Regresión Multi-Variable Linear Regression

Nuestra empresa de desarrollo de videojuegos sigue son su proceso de expansión y quiere comprar una nueva sede. Para ello busca una casa para convertirla en oficina.

El departamento de marketing ha recopilado información acerca del precio de diferentes casas en diferentes ciudades y queremos predecir usando una regresión multi variable el precio de una casa en función de los parámetros establecidos que son: - Tamaño - Número de dormitorios - Número de plantas - Antigüedad de la casa

Para ello se proporciona un dataset con los datos anteriores.

Como ejemplo, si vemos el primer caso del dataset:

9.520000000000000e+02 2.000000000000000000e+00 1.000000000000000000e+00 6.500000000000000000e+01 2.715000000000000000e+02

Lo podemos interpretar como: Tamaño: 952 pies cuadrados. Número de habitaciones: 2 Número de plantas: 1 Antigüedad: 65 Precio: 271.5 en miles de dólares.

Ficheros incluidos en la práctica: - data/houses.txt dataset con los datos de partida. - public_tests.py: funciones para comprobar que el ejercicio es correcto - multi_linear_reg.py: Fichero donde tendréis que implementar la regresión lineal

Se pide:

1. Visualiza los datos del dataset para corroborar visualmente que se ajustan a una recta. Para hacer esto debéis elegir dos atributos y realizar la representación en 3D. Adicionalmente, haced con los otros dos atributos una visualización en 2D.
2. Ajusta el input usando z-score normalization. Para ello implementad la función:

```
"""
computes X, zcore normalized by column
Args:
X (ndarray (m,n)) : input data, m examples, n features

Returns:
X_norm (ndarray (m,n)): input normalized by column
1mu (ndarray (n,)) : mean of each feature
1sigma (ndarray (n,)) : standard deviation of each feature
"""

def zscore_normalize_features(X):
    return (X_norm, mu, sigma)
```

3. Calcula el coste usando la función:

```
"""
compute cost
Args:
X (ndarray (m,n)): Data, m examples with n features
y (ndarray (m,)) : target values
w (ndarray (n,)) : model parameters
b (scalar) : model parameter
Returns
cost (scalar) : cost
"""

def compute_cost(X, y, w, b):
    return cost
```

4. Computación del gradiente

```
"""
Computes the gradient for linear regression
Args:
X : (ndarray Shape (m,n)) matrix of examples
y : (ndarray Shape (m,)) target value of each example
w : (ndarray Shape (n,)) parameters of the model
b : (scalar) parameter of the model
Returns
dj_dw : (ndarray Shape (n,)) The gradient of the cost w.r.t. the
parameters w.
dj_db : (scalar) The gradient of the cost w.r.t. the
parameter b.
"""

def compute_gradient(X, y, w, b):
    return dj_db, dj_dw
```

5. Descenso de gradiente

```
"""
Performs batch gradient descent to learn theta. Updates theta by taking
num_iters gradient steps with learning rate alpha

Args:
X : (array_like Shape (m,n) matrix of examples
y : (array_like Shape (m,)) target value of each example
w_in : (array_like Shape (n,)) Initial values of parameters of the
model
b_in : (scalar) Initial value of parameter of the model
cost_function: function to compute cost
```

```
gradient_function: function to compute the gradient
alpha : (float) Learning rate
num_iters : (int) number of iterations to run gradient descent
Returns
w : (array_like Shape (n,)) Updated values of parameters of the model
after running gradient descent
b : (scalar) Updated value of parameter of the model
after running gradient descent
J_history : (ndarray): Shape (num_iters,) J at each iteration,
primarily for graphing later
"""
def gradient_descent(X, y, w_in, b_in, cost_function,
2 gradient_function, alpha, num_iters):
    return w, b, J_history
```

6. Dibuja la recta de regresión usando el modelo optimizado por descenso de gradiente.

Entregar un zip con: - Los ficheros de la práctica en Python. - Un Jupiter Notebook con la ejecución de los ejercicios y el dibujado de las gráficas.

English version:

Our video game development company is still in the process of expanding and wants to buy a new headquarters. To do so, it is looking for a house to convert it into an office.

The marketing department has collected information about the price of different houses in different cities and we want to predict using a multi-variable regression the price of a house depending on the established parameters which are: - Size - Number of bedrooms - Number of floors - Age of the house

For this purpose, a dataset with the above data is provided.

As an example, if we look at the first case of the dataset:

9.520000000000000e+02 2.000000000000000000e+00 1.000000000000000000e+00 6.500000000000000000e+01 2.715000000000000000e+02

We can interpret this as: Size: 952 sq ft. Number of rooms: 2 Number of floors: 1 Age: 65 Price: 271.5 in thousands of dollars.

Files included in the practice: - data/houses.txt dataset with the starting data. - public_tests.py: functions to check that the exercise is correct. - multi_linear_reg.py: File where you will have to implement the linear regression.

You are asked to:

1. Visualise the dataset data to visually corroborate that it fits a line. To do this you must choose two attributes and make the 3D representation. In

addition, make with the other two attributes a 2D visualisation.

2. Adjust the input using z-score normalization. To do this, implement the function:

```
"""
computes X, zcore normalized by column
Args:
X (ndarray (m,n)) : input data, m examples, n features

Returns:
X_norm (ndarray (m,n)): input normalized by column
1mu (ndarray (n,)) : mean of each feature
1sigma (ndarray (n,)) : standard deviation of each feature
"""

def zscore_normalize_features(X):
    return (X_norm, mu, sigma)
```

3. Calculate the cost using the function:

```
"""
compute cost
Args:
X (ndarray (m,n)): Data, m examples with n features
y (ndarray (m,)) : target values
w (ndarray (n,)) : model parameters
b (scalar) : model parameter
Returns
cost (scalar) : cost
"""

def compute_cost(X, y, w, b):
    return cost
```

4. Gradient computing

```
"""
Computes the gradient for linear regression
Args:
X : (ndarray Shape (m,n)) matrix of examples
y : (ndarray Shape (m,)) target value of each example
w : (ndarray Shape (n,)) parameters of the model
b : (scalar) parameter of the model
Returns
dj_dw : (ndarray Shape (n,)) The gradient of the cost w.r.t. the
parameters w.
```

```
dj_db : (scalar) The gradient of the cost w.r.t. the
parameter b.
"""

def compute_gradient(X, y, w, b):
    return dj_db, dj_dw
```

5. Gradient descent

```
"""
Performs batch gradient descent to learn theta. Updates theta by taking
num_iters gradient steps with learning rate alpha

Args:
X : (array_like Shape (m,n) matrix of examples
y : (array_like Shape (m,)) target value of each example
w_in : (array_like Shape (n,)) Initial values of parameters of the
model
b_in : (scalar) Initial value of parameter of the model
cost_function: function to compute cost
gradient_function: function to compute the gradient
alpha : (float) Learning rate
num_iters : (int) number of iterations to run gradient descent
Returns
w : (array_like Shape (n,)) Updated values of parameters of the model
after running gradient descent
b : (scalar) Updated value of parameter of the model
after running gradient descent
J_history : (ndarray): Shape (num_iters,) J at each iteration,
primarily for graphing later
"""
def gradient_descent(X, y, w_in, b_in, cost_function,
2 gradient_function, alpha, num_iters):
    return w, b, J_history
```

6. Draw the regression line using the optimised gradient descent model.

Deliver a zip file with: - The files of the practice in Python. - A Jupiter Notebook with the execution of the exercises and the drawing of the graphs.