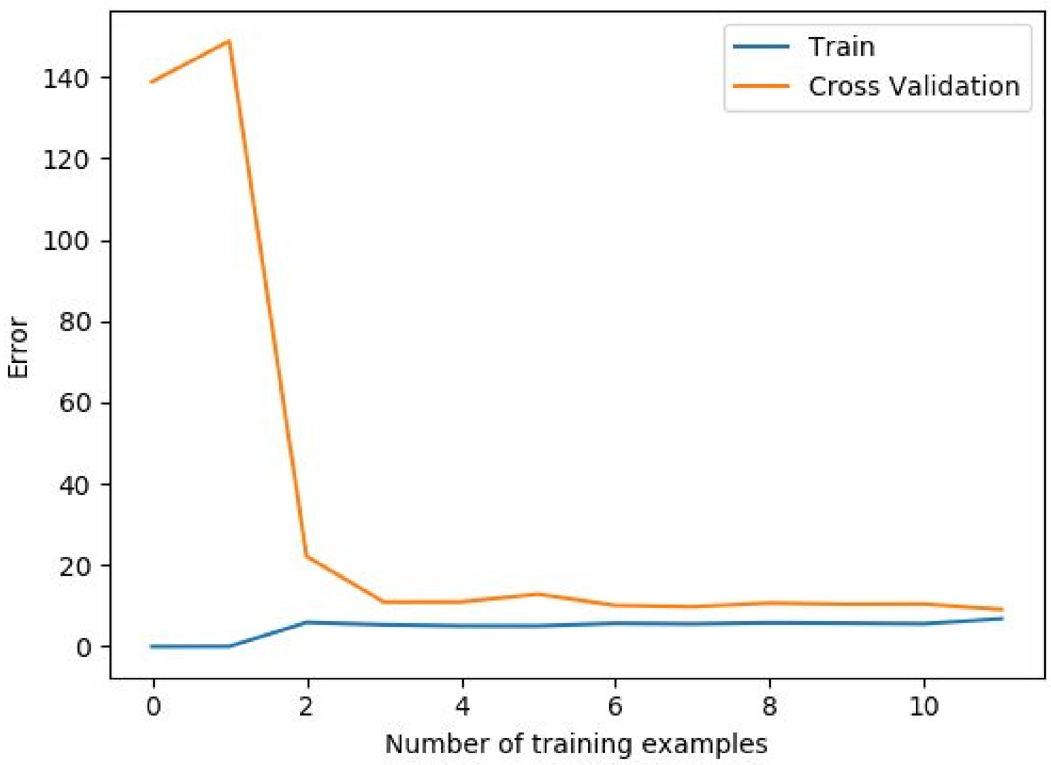
```
def graphic_pol(X, y, X_val, y_val, reg):
    dots = np.arange(m)
    error_dots = np.zeros(m)
    error_val = np.zeros(m)
    for i in range(1, m + 1):
        theta = model_polinomial_regression(None, X[0 : i], None, y[0 :
i], reg)
        error_dots[i - 1] = gradient(theta, X[0:i], y[0:i], reg)[0]
        error_val[i - 1] = gradient(theta, X_val, y_val, reg)[0]
    plt.plot(dots, error_dots, label='Train')
    plt.plot(dots, error_val,  label='Cross Validation')
    plt.xlabel("Number of training examples")
    plt.ylabel("Error")
    plt.legend(loc = 'upper right')


graphic_pol(X_pol, y, X_val_pol, y_val, 0)
```

After training the model the next step was to generate the learning curves implementing the same methodology described in part 2 of this practice- using again the error of our training set in comparison with the error from our cross-validation one.

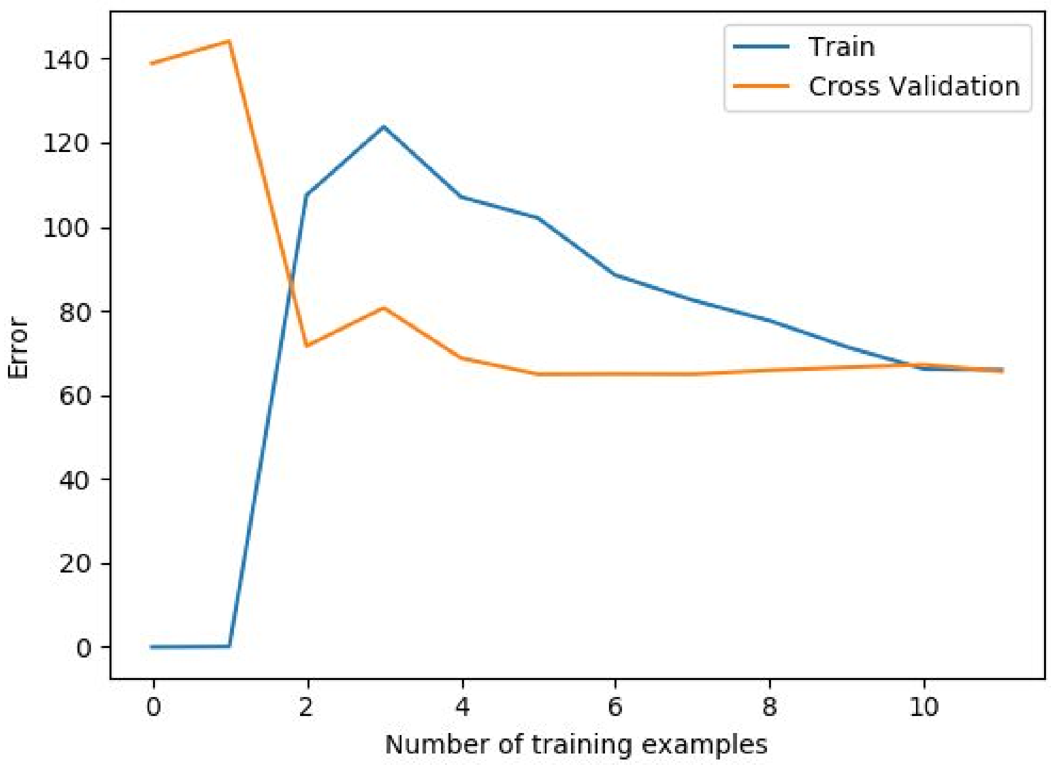
When **λ** = 0:



When **λ** = 1:



When λ = 100:

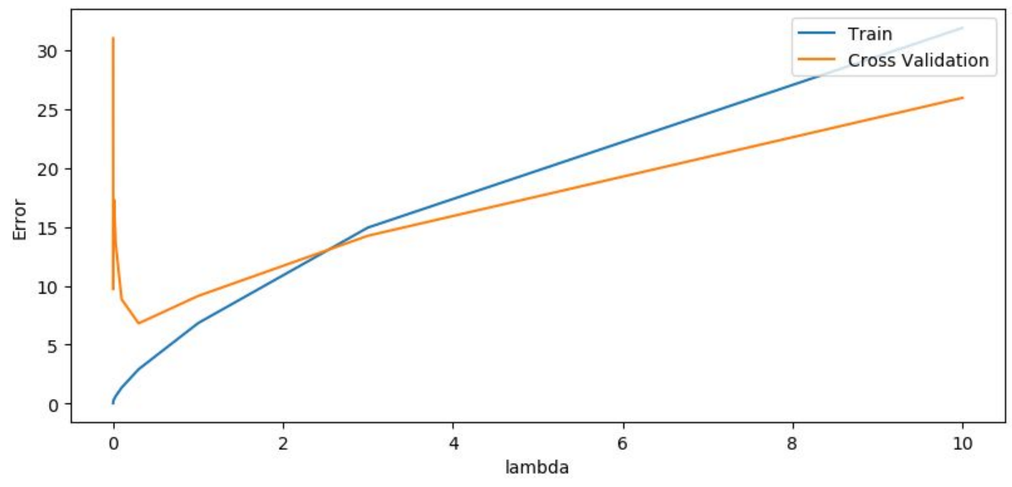


<u>Fourth part- λ selection</u>

<u>Our code:</u>

```
def choose_lambda(X, y, X_val, y_val):
    lambd = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10])
    error_dots = np.zeros(len(lambd))
    error_val = np.zeros(len(lambd))
    for i in range(len(lambd)):
        theta = model_polinomial_regression(None, X, None, y, lambd[i])
        error_dots[i] = gradient(theta, X, y, lambd[i])[0]
        error_val[i] = gradient(theta, X_val, y_val, lambd[i])[0]
    plt.plot(lambd, error_dots, label='Train')
    plt.plot(lambd, error_val, label='Cross Validation')
    plt.xlabel("lambda")
    plt.ylabel("Error")
    plt.legend(loc = 'upper right')

choose_lambda(X_pol, y, X_val_pol, y_val)
```

Executing the above code we generated the corresponding graph where we can see that the optimal value for our parameter **λ** seems to be around 3.



Finally, we have calculated the error of our test set- 3.5720416282307994.

```
X_test_pol = modify_x(X_test, 8)
X_test_pol = (X_test_pol - norm[1]) / norm[2]

X_test_pol = np.hstack([np.ones([len(X_test), 1]), X_test_pol])

theta = model_polinomial_regression(None, X_pol, None, y, 3)
```

```
print(gradient(theta, X_test_pol, y_test, 0)[0])


plt.show()
```

Conclusion:

In this practice, we have implemented two linear regressions with the object to get familiar and understand when a model is biased and/or has a variance problem. We have implemented a mechanism to compare train and cross-validation datasets. Additionally, we have searched the best value of the parameter **λ** and applied it to our test dataset.

Gasan Nazer and Veronika Yankova