Práctica 5: Regresión lineal regularizada: sesgo y varianza

Álvar Domingo Fernández y Pablo Jurado López

Preparación inicial

A continuación se importan todas las librerías externas que serán utilizadas en esta práctica y se cargan los datos para hacer las distintas matrices sobre las que se va a trabajar.

[ ]:
```
import numpy as np
from numpy.lib.function_base import gradient
import operator
import matplotlib.pyplot as plt
from numpy.lib.npyio import load
import scipy.optimize as opt
from scipy.io import loadmat

X = data['X']
X_orig = X
y = data['y']

X_test = data['Xtest']
X_test_orig = X_test
y_test = data['ytest']

X_val = data['Xval']
X_val_orig = X_val
y_val = data['yval']

X = np.hstack([np.ones([X.shape[0],1]),X])
X_val=np.hstack([np.ones([X_val.shape[0],1]),X_val])
X_test=np.hstack([np.ones([X_test.shape[0],1]),X_test])

thetas = np.ones(X.shape[1])
```

Utilizaremos durante toda la práctica las siguientes funciones para calcular el coste y el gradiente:

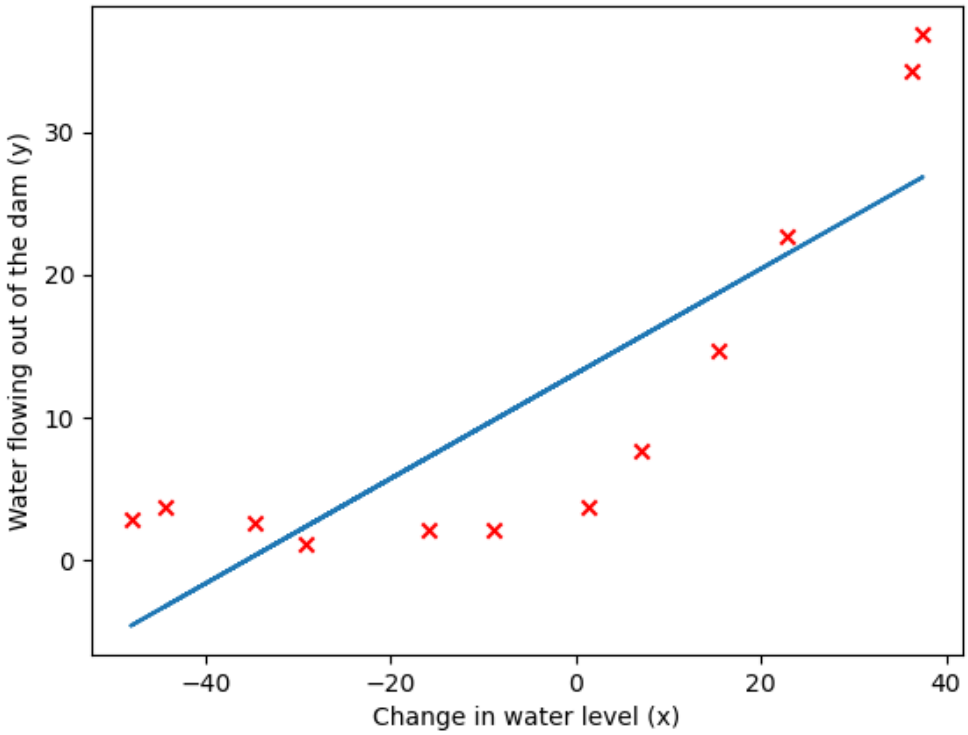
[ ]:
```
def cost(thetas, X, Y, reg=0):
    m = X.shape[0]
    H = np.dot(X, thetas)
```

```
    cost = (1/(2*m)) * np.sum(np.square((H-Y.T))) + (reg/(2*m)) * np.sum(np.
 ↪square(thetas[1:]))
    return cost

def gradient(thetas, X, Y, reg=0):
    aux = np.hstack(([0], thetas[1:]))
    m = X.shape[0]
    H = np.dot(X, thetas)
    grad = (1/m) * np.dot((H-Y.T), X) + (reg/m) * aux
    return grad
```

1- Regresión lineal regularizada

Hemos utilizado la función scipy.optimize.minimize para encontrar el valor de θ que minimiza el error sobre los ejemplos de entrenamiento y posteriormente lo hemos dibujado sobre una gráfica:

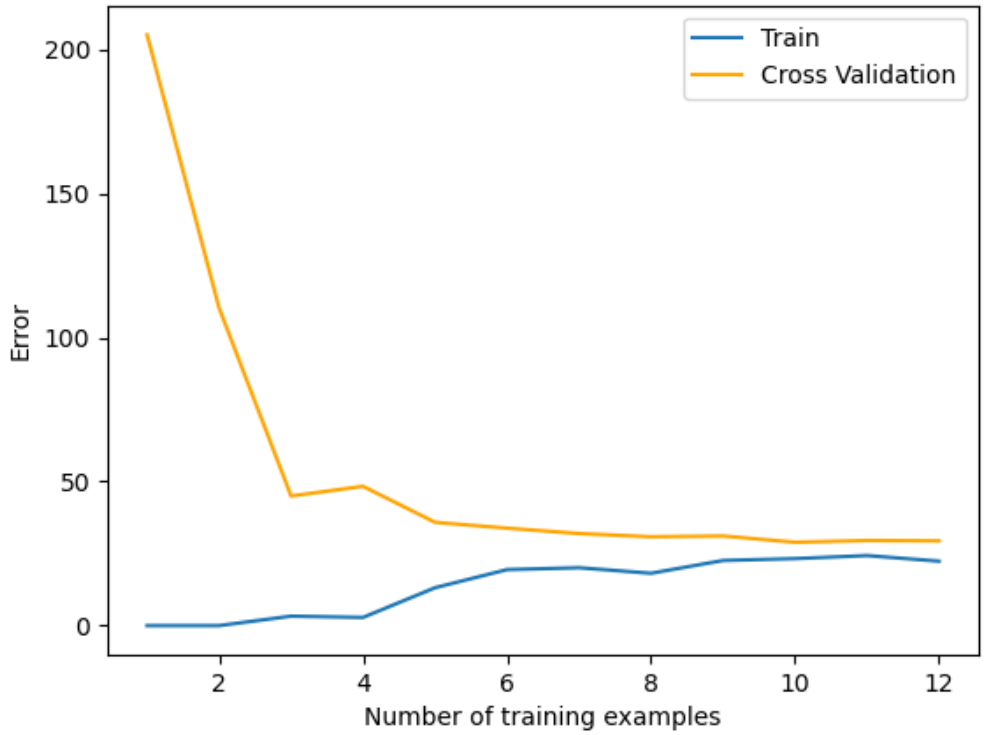
[ ]:
```
def apartado_1():
    print("Cost: " + str(cost(thetas, X, y, 1)))
    print("Gradient: " + str(gradient(thetas, X, y, 1)))

    reg = 0
    thetas_opt = opt.minimize(fun= cost, x0= thetas, args= (X, y, reg)).x

    plt.figure()
    plt.scatter(X[:,1], y, marker= "x", color="red")
    Y_pred = np.dot(X, thetas_opt)
    plt.plot(X[:,1], Y_pred)
    plt.xlabel('Change in water level (x)')
    plt.ylabel('Water flowing out of the dam (y)')
    plt.savefig("apartado1.png")
```



2- Curvas de aprendizaje

Mediante la función get_errors se ha repetido la operación del apartado anterior utilizando diferentes subconjuntos de los ejemplos de entrenamiento (de 1 a m, siendo los subconjuntos X[:i] e y[:i]), tras lo cual se han obtenido los errores para los distintos conjuntos de entrenamiento y validación. Después se han representado gráficamente:

[ ]:
```
def apartado_2():
    m = X.shape[0]
    reg = 0

    train_errors, val_errors = get_errors(X, y, X_val, y_val, reg)

    plt.figure()
    plt.plot(range(1, m+1), train_errors)
    plt.plot(range(1, m+1), val_errors, c='orange')
    plt.legend(("Train", "Cross Validation"))
    plt.xlabel("Number of training examples")
    plt.ylabel("Error")
    plt.savefig("apartado2.png")
```



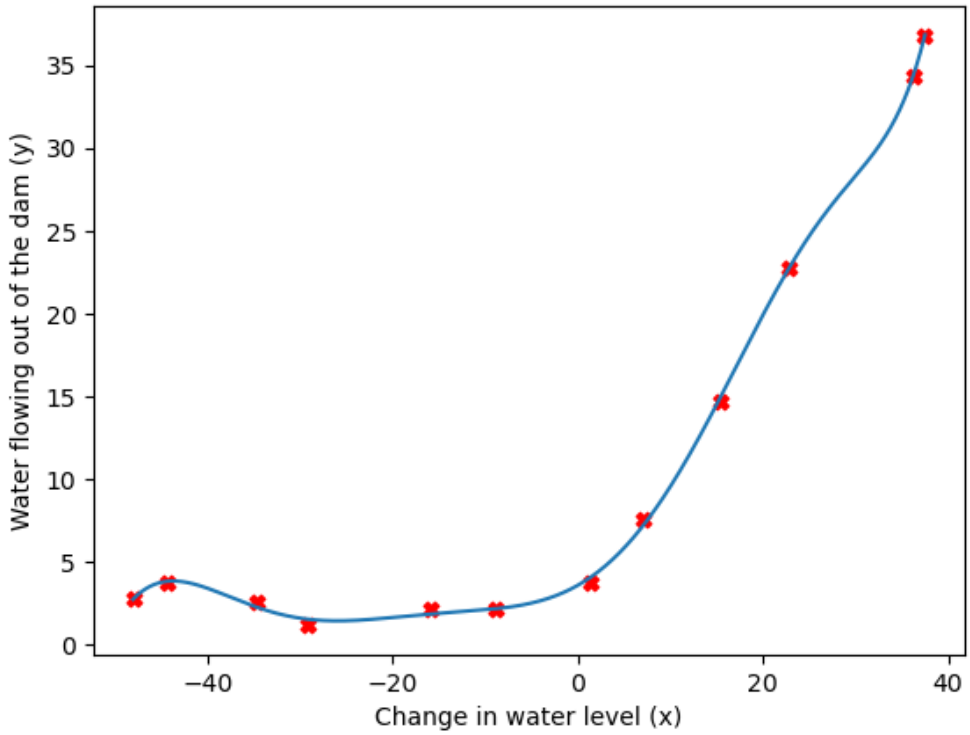
Como se puede ver, el error se aproxima a los conjuntos de entrenamiento cuantos más ejemplos hay, por lo que se puede concluir que el aprendizaje está sesgado.

3- Regresión Polinomial

Hemos comenzado implementando una función que genera datos adecuados de entrada a partir de una matriz X y un número p (el grado del polinomio sobre el que se va a trabajar). A partir de dichos datos hemos vuelto a aplicar el método de regresión lineal para obtener un vector θ que minimiza el error para un valor de $\lambda = 0$. Finalmente hemos representado la curva en un gráfico.

[ ]:
```
def apartado_3_1():
    p = 8
    reg = 0

    X_pol, mu, sigma = normalizar(polinomial(X_orig, p))
    X_pol = np.hstack([np.ones((X_pol.shape[0], 1)), X_pol])

    thetas = np.zeros(X_pol.shape[1])

    thetas_opt = opt.minimize(fun = cost, x0 = thetas, args = (X_pol, y, reg)).x
    plt.figure()

    X_test = np.arange(np.min(X),np.max(X),0.05)
```

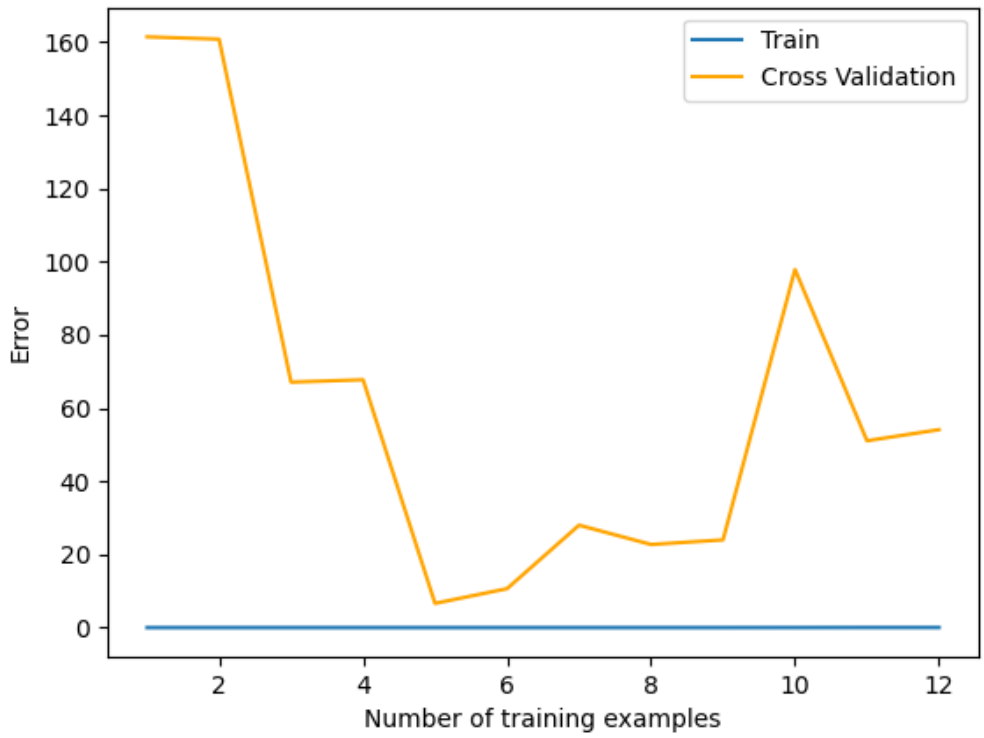
```
    X_test = polinomial(X_test,8)
    X_test = (X_test - mu) / sigma
    X_test =np.hstack([np.ones([X_test.shape[0],1]),X_test])
    Y_pred = np.dot(X_test, thetas_opt)
    plt.plot(np.arange(np.min(X),np.max(X),0.05),Y_pred)
    plt.scatter(X_orig,y,marker="X", color="red")
    plt.xlabel('Change in water level (x)')
    plt.ylabel('Water flowing out of the dam (y)')
    plt.savefig("apartado3_1.png")
```



A continuación hemos generado las curvas de aprendizaje para la hipótesis polinomial con distintos números de conjuntos de ejemplo cada vez mayores y hemos evaluado los errores junto con los de otro conjunto independiente.

[ ]:
```
def apartado_3_2():
    m = X.shape[0]
    reg = 0
    p = 8

    X_pol, mu, sigma = normalizar(polinomial(X_orig, p))
    X_pol = np.hstack([np.ones((X_pol.shape[0], 1)), X_pol])
```

```
    X_val_pol = ((polinomial(X_val_orig, p)) - mu) / sigma
    X_val_pol = np.hstack([np.ones((X_val_pol.shape[0], 1)), X_val_pol])

    train_errors, val_errors = get_errors(X_pol, y, X_val_pol, y_val, reg)

    plt.figure()
    plt.plot(range(1, m+1), train_errors)
    plt.plot(range(1, m+1), val_errors, c='orange')
    plt.legend(("Train", "Cross Validation"))
    plt.xlabel("Number of training examples")
    plt.ylabel("Error")
    plt.savefig("apartado3_2.png")
```

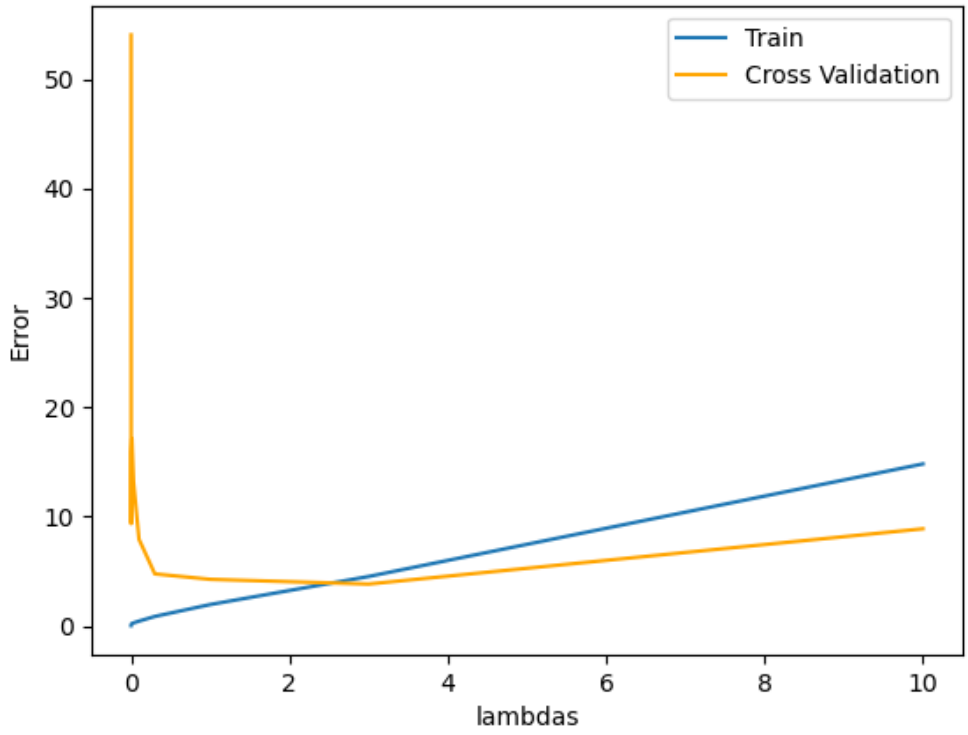


4- Selección del parámetro λ

Se ha vuelto a evaluar la hipótesis generada con los ejemplos de entrenamiento probando distintos valores de lambda { 0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10 }, y se ha dibujado una gráfica con los errores de los ejemplos de entrenamiento y los de validación:

[ ]:
```
def apartado_4():
    lambdas = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
    p = 8
    train_errors = []
```

```
    val_errors = []

    X_pol, mu, sigma = normalizar(polinomial(X_orig, p))
    X_pol = np.hstack([np.ones((X_pol.shape[0], 1)), X_pol])

    X_val_pol = ((polinomial(X_val_orig, p)) - mu) / sigma
    X_val_pol = np.hstack([np.ones((X_val_pol.shape[0], 1)), X_val_pol])

    thetas = np.zeros(p + 1)
    i = 0

    for l in lambdas:
        thetas_opt = opt.minimize(fun=cost, x0=thetas, args=(X_pol, y, l)).x
        train_errors.append(cost(thetas_opt, X_pol, y))
        val_errors.append(cost(thetas_opt, X_val_pol, y_val))
        i += 1

    plt.figure()
    plt.plot(lambdas, train_errors)
    plt.plot(lambdas, val_errors, c='orange')
    plt.legend(("Train", "Cross Validation"))
    plt.xlabel("lambdas")
    plt.ylabel("Error")
    plt.savefig("apartado_4.png")
    plt.show()
```



Por último se ha estimado el error de la hipótesis aplicándola a un nuevo vonjunto de datos con $\lambda = 3$

[ ]:
```
##estimación el error
reg = 3
X_test_pol = ((polinomial(X_test_orig, p)) - mu) / sigma
X_test_pol = np.c_[np.ones((len(X_test_pol), 1)), X_test_pol]
theta = np.zeros(p+1)

thetas_opt = opt.minimize(fun=cost, x0=theta, args=(X_pol, y, reg)).x

print('El error obtenido para lambda = {} es {}'.format(
    reg, cost(thetas_opt, X_test_pol, y_test)))
```

Se ha obtenido un error de aproximadamente 3,57