```python
# PARTES 1 Y 2
from ML_UtilsModule import Data_Management
from ML_UtilsModule import Normalization
from scipy.optimize import minimize as sciMin
from scipy.io import loadmat
from matplotlib import pyplot as plt
from sklearn.preprocessing import PolynomialFeatures as pf
from sklearn import preprocessing
import numpy as np

def polinomial_features(X, grado):
    poly = pf(grado)
    return (poly, poly.fit_transform(X))

def h(x, _theta):
    """
    H = O^T * X
    """
    return (np.dot(x, np.transpose(_theta))) #scalar product

def error_hipotesis(_theta, X, y):
    m = np.shape(X)[0]
    n = np.shape(X)[1]
    _theta = np.reshape(_theta, (1, n))

    diff = (h(X, _theta) - y)**2

    return np.sum(diff) / (2 * m)

def J(_theta, X, Y, lamb):
    """
    Cost function
    """
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    _theta = np.reshape(_theta, (1, n))

    diff = (h(X, _theta) - Y)**2
    cost = np.sum(diff)/(2*m)
    cost += (lamb * (np.sum(_theta**2)) / (2 * len(Y)))
    return cost

def gradient(_theta, X, y, lamb):
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    theta = np.reshape(_theta, (1, n))
    var1 = np.transpose(X)
    var2  = h(X, theta)-y

    theta = np.c_[[0], theta[:, 1:]]
    var3 = (lamb/m) * theta
    return ((1/m) * np.dot(var1, var2)) + np.transpose(var3)

def pesos_aleat(L_in, L_out):
    pesos = np.random.uniform(-0.12, 0.12, (L_out, 1+L_in))

    return pesos

def generate_polynom_data(X, p):
    newMatrix = np.zeros((np.shape(X)[0], p))
    grades = np.arange(1, p + 1)
    newMatrix = X ** grades
    return newMatrix

def normalize_matrix(X):
    return Normalization.normalize_data_matrix(X)

def minimizar(theta, X, y, lamb):
    return J(theta, X, y, lamb), gradient(theta, X, y, lamb)

def draw_points_plot(X, Y, _theta):
```

```python
    """
    Draw linear function with X points
    """
    plt.figure()
    plt.scatter(X[:, 1], Y, 20,marker='$F$',color= "red")
    plt.plot(X[:, 1:], h(X, _theta), color="grey")
    plt.show()

def draw_decision_boundary(theta, X, Y, orX, mu, sigma):
    plt.figure()
    x0_min, x0_max = np.min(orX), np.max(orX)
    arrayX = np.arange(x0_min, x0_max, 0.05)
    arrayX = np.reshape(arrayX, (np.shape(arrayX)[0], 1))
    arrayXaux = Normalization.normalize2(generate_polynom_data(arrayX, 8), mu, sigma)
    arrayXaux = Data_Management.add_column_left_of_matrix(arrayXaux)
    theta = np.reshape(theta, (np.shape(theta)[0], 1))

    arrayY = h(arrayXaux, theta.T)
    plt.plot(arrayX, arrayY)
    plt.scatter(orX, Y, 20,marker='$F$',color= "red")
    plt.show()

def draw_plot(X, Y):
    plt.plot(X, Y)


data = loadmat('ex5data1.mat')
X, y, Xval, yval, Xtest, ytest = data['X'], data['y'],  data['Xval'], data['yval'], data['Xtest'], data['ytest
']
XPoly = generate_polynom_data(X, 8)
XPoly, mu, sigma = normalize_matrix(XPoly)
mu = np.reshape(mu, (1, np.shape(mu)[0]))
sigma = np.reshape(sigma, (1, np.shape(sigma)[0]))
XPoly = Data_Management.add_column_left_of_matrix(XPoly)
XPolyVal = Normalization.normalize2(generate_polynom_data(Xval, 8), mu, sigma)
XPolyVal = Data_Management.add_column_left_of_matrix(XPolyVal)
XPolyTest = Normalization.normalize2(generate_polynom_data(Xtest, 8), mu, sigma)
XPolyTest = Data_Management.add_column_left_of_matrix(XPolyTest)

#--------------------------Parte 1-2 ------------------------------------
X_transformed = Data_Management.add_column_left_of_matrix(X)
Xval_transformed = Data_Management.add_column_left_of_matrix(Xval)

error_array = np.array([], dtype=float)
thetas = np.array([], dtype=float)
error_array_val = np.array([], dtype=float)
for i in range(1, np.shape(X_transformed)[0]):
    theta = np.ones(X_transformed.shape[1], dtype=float)

    theta_min = sciMin(fun=minimizar, x0=theta,
    args=(X_transformed[0:  i], y[0: i], 0),
    method='TNC', jac=True,
    options={'maxiter': 70}).x

    error_array = np.append(error_array, J(theta_min, X_transformed[0:  i], y[0: i], 0))
    error_array_val = np.append(error_array_val, J(theta_min, Xval_transformed, yval, 0))
    thetas = np.append(thetas, theta_min)

plt.figure()
draw_plot(np.linspace(0, 10, len(error_array)), error_array)
draw_plot(np.linspace(0, 10, len(error_array_val)), error_array_val)
plt.show()

theta = np.ones(X_transformed.shape[1], dtype=float)
theta_min = sciMin(fun=minimizar, x0=theta,
 args=(X_transformed, y),
 method='TNC', jac=True,
 options={'maxiter': 70}).x

draw_points_plot(X_transformed, y, theta_min)
```

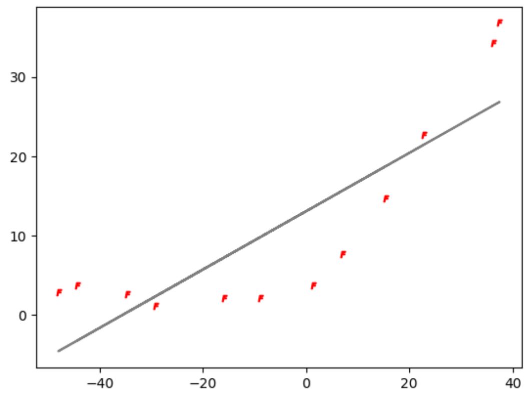

Ilustración 1: primera predicción. Regresión lineal

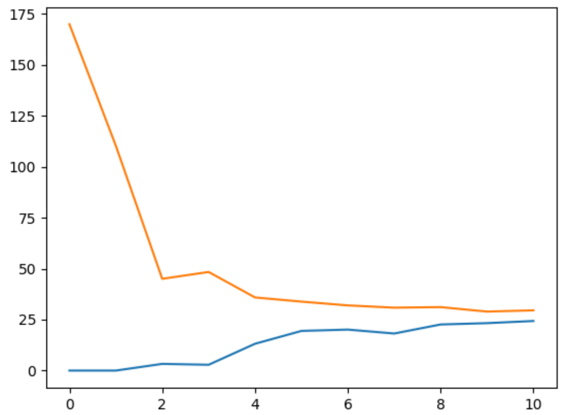

Ilustración 2: primera curva de aprendizaje

```
# PARTE 3
from ML_UtilsModule import Data_Management
from ML_UtilsModule import Normalization
from scipy.optimize import minimize as sciMin
from scipy.io import loadmat
from matplotlib import pyplot as plt
from sklearn.preprocessing import PolynomialFeatures as pf
from sklearn import preprocessing
import numpy as np

def polinomial_features(X, grado):
    poly = pf(grado)
    return (poly, poly.fit_transform(X))

def h(x, _theta):
    """
    H = O^T * X
    """
    return (np.dot(x, np.transpose(_theta))) #scalar product

def error_hipotesis(_theta, X, y):
    m = np.shape(X)[0]
    n = np.shape(X)[1]
```

```python
    _theta = np.reshape(_theta, (1, n))

    diff = (h(X, _theta) - y)**2

    return np.sum(diff) / (2 * m)

def J(_theta, X, Y, lamb):
    """
    Cost function
    """
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    _theta = np.reshape(_theta, (1, n))

    diff = (h(X, _theta) - Y)**2
    cost = np.sum(diff)/(2*m)
    cost += (lamb * (np.sum(_theta**2)) / (2 * len(Y)))
    return cost

def gradient(_theta, X, y, lamb):
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    theta = np.reshape(_theta, (1, n))
    var1 = np.transpose(X)
    var2  = h(X, theta)-y

    theta = np.c_[[0], theta[:, 1:]]
    var3 = (lamb/m) * theta
    return ((1/m) * np.dot(var1, var2)) + np.transpose(var3)

def pesos_aleat(L_in, L_out):
    pesos = np.random.uniform(-0.12, 0.12, (L_out, 1+L_in))

    return pesos

def generate_polynom_data(X, p):
    newMatrix = np.zeros((np.shape(X)[0], p))
    grades = np.arange(1, p + 1)
    newMatrix = X ** grades
    return newMatrix

def normalize_matrix(X):
    return Normalization.normalize_data_matrix(X)

def minimizar(theta, X, y, lamb):
    return J(theta, X, y, lamb), gradient(theta, X, y, lamb)

def draw_points_plot(X, Y, _theta):
    """
    Draw linear function with X points
    """
    plt.figure()
    plt.scatter(X[:, 1], Y, 20,marker='$F$',color= "red")
    plt.plot(X[:, 1:], h(X, _theta), color="grey")
    plt.show()

def draw_decision_boundary(theta, X, Y, orX, mu, sigma):
    plt.figure()
    x0_min, x0_max = np.min(orX), np.max(orX)
    arrayX = np.arange(x0_min, x0_max, 0.05)
    arrayX = np.reshape(arrayX, (np.shape(arrayX)[0], 1))
    arrayXaux = Normalization.normalize2(generate_polynom_data(arrayX, 8), mu, sigma)
    arrayXaux = Data_Management.add_column_left_of_matrix(arrayXaux)
    theta = np.reshape(theta, (np.shape(theta)[0], 1))

    arrayY = h(arrayXaux, theta.T)
    plt.plot(arrayX, arrayY)
    plt.scatter(orX, Y, 20,marker='$F$',color= "red")
    plt.show()

def draw_plot(X, Y):
```

```
    plt.plot(X, Y)


data = loadmat('ex5data1.mat')
X, y, Xval, yval, Xtest, ytest = data['X'], data['y'],  data['Xval'], data['yval'], data['Xtest'], data['ytest
']
XPoly = generate_polynom_data(X, 8)
XPoly, mu, sigma = normalize_matrix(XPoly)
mu = np.reshape(mu, (1, np.shape(mu)[0]))
sigma = np.reshape(sigma, (1, np.shape(sigma)[0]))
XPoly = Data_Management.add_column_left_of_matrix(XPoly)
XPolyVal = Normalization.normalize2(generate_polynom_data(Xval, 8), mu, sigma)
XPolyVal = Data_Management.add_column_left_of_matrix(XPolyVal)
XPolyTest = Normalization.normalize2(generate_polynom_data(Xtest, 8), mu, sigma)
XPolyTest = Data_Management.add_column_left_of_matrix(XPolyTest)

theta = np.ones(XPoly.shape[1], dtype=float)

theta_min = sciMin(fun=minimizar, x0=theta,
 args=(XPoly, y, 0),
 method='TNC', jac=True,
 options={'maxiter': 70}).x

draw_decision_boundary(theta_min, XPoly, y, X, mu, sigma)

error_array = np.array([], dtype=float)
error_array_val = np.array([], dtype=float)
for i in range(1, np.shape(XPoly)[0]):
    theta = np.ones(XPoly.shape[1], dtype=float)

    theta_min = sciMin(fun=minimizar, x0=theta,
    args=(XPoly[0:  i], y[0: i], 0),
    method='TNC', jac=True,
    options={'maxiter': 70}).x

    error_array = np.append(error_array, error_hipotesis(theta_min, XPoly[0:  i], y[0: i]))
    error_array_val = np.append(error_array_val, error_hipotesis(theta_min, XPolyVal, yval))

plt.figure()
draw_plot(np.linspace(0, np.shape(XPoly)[0], len(error_array)), error_array)
draw_plot(np.linspace(0, np.shape(XPoly)[0], len(error_array_val)), error_array_val)
plt.show()
```

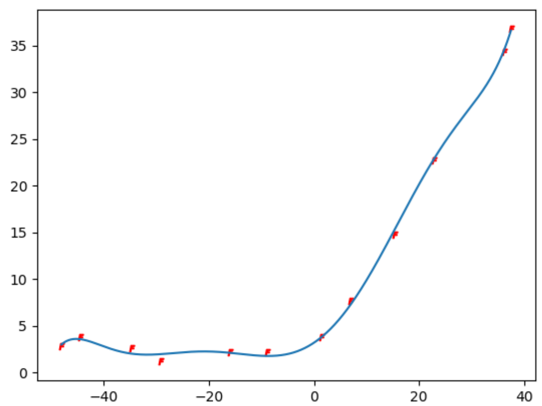

Ilustración 3: Segunda predicción con polinomios

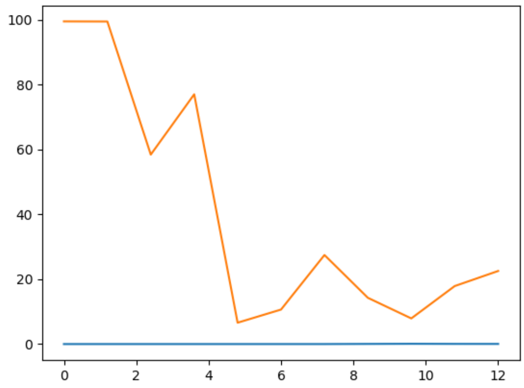

Ilustración 4: curvas de aprendizaje con polinomio de grado 8

```
# PARTE 4

from ML_UtilsModule import Data_Management

from ML_UtilsModule import Normalization
from scipy.optimize import minimize as sciMin
from scipy.io import loadmat
from matplotlib import pyplot as plt
from sklearn.preprocessing import PolynomialFeatures as pf
from sklearn import preprocessing
import numpy as np

def polinomial_features(X, grado):
    poly = pf(grado)
    return (poly, poly.fit_transform(X))

def h(x, _theta):
    """
    H = O^T * X
    """
    return (np.dot(x, np.transpose(_theta))) #scalar product

def error_hipotesis(_theta, X, y):
    m = np.shape(X)[0]
    n = np.shape(X)[1]
    _theta = np.reshape(_theta, (1, n))

    diff = (h(X, _theta) - y)**2

    return np.sum(diff) / (2 * m)

def J(_theta, X, Y, lamb):
    """
    Cost function
    """
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    _theta = np.reshape(_theta, (1, n))

    diff = (h(X, _theta) - Y)**2
    cost = np.sum(diff)/(2*m)
    cost += (lamb * (np.sum(_theta**2)) / (2 * len(Y)))
    return cost


def gradient(_theta, X, y, lamb):
    m = np.shape(X)[0]
    n = np.shape(X)[1]

    theta = np.reshape(_theta, (1, n))
```

```python
    var1 = np.transpose(X)
    var2  = h(X, theta)-y

    theta = np.c_[[0], theta[:, 1:]]
    var3 = (lamb/m) * theta
    return ((1/m) * np.dot(var1, var2)) + np.transpose(var3)

def pesos_aleat(L_in, L_out):
    pesos = np.random.uniform(-0.12, 0.12, (L_out, 1+L_in))

    return pesos

def generate_polynom_data(X, p):
    newMatrix = np.zeros((np.shape(X)[0], p))
    grades = np.arange(1, p + 1)
    newMatrix = X ** grades
    return newMatrix

def normalize_matrix(X):
    return Normalization.normalize_data_matrix(X)

def minimizar(theta, X, y, lamb):
    return J(theta, X, y, lamb), gradient(theta, X, y, lamb)

def draw_points_plot(X, Y, _theta):
    """
    Draw linear function with X points
    """
    plt.figure()
    plt.scatter(X[:, 1], Y, 20,marker='$F$',color= "red")
    plt.plot(X[:, 1:], h(X, _theta), color="grey")
    plt.show()

def draw_decision_boundary(theta, X, Y, orX, mu, sigma):
    plt.figure()
    x0_min, x0_max = np.min(orX), np.max(orX)
    arrayX = np.arange(x0_min, x0_max, 0.05)
    arrayX = np.reshape(arrayX, (np.shape(arrayX)[0], 1))
    arrayXaux = Normalization.normalize2(generate_polynom_data(arrayX, 8), mu, sigma)
    arrayXaux = Data_Management.add_column_left_of_matrix(arrayXaux)
    theta = np.reshape(theta, (np.shape(theta)[0], 1))

    arrayY = h(arrayXaux, theta.T)
    plt.plot(arrayX, arrayY)
    plt.scatter(orX, Y, 20,marker='$F$',color= "red")
    plt.show()

def draw_plot(X, Y):
    plt.plot(X, Y)


data = loadmat('ex5data1.mat')
X, y, Xval, yval, Xtest, ytest = data['X'], data['y'],  data['Xval'], data['yval'], data['Xtest'], data['ytest
']
XPoly = generate_polynom_data(X, 8)
XPoly, mu, sigma = normalize_matrix(XPoly)
mu = np.reshape(mu, (1, np.shape(mu)[0]))
sigma = np.reshape(sigma, (1, np.shape(sigma)[0]))
XPoly = Data_Management.add_column_left_of_matrix(XPoly)
XPolyVal = Normalization.normalize2(generate_polynom_data(Xval, 8), mu, sigma)
XPolyVal = Data_Management.add_column_left_of_matrix(XPolyVal)
XPolyTest = Normalization.normalize2(generate_polynom_data(Xtest, 8), mu, sigma)
XPolyTest = Data_Management.add_column_left_of_matrix(XPolyTest)

lambdaAux = [ 0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10  ]

error_array = np.array([], dtype=float)
error_array_val = np.array([], dtype=float)
thetas = np.array([], dtype=float)
for l in range(len(lambdaAux)):
    theta = np.ones(XPoly.shape[1], dtype=float)

    theta_min = sciMin(fun=minimizar, x0=theta,
```

```python
    args=(XPoly, y, lambdaAux[l]),
    method='TNC', jac=True,
    options={'maxiter': 70}).x

    error_array = np.append(error_array, J(theta_min, XPoly, y, lambdaAux[l]))
    error_array_val = np.append(error_array_val, J(theta_min, XPolyVal, yval, lambdaAux[l]))
    thetas = np.append(thetas, theta_min)

lambdaIndex = np.argmin(error_array_val)
plt.figure()
draw_plot(lambdaAux, error_array)
draw_plot(lambdaAux, error_array_val)
plt.show()

theta = np.ones(XPoly.shape[1], dtype=float)
theta_min = sciMin(fun=minimizar, x0=theta,
    args=(XPoly, y, lambdaAux[lambdaIndex]),
    method='TNC', jac=True,
    options={'maxiter': 70}).x

print("Best lambda: " + str(lambdaAux[lambdaIndex]))
print(J(theta_min, XPolyTest, ytest, lambdaAux[lambdaIndex]))
```

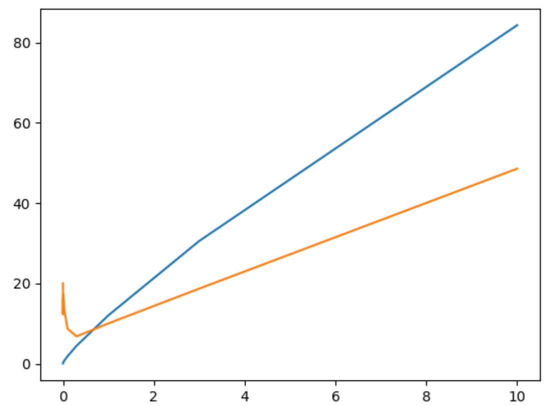

Ilustración 5: representación de la elección de la mejor lambda

```
F:\Universidad\4Videojuegos\ML\ML\Practica5>python practica5.py
Best lambda: 0.3
7.386466960580218
```

Ilustración 6: mejor lambda y coste mínimo