Práctica 5
Regresión lineal regularizada: sesgo y varianza

Rafael Herrera Troca
Rubén Ruperto Díaz

Aprendizaje Automático y Big Data
Doble Grado en Ingeniería Informática y Matemáticas

4 de septiembre de 2020

1. Introducción

En esta ocasión, vamos a analizar los efectos del sesgo (*bias*) y la varianza (*variance*) a través de un predictor entrenado mediante regresión lineal y polinomial. Para ello, elegiremos parámetros con los que el modelo no sea capaz de realizar predicciones lo suficientemente correctas ni siquiera para los ejemplos de entrenamiento y luego probaremos otras que lo conduzcan a ajustarse demasiado a esos ejemplos, tanto que lo aprendido no se pueda generalizar a datos nuevos.

Hemos programado el código en el entorno *Spyder* con Python 3.7.

2. Procedimiento

El procedimiento seguido para la realización de esta práctica es extremadamente sencillo y repetitivo. Comenzamos por definir la función de coste para la regresión, para lo cual vectorizamos la fórmula dada en el enunciado, obteniendo:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}((X\theta - y)^{(i)})^2 + \frac{\lambda}{2N}\sum_{j=1}^{M}\theta_j^2$$

donde $X \in \mathcal{M}_{N\times(M+1)}(\mathbb{R})$ son los datos de entrada, $\theta \in \mathcal{M}_{(M+1)\times 1}(\mathbb{R})$ son los parámetros de la regresión, $y \in \mathcal{M}_{N\times 1}(\mathbb{R})$ son los resultados y $\lambda \in \mathbb{R}$ es el parámetro de regularización.

Definimos también el gradiente de la función de coste vectorizando la fórmula dada en el enunciado y obteniendo:

$$\nabla J(\theta) = \frac{(X\theta - y)^T X + \lambda(0, \theta_1, \ldots, \theta_M)}{N}$$

donde todos los parámetros tienen el mismo significado que en la fórmula de la función de coste. Nótese que para el término de regularización no se debe tener en cuenta θ_0, por lo que la sustituimos por un cero.

Además de estas funciones, esenciales para poder entrenar la regresión, definimos otras que nos permiten modificar los datos dados como nos es conveniente. Éstas son *potencia*, que dado el vector $X \in \mathcal{M}_{N\times 1}(\mathbb{R})$ y un valor p devuelve la matriz $X' = (X, X^2, \ldots, X^p) \in \mathcal{M}_{N\times p}(\mathbb{R})$, y *normalizar*, que dada la matriz de datos $X \in \mathcal{M}_{N\times M}(\mathbb{R})$ normaliza las columnas restando la media y dividiendo por la desviación típica dadas o, si no se dan estos datos, de forma que la media sea 0 y la desviación típica 1.

Una vez definidas estas funciones usamos *minimize*, de *scipy.optimize*, dándole nuestras funciones de coste y gradiente para entrenar la regresión correspondiente a cada apartado con los datos adaptados a cada caso usando *potencia* y *normalizar* y el parámetro de regresión adecuado.

3. Resultados

3.1. Regresión lineal

En primer lugar, hemos tratado de aplicar regresión lineal para entrenar un modelo capaz de predecir la cantidad de agua que sale de una presa (y) en función de la variación del nivel del agua en la presa (x). Una vez hemos verificado que la implementación era correcta, hemos entrenado el predictor con parámetro de regularización $\lambda = 0$ (ya que no hay términos en θ que necesiten ser regularizados), resultando la recta $h_\theta(x) = \theta_0 + \theta_1 x$ que mejor se ajusta a los ejemplos de entrenamiento.

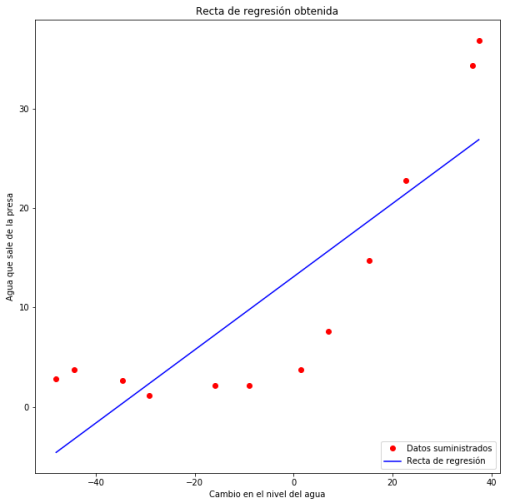

Figura 1: Resultado de la regresión lineal

Sin embargo, a la vista de cómo se distribuyen los datos de entrenamiento en la gráfica, podemos concluir que una recta no es capaz de captar cómo se relaciona la cantidad de agua liberada con la variación del nivel de agua. Por tanto, este es un caso de sesgo o ajuste insuficiente, en el que el modelo elegido no es lo suficientemente complejo como para aprender las relaciones no lineales entre los atributos de los datos de entrenamiento.

3.2. Curvas de aprendizaje

Otra forma de detectar el sesgo que sufre un modelo es mediante las curvas de aprendizaje, en las que se muestra la evolución del error al predecir ejemplos de entrenamiento y otros nuevos extraídos de un conjunto de datos de validación, en función del tamaño de la muestra de entrenamiento. Esto es especialmente útil cuando los datos con los que trabajamos tienen muchos atributos y por tanto no se puede realizar una gráfica como la que hemos obtenido nosotros en la figura 1.

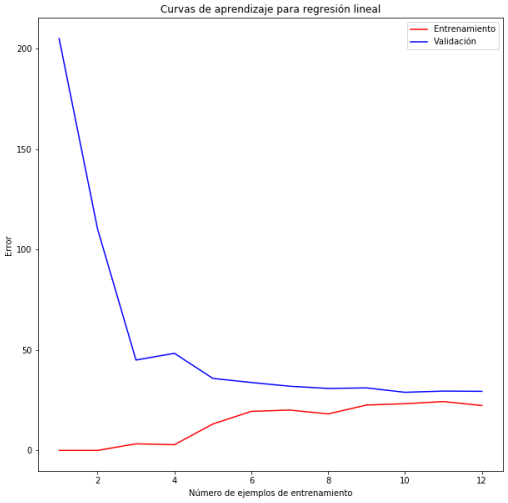

Figura 2: Error en función el tamaño del subconjunto de entrenamiento

Hemos aplicado esta técnica al modelo de regresión lineal con $\lambda = 0$, entrenándolo con subconjuntos de los datos de entrenamiento cada vez más grandes y viendo

cómo evoluciona el error. Observamos que al llegar a utilizar todos los ejemplos durante el entrenamiento el error es demasiado alto tanto para los ejemplos de entrenamiento como los de validación, lo cual indica lo que ya hemos explicado, que el modelo sufre sesgo por ser demasiado simple para aprender las relaciones entre los atributos de los ejemplos a predecir.

3.3. Regresión polinomial y curvas de aprendizaje

A continuación, analizaremos una situación en la que el problema es la varianza (*variance*), es decir, el modelo está sobreajustado a los datos de entrenamiento, por lo que la hipótesis aprendida no se puede generalizar a ejemplos nuevos y falla al realizar predicciones con ellos. Para conseguir el sobreajuste, utilizaremos regresión polinomial, ya que un polinomio de grado alto tiene una mayor flexibilidad para atravesar todos los ejemplos de entrenamiento. Por tanto, crearemos nuevos atributos, que serán las potencias sucesivas del valor de x, de modo que el polinomio vendrá dado por $h_\theta(x) = \theta_0 + \theta_1 x + \ldots \theta_p x^p$, donde p es el grado del polinomio, que debemos elegir previamente. En este caso, fijaremos $p = 8$. Además, al elevar los valores de x a números grandes (hasta $p = 8$), estamos introduciendo diferencias de varios órdenes de magnitud entre los valores de distintos atributos, por lo que es necesario normalizarlos antes de empezar el aprendizaje para evitar que se alargue más de lo habitual.

Al haber introducido en el vector θ los nuevos términos que acompañan a las potencias de x, tiene sentido introducir también cierta regularización que controle su relevancia en la función $J(\theta)$. Cuanto mayor sea λ, más grande será el término de regularización, lo que obligará a reducir los valores de θ para minimizar $J(\theta)$, y por tanto se reducirán los coeficientes que acompañan a x, $\ldots$, x^p en el polinomio $h_\theta(x)$. Esto tiene como consecuencia que se reducirá su capacidad de ajustarse a los ejemplos de entrenamiento.

En la primera gráfica de la figura 3 se muestra la predicción realizada por la regresión polinomial con $\lambda = 0$ en función de cada valor de x, y cómo se ajusta con mucha precisión a los ejemplos de entrenamiento. Sin embargo, en la siguiente gráfica, donde mostramos el error en las predicciones en función del tamaño del subconjunto de entrenamiento, podemos comprobar que, aunque el error con los ejemplos de entrenamiento sea muy pequeño, al predecir los ejemplos de validación se producen grandes errores, lo que se traduce en que el modelo sufre varianza.

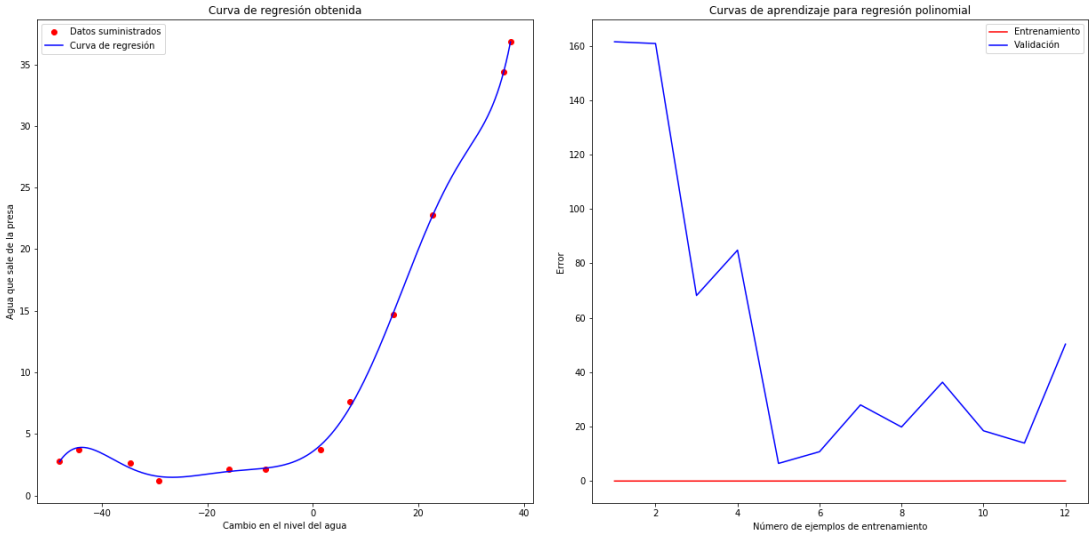

Figura 3: Resultados de la regresión polinomial con $\lambda = 0$

Si probamos a repetir el entrenamiento con $\lambda = 1$, resulta que la regularización reduce la influencia de los términos con x del polinomio, especialmente los de mayor grado, y por tanto reduce su capacidad de adoptar una forma muy ajustada a los ejemplos de entrenamiento. La forma de la curva de regresión es más generalizable a casos nuevos distintos de los de entrenamiento, por lo que los resultados mejoran drásticamente.

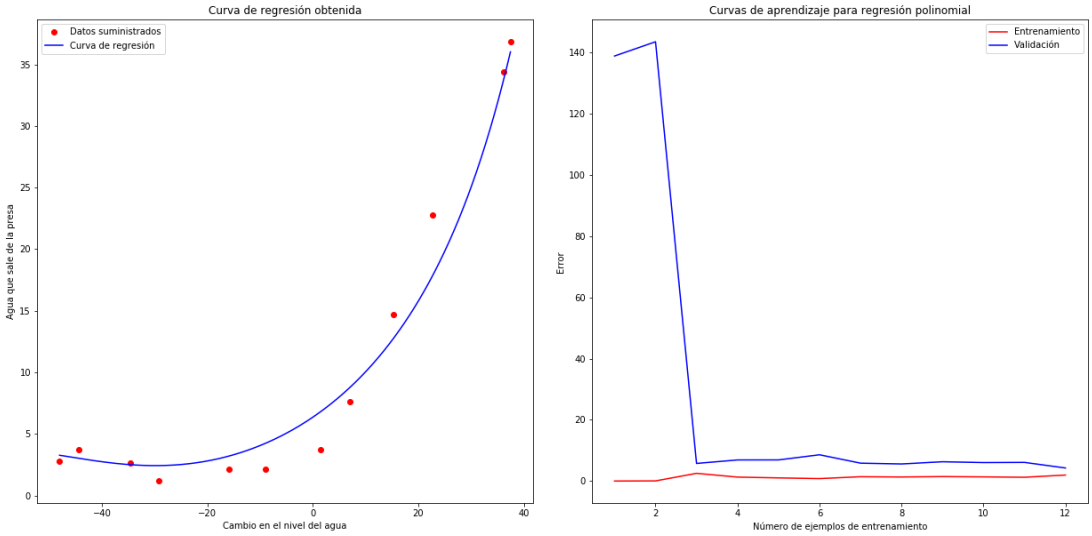

Figura 4: Resultados de la regresión polinomial con $\lambda = 1$

También hemos probado a entrenar la regresión polinomial con $\lambda = 100$. En este caso el término de regularización en $J(\theta)$ se vuelve tan grande que al minimizar tiene más peso que el propio error cometido al predecir los ejemplos de entrenamiento. La minimización resulta en una elección de valores de θ tan pequeños que la curva queda casi plana. En este caso el problema es el sesgo: la elección de λ restringe tanto el aprendizaje que no es capaz de acertar ni siquiera con los ejemplos de entrenamiento.

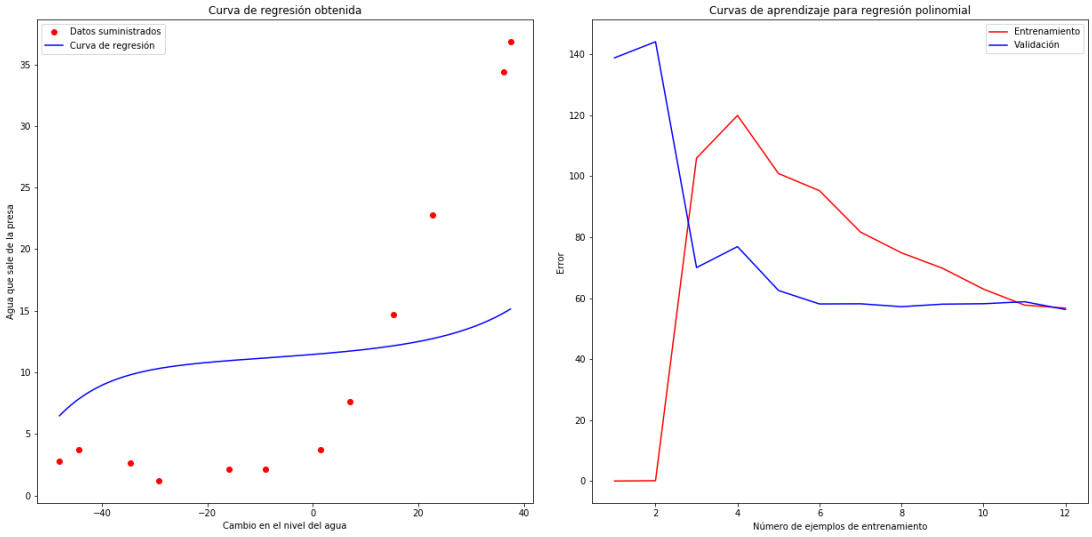

Figura 5: Resultados de la regresión polinomial con $\lambda = 100$

3.4. Elección del parámetro λ

Por último, hemos estudiado cuál es el valor óptimo de λ para el término de regularización de la regresión polinomial. Para ello, hemos entrenado el modelo varias veces variando el valor del parámetro $\lambda \in \{0, 0{,}001, 0{,}003, 0{,}01, 0{,}03, 0{,}1, 0{,}3, 1, 3, 10\}$ y hemos representado en una gráfica la evolución del error cometido al predecir los ejemplos de entrenamiento y los de validación. Esta vez durante el entrenamiento hemos utilizado todos los ejemplos del conjunto dedicado a tal efecto.

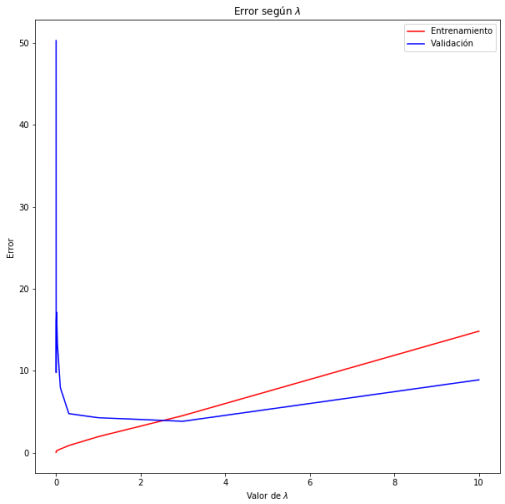

Figura 6: Error en función el tamaño del subconjunto de entrenamiento

En la gráfica vemos que el valor del parámetro de regularización para el que se obtiene un mejor resultado con los datos de validación es $\lambda = 3$, obteniendo un resultado similar en el caso de los datos de entrenamiento. Para valores menores de λ, se obtiene mejor resultado en los ejemplos de entrenamiento, a costa de grandes errores en los de validación (varianza), mientras que para valores mayores, el error va creciendo en ambos casos (sesgo).

Finalmente, al evaluar el resultado de la regresión polinomial con $\lambda = 3$ sobre los ejemplos de test, que no hemos usado hasta ahora, obtenemos un error de 3,572.

4. Conclusiones

Esta práctica, aunque muy sencilla en cuanto programación, nos ha permitido comprobar que cuando se trabaja con aprendizaje automático, aunque sea una de las técnicas más simples como es la regresión, es importante calibrar bien todos los parámetros para conseguir buenos resultados. En particular, hemos visto como para la regresión hay que buscar el grado adecuado para la curva y el parámetro de regularización adecuados y como no hacerlo puede provocar sesgo, sobreajuste o simplemente un entrenamiento no eficaz.

5. Anexo: Código

A continuación se ofrece el código escrito para resolver los ejercicios propuestos en la práctica.

```
'''
Práctica 5

Rubén Ruperto Díaz y Rafael Herrera Troca
'''

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


os.chdir("./resources")


# Función de coste para la regresión lineal regularizada
def coste(theta, X, Y, reg=0):
    return np.sum((np.dot(X, theta) - Y)**2) / (2*len(Y)) + reg / (2*
    len(Y)) * np.sum(theta[1:]**2)

# Gradiente de la función de coste para la regresión lineal
# regularizada
def gradiente(theta, X, Y, reg=0):
    return np.dot((np.dot(X, theta) - Y).T, X) / len(Y) + reg / len(Y)
    * np.concatenate(([0], theta[1:]))

# Dada una matriz columna X, devuelve la matriz que en la columna i
# tiene X^i, con i en [1,p]
def potencia(X, p):
    res = X
    pot = X
    for i in range(2,p+1):
        pot = pot * X
        res = np.hstack((res, pot))
    return res
```

```
35
36 # Dada una matriz de datos, los normaliza por columnas con la media
37 # y desviación típica dadas o, si son vacíos, para que tengan
38 # media 0 y desviación típica 1, y devuelve los datos normalizados,
39 # la media y la desviación típica
40 def normalizar(X, mu=[], sigma=[]):
41     if len(mu)==0 or len(sigma)==0:
42         mu = np.mean(X, 0)
43         sigma = np.std(X, 0)
44     X2 = (X - mu) / sigma
45     return X2, mu, sigma
46
47
48 ## Parte 1: Regresión lineal
49
50 # Leemos los datos
51 data = loadmat('ex5data1.mat')
52 y = data['y'].ravel()
53 yval = data['yval'].ravel()
54 ytest = data['ytest'].ravel()
55 X = data['X']
56 Xval = data['Xval']
57 Xtest = data['Xtest']
58 X2 = np.hstack((np.array([np.ones(len(y))]).T, X))
59 Xval2 = np.hstack((np.array([np.ones(len(yval))]).T, Xval))
60 Xtest2 = np.hstack((np.array([np.ones(len(ytest))]).T, Xtest))
61
62 # Entrenamos la regresión
63 theta0 = np.array([0,0])
64 reg = 0
65 theta = minimize(fun=coste, x0=theta0, args=(X2, y, reg), method='TNC'
    , jac=gradiente)['x']
66
67 # Representamos los datos y la recta obtenida
68 part = np.linspace(min(X),max(X),1000)
69 plt.figure(figsize=(10,10))
70 plt.plot(X, y, 'ro', label="Datos suministrados")
71 plt.plot(part,theta[0]+theta[1]*part,'b',label="Recta de regresión")
72 plt.title("Recta de regresión obtenida")
73 plt.xlabel("Cambio en el nivel del agua")
74 plt.ylabel("Agua que sale de la presa")
75 plt.legend(loc="lower right")
76 plt.savefig("regr1.png")
77 plt.show()
78
79
80 ## Parte 2: Curvas de aprendizaje para la regresión lineal
81
82 # Calculamos el error con el conjunto de entrenamiento y el de
83 # validación para subconjuntos crecientes de entrenamiento.
84 # Para calcular el error usamos la función de coste sin regularizar
85 errorTrain = []
86 errorVal = []
87 for i in range(1,len(X2)+1):
88     theta = minimize(fun=coste, x0=theta0, args=(X2[:i],y[:i],reg),
```

```
    method='TNC', jac=gradiente)['x']
89      errorTrain.append(coste(theta, X2[:i], y[:i]))
90      errorVal.append(coste(theta, Xval2, yval))
91
92  # Representamos las curvas de aprendizaje obtenidas
93  plt.figure(figsize=(10,10))
94  plt.plot(range(1,len(errorTrain)+1),errorTrain,'r',label="
        Entrenamiento")
95  plt.plot(range(1,len(errorVal)+1),errorVal,'b',label="Validación")
96  plt.title("Curvas de aprendizaje para regresión lineal")
97  plt.xlabel("Número de ejemplos de entrenamiento")
98  plt.ylabel("Error")
99  plt.legend(loc="upper right")
100 plt.savefig("curvas1.png")
101 plt.show()
102
103
104 ## Parte 3: Regresión polinomial y curvas de aprendizaje
105
106 # Normalizamos y preparamos los datos
107 X2, mu, sigma = normalizar(potencia(X, 8))
108 X2 = np.hstack((np.array([np.ones(len(y))]).T, X2))
109 Xval2 = normalizar(potencia(Xval, 8), mu, sigma)[0]
110 Xval2 = np.hstack((np.array([np.ones(len(yval))]).T, Xval2))
111 Xtest2 = normalizar(potencia(Xtest, 8), mu, sigma)[0]
112 Xtest2 = np.hstack((np.array([np.ones(len(yval))]).T, Xtest2))
113
114 # Hacemos el proceso para términos de regularización 0, 1 y 100
115 for reg in [0,1,100]:
116
117     # Entrenamos la regresión
118     theta0 = np.zeros(np.shape(X2)[1])
119     theta = minimize(fun=coste, x0=theta0, args=(X2,y,reg), method='
        TNC', jac=gradiente)['x']
120
121     # Representamos los datos y la curva obtenida
122     partX = np.arange(min(X),max(X),0.05)
123     partX2 = normalizar(potencia(np.array([partX]).T,8),mu,sigma)[0]
124     partX2 = np.hstack((np.array([np.ones(len(partX2))]).T,partX2))
125     partY = np.dot(partX2, theta)
126     plt.figure(figsize=(10,10))
127     plt.plot(X, y, 'ro', label="Datos suministrados")
128     plt.plot(partX, partY, 'b', label="Curva de regresión")
129     plt.title("Curva de regresión obtenida")
130     plt.xlabel("Cambio en el nivel del agua")
131     plt.ylabel("Agua que sale de la presa")
132     plt.legend(loc="upper left")
133     plt.savefig("regr2R"+str(reg)+".png")
134     plt.show()
135
136     # Calculamos el error con el conjunto de entrenamiento y el de
137     # validación para subconjuntos crecientes de entrenamiento.
138     # Para calcular el error usamos la función de coste sin
139     # término de regularización
140     errorTrain = []
```

```
    errorVal = []
    for i in range(1,len(X2)+1):
        theta = minimize(fun=coste,x0=theta0,args=(X2[:i],y[:i],reg),
    method='TNC',jac=gradiente)['x']
        errorTrain.append(coste(theta, X2[:i], y[:i]))
        errorVal.append(coste(theta, Xval2, yval))

    # Representamos las curvas de aprendizaje obtenidas
    plt.figure(figsize=(10,10))
    plt.plot(range(1,len(errorTrain)+1), errorTrain, 'r', label="
    Entrenamiento")
    plt.plot(range(1,len(errorVal)+1), errorVal, 'b', label="
    Validación")
    plt.title("Curvas de aprendizaje para regresión polinomial")
    plt.xlabel("Número de ejemplos de entrenamiento")
    plt.ylabel("Error")
    plt.legend(loc="upper right")
    plt.savefig("curvas2"+str(reg)+".png")
    plt.show()


## Parte 4: Selección del parámetro de regularización

# Entrenamos la regresión para distintos valores del parámetro de
# regularización y calculamos el error con cada uno
errorTrain = []
errorVal = []
regValues = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
for reg in regValues:

    # Entrenamos la regresión
    theta0 = np.zeros(np.shape(X2)[1])
    theta = minimize(fun=coste, x0=theta0, args=(X2, y, reg), method='
    TNC', jac=gradiente)['x']

    # Calculamos y almacenamos el error cometido con los datos de
    # entrenamiento y los de validación
    errorTrain.append(coste(theta, X2[:i], y[:i]))
    errorVal.append(coste(theta, Xval2, yval))

# Representamos el error obtenido según el valor del
# parámetro de regularización
plt.figure(figsize=(10,10))
plt.plot(regValues, errorTrain, 'r', label="Entrenamiento")
plt.plot(regValues, errorVal, 'b', label="Validación")
plt.title(r"Error según $\lambda$")
plt.xlabel(r"Valor de $\lambda$")
plt.ylabel("Error")
plt.legend(loc="upper right")
plt.savefig("errorReg.png")
plt.show()

# Tomamos el parámetro de regularización que reduce la diferencia
# entre el error con los datos de entrenamiento y el error con
# los datos de validación
```

```
reg = regValues[np.argmin(abs(np.array(errorTrain) - np.array(errorVal
    )))]
print("El mejor parámetro de regularización es", reg)

# Entrenamos la regresión con dicho parámetro de regularización y
# comprobamos el error con los datos de test
theta0 = np.zeros(np.shape(X2)[1])
theta = minimize(fun=coste, x0=theta0, args=(X2, y, reg), method='TNC'
    , jac=gradiente)['x']
error = coste(theta, Xtest2, ytest)
print("El error con los datos de test es", error)
```