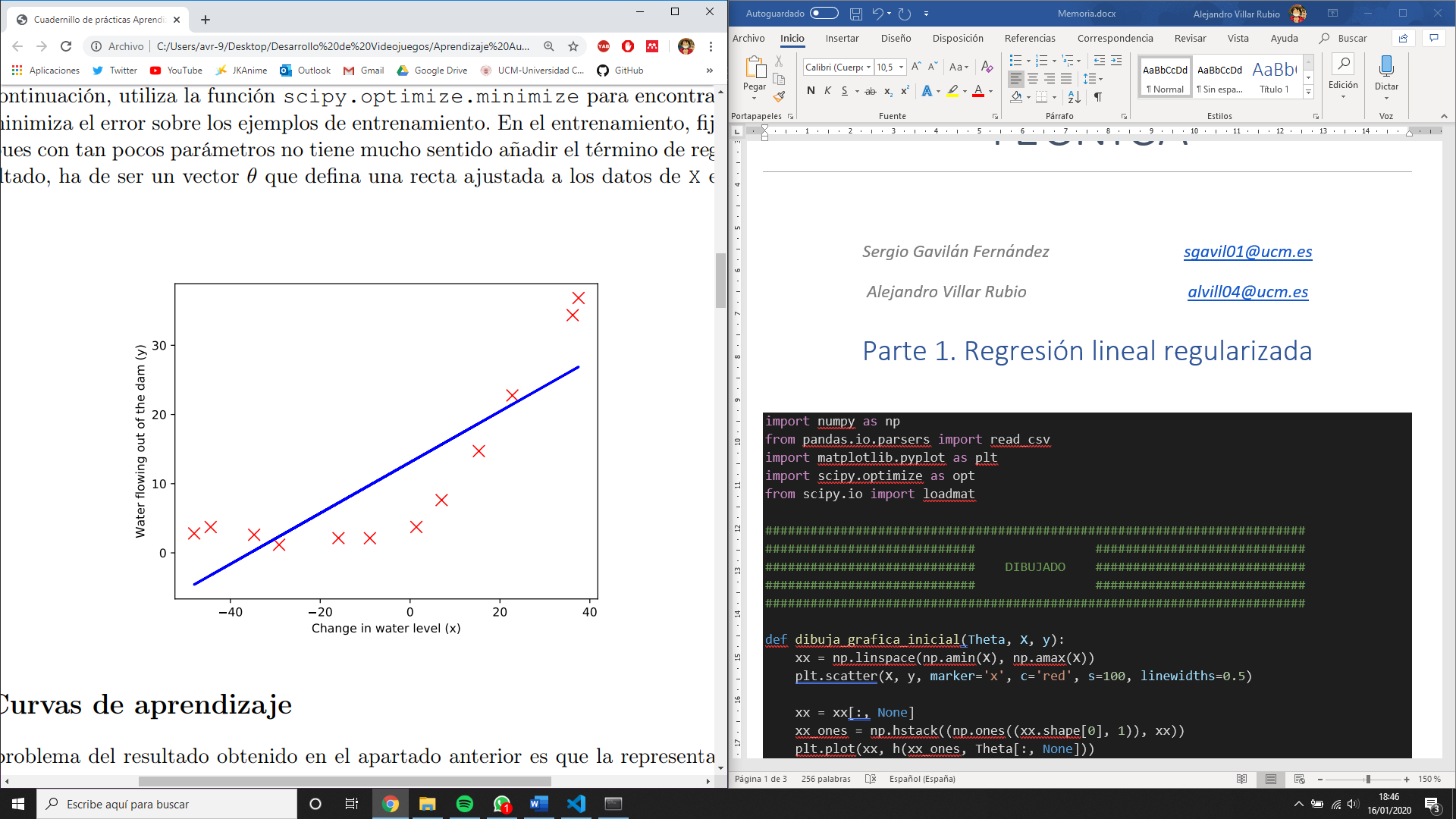
Práctica 5. Memoria Técnica

Sergio Gavilán Fernández [sgavil01@ucm.es](mailto:sgavil01@ucm.es)

Alejandro Villar Rubio [alvill04@ucm.es](mailto:alvill04@ucm.es)

# Parte 1. Regresión lineal regularizada



import numpy as np

from pandas.io.parsers import read\_csv

import matplotlib.pyplot as plt

import scipy.optimize as opt

from scipy.io import loadmat

########################################################################

############################                ############################

############################    DIBUJADO    ############################

############################                ############################

########################################################################

def dibuja\_grafica\_inicial(Theta, X, y):

    xx = np.linspace(np.amin(X), np.amax(X))

    plt.scatter(X, y, marker='x', c='red', s=100, linewidths=0.5)

    xx = xx[:, None]

    xx\_ones = np.hstack((np.ones((xx.shape[0], 1)), xx))

    plt.plot(xx, h(xx\_ones, Theta[:, None]))

    plt.xlabel('Change in water level (x)')

    plt.ylabel('Water flowing out of the dam (y)')

    plt.show()

########################################################################

################                                        ################

################  CALCULOS DE COSTE, GRADIENTE Y THETA  ################

################                                        ################

########################################################################

def h(X, Theta):

    return np.dot(X, Theta)

def f\_coste(Theta, X, y, reg):

    m = len(X)

    Theta = Theta[:, None]

    return (1 / (2 \* m)) \* np.sum(np.square(h(X, Theta) - y)) \

        + (reg / (2 \* m)) \* np.sum(np.square(Theta[1:]))

def f\_gradiente(Theta, X, y, reg):

    m = len(X)

    return (1 / m) \* (np.sum(np.dot((h(X, Theta[:, None]) - y).T, X), axis=0)) \

        + (reg / m) \* Theta

def f\_optimizacion(Theta, X, y, reg):

    return f\_coste(Theta, X, y, reg), f\_gradiente(Theta, X, y, reg)

########################################################################

################################        ################################

################################  MAIN  ################################

################################        ################################

########################################################################

def main():

    data = loadmat("ex5data1.mat")

    y = data["y"]

    X = data["X"]

    yval = data["yval"]

    Xval = data["Xval"]

    ytest = data["ytest"]

    Xtest = data["Xtest"]

    X\_ones = np.hstack((np.ones((X.shape[0], 1)), X))

    n = X\_ones.shape[1]

    Theta = np.array([1, 1])

    reg = 0

    optTheta = opt.minimize(fun=f\_optimizacion, x0=Theta,

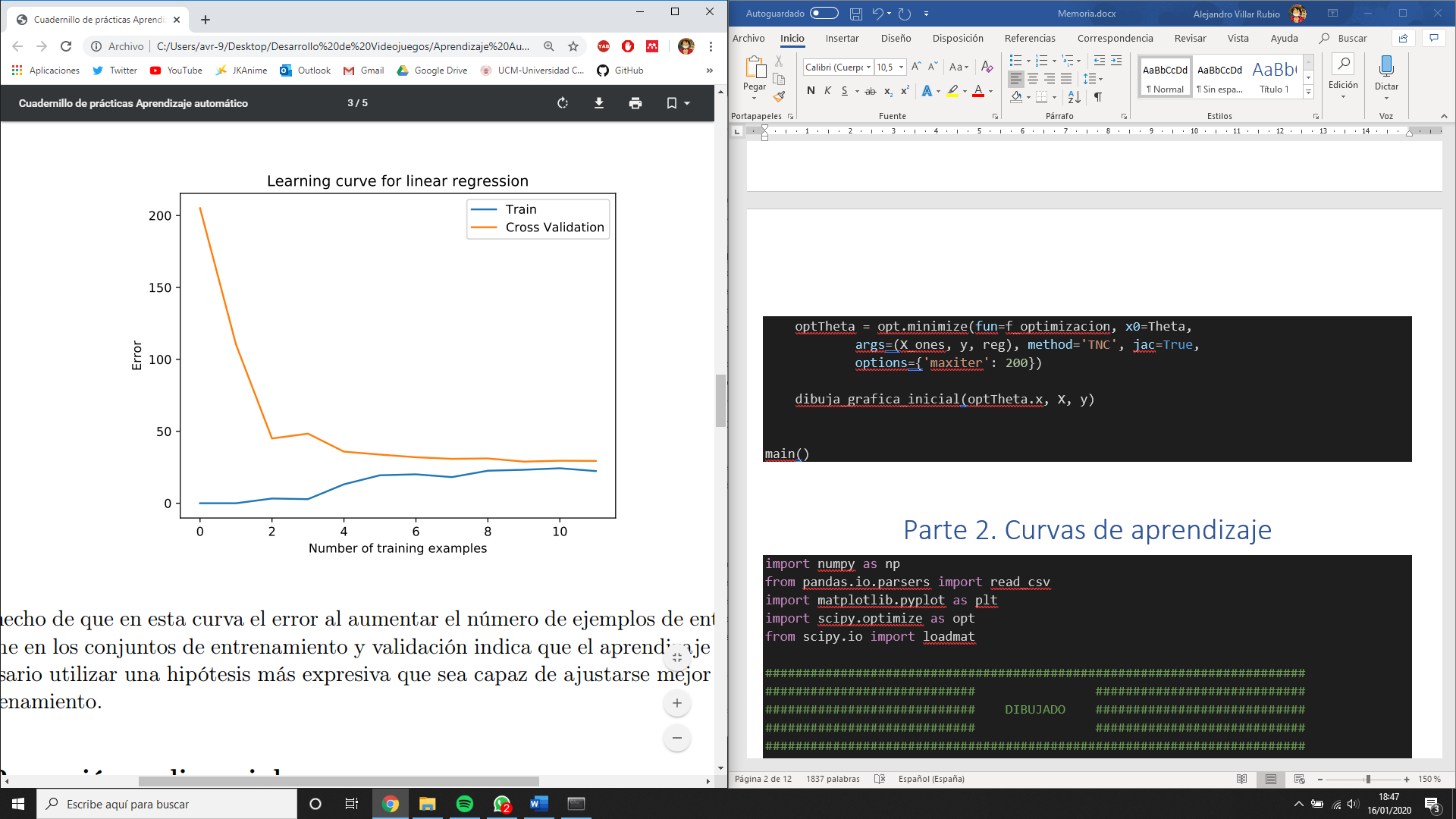
            args=(X\_ones, y, reg), method='TNC', jac=True,

            options={'maxiter': 200})

    dibuja\_grafica\_inicial(optTheta.x, X, y)

main()

# Parte 2. Curvas de aprendizaje



import numpy as np

from pandas.io.parsers import read\_csv

import matplotlib.pyplot as plt

import scipy.optimize as opt

from scipy.io import loadmat

########################################################################

############################                ############################

############################    DIBUJADO    ############################

############################                ############################

########################################################################

def dibuja\_grafica\_inicial(Theta, X, y):

    xx = np.linspace(np.amin(X), np.amax(X))

    plt.scatter(X, y, marker='x', c='red', s=100, linewidths=0.5)

    xx = xx[:, None]

    xx\_ones = np.hstack((np.ones((xx.shape[0], 1)), xx))

    plt.plot(xx, h(xx\_ones, Theta[:, None]))

    plt.xlabel('Change in water level (x)')

    plt.ylabel('Water flowing out of the dam (y)')

    plt.show()

def dibuja\_learning\_curve(error, error\_val):

    xx = np.linspace(0, 12, 12)

    plt.plot(xx, error, label='Train')

    plt.plot(xx, error\_val, label='Cross Validation')

    plt.title("Learning curve for linear regression")

    plt.xlabel('Number of training examples')

    plt.ylabel('Error')

    plt.legend()

    plt.show()

########################################################################

################                                        ################

################  CALCULOS DE COSTE, GRADIENTE Y THETA  ################

################                                        ################

########################################################################

def h(X, Theta):

    return np.dot(X, Theta)

def f\_coste(Theta, X, y, reg):

    m = len(X)

    Theta = Theta[:, None]

    return (1 / (2 \* m)) \* np.sum(np.square(h(X, Theta) - y)) \

        + (reg / (2 \* m)) \* np.sum(np.square(Theta[1:]))

def f\_gradiente(Theta, X, y, reg):

    m = len(X)

    return (1 / m) \* (np.sum(np.dot((h(X, Theta[:, None]) - y).T, X), axis=0)) \

        + (reg / m) \* Theta[1:]

def f\_optimizacion(Theta, X, y, reg):

    return f\_coste(Theta, X, y, reg), f\_gradiente(Theta, X, y, reg)

def get\_optimize\_theta(X, y, reg):

    initial\_theta = np.zeros((X.shape[1], 1))

    optTheta = opt.minimize(fun=f\_optimizacion, x0=initial\_theta,

            args=(X, y, reg), method='TNC', jac=True,

            options={'maxiter': 200})

    return optTheta.x

########################################################################

######################                            ######################

######################  APARTADOS DE LA PRACTICA  ######################

######################                            ######################

########################################################################

def learning\_curve(X, y, Xval, yval, reg):

    m = len(X)

    error\_train = np.zeros((m, 1))

    error\_val = np.zeros((m, 1))

    for i in range(1, m + 1):

        Theta = get\_optimize\_theta(X[: i], y[: i], reg)

        error\_train[i - 1] = f\_optimizacion(Theta, X[: i], y[: i], 0)[0]

        error\_val[i - 1] = f\_optimizacion(Theta, Xval, yval, 0)[0]

    dibuja\_learning\_curve(error\_train, error\_val)

########################################################################

################################        ################################

################################  MAIN  ################################

################################        ################################

########################################################################

def main():

    data = loadmat("ex5data1.mat")

    y = data["y"]

    X = data["X"]

    X\_ones = np.hstack((np.ones((X.shape[0], 1)), X))

    yval = data["yval"]

    Xval = data["Xval"]

    Xval\_ones = np.hstack((np.ones((Xval.shape[0], 1)), Xval))

    ytest = data["ytest"]

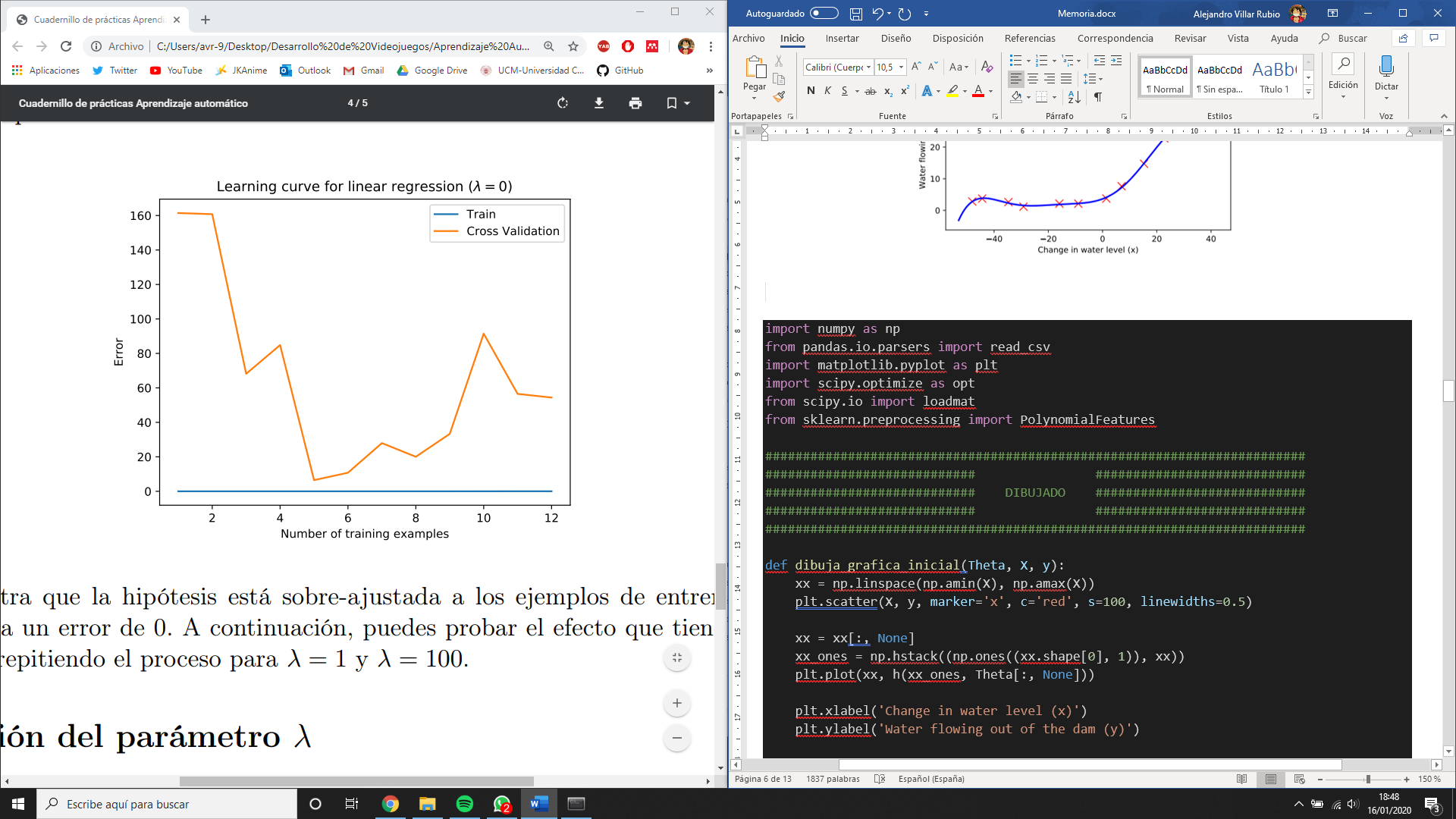
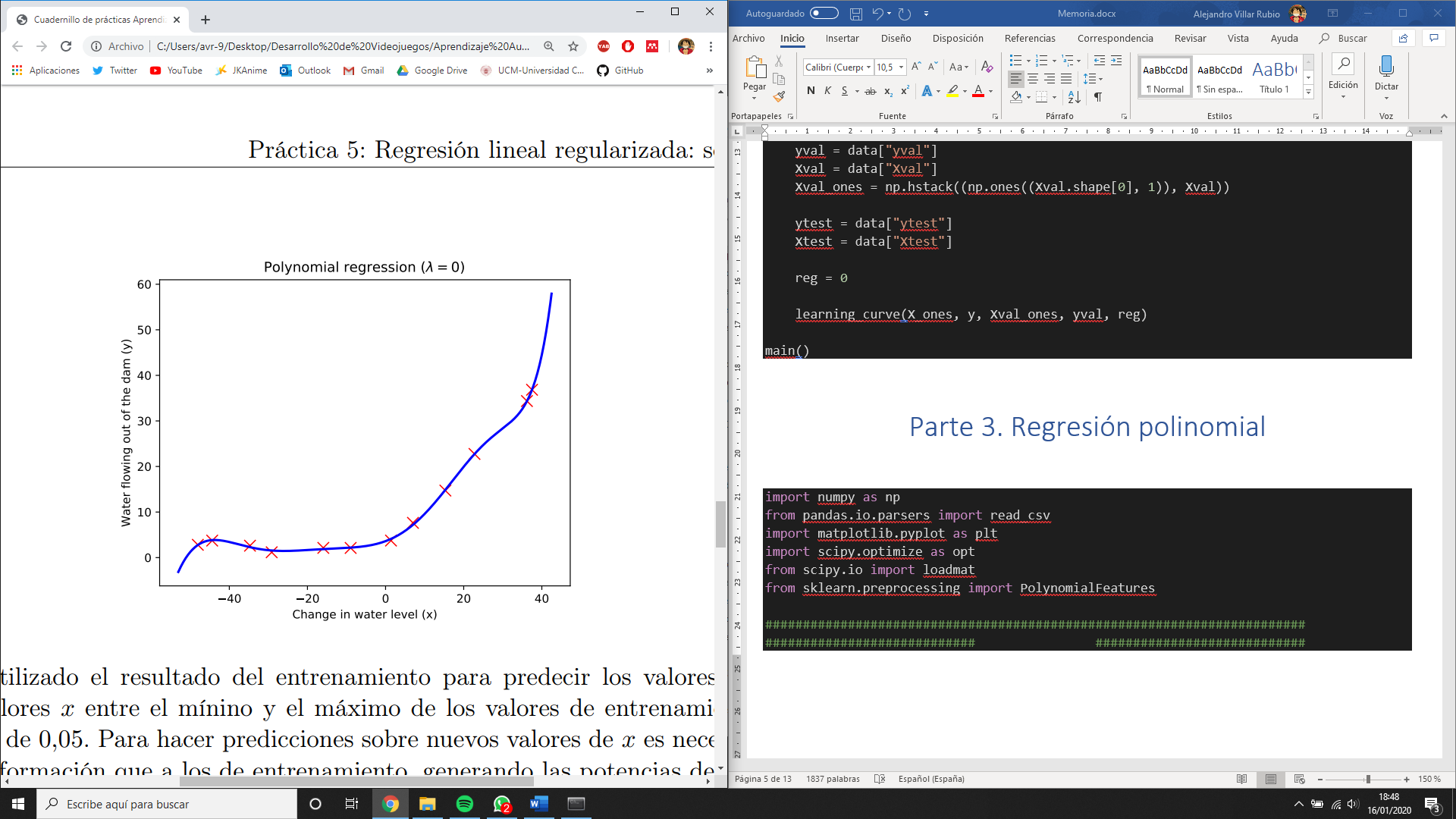
    Xtest = data["Xtest"]

    reg = 0

    learning\_curve(X\_ones, y, Xval\_ones, yval, reg)

main()

# Parte 3. Regresión polinomial



import numpy as np

from pandas.io.parsers import read\_csv

import matplotlib.pyplot as plt

import scipy.optimize as opt

from scipy.io import loadmat

from sklearn.preprocessing import PolynomialFeatures

########################################################################

############################                ############################

############################    DIBUJADO    ############################

############################                ############################

########################################################################

def dibuja\_grafica\_inicial(Theta, X, y):

    xx = np.linspace(np.amin(X), np.amax(X))

    plt.scatter(X, y, marker='x', c='red', s=100, linewidths=0.5)

    xx = xx[:, None]

    xx\_ones = np.hstack((np.ones((xx.shape[0], 1)), xx))

    plt.plot(xx, h(xx\_ones, Theta[:, None]))

    plt.xlabel('Change in water level (x)')

    plt.ylabel('Water flowing out of the dam (y)')

def dibuja\_learning\_curve(error\_train, error\_val, reg, axs):

    m = len(error\_train)

    axs[1].plot(range(1, m + 1), error\_train, label='Train')

    axs[1].plot(range(1, m + 1), error\_val, label='Cross Validation')

    axs[1].title.set\_text("Learning curve for linear regression " + r'$(\lambda = {})$'.format(reg))

    axs[1].set\_xlabel('Number of training examples')

    axs[1].set\_ylabel('Error')

    axs[1].legend()

def dibuja\_polynomial\_regression(Theta, X, y, mu, sigma, reg, p, axs):

    axs[0].scatter(X, y, marker='x', c='red', linewidths=0.5, s = 100)

    axs[0].title.set\_text("Polinomial regression " r'$(\lambda = {})$'.format(reg))

    axs[0].set\_xlabel('Change in water level (x)')

    axs[0].set\_ylabel('Water flowing out of the dam (y)')

    x = np.array(np.arange(min(X) - 5, max(X) + 5, 0.05))

    X\_pol = polinomial\_matrix(x, p)

    X\_pol = (X\_pol - mu) / sigma

    X\_pol = np.insert(X\_pol, 0, 1, axis=1)

    axs[0].plot(x, np.dot(X\_pol, Theta))

########################################################################

################                                        ################

################  CALCULOS DE COSTE, GRADIENTE Y THETA  ################

################                                        ################

########################################################################

def h(X, Theta):

    return np.dot(X, Theta)

def f\_coste(Theta, X, y, reg):

    m = len(X)

    return (1 / (2 \* m)) \* np.sum(np.square(h(X, Theta[:, None]) - y)) \

        + (reg / (2 \* m)) \* np.sum(np.square(Theta[1:]))

def f\_gradiente(Theta, X, y, reg):

    m = len(X)

    return (1 / m) \* (np.sum(np.dot((h(X, Theta[:, None]) - y).T, X), axis=0)) \

        + (reg / m) \* Theta

def f\_optimizacion(Theta, X, y, reg):

    return f\_coste(Theta, X, y, reg), f\_gradiente(Theta, X, y, reg)

def get\_optimize\_theta(X, y, reg):

    initial\_theta = np.zeros((X.shape[1], 1))

    optTheta = opt.minimize(fun=f\_optimizacion, x0=initial\_theta,

            args=(X, y, reg), method='TNC', jac=True,

            options={'maxiter': 200})

    return optTheta.x

########################################################################

###############                                        #################

###############  NORMALIZACION DE MATRICES POLINOMICAS #################

###############                                        #################

########################################################################

def get\_polynomial\_matrix(X, Xval, Xtest, p):

    # X

    X\_pol = polinomial\_matrix(X, p)

    X\_pol, mu, sigma = normalize\_matrix(X\_pol)

    X\_pol = np.hstack((np.ones((X\_pol.shape[0], 1)), X\_pol))

    # Xval

    Xval\_pol = polinomial\_matrix(Xval, p)

    Xval\_pol = (Xval\_pol - mu) / sigma

    Xval\_pol = np.hstack((np.ones((Xval\_pol.shape[0], 1)), Xval\_pol))

    # Xtest

    Xtest\_pol = polinomial\_matrix(Xtest, p)

    Xtest\_pol = (Xtest\_pol - mu) / sigma

    Xtest\_pol = np.hstack((np.ones((Xtest\_pol.shape[0], 1)), Xtest\_pol))

    return X\_pol, Xval\_pol, Xtest\_pol, mu, sigma

def polinomial\_matrix(X, p):

    X\_poly = X

    for i in range(1, p):

        X\_poly = np.column\_stack((X\_poly, np.power(X, i+1)))

    return X\_poly

def normalize\_matrix(X):

    mu = np.mean(X, axis=0)

    X\_norm = X - mu

    sigma = np.std(X\_norm, axis=0)

    X\_norm = X\_norm / sigma

    return X\_norm, mu, sigma

########################################################################

######################                            ######################

######################  APARTADOS DE LA PRACTICA  ######################

######################                            ######################

########################################################################

def polynomial\_regression(X, y, X\_pol, mu, sigma, reg, p, axs):

    Theta = get\_optimize\_theta(X\_pol, y, reg)

    dibuja\_polynomial\_regression(Theta, X, y, mu, sigma, reg, p, axs)

def learning\_curve(X, y, Xval, yval, reg, axs):

    m = len(X)

    error\_train = np.zeros((m, 1))

    error\_val = np.zeros((m, 1))

    for i in range(1, m + 1):

        Theta = get\_optimize\_theta(X[: i], y[: i], reg)

        error\_train[i - 1] = f\_optimizacion(Theta, X[: i], y[: i], 0)[0]

        error\_val[i - 1] = f\_optimizacion(Theta, Xval, yval, 0)[0]

    dibuja\_learning\_curve(error\_train, error\_val, reg, axs)

def regression(X, y, X\_pol, Xval\_pol, yval, mu, sigma, reg, p, axs):

    polynomial\_regression(X, y, X\_pol, mu, sigma, reg, p, axs)

    learning\_curve(X\_pol, y, Xval\_pol, yval, reg, axs)

########################################################################

################################        ################################

################################  MAIN  ################################

################################        ################################

########################################################################

def main():

    data = loadmat("ex5data1.mat")

    y = data["y"]

    X = data["X"]

    yval = data["yval"]

    Xval = data["Xval"]

    ytest = data["ytest"]

    Xtest = data["Xtest"]

    p = 8

    X\_pol, Xval\_pol, Xtest\_pol, mu, sigma = get\_polynomial\_matrix(X, Xval, Xtest, p)

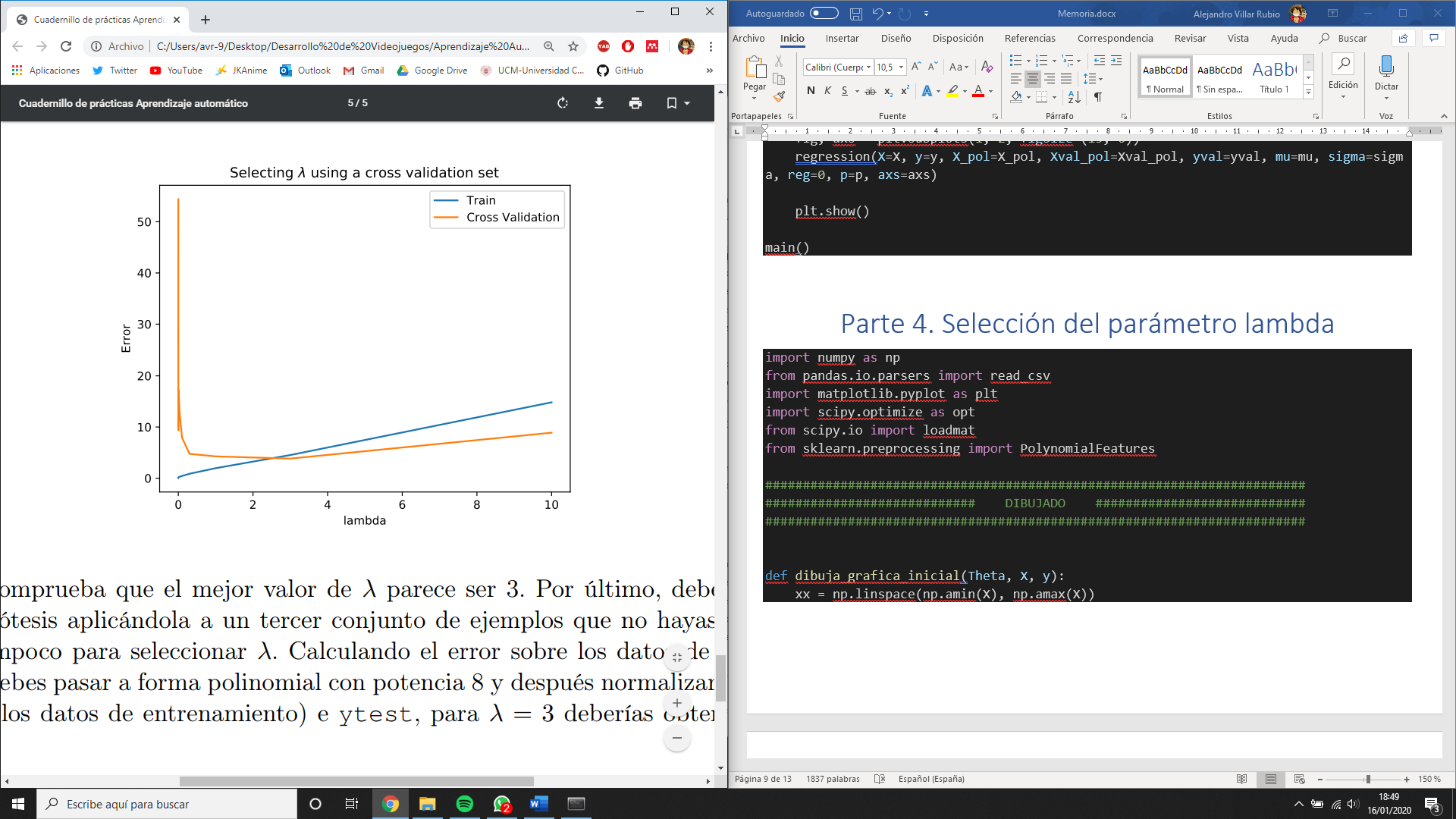
    fig, axs = plt.subplots(1, 2, figsize=(15, 6))

    regression(X=X, y=y, X\_pol=X\_pol, Xval\_pol=Xval\_pol, yval=yval, mu=mu, sigma=sigma, reg=0, p=p, axs=axs)

    plt.show()

main()

# Parte 4. Selección del parámetro lambda



import numpy as np

from pandas.io.parsers import read\_csv

import matplotlib.pyplot as plt

import scipy.optimize as opt

from scipy.io import loadmat

from sklearn.preprocessing import PolynomialFeatures

########################################################################

############################    DIBUJADO    ############################

########################################################################

def dibuja\_grafica\_inicial(Theta, X, y):

    xx = np.linspace(np.amin(X), np.amax(X))

    plt.scatter(X, y, marker='x', c='red', s=100, linewidths=0.5)

    xx = xx[:, None]

    xx\_ones = np.hstack((np.ones((xx.shape[0], 1)), xx))

    plt.plot(xx, h(xx\_ones, Theta[:, None]))

    plt.xlabel('Change in water level (x)')

    plt.ylabel('Water flowing out of the dam (y)')

def dibuja\_learning\_curve(error\_train, error\_val, reg, axs):

    m = len(error\_train)

    axs[1].plot(range(1, m + 1), error\_train, label='Train')

    axs[1].plot(range(1, m + 1), error\_val, label='Cross Validation')

    axs[1].title.set\_text(

        "Learning curve for linear regression " + r'$(\lambda = {})$'.format(reg))

    axs[1].set\_xlabel('Number of training examples')

    axs[1].set\_ylabel('Error')

    axs[1].legend()

def dibuja\_polynomial\_regression(Theta, X, y, mu, sigma, reg, p, axs):

    axs[0].scatter(X, y, marker='x', c='red', linewidths=0.5, s=100)

    axs[0].title.set\_text(

        "Polinomial regression " r'$(\lambda = {})$'.format(reg))

    axs[0].set\_xlabel('Change in water level (x)')

    axs[0].set\_ylabel('Water flowing out of the dam (y)')

    x = np.array(np.arange(min(X) - 5, max(X) + 5, 0.05))

    X\_pol = polinomial\_matrix(x, p)

    X\_pol = (X\_pol - mu) / sigma

    X\_pol = np.insert(X\_pol, 0, 1, axis=1)

    axs[0].plot(x, np.dot(X\_pol, Theta))

def dibuja\_lambda\_selection(lambda\_vec, error\_train, error\_val):

    plt.figure(figsize=(8, 6))

    plt.xlabel('$\lambda$')

    plt.ylabel('Error')

    plt.title('Selecting $\lambda$ using a cross validation set')

    plt.plot(lambda\_vec, error\_train, label='Train')

    plt.plot(lambda\_vec, error\_val, label='Cross Validation')

    plt.legend()

########################################################################

################  CALCULOS DE COSTE, GRADIENTE Y THETA  ################

########################################################################

def h(X, Theta):

    return np.dot(X, Theta)

def f\_coste(Theta, X, y, reg):

    m = len(X)

    return (1 / (2 \* m)) \* np.sum(np.square(h(X, Theta[:, None]) - y), initial=1) + (reg / (2 \* m)) \* np.sum(np.square(Theta), initial=1)

def f\_gradiente(Theta, X, y, reg):

    m = len(X)

    return (1 / m) \* (np.sum(np.dot((h(X, Theta[:, None]) - y).T, X), axis=0)) \

        + (reg / m) \* Theta

def f\_optimizacion(Theta, X, y, reg):

    return f\_coste(Theta, X, y, reg), f\_gradiente(Theta, X, y, reg)

def get\_optimize\_theta(X, y, reg):

    initial\_theta = np.zeros((X.shape[1], 1))

    optTheta = opt.minimize(fun=f\_optimizacion, x0=initial\_theta,

                            args=(X, y, reg), method='TNC', jac=True,

                            options={'maxiter': 200})

    return optTheta.x

########################################################################

###############  NORMALIZACION DE MATRICES POLINOMICAS #################

########################################################################

def get\_polynomial\_matrix(X, Xval, Xtest, p):

    # X

    X\_pol = polinomial\_matrix(X, p)

    X\_pol, mu, sigma = normalize\_matrix(X\_pol)

    X\_pol = np.insert(X\_pol, 0, 1, axis=1)

    # Xval

    Xval\_pol = polinomial\_matrix(Xval, p)

    Xval\_pol = Xval\_pol - mu

    Xval\_pol = Xval\_pol / sigma

    Xval\_pol = np.insert(Xval\_pol, 0, 1, axis=1)

    # Xtest

    Xtest\_pol = polinomial\_matrix(Xtest, p)

    Xtest\_pol = Xtest\_pol - mu

    Xtest\_pol = Xtest\_pol / sigma

    Xtest\_pol = np.insert(Xtest\_pol, 0, 1, axis=1)

    return X\_pol, Xval\_pol, Xtest\_pol, mu, sigma

def polinomial\_matrix(X, p):

    X\_poly = X

    for i in range(1, p):

        X\_poly = np.column\_stack((X\_poly, np.power(X, i+1)))

    return X\_poly

def normalize\_matrix(X):

    mu = np.mean(X, axis=0)

    X\_norm = X - mu

    sigma = np.std(X\_norm, axis=0)

    X\_norm = X\_norm / sigma

    return X\_norm, mu, sigma

########################################################################

######################  APARTADOS DE LA PRACTICA  ######################

########################################################################

def polynomial\_regression(X, y, X\_pol, mu, sigma, reg, p, axs):

    Theta = get\_optimize\_theta(X\_pol, y, reg)

    dibuja\_polynomial\_regression(Theta, X, y, mu, sigma, reg, p, axs)

def learning\_curve(X, y, Xval, yval, reg, axs):

    m = len(X)

    error\_train = np.zeros((m, 1))

    error\_val = np.zeros((m, 1))

    for i in range(1, m + 1):

        Theta = get\_optimize\_theta(X[: i], y[: i], reg)

        error\_train[i - 1] = f\_optimizacion(Theta, X[: i], y[: i], 0)[0]

        error\_val[i - 1] = f\_optimizacion(Theta, Xval, yval, 0)[0]

    dibuja\_learning\_curve(error\_train, error\_val, reg, axs)

def regression(X, y, X\_pol, Xval\_pol, yval, mu, sigma, reg, p, axs):

    polynomial\_regression(X, y, X\_pol, mu, sigma, reg, p, axs)

    learning\_curve(X\_pol, y, Xval\_pol, yval, reg, axs)

def lambda\_selection(X, y, Xval, yval):

    lambda\_vec = np.array([0, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3, 10])

    error\_train = np.zeros((len(lambda\_vec), 1))

    error\_val = np.zeros((len(lambda\_vec), 1))

    for i in range(len(lambda\_vec)):

        reg = lambda\_vec[i]

        Theta = get\_optimize\_theta(X, y, reg)

        error\_train[i] = f\_optimizacion(Theta, X, y, 0)[0]

        error\_val[i] = f\_optimizacion(Theta, Xval, yval, 0)[0]

    print('lambda\tTrain Error\tValidation Error\n')

    for i in range(len(lambda\_vec)):

        print('{}\t{}\t{}\n'.format(

            lambda\_vec[i], error\_train[i], error\_val[i]))

    dibuja\_lambda\_selection(lambda\_vec, error\_train, error\_val)

    return lambda\_vec[np.argmin(error\_val)]

def test\_error(X, y, Xtest, ytest, reg):

    Theta = get\_optimize\_theta(X, y, reg)

    error\_test = f\_optimizacion(Theta, Xtest, ytest, 0)[0]

    print("Test error for the best lambda: {0:.4f}".format(error\_test))

########################################################################

################################  MAIN  ################################

########################################################################

def main():

    data = loadmat("ex5data1.mat")

    y = data["y"]

    X = data["X"]

    yval = data["yval"]

    Xval = data["Xval"]

    ytest = data["ytest"]

    Xtest = data["Xtest"]

    p = 8

    X\_pol, Xval\_pol, Xtest\_pol, mu, sigma = get\_polynomial\_matrix(

        X, Xval, Xtest, p)

    bestLambda = lambda\_selection(X\_pol, y, Xval\_pol, yval)

    print("Best lambda value:", bestLambda)

    test\_error(X\_pol, y, Xtest\_pol, ytest, bestLambda)

    plt.show()

main()