Aprendizaje Automático y minería de datos: Práctica 5

Autores: *Tatiana Duarte Balvís, Miguel Mur Cortés*

1. Objetivo:

El objetivo de esta práctica es comprobar los efectos de sesgo y varianza.

Se aplicará regresión lineal regularizada para aprender una hipótesis sesgada, que no es capaz de clasificar correctamente a los ejemplos de entrenamiento con una curva de aprendizaje para observar el sesgo producido.

En siguiente lugar, se implementará una regresión polinomial con sus curvas de aprendizaje asociadas.

Por último, ajustaremos los valores λ para averiguar el mejor grado de ajuste a los ejemplos de entrenamiento.

2. Regresión lineal regularizada:

En este apartado, se aplicará el método de regresión lineal regularizada sobre una base de datos.

Primero, realizamos la carga de datos y obtenemos los datos de entrenamiento.

```
datos = loadmat("ex5data1.mat")
    X, Xval, Xtest = datos["X"], datos["Xval"], datos["Xtest"]
    y, yval, ytest = datos["y"].ravel(), datos["yval"].ravel(),
datos["ytest"].ravel()
```

A continuación, implementamos dos funciones para obtener el coste y el gradiente de la regresión lineal regularizada.

```
def coste(Theta, X, y, lamda):
    H = np.dot(X, Theta)
    op1 = np.sum((H - y)**2) / (2 * np.shape(X)[0])
    op2 = lamda * np.sum(Theta[1:]**2) / (2 * np.shape(X)[0])
    return op1 + op2


def gradiente(Theta, X, y, lamda):
    H = np.dot(X, Theta)
    op1 = np.dot((H - y), X) / np.shape(X)[0]
    op2 = lamda * Theta[1:] / np.shape(X)[0]
    op1[1:] += op2
    return op1
```

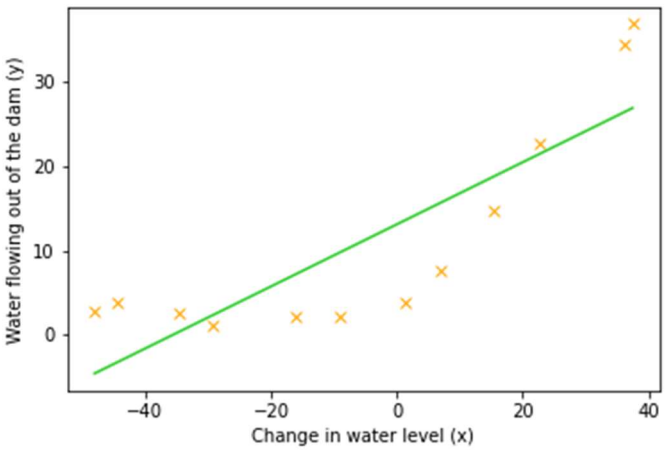
Para un valor de λ = 1 y θ = [-1, 1], se espera un coste de 303,993 y un gradiente de [-15,303;598,250]

```
Theta = np.array([1,1])
X_aux = np.hstack([np.ones([np.shape(X)[0], 1]), X])

lamda = 1
coste_ini = coste(Theta, X_aux, y, lamda)
gradiente_ini = gradiente(Theta, X_aux, y, lamda)
print("---Regresión lineal regularizada---")
print(f"Coste inicial: {str(coste_ini)[:7]}")
print(f"Gradiente inicial: [{str(gradiente_ini[0])[:7]};
{str(gradiente_ini[1])[:7]}]")
print()

Resultado:
---Regresión lineal regularizada---
Coste inicial: 303.993
Gradiente inicial: [-15.303; 598.250]
```

A continuación, encontramos el valor θ para minimizar el error sobre los ejemplos de entrenamiento.

```
#Calculo del valor de Theta que minimiza el error sobre los ejemplos de
entrenamiento
lamda = 0
fmin = opt.minimize(fun=coste, x0=Theta, args=(X_aux, y, lamda))
grafica_regresion_lineal_reg(fmin.x, X, y, nombrePlot)
```

Y representamos la figura:



```
def grafica_regresion_lineal_reg(Theta, X, y, nombrePlot):
    plt.figure()
    plt.plot(X, y, "x", c="orange")

    #Linea
    minX = np.amin(X)
    maxX = np.amax(X)
    minY = Theta[0] + Theta[1] * minX
    maxY = Theta[0] + Theta[1] * maxX
    plt.plot([minX, maxX], [minY, maxY], c="limegreen")

    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.savefig(nombrePlot + ".png")
```

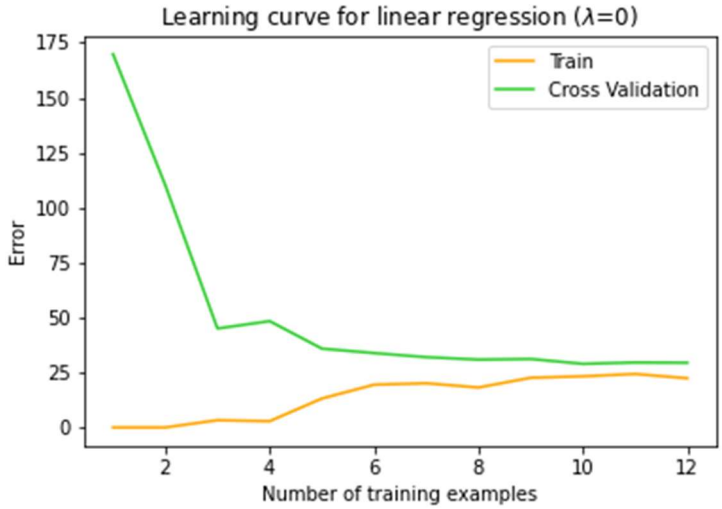
3. Curvas de aprendizaje:

El problema del apartado anterior es que la representación elegida por las hipótesis es demasiado simple para ajustarse a los datos de entrenamiento. Para ello, la representación que se utilizará será una curva de aprendizaje para identificar situaciones de sesgo y varianza.

Para ello, repetimos el entrenamiento por regresión lineal utilizando diferentes subconjuntos de los datos de entrenamiento. Con cada resultado, evaluamos el error aplicado sobre ese mismo subconjunto y el error al clasificar a todos los ejemplos del conjunto de validación (Xval, Yval)

```
def error(Theta, X, y):
    H = np.dot(X, Theta)
    return np.sum((H - y)**2) / (2 * np.shape(X)[0])


def curvas_aprendizaje(X, Xval, y, yval, lamda, nombrePlot):
    m = np.shape(X)[0]
    X_aux = np.hstack([np.ones([m, 1]), X])
    Xval_aux = np.hstack([np.ones([np.shape(Xval)[0], 1]), Xval])
    Theta = np.ones(np.shape(X_aux)[1])

    errors = []
    errorsval = []

    for i in range(1, m+1):
        fmin = opt.minimize(fun=coste, x0=Theta, args=(X_aux[:i], y[:i], lamda))
        errors.append(error(fmin.x, X_aux[:i], y[:i]))
        errorsval.append(error(fmin.x, Xval_aux, yval))
```

Con los datos obtenidos, representamos la figura evaluando la evolución del nivel de error.



4. Regresión polinomial:

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

Para ello, implementamos una función que genera nuevos datos de entrenamiento a partir de los datos originales X de dimensión $m * 1$ y un número p. Esta función devuelve una matriz de dimensión $m * p$ que en la primera columna contiene los valores de X, en la segunda X^2, en la tercera X^3 …

```
def genera_datos(X, p):
    H = np.empty((np.shape(X)[0], p))

    for i in range(0, p):
        H[:,i] = (X**(i+1)).ravel()

    #print(np.shape(H))
    return H
```

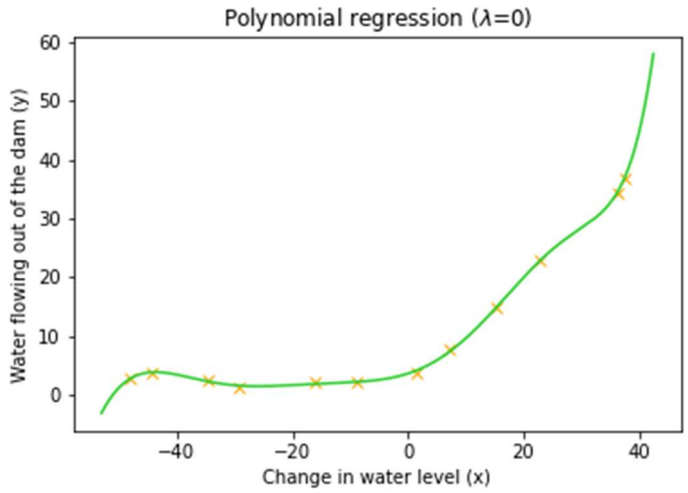
El problema de esta aproximación es que generaremos valores demasiado grandes, por lo que necesitamos normalizar la matriz.

```
def normaliza_matriz(X):
    mu = np.mean(X, axis = 0)
    sigma = np.std(X, axis = 0)
    X_norm = (X - mu)/sigma
    return X_norm, mu, sigma
```

A continuación, generamos de esta forma los nuevos datos de entrada para aprender un polinomio de grado $p = 8$ y volvemos a aplicar el método de regresión lineal para obtener un vector θ que minimiza el error.

```
def regresion_polinomial(X, Xval, y, yval, lamda, p, nombrePlot1,
nombrePlot2):
    X_norm, mu, sigma = normaliza_matriz(genera_datos(X, p))
    X_norm = np.hstack([np.ones([np.shape(X_norm)[0], 1]), X_norm])
    Theta = np.ones(np.shape(X_norm)[1])
    fmin = opt.minimize(fun=coste, x0=Theta, args=(X_norm, y, lamda))
```

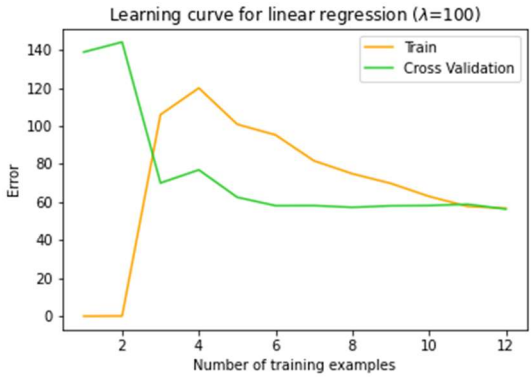
Para generar la curva, seleccionamos los valores y para un conjunto de valores x entre el mínimo y el máximo de los valores de entrenamiento X, a intervalos de $0{,}05$.
Para un valor de $\lambda = 0$, obtenemos esta curva.

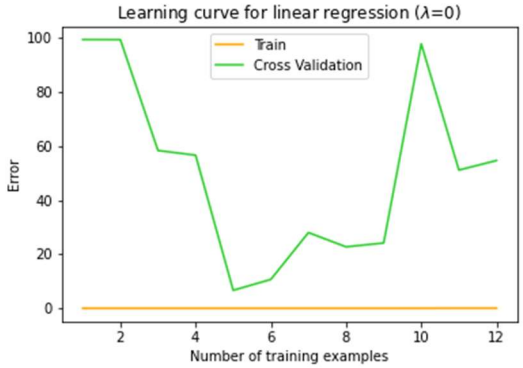


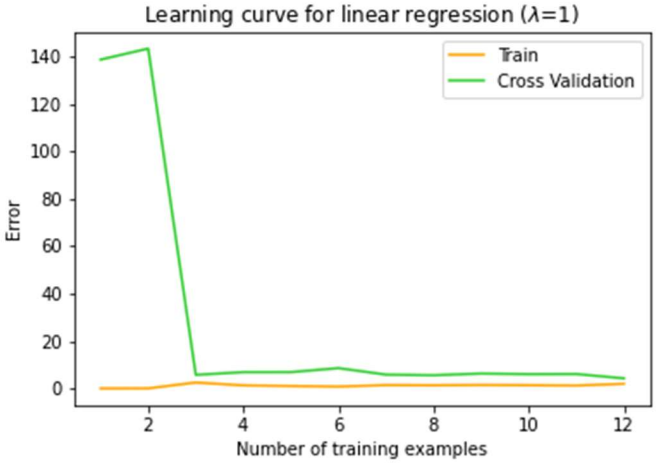

```
def grafica_regresion_polinomial(Theta, X, y, lamda, p, mu,
sigma, nombrePlot):
    plt.figure()
    plt.plot(X, y, "x", c="orange")

    #Linea
    lineX = np.arange(np.amin(X)-5, np.amax(X)+5, 0.05)
    auxX = genera_datos(lineX, p)
    auxX = (auxX - mu)/sigma #Normalizamos
    auxX = np.hstack([np.ones([np.shape(auxX)[0], 1]), auxX])
    lineY = np.dot(auxX, Theta)
    plt.plot(lineX, lineY, c="limegreen")

    plt.title(f"Polynomial regression ($\lambda$={lamda})")
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.savefig(nombrePlot + ".png")
```

A continuación, generamos las curvas de aprendizaje para la hipótesis polinomial del mismo modo que realizamos las del apartado anterior. Presentaremos una gráfica para los valores λ [1, 10, 100]

```
def regresion_polinomial(X, Xval, y, yval, lamda, p, nombrePlot1,
nombrePlot2):
    X_norm, mu, sigma = normaliza_matriz(genera_datos(X, p))
    X_norm = np.hstack([np.ones([np.shape(X_norm)[0], 1]), X_norm])
    Theta = np.ones(np.shape(X_norm)[1])

    fmin = opt.minimize(fun=coste, x0=Theta, args=(X_norm, y, lamda))
    grafica_regresion_polinomial(fmin.x, X, y, lamda, p, mu, sigma, nombrePlot1)

    Xval_norm = (genera_datos(Xval, p) - mu)/sigma #Normalizamos

    curvas_aprendizaje(X_norm[:,1:], Xval_norm, y, yval, lamda, nombrePlot2)
    curvas_aprendizaje(X_norm[:,1:], Xval_norm, y, yval, 1, nombrePlot2 + "-1")
    curvas_aprendizaje(X_norm[:,1:], Xval_norm, y, yval, 100, nombrePlot2 + "-2")
```







5. Elección del parámetro λ:

La técnica que utilizaremos para elegir el valor de λ es evaluar la hipótesis generada sobre los ejemplos de entrenamiento con un segundo conjunto de ejemplos de validación y seleccionar aquel valor de λ que minimice el error λ.

Aplicamos esta técnica con los valores $\lambda = \{0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10\}$ y dibujamos una gráfica con los valores de error para los ejemplos de entrenamiento y los de validación.

```
def seleccion_parametro_lamda(X, Xval, Xtest, y, yval, ytest, p,
nombrePlot):
    lamdas = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]

    X_norm, mu, sigma = normaliza_matriz(genera_datos(X, p))
    X_norm = np.hstack([np.ones([np.shape(X_norm)[0], 1]), X_norm])
    Theta = np.ones(np.shape(X_norm)[1])

    Xval_norm = (genera_datos(Xval, p) - mu)/sigma #Normalizamos
    Xval_norm = np.hstack([np.ones([np.shape(Xval_norm)[0], 1]), Xval_norm])

    errors = []
    errorsval = []

    for lamda in lamdas:
        fmin = opt.minimize(fun=coste, x0=Theta, args=(X_norm, y, lamda))
        errors.append(error(fmin.x, X_norm, y))
        errorsval.append(error(fmin.x, Xval_norm, yval))

    grafica_seleccion_lamda(lamdas, errors, errorsval, nombrePlot)

def grafica_seleccion_lamda(lamdas, errors, errorsval, nombrePlot):
    plt.figure()

    plt.plot(lamdas, errors, c="orange", label="Train")
    plt.plot(lamdas, errorsval, c="limegreen", label="Cross
Validation")

    plt.legend()
    plt.title("Selecting $\lambda$ using a cross validation set")
    plt.xlabel("lambda")
    plt.ylabel("Error")
    plt.savefig(nombrePlot + ".png")
```

En último lugar, estimamos el error de la hipótesis aplicándola a un tercer conjunto de ejemplos que no hemos utilizado para entrenar ni seleccionar λ. Calculando el error sobre los datos de prueba Xtest e ytest, para λ = 3 deberíamos obtener un error en torno a 3,572.

```
def estimar_error(X_norm, Xtest, y, ytest, lamda, p, mu, sigma):
    Theta = np.ones(np.shape(X_norm)[1])

    Xtest_norm = genera_datos(Xtest, p)
    Xtest_norm = (Xtest_norm - mu)/sigma #Normalizamos
    Xtest_norm = np.hstack([np.ones([np.shape(Xtest_norm)[0], 1]),
Xtest_norm])

    fmin = opt.minimize(fun=coste, x0=Theta, args=(X_norm, y, lamda))
    errorLamda = error(fmin.x, Xtest_norm, ytest)

    print(f"Error obtenido para lambda = {lamda}:
{str(errorLamda)[:5]}")

Output: Error obtenido para lambda = 3: 3.572
```

6. Código:

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


def coste(Theta, X, y, lamda):
    H = np.dot(X, Theta)
    op1 = np.sum((H - y)**2) / (2 * np.shape(X)[0])
    op2 = lamda * np.sum(Theta[1:]**2) / (2 * np.shape(X)[0])
    return op1 + op2


def gradiente(Theta, X, y, lamda):
    H = np.dot(X, Theta)
    op1 = np.dot((H - y), X) / np.shape(X)[0]
    op2 = lamda * Theta[1:] / np.shape(X)[0]
    op1[1:] += op2
    return op1


def regresion_lineal_reg(X, y, lamda, nombrePlot):
    Theta = np.array([1,1])
    X_aux = np.hstack([np.ones([np.shape(X)[0], 1]), X])

    #Comprobar si son correctos los calculos
```

```
    lamda = 1
    coste_ini = coste(Theta, X_aux, y, lamda)
    gradiente_ini = gradiente(Theta, X_aux, y, lamda)
    print("---Regresión lineal regularizada---")
    print(f"Coste inicial: {str(coste_ini)[:7]}")
    print(f"Gradiente inicial: [{str(gradiente_ini[0])[:7]};
{str(gradiente_ini[1])[:7]}]")
    print()

    #Calculo del valor de Theta que minimiza el error sobre los ejemplos de
entrenamiento
    lamda = 0
    fmin = opt.minimize(fun=coste, x0=Theta, args=(X_aux, y, lamda))
    grafica_regresion_lineal_reg(fmin.x, X, y, nombrePlot)

    return fmin.x


def grafica_regresion_lineal_reg(Theta, X, y, nombrePlot):
    plt.figure()
    plt.plot(X, y, "x", c="orange")

    #Linea
    minX = np.amin(X)
    maxX = np.amax(X)
    minY = Theta[0] + Theta[1] * minX
    maxY = Theta[0] + Theta[1] * maxX
    plt.plot([minX, maxX], [minY, maxY], c="limegreen")

    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.savefig(nombrePlot + ".png")

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def error(Theta, X, y):
    H = np.dot(X, Theta)
    return np.sum((H - y)**2) / (2 * np.shape(X)[0])


def curvas_aprendizaje(X, Xval, y, yval, lamda, nombrePlot):
    m = np.shape(X)[0]
    X_aux = np.hstack([np.ones([m, 1]), X])
    Xval_aux = np.hstack([np.ones([np.shape(Xval)[0], 1]), Xval])
    Theta = np.ones(np.shape(X_aux)[1])

    errors = []
    errorsval = []
```

```
    for i in range(1, m+1):
        fmin = opt.minimize(fun=coste, x0=Theta, args=(X_aux[:i], y[:i],
lamda))
        errors.append(error(fmin.x, X_aux[:i], y[:i]))
        errorsval.append(error(fmin.x, Xval_aux, yval))

    grafica_curvas_aprendizaje(m, errors, errorsval, lamda, nombrePlot)


def grafica_curvas_aprendizaje(m, errors, errorsval, lamda, nombrePlot):
    plt.figure()

    plt.plot(range(1, m+1), errors, c="orange", label="Train")
    plt.plot(range(1, m+1), errorsval, c="limegreen", label="Cross
Validation")

    plt.legend()
    plt.title(f"Learning curve for linear regression ($\lambda$={lamda})")
    plt.xlabel("Number of training examples")
    plt.ylabel("Error")
    plt.savefig(nombrePlot + ".png")

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def normaliza_matriz(X):
    mu = np.mean(X, axis = 0)
    sigma = np.std(X, axis = 0)
    X_norm = (X - mu)/sigma
    return X_norm, mu, sigma


def genera_datos(X, p):
    H = np.empty((np.shape(X)[0], p))

    for i in range(0, p):
        H[:,i] = (X**(i+1)).ravel()

    #print(np.shape(H))
    return H


def regresion_polinomial(X, Xval, y, yval, lamda, p, nombrePlot1,
nombrePlot2):
    X_norm, mu, sigma = normaliza_matriz(genera_datos(X, p))
    X_norm = np.hstack([np.ones([np.shape(X_norm)[0], 1]), X_norm])
    Theta = np.ones(np.shape(X_norm)[1])

    fmin = opt.minimize(fun=coste, x0=Theta, args=(X_norm, y, lamda))
```

```
    grafica_regresion_polinomial(fmin.x, X, y, lamda, p, mu, sigma,
nombrePlot1)

    Xval_norm = (genera_datos(Xval, p) - mu)/sigma #Normalizamos

    curvas_aprendizaje(X_norm[:,1:], Xval_norm, y, yval, lamda, nombrePlot2)
    curvas_aprendizaje(X_norm[:,1:], Xval_norm, y, yval, 1, nombrePlot2 + "-
1")
    curvas_aprendizaje(X_norm[:,1:], Xval_norm, y, yval, 100, nombrePlot2 + "-
2")


def grafica_regresion_polinomial(Theta, X, y, lamda, p, mu, sigma,
nombrePlot):
    plt.figure()
    plt.plot(X, y, "x", c="orange")

    #Linea
    lineX = np.arange(np.amin(X)-5, np.amax(X)+5, 0.05)
    auxX = genera_datos(lineX, p)
    auxX = (auxX - mu)/sigma #Normalizamos
    auxX = np.hstack([np.ones([np.shape(auxX)[0], 1]), auxX])
    lineY = np.dot(auxX, Theta)
    plt.plot(lineX, lineY, c="limegreen")

    plt.title(f"Polynomial regression ($\lambda$={lamda})")
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.savefig(nombrePlot + ".png")

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def estimar_error(X_norm, Xtest, y, ytest, lamda, p, mu, sigma):
    Theta = np.ones(np.shape(X_norm)[1])

    Xtest_norm = genera_datos(Xtest, p)
    Xtest_norm = (Xtest_norm - mu)/sigma #Normalizamos
    Xtest_norm = np.hstack([np.ones([np.shape(Xtest_norm)[0], 1]),
Xtest_norm])

    fmin = opt.minimize(fun=coste, x0=Theta, args=(X_norm, y, lamda))
    errorLamda = error(fmin.x, Xtest_norm, ytest)

    print(f"Error obtenido para lambda = {lamda}: {str(errorLamda)[:5]}")


def seleccion_parametro_lamda(X, Xval, Xtest, y, yval, ytest, p, nombrePlot):
    lamdas = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
```

```
    X_norm, mu, sigma = normaliza_matriz(genera_datos(X, p))
    X_norm = np.hstack([np.ones([np.shape(X_norm)[0], 1]), X_norm])
    Theta = np.ones(np.shape(X_norm)[1])

    Xval_norm = (genera_datos(Xval, p) - mu)/sigma #Normalizamos
    Xval_norm = np.hstack([np.ones([np.shape(Xval_norm)[0], 1]), Xval_norm])

    errors = []
    errorsval = []

    for lamda in lamdas:
        fmin = opt.minimize(fun=coste, x0=Theta, args=(X_norm, y, lamda))
        errors.append(error(fmin.x, X_norm, y))
        errorsval.append(error(fmin.x, Xval_norm, yval))

    grafica_seleccion_lamda(lamdas, errors, errorsval, nombrePlot)

    #Sabemos que la lamda mejor es 3
    estimar_error(X_norm, Xtest, y, ytest, 3, p, mu, sigma)


def grafica_seleccion_lamda(lamdas, errors, errorsval, nombrePlot):
    plt.figure()

    plt.plot(lamdas, errors, c="orange", label="Train")
    plt.plot(lamdas, errorsval, c="limegreen", label="Cross Validation")

    plt.legend()
    plt.title("Selecting $\lambda$ using a cross validation set")
    plt.xlabel("lambda")
    plt.ylabel("Error")
    plt.savefig(nombrePlot + ".png")

#++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

def main():
    datos = loadmat("ex5data1.mat")
    X, Xval, Xtest = datos["X"], datos["Xval"], datos["Xtest"]
    y, yval, ytest = datos["y"].ravel(), datos["yval"].ravel(),
datos["ytest"].ravel()

    lamda = 0
    p = 8

    regresion_lineal_reg(X, y, lamda, "figura1")
    curvas_aprendizaje(X, Xval, y, yval, lamda, "figura2")
    regresion_polinomial(X, Xval, y, yval, lamda, p, "figura3", "figura4")
    seleccion_parametro_lamda(X, Xval, Xtest, y, yval, ytest, p, "figura5")
```

```
main()
```