Javier Cordero Calvo y Juan Ruiz Jiménez

Aprendizaje Automático y Minería de Datos – Práctica 5

1. Regresión lineal regularizada:

Para la primera parte de la práctica, en primer lugar cargaremos en la variable data y gracias a la función loadmat el archivo “ex5data1.mat”.

Tras esto, guardaremos en sus correspondientes variables las tablas de X e y.

Definiremos lambda a 0 y ampliaremos la matriz de X con la fila de 1’s necesaria, así como la variable theta cuya dimensión es una de las filas de X.

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

X = data["X"]
y = data["y"]
landa = 0

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

theta = np.ones(XwithOnes.shape[1])
```

Una vez temenos todo dispuesto, calcularemos la theta óptima de nuestro sistema mediante la función calcOptTheta, la cual recibe como parámetros las matrices de X e y, el valor de lambda y el vector de thetas.

```
thetaOpt = calcOptTheta(XwithOnes, y, landa, theta)
```

Esta función, calculará la theta óptima gracias a las funciones coste y gradiente.

```
def calcOptTheta(X, Y, landa, theta):
    result = opt.fmin_tnc(func=coste, x0=theta, fprime=gradiente,
      args=(X, Y, landa, X.shape[0]))
    return result[0]
```

```
#Función para el coste
def coste(theta, X, y, landa, m):
    h = np.dot(X, theta[:, None])

    thetaAux = np.delete(theta, 0, 0)
    return ((1 / (2 * m)) * (np.sum(np.square(h - y)))) + ((landa /
 (2 * m)) * np.sum(np.square(thetaAux)))
```

```
#Función para calculo de gradiente
def gradiente(theta, XX, Y, landa, m):
    h = np.dot(XX, theta[:, None])

    #Eliminamos la primera columna de theta
    thetaAux = np.delete(theta, 0, 0)
    return (1 / m) * np.matmul(XX.T, h - Y)+((landa/m) * thetaAux)
```

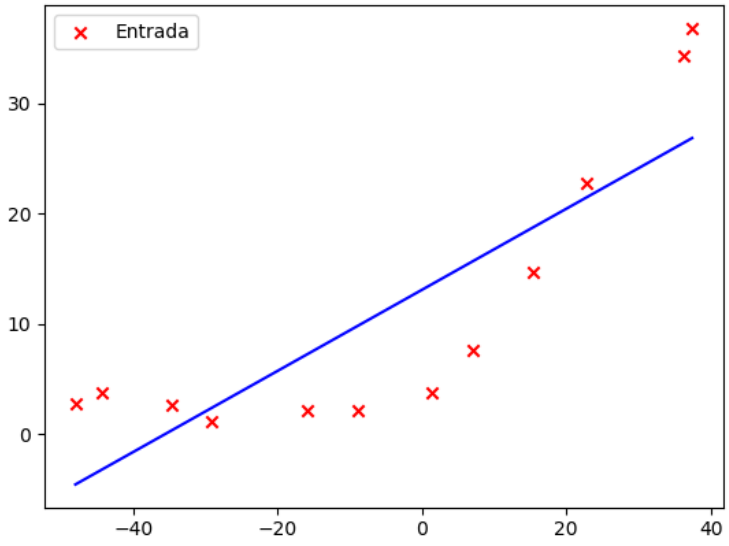
Destacar que ambas funciones están vectorizadas y que ambos resultados, tanto el de gradiente como el de coste son los esperados por el enunciado sin minimizar aún el coste de la función.

Finalmente, pintaremos los puntos de los datos de ejemplo y la recta obtenida gracias a la thetaOpt objetinda, la matriz de X y la de y's.

```
pinta_puntos(X, y)
pinta_Recta(thetaOpt, XwithOnes, y)

plt.legend()
plt.show()
```

```
def pinta_puntos(X, Y):
    plt.scatter(X, Y, marker = 'x', c = 'red', label = 'Entrada')

def pinta_Recta(T, x, y):
    x =  np.arange(np.min(x[:, 1]), np.max(x[:, 1]), 0.1)
    y = x.copy()

    a = len(x)
    i = 0

    while (i < a):
            y[i] = (x[i] * T[1]) + T[0]
            i = i + 1

    plt.plot(x, y, c='blue')
```

El resultado obtenido es una recta demasiado simple para ajustarse a los casos de entrenamiento, por lo que predice valores sesgados.



2. Curvas de aprendizaje:

En esta segunda parte cargaremos los datos igual que en el apartado anterior, aunque añadiremos Xval e yval, que son los datos de validación proporcionados.

Igual que en el apartado anterior, tanto X como Xval han de ser rellenados con la columna de 1's correspondiente, al igual que se debe crear la matriz de thetas con tamaño una fila de X.

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

X = data["X"]
y = data["y"]
Xval = data["Xval"]
yval = data["yval"]

landa = 0

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

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

theta = np.ones(XwithOnes.shape[1])
```

Una vez tengamos todo preparado, nos disponemos a llamar a la función curva_aprendizaje, la cual calculará el error de nuestro sistema de aprendizaje así como el de los ejemplos de entrenamiento, y devolverá dos arrays, cada uno con el error cometido por cada ejemplo en un punto determinado.

```
err1, err2 = curva_aprendizaje(XwithOnes, y, landa, theta,
 XvalWithOnes, yval)
```

Esta función, en primer lugar, crea dos vectores para los errores que vamos a guardar. Pese a que cada uno se le asigna la longitud correspondiente de cada conjunto de ejemplo a evaluar, finalmente los dos tendrán la longitud del menor de los ejemplos de entrenamiento.

Tras tener ambos vectores, en un bucle del tamaño de la longitud de X, calcularemos la Theta óptima para el vector [0, i], donde i es el punto del vector por el que vamos recorriendo, ya que tenemos que determinar el error por cada parte del vector de X.

Una vez tenemos la theta calculada, procedemos a calcular el error con dicha theta para nuestros datos de X así como para los datos de validación y lo añadimos a su correspondiente vector.

```
def curva_aprendizaje(X, y, landa, theta, Xval, yval):

    err1 = np.zeros((len(X)))
    err2 = np.zeros((len(Xval)))

    i = 1
    while (i < len(X) + 1):
        thetas = calcOptTheta(X[0:i], y[0:i], landa, theta)

        err1[i - 1] = calc_error(thetas, X[0:i], y[0:i], landa, len(X))
        err2[i - 1] = calc_error(thetas, Xval, yval, landa, len(Xval))
        i += 1

    return err1, err2
```

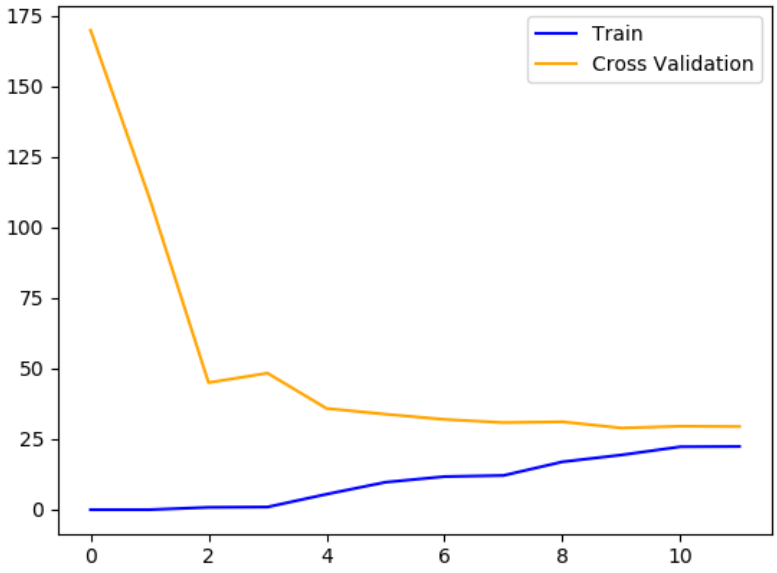
La función calcOptTheta es la misma que la del ejemplo anterior, así como las funciones de coste y gradiente que derivan de ella. Sin embargo, por claridad, pese a que la función calc_error no deja de ser la función de coste, se ha decidido hacer una función a parte.

```
def calc_error(theta, X, y, landa, m):
    h = np.dot(X, theta[:, None])
    return (1 / (2 * m)) * ((np.sum(np.square(h - y))))
```

Una vez tenemos ambos errores calculados, finalmente los pintamos con la función pinta_Curva_Aprendizaje:

```
pinta_Curva_Aprendizaje(err1, err2)

def pinta_Curva_Aprendizaje(err1, err2):

    a = np.arange(len(err1))
    b = err1
    plt.plot(a, b, c="blue", label="Train")

    d = err2[0:len(err1)]
    plt.plot(a, d, c="orange", label="Cross Validation")
```

El resultado finalmente obtenido después de realizar los cálculos es que el aprendizaje está sesgado, por lo que habrá que ajustarse mejor a los ejemplos de entrenamiento.



3. Regresión polinomial:

La carga de datos de esta parte de la práctica es exactamente igual a la anterior, a excepción de que añadiremos una nueva variable p para crear nuestras nuevas matrices.

En primer lugar, creamos la función transforma_entrada, la cual recibe una matriz X de mx1 y un número p y devuelve una matriz mxp, donde cada columna de p es un vector con X^p[i].

Tras tener esta matriz, como tiene los datos muy dispersos, será necesario normalizarla. Para ello crearemos la función normaliza_Matriz, que recibe una matriz y

la devuelve normalizada e incluye también los parámetros mu y sigma que han sido necesarios para normalizarla y que en el futuro tendremos que utilizar.

Finalmente, insertaremos la columna de 1's necesaria a esta nueva matriz generada.

```
nuevaentrada = transforma_entrada(X, p)
nuevaentrada, mu, sigma = normaliza_Matriz(nuevaentrada)
nuevaentrada = np.insert(nuevaentrada, 0, 1, axis=1)
```

```
def transforma_entrada(X, p):
    nX = X
    for i in range(1, p):
        nX = np.column_stack((nX, np.power(X, i+1)))

    return nX
```

```
def normaliza_Matriz(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
```

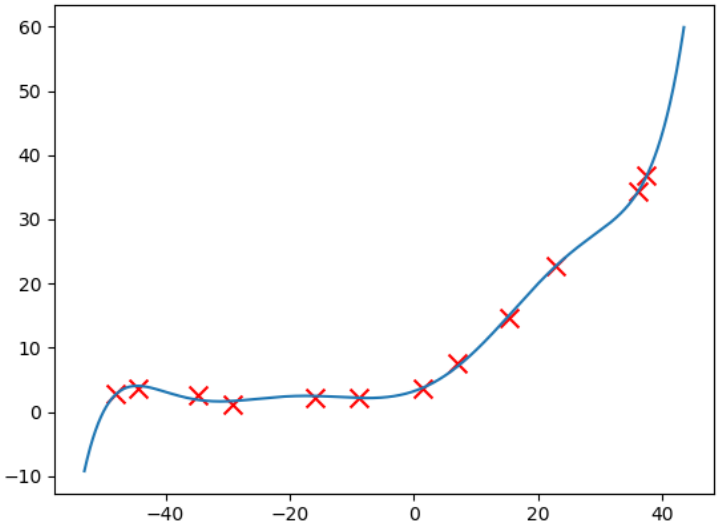
Tras tener en nuevaentrada la, valga la redundancia, nueva entrada del ejercicio ya normalizada, calculamos la theta óptima con la función calcOptTheta previamente descrita, aunque algo cambiada para ajustarla a los ejemplos de esta parte y con más funciones intermedias que faciliten la claridad de código y la función minimize.

Tras tener la theta óptima, pintaremos la regresión polinomial así como los puntos de entrada del problema.

```
thetaOpt = calcOptTheta(nuevaentrada, y, landa)
pinta_regresion_Polinomial(X, p, mu, sigma, thetaOpt)
```

```
def pinta_regresion_Polinomial(X, p, mu, sigma, theta):
    x = np.array(np.arange(min(X) - 5,  max(X) + 6, 0.02))
    nX = transforma_entrada(x, p)
    nX = nX - mu
    nX = nX / sigma
    nX = np.insert(nX, 0, 1, axis=1)
    plt.plot(x, np.dot(nX, theta))
```

El resultado obtenido es el siguiente:

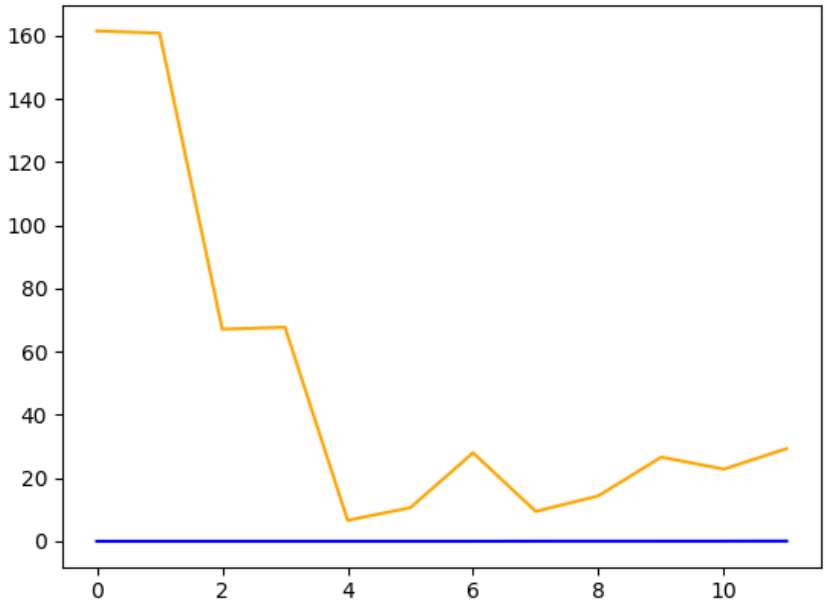


Para essta segunda parte, calcularemos los errores producidos, al igual que en el apartado anterior, de X y de los datos de validación, aunque ambos normalizados y ampliados.

Al igual que hicimos con X, con Xval generamos una nueva entrada de validación y calculamos ambos errores con estas dos nuevas matrices, exactamente igual que en el ejercicio anterior.

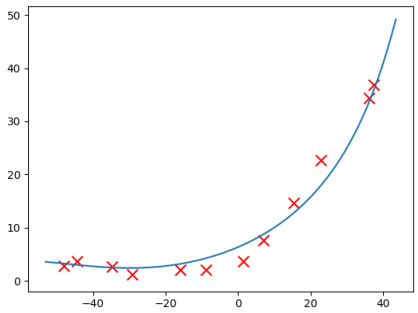
```
neuvaEntradaValidacion = transforma_entrada(Xval, p)
neuvaEntradaValidacion = neuvaEntradaValidacion - mu
neuvaEntradaValidacion = neuvaEntradaValidacion / sigma
neuvaEntradaValidacion = np.insert(neuvaEntradaValidacion, 0, 1,
 axis=1)

err1, err2 = curva_aprendizaje(nuevaentrada, y, landa,
 neuvaEntradaValidacion, yval)
```

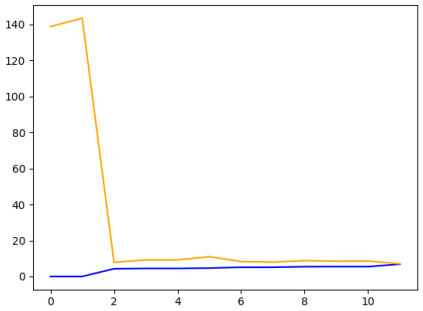
El resultado obtenido para lambda = 0 es que la hipótesis está sobre ajustada a los ejemplos de entrenamiento, como se ve en la siguiente gráfica:



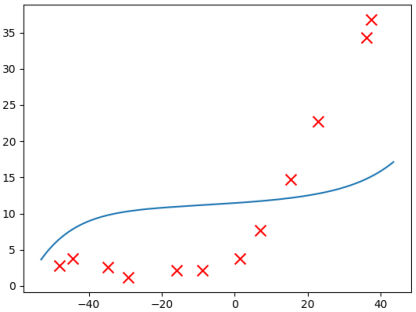
Sin embargo, probaremos lambda = 1 y lambda = 100 para ver la diferencia entre ellas:

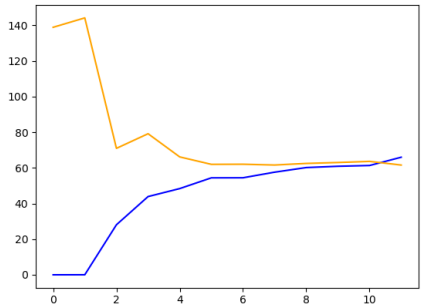
Lambda = 1, la entrada está bien ajustada junto a los ejemplos.





Lambda = 100, la tasa de fallo es demasiado alta.





Cambio en las funciones de coste, gradiante y calcOptTheta:

```
def sigmoide(x):
    return 1 / (1 + np.exp(-x))
#Función para el coste
def coste(theta, X, y, landa, m):
    h = np.dot(X, theta)
    return ((1 / (2 * m)) * (np.sum(np.square(h - y)))) + ((landa / (2 * m)) *
       np.sum(np.square(theta[1:len(theta)])))

#Función para calculo de gradiente
def gradiente(theta, XX, Y, landa, m):
    h = np.dot(XX, theta)
    grad = (1 / m) * np.dot(XX.T, h - Y)+((landa/m) * theta)
    return grad

def coste_y_gradiente(theta, X, Y, landa, m):

    theta = theta.reshape(-1, Y.shape[1])
    cost = coste(theta, X, Y, landa, m)
    grad = gradiente(theta, X, Y, landa, m)
    grad[0] = (1 / m) * np.dot(X.T, np.dot(X, theta) - Y)[0]

    return (cost, grad.flatten())
def calcOptTheta(X, Y, landa):
    theta = np.zeros((X.shape[1], 1))

    def costFunction(theta):
        return coste_y_gradiente(theta, X, Y, landa, len(X))

    result = minimize(fun=costFunction, x0=theta, method='CG', jac=True, optio
ns={'maxiter':200})

    return result.x
```

4. Selección del parámetro λ:

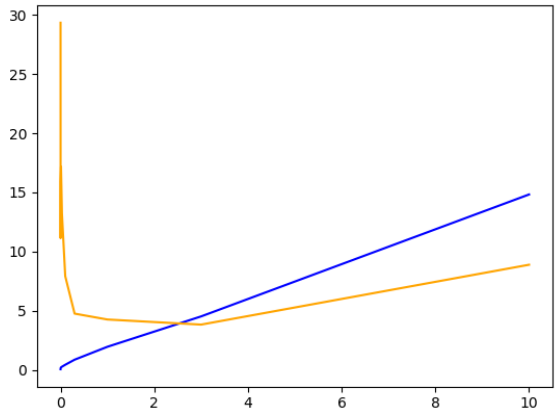
La entrada es igual que la del ejemplo anterior, aunque ahora añadimos Xtest e ytest para comprobar los resultados que obtendremos al entrenar nuestro sistema.

Ahora tampoco contaremos con una única lambda, sino que tendremos un vector de lambdas por el que iremos comprobando cual es el que minimiza el error cometido.

Entrenaremos el sistema igual que hicimos en el ejemplo anterior para obtener los errores y dibujaremos dichos errores, con los datos de entrada y los ejemplos de validación, aunque ahora en lugar de enviar una sola lambda, el método curva_aprendizaje tendrá una dimensión del tamaño del vector de lambdas y el bucle

no recorrerá el tamaño de X, sino las lambdas que tenemos para probar. También hay que tener en cuenta, que para el calculo del error hay que enviar lambda 0 mientras que para el calculo de theta es la lambda que corresponda al índice de ese vector.

```
def curva_aprendizaje(X, y, landa, Xval, yval):

    err1 = np.zeros((len(landa)))
    err2 = np.zeros((len(landa)))

    i = 0
    while (i < len(landa)):
        thetas = calcOptTheta(X, y, landa[i])

        #IMPORTANTE que landa tiene que ser 0 aquí
        err1[i] = coste_y_gradiente(thetas, X, y, 0, len(X))[0]
        err2[i] = coste_y_gradiente(thetas, Xval, yval, 0, len(Xval))[0]
        i += 1

    return err1, err2
```



Tras realizar esto, nos fijamos en que la lambda óptima está en torno a 3, por lo tanto ahora entrenaremos las entradas de test con esta nueva lambda, después de realizar todas las transformaciones necesarias a las matrices de test.

```
landa = 3

neuvaEntradaTest = transforma_entrada(Xtest, p)
neuvaEntradaTest = neuvaEntradaTest - mu
neuvaEntradaTest = neuvaEntradaTest / sigma
neuvaEntradaTest = np.insert(neuvaEntradaTest, 0, 1, axis=1)

optTheta = calcOptTheta(nuevaentrada, y, landa)
```

Ahora que tenemos la theta óptima con la entrada, vamos a ver qué error obtenemos frente a los ejemplos de prueba proporcionados.

```
#Lambda siempre = 0

error_test = coste_y_gradiente(optTheta, neuvaEntradaTest, ytest, 0
,len(neuvaEntradaTest))[0] #[0] para que lo que devuelva sea el
coste

print("Error obtenido de: ", error_test)
```

Finalmente, la salida del sistema es: "***Error obtenido de: 3.572026619015027"***, justo el error que se esperaba en las instrucciones del enunciado.