Memoria Técnica. Práctica 1

Sergio Gavilán Fernández sgavil01@ucm.es

Alejandro Villar Rubio alvill04@ucm.es

1. Regresión lineal con una variable

Código

```
import numpy as np
from pandas.io.parsers import read_csv
import matplotlib.pyplot as plt

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


def axis_lim_grafica(subPlt, minX, maxX, minY, maxY):
    subPlt.set_xlabel('Población de la ciudad en 10.000s')
    subPlt.set_ylabel('Ingresos en $10.000s')

    subPlt.set_xlim([minX - 1, maxX + 1])
    subPlt.set_ylim([minY - 4, maxY + 0.3])

# Dibuja la gráfica principal donde se mostrará la función h
def dibuja_grafica(subPlt, fArray, funH, theta):
    X = np.linspace(5.0, 22.5, len(fArray), endpoint=True)
    subPlt.scatter(fArray[:, 0], fArray[:, 1], s=50, c='red', marker="x")

    axis_lim_grafica(subPlt, 5.0, 22.5, 0, 25)

    subPlt.plot(fArray[:, 0], funH)
```

```
    max_value = np.amax(fArray[:, 0])
    t0 = theta[0]
    t1 = theta[1]
    subPlt.annotate(r'$h(x)={}+{}x$'.format(t0, t1) ,
        xy=(max_value, h(max_value, theta)), xycoords='data',
        xytext=(5, 26.5), fontsize=10,
        arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=-.2"))


def axis_lim_costes(subPlt, minX, maxX, minY, maxY):
    subPlt.set_xlabel('Número de iteraciones')
    subPlt.set_ylabel('Coste')

    subPlt.set_xlim([minX - 50, maxX + 200])
    subPlt.set_ylim([minY - 0.5, maxY + 0.5])

# Dibuja la gráfica donde se verá mostrará el valor del coste en función
del número de iteraciones
def dibuja_costes(subPlt, numCasos, costeArray):
    X = np.linspace(0, numCasos, numCasos, endpoint=True)
    axis_lim_costes(subPlt, 0, numCasos, costeArray[len(costeArray) - 1],
 costeArray[0])
    subPlt.plot(range(numCasos), costeArray)


# Función principal de la práctica que realiza el algoritmo de "Descenso
de Gradiente"
def descenso_gradiente(casos, alpha=0.01, iter=1500):
    # Inicialización de los valores de theta a 0
    theta = np.zeros(2)

    # Inicialización de un array que guarda el historial de los costes
    costeArray = np.zeros(iter)

    # m es el número de ejemplos
    m = len(casos)
```

```
    # Bucle de "iter" iteraciones (por defecto son 1500) donde calculamos
 el valor de theta que minimice la función de coste
    for i in range(iter):
        temp0 = theta[0] - alpha * (1 / m) * np.sum(h(casos[:,0], theta)
- casos[:,1], axis=0)
        temp1 = theta[1] - alpha * (1 / m) * np.sum((h(casos[:,0], theta)
 - casos[:,1]) * casos[:, 0], axis=0)
        theta[0] = temp0
        theta[1] = temp1

        funH = h(casos[:,0], theta)
        costeArray[i] = (1 / (2 * m)) * np.sum(np.square(h(casos[:,0], th
eta) - casos[:,1]), axis=0)
        plt.clf()
        print(costeArray[i])

    # Llamada a métodos para dibujar las gráficas
    fig, subPlot = plt.subplots(1, 2, figsize=(12, 5))

    dibuja_grafica(subPlot[0], casos, funH, theta)
    dibuja_costes(subPlot[1], iter, costeArray)
    plt.show()

# Función h
def h(x, theta):
   return theta[0] + x * theta[1]


def main(file_name):
    a = carga_csv(file_name)
    descenso_gradiente(casos=a)


main("ex1data1.csv")
```

Desarrollo y Resultados

Como se puede observar en la *Ilustración 1.* tenemos dos gráficas donde aparecen las soluciones a esta primera parte: la gráfica de la izquierda (A) se representa la hipótesis que viene dada por el modelo lineal:

$$h_\theta(x) = \theta_0 + \theta_1 * x$$

y la de la derecha (B) donde se observa la función de coste:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

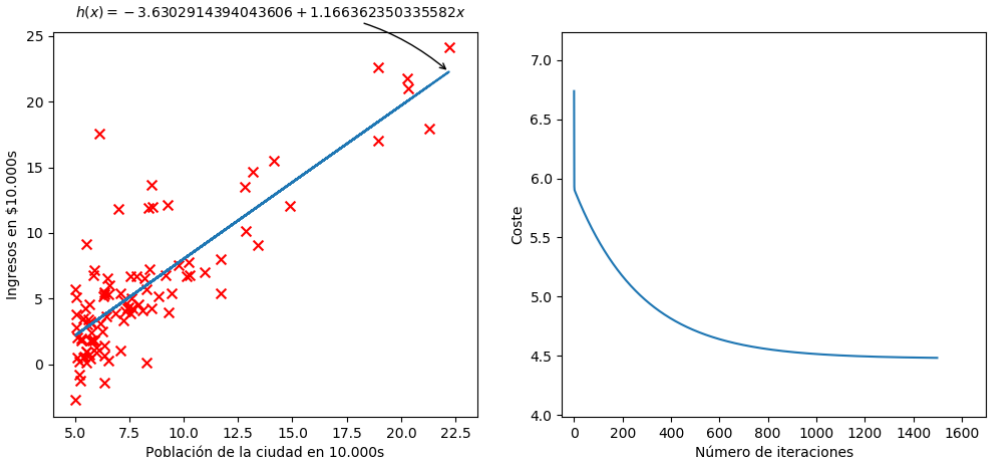

Ilustración 1. Representación gráfica de la solución de la Parte 1

En la gráfica A tenemos representados los datos del fichero *ex1data.csv* lo más similar posible al enunciado de la práctica, donde estos aparecen con una "x". El eje X está formado por aquellos valores de *Población de la ciudad en 10.000s* que se encuentran en un intervalo de [5.0, 22.5] y el eje Y mide la variable de *Ingresos en $10.000s* dentro del intervalo [-4.0, 25.0]. El objetivo principal de esta gráfica es representar la hipótesis $h_\theta(x)$. Para ello se ha creado el método ***descenso_gradiente*** donde nos vamos acercando iterativamente al valor de θ que minimiza la función de coste $J(\theta)$ actualizando cada componente de θ con la expresión:

$$\theta_j := \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})\,x_j^{(i)}$$

Como se puede observar, el resultado de aplicar estas expresiones nos da una solución mostrada en la parte superior de esta misma gráfica:

$$h_\theta(x) = -3.63 + * 1.16x$$

En la gráfica B se puede ver representada la función de coste $J(\theta)$. En el eje X tenemos el *Número de iteraciones* que se mueve en un intervalo de [0, 1500] y el eje Y se corresponde con el *Coste* en función de las iteraciones, en este caso tenemos un intervalo de [4.5, 7.0], en función del número de iteraciones. Como se puede ver el coste va disminuyendo a medida que se realizan más iteraciones debido a que el algoritmo de ***descenso_gradiente*** está encontrando la hipótesis $h_\theta(x)$ que lo minimiza.

Nota: no se ha añadido un 1 como primera componente de cada ejemplo de entrenamiento x.

2. Regresión lineal con varias variables

Código

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np
from pandas.io.parsers import read_csv


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


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


def sumatory(X, Y, j, thetha):
    sum = 0
    for i in range(X.shape[0]):
        sum = sum + (h(np.array([X[i, 0], X[i, 1], X[i, 2]]),
                       thetha) - Y[i]) * X[i, j]
    return sum


def graph_cost_alpha(X, Y):
    alphas = np.array([0.3, 0.1, 0.03, 0.01])

    plt.figure(figsize=(10, 6), dpi=80)
    xx = np.linspace(0, 1500, 1500)

    colors = ["blue", "green", "purple", "red"]
    for i in range(len(alphas)):
        c = (descenso_gradiente(X, Y, alpha=alphas[i]))[1]
        plt.plot(xx, c, color=colors[i], linewidth=2.5,
                 linestyle="-", label="Alpha: " + str(alphas[i]))

    plt.legend(loc='upper right')

    plt.xlabel('Number of iterations')
```

```
    plt.ylabel(r'$J(\theta)$ ')

    plt.show()


def norm_matrix(X):
    fils = X.shape[0]
    cols = X.shape[1]

    onesColumn = np.ones((fils, 1))
    X_norm = np.zeros((fils, cols))

    mu = np.zeros(cols)
    sigma = np.zeros(cols)

    for i in range(cols):
        mu[i] = np.mean(X[:, i])
        sigma[i] = np.std(X[:, i])

    for i in range(cols):
        X_norm[:, i] = (X[:, i] - mu[i]) / sigma[i]

    X_norm = np.hstack((onesColumn, X_norm))

    return X_norm, mu, sigma


def descenso_gradiente(X, Y, alpha=0.1, iter=1500):
    # Inicialización de los valores de theta a 0, con cada iteración su v
alor irá cambiando y siempre para mejor, en el caso de que vaya a peor es
 que esta mal hecho
    n = X.shape[1]
    theta = np.zeros(n)

    m = X.shape[0]
    costes = []
    for i in range(iter):
        for j in range(n):
            theta[j] = theta[j] - alpha * (1/m) * sumatory(X, Y, j, theta
)
        costes.append(coste(X, Y, theta))
    return theta, costes


def ecuacion_normal(X, Y):
    a = np.linalg.inv((np.dot(X.T, X)))
    b = np.dot(X.T, Y)
    return np.dot(a, b)
```

```python
def h(X, Theta):
    return np.dot(X, Theta)


def main(file_name):
    file = carga_csv(file_name)
    X = np.delete(file, np.shape(file)[1]-1, axis=1)
    Y = file[:, file.shape[1]-1]
    X_norm, mu, sigma = norm_matrix(X)

    t_grad = descenso_gradiente(X_norm, Y)[0]
    t_ecnormal = ecuacion_normal(X, Y)
    print("Thetha Grad:", t_grad)
    print("Thetha Ec.Normal:", t_ecnormal)

    x1 = (1650 - mu[0]) / sigma[0]
    x2 = (3 - mu[1]) / sigma[1]

    print("Test Gradiente", np.dot(np.array([1, x1, x2]), t_grad.T))
    print("Test Ecnormal", np.dot(np.array([1650, 3]), t_ecnormal.T))

    graph_cost_alpha(X_norm, Y)


main("ex1data2.csv")
```

Desarrollo y Resultados

Podemos observar que los resultados obtenidos utilizando el descenso de gradiente y la ecuación normal son muy similares, además, en la siguiente gráfica se puede ver que cuanto menor sea el Alpha se acerca de una manera más lenta al valor mínimo de la función de coste según van ocurriendo las iteraciones.