Práctica 5 – Regresión lineal regularizada: sesgo y varianza

Realizada por Mario Blanco Domínguez y Juan Tecedor Roa

- Objetivo de la práctica

  En esta práctica, comprobaremos los efectos del sesgo y la varianza. Para ello, aplicaremos regresión lineal lineal regularizada a un conjunto de datos, con lo cual no obtendremos un resultado, pero reajustaremos con un polinomio de grado superior.

- Código de la práctica

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


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


def cost(T, X, Y, l):
    m = X.shape[0]
    ret = 0
    for row in range(len(X)):
        H = hypothesis(X[row], T)
        ret += np.square(H - Y[row])

    ret = (1 / (2 * m)) * ret
    ret += (l / (2 * m)) * (np.sum(np.square(T[1:])))
    return ret


def gradient(X, Y, l, T):
    m = X.shape[0]
    D1 = np.zeros(T.shape)

    for row in range(len(X)):
        H = hypothesis(X[row], T)
        D1 += (H - Y[row]) * X[row]
```

```python
    D1 /= m
    # Regularizacion de todos menos j=0
    D1[1:] += (l / m * T[1:])

    return D1


def costAndGrad(X, Y, T, l):
    return cost(T, X, Y, l), gradient(X, Y, l, T)


def get_errors(X, Y, Xval, Yval, l):
    train_errors = []
    validation_errors = []
    m = X.shape[0]
    for i in range(1, m):
        T = np.zeros(X.shape[1])
        thetas = opt.minimize(fun=cost, x0=T, args=(X[0:i], Y[0:i], l)).x
        train_errors.append(cost(thetas, X[0:i], Y[0:i], l))
        validation_errors.append(cost(thetas, Xval, Yval, l))

    return (train_errors, validation_errors)


def polynomial(X, p):
    polynomial = X
    for i in range(1, p):
        polynomial = np.column_stack((polynomial, np.power(X, i+1)))
    return polynomial


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

    return (normalized, avg, standard_deviation)


def main():
    #1. Regresión lineal regularizada

    data = io.loadmat('./ex5data1.mat')
    X, Y = data['X'], data['y']
    Xval, Yval = data['Xval'], data['yval']
    Xtest, Ytest = data['Xtest'], data['ytest']
    plt.plot(X, Y, 'x')
    X_ones = np.hstack([np.ones([np.shape(X)[0], 1]), X])
    Xval_ones = np.hstack([np.ones([np.shape(Xval)[0], 1]),Xval])
```

```
    Xtest_ones = np.hstack([np.ones([np.shape(Xtest)[0], 1]),Xtest])
    l = 1
    T = np.array([1, 1])

    print(costAndGrad(X_ones, Y, T, l))

    l=0
    res = (opt.minimize(fun=cost, x0=T, args=(X_ones, Y, l))).x

    theta_0, theta_1 = res[0], res[1]
    x = np.linspace(min(X), max(X), 100)
    y = theta_0 + theta_1 * x
    plt.plot(x, y, label=('y = ' + str(theta_1) + 'x + ' + str(theta_0)))
    plt.suptitle('Result')
    plt.xlabel('Change in water level\'s')
    plt.ylabel('Water flowing out of the dam\'s')
    # plt.savefig('result.png')
    plt.show()

    #2. Curvas de aprendizaje
    l = 0
    train_errors, value_errors = get_errors(X_ones, Y, Xval_ones, Yval, l
)

    x = np.linspace(1, 11, 11)
    plt.plot(x, train_errors, label='Train')
    x = np.linspace(1, 11, 11)
    plt.plot(x, value_errors, label='CrossVal')
    plt.legend()
    plt.suptitle('Learning curve for linear regression (lambda =' +str(l)
 + ')')
    plt.xlabel('Number of training examples')
    plt.ylabel('Error')
    plt.show()
    #3. Regresión polinomial
    X_normalized, avg, standard_deviation = normalize(polynomial(X, 8))
    X_normalized_ones = np.hstack(
        [np.ones([X_normalized.shape[0], 1]), X_normalized])

    T = np.zeros(X_normalized_ones.shape[1])

    l=0
    res = (opt.minimize(fun=cost, x0=T, args=(X_normalized_ones, Y, l))).
x

    X_points = np.arange(np.min(X),np.max(X),0.05)
    X_points_polynomial = polynomial(X_points,8)
    X_points_polynomial_normal = (X_points_polynomial - avg) / standard_d
eviation
```

```python
    X_points_polynomial_normal_ones = np.hstack([np.ones([X_points_polynomial_normal.shape[0],1]),X_points_polynomial_normal])
    Y_pred = np.matmul(X_points_polynomial_normal_ones , res)
    plt.plot(X_points,Y_pred)
    plt.scatter(X,Y,marker="X", color="red")
    plt.suptitle('Result (lambda =' +str(l) + ')')
    plt.xlabel('Change in water level\'s')
    plt.ylabel('Water flowing out of the dam\'s')

    plt.show()
    print(res)

    l = 0
    Xval_normalized, avg, standard_deviation = normalize(polynomial(Xval, 8))
    Xval_normalized_ones = np.hstack([np.ones([Xval_normalized.shape[0], 1]), Xval_normalized])
    train_errors, value_errors = get_errors(X_normalized_ones, Y, Xval_normalized_ones, Yval, l)

    x = np.linspace(1, 11, 11)
    plt.plot(x, train_errors, label='Train')
    x = np.linspace(1, 11, 11)
    plt.plot(x, value_errors, label='CrossVal')
    plt.legend()
    plt.suptitle('Learning curve for linear regression (lambda =' +str(l) + ')')
    plt.xlabel('Number of training examples')
    plt.ylabel('Error')
    plt.show()

    l = 1
    train_errors, value_errors = get_errors(X_normalized_ones, Y, Xval_normalized_ones, Yval, l)

    x = np.linspace(1, 11, 11)
    plt.plot(x, train_errors, label='Train')
    x = np.linspace(1, 11, 11)
    plt.plot(x, value_errors, label='CrossVal')
    plt.legend()
    plt.suptitle('Learning curve for linear regression (lambda =' +str(l) + ')')
    plt.xlabel('Number of training examples')
    plt.ylabel('Error')
    plt.show()

    l=100
    train_errors, value_errors = get_errors(X_normalized_ones, Y, Xval_normalized_ones, Yval, l)
```

```
    x = np.linspace(1, 11, 11)
    plt.plot(x, train_errors, label='Train')
    x = np.linspace(1, 11, 11)
    plt.plot(x, value_errors, label='CrossVal')
    plt.legend()
    plt.suptitle('Learning curve for linear regression (lambda =' +str(l)
 + ')')
    plt.xlabel('Number of training examples')
    plt.ylabel('Error')
    plt.show()

    #4. Selección del parámetro λ

    lambdas =[0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
    train_errors = []
    validation_errors = []

    for i in lambdas:
        initial_thetas = np.zeros(X_normalized_ones.shape[1])
        T = opt.minimize(fun=cost, x0=initial_thetas, args=(X_normalized_
ones,Y,i)).x
        train_errors.append(cost(T,X_normalized_ones,Y,0))
        validation_errors.append(cost(T,Xval_normalized_ones,Yval,0))

    plt.plot(lambdas,train_errors, label='Train')
    plt.plot(lambdas,validation_errors, label='CrossVal')
    plt.suptitle('Selecting lambda using a cross validation set')
    plt.xlabel('lambda')
    plt.ylabel('Error')
    plt.legend()

    l=3


    Xtest_polynomial = polynomial(Xtest,8)

    Xtest_polynomial_normal = (Xtest_polynomial - avg) / standard_deviati
on
    Xtest_polynomial_normal_ones = np.hstack([np.ones([Xtest_polynomial_n
ormal.shape[0],1]),Xtest_polynomial_normal])

    initial_thetas = np.zeros(Xtest_polynomial_normal_ones.shape[1])
    T = opt.minimize(fun=cost, x0=initial_thetas, args=(Xtest_polynomial_
normal_ones,Ytest,l)).x

    errors_test=cost(T,Xtest_polynomial_normal_ones,Ytest,l)
    print(errors_test)
```

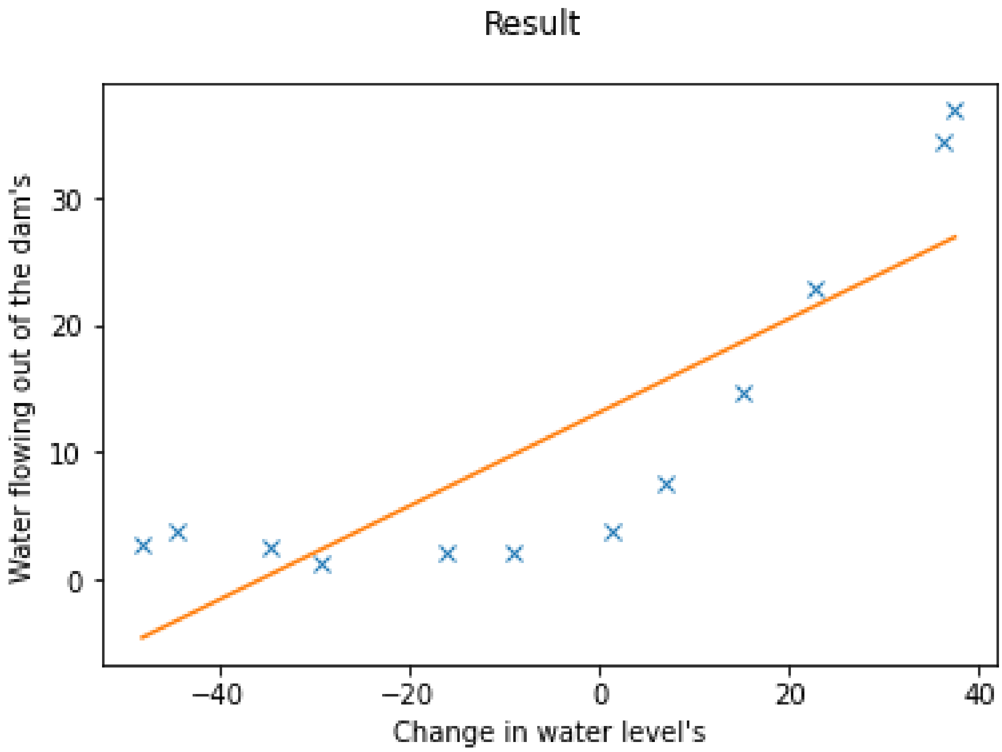
```
main()
```

- Resultados de ejecución

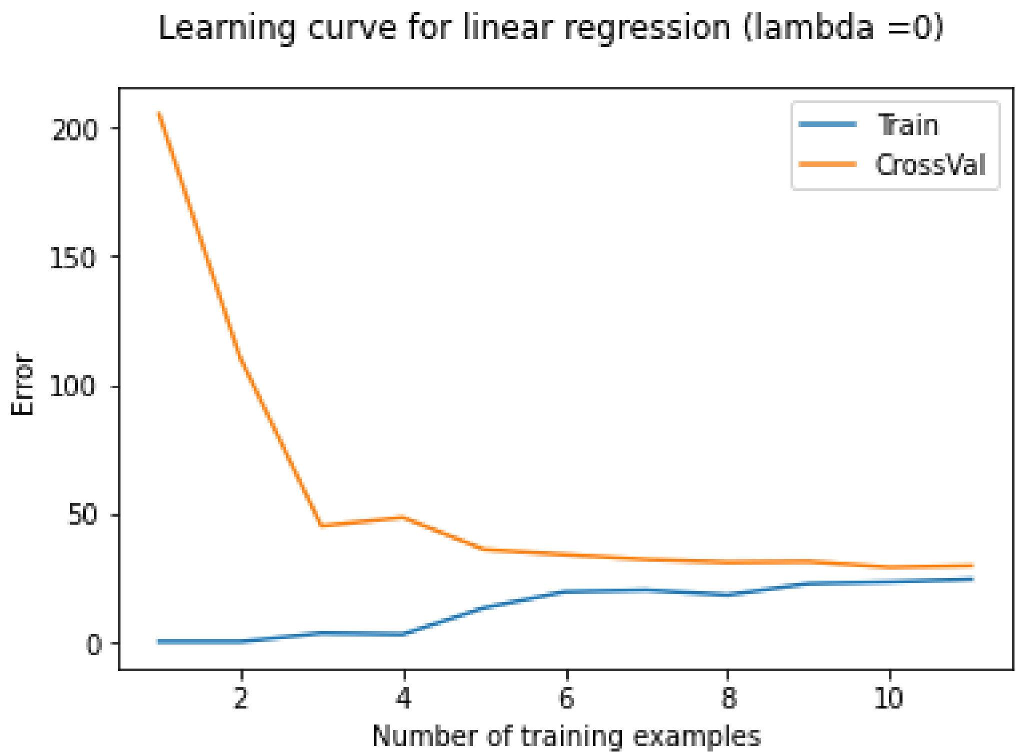
Tras leer los datos, probamos la función que hemos creado que calcula el coste y el gradiente, obteniendo el siguiente resultado para lambda =1 y theta inicial = [1;1]

```
(array([303.99319222]), array([-15.30301567, 598.25074417]))
```

A continuación, usaremos la función scipy.optimize.minimize para encontrar los theta óptimos. Obtenemos la siguiente recta usando regresión lineal ajustada a los datos, y podemos comprobar que, efectivamente, no ajusta bien. La hipótesis elegida, por ello, no es la correcta.

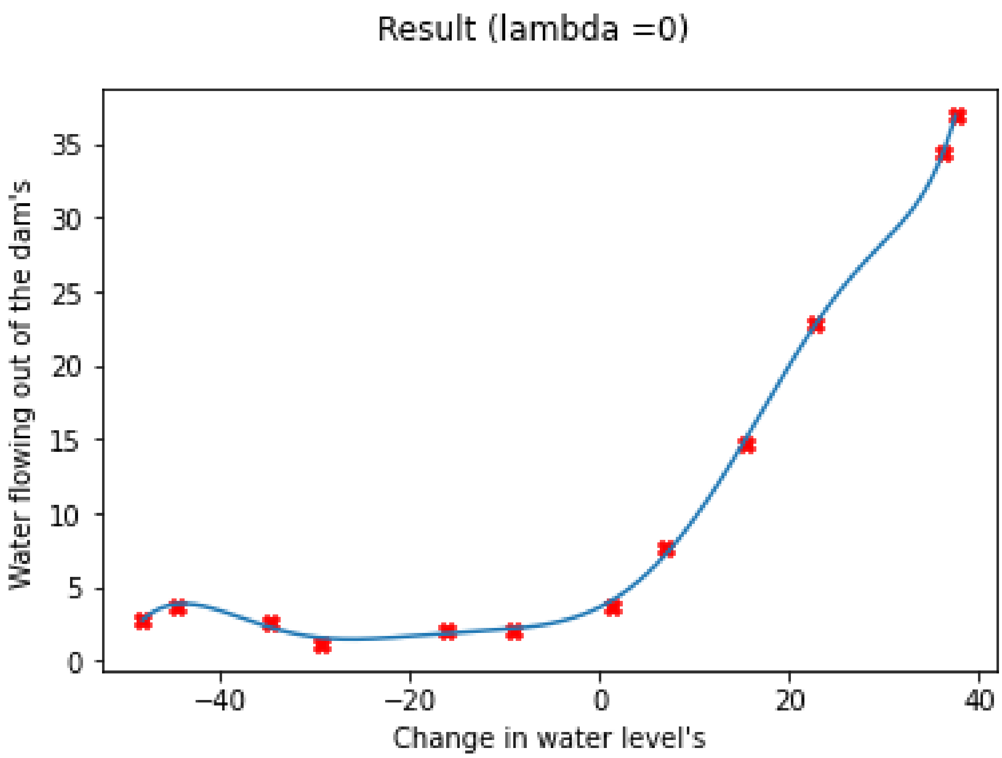


Ahora usaremos las curvas de aprendizaje con el mismo ejemplo de entrenamiento para identificar sub-ajustes y sobre-ajustes. Tras hacer el entrenamiento por regresión lineal usando partes ascendentes del conjunto de datos, obtenemos el error sobre los conjuntos de entrenamiento y el error sobre los de validación.

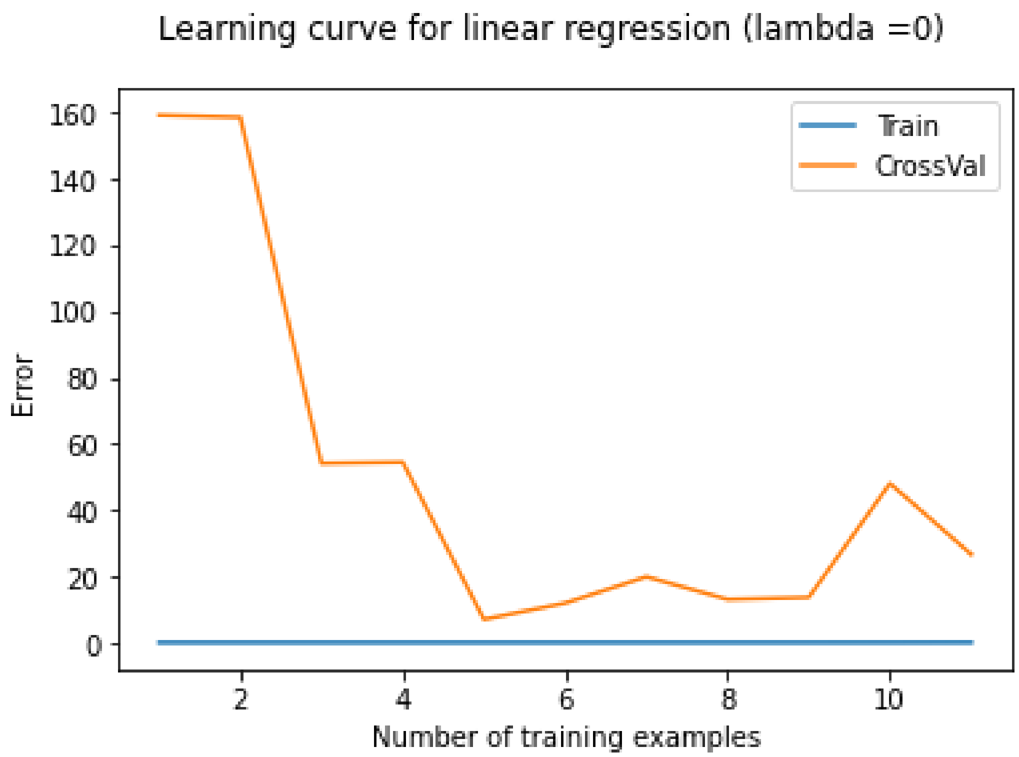


Como al aumentar los ejemplos de entrenamiento obtenemos un aumento del error y las curvas se aproximan, obtenemos la conclusión de que necesitamos un polinomio de mayor grado para ajustarnos mejor.

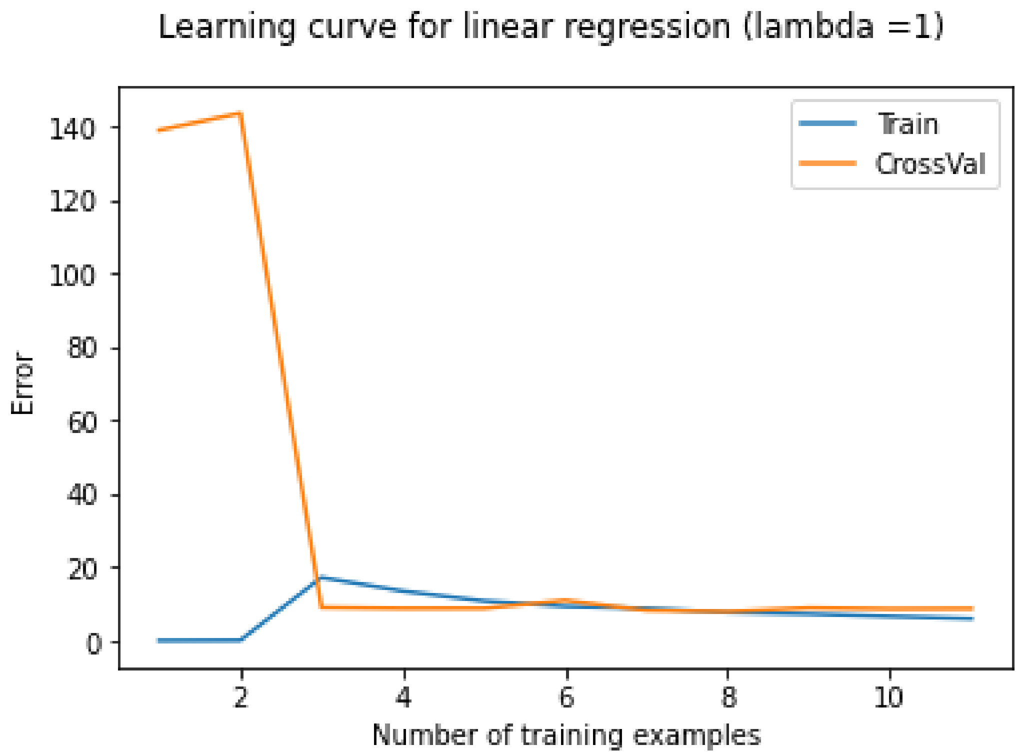
Por ello, usamos como hipótesis una función polinómica, es decir, usar más atributos. En este caso será hasta grado 8. La curva obtenida de los theta óptimos es la siguiente:

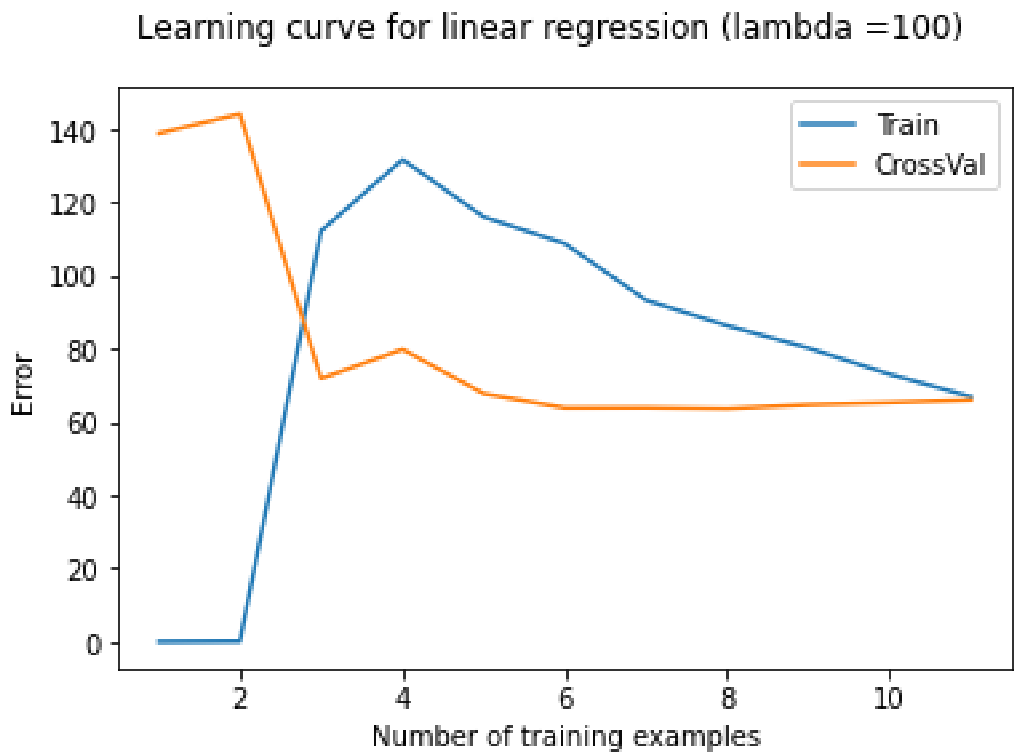


Si volvemos a obtener las curvas de aprendizaje, podemos ver que hay un sobre-ajuste, ya que los ejemplos de entrenamiento han dado un resultado de 0. Esta hipótesis tampoco sería valida, no generaliza bien.

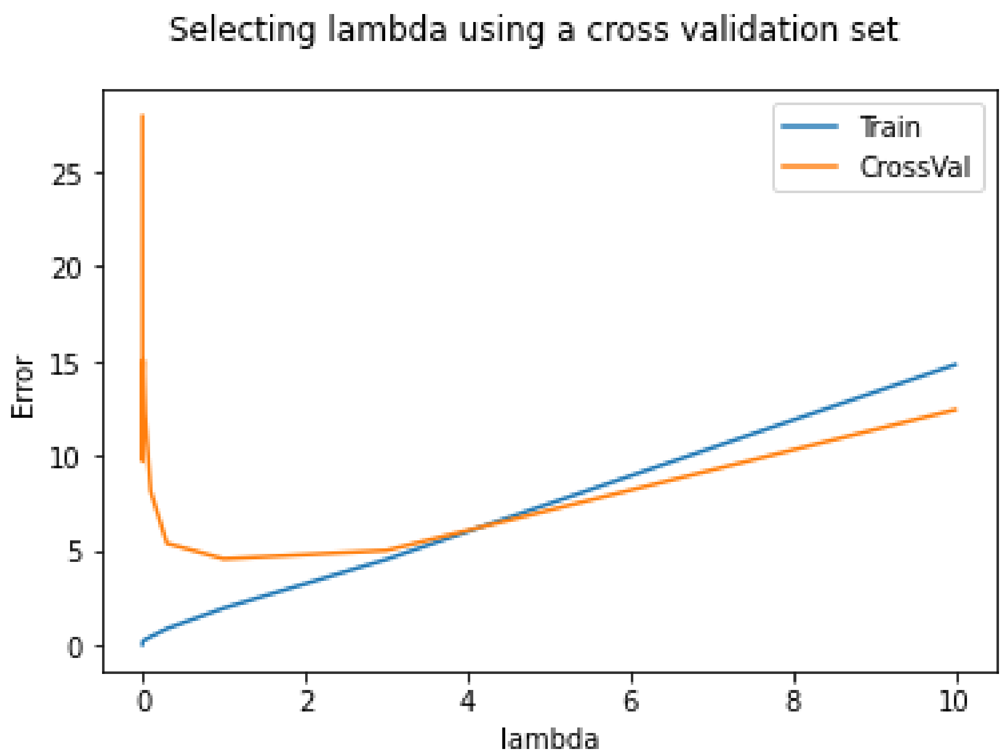


Ajustando el parámetro de regularización lambda, a 1 y a 100, obtenemos otros resultados.





No somos capaces de distinguir cual sería mejor valor de lambda. Por ello, se prueba en la última parte el error de evaluar los ejemplos con un conjunto de varios valores lambda.



Resulta que el mejor valor es aproximadamente 3, ya que es aproximadamente por donde cruzan las curvas de entrenamiento y validación.

- Conclusiones

Pensamos que esta práctica es útil, ya que cuando nos enfrentemos a un conjunto con pocos datos y la hipótesis lineal no sea suficiente, tendremos técnicas para tratar de evitar el sobre-ajuste y el sub-ajuste.