Linear Regression

The objective of this practice was to implement two different linear regression solutions to two problems.

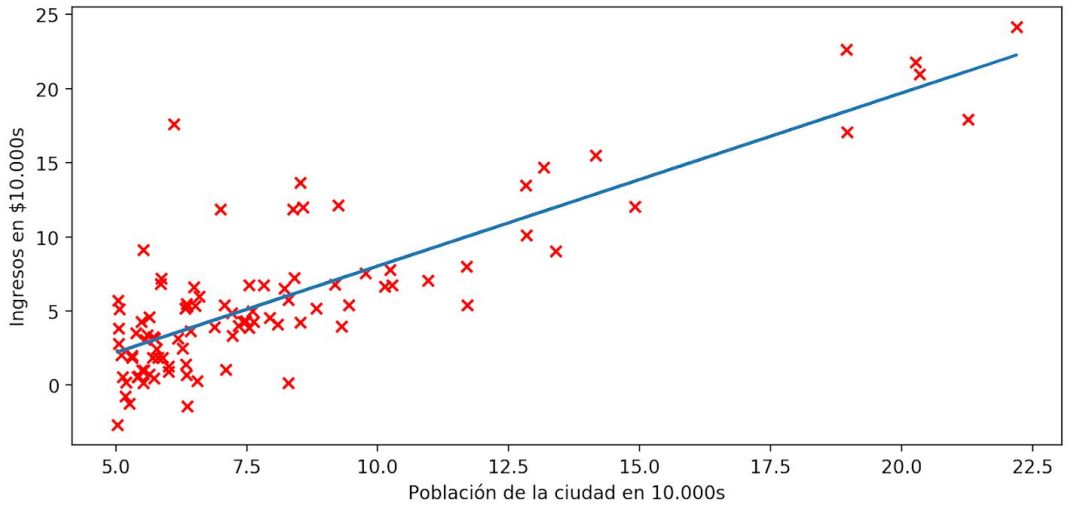
The first problem consisted of finding a function that for a given population of a city will predict the possible benefits for the subject of the study- a food company. To obtain the solution of this problem we read the given data, used the gradient descent method to find the local minimum of our cost function, and specified a function to predict the benefit, taking the city population, finally, we presented the results in a graph.

First part code:

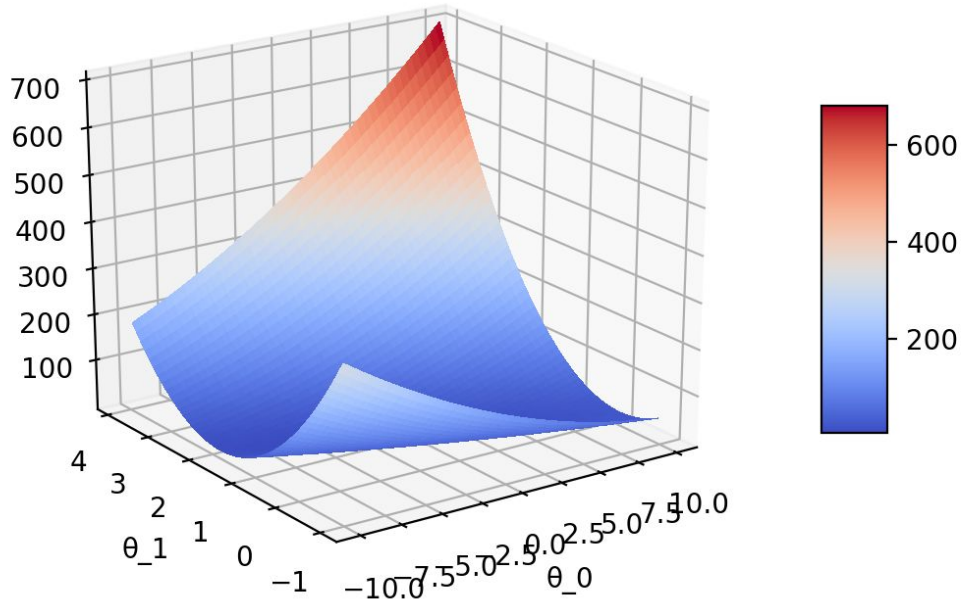
```
import numpy as np
import matplotlib.pyplot as plt
from pandas.io.parsers import read_csv
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

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)

datos = carga_csv('ex1data1.csv')
X = datos[:, :-1] # (97, 1)
Y = datos[:, -1] # (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])

def init_theta(X):
    return np.zeros((X.shape[1], 1))
```

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


def cost(X, Theta, Y):
    return np.sum((hypothesis(X, Theta).flatten() - Y) ** 2) / (2 * m)

def descenso_gradiente(X, Y, alpha):
    Theta = init_theta(X)
    costes = []
    for i in range(1500):
        for j in range(X.shape[1]):
            Theta[j] -= (alpha / m) * np.sum((hypothesis(X, Theta).flatten()
- Y) * X[:, j])
        costes.append(cost(X, Theta, Y))
    return (Theta, costes)

def make_data(X, Y):
    step = 0.1
    T0 = np.arange(-10, 10, step)
    T1 = np.arange(-1, 4, step)
    T0, T1 = np.meshgrid(T0, T1)

    Coste = np.empty_like(T0)
    for ix, iy in np.ndindex(T0.shape):
        Coste[ix, iy] = cost(X, [T0[ix, iy], T1[ix, iy]], Y)

    return (T0, T1, Coste)

def graphic_3D(X, Y):
    fig = plt.figure()
    ax = fig.gca(projection='3d')

    T0, T1, Z = make_data(X, Y)
    surf = ax.plot_surface(T0, T1, Z, cmap = cm.coolwarm, linewidth = 1,
antialiased=False)
    fig.colorbar(surf, shrink=0.5, aspect=5)
```

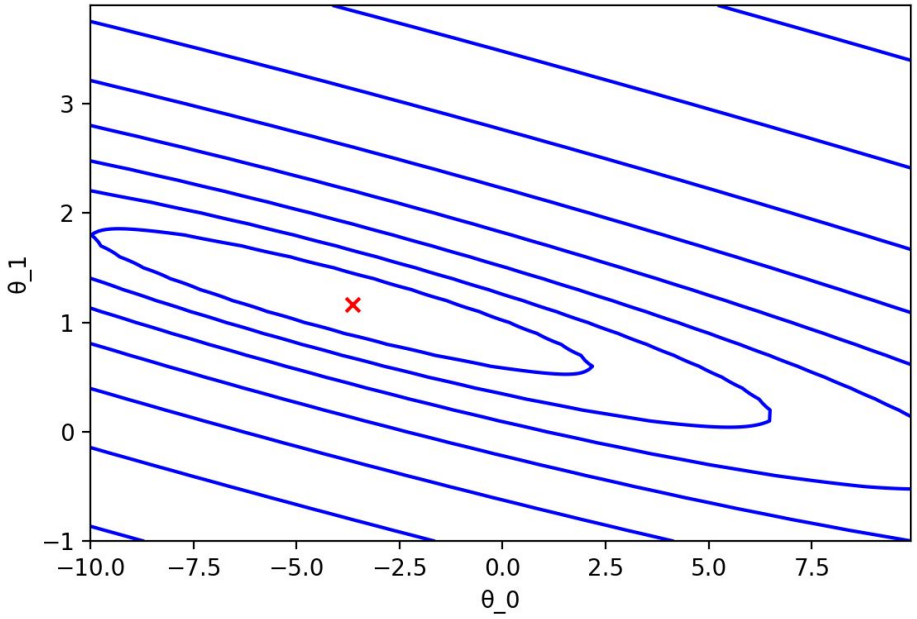
```
def contour(X, Y):
    T0, T1, Z = make_data(X, Y)
    plt.contour(T0, T1, Z, np.logspace(-2, 3, 20), colors='blue')

alpha = 0.01
Thetas, costes = descenso_gradiente(X, Y, alpha)

#cost function graphics:

#3D graphic
#graphic_3D(X, Y)

#contour
#contour(X, Y)
#plt.scatter(Thetas[0], Thetas[1], c = 'red', marker = 'x')

#graphic with data
plt.xlabel("Población de la ciudad en 10.000s")
plt.ylabel("Ingresos en $10.000s")
plt.scatter(X[:, 1:], Y, c = 'red', marker = 'x')
plt.plot(X[:, 1], hypothesis(X, Thetas))
plt.show()
```



In the graph above, we used red crosses to represent the input data and the blue line, the function solution.



The 3D graphic represents the cost function with the respective Thetas (in the interval [-10; 10] for theta 0, and [-1; 4] for theta 1).



In the above contour graph is shown another representation of the minimal cost obtained by the gradient descent.

In the second problem, we had data about the prices of houses in Portland, Oregon with their sizes and the number of rooms they have. Here we needed to calculate a function that will predict the price of a house from a given size and number of rooms. To resolve this problem we used the same approach as in the previous problem- using linear regression with gradient descent. This time we normalized the input data as it was measured in different units and it would be more difficult to converge while calculating the gradient. After normalizing, the calculations were similar to the previous solution. Finally, we verified the correctness of our algorithm using normal equations.

Second part code:

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

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)

datos = carga_csv('ex1data2.csv')
X = datos[:, :-1] # (47, 2)
Y = datos[:, -1] # (47,)

m = np.shape(X)[0]
n = np.shape(X)[1]

def normalizar(X):
    return ((X - np.mean(X, axis = 0)) / np.std(X, axis = 0), np.mean(X, axis
= 0), np.std(X, axis = 0))

# añadimos una columna de 1's a la X_norm
X_norm, mean, std = normalizar(X)
X_norm = np.hstack([np.ones([m, 1]), X_norm])
X = np.hstack([np.ones([m, 1]), X])
```

```
def init_theta(X):
    return np.zeros((X.shape[1], 1))

def hypothesis(X, Theta):
    return np.dot(X, Theta)



def cost(X, Theta, Y):
    return np.dot((hypothesis(X, Theta).flatten() - Y).T, hypothesis(X,
Theta).flatten() - Y)  / (2 * m)

def descenso_gradiente(X, Y, alpha):
    Theta = init_theta(X)
    costes = []
    for i in range(10000):
        for j in range(X.shape[1]):
            Theta[j] -= (alpha / m) * np.sum((hypothesis(X, Theta).flatten() -
Y) * X[:, j])
        costes.append(cost(X, Theta, Y))
    return (Theta, costes)

def eq_normal(X, Y):
    return np.dot(np.dot(np.linalg.pinv(np.dot(X.T, X), rcond=1e-15), X.T),
Y.reshape((Y.shape[0], 1)))



alpha = 0.001
Thetas, costes = descenso_gradiente(X_norm, Y, alpha)
Thetas_qe_normal = eq_normal(X, Y)

#test

test = np.array([[1650, 3]])
test_norm = (test - mean) / std
test = np.hstack([np.ones([np.shape(test)[0], 1]), test])
test_norm = np.hstack([np.ones([np.shape(test)[0], 1]), test_norm])
print("gradient" + str(hypothesis(test_norm, Thetas)))
print("norm" + str(hypothesis(test, Thetas_qe_normal)))

#cost function graphic
```

```
#plt.plot(np.linspace(0, 9999, 10000), np.array(costes))

#graphic with data

plt.show()
```

The solution above prints:

```
gradient[[293222.3541611]]
norm[[293081.46433499]]
```

Where the first result is the price of a house with 3 rooms and size of 1650 using our linear regression algorithm and the second one is with normal equations. As shown the results are similar.

Conclusions:

In this practice, we created two solutions predicting the benefits of a food company in a city with a certain population and the price of a house knowing it's rooms and size. Both of these solutions are based on linear regression with gradient descent and in the second we normalized the input data.

Gasan Nazer and Veronika Yankova