Universidad Complutense de Madrid

IAAC - PRÁCTICA 5

Yaco Alejandro Santiago Pérez

Asignatura: INTELIGENCIA ARTIFICIAL APLICADA A INTERNET DE LAS COSAS

P5:Regresión lineal regularizada: sesgo y varianza

Master IOT

19 de abril de 2020

Índice general

Capítulo 1

Introducción

Esta práctica nos permitirá entender *como hacer* y *para qué sirve* la **Regresión lineal regularizada** mediante lo efectos de *sesgo* y de *varianza*.

Se utilizan los datos históricos sobre el agua que ha derramado una presa en base a los cambios en el nivel del agua.

El objetivo es sobre-ajustar los datos de entrenamiento mediante regresión lineal para conseguir una predicción más exacta.

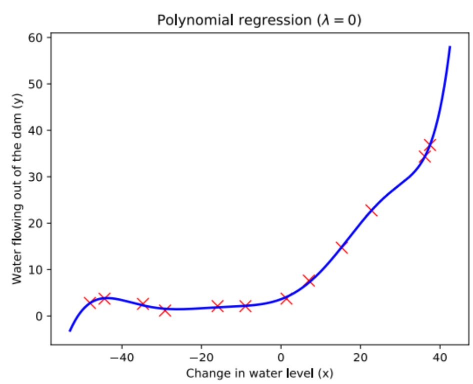


Figure 1.1: *Ejemplo de datos de entrenamiento*

Capítulo 2

Objetivos

En esta práctica, los objetivos son los siguientes:

- Implementar la función de calculo del coste
- Implementar la función de calculo del gradiente
- Modificar las funciones de manera que los resultados devueltos estén regularizados
- Comprobar los costes obtenidos
- Calcular y representar las curvas de aprendizaje
- Conseguir un mayor ajuste mediante formula polinomial
- Disminuir los costes
- Obtener el valor *lambda* óptimo

Capítulo 3

Parte 1: Implementación y Testeo

3.1. Funciones

Las funciones empleadas, en orden de llamada, son las siguientes:

- **pintar(X, y, theta = np.array(([0],[0])), reg = 0):** Es la función encargada de generar el gráfico donde se plasman los puntos, y la recta ajustada a los datos de X e y en caso de recibir el parámetro *theta*.
- **coste(X, y, theta):** Realiza el calculo del coste sin regularizar.
- **coste_regularizado(theta, X, y, l=1):** Realiza el calculo del coste **regularizado** mediante la llamada a la función *coste(X, y, theta)* y añadiéndole la regularización, quedando el calculo total en base a la siguiente formula:

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

Figure 3.1: *Fórmula del coste* ***con*** *regularización*

- **gradiante(theta, X, y):** Función encargada de obtener el gradiente **sin** regularizar.
- **gradiente_regularizado(theta, X, y, l=1):** Realiza el calculo del gradiente **regularizado** mediante la llamada a la función *gradiante(theta, X, y)* y añadiéndole la regularización, quedando el calculo total en base a la siguiente formula:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_0} &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} && \text{para } j = 0 \\ \frac{\partial J(\theta)}{\partial \theta_j} &= \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j && \text{para } j \geq 1 \end{aligned}$$

Figure 3.2: *Fórmula del gradiente* ***con*** *regularización*

- **minTheta(theta, X, y, l = 0):** Función encargada de calcular y devolver las *thetas* óptimas mediante el uso de *scipy.optimize.minimize*.

- **pintarcurvaAprendizaje(theta, X, y, Xval, yval, reg = 0):** Es la función encargada de generar el gráfico donde se muestra la evolución de los errores, tanto de entrenamiento como de validación.
  Para ello a partir de una *theta* inicial, se calculan las *thetas* óptimas y sus correspondientes coste y gradiente para posteriormente hacer la llamada para pintar los datos.

- **printarErroresCurvaAprendizaje(example_num, error_train, error_val, reg):** Función que pinta en una gráfica los datos de los errores calculados por la función anterior.

- **polyFeatures(X, p):** Función encargada de calcular la forma de polinómica, en base al grado que se le indique como parámetro.
  Esta función devuelva la forma polimórfica normal y la normalizada, tras hacer el reajuste en base a la media y varianza de cada columna.

- **plotFit(X, y, degree, num_points, reg = 0):** Realiza las llamadas a *polyFeatures(X, p)* para hacer las transformaciones, y a continuación realiza el calculo de los rangos y pinta los puntos y el ajuste polinómico.

- **main():** Función desde la cual se realizan las distintas llamadas y pruebas precisadas a lo largo del guión de la práctica.

3.2. Ejecución y desarrollo

3.2.1. Coste y Gradiente

Testeo de las funciones, para un valor de *lambda* = 1 y *theta*= [1; 1].
Se debería devolver un coste de **303,993** y un gradiente de **[-15,303;598,250]**.

El resultado de la ejecución es **correcto**:

```
################
Coste: 303.9515255535976
Gradiente: [-15.30301567 598.16741084]
################
Coste reg: 303.9931922202643
Gradiente reg: [-15.30301567 598.25074417]
################
```

3.2.2. Encontrar la Theta óptima

Mediante la función *scipy.optimize.minimize* encuentro el valor de *Theta* que minimiza el error sobre los ejemplos de entrenamiento. Con un valor de $lambda = 0$.

```
lamda=0
theta = np.ones(np.shape(X)[1])
theta_min = minTheta(theta,X, y, lamda)
print("Theta: "+str(theta_min))
pintar(X,y,theta_min,lamda)
```

```
Theta: [13.08790348  0.36777923]
```

La *theta* obtenida define esta recta ajustada a los datos de X e y:

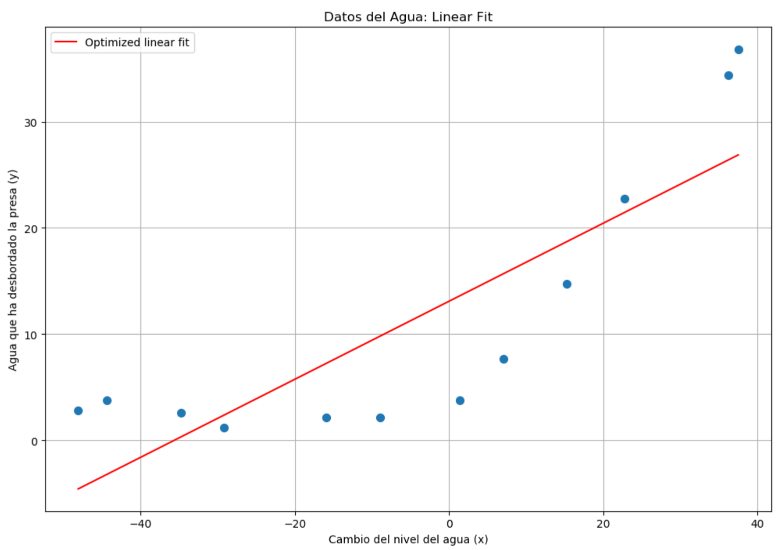


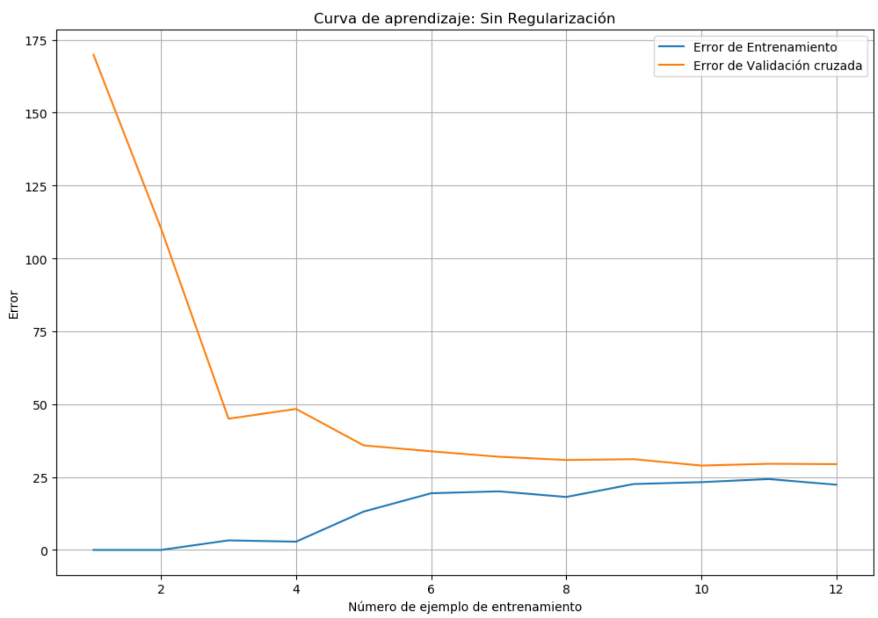
Figure 3.3: *Ajuste lineal obtenido sobre los datos*

3.2.3. Curvas de aprendizaje

La recta obtenida es demasiado simple para ajustarse a los datos de entrenamiento y por ello predice valores sesgados a la recta.

Ahora se calcula el error para el entrenamiento y la validación cruzada conforme aumenta el número de ejemplos de entrenamiento.

```
pintarcurvaAprendizaje(theta, X, y, Xval, yval, reg = 0)
```



El hecho de que en esta curva el error al aumentar el número de ejemplos de entrenamiento se aproxime en los conjuntos de entrenamiento y validación indica que el aprendizaje está sesgado y es necesario utilizar una hipótesis más expresiva que sea capaz de ajustarse mejor a los ejemplos de entrenamiento.

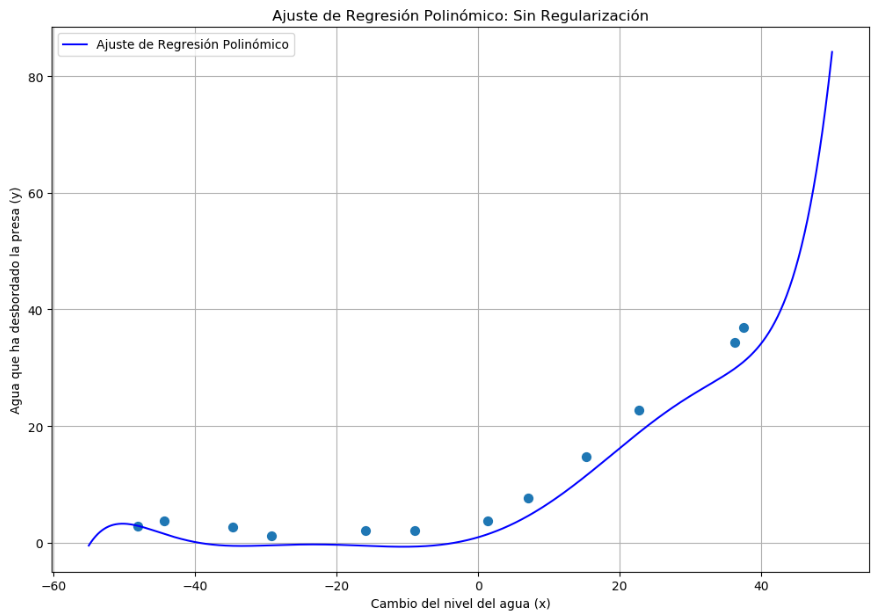
3.2.4. Regresión polinomial

Para conseguir un mayor ajuste a los datos de entrenamiento, usaremos como hipótesis un polinomio de la variable de entrada x que representa el nivel de agua en la presa:

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

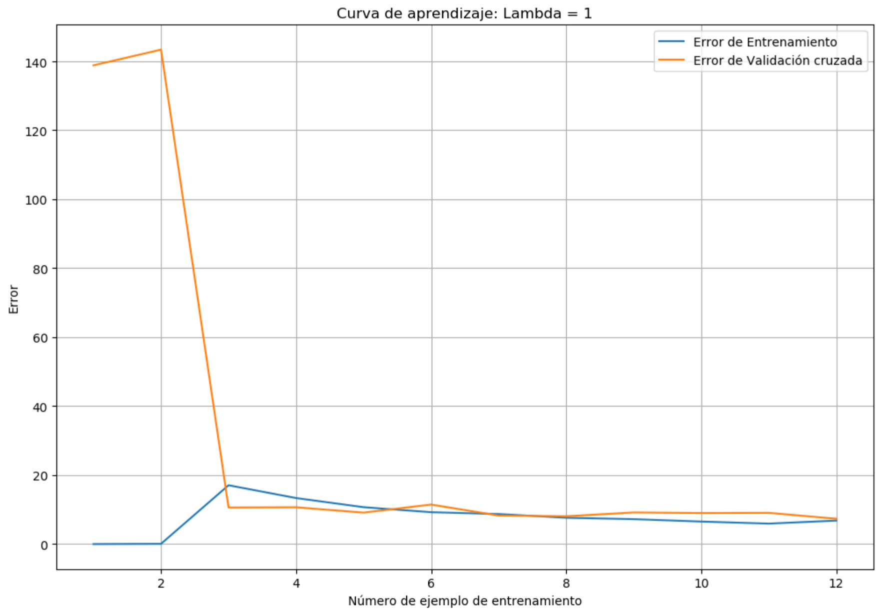
Genero en base a esta formula, tras normalizar los nuevos datos de entrada para aprender un polinomio de grado p = 8.

Vuelvo a aplicar el método de regresión lineal para obtenerla nueva *theta* que minimiza el error para un valor de *lambda* = 0.

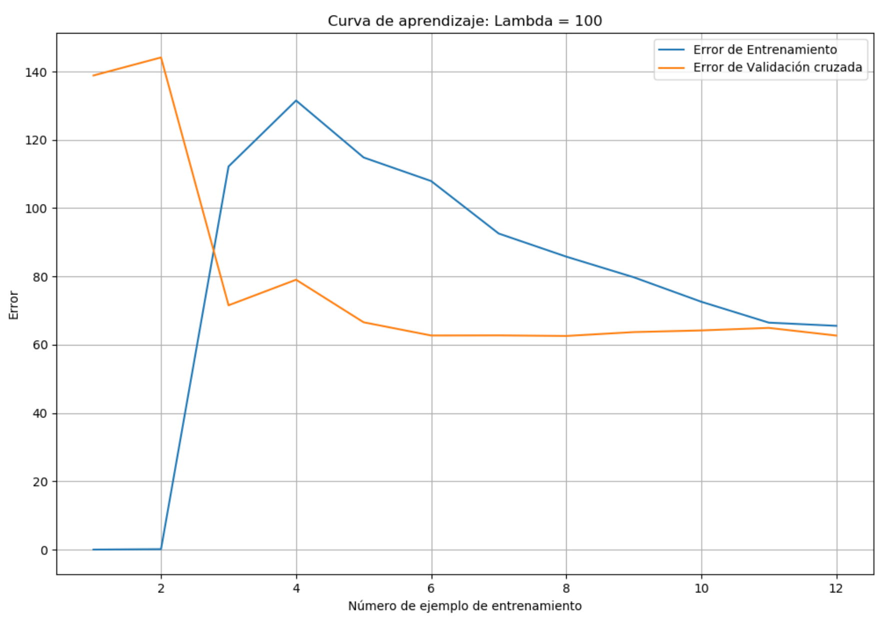


Se ve en la **curva de aprendizaje** que la hipótesis está sobre-ajustada al entrenamiento:

3.2.5. Termino de regularización: Lambda = 1



3.2.6. Termino de regularización: Lambda = 100



Capítulo 4

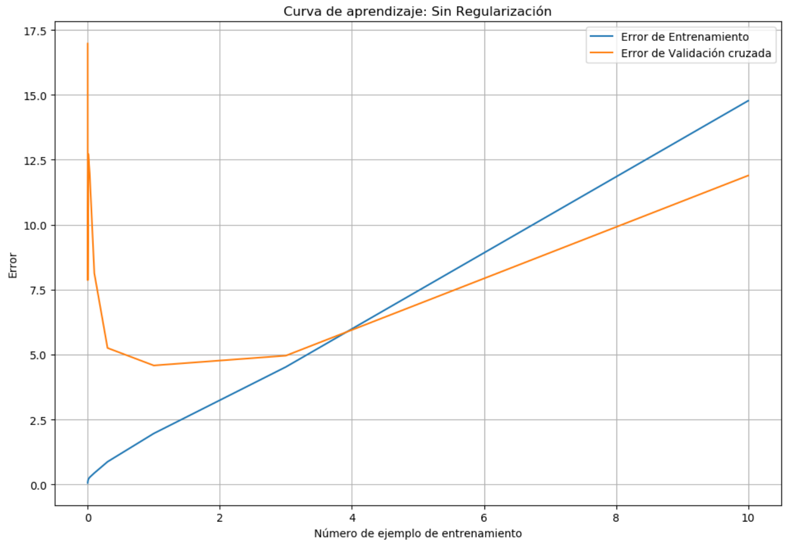
Parte 2: Selección del parámetro Lambda

Como se ha comprobado, el parámetro del término de regularización permite controlar el grado de ajuste a los ejemplos de entrenamiento.

Para obtener el valor óptimo, voy a probar uno a uno los siguientes valores: *[0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]*.

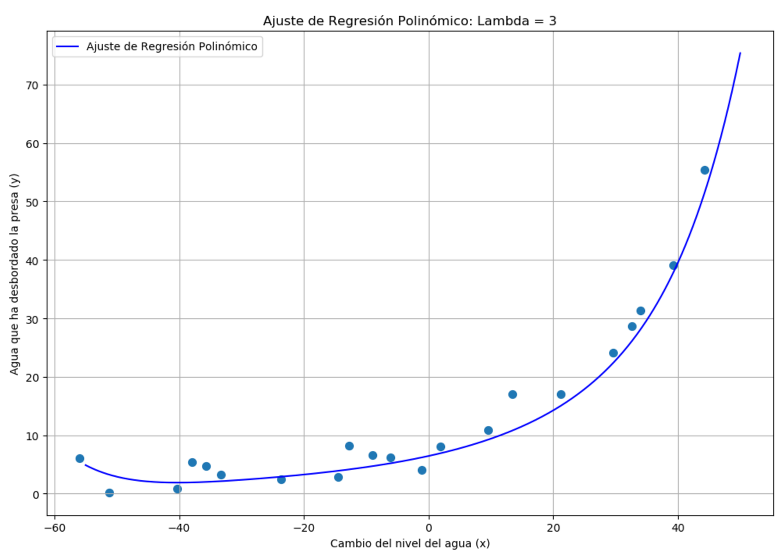
```
landaList = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
costeE, costeVal = [], []
for l in landaList:
    res = minTheta(starting_theta,X_poly, y, l)
    print("aqui␣res:␣"+str(res))
    tramic = coste( X_poly, y, res)
    costeE.append(tramic)
    validac = coste(X_poly_val, yval, res)
    costeVal.append(validac)
printarErroresCurvaAprendizaje(landaList, costeE, costeVal, ↩
    0)
```

La gráfica obtenida es la siguiente:



Donde se puede comprueba que el mejor valor de *lambda* parece estar cerca de 3.

Probando sobre el conjunto Xtest (pasado a forma polinomial con potencia 8) e ytest, para $lambda = 3$ se obtiene la siguiente curva:



Capítulo 5

Código

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

def pintar(X, y, theta = np.array(([0],[0])), reg = 0):
    plt.figure(figsize=(12, 8))
    plt.scatter(X[:, 1], y, s = 50,  linewidths = 1)
    plt.grid(True)
    plt.title('Datos del Agua')
    plt.xlabel('Cambio del nivel del agua (x)')
    plt.ylabel('Agua que ha desbordado la presa (y)')
    if theta.any() != 0:
        plt.plot(np.linspace(X.min(), X.max()), theta[0] + theta[1] * np.linspace(X.min(), X.max()),  color='red', label = 'Optimized linear fit')
        plt.title('Datos del Agua: Linear Fit')

    plt.legend()
    #plt.show()

def coste(X, y, theta):
    h = np.dot(X, theta)
    tmp = (h-y)** 2
    return tmp.sum()/(2*len(X))


def gradiante(theta, X, y):
    m = X.shape[0]
    inner = X.T @ (X @ theta - y)
    return inner / m

```

```python
def coste_regularizado(theta, X, y, l=1):
    m = X.shape[0]
    reg = (l / (2 * m)) * np.power(theta[1:], 2).sum()

    return coste(X, y, theta) + reg

def gradiente_regularizado(theta, X, y, l=1):
    m = X.shape[0]
    reg = theta.copy()
    reg[0] = 0

    reg = (l / m) * reg

    return gradiante(theta, X, y) + reg

def minTheta(theta, X, y, l = 0):
    return minimize(fun=coste_regularizado,x0=theta,args=(X, y, l),method='TNC',jac=gradiente_regularizado,options={'disp':True}).x

def pintarcurvaAprendizaje(theta, X, y, Xval, yval, reg = 0):
    m = y.size

    error_train = np.zeros((m, 1))
    error_val = np.zeros((m, 1))

    example_num = np.arange(1, (X.shape[0] + 1))
    for i in np.arange(m):

        opt_theta = minTheta(theta, X[:i + 1], y[:i + 1], reg)
        error_train[i] = coste_regularizado(opt_theta, X[:i + 1], y[:i + 1], reg)
        error_val[i] = coste_regularizado(opt_theta, Xval, yval, reg)

    printarErroresCurvaAprendizaje(example_num, error_train, error_val, reg)
    return opt_theta

def printarErroresCurvaAprendizaje(example_num, error_train, error_val, reg):
    plt.figure(figsize = (12, 8))
    plt.plot(example_num, error_train, label = 'Error de Entrenamiento')
```

```python
    plt.plot(example_num, error_val, label = 'Error de Validaci n cruzada')
    plt.title('Curva de aprendizaje: Sin Regularizaci n')
    if reg != 0:
        plt.title('Curva de aprendizaje: Lambda = {0}'.format(reg))
    plt.xlabel('N mero de ejemplo de entrenamiento')
    plt.ylabel('Error')
    plt.legend()
    plt.grid(True)
    plt.show()

def polyFeatures(X, p):
    for i in np.arange(p):
        dim = i + 2
        X = np.insert(X, X.shape[1], np.power(X[:,1], dim), axis = 1)

    X_norm = X
    #Normalizar
    means = np.mean(X_norm, axis=0)
    X_norm[:, 1:] = X_norm[:, 1:] - means[1:]
    stds = np.std(X_norm, axis = 0)
    X_norm[:, 1:] = X_norm[:, 1:] / stds[1:]

    return X, X_norm

def plotFit(X, y, degree, num_points, reg = 0):
    X_poly = polyFeatures(X, degree)[1]
    starting_theta = np.ones((X_poly.shape[1], 1))
    opt_theta = minTheta(starting_theta, X_poly, y, reg)
    x_range = np.linspace(-55, 50, num_points)
    x_range_poly = np.ones((num_points, 1))
    x_range_poly = np.insert(x_range_poly, x_range_poly.shape[1], x_range.T, axis = 1)
    x_range_poly = polyFeatures(x_range_poly, len(starting_theta)-2)[0]
    y_range = x_range_poly @ opt_theta
    pintar(X, y)
    plt.plot(x_range, y_range,  color = "blue", label = "Ajuste de Regresi n Polin mico")
    plt.title('Ajuste de Regresi n Polin mico: Sin Regularizaci n')
    if reg != 0:
```

```
        plt.title('Ajuste de Regresi n Polin mico: Lambda = {0}'.format(reg))
    plt.legend()
    plt.show()

def main():
    dato = loadmat('ex5data1.mat')
    X, y, Xval, yval, Xtest, ytest = map(np.ravel, [dato['X'], dato['y'], dato['Xval'], dato['yval'], dato['Xtest'], dato['ytest']])
    X, Xval, Xtest = [np.insert(x.reshape(x.shape[0], 1), 0, np.ones(x.shape[0]), axis=1) for x in (X, Xval, Xtest)]

    pintar(X,y)
    plt.show()

    ###############
    print("################")
    lamda = 1
    theta = np.ones(X.shape[1]) #[1. 1.]
    cost=coste(X, y, theta)
    print("Coste: "+str(cost))
    g=gradiante(theta, X, y)
    print("Gradiente: "+str(g))
    print("################")

    ################ Regularizado
    cost=coste_regularizado(theta, X, y, lamda)
    print("Coste reg: "+str(cost))
    g=gradiente_regularizado(theta, X, y, lamda)
    print("Gradiente reg: "+str(g))
    print("################")

    ################ Linear Fit
    lamda=0
    theta = np.ones(np.shape(X)[1])
    theta_min = minTheta(theta,X, y, lamda)
    print("Theta: "+str(theta_min))
    pintar(X,y,theta_min,lamda)
    plt.show()
    print("################")

    ################ Curvas de aprendizaje
    pintarcurvaAprendizaje(theta, X, y, Xval, yval, reg = 0)

```

```
    ################ Regresi n polinomial
    X_poly = polyFeatures(X, 8)[1]
    X_poly_val = polyFeatures(Xval, 8)[1]

    plotFit(X, y, 8, 1000, reg = 0)

    starting_theta = np.ones((X_poly.shape[1], 1))
    pintarcurvaAprendizaje(starting_theta, X_poly, y, X_poly_val, yval,0)

    #Lamda 1
    pintarcurvaAprendizaje(starting_theta, X_poly, y, X_poly_val, yval,1)
    #Lamda 100
    pintarcurvaAprendizaje(starting_theta, X_poly, y, X_poly_val, yval,100)

    ################ Descubrir valor optimo

    landaList = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
    costeE, costeVal = [], []
    for l in landaList:
        res = minTheta(starting_theta,X_poly, y, l)
        print("aqui res: "+str(res))
        tramic = coste( X_poly, y, res)
        costeE.append(tramic)
        validac = coste(X_poly_val, yval, res)
        costeVal.append(validac)
    printarErroresCurvaAprendizaje(landaList, costeE, costeVal, 0)

    ############## Prueba para Xtest, ytest con lamda 3

    plotFit(Xtest, ytest, 8,1000,3)

    Xtest_poly = polyFeatures(Xtest, 8)[0]

    #Normalizar
    #A partir de a qu , no me da lo esperado, por lo que me he ahorrado incluirlo en la memoria
    means = np.mean(X, axis=0)
    stds = np.std(X, axis = 0)

    Xtest_poly[:, 1:] = Xtest_poly[:, 1:] - means[1:]
    Xtest_poly[:, 1:] = Xtest_poly[:, 1:] / stds[1:]
```

```python

    starting_theta = np.ones((Xtest_poly.shape[1], 1))
    res = minTheta(starting_theta,Xtest_poly, ytest, 3)
    error_train = coste_regularizado(res, Xtest_poly, ytest, 3)
    print(error_train)
    pintarcurvaAprendizaje(res, Xtest_poly, ytest, X_poly_val, ↩
       yval,3)

    ##############
main()
```

Código: p5.py