Programming Exercise 2: Multi-variable Linear Regression

Problem Statement: housing price prediction

The training data set contains examples with 4 features (size, bedrooms, floors and age) shown in the table below:

Size (sqft)	Number of Bedrooms	Number of floors	Age of Home	Price (1000s dollars)
952	2	1	65	271.5
1244	3	2	64	232
1947	3	2	17	509.8
…	…	…	…	…

We would like to build a linear regression model using these values so we can then predict the price for other houses.

Files included in this exercise

File	Description
`data/houses.txt`	Dataset for linear regression with multiple variables
`public_tests.py`	Functions to test cost and gradient calculation

File	Description
[⋆] `multi_linear_reg.py`	Functions to compute the cost and the gradient of multi-variable linear regression and to run gradient descent

[⋆] indicates files you will need to complete

Visualize your data

For this dataset, you can use scatter plots to visualize each feature vs. the target (price) providing some indication of which features have the strongest influence on price, as shown in Figure 1.1.

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

X_features = ['size(sqft)', 'bedrooms', 'floors', 'age']
fig, ax = plt.subplots(1, 4, figsize=(25, 5), sharey=True)
for i in range(len(ax)):
    ax[i].scatter(X_train[:, i], y_train)
    ax[i].set_xlabel(X_features[i])
ax[0].set_ylabel("Price (1000's)")
```

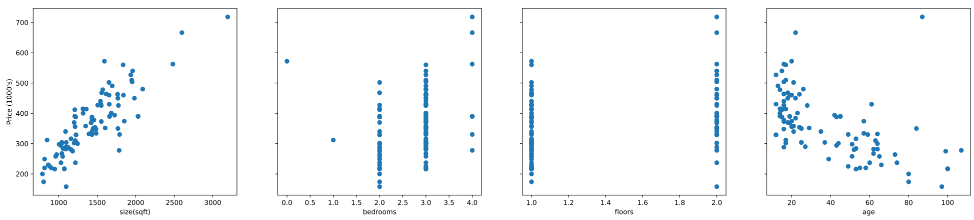

Figure 1.1: Dataset

Normalize

Adjust your input values using z-score normalization:

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$

where j selects a feature or a column in the $\mathbf{X}$ matrix. μ_j is the mean of all the values for feature (j) and σ_j is the standard deviation of feature (j).

Complete the `multi_linear_reg.zscore_normalize_features` function to implement the computation of the cost:

```
def zscore_normalize_features(X):
    """
    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
      mu (ndarray (n,))     : mean of each feature
      sigma (ndarray (n,))  : standard deviation of each feature
    """

    return (X_norm, mu, sigma)
```

When normalizing the features, it is important to store the values used for normalization – the mean value and the standard deviation used for the computations. After learning the parameters from the model, we often want to predict the prices of houses we have not seen before. Given a new x value (living room area and number of bedrooms), we must first normalize x using the mean and standard deviation that we had previously computed from the training set.

Compute cost

The equation for the cost function with multiple variables $J(\vec{w}, b)$ is:

$$J(\mathbf{w}, b) = \frac{1}{2m} \sum_{i=0}^{m-1} (f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})^2$$

where:

$$f_{\mathbf{w},b}(\mathbf{x}^{(i)}) = \mathbf{w} \cdot \mathbf{x}^{(i)} + b$$

and $\mathbf{w}$ and $\mathbf{x}^{(i)}$ are vectors rather than scalars supporting multiple features.

Complete the `multi_linear_reg.compute_cost` function to implement the computation of the cost:

```
def compute_cost(X, y, w, b):
    """
    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
    """

    return cost
```

You should run and pass the tests from `public_tests.compute_cost_test`.

Compute gradient

The gradient for linear regression with multiple variables is defined as:

$$\frac{\partial J(\mathbf{w},b)}{\partial w_j} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})x_j^{(i)}$$
$$\frac{\partial J(\mathbf{w},b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - y^{(i)})$$

Complete the `multi_linear_reg.compute_gradient` function to implement the computation of the gradient:

```
def compute_gradient(X, y, w, b):
    """
    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.
    """

    return dj_db, dj_dw
```

You should run and pass the tests from `public_tests.compute_gradient_test`.

Gradient descent

The gradient descent algorithm is:

$$\begin{aligned}&\text{repeat until convergence: } \{\\ &\qquad w_j = w_j - \alpha \frac{\partial J(\mathbf{w},b)}{\partial w_j} \quad \text{for j = 0..n-1}\\ &\qquad b\ \ = b - \alpha \frac{\partial J(\mathbf{w},b)}{\partial b}\\ &\qquad \}\end{aligned}$$

where n is the number of features, and parameters w_j, b are updated simultaniously.

Complete the `multi_linear_reg.gradient_descent` function to implement the batch gradient descent algorithm:

```
def gradient_descent(X, y, w_in, b_in, cost_function,
                     gradient_function, alpha, num_iters):
    """
    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
    """

    return w, b, J_history
```

You can now check your implementation of gradient descent by computing the predicted value for a 1200 sqft, 3 bedrooms, 1 floor, 40 years old house which should be around $318709.

You can now use the learned model on the training data and visually compared predicted and actual valuesas shown in Figure 1.2.

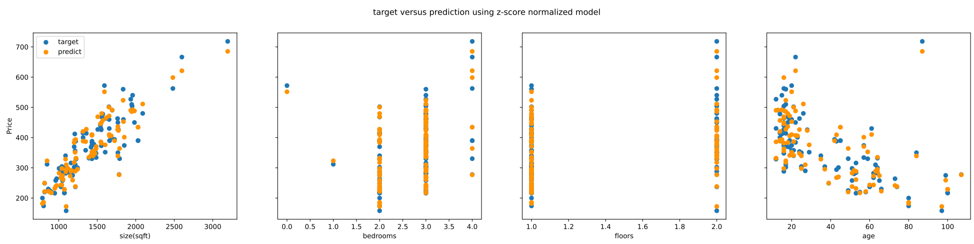

Figure 1.2: Predictions on training data