Práctica 5

Antonio Rodríguez Hurtado, Miguel Ferreras Chumillas

Primero se importan las librerias y se cargan los datos necesarios para la practica

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


#   Cargamos los datos
data = loadmat("ex5data1.mat")
X = data["X"]
y = data["y"].ravel()
Xtest = data["Xtest"]
ytest = data["ytest"].ravel()
Xval = data["Xval"]
yval = data["yval"].ravel()

m = np.shape(X)[0]
n = np.shape(X)[1]

OX = np.hstack([np.ones([m,1]),X])
OXval = np.hstack([np.ones([np.shape(Xval)[0], 1]), Xval])
```

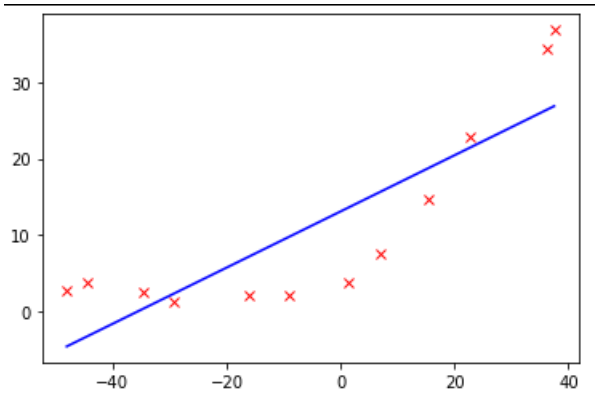
Regresión lineal regularizada

En primer lugar definimos las funciones de hipotesis, coste y gradiente y calculamos con ellas las thetas optimas para la regresion lineal.

```
theta = np.array([1] * (n + 1))

#   Definimos la funcion de hipotesis, el coste y el gradiente
def hip(X, theta):
    return np.dot(X, theta)

def cost(theta, X, Y, reg):
    m = np.shape(X)[0]
    return (1 / (2 * m)) * np.sum(((hip(X, theta) - Y) ** 2)) + ((reg / (2 * m)) * np.sum(theta[1:] ** 2))

def gradient(theta, X, Y, reg):
    gradiente = (1 / m) * np.dot((hip(X, theta) - Y), X)
    result = np.zeros(np.shape(X)[1])
    result[0] = gradiente[0]
    result[1:] = gradiente[1:] + (theta[1:] * (reg / m))
    return  result

#   Se grafica la recta de la regresion lineal
def grafica_regresion_lineal(X,Y,theta):
    plt.plot(X,Y, "x", color="red")
    min_x = min(X)
    max_x = max(X)
    min_y = theta[0] + (theta[1]*min_x)
    max_y = theta[0] + (theta[1]*max_x)
    plt.plot([min_x, max_x], [min_y, max_y], color="blue")
```

```
#   Se calcula la regresion con un termino regulatorio de 1
reg = 1
theta_opt = minimize(fun=cost, x0=theta, args=(OX, y, reg), jac=gradient).x
plt.figure(1)
grafica_regresion_lineal(X,y,theta_opt)
```

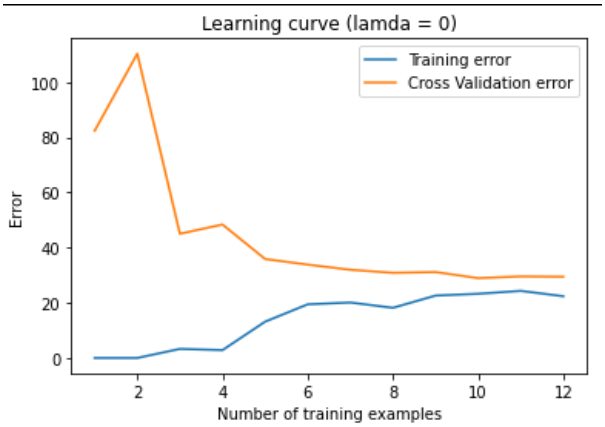
La grafica de la recta resultante es la siguiente:



Curva de aprendizaje

Para calcular la curva de aprendizaje creamos una funcion donde se calcula la theta optima de cada conjunto de entrenamiento X[:i] y se guarda el error que produce tanto en el propio conjunto como en el de validacion

```
def learning_curve(theta, X, Y, Xval, yval,reg):
    cross_validation_error = []
    train_error = []
    n_training_sets = []
    for i in range (1, m + 1):
        theta_opt = minimize(fun=cost, x0=theta, args=(X[:i], y[:i], reg), jac=gradient).x
        cross_validation_error.append(cost(theta_opt,Xval,yval,reg))
        train_error.append(cost(theta_opt,X[:i],y[:i],reg))
        n_training_sets.append(i)
    return n_training_sets, train_error, cross_validation_error


#    Se grafica la curva de aprendizaje
def grafica_learning_curve(n_training_sets,train_error,cross_validation_error, reg):
    plt.plot(n_training_sets,train_error,label='Training error')
    plt.plot(n_training_sets,cross_validation_error,label='Cross Validation error')
    plt.xlabel('Number of training examples')
    plt.ylabel('Error')
    plt.title('Learning curve (lamda = ' + str(reg) + ')')
    plt.legend()
    plt.show()


#   Se calculan los errores con un coeficiente de regulacion de 0
reg = 0
n_training_sets, train_error, cross_validation_error = learning_curve(theta_opt, OX, y, OXval, yval, reg)
plt.figure(2)
grafica_learning_curve(n_training_sets, train_error, cross_validation_error, reg)
```



En esta curca se aprecia como a mayor numero de conjuntos de datos mas se aproximan los errores de entrenamiento y validacion, lo que indica un sobreajuste.

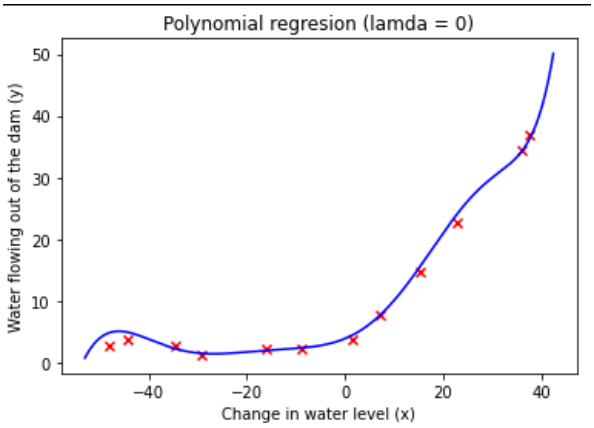
Regresión polinomial

En esta parte de la practica se convierten los datos de entrenamiento en polinomios de 'p' terminos con la funcion nuevos atributos y se normalizan.

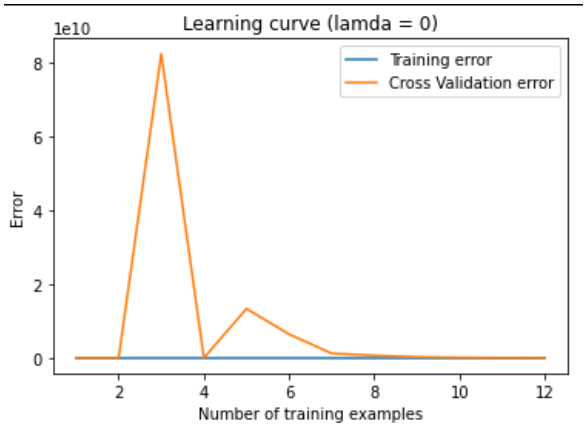
```
#    Convierte una matriz de m*1 en una de m*p
def nuevos_atributos(matrix, columnas):
    for i in range(2, columnas + 1):
        aux = matrix.T[0]**i
        aux = aux.reshape((matrix.shape[0], 1))
        matrix = np.hstack((matrix,aux))
    return matrix

#   Normaliza la matriz
def normalize(X):
    mean = np.mean(X, axis=0)
    X_mean = X - mean
    std = np.std(X, axis=0)
    X_norm = X_mean / std
    return X_norm

def grafica_polinomial(X_pol, Y_pol, theta):
    plt.scatter(X, y, color='red', marker='x')
    plt.plot(X_pol, Y_pol, color='blue')
    plt.xlabel("Change in water level (x)")
    plt.ylabel("Water flowing out of the dam (y)")
    plt.title("Polynomial regresion (lamda = 0)")
    plt.show()

#   Se establecen los valores de reg y p
reg = 0
p = 8

#   Se obtienen las thetas que minimizan el error con los datos de entrenamiento como polinomio
X_pol = nuevos_atributos(X,p)
X_pol = normalize(X_pol)
X_pol = np.hstack([np.ones([np.shape(X_pol)[0], 1]), X_pol])

theta = np.array([1] * (X_pol.shape[1]))
theta_opt = minimize(fun=cost, x0=theta, args=(X_pol, y, reg), jac=gradient).x

#   Se genera un conjunto de datos entre el minimo y el maximo de X separados por 0.05
x_new_data = np.arange(min(X.ravel())-5, max(X.ravel())+5, 0.05)
y_new_data = []

x_aux = np.reshape(x_new_data,(-1, 1))
```

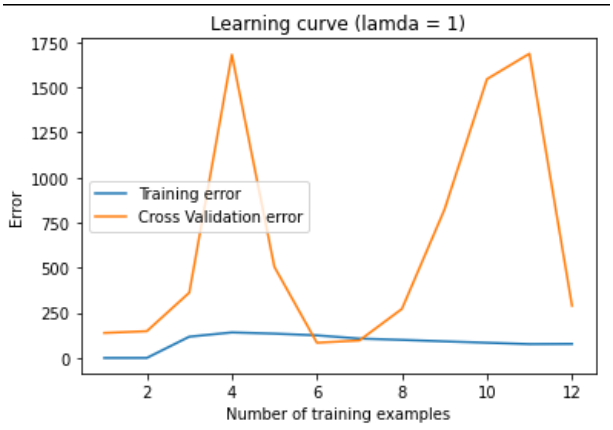
```
x_aux = nuevos_atributos(x_aux, p)
x_aux = normalize(x_aux)
x_aux = np.hstack([np.ones([np.shape(x_aux)[0], 1]), x_aux])

#   Se calcula la hipotesis para ese nuevo conjunto con las thetas calculadas anteriormente
y_new_data = hip(x_aux, theta_opt)

grafica_polinomial(x_new_data, y_new_data, theta_opt)

#   A continuacion se compara el error con el conjunto de validacion para lamda 0, 1 y 100
Xval_pol = nuevos_atributos(Xval,p)
Xval_pol = normalize(Xval_pol)
Xval_pol = np.hstack([np.ones([np.shape(Xval_pol)[0], 1]), Xval_pol])

n_training_sets, train_error, cross_validation_error = learning_curve(theta_opt, x_aux, y, Xval_pol, yval, reg)
grafica_learning_curve(n_training_sets, train_error, cross_validation_error, reg)

reg = 1
n_training_sets, train_error, cross_validation_error = learning_curve(theta_opt, x_aux, y, Xval_pol, yval, reg)
grafica_learning_curve(n_training_sets, train_error, cross_validation_error, reg)

reg = 100
n_training_sets, train_error, cross_validation_error = learning_curve(theta_opt, x_aux, y, Xval_pol, yval, reg)
grafica_learning_curve(n_training_sets, train_error, cross_validation_error, reg)
```

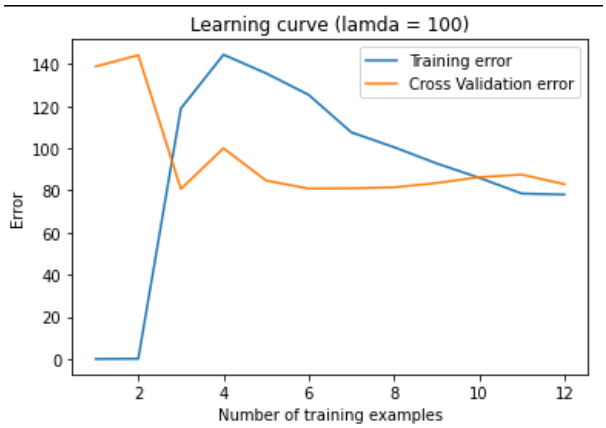
La funcion polinomica tiene este aspecto:



Graficas resultantes de las curvas de aprendizaje con distintas lamdas:







Selección de lamda

En este apartado probamos el error que produce el uso de distintas lamdas.

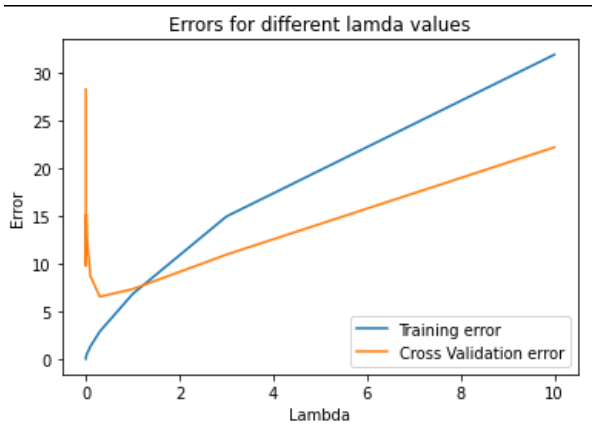
```
def lamdas_error(n, training_y, crossval_y):
    plt.plot(n, training_y, label='Training error')
    plt.plot(n, crossval_y, label='Cross Validation error')
    plt.xlabel('Lambda')
    plt.ylabel('Error')
    plt.title('Errors for different lamda values')
    plt.legend()
    plt.show()

lamdas = [0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]
X_error = []
Val_error = []
Total_error = []
#   Seleccionamos el lambda que da menor error entre los ejemplos de entrenamiento
# y los datos de validacion
for e in lamdas:
    theta_opt = minimize(fun=cost, x0=theta, args=(X_pol, y, e), jac=gradient).x
    Xe = cost(theta_opt, X_pol, y, e)
    Ve = cost(theta_opt, Xval_pol, yval, e)
    X_error.append(Xe)
    Val_error.append(Ve)
    Total_error.append(np.abs(Xe - Ve))
    print('lamda: ', e,' Error: ', Xe,'Validation error: ', Ve)

i = np.argmin(Total_error)
min_lambda = lamdas[i];
print("Menor lambda: ", min_lambda)

lamdas_error(lamdas, X_error, Val_error)
```

```
Xtest_pol = nuevos_atributos(Xtest ,8)
Xtest_pol = normalize(Xtest_pol)
Xtest_pol = np.hstack([np.ones([np.shape(Xtest_pol)[0], 1]), Xtest_pol])

# Entrenamos al modelo con la lambda que de menor error
theta = np.zeros(X_pol.shape[1])
reg = min_lambda
theta_opt = minimize(fun=cost, x0=theta, args=(X_pol, y, reg)).x


print('Cost Train Data: ', cost(theta_opt, X_pol, y, reg))
print('Cost Test Data: ', cost(theta_opt, Xtest_pol, ytest, reg))
print("Error: ",np.abs(cost(theta_opt, X_pol, y, reg) -  cost(theta_opt, Xtest_pol, ytest, reg)))
```

Como resultado de la ejecucion obtenemos los errores en los conjuntos de entrenamiento y validacion para cada lamda.

```
lamda:  0  Error:  0.028895942726835336 Validation error:  28.247895435912586
lamda:  0.001  Error:  0.16941799256823964 Validation error:  9.76658367823541
lamda:  0.003  Error:  0.2431715249673177 Validation error:  14.426734817351512
lamda:  0.01  Error:  0.3712825202497497 Validation error:  15.15100146264909
lamda:  0.03  Error:  0.6367028570832858 Validation error:  12.402053050122248
lamda:  0.1  Error:  1.3605722519127155 Validation error:  8.676472696099585
lamda:  0.3  Error:  2.917649344852081 Validation error:  6.549143296930467
lamda:  1  Error:  6.830463317329151 Validation error:  7.340408304366589
lamda:  3  Error:  14.937645805950392 Validation error:  10.94885042011363
lamda:  10  Error:  31.878154019275883 Validation error:  22.181153476122528
```

En ellos podemos observar como el error mas pequeño se obtiene con lamda = 0.3 y los utilizamos para crear una grafica donde se vea al evolucion de los errores.



Por ultimo utilizamos ese valor de lamda sobre el conjunto de test y obtenemos los siguientes valores

```
Cost Train Data:  6.830463317365309
Cost Test Data:  10.63545119424568
Error:  3.8049878768803715
```