Jose Daniel Rave Robayo
Ángel López Benítez

Practica 2

Regresión Multivariable

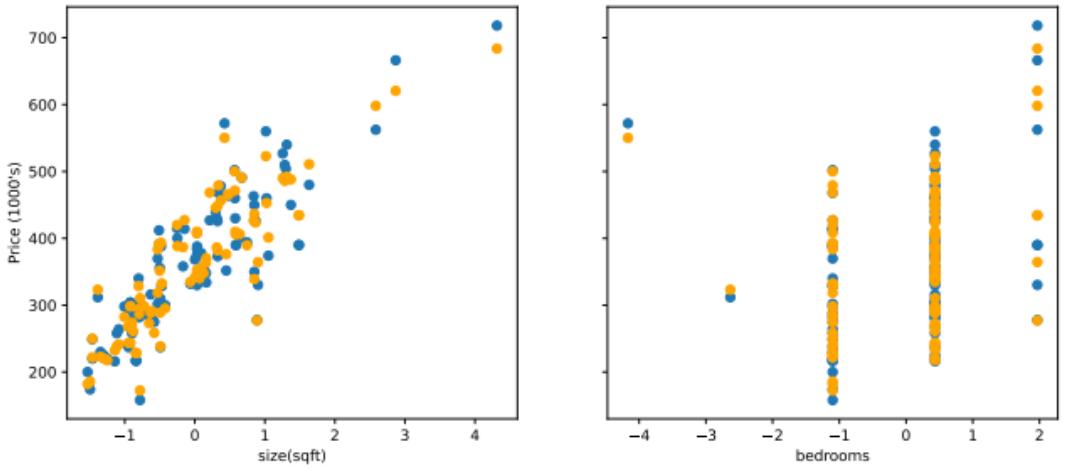
CÓDIGO FUENTE:

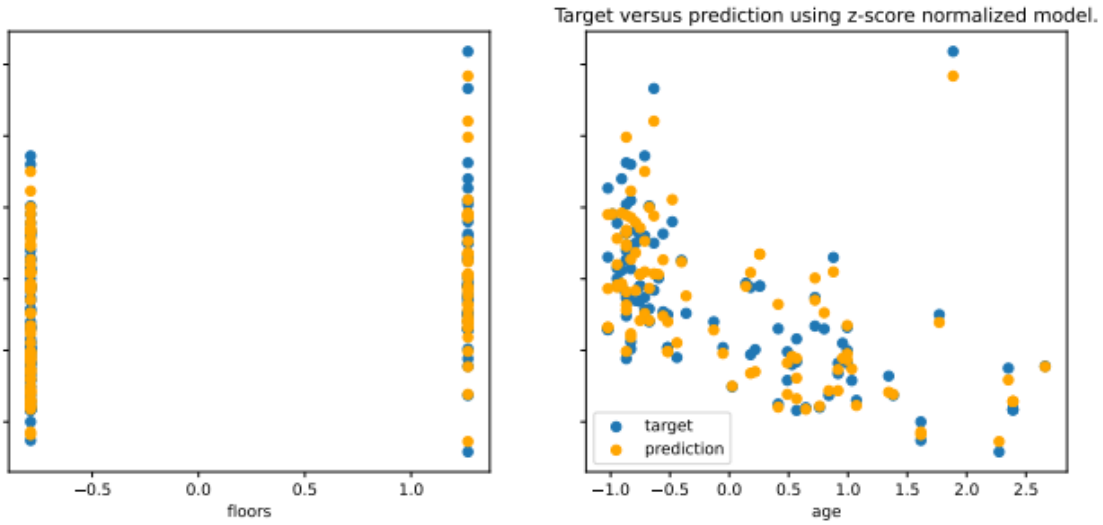
```
def our_test():
    data = np.loadtxt("./data/houses.txt", delimiter=',', skiprows=1)
    X_train = data[:, :4]
    y_train = data[:, 4]

    b_init = 785.1811367994083
    w_init = np.array([0.39133535, 18.75376741, -53.36032453, -26.42131618])

    X_train_norm, mu, sigma = ml.zscore_normalize_features(X_train)

    iterations = 1500
    alpha = 0.01
    # #TRAINING
    w , b, history = ml.gradient_descent(X_train_norm, y_train, w_init, b_init,
ml.compute_cost,
                                         ml.compute_gradient, alpha, iterations)

    X = np.array([1200, 3, 1, 40])
    X = (X - mu) / sigma
    test = np.sum((X @ w)) + b
    target_value = 318.85441519816953
    assert np.isclose(test, target_value,
                      rtol=1e-4), f"Case 1: prediction is wrong: {test} != {target_value}"
    print("\033[92mTest prediction passed!")

    Y_prediction = (X_train_norm @ w) + b

    X_features = ['size(sqft)', 'bedrooms', 'floors', 'age']
    fig, ax = plt.subplots(1, 4, figsize=(25, 5), sharey=True)
    plt.title('Target versus prediction using z-score normalized model.')
    for i in range(len(ax)):
        ax[i].scatter(X_train_norm[:, i], y_train, label = 'target')
        ax[i].scatter(X_train_norm[:, i], Y_prediction, color = 'orange', label =
'prediction')
        ax[i].set_xlabel(X_features[i])

    plt.legend()
    ax[0].set_ylabel("Price (1000's)")
    # plt.show()
    plt.savefig('linearRegression_prediction.pdf')


def public_Test():
    test.compute_cost_test(ml.compute_cost)
    test.compute_gradient_test(ml.compute_gradient)

def main():
    public_Test()
    our_test()
```

Jose Daniel Rave Robayo
Ángel López Benítez

Ejecutando los datos almacenados en el archivo “houses.txt”, obtenemos cuatro gráficas con sus datos representados, y una representación en naranja que predice los valores de dichos datos habiendo obtenido *w* y *b* una vez se ha realizado el entrenamiento.

GRÁFICA:





Para comprobar que los métodos programados funcionan correctamente, pasamos estos por las funciones de testeo proporcionadas en el archivo public_test.py.

```
PS C:\Users\joseda\Desktop\Uni\Cuarto\APA\AprendizajeAutomaticoP0\P2> python .\main.py
All tests passed!
All tests passed!
Test prediction passed!
```

Jose Daniel Rave Robayo
Ángel López Benítez

IMPLEMENTACIÓN

Los métodos implementados para el Descenso de Gradiente son:

Calcular el costo de la función

```
def compute_cost(X, y, w, b):
    m = y.shape[0]
    return  np.sum(((X @ w + b) - y ) ** 2) / (2 * m
)
```

Calcular el gradiente

```
def compute_gradient(X, y, w, b):
    m = y.shape[0]

    fun = X @ w + b
    e = fun - y
    dj_db = np.sum(e / m)
    dj_dw = (X.T @ e) / m

    return dj_db , dj_dw
```

Descenso de Gradiente

```
def gradient_descent(X, y, w_in, b_in, cost_function,
                     gradient_function, alpha,
num_iters):
    J_history = []
    w = copy.deepcopy(w_in)
    b = b_in

    for i in range(num_iters):
      dj_db, dj_dw = gradient_function(X, y, w, b)
      w -= alpha * dj_dw
      b -= alpha * dj_db

      if i < 100000:
```

```
        cost = cost_function(X,y,w,b)
        J_history.append(cost)

    return w, b, J_history
```

Y por último, el método que nos permite normalizar los valores de entrada a datos más manejables (pequeños), ya que de lo contrario tendríamos números tan grandes que complicarían la manipulación con ellos.

Normalización

```
def zscore_normalize_features(X):
    X_norm = 0

    X_norm = np.empty((X.shape[0], X.shape[1]))
    # if(len(X.shape) > 1):
    # else:
    #   X_norm = np.empty((X.shape[0]))

    mu = np.mean(X , axis = 0)
    sigma = np.std(X , axis = 0)

    # X_norm = (X - mu) / sigma
    for i in range (len(X)):
      X_norm[i] = (X[i] - mu) / sigma
    return (X_norm, mu, sigma)
```