Javier Cordero Calvo y Juan Ruiz Jiménez

Aprendizaje Automático y Minería de Datos – Práctica 1

Parte 1:

```
datos = carga_csv('ex1data1.csv')

X = datos[:, :-1]
np.shape(X)         # (97, 1)
Y = datos[:, -1]
np.shape(Y)         # (97,)
m = np.shape(X)[0]
n = np.shape(X)[1]

# añadimos una columna de 1's a la X
X = np.hstack([np.ones([m, 1]), X])
alpha = 0.01
```

Preparamos la carga de los datos gracias al método proporcionado “carga_csv”.

Guardamos todos esos datos en sus matrices correspondientes y ampliamos la matriz X con una columna de 1’s. También definimos el Alpha.

```
descenso_gradiente(X, Y, alpha, m, n)
pintaPuntosFuncion(X,Y)
plt.show()
```

Después, llamamos a la función “descenso_gradiante” que realiazará los cálculos que necesitamos. Tras haber resuelto el problema, pintamos en pantalla el resultado de nuestra función.

```
def descenso_gradiente(X, Y, alpha, m, n):
    Thetas = np.zeros(2)
    i = 0

    while (i < 2500):

        Thetas = Thetas - alpha * (1 / m) * ( X.T.dot(np.dot(X, Thetas) -
 Y) )

        coste(X, Y, Thetas)
        costes = []
        costes = np.append(costes, coste(X, Y, Thetas))
        #Costes sirve para depurar y ver si actúan de la manera esperada
(bajando)

        i = i + 1

    pintaRecta(Thetas, X, Y)
```

En la función “descenso_gradiante”, primero creamos un vector de dos componentes llamado “Thetas” y lo inicializamos con 0’s.

Tras esto, realizamos n iteraciones (en este caso 2500) para sacar las Thetas más aproximadas que podamos.

La función coste:

```
def coste(X, Y, T):
    H = np.dot(X, T)
    Aux = (H - Y) ** 2
    return Aux.sum() / (2 * len(X))
```

Nos devuelve el coste menor que alcanza esa función en la X, Y dadas con las Thetas (T) aportadas. Para saber que el algoritmo funciona, esta debe ser cada vez menor.

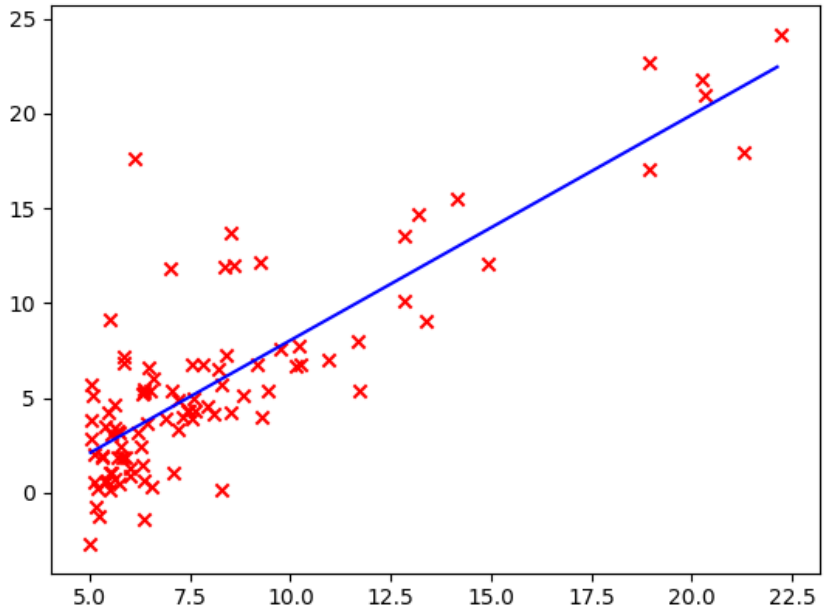
Funciones para pintar:

```
def pintaRecta(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')

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

Función para carga de datos:

```
def carga_csv(file_name):
    """carga el fichero csv especificado y lo
    devuelve en un array de numpy"""
    valores = read_csv(file_name, header=None).values
    # suponemos que siempre trabajaremos con float
    return valores.astype(float)
```

Resultado:



Parte 2:

La primera parte de carga de recursos, así como de inicialización de las matrices es la misma que en la parte uno aunque con un pequeño matiz.

Ya que los datos pueden ser muy dispares, tenemos que normalizarlos con la función “normaliza_datos” tal y como se ve a continuación:

```
def normaliza_datos(X):

    numColumnas = X[1].size

    media = np.zeros(numColumnas)
    varianza = np.zeros(numColumnas)

    for x in range(numColumnas):
        xData = X[:,x]
        media[x] = np.median(xData)
        varianza[x] = np.std(xData)

    X_normalizadas = (X - media) / varianza

    return X_normalizadas
```

Después, ya podemos llamar a la función “descenso_gradiante”:

```
T1 = descenso_gradiente(nX, Y, alpha, m, n)
```

Esta vez nos devuelve un vector de Thetas, las calculadas para ser la función mínima.

```
def descenso_gradiente(X, Y, alpha, m, n):
    Thetas = np.zeros(X[1].size)

    for i in range(400):

        Thetas = Thetas - alpha * (1 / m) * ( X.T.dot(np.dot(X, Thetas) -
 Y) )

        coste(X, Y, Thetas)
        costes = []
        costes = np.append(costes, coste(X, Y, Thetas))

    return Thetas
```

Gracias a la implementación anterior con funciones de numpy, el cambio que hemos tenido que realizar en nuestra función ha sido mínimo.

Una vez tenemos calculadas nuestras Thetas, debemos comprobar que efectivamente el resultado obtenido es el deseado.

Para ello, llamamos al método “ecuacionNormal”:

```
def ecuacionNormal(X, Y):
     Thetas = np.zeros(X[1].size)

     Xt = X.T
     aux = np.dot(Xt, X)
     aux2 = np.linalg.pinv(aux)
     aux3 = np.dot(aux2, Xt)
     Thetas = np.dot(aux3, Y)

     return Thetas
```

Este, mediante una fórmula y funciones numpy nos devuelve las thetas mínimas ya calculadas.

Tras esto, las guardamos en un vector y sólo tenemos que coger un punto de los datos y obtener su recta mediante T1(Las thetas calculadas por nosotros) y T2(Las thetas de la ecuación) y ver que ambos valores o coinciden o se aproximan bastante.