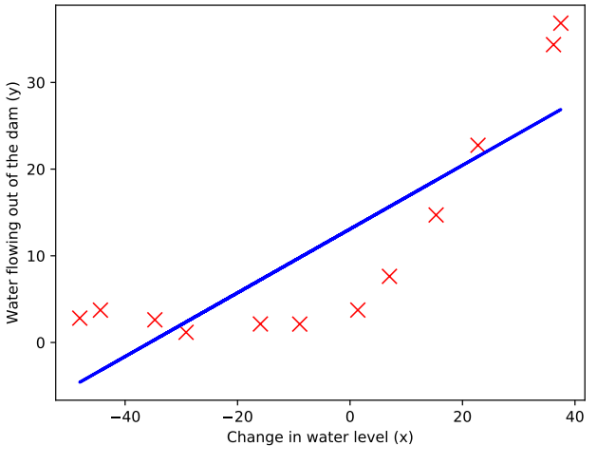
PRÁCTICA 5. MEMORIA TÉCNICA

Sergio Gavilán Fernández sgavil01@ucm.es

Alejandro Villar Rubio alvill04@ucm.es

Parte 1. Regresión lineal regularizada

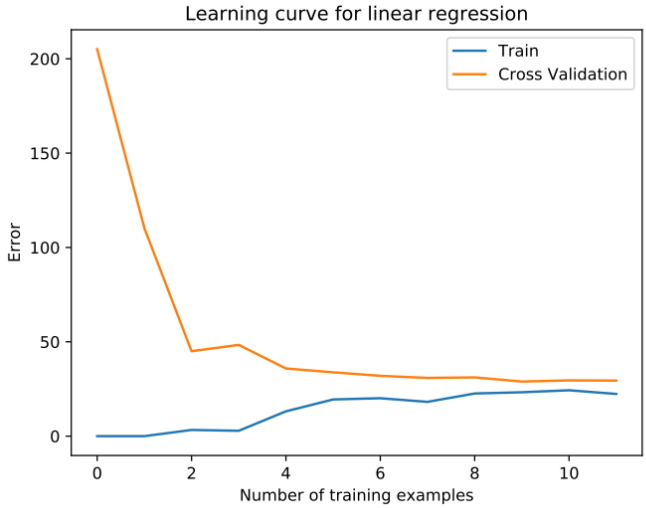



```
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
import scipy.optimize as opt
from scipy.io import loadmat

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

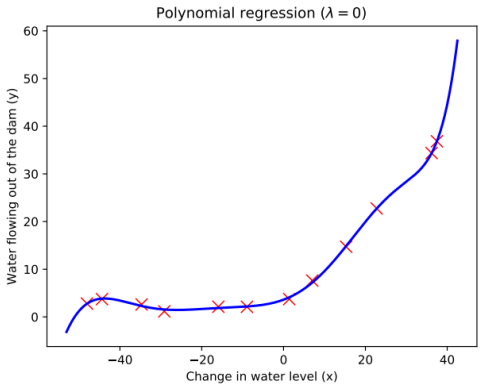



```
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
import scipy.optimize as opt
from scipy.io import loadmat

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



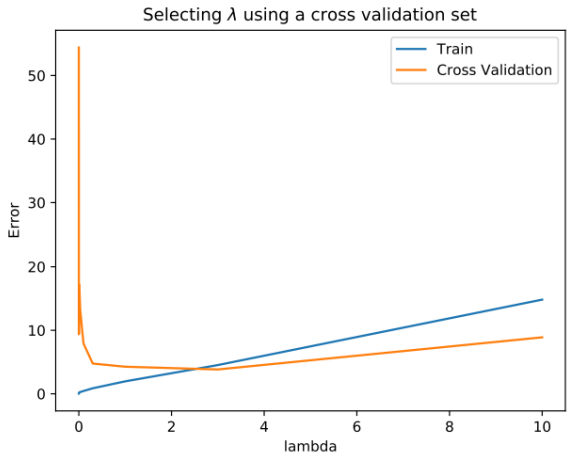

```
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt
import scipy.optimize as opt
from scipy.io import loadmat
from sklearn.preprocessing import PolynomialFeatures

#####################################################################
##############################                 ##############################
##############################    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=sigm
a, 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()
```