Aprendizaje automático y minería de datos: Práctica 5

Jorge Rodríguez García y Gonzalo Sanz Lastra

Código de la práctica:

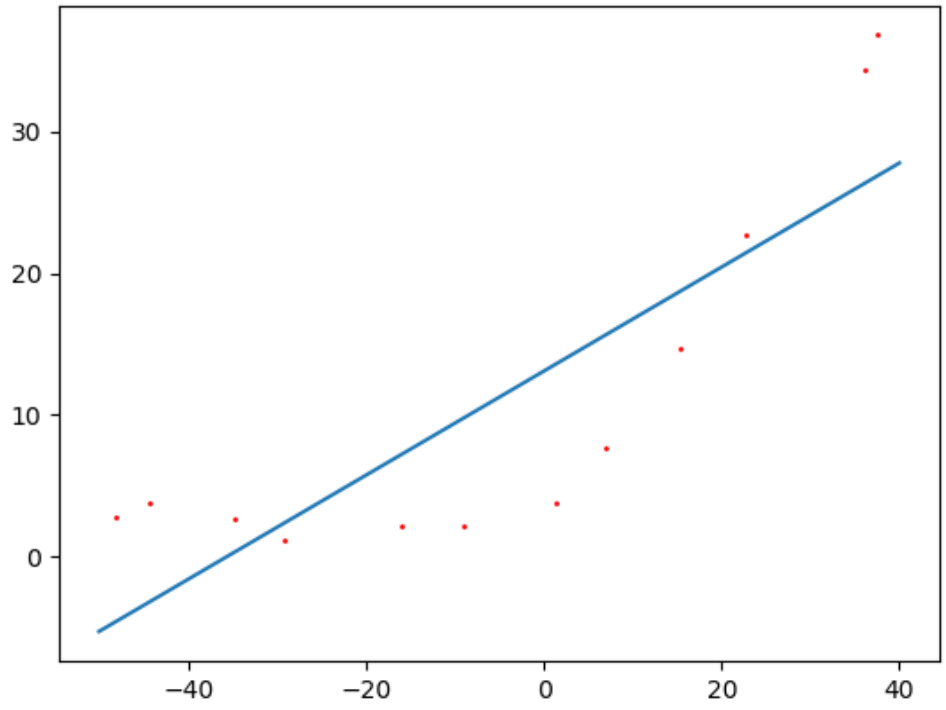
```python
import numpy as np
import scipy.optimize as opt            # para la funcion de gradiente
from matplotlib import pyplot as plt    # para dibujar las graficas
from scipy.io import loadmat
from sklearn import preprocessing       # para polinomizar las Xs

def load_mat(file_name):
    """carga el fichero mat especificado y lo devuelve en una matriz data"""
    return loadmat(file_name)

def hLine(X, O):
    """devuelve la funcion h(x) usando la ecuacion de la recta"""
    return O[0] + O[1]*X

def functionGraphic(X, Y, O, mu, sigma):
    """pinta la funcion h(x)"""
    minValue = np.amin(X)
    maxValue = np.amax(X)
    x = np.linspace(minValue, maxValue, 256, endpoint=True)[np.newaxis].T
    xPoly = polynomize(x, 8)
    xNorm = normalizeValues(xPoly[:, 1:], mu, sigma)
    xNorm = np.hstack([np.ones([xNorm.shape[0], 1]), xNorm])

    y = np.dot(xNorm, O)

    # pintamos muestras de entrenamiento
    plt.scatter(X[:, 0], Y, 1, 'red')
    # pintamos funcion de estimacion
    plt.plot(x, y)
    plt.savefig('PolynomialH(X).png')
    plt.show()

def learningCurve(errorX, errorXVal):
    """muestra el grafico de la funcion h(x)"""

    x = np.linspace(0, 12, errorX.shape[0], endpoint=True)
    xVal = np.linspace(0, 12, errorXVal.shape[0], endpoint=True)

    # pintamos funcion de estimacion
    plt.plot(x, errorX)
    plt.plot(xVal, errorXVal)
    plt.savefig('LearningCurvePolyLambda100.png')
    plt.show()

def lambdaGraphic(errorX, errorXVal, lambdas):
    plt.figure()
    plt.plot(lambdas, errorX)
    plt.plot(lambdas, errorXVal)
    plt.show()
```

```python
def costeLineal(X, Y, O, reg):
    """devuelve la funcion de coste, dadas X, Y, y thetas"""
    O = O[np.newaxis]
    AuxO = O[:, 1:]

    H = np.dot(X, O.T)
    Aux = (H-Y)**2
    cost = Aux.sum()/(2*len(X))

    return cost + (AuxO**2).sum()*reg/(2*X.shape[0])

def gradienteLineal(X, Y, O, reg):
    """la operacion que hace el gradiente por dentro -> devuelve un vector de valores"""
    AuxO = np.hstack([np.zeros([1]), O[1:,]])
    O = O[np.newaxis]
    AuxO = AuxO[np.newaxis].T

    return ((X.T.dot(np.dot(X, O.T)-Y))/X.shape[0] + (reg/X.shape[0])*AuxO)

def minimizeFunc(O, X, Y, reg):
    return (costeLineal(X, Y, O, reg), gradienteLineal(X, Y, O, reg))

def polynomize(X, p):
    poly = preprocessing.PolynomialFeatures(p)
    return poly.fit_transform(X) # añade automaticamente la columna de 1s

def normalize(X):
    """normalizacion de escalas, para cuando haya mas de un atributo"""
    mu = X.mean(0)[np.newaxis]   # media de cada columna de X
    sigma = X.std(0)[np.newaxis] # desviacion estandar de cada columna de X

    X_norm = (X - mu)/sigma

    return X_norm, mu, sigma

def normalizeValues(valoresPrueba, mu, sigma):
    """normaliza los valores de prueba con la mu y sigma de los atributos X (al normalizarlos)"""
    return (valoresPrueba - mu)/sigma

def main():
    valores = load_mat("ex5data1.mat")

    X = valores['X']         # datos de entrenamiento
    Y = valores['y']
    Xval = valores['Xval']   # ejemplos de validacion
    Yval = valores['yval']
    Xtest = valores['Xtest'] # prueba
    Ytest = valores['ytest']

```

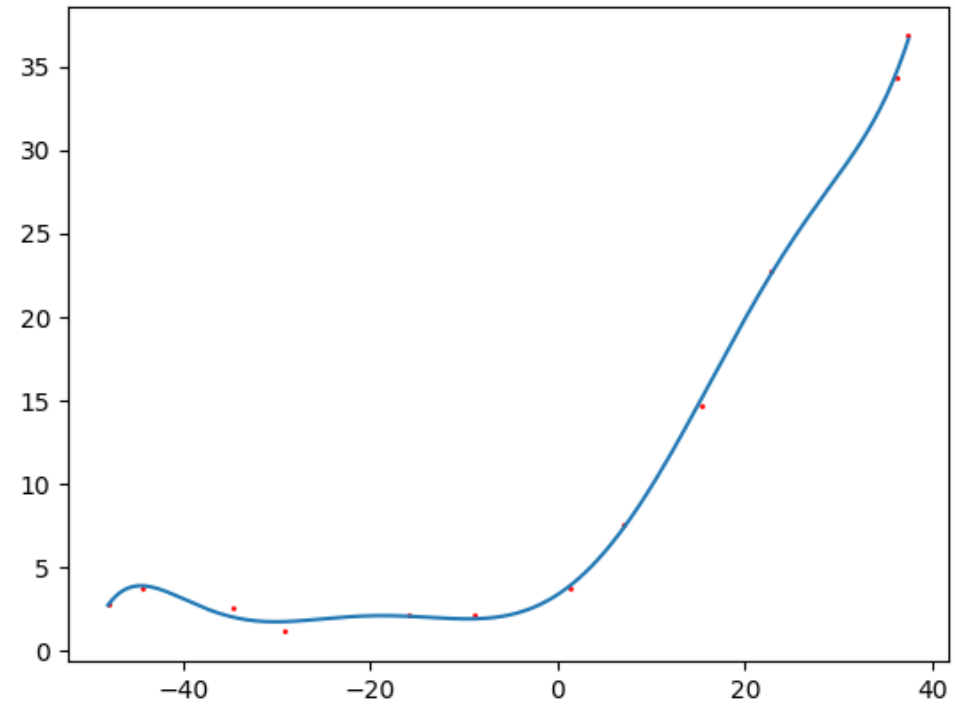
```python
    Xpoly = polynomize(X, 8)                    # pone automaticamente columna de 1s
    Xnorm, mu, sigma = normalize(Xpoly[:, 1:]) # se pasa sin la columna de 1s (evitar division entre 0)
    Xnorm = np.hstack([np.ones([Xnorm.shape[0], 1]), Xnorm]) # volvemos a poner columna de 1s

    XpolyVal = polynomize(Xval, 8)
    XnormVal = normalizeValues(XpolyVal[:, 1:], mu, sigma)
    XnormVal = np.hstack([np.ones([XnormVal.shape[0], 1]), XnormVal])

    XpolyTest = polynomize(Xtest, 8)
    XnormTest = normalizeValues(XpolyTest[:, 1:], mu, sigma)
    XnormTest = np.hstack([np.ones([XnormTest.shape[0], 1]), XnormTest])

    m = Xnorm.shape[0]      # numero de muestras de entrenamiento
    n = Xnorm.shape[1]      # numero de variables x que influyen en el resultado y, mas la columna de 1s

    l = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10]) # posibles valores de lambda

    thetaVec = np.zeros([n])

    errorX = np.zeros(l.shape[0])
    errorXVal = np.zeros(l.shape[0])

    # errores para cada valor de lambda
    for i in range(l.shape[0]):
        result = opt.minimize(fun = minimizeFunc, x0 = thetaVec,
        args = (Xnorm, Y, l[i]), method = 'TNC', jac = True, options = {'maxiter':70})
        O = result.x

        errorX[i] = costeLineal(Xnorm, Y, O, l[i])
        errorXVal[i] = costeLineal(XnormVal, Yval, O, l[i])

    lambdaGraphic(errorX, errorXVal, l)

    # lambda que hace el error minimo en los ejemplos de validacion
    lambdaIndex = np.argmin(errorXVal)
    print("Best lambda: " + str(l[lambdaIndex]))

    # thetas usando la lambda que hace el error minimo (sobre ejemplos de entrenamiento)
    result = opt.minimize(fun = minimizeFunc, x0 = thetaVec,
        args = (Xnorm, Y, l[lambdaIndex]), method = 'TNC', jac = True, options = {'maxiter':70})

    O = result.x

    print(costeLineal(XnormTest, Ytest, O, l[lambdaIndex])) # error para los datos de testeo (nunca antes vistos)
    #functionGraphic(X, Y, O, mu, sigma)

    # curvas de aprendizaje cogiendo subconjuntos
    """errorX = np.zeros(m - 1)
    errorXVal = np.zeros(m - 1)
```

```
132
133     # lambda que hace el error minimo en los ejemplos de validacion
134     lambdaIndex = np.argmin(errorXVal)
135     print("Best lambda: " + str(l[lambdaIndex]))
136
137     # thetas usando la lambda que hace el error minimo (sobre ejemplos de entrenamiento)
138     result = opt.minimize(fun = minimizeFunc, x0 = thetaVec,
139         args = (Xnorm, Y, l[lambdaIndex]), method = 'TNC', jac = True, options = {'maxiter':70})
140
141     O = result.x
142
143     print(costeLineal(XnormTest, Ytest, O, l[lambdaIndex])) # error para los datos de testeo (nunca antes vistos)
144     #functionGraphic(X, Y, O, mu, sigma)
145
146     # curvas de aprendizaje cogiendo subconjuntos
147     """errorX = np.zeros(m - 1)
148     errorXVal = np.zeros(m - 1)
149
150     for i in range(1, m):
151         result = opt.minimize(fun = minimizeFunc, x0 = thetaVec,
152          args = (Xnorm[0:i], Y[0:i], 100), method = 'TNC', jac = True, options = {'maxiter':70})
153         O = result.x
154
155         errorX[i-1] = costeLineal(Xnorm[0:i], Y[0:i], O, 100)
156         errorXVal[i-1] = costeLineal(XnormVal, Yval, O, 100)
157
158     learningCurve(errorX, errorXVal)"""
159
160 main()
```

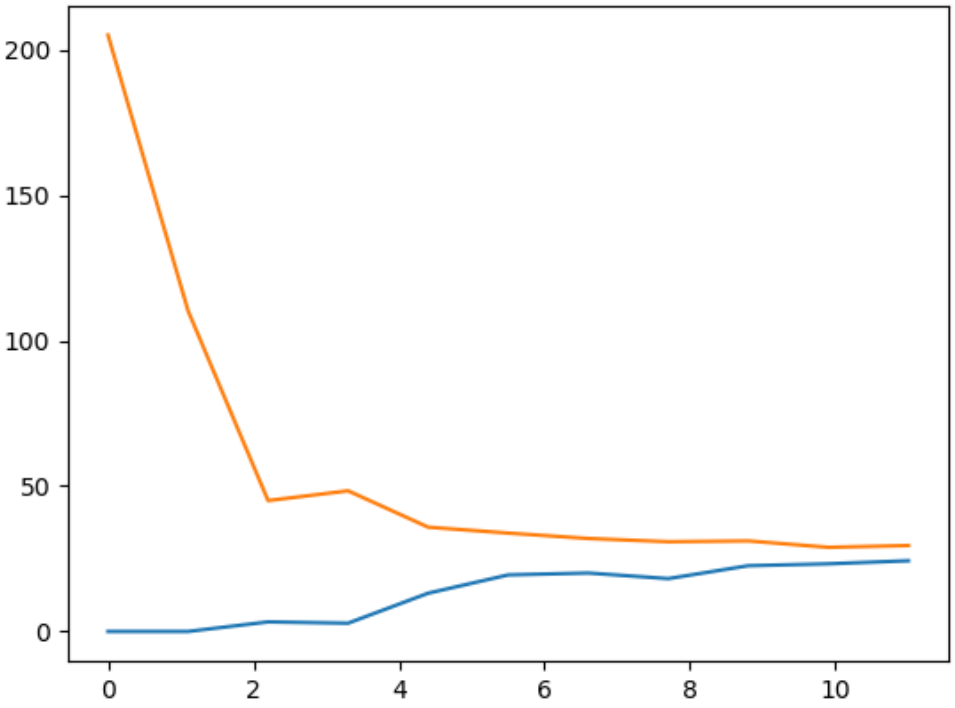
Linear H(X)



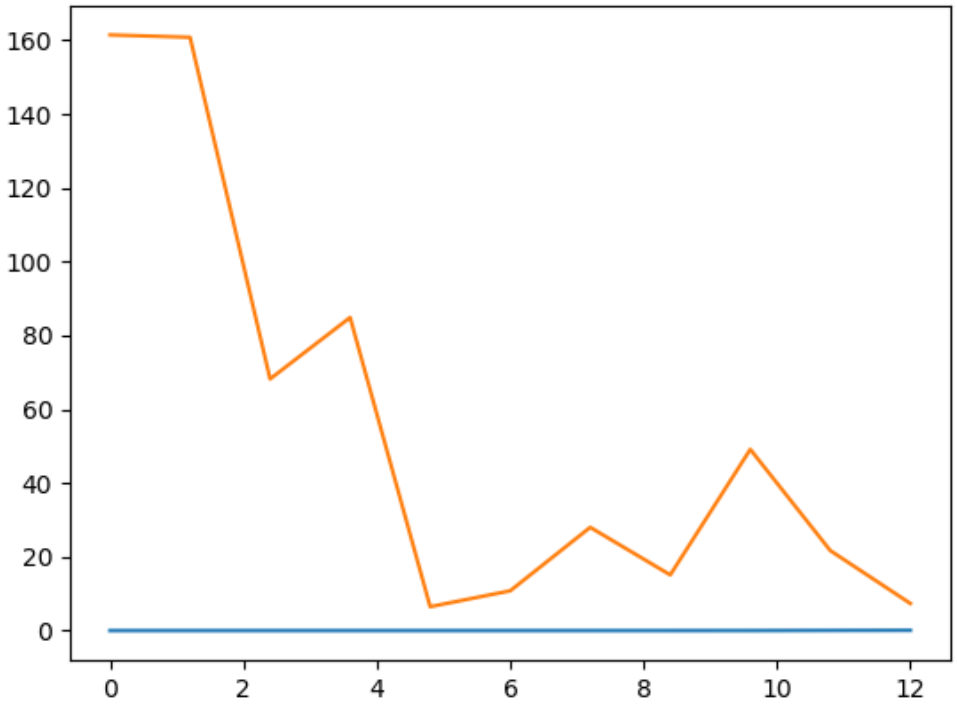
Polynomial H(X)



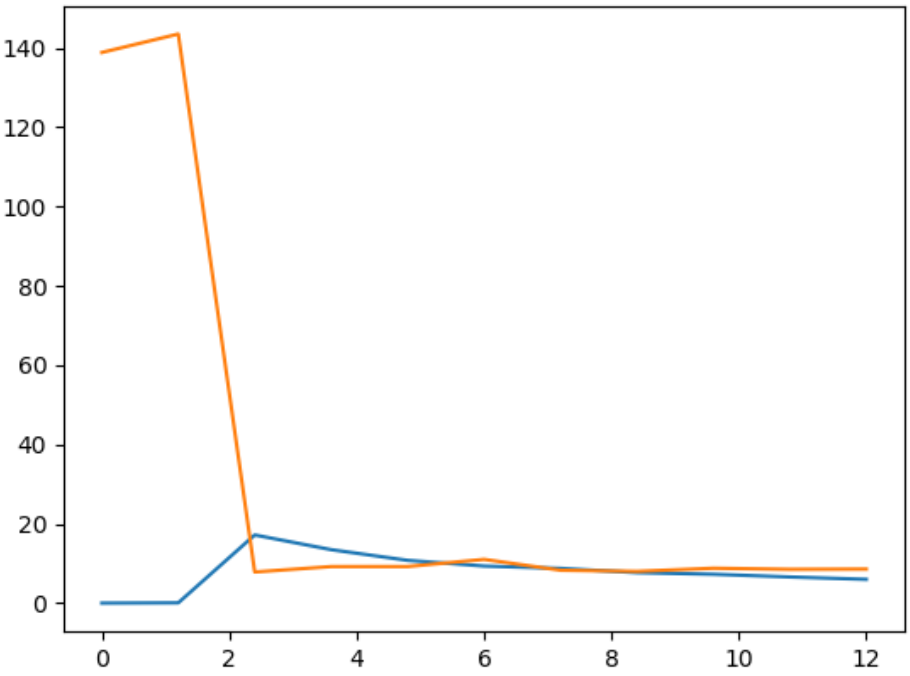
Learning curve



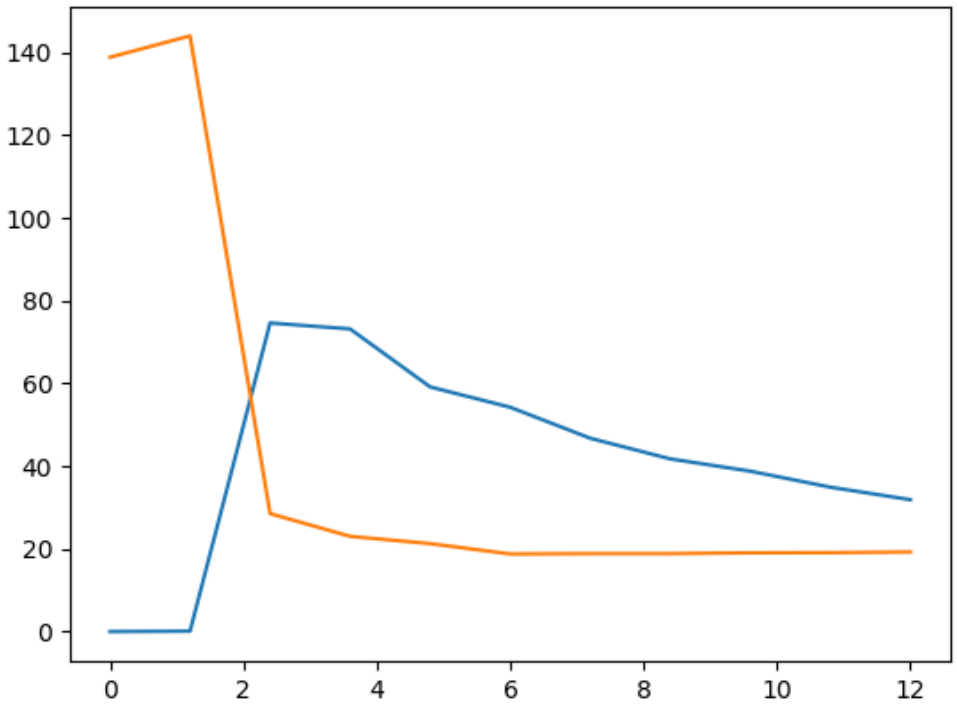
Learning curve lambda = 0



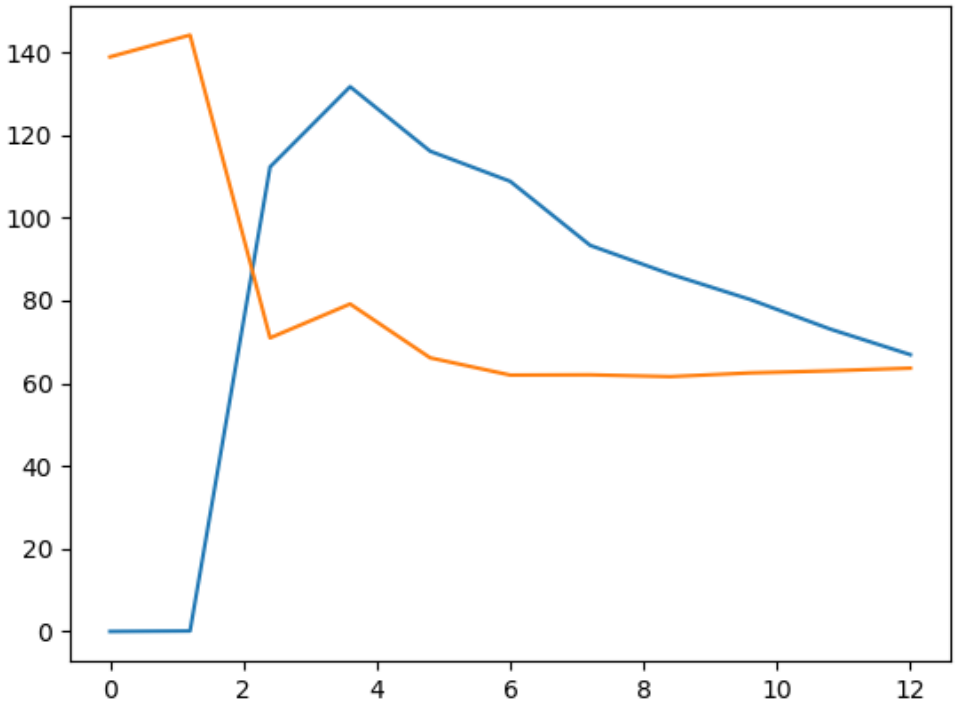
Learning curve lambda = 1



Learning curve lambda = 10



Learning curve lambda = 100



Lambda errors