Universidad Complutense de Madrid
Facultad de Informática
Aprendizaje Automático y Big Data

Memoria Práctica 5:
Regresión Lineal Regularizada:
Sesgo y varianza.

Profesor:
- Alberto Díaz Esteban.

Alumnos:
- Marina de la Cruz López.
- Diego Alejandro Rodríguez Pereira.

Introducción.

El objetivo en esta práctica es utilizar un algoritmo de regresión lineal para comprobar cómo se muestra el sesgo y la varianza en nuestros algoritmos y como podemos ver si un algoritmo se está o no sobre ajustando a los datos de entrenamiento o si la hipótesis está sesgada desde un inicio.

Los datos utilizados en esta práctica son un conjunto de datos históricos del agua derramada por una presa en base al nivel del agua.

Parte 1: Regresión lineal regularizada

En primer lugar hemos implementado para regresión lineal regularizada la función de coste y gradiente de forma vectorizada. Ambas funciones toman como datos de entrada la matriz X con la entrada del algoritmo, el vector de salida Y y la lambda que se usa para regularizar. Como estamos en regresión lineal la función h será la multiplicación matricial de los datos de entrada y las thetas.

Función de coste:

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

Figura 1

Función de gradiente:

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

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

Figura 2

Hemos comprobado que estas funciones están bien implementadas, ya que para los valores $\lambda = 1$ y $\theta = [1; 1]$ obtenemos un coste de **303,993** y un gradiente de **[−15,303; 598,250]** que coinciden con los proporcionados por el profesor en el enunciado.

El resultado de ambas funciones se juntan en una tupla, la cual se pasa como parámetro a la función de spicy: *scipy.optimize.minimize*m, la cual nos realiza la optimización de las Thetas que minimiza el error sobre los ejemplos de entrenamiento.

Como los datos de entrada eran una matriz de una sola columna, y hemos usado esas dimensiones para la Thetas, lo que obtenemos es un vector de Thetas de dos elementos que nos dibujan una recta. Si la mostramos por pantalla podemos ver claramente que no se está ajustando nada bien a los datos porque una representación lineal no es adecuada para este problema lo que ocasiona una predicción sesgada de los valores.

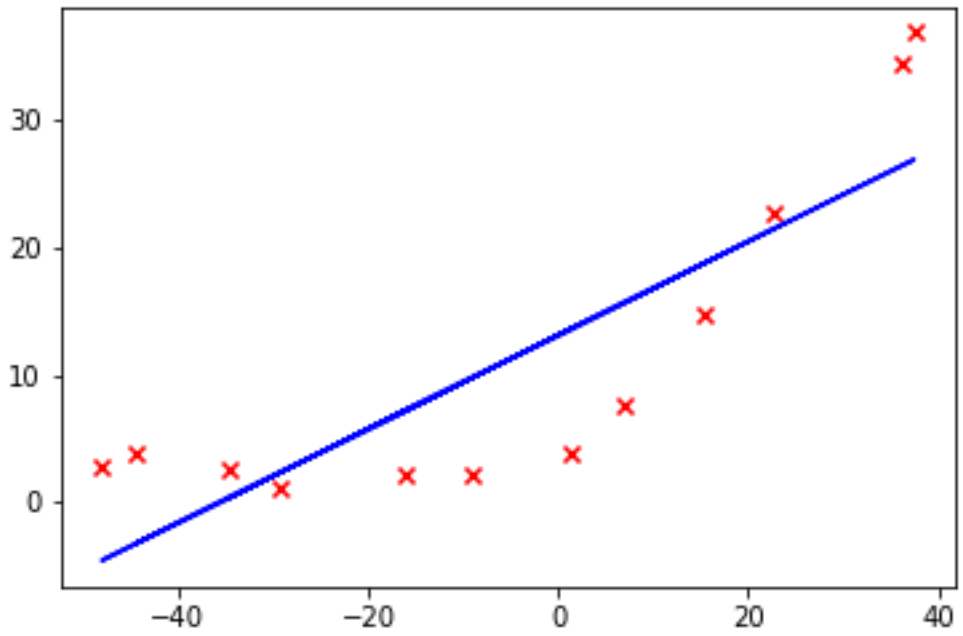

Figura 3

Parte 2: Curvas de aprendizaje

En este apartado hemos comprobado que efectivamente el problema con el algoritmo es porque la hipótesis está sesgada y que no es por ningún otro motivo. Para ello hemos pintado las curvas de aprendizaje para los datos de entrenamiento (los que estábamos usando en el apartado anterior) y unos datos de validación nuevos que el algoritmo no ha visto nunca.

Las curvas de aprendizaje nos permiten identificar situaciones de sesgo y de varianzas, que a pesar de que para este problema es fácil de ver, ya que se trata de un problema con una sola variable y que por tanto se puede pintar en una gráfica de dos dimensiones (X e Y), no siempre es el caso. Por tanto, para problemas con mas atributos, es imposible obtener una gráfica tan sencilla y evidente como la obtenida en el apartado anterior, por lo que en estos casos se utiliza las curvas de aprendizaje.

Por un lado, para calcular el error cuadrático medio y saber el error que tiene el conjunto de datos frente al modelo, hemos utilizado la parte de la función de coste sin la regularización que ya lo calcula. Luego, hemos programado otra función que llama a la optimización en los datos de entrenamiento y validación para distintos subsets de datos, desde

i= 0 a i = m siendo m la cantidad total de datos que tenemos. Los errores para cada cantidad de elementos los guardamos en una lista y luego los mostramos por pantalla. El error se calcula utilizando la siente expresión de hipótesis, la cual se realiza sobre el conjunto de ejemplos:

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

Figura 4

En el gráfico obtenido se observa que las líneas del error para el entrenamiento y la validación se juntan a más datos se introducen, esto significa que no estamos mejorando el coste en el entrenamiento frente a la validación lo cual indica que la hipótesis que estamos usando no es la indicada. Lo que confirma que el aprendizaje del algoritmo está sesgado por los valores, por lo que es necesario calcular una hipótesis mas compleja con la que se puedan ajustar mejor los ejemplos de entrenamiento.

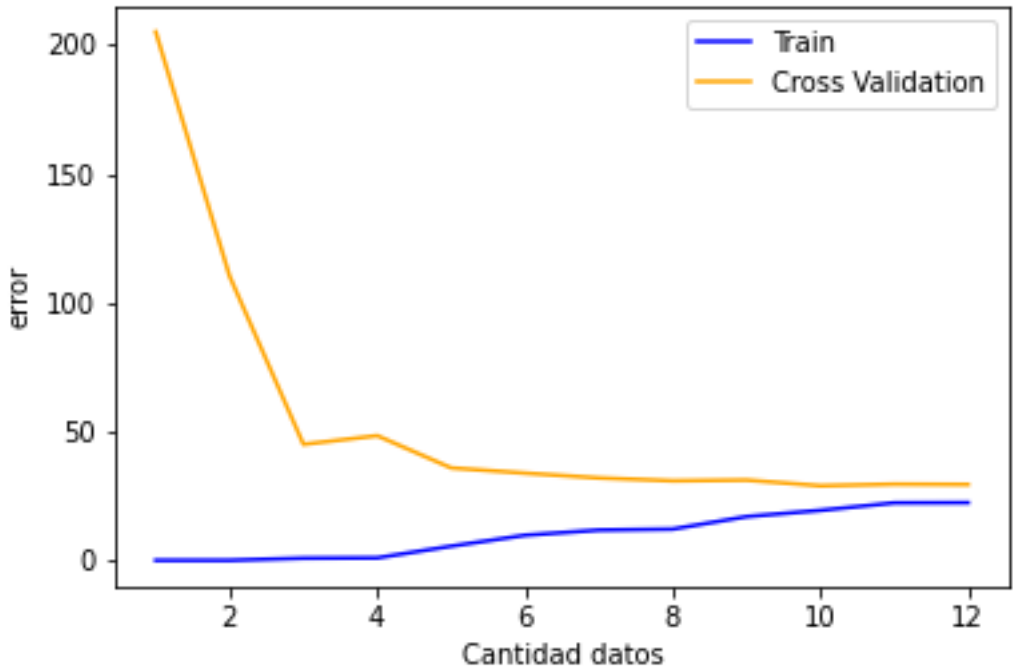

Figura 5

Parte 3: Regresión polinomial

Hemos cambiado la hipótesis para que use polinomios de grado 8, para implementar la nueva hipótesis primero hemos creado una función que transforma nuestra matriz de entrada de una sola columna en una matriz con 8 columnas, cada una es la primera elevada a un grado. Debido a la generación de nuevos atributos a través de la fórmula siguiente es que se crean rangos de valores bastantes significativos entre si. Por lo que debemos normalizar los atributos antes de ejecutar el algoritmo para el aprendizaje de las thetas.
Una vez obtenida la matriz la vamos a normalizar para que todas las columnas se encuentren en el mismo rango y sea más fácil optimizar las Thetas.

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 * (\text{nivelAgua}) + \theta_2 * (\text{nivelAgua})^2 + \ldots + \theta_p * (\text{nivelAgua})^p \\ &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p \end{aligned}$$

Volvemos a ejecutar el algoritmo de optimización con lambda = 0. Lo que nos devuelve el array de Thetas optimizado, si mostramos la función en forma de gráfico obtenemos:

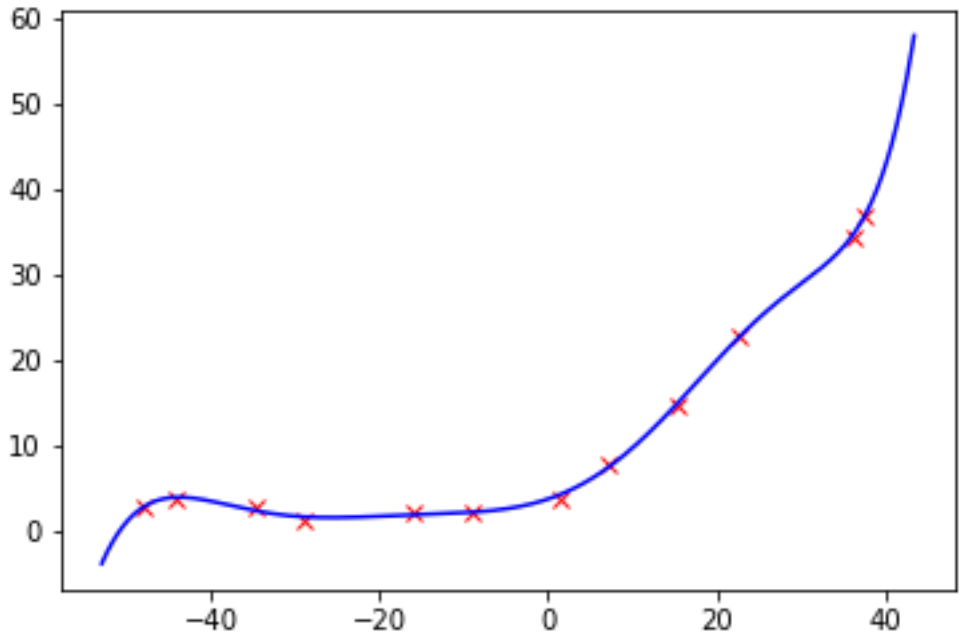

Figura 6

Con este método vamos a observar cómo se comporta la nueva hipótesis con los datos de validación que hemos cargado. Para ello tendremos que generar las potencias y normalizarlas para que dadas las Thetas podamos calcular la función correspondiente.

Las curvas de aprendizaje las vamos a generar de la misma forma que en el caso anterior, calculamos el error usando la misma fórmula para training y validación que en el

apartado anterior de las Thetas que nos devuelve usar de i = 1 a i = m ejemplos (siendo m el total de ejemplos de entrenamiento y validación).

La gráfica obtenida es:

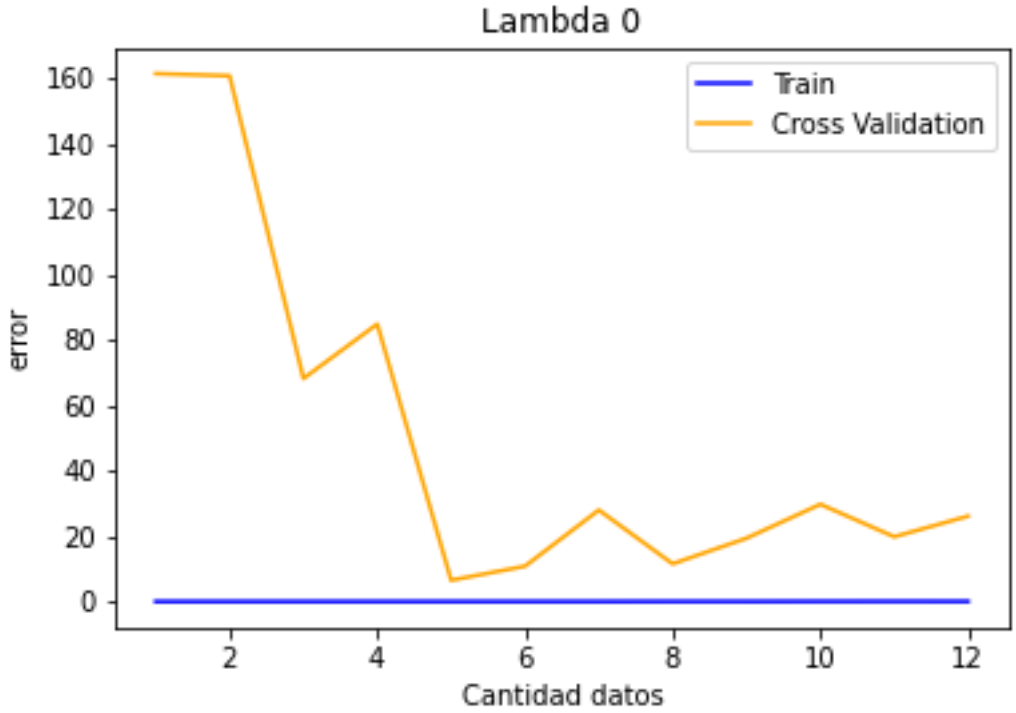

Figura 7

Podemos observar que se está produciendo un sobreajuste de la función ya que el error en el entrenamiento en prácticamente 0 pero el error en la validación sube conforme añadimos más ejemplos de entrenamiento. Esto es algo esperable porque estamos haciendo la optimización con lambda = 0, es decir que no regularizamos.

Si repetimos el proceso para lambda= 1 y lambda=100 obtenemos:

Lambda= 1

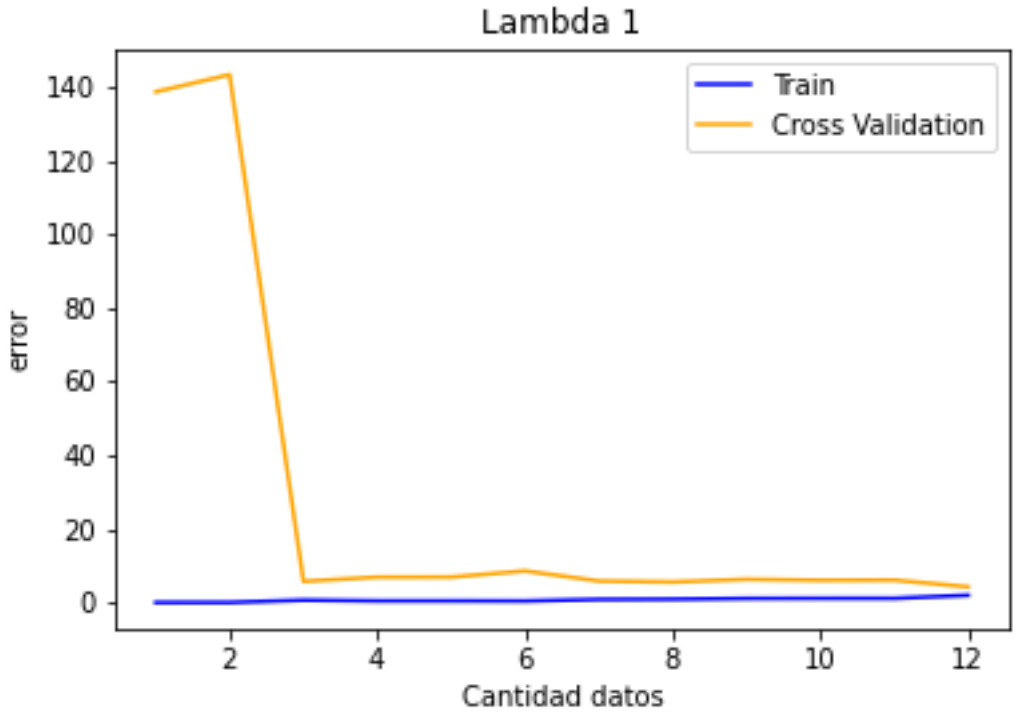

Figura 8

Podemos ver que con lambda = 1 los resultados mejoran mucho en comparación con lambda = 0 ya que al aumentar la cantidad de datos el error en la validación no aumenta como sí ocurría en el otro caso.

Lambda = 100

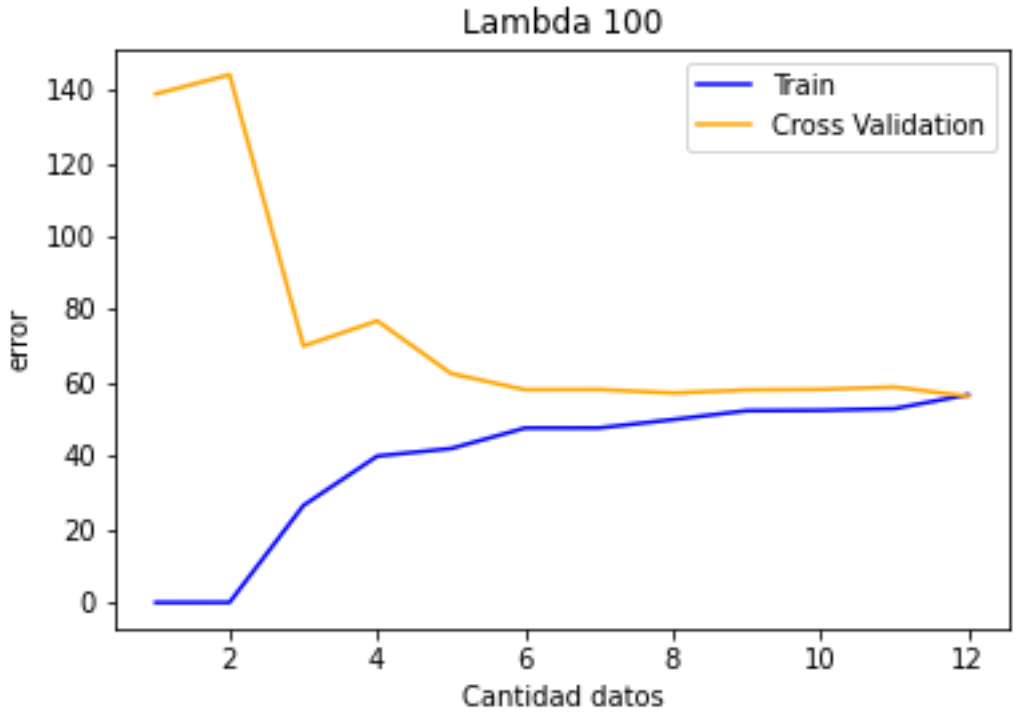

Figura 9

En el caso de lambda = 100 vemos que el error en los datos de entrenamiento sube conforme aumentan los datos lo que significa que tenemos un bias en la hipótesis porque la lambda es demasiado grande y regulariza de más.

Parte 4: Selección del parámetro λ

La elección de la lambda es muy importante para evitar un subajuste o sobreajuste en los modelos obtenidos y que a su vez minimice el error.

En nuestro caso vamos a probar con varios valores de lambda para elegir cual es el mejor valor. Con todos los datos de entrenamiento y validación ejecutamos el algoritmo de optimización sobre el conjunto de entrenamiento y calculamos el error cuadrático medio para el entrenamiento y la validación para lambda = { 0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10 }, metemos los errores en una lista y los mostramos en forma de gráfica:

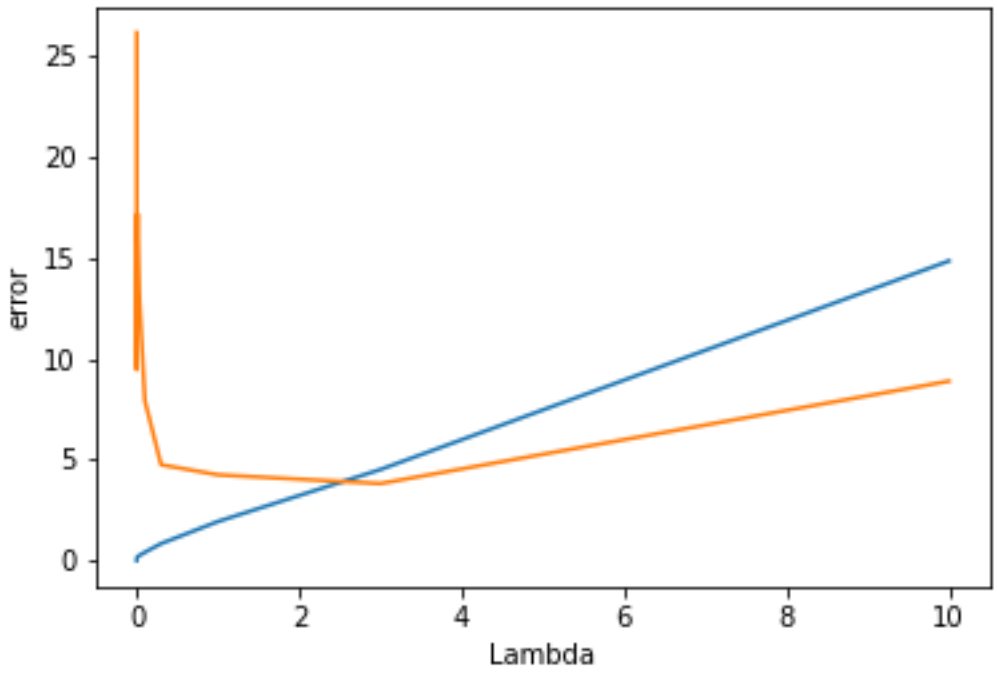

Figura 10

De la gráfica vemos que el mejor valor de lambda de la lista es lambda = 3 porque el error en el conjunto de validación es el más bajo que para el resto de valores de lambda.

Si probamos valores intermedios para ver qué está pasando entre lambda= 3 y lambda = 5.

Lambdas= {2.5, 3,3.5, 4, 4.5 ,5}

Obtenemos que efectivamente los mejores valores son lambda= 3 o lambda = 3.5.

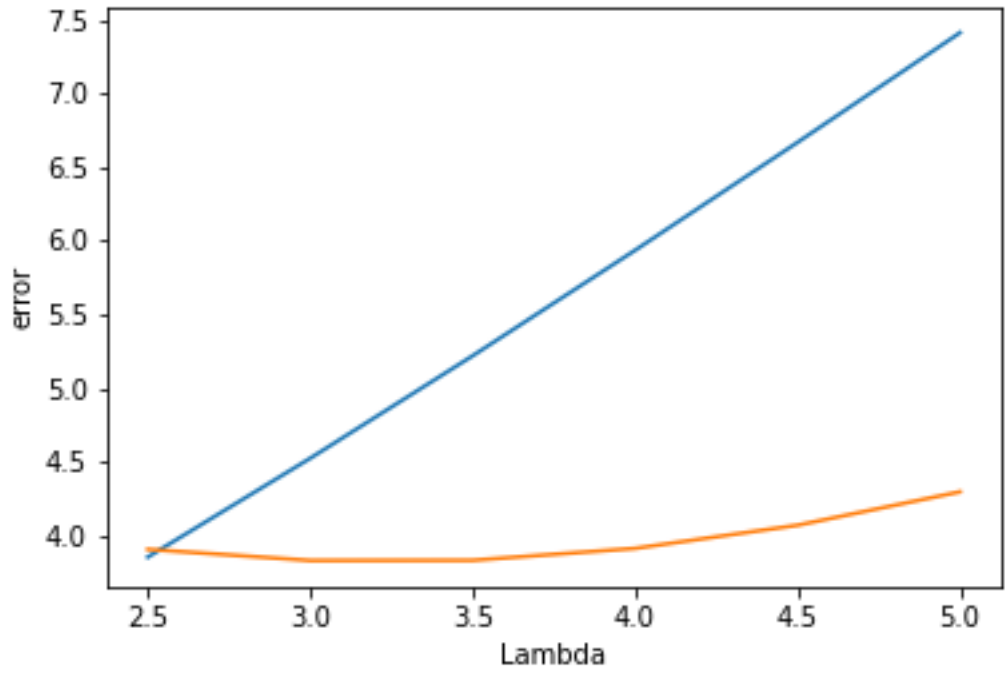

Figura 11

Podemos mostrar cómo se comporta este valor de lambda cuando añadimos ejemplos de entrenamiento.

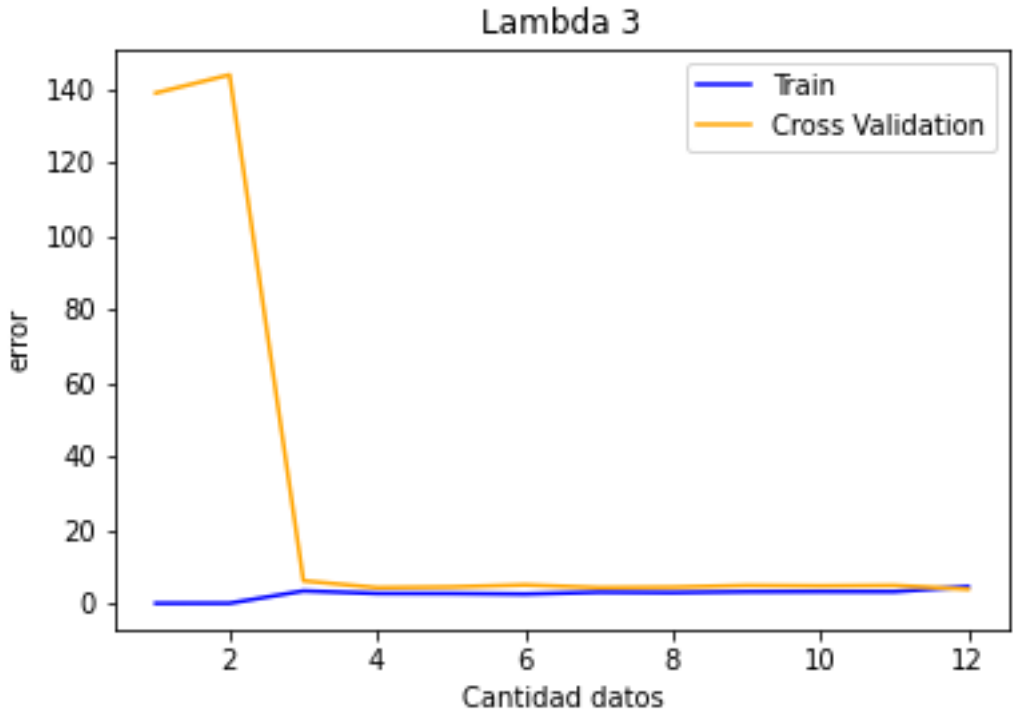

Figura 12

Una vez hemos optimizado los parámetros vamos a volver a probar el modelo en un conjunto de datos de test: "Xtest" y "ytest". Hemos realizado las mismas transformaciones que antes (transformación polinómica y normalización para la X, poner en forma de lista para la Y) y hemos calculado el error obtenido en el test con el modelo de lambda=3. El error es **3.5720374973494597** lo cual concuerda con lo que esperábamos.

Código

```
#Imports
import numpy as np
from scipy.io import loadmat
import scipy.optimize as opt
import matplotlib.pyplot as plt
from scipy.optimize import minimize
```

```
#Funcion de la Hipotesis
def h_vec(x, theta):
    return np.dot(x, theta)
```

```
# Funcion de coste
def cost_funct(Theta, X, y, Lambda):
    m = X.shape[0]
    J = cost(Theta, X, y ,m)

    reg_term = np.sum(Theta[1:]**2)
    reg_term = (reg_term*Lambda)/(2*m)
    J = J + reg_term

    return J
```

```
#Función que calcula el error cuadrático medio, se usa en el coste y en las prueba
de training y validación
def cost(Theta, X, y, m):
    h = h_vec(X, Theta)
    J = np.sum((h - y)**2)
    J = J/(2*m)
    return J
```

```
#Función de gradiente regularizada
def gradient_funct(Theta, X, y, Lambda):

    m = np.shape(X)[0]
    h = h_vec(X, Theta)
    grad = np.dot(X.T, h - y)
    grad = grad/m
    reg_term = (Lambda/m)*np.insert(Theta[1:],0,0)
    grad = grad + reg_term
```

```
    return grad
```

```
#coste y gradiente que se le pasa al optimize
def cost_and_gradient(Theta, X, y, Lambda):
    return cost_funct(Theta, X, y, Lambda), gradient_funct(Theta, X, y, Lambda)
```

Apartado 1. Regresión Lineal Regularizada

```
def draw_graph(Theta, X_ones, X, y):
    plt.scatter(X, y, marker = 'x', c = 'red', label="Entry")
    y = h_vec(X_ones, Theta)
    plt.plot(X, y, c='blue')
    plt.savefig("Apartado1.png")
    plt.legend()

    plt.show()
```

```
def main_parte1():
    #Cargamos los datos del fichero "ex5data1.mat'
    datafile = 'ex5data1.mat'
    mat = loadmat(datafile)
    X = mat.get("X")
    y = mat.get("y")
    #Ponemos la y en forma de fila
    y = y[:, -1]

    Lambda = 1
    X_ones = np.hstack((np.ones(shape=(X.shape[0], 1)), X))

    Theta = np.ones(X_ones.shape[1])

    cost = cost_funct(Theta, X_ones, y, Lambda)
    grad = gradient_funct(Theta, X_ones, y, Lambda)
    m = np.shape(X)[0]

    print("cost:", cost)
    print("grad:", grad)

    #result = opt.fmin_tnc(func=cost_funct, x0=Theta, fprime=gradient_funct,
args=(X, y, Lambda))
    result = opt.minimize(cost_and_gradient, Theta, args=(X_ones, y, Lambda),
jac=True, method='TNC')
    print(result)
    #ThetaOpt = result[0]
    ThetaOpt = result.x

    draw_graph(ThetaOpt, X_ones,X, y)
```

```
main_parte1()
```

Apartado 2. Curvas de aprendizaje

```
def learning_curve(X, y, Lambda, Theta, Xval, yval):

    m1 = X.shape[0]
    m2 = Xval.shape[0]
    #Inicializamos la lista de los errores a 0
    err1 = np.zeros(m1)
    err2 = np.zeros(m2)

    for i in range(1, X.shape[0] + 1):
        result = opt.minimize(cost_and_gradient, Theta, args=(X[0:i], y[0:i],
Lambda), jac=True, method='TNC')
        ThetasOpt = result.x

        #Añadimos el error para X e y de entrenamiento
        err1[i - 1] = cost(ThetasOpt, X[0:i], y[0:i], m1)
        #Añadimos el error para X e y del crossvalidation
        err2[i - 1] = cost(ThetasOpt, Xval, yval, m2)

    return err1, err2
```

```
def draw_learning_curve(err1, err2, title):
    l = np.arange(len(err1))
    l = l+1
    b = err1
    plt.title(title)
    plt.xlabel("Cantidad datos")
    plt.ylabel("error")
    plt.plot(l, b, c="blue", label="Train")

    d = err2[0:len(err1)]
    plt.plot(l, d, c="orange", label="Cross Validation")
    plt.legend()
```

```
def main_part2():
    #Cargamos los datos del fichero "ex5data1.mat'
    datafile = 'ex5data1.mat'
    mat = loadmat(datafile)
    X = mat.get("X")
    y = mat.get("y")
    y = y[:, -1]
    Xval = mat.get("Xval")
    yval = mat.get("yval")
    yval = yval[:, -1]

    print(X.shape, y.shape)
```

```
    print(Xval.shape, yval.shape)

    print(np.mean(X), np.std(X))
    print(np.mean(y), np.std(y))

    Lambda = 0
    X_ones = np.hstack((np.ones(shape=(X.shape[0], 1)), X))
    Xval_ones = np.hstack((np.ones(shape=(Xval.shape[0],1)),Xval))
    Theta = np.zeros(X_ones.shape[1])

    cost = cost_funct(Theta, X_ones, y, Lambda)
    grad = gradient_funct(Theta, X_ones, y, Lambda)

    print("cost:", cost)
    print("grad:", grad)

    res_error = opt.minimize(cost_and_gradient, Theta, args=(X_ones, y, Lambda),
jac=True, method='TNC')
    print(res_error)
    ThetaOpt = res_error.x

    err1, err2 = learning_curve(X_ones, y, Lambda, Theta, Xval_ones, yval)
    draw_learning_curve(err1, err2,"")

    plt.savefig("Apartado2.png")
    plt.show()
```

```
main_part2()
```

Apartado 3. Regresión polinomial

```
def generate_new_training_data(X, p):
    newX = X
    for i in range(2, p + 1):
        #newX = np.hstack([newX, X ** i])
        newX = np.column_stack([newX, X ** i])
    return newX
```

```
def normalize_attributes(X, mean, std_des):
    X_norm = X - mean
    X_norm = X_norm / std_des
    return X_norm
```

```
def draw_points(X, y, p, mean, std_des, result):
    # Pintamos grafica
    plt.figure()
    plt.plot(X, y, "x", color='red')
    lineX = np.arange(np.min(X) - 5,np.max(X) + 6,0.05)
    aux_x = (generate_new_training_data(lineX, p) - mean) / std_des
```

```
    lineY = np.hstack([np.ones([len(aux_x),1]),aux_x]).dot(result.x)
    plt.plot(lineX, lineY, '-', c = 'blue')
    plt.savefig("Apartado3.1.png")
    plt.show()

    plt.close()
    #draw_graph(ThetaOpt, X, y, result, newX)
```

```
def main_parte3():

    #Cargamos los datos del fichero "ex5data1.mat'
    datafile = 'ex5data1.mat'
    mat = loadmat(datafile)

    #Cargamos los datos de entranamiento y validación para la X y la y
    X = mat.get("X")
    y = mat.get("y")
    y = y[:, -1]
    Xval = mat.get("Xval")
    yval = mat.get("yval")
    yval = yval[:, -1]

    print(X.shape, y.shape)
    print(Xval.shape, yval.shape)

    print(np.mean(X), np.std(X))
    print(np.mean(y), np.std(y))

    Lambda = 0
    #Grado del polinomio
    p = 8

    #Se genera nuevos datos de entrenamiento a partir de los datos originales X
    newX = generate_new_training_data(X, p)
    #Media
    mean = np.mean(newX, axis = 0)
    #Desviacion Estandar
    std_des = np.std(newX, axis = 0)
    #Se normalizan los atributos
    X_norm = normalize_attributes(newX, mean, std_des)
    #Se añade columna de 1s
    newX = np.hstack([np.ones([X_norm.shape[0], 1]), X_norm])
    #Tambien se puede escribir: newX = np.insert(X_norm, 0, 1, axis=1)
    #print("newX:", newX)

    Theta = np.zeros(newX.shape[1])


    #result = opt.fmin_tnc(func=cost_funct, x0=Theta, fprime=gradient_funct,
args=(newX, y, Lambda))
```

```
    result = opt.minimize(cost_and_gradient, Theta, args=(newX, y, Lambda),
jac=True, method='TNC')

    draw_points(X, y, p, mean, std_des, result)


    #Apartado 3.2

    #Se genera nuevos datos de VALIDACION a partir de los datos originales Xval
    newXval = generate_new_training_data(Xval, p)
    #Media
    mean_val = np.mean(newXval, axis = 0)
    #Desviacion Estandar
    std_des_val = np.std(newXval, axis = 0)
    #Se normalizan los atributos
    X_norm_val = normalize_attributes(newXval, mean, std_des)
    #Se añade columna de 1s
    newXval = np.hstack([np.ones([X_norm_val.shape[0], 1]), X_norm_val])
    #newXval = X_norm_val

    #print("newX", newX)
    #print("newXval", newXval)

    err1, err2 = learning_curve(newX, y, Lambda, Theta, newXval, yval)
    draw_learning_curve(err1, err2, "Lambda 0")
    plt.savefig("Apartado3.2.png")
    plt.show()


    #Apartado 3.2 tests lambda= 1, lambda= 100

    Lambda = 1

    err1, err2 = learning_curve(newX, y, Lambda, Theta, newXval, yval)
    draw_learning_curve(err1, err2, "Lambda 1")
    plt.savefig("Apartado3.2_2.png")
    plt.show()

    Lambda = 100

    err1, err2 = learning_curve(newX, y, Lambda, Theta, newXval, yval)
    draw_learning_curve(err1, err2, "Lambda 100")
    plt.savefig("Apartado3.2_3.png")
    plt.show()
```

```
main_parte3()
```

Apartado 4. Regresión polinomial

```
def cal_func_lambdas(Lambdas, newX, newXval, y, yval, name):
    training = np.zeros((Lambdas.shape[0], 1))
    validation = np.zeros((Lambdas.shape[0], 1))

    i = 0
    for Lambda in Lambdas:

        Theta = np.zeros(newX.shape[1])
        result = opt.minimize(cost_and_gradient, Theta, args=(newX, y, Lambda),
jac=True, method='TNC')
        #result = opt.fmin_tnc(func=cost_funct, x0=Theta, fprime=gradient_funct,
args=(newX, y, Lambda))

        ThetaOpt = result.x

        training[i] = cost(ThetaOpt, newX, y, newX.shape[0])
        validation[i] = cost(ThetaOpt, newXval, yval, newXval.shape[0])
        i = i + 1

    plt.figure()

    plt.xlabel("Lambda")
    plt.ylabel("error")

    plt.plot(Lambdas,training, label="Entrenamiento")
    plt.plot(Lambdas,validation, label="Validacion")
    plt.savefig(name)
    plt.legend()
    plt.show()

    plt.close()
```

```
def main_parte4():

    #Cargamos los datos del fichero "ex5data1.mat'
    datafile = 'ex5data1.mat'
    mat = loadmat(datafile)

    X = mat.get("X")
    y = mat.get("y")
    y = y[:, -1]
    Xval = mat.get("Xval")
    yval = mat.get("yval")
    yval = yval[:, -1]
    Xtest = mat.get("Xtest")
    ytest = mat.get("ytest")
    ytest = ytest[:, -1]

    Lambda = 0
    #Grado del polinomio
    p = 8
```

```
    #Se genera nuevos datos de entrenamiento a partir de los datos originales X
    newX = generate_new_training_data(X, p)
    #Media
    mean = np.mean(newX, axis = 0)
    #Desviacion Estandar
    std_des = np.std(newX, axis = 0)
    #Se normalizan los atributos
    X_norm = normalize_attributes(newX, mean, std_des)
    #Se añade columna de 1s
    newX = np.hstack([np.ones([X_norm.shape[0], 1]), X_norm])

    #Se genera nuevos datos de VALIDACION a partir de los datos originales Xval
    newXval = generate_new_training_data(Xval, p)
    #Se normalizan los atributos
    X_norm_val = normalize_attributes(newXval, mean, std_des)
    #Se añade columna de 1s
    newXval = np.hstack([np.ones([X_norm_val.shape[0], 1]), X_norm_val])
    #newXval = X_norm_val

    #Se genera nuevos datos de TEST a partir de los datos originales Xtest
    newXtest = generate_new_training_data(Xtest, p)
    #Se normalizan los atributos
    X_norm_test = normalize_attributes(newXtest, mean, std_des)
    #Se añade columna de 1s
    newXtest = np.hstack([np.ones([X_norm_test.shape[0], 1]), X_norm_test])


    #Apartado 4.1
    Lambdas = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10])
    LambdasTest = np.array([2.5, 3,3.5, 4, 4.5 ,5])

    cal_func_lambdas(Lambdas, newX, newXval, y, yval, "figure4.1_1.png")

    cal_func_lambdas(LambdasTest, newX, newXval, y, yval, "figure4.1_2.png")

    #Sacamos la gráfica para lamdba= 3
    Lambda = 3
    Theta = np.zeros(newX.shape[1])
    err1, err2 = learning_curve(newX, y, Lambda, Theta, newXval, yval)
    draw_learning_curve(err1, err2, "Lambda 3")
    plt.savefig("Apartado4_1_lambda3.png")
    plt.show()

    #Apartado 4.2

    Theta = np.zeros(newX.shape[1])
    Lambda_3 = 3
    res_error = opt.minimize(cost_and_gradient, Theta, args=(newX, y, Lambda_3),
jac=True, method='TNC')
```

```
    m=  newXtest.shape[0]

    res_cost = cost(res_error.x, newXtest, ytest, m)

    print(res_cost)
```

```
main_parte4()
```