Aprendizaje automático y minería de datos

Práctica 5 – Regresión lineal regularizada: sesgo y varianza

Ramón Arjona Quiñones
Celia Castaños Bornaechea

Contenido

Objetivo

Comprobar los diferentes efectos del sesgo y la varianza a través de datos históricos sobre el agua que ha derramado una presa según los cambios en el nivel del agua.

Método de carga

Carga el archivo como un diccionario con seis claves distintas. Con cada clave se asignan y guarda los datos de entrenamiento, validación y test.

```
datos = io.loadmat("ex5data1.mat")
trainingX, trainingY = datos ["X"], datos ["y"]
validationX, validationY = datos ["Xval"], datos ["yval"]
testX, testY = datos ["Xtest"], datos ["ytest"]
```

1. Regresión Lineal Regularizada

Resumen

Calcular el coste y el y el gradiente de la regresión lineal regularizada a partir de los valores de entrada “X”, salida “Y”, el vector de parámetros θ y el parámetro de regularización λ.

1.1 Coste y Gradiente Regularizado

Ambos cálculos se dividen en dos métodos, uno calcula el término sin regularizar y el otro el término regularizado y el total con ambos.

Coste

Ecuación:

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

```
def coste(X, Y, theta):
    m = np.shape(X)[0]
    sumatorio =  np.sum(np.square(np.dot(X, theta) - Y))
    return (1 / (2 * m)) * sumatorio

def costeRegularizado(X, Y, theta, lamda):
    m = np.shape(X)[0]

    #Convierte a theta en un vector de dos dimensiones
    theta = theta.reshape(-1, Y.shape[1])
    #Calcula el término sin regularizar
    term0 = coste(X, Y, theta)

    #Calcula el término con regularización
    sumatorio = np.sum(np.square(theta[1:len(theta)]))
    term1 = (lamda / (2 * m)) * sumatorio

    cost = term0 + term1
```

```
    return cost
```

El coste regularizado resultante para λ = 1 y θ = [1,1] es: `[[303.99319222]]`

Gradiente

Ecuación:

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

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

```
def gradiente(X, Y, theta):
    m = X.shape[0]
    grad = (1 / m) * np.dot(X.T, np.dot(X, theta) - Y)
    return grad

def gradienteRegularizado(X, Y, theta, lamda):
    m = X.shape[0]

    #Convierte a theta en un vector de dos dimensiones
    theta = theta.reshape(-1, Y.shape[1])
    #Calcula el término sin regularizar
    auxGrad = gradiente(X, Y, theta)
    #Calcula el término con regularización
    gradReg = auxGrad + (lamda / m ) * theta

    gradReg[0] = auxGrad[0]
    return gradReg.flatten()
```

El gradiente regularizado resultante para λ = 1 y θ = [1,1] es:

```
[[-15.30301567]
 [598.25074417]]
```

1.2 Minimización del error

Ahora hay que encontrar el valor para θ que minimiza el error. Para ello apartado hay que usar la función *optimize.minimize* de la librería *scipy*.

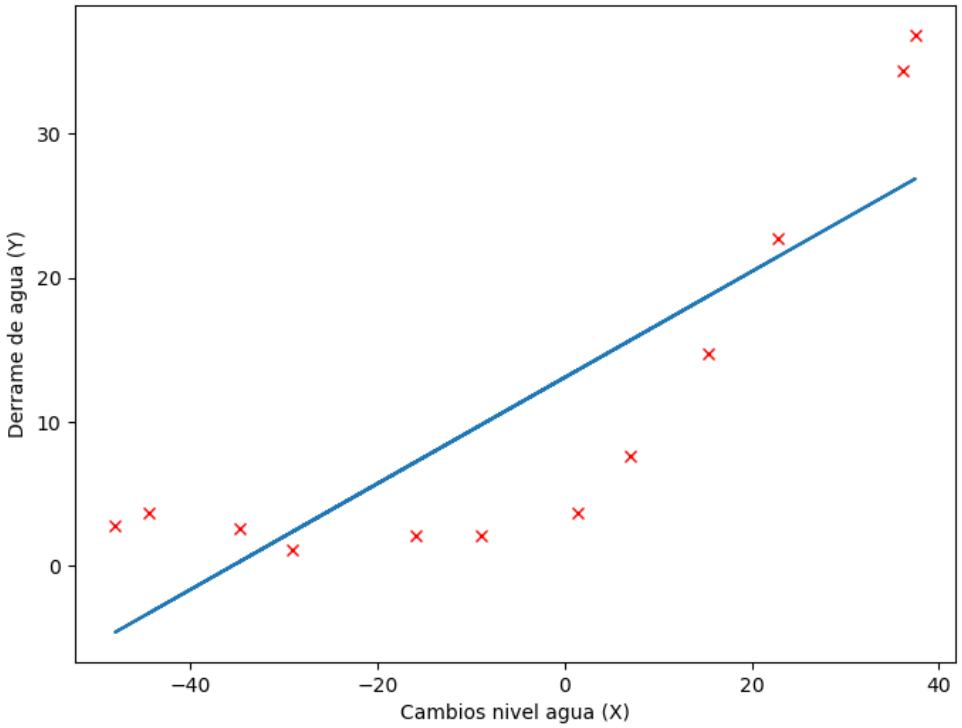
Para facilitar su uso hemos creado dos métodos anidados (un método dentro de otro método). Al método padre o raíz le pasamos todos los parámetros necesarios para el cálculo del coste y el gradiente ("X", "Y" y "λ") y el método hijo recibe como parámetro solamente θ. Este último tiene una llamada a otro método el cual a su vez llama y devuelve el resultado de los métodos del coste y el gradiente regularizados.

En el método padre se llama a la función *scipy.optimize.minimize* pasándole en el parámetro "*fun*" el nombre de la función hija y su parámetro. El resultado de todo esto se guarda en θ, que a su vez es el objeto que se devuelve.

```
def costeYGradiente(X, Y, theta, lamda):
    c = costeRegularizado( X, Y, theta, lamda)
    g = gradienteRegularizado(X, Y, theta, lamda)
    return c, g

def entrenamientoMinimizarTheta(X, Y, lamda):
    theta = np.zeros([X.shape[1], 1])

    def costFunction(theta):
        return costeYGradiente(X, Y, theta, lamda)

    theta = opt.minimize(fun=costFunction, x0=theta, method='CG', jac=Tru
e, options={'maxiter':200})
    return theta.x
```

El valor resultante para θ que minimiza el error es: `[13.08790351  0.36777923]`

Por último, pintamos la gráfica con la recta que ajustada a los datos de “X” e “Y”.



Código del Ejercicio

```
def Ejercicio1():
    #Leemos los valores de la matriz de datos y los guardamos
    datos = io.loadmat("ex5data1.mat")
    trainingX, trainingY = datos ["X"], datos ["y"]
```

```
    validationX, validationY = datos ["Xval"], datos ["yval"]
    testX, testY = datos ["Xtest"], datos ["ytest"]

    #Añadimos la columna de unos
    m = trainingX.shape[0]
    unos = np.ones((m, 1))
    unosX = np.hstack((unos, trainingX))

    theta = np.array([[1], [1]])

    print(costeRegularizado(unosX, trainingY, theta, 1))
    print(gradienteRegularizado(unosX, trainingY, theta, 1))


    theta = entrenamientoMinimizarTheta(unosX, trainingY, 0)
    print(theta)

    pintaRecta(trainingX, trainingY, theta)
```

2. Curvas de aprendizaje

Resumen

Repetimos el entrenamiento por regresión lineal, ahora utilizando diferentes subconjuntos de los datos de entrenamiento. Tras esto hay que evaluar el error del resultado y, además, el error al clasificar los ejemplos del conjunto de validación.

2.1 Curva de aprendizaje

Creamos una función que genera los errores. Para ello usaremos la ecuación:

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

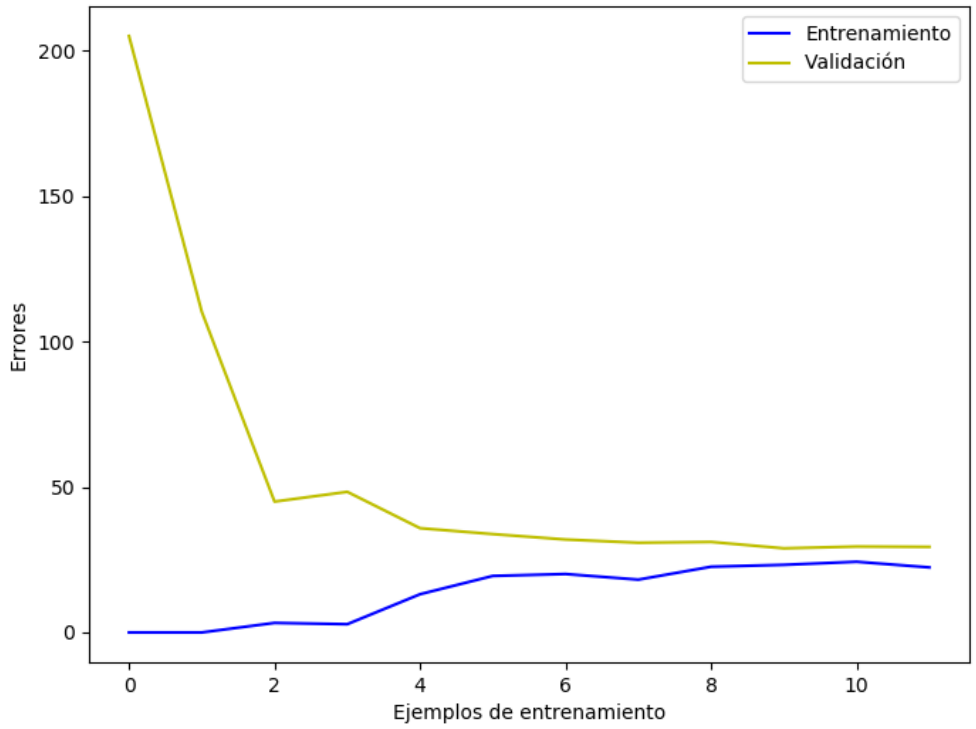
O lo que sería igual, la ecuación del coste regularizado sin el término de regularización, o sea siendo λ = 0.

Se lleva a cabo un bucle para ir cogiendo en cada vuelta subconjuntos distintos y cada vez más grandes de los casos de entrenamiento. Con esos subconjuntos calculamos el valor para θ que minimiza el error y con esto el error resultante ha dicho subconjunto y al clasificar los ejemplos de validación.

```
def curvaAprendizaje(X, Y, valX, valY, lamda):
    m = X.shape[0]
```

```
    errorEnt = np.zeros(m)
    errorVal = np.zeros(m)


    for i in range(0, m):
        j = i + 1
        theta = entrenamientoMinimizarTheta(X[:j], Y[:j], lamda)

        errorEnt[i] = costeYGradiente(X[:j], Y[:j], theta, lamda)[0]
        errorVal[i] = costeYGradiente(valX, valY, theta, lamda)[0]

    return errorEntrenamiento, errorValidation
```

Tras calcular esto se representa en una gráfica.



Código del Ejercicio

```
def Ejercicio2():
    #Carga de los datos
    datos = io.loadmat("ex5data1.mat")
    trainingX, trainingY = datos ["X"], datos ["y"]
    validationX, validationY = datos ["Xval"], datos ["yval"]
    testX, testY = datos ["Xtest"], datos ["ytest"]

    #Añade la columna de unos
    m = trainingX.shape[0]
    unosEntreno = np.ones((m, 1))
    entrenoUnosX = np.hstack((unosEntreno, trainingX))
```

```
    n = validationX.shape[0]
    unosValidation = np.ones((n, 1))
    validationUnosX = np.hstack((unosValidation, validationX))

    #Calcula la curva de aprendizaje
    errorEntrenamiento, errorValidation = curvaAprendizaje(entrenoUnosX,
trainingY, validationUnosX, validationY, 0)


    plt.figure(figsize=(8, 6))
    plt.xlabel('Ejemplos de entrenamiento')
    plt.ylabel('Errores')
    plt.title('Curva de aprendizaje de la regresión lineal')
    plt.plot(range(0,m), errorEntrenamiento, 'b', label='Entrenamiento')
    plt.plot(range(0,m), errorValidation, 'y', label='Validación')
    plt.legend()
    plt.show()
```

3. Regresión polinomial

Resumen

Ahora como hipótesis se usará un polinomio que depende de la entrada X, esto sirve para conseguir un mayor ajuste de los datos de entrenamiento.

3.1. Generación de datos

Generamos nuevos datos a partir de las "X" originales y los guardamos en una matriz. Tendrá el mismo número de filas que la matriz "X" y tantas columnas como nuevos datos queramos generar.

A su vez, los datos están elevados al valor del índice de la columna en la que están, más uno.

```
def generaDatos(X, grado):
    Xres = X
    for i in range(1, grado):
        auxX = np.power(X, I + 1)
        Xres = np.column_stack((Xres, auxX))
    return Xres
```

3.2. Normalización

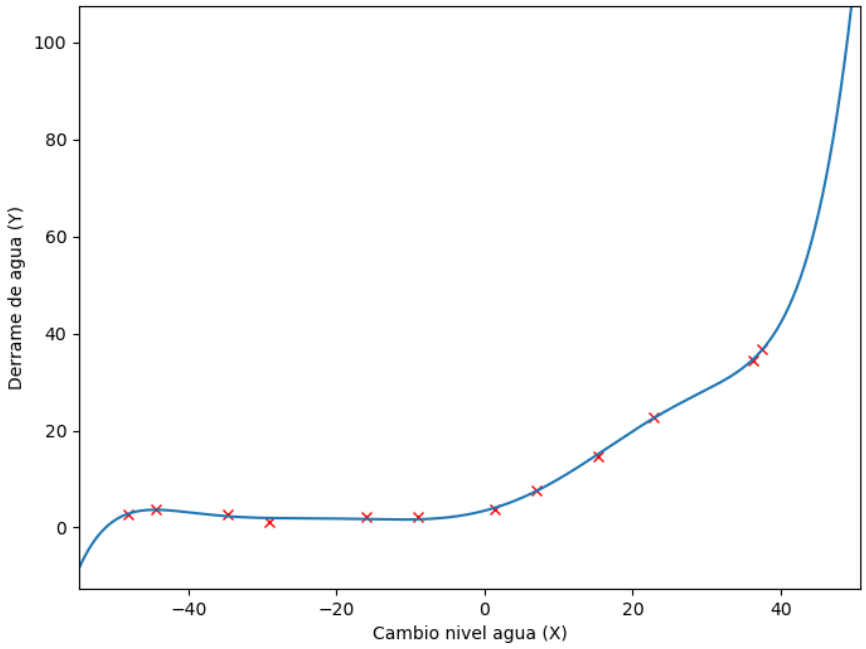
Como al generar datos de esta forma hay unas diferencias de grado muy grandes, normalizamos los datos de la nueva matriz antes de llevar a cabo el aprendizaje de θ.

```
def normaliza(X, mu=None, sigma=None):
    # Nos pasan mu y sigma existentes
```

```
    if(mu is None or sigma is None):
        mu = np.mean(X, axis=0)
        sigma = np.std(X, axis=0)
    # Calculamos el valor normalizado
    X = (X - mu) / sigma
    return X, mu, sigma
```

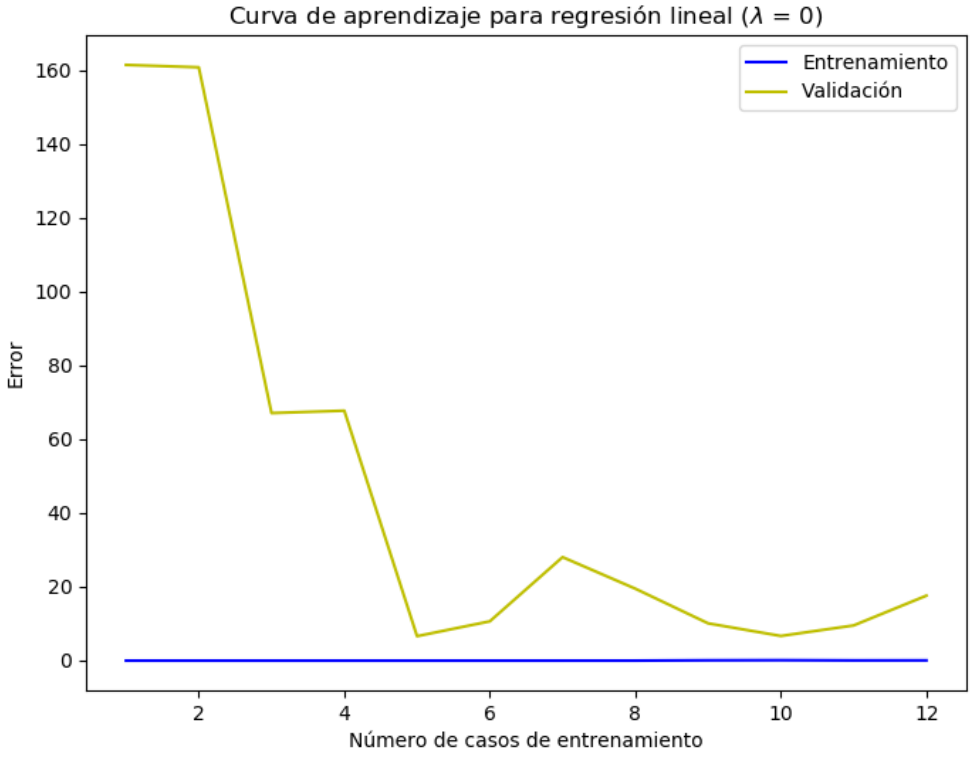
3.3. Cálculo de θ

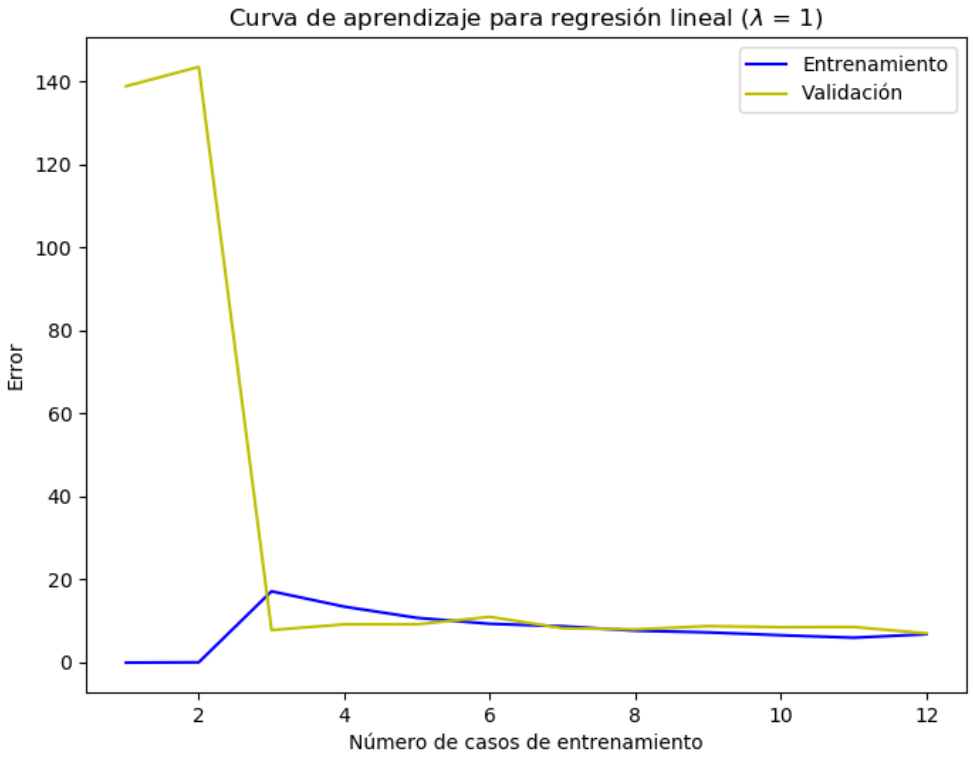
Volvemos a aplicar el método de regresión lineal para obtener el vector θ que minimiza el error, siendo λ = 0, y lo representamos en una gráfica.

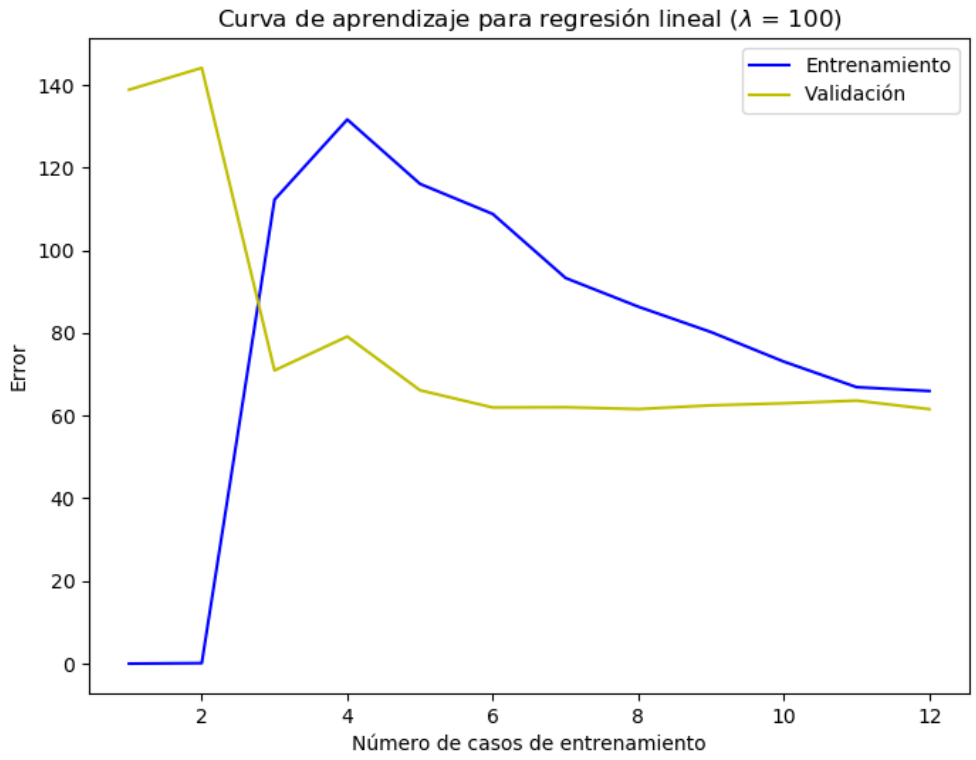


3.3. Curva de aprendizaje validación

Calculamos de nuevo el error tanto para los casos de entrenamiento como para los de validación con los nuevos datos generados y normalizados. Y lo hacemos con tres valores de λ distintos (0, 1 y 100).


Curva de aprendizaje para regresión lineal (λ = 0)
Entrenamiento
Validación
Error
Número de casos de entrenamiento
0
20
40
60
80
100
120
140
160
2
4
6
8
10
12


Curva de aprendizaje para regresión lineal (λ = 1)
Entrenamiento
Validación
Error
Número de casos de entrenamiento
0
20
40
60
80
100
120
140
2
4
6
8
10
12



Código del Ejercicio

```
def Ejercicio3():
    datos = io.loadmat("ex5data1.mat")
    trainingX, trainingY = datos ["X"], datos ["y"]
    validationX, validationY = datos ["Xval"], datos ["yval"]
    testX, testY = datos ["Xtest"], datos ["ytest"]

    #Genera nuevos datos de entrada y los normaliza
    # 1º Casos de entrenamiento
    newX = generaDatos(trainingX, 8)
    normX, mu, sigma = normaliza(newX)

    #2º Casos de validación
    newValX = generaDatos(validationX, 8)
    normValX = newValX - mu
    normValX = normValX / sigma

    #Añade la columna de unos
    m = normX.shape[0]
    unosEntreno = np.ones((m, 1))
    entrenoUnosX = np.hstack((unosEntreno, normX))

    n = normValX.shape[0]
    unosValidacion = np.ones((n, 1))
    validacionUnosX = np.hstack((unosValidacion, normValX))
```

```
    #Calcula el vector theta que minimiza el error con lambda = 0
    theta = entrenamientoMinimizarTheta(entrenoUnosX, trainingY,0)

    # Muestra la curva que se genera
    plt.figure(figsize=(8, 6))
    plt.xlabel('Cambio nivel agua (X)')
    plt.ylabel('Derrame de agua (Y)')
    plt.title('Regresión polinomial ($\lambda$ = 0)')
    plt.plot(trainingX, trainingY, 'rx')
    ajusta(min(trainingX), max(trainingX), mu, sigma, theta, 8)


    #Curvas de aprendizaje
    errorEntreno, errorValidacion = curvaAprendizaje(entrenoUnosX, traini
ngY, validacionUnosX, validationY, 1)
    plt.figure(figsize=(8, 6))
    plt.xlabel('Número de casos de entrenamiento')
    plt.ylabel('Error')
    plt.title('Curva de aprendizaje para regresión lineal ($\lambda$ = 1)
')
    plt.plot(range(1,m+1), errorEntreno, 'b', label='Entrenamiento')
    plt.plot(range(1,m+1), errorValidacion, 'y', label='Validación')
    plt.legend()
    plt.show()
```

4. Selección del parámetro λ

Resumen

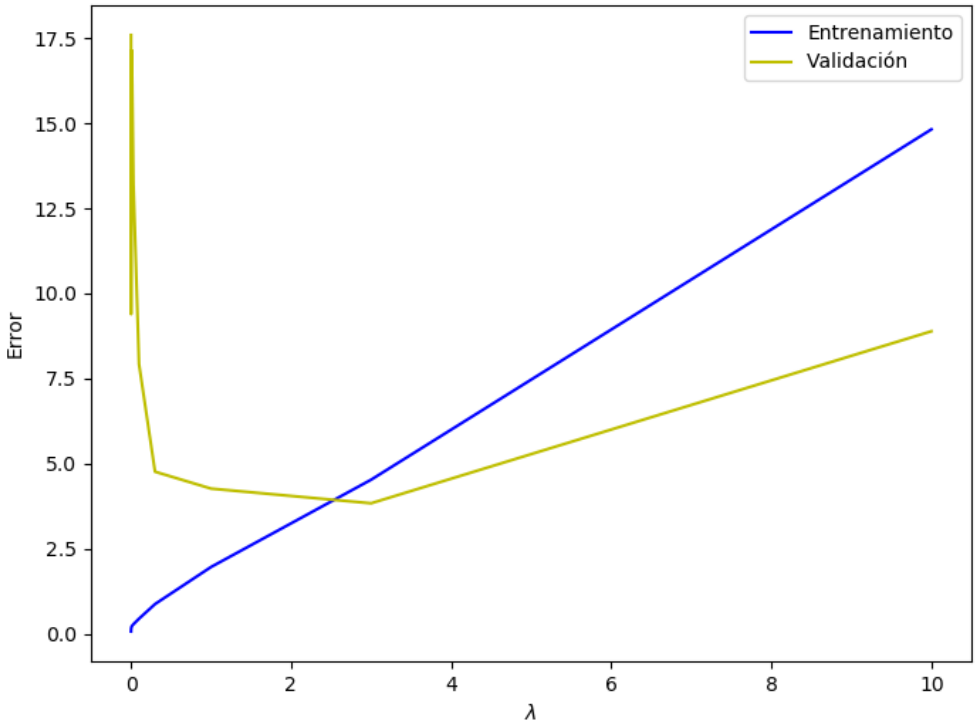
Encontrar el mejor valor para el parámetro de regularización, λ.

4.1 Curva de validación

Se calcula θ con distintos valores de λ y con ese resultado se calcula el error tanto en los casos de entrenamiento como en los de validación.

```
def curvaValidacion(X, Y, validationX, validationY):
    lambdas = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10])
    errorEntreno = np.zeros((len(lambdas), 1))
    errorValidacion = np.zeros((len(lambdas), 1))

    # Recorremos el vector de lambdas.
    for i in range(len(lambdas)):
        lamda = lambdas[i]

        # Minimizamos el valor de theta con el valor actual de lambda
```

```
        theta = entrenamientoMinimizarTheta(X, Y, lamda)

        # Calculamos el error con lambda a 0
        errorEntreno[i] = costeYGradiente(X, Y, theta, 0)[0]
        errorValidacion[i] = costeYGradiente(validationX, validationY, th
eta, 0)[0]

    return lambdas, errorEntreno, errorValidacion
```



Con la gráfica deducimos que el mejor valor para λ es 3.

4.2 Error valores de test

Por último, se calcula el error para el último conjunto de datos, los de *test*. Para estos también generamos tantos datos como para los de entrenamiento y validación y los normalizamos. Una vez hecho esto y añadida la columna de unos, calculamos el error.

```
Error de los valores de Test para el mejo lambda: 3.5720
```

Código del Ejercicio

```
def Ejercicio4():
    datos = io.loadmat("ex5data1.mat")
    trainingX, trainingY = datos ["X"], datos ["y"]
    validationX, validationY = datos ["Xval"], datos ["yval"]
    testX, testY = datos ["Xtest"], datos ["ytest"]

    #Genera nuevos datos de entrada y los normaliza
```

```
    # 1º Casos de entrenamiento
    newX = generaDatos(trainingX, 8)
    normX, mu, sigma = normaliza(newX)

    #2º Casos de validación
    newValX = generaDatos(validationX, 8)
    normValX = newValX - mu
    normValX = normValX / sigma

    #3º Casos de test
    newTestX = generaDatos(testX, 8)
    normTestX = newTestX - mu
    normTestX = normTestX / sigma

    #Añade la columna de unos
    m = normX.shape[0]
    unosEntreno = np.ones((m, 1))
    entrenoUnosX = np.hstack((unosEntreno, normX))

    n = normValX.shape[0]
    unosValidacion = np.ones((n, 1))
    validacionUnosX = np.hstack((unosValidacion, normValX))

    o = normTestX.shape[0]
    unosTest = np.ones((o, 1))
    testUnosX = np.hstack((unosTest, normTestX))

    #Calcula el vector theta que minimiza el error con lambda = 0
    lambdas, errorEntreno, errorValidacion = curvaValidacion(entrenoUnosX
, trainingY, validacionUnosX, validationY)


    plt.figure(figsize=(8, 6))
    plt.xlabel('$\lambda$')
    plt.ylabel('Error')
    plt.title('Selección del parámetro $\lambda$')
    plt.plot(lambdas, errorEntreno, 'b', label='Entrenamiento')
    plt.plot(lambdas, errorValidacion, 'y', label='Validación')
    plt.legend()
    plt.show()

    theta = entrenamientoMinimizarTheta(entrenoUnosX, trainingY, 3)

    errorTest = costeYGradiente(testUnosX, testY, theta, 0)[0]
    print("Error de los valores de Test para el mejor lambda: {0:.4f}".fo
rmat(errorTest))
```