

# 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
<code>data/houses.txt</code>	Dataset for linear regression with multiple variables
<code>public_tests.py</code>	Functions to test cost and gradient calculation

## Programming Exercise 2: Multi-variable Linear Regression

---

File	Description
[★] <code>multi_linear_reg.py</code>	Functions to compute the cost and the gradient of multi-variable linear regression and to run gradient descent

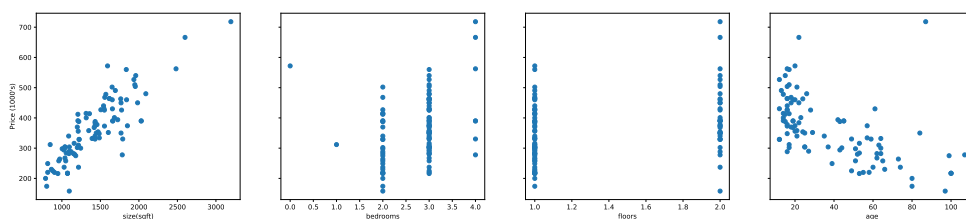
---

[★] 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.

```
1 data = np.loadtxt("./data/houses.txt", delimiter=',', skiprows=1)
2 X_train = data[:, :4]
3 y_train = data[:, 4]
4
5 X_features = ['size(sqft)', 'bedrooms', 'floors', 'age']
6 fig, ax = plt.subplots(1, 4, figsize=(25, 5), sharey=True)
7 for i in range(len(ax)):
8     ax[i].scatter(X_train[:, i], y_train)
9     ax[i].set_xlabel(X_features[i])
10 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.  $\mu_j$  is the mean of all the values for feature ( $j$ ) and  $\sigma_j$  is the standard deviation of feature ( $j$ ).

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

```
1 def zscore_normalize_features(X):
2     """
3     computes X, zcore normalized by column
4
5     Args:
6         X (ndarray (m,n)) : input data, m examples, n features
7
8     Returns:
9         X_norm (ndarray (m,n)): input normalized by column
10        mu (ndarray (n,)) : mean of each feature
11        sigma (ndarray (n,)) : standard deviation of each feature
12    """
13
14    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:

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

## Programming Exercise 2: Multi-variable Linear Regression

---

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:

```
1 def compute_cost(X, y, w, b):
2     """
3     compute cost
4     Args:
5         X (ndarray (m,n)): Data, m examples with n features
6         y (ndarray (m,)) : target values
7         w (ndarray (n,)) : model parameters
8         b (scalar)       : model parameter
9     Returns
10        cost (scalar)    : cost
11    """
12
13    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:

```
1 def compute_gradient(X, y, w, b):
2     """
3     Computes the gradient for linear regression
4     Args:
5         X : (ndarray Shape (m,n)) matrix of examples
6         y : (ndarray Shape (m,)) target value of each example
7         w : (ndarray Shape (n,)) parameters of the model
```

## Programming Exercise 2: Multi-variable Linear Regression

---

```
8     b : (scalar)                parameter of the model
9     Returns
10    dj_dw : (ndarray Shape (n,)) The gradient of the cost w.r.t. the
        parameters w.
11    dj_db : (scalar)            The gradient of the cost w.r.t. the
        parameter b.
12    """
13
14    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:

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

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

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

```
1 def gradient_descent(X, y, w_in, b_in, cost_function,
2                     gradient_function, alpha, num_iters):
3     """
4     Performs batch gradient descent to learn theta. Updates theta by taking
5     num_iters gradient steps with learning rate alpha
6
7     Args:
8     X : (array_like Shape (m,n))    matrix of examples
9     y : (array_like Shape (m,))     target value of each example
10    w_in : (array_like Shape (n,))   Initial values of parameters of the
        model
11    b_in : (scalar)                  Initial value of parameter of the model
12    cost_function: function to compute cost
```

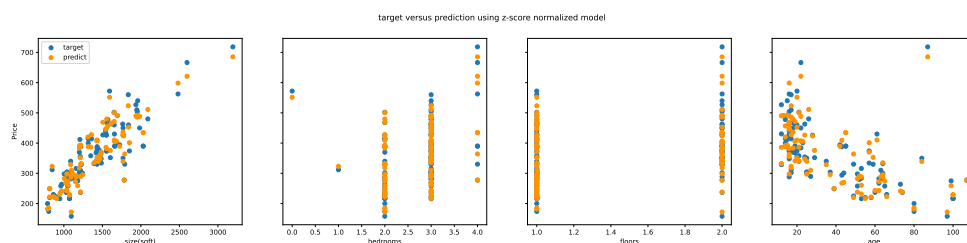
## Programming Exercise 2: Multi-variable Linear Regression

---

```
13     gradient_function: function to compute the gradient
14     alpha : (float) Learning rate
15     num_iters : (int) number of iterations to run gradient descent
16 Returns
17     w : (array_like Shape (n,)) Updated values of parameters of the model
18         after running gradient descent
19     b : (scalar) Updated value of parameter of the model
20         after running gradient descent
21     J_history : (ndarray): Shape (num_iters,) J at each iteration,
22                 primarily for graphing later
23     """
24
25     return w, b, J_history
```

Running gradient descent with a learning rate of 0.1 and 1000 iterations you should obtain an accurate model. You can check your implementation of gradient descent by using the learned model to compute the predicted value for a 1200 sqft, 3 bedrooms, 1 floor, 40 years old house which should be around \$318709. Recall, that you must normalize the data with the mean and standard deviation derived when the training data was normalized.

Finally, you can use the learned model on the training data itself and visually compare predicted and actual values as shown in Figure 1.2.



**Figure 1.2:** Predictions on training data

## Submission of the assignment

The assignment must be delivered using the submission mechanism of the virtual campus. A single file will be delivered in pdf format containing the memory of the practice, including the code developed and the comments and graphs that are considered most appropriate to explain the results obtained.