Práctica 5 (Train & Cross Validation)

¿Cómo he planteado la práctica?

Primero necesitamos los datos, en este caso leemos el .MAT ‘ex5data1’, y separamos los datos en X e Y, Xval e Yval, Xtest e Ytest.

Posteriormente hay que implementar la regresión lineal multivariable regularizada, que devolverá el coste y la gradiente.

Finalmente hay que dibujar la recta de resultados y la comparativa de los errores de aprendizaje.

Código comentado y gráficas

1. Importar las librerías

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt

from scipy.io import loadmat
```

2. Obtener los datos del CSV:

```
data = loadmat('ex5data1.mat')

X = data['X']
Y = data['y'].ravel()
X_val = data['Xval']
Y_val = data['yval'].ravel()
X_test = data['Xtest']
Y_test = data['ytest'].ravel()

m = X.shape[0]
X = np.hstack([np.ones([m, 1]), X])
m_val = X_val.shape[0]
X_val = np.hstack([np.ones([m_val, 1]), X_val])
```

3. Ahora implementamos la regresión lineal multivariable, por un parte el coste, por otra parte la gradiente, y finalmente regularizamos ambas

```python
def reg_lin_mul_var (theta, X, Y, lambd):

    m = X.shape[0]
    n = X.shape[1]
    J_theta = 0
    gradient = []

    H = np.dot(X, theta)
    diff = (H - Y)

    # Coste
    J_theta = (1 / (2 * m)) * np.sum(np.square(diff))

    # Coste Regularizado
    reg = lambd / (2 * m) * np.sum(np.square(theta[1:]))
    J_theta += reg

    # Gradiente
    gradient = (1 / m) * np.sum(diff * X.T, axis=1)

    # Gradiente Regularizada
    reg = (lambd/m) * theta[1:]
    gradient[1:] += reg


    return (J_theta, gradient)
```

4. Y comprobamos los resultados

```python
# PRUEBAS
theta = np.array([1, 1])
lambd = 1
cost, gradient = reg_lin_mul_var(theta, X, Y, lambd)
print("El coste es:\t\t{}\nLa gradiante es:\t{}".format(cost, gradient))
```

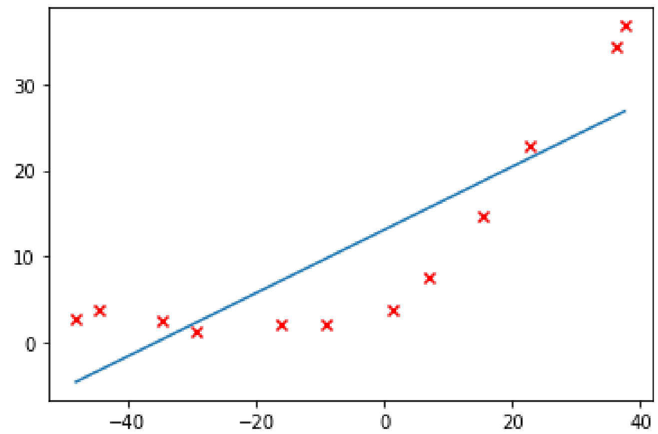
```
El coste es:            303.9931922202643
La gradiante es:        [-15.30301567 598.25074417]
```

5. Posteriormente minimizamos con la función MINIMIZE de Scipy, con $\lambda = 0$, el método escogido en este, y los siguientes puntos es 'TNC' → Truncated Newton

```
res = opt.minimize(
    fun = reg_lin_mul_var,
    x0 = theta,
    args=(X, Y, 0),
    method='TNC',
    jac=True
)

theta_opt = res.x
```

6. Finalmente mostramos la recta de resultados, como en anteriores prácticas, nos quedamos con el mínimo y máximo, y predecimos la Y con las $\theta_{OPTIMAS}$

```
plt.figure()
plt.scatter(X[:, 1:], Y, marker='x', c='red')

X_min, X_max = np.min(X, axis=0), np.max(X, axis=0)
Y_min = np.sum(theta_opt * X_min)
Y_max = np.sum(theta_opt * X_max)
X_min = X_min[1:]
X_max = X_max[1:]

plt.plot([X_min, X_max], [Y_min, Y_max])

plt.show()
```



Podemos apreciar un alto sesgo (high bias), ya que los datos no se ajustan a la gráfica

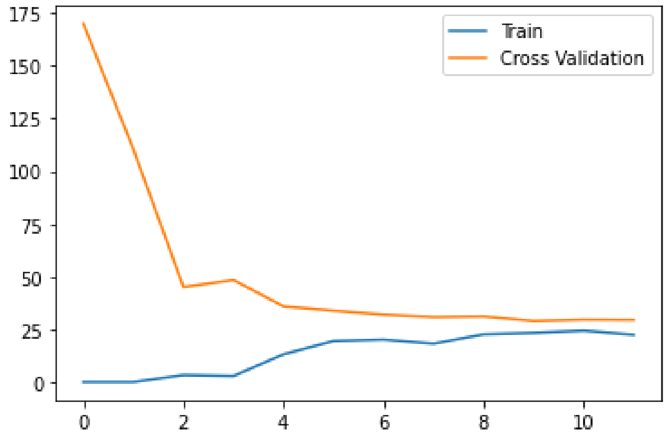
7. Ahora calculamos la tasa de error de aprendizaje, acumulando todos los ejemplos de X (X_i), el error se calcula con la regresión lineal multivariable y con $\lambda = 0$. Esta función devuelve un array concatenado de todos los errores.

```python
def learningErrors(theta, X, Y, X_val, Y_val, lambd = 0):
    """
    Returns the Train error vector and the Cross Validation error vector
    """

    trainError = []
    cvError = []
    m = X.shape[0]

    for i in range(1, m + 1):

        X_i = X[0:i]
        Y_i = Y[0:i]

        res = opt.minimize(
            fun = reg_lin_mul_var,
            x0 = theta,
            args=(X_i, Y_i, lambd),
            method='TNC',
            jac=True
        )

        trainError.append(reg_lin_mul_var(res.x, X_i, Y_i, lambd)[0])
        cvError.append(reg_lin_mul_var(res.x, X_val, Y_val, lambd)[0])

    return np.array(trainError), np.array(cvError)
```

8. Finalmente, dibujamos el error de aprendizaje apoyándonos en la función anterior

```
def printLearningErrors(trainError, cvError):
    """
    Prints the Train error and the Cross Validation
    Doesnt return anything
    """

    plt.figure()

    r = range(0, len(trainError))

    plt.plot(r, trainError)
    plt.plot(r, cvError)
    plt.legend(['Train', 'Cross Validation'])

    plt.show()
```

```
trainError, cvError = learningErrors(theta, X, Y, X_val, Y_val)
printLearningErrors(trainError, cvError)
```



Como habíamos comentado anteriormente, tenemos un alto sesgo, ya que el error de la Cross Validation es relativamente alto.

Práctica 5 (Regresión polinomial)

¿Cómo he planteado la práctica?

Primero necesitamos los datos, en este caso leemos el .MAT 'ex5data1', y separamos los datos en X e Y, Xval e Yval, Xtest e Ytest.

Posteriormente hay que implementar una función que convierte los datos en vectores polinómicos $x_1, x_2^2, \ldots, x_p^p$, y las funciones de normalización, una para normalizar un valor, y otra para normalizar a partir de unos valores μ y σ.

En el siguiente paso, calculamos las predicciones y comprobamos si se ajusta a los datos.

Finalmente probamos varios parámetros λ y dibujamos el error que tiene respecto a su incremento, para así sacar un λ_{OPTIMO}.

Código comentado y gráficas

1. Importar las librerías

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as opt

from scipy.io import loadmat
```

2. Obtener los datos del CSV:

```python
data = loadmat('ex5data1.mat')

X = data['X']
Y = data['y'].ravel()
X_val = data['Xval']
Y_val = data['yval'].ravel()
X_test = data['Xtest']
Y_test = data['ytest'].ravel()

m = X.shape[0]
m_val = X_val.shape[0]
```

3. Creamos la función polinómica que nos devuelve un vector con $x_1, x_2^2, \dots, x_p^p$:

```python
def poly_data(X, p=1):
    """
    INPUT: Matix (m, 1) && p
    OUTPUT: Matrix (m, p)
    """
    res = np.zeros((X.shape[0], p))

    for i in range(0, p):
        res[:, i] = X ** (i + 1)

    return res
```

4. Implementamos una función de normalización a partir de un vector, con la siguiente fórmula: $\frac{X-\mu}{\sigma}$
   Además, añade una columna de 1s.

```python
def normalize(X):
    m = X.shape[0]
    mu = np.array([np.mean(X, axis = 0)])
    sigma = np.array([np.std(X, axis = 0)])
    X_norm = np.divide(np.subtract(X, mu), sigma)

    X_norm = np.hstack([np.ones([m, 1]), X_norm])

    return X_norm, mu, sigma
```

   Por otro lado, implementamos una función de normalización a partir de unos valores μ y σ

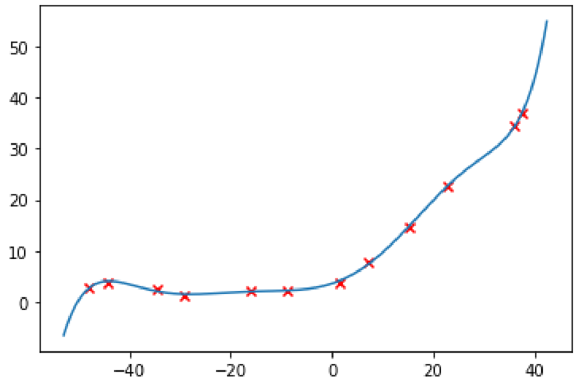
```python
def normalize_with_values(X, mu, sigma):
    m = X.shape[0]

    X_norm = np.divide(np.subtract(X, mu), sigma)
    X_norm = np.hstack([np.ones([m, 1]), X_norm])

    return X_norm
```

5. Ahora usamos la función de MINIMIZE para obtener las $\theta_{OPTIMAS}$, para ello primero convertimos la X en polinómica y después la normalizamos (Recordar que también se añade la columna de 1s).

```
p = 8
theta = np.zeros(p + 1)
lambd = 0

X_poly = poly_data(X.ravel(), p)
X_poly_norm, mu, sigma = normalize(X_poly)

res = opt.minimize(
    fun = reg_lin_mul_var,
    x0 = theta,
    args=(X_poly_norm, Y, lambd),
    method='TNC',
    jac=True
)

theta_opt = res.x
```

6. Y dibujamos la recta de resultados para comprobar si la polinomización se ajusta a los datos:

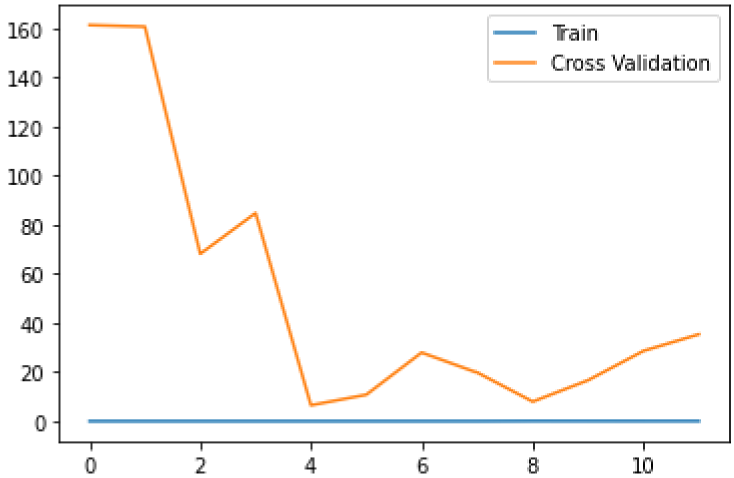
```
# Range of values -> -5 and +5 to show beyond the actual data
ranX = np.arange(
    np.min(X) - 5,
    np.max(X) + 5,
    .05
)

# Need to normalize with mu and sigma
ranX_poly_norm = normalize_with_values(poly_data(ranX, p), mu, sigma)

# Predict values
Y_hat = np.dot(ranX_poly_norm, theta_opt)

# Plot the figure
plt.figure()
plt.scatter(X, Y, marker='x', c='red')
plt.plot(ranX, Y_hat)
plt.show()
```



Y podemos comprobar que sí, se ajusta a los datos, quizás demasiado, por ello vamos a dibujar la curva de aprendizaje.

7. Dibujamos la curva de aprendizaje, para ello necesitamos polinomizar y normalizar ambas X, la X_{TRAIN} y la X_{CV}.

```
X_poly_norm, mu, sigma = normalize(poly_data(X.ravel(), p))

# Need to normalize with mu and sigma
X_val_poly_norm = normalize_with_values(
    poly_data(X_val.ravel(), p),
    mu,
    sigma
)

trainError, cvError = learningErrors(theta,
                                     X_poly_norm,
                                     Y,
                                     X_val_poly_norm,
                                     Y_val,
                                     0
                                    )

printLearningErrors(trainError, cvError)
```

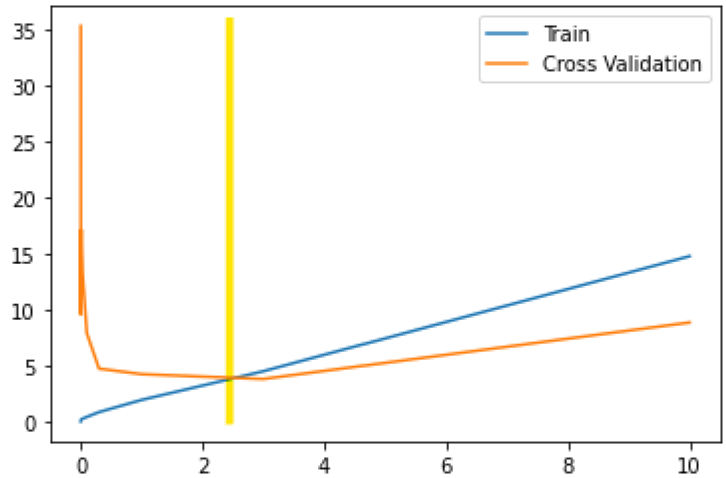


La curva de aprendizaje en el Train set es bajo y estable, lo cual está bien, por otro lado, la Cross Validation es baja pero irregular, por lo que necesitaríamos más datos para conseguir que bajase, o más iteraciones.

NOTA: Recordar que en la curva de aprendizaje λ siempre debe ser 0

8. Ahora dibujamos varias λ y su error, para conocer la λ_{OPTIMA}, al igual que antes, polinomizamos y normalizamos.

```
lambdas = [ 0, .001, .003, .01, .03, .1, .3, 1, 3, 10 ]
p = 8
trainError_vector = []
cvError_vector = []

plt.figure()

X_poly_norm, mu, sigma = normalize(poly_data(X.ravel(), p))

X_val_poly_norm = normalize_with_values(
    poly_data(X_val.ravel(), p),
    mu,
    sigma
)


for lambd in lambdas:
    theta = np.zeros(p + 1)

    res = opt.minimize(
        fun = reg_lin_mul_var,
        x0 = theta,
        args=(X_poly_norm, Y, lambd),
        method='TNC',
        jac=True,
    )

    theta_opt = res.x

    trainError_vector.append(reg_lin_mul_var(theta_opt, X_poly_norm, Y, 0)[0])
    cvError_vector.append(reg_lin_mul_var(theta_opt, X_val_poly_norm, Y_val, 0)[0])

plt.plot(lambdas, trainError_vector, label="Train")
plt.plot(lambdas, cvError_vector, label="Cross Validation")
plt.legend()
plt.show()
```



Podemos observar que el valor λ_{OPTIMO} está en el intervalo $2 < \lambda < 3$

9. Finalmente, el valor más cercano a λ_{OPTIMA} que hemos probado es 3, por lo que comprobamos el error, y en este caso es de 3.5720, como pide el enunciado.

```
lambd_opt = 3
p = 8
theta = np.zeros(p + 1)

X_poly_norm, mu, sigma = normalize(poly_data(X.ravel(), p))

res = opt.minimize(
    fun = reg_lin_mul_var,
    x0 = theta,
    args=(X_poly_norm, Y, lambd_opt),
    method='TNC',
    jac=True,
)

X_test_poly_norm = normalize_with_values(
    poly_data(X_test.ravel(), p),
    mu,
    sigma
)

reg_lin_mul_var(res.x, X_test_poly_norm, Y_test, 0)[0]
```

```
3.572044856986457
```